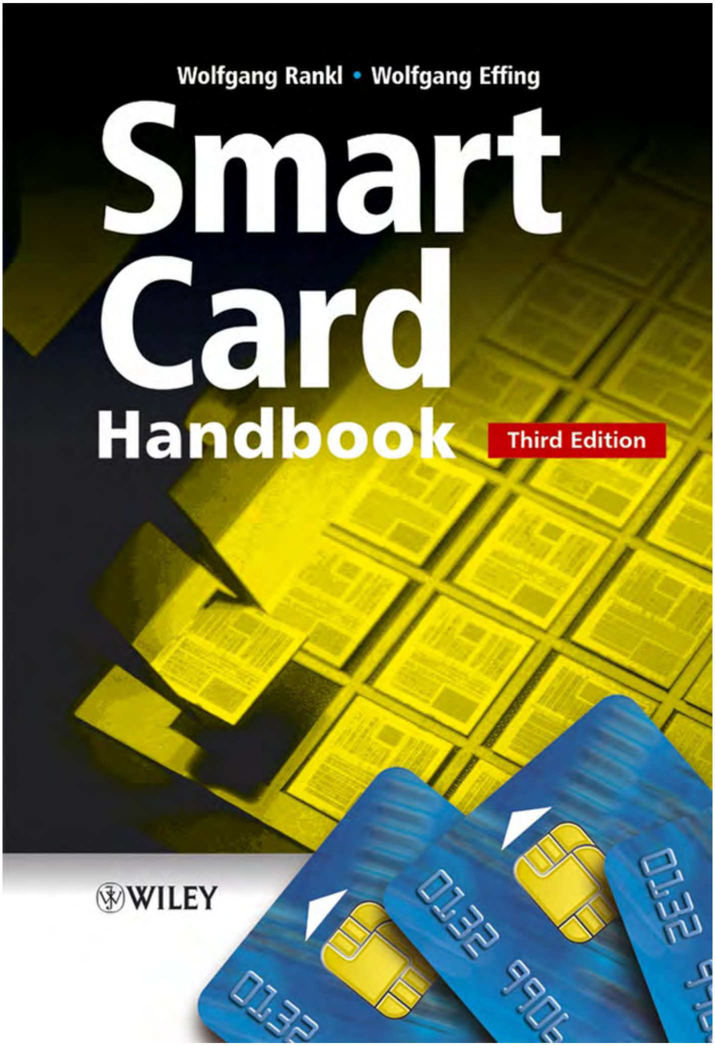
Wolfgang Rankl • Wolfgang Effing

Smart Card Handbook

Third Edition



WILEY

Smart Card Handbook

Third Edition

Wolfgang Rankl and Wolfgang Effing
Giesecke & Devrient GmbH, Munich, Germany

Translated by
Kenneth Cox
Kenneth Cox Technical Translations, Wassenaar, The Netherlands

John Wiley & Sons, Ltd

Smart Card Handbook

Third Edition

Smart Card Handbook

Third Edition

Wolfgang Rankl and Wolfgang Effing
Giesecke & Devrient GmbH, Munich, Germany

Translated by
Kenneth Cox
Kenneth Cox Technical Translations, Wassenaar, The Netherlands

John Wiley & Sons, Ltd

Other Wiley Editorial Offices

John Wiley & Sons Inc., 111 River Street, Hoboken, NJ 07030, USA

Jossey-Bass, 989 Market Street, San Francisco, CA 94103-1741, USA

Wiley-VCH Verlag GmbH, Boschstr. 12, D-69469 Weinheim, Germany

John Wiley & Sons Australia Ltd, 33 Park Road, Milton, Queensland 4064, Australia

John Wiley & Sons (Asia) Pte Ltd, 2 Clementi Loop #02-01, Jin Xing Distripark, Singapore 129809

John Wiley & Sons Canada Ltd, 22 Worcester Road, Etobicoke, Ontario, Canada M9W 1L1

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Library of Congress Cataloging-in-Publication Data
Rankl, W. (Wolfgang)
[Handbuch der Chipkarten. English]
Smart card handbook / Wolfgang Rankl and Wolfgang Effing. – 3rd ed.
p. cm.
Includes bibliographical references and index.
ISBN 0-470-85668-8 (alk. paper)
1. Smart cards–Handbooks, manuals, etc. I. Effing, W. (Wolfgang) II. Title.
TK7895.S62R3613 2003
006 – dc22 2003062750

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

ISBN 0-470-85668-8

Typeset in 10/12pt Times by TechBooks, New Delhi, India
Printed and bound in Great Britain by Antony Rowe Ltd, Chippenham Wiltshire
This book is printed on acid-free paper responsibly manufactured from sustainable forestry
in which at least two trees are planted for each one used for paper production.

Contents

Preface to the Third Edition

The English version of the *Smart Card Handbook* has now reached its third edition. In comparison with the previous edition, it has been considerably expanded and thoroughly updated to represent the current state of the technology. In this book, we attempt to cover all aspects of smart card technology, with the term ‘technology’ intentionally being understood in a very broad sense.

As in previous editions, we have remained true to our motto, ‘better one sentence too many than one word too few’. We have described this ever-expanding subject in as much detail as possible. Even more examples, drawings and photographs have been added to make it easier to understand complicated relationships. The glossary has been enlarged to include many new terms covering all essential concepts related to smart cards, and it has been enhanced with cross-references. In many cases, it can provide a quick introduction to a particular subject. Altogether, these additions, extensions and improvements have resulted in a book that is more than three times as large as the first edition.

Here we can make a small comparison. Modern smart card operating systems currently comprise 120,000 lines of source code, which roughly corresponds to two books the size of the present edition. Even if you are not familiar with programming, you can readily appreciate how sophisticated these operating systems have become.

These small, colorful plastic cards with their semiconductor chips continue to spread from their original countries, Germany and France, throughout the world. In the coming years, this technology can be expected to outstrip all others, especially since it is still in its infancy and there is no end or consolidation in sight.

Smart card technology progresses in leaps and bounds, and we attempt to keep pace by publishing a new edition of the *Smart Card Handbook* every two to three years. The *Smart Card Handbook* represents the present state of technical knowledge, and in areas that are presently undergoing rapid change, we indicate possible paths of evolution. If certain things come to be seen differently at a later date, we can only remark that no one knows what the future will bring. Despite this, or perhaps just because of this, we welcome all comments, suggestions and proposed improvements, so that this book can continue to cover the subject of smart cards as completely as possible. Here we would like to explicitly thank the many attentive and interested readers who have pointed out unclear or ambiguous passages and errors. Once again, an errata list for this edition will be made available at www.wiley.co.uk/commstech/.

We would also like to thank our many friends and colleagues who have repeatedly offered valuable (and occasionally somewhat uncomfortable) suggestions for making this book better

and more complete. We would particularly like to thank Hermann Altschäfl, Peter van Elst, Klaus Finkenzeller, Thomas Graßl, Michael Schnellinger, Harald Vater and Dieter Weiß, as well as Kathryn Sharples at Wiley for her helpful support and Kenneth Cox for the translation.

Munich, June 2002

Wolfgang Rankl
[Rankl@gmx.net], [www.wiley.co.uk/commstech/]

Wolfgang Effing
[WEffing@gmx.net]

Symbols and Notation

General

- In accordance with ISO standards, the least-significant bit is always designated 1, rather than 0.
- In accordance with common usage, the term 'byte' refers to a sequence of eight bits and is equivalent to the term 'octet', which is often used in international standards.
- Length specifications for data, objects and all countable quantities are shown in decimal form, in agreement with the usual practice in smart card standards. All other values are usually shown as hexadecimal numbers and identified as such.
- The prefixes 'kilo' and 'mega' have the values of 1024 (2^{10}) and 1,048,576 (2^{20}), respectively, as is customary in the field of information technology.
- Depending on the context, binary values may not be explicitly identified as such.
- Commands used with smart cards are printed in upper-case characters (for example: SELECT FILE).

Representation of characters and numbers

42	decimal value
'00'	hexadecimal value
°0°, °1°	binary values
"ABC"	ASCII value
B*n*	byte number *n* (for example: B1)
b*n*	bit number *n* (for example: b2)
D*n*	digit number *n* (for example: D3)

Logical functions

||	concatenation (of data elements or objects)
$\oplus$	logical XOR operation

$\wedge$	logical AND operation
$\vee$	logical OR operation
$a \in M$	a is an element of the set M
$a \notin M$	a is not an element of the set M
$\{a, b, c\}$	the set of elements a, b, c

Cryptographic functions

enc $_{Xn}$ (K; D)	encryption using the algorithm X and an n-bit key, with the key K and the data D [for example: **enc** $_{\text{DES56}}$ ('1 . . . 0'; 42)]
dec $_{Xn}$ (K; D)	decryption using the algorithm X and an n-bit key, with the key K and the data D [for example: **dec** $_{\text{IDEA128}}$ ('1 . . . 0'; 42)]
S := **sign** $_{Xn}$ (K; D)	generating the signature S using the algorithm X and an n -bit key, with the key K and the data D [for example: **sign** $_{\text{RSA512}}$ ('1 . . . 0'; "Wolf")]
R := **verify** $_{Xn}$ (K; S)	verifying the signature S using the algorithm X and an n-bit key, with the key K [for example: **verify** $_{\text{RSA512}}$ ('1 . . . 9'; 42)]
Result = OK/NOK	

References

See: '. . . '	This is a cross-reference to another location in the book.
See also: '. . . '	This is a cross-reference to another location in the book where more information on the subject can be found.
[. . .]	This is a reference to a World Wide Web site listed in the Appendix.
[X Y]	This is a cross-reference to additional literature or standards listed in the Appendix. The format is: X ∈ {surname of the first-named author} Y ∈ {last two digits of the year of publication}

Program Code Conventions

The syntax and semantics of the program code used in this book are based on the standard dialects of Basic. However, the use of explanations in natural language within a program listing is allowed, in order to promote the understandability of the code. Naturally, although this makes it easier for the reader to understand the code, it means that it is not possible to automatically convert the code into machine code. This compromise is justified by the significant improvement in readability that it provides.

:=	assignment operator
::=	definition operator
=, !=, <, <=, >, =>	comparison operators
+, −, ×, /	arithmetic operators
NOT	logical not
AND	logical and
OR	logical or
||	concatenation operator (e.g., coupling two byte strings)
_	end-of-line marker for multiline instructions
//...	comment
IO_Buffer	variable (printed in italics)
Label:	jump or call location (printed in bold)
GOTO...	jump
CALL...	function call (subroutine call)
RETURN	return from a function (subroutine)
IF...THEN...	decision, type 1
IF...THEN...ELSE...	decision, type 2
SEARCH (...)	search in a list; search string in parentheses
STATUS	query the result of a previously executed function call
STOP	terminate a process
LENGTH (...)	calculate the length
EXIST	test for presence (for example: an object or data element)
WITH...	starts the definition of a variable or object as a reference
END WITH	ends the definition of a variable or object as a reference

Abbreviations

3DES	triple DES *(see glossary)*
3GPP	Third Generation Partnership Project *(see glossary)*
3GPP2	Third Generation Partnership Project 2 *(see glossary)*
A3, A5, A8	GSM algorithm 3, 5, 8 *(see glossary)*
AAM	application abstract machine
ABA	American Bankers' Association
ABS	acrylonitrile butadiene styrene
AC	access conditions *(see glossary)*
ACD	access control descriptor
ACK	acknowledge
ACM	accumulated call meter
ADF	application dedicated file
ADN	abbreviated dialing number
AES	Advanced Encryption Standard *(see glossary)*
AFI	application family identifier
AFNOR	*Association Française de Normalisation (see glossary)*
AGE	*Autobahngebührenerfassung* [motorway toll collection]
AGE	*automatische Gebührenerfassung* [automatic toll collection]
AID	application identifier *(see glossary)*
AM	access mode
Amd.	Amendment
AMPS	Advanced Mobile Phone Service *(see glossary)*
AND	logical AND operation
ANSI	American National Standards Institute *(see glossary)*
AoC	Advice of Charge
AODF	authentication object directory file
APACS	Association for Payment Clearing Services
APDU	application protocol data unit *(see glossary)*
A-PET	amorphous polyethylene terephthalate
API	application programming interface *(see glossary)*
AR	access rules
ARM	advanced RISC machine

ARR	access rule reference
ASC	application-specific command
ASCII	American Standard Code for Information Interchange
ASIC	application-specific integrated circuit
ASK	amplitude shift keying *(see glossary)*
ASN.1	Abstract Syntax Notation 1 *(see glossary)*
AT	attention
ATM	automated teller machine
ATQA	answer to request, type A
ATQB	answer to request, type B
ATR	answer to reset *(see glossary)*
ATS	answer to select
ATTRIB	PICC selection command, type B
AUX	auxiliary
B2A	business-to-administration *(see glossary)*
B2B	business-to-business *(see glossary)*
B2C	business-to-consumer *(see glossary)*
Basic	Beginners All Purpose Symbolic Instruction Code
BCD	binary-coded digit
Bellcore	Bell Communications Research Laboratories
BER	Basic Encoding Rules *(see glossary)*
BER-TLV	Basic Encoding Rules – tag, length, value
BEZ	*Börsenevidenzzentrale* [electronic purse clearing center for Geldkarte]
BGT	block guard time
BIN	bank identification number
bit	binary digit
BPF	basic processor functions
BPSK	binary phase-shift keying *(see glossary)*
BS	base station
BWT	block waiting time
CA	certification authority *(see glossary)*
CAD	chip accepting device *(see glossary)*
CAFE	Conditional Access for Europe (EU project)
CAMEL	Customized Applications for Mobile Enhanced Logic
CAP	card application *(see glossary)*
C-APDU	command APDU *(see glossary)*
CAPI	crypto API (application programming interface)
CASCADE	Chip Architecture for Smart Card and Portable Intelligent Devices
CASE	computer-aided software engineering
CAT	card application toolkit
CAVE	Cellular Authentication, Voice Privacy and Encryption
CBC	cipher block chaining

CC	Common Criteria *(see glossary)*
CCD	card-coupling device
CCD	charge-coupled device
CCITT	*Comité Consultatif International Télégraphique et Téléphonique* (now ITU) *(see glossary)*
CCR	chip-card reader
CCS	cryptographic checksum *(see glossary)*
CD	committee draft
CDF	certificate directory file
CDM	card-dispensing machine
CDMA	code division multiple access *(see glossary)*
CEN	*Comité Européen de Normalisation (see glossary)*
CENELEC	*Comité Européen de Normalisation Eléctrotechnique* [European Committee for Electronics Standardization]
CEPS	Common Electronic Purse Specifications, (previously: Common European Purse System) *(see glossary)*
CEPT	*Conférence Européenne des Postes et Télécommunications (see glossary)*
CFB	cipher feedback
CGI	common gateway interface
CHV	cardholder verification
CICC	contactless integrated circuit card
CID	card identifier
CISC	complex instruction set computer
CLA	class
CLK	clock
CLn	cascade level n, type A
CMM	capability maturity model *(see glossary)*
CMOS	complementary metal-oxide semiconductor
CMS	card management system
COS	chip operating system *(see glossary)*
COT	chip-on-tape *(see glossary)*
CRC	cyclic redundancy check *(see glossary)*
CRCF	clock rate conversion factor
CRT	Chinese remainder theorem
CRT	control reference template
Cryptoki	cryptographic token interface
CSD	circuit-switched data
C-SET	Chip-SET (secure electronic transaction)
CT	chipcard terminal
CT	card terminal
CT	cascade tag, type A
CT	cordless telephone
CT-API	chipcard terminal (CT) API *(see glossary)*
CTDE	cryptographic token data element
CTI	cryptographic token information

CTIO	cryptographic token information object
CVM	cardholder verification method
CWT	character waiting time
D	divisor
DAD	destination address
DAM	DECT authentication module *(see glossary)*
DAM	draft amendment
D-AMPS	Digital Advanced Mobile Phone Service *(see glossary)*
DAP	data authentication pattern
DB	database
DBF	database file
DBMS	database management system
DC/SC	Digital Certificates on Smart Cards
DCODF	data container object directory file
DCS	digital cellular system
DEA	data encryption algorithm *(see glossary)*
DECT	Digital Enhanced Cordless Telecommunications (previously: Digital European Cordless Telecommunications) *(see glossary)*
DER	Distinguished Encoding rules *(see glossary)*
DES	Data Encryption Standard *(see glossary)*
DF	dedicated file (also often: directory file) *(see glossary)*
DFA	differential fault analysis *(see glossary)*
DFÜ	*Datenfernübertragung* [data telecommunications]
DIL	dual in-line
DIN	*Deutsche Industrienorm* [German industrial standard]
DIS	draft international standard
DLL	dynamic link library
DMA	direct memory access
DO	data object
DoD	US Department of Defense
DOM	document object model
DOV	data over voice
DPA	differential power analysis *(see glossary)*
dpi	dots per inch
DR	divisor receive (PCD to PICC)
DRAM	dynamic random-access memory *(see glossary)*
DRI	divisor receive integer (PCD to PICC)
DS	divisor send (PICC to PCD)
DSA	digital signature algorithm
DSI	divisor send integer (PICC to PCD)
DTAUS	*Datenträgeraustausch* [data storage medium exchange]
DTD	document type definition
DTMF	dual-tone multiple-frequency
DVD	digital versatile disc
DVS	*Dateiverwaltungssystem* [file management system]

E	end of communication, type A
EBCDIC	extended binary-coded decimal interchange code
EC	elliptic curve
ec	Eurocheque
ECB	electronic codebook
ECBS	European Committee for Banking Standards *(see glossary)*
ECC	elliptic curve cryptosystems *(see glossary)*
ECC	error correction code *(see glossary)*
ECDSA	elliptic curve DSA
ECML	Electronic Commerce Modeling Language
ECTEL	European Telecom Equipment and Systems Industry
EDC	error detection code *(see glossary)*
EDGE	Enhanced Data Rates for GSM and TDMA Evolution *(see glossary)*
EDI	electronic data interchange
EDIFACT	electronic data interchange for administration, commerce and transport
EEPROM, E^{2}PROM	electrically erasable programmable read-only memory *(see glossary)*
EF	elementary file *(see glossary)*
EFF	Electronic Frontier Foundation
EFI	EF internal
EFTPOS	electronic fund transfer at point of sale
EFW	EF working
EGT	extra guard time, type B
EMV	Europay, MasterCard, Visa *(see glossary)*
EOF	end of frame, type B
EPROM	erasable programmable read-only memory *(see glossary)*
ESD	electrostatic discharge
ESPRIT	European Strategic Programme of Research and Development in Information Technology (EU project)
ETS	European Telecommunication Standard *(see glossary)*
ETSI	European Telecommunications Standards Institute *(see glossary)*
etu	elementary time unit *(see glossary)*
f	following page
FAR	false acceptance rate
FAT	file allocation table *(see glossary)*
FBZ	*Fehlbedienungszähler* [error counter, key fault presentation counter, retry counter] *(see glossary)*
fC	frequency of operating field (carrier frequency)
FCB	file control block
FCC	Federal Communications Commission
FCFS	first-come, first-serve
FCI	file control information
FCOS	flip chip on substrate

FCP	file control parameters
FD/CDMA	frequency division / code division multiple access *(see glossary)*
FDMA	frequency division multiple access *(see glossary)*
FDN	fixed dialing number
FDT	frame delay time, type A
FEAL	fast data encipherment algorithm
FET	field-effect transistor
ff	following pages
FID	file identifier *(see glossary)*
FIFO	first in, first out
FINEID	Finnish Electronic Identification Card
FIPS	Federal Information Processing Standard *(see glossary)*
FMD	file management data
FO	frame option
FPGA	field-programmable gate array *(see glossary)*
FPLMTS	Future Public Land Mobile Telecommunication Service *(see glossary)*
FRAM	ferroelectric random-access memory *(see glossary)*
FRR	false rejection rate
FS	file system
fS	frequency of subcarrier modulation
FSC	frame size for proximity card
FSCI	frame size for proximity card integer
FSD	frame size for coupling device
FSDI	frame size for coupling device integer
FSK	frequency-shift keying
FTAM	file transfer, access and management
FWI	frame waiting time integer
FWT	frame waiting time
FWTTEMP	temporary frame waiting time
gcd	greatest common denominator
GF	Galois fields
GGSN	gateway GPRS support node
GND	ground
GP	Global Platform *(see glossary)*
GPL	GNU public license
GPRS	General Packet Radio System *(see glossary)*
GPS	Global Positioning System
GSM	Global System for Mobile Communications (previously: Groupe Spécial Mobile) *(see glossary)*
GTS	GSM Technical Specification
GUI	graphical user interface
HAL	hardware abstraction layer *(see glossary)*
HBCI	Home Banking Computer Interface *(see glossary)*
HiCo	high coercivity

HLTA	Halt command, type A
HLTB	Halt command, type B
HSCSD	high-speed circuit switched data
HSM	hardware security module
HSM	high-security module
HSM	host security module
HTML	hypertext markup language
HTTP	hypertext transfer protocol
HV	Vickers hardness
HW	hardware
I/O	input/output
I^2C	inter-integrated circuit
IATA	International Air Transport Association
IBAN	international bank account number
I-block	information block
ICC	integrated-circuit card *(see glossary)*
ID	identifier
IDEA	international data encryption algorithm
IEC	International Electrotechnical Commission *(see glossary)*
IEEE	Institute of Electrical and Electronics Engineers
IEP	intersector electronic purse
IFD	interface device *(see glossary)*
IFS	information field size
IFSC	information field size for the card
IFSD	information field size for the interface device
IIC	institution identification codes
IMEI	international mobile equipment identity
IMSI	international mobile subscriber identity
IMT-2000	International Mobile Telecommunication 2000 *(see glossary)*
IN	intelligent network
INF	information field
INS	instruction
INTAMIC	International Association of Microcircuit Cards
IP	Internet protocol
IPES	Improved Proposed Encryption Standard
IrDA	Infrared Data Association
ISDN	Integrated Services Digital Network *(see glossary)*
ISF	internal secret file
ISIM	IP security identity module
ISO	International Organization for Standardization *(see glossary)*
IT	information technology
ITSEC	Information Technology Security Evaluation Criteria *(see glossary)*
ITU	International Telecommunications Union *(see glossary)*
IuKDG	*Informations- und Kommunikations-Gesetz* [German Information and Communications Act]

IV	initialization vector
IVU	in-vehicle unit
J2ME	Java 2 Micro Edition
JCF	Java Card Forum *(see glossary)*
JCRE	Java Card runtime environment *(see glossary)*
JCVM	Java Card virtual machine *(see glossary)*
JDK	Java development kit *(see glossary)*
JECF	Java Electronic Commerce Framework
JIT	just in time
JTC1	Joint Technical Committee One
JVM	Java virtual machine
K	key
Kc	ciphering key
KD	derived key
KFPC	key fault presentation counter
Ki	individual key
KID	identifier key
KM	master key
KS	session key
KVK	*Krankenversichertenkarte* [German medical insurance card]
LA	location area
LAN	local-area network
Lc	command length
LCSI	life cycle status indicator
Le	expected length
LEN	length
LFSR	linear-feedback shift register
LIFO	last in, first out
LND	last number dialed
LOC	lines of code
LoCo	low coercivity
LRC	longitudinal redundancy check
LSAM	load secure application module
lsb	least significant bit
LSB	least significant byte
M	month
MAC	message authentication code / data security code *(see glossary)*
MAOS	multi-application operating system
MBL	maximum buffer length
MBLI	maximum buffer length index
MCT	multifunctional card terminal *(see glossary)*
ME	mobile equipment
MEL	Multos Executable Language
MExE	mobile station execution environment *(see glossary)*

MF	master file *(see glossary)*
MFC	multi-function card, multifunctional smart card
MIME	Multipurpose Internet Mail Extensions
MIPS	million instructions per second
MLI	multiple laser image
MM	*moduliertes Merkmal* [modulated feature]
MMI	man–machine interface
MMS	multimedia messaging service
MMU	memory-management unit
MOC	matching-on-chip
MOO	mode of operation
MOSAIC	Microchip On-Surface and In-Card
MOSFET	metal-oxide semiconductor field-effect transistor
MoU	Memorandum of Understanding *(see glossary)*
MS	mobile station
msb	most significant bit
MSB	most significant byte
MSE	MANAGE SECURITY ENVIRONMENT
MTBF	mean time between failures
MUSCLE	Movement for the Use of Smart Cards in a Linux Environment
NAD	node address
NAK	negative acknowledgement
NBS	US National Bureau of Standards *(see glossary)*
NCSC	National Computer Security Center *(see glossary)*
NDA	nondisclosure agreement
NIST	US National Institute of Standards and Technology *(see glossary)*
nok	not OK
NPU	numeric processing unit *(see glossary)*
NRZ	non-return to zero
NSA	US National Security Agency *(see glossary)*
NU	not used
NVB	number of valid bits
OBU	onboard unit
ODF	object directory file
OFB	output feedback
OID	object identifier
OOK	on/off keying
OP	Open Platform *(see glossary)*
OR	logical OR operation
OS	operating system
OSI	Open Systems Interconnections
OTA	Open Terminal Architecture
OTA	over-the-air *(see glossary)*
OTASS	over-the-air SIM services
OTP	one-time password

OTP	one-time programmable
OTP	Open Trading Protocol
OVI	optically variable ink
P1, P2, P3	parameter 1, 2, 3
PA	power analysis
PB	procedure byte
PC	personal computer
PC	polycarbonate
PC/SC	personal computer / smart card *(see glossary)*
PCB	protocol control byte
PCD	proximity coupling device *(see glossary)*
PCMCIA	Personal Computer Memory Card International Association
PCN	personal communication networks
PCS	personal communication system
PDA	personal digital assistant
PES	proposed encryption standard
PET	polyethylene terephthalate
PETP	partially crystalline polyethylene terephthalate
PGP	Pretty Good Privacy
PICC	proximity ICC *(see glossary)*
PIN	personal identification number
PIX	proprietary application identifier extension
PKCS	public-key cryptography standards *(see glossary)*
PKI	public-key infrastructure *(see glossary)*
PLL	phase-locked loop
PLMN	public land mobile network *(see glossary)*
PM	person–month
POS	point of sale *(see glossary)*
POZ	*POS ohne Zahlungsgarantie* [POS without payment guarantee]
PP	protection profile *(see glossary)*
PPM	pulse position modulation
PPC	production planning and control
PPS	protocol parameter selection
prEN	*pre Norme Européenne* [preliminary European standard]
prETS	pre European Telecommunication Standard
PrKDF	private key directory file
PRNG	pseudorandom number generator *(see glossary)*
PROM	programmable read-only memory
PSAM	purchase secure application module
PSK	phase shift keying
PSO	PERFORM SECURITY OPERATION
PSTN	public switched telephone network *(see glossary)*
PTS	protocol type selection
PTT	*Postes Télégraphes et Téléphones* [post, telegraph and telephone]
Pub	publication

PUK	personal unblocking key *(see glossary)*
PuKDF	public key directory file
PUPI	pseudo-unique PICC identifier
PVC	polyvinyl chloride
PWM	pulse width modulation
RAM	random-access memory *(see glossary)*
R-APDU	response APDU *(see glossary)*
RATS	request to answer to select
REJ	reject
REQA	request command, type A
REQB	request command, type B
RES	resynchronization
RF	radio frequency
RFC	request for comment
RFID	radio frequency identification
RFU	reserved for future use
RID	record identifier
RID	registered application provider identifier
RIPE	RACE (EU project) integrity primitives evaluation
RIPE-MD	RACE integrity primitives evaluation message digest
RISC	reduced instruction set computer
RND	random number
RNG	random number generator
ROM	read-only memory *(see glossary)*
RS	Reed–Solomon
RSA	Rivest, Shamir and Adleman cryptographic algorithm
RTE	runtime environment
R-UIM	removable user identity module *(see glossary)*
S	start of communication
S@T	SIM Alliance Toolbox
S@T	SIM Alliance Toolkit
S@TML	SIM Alliance Toolbox Markup Language
SA	security attributes
SA	service area
SAD	source address
SAGE	Security Algorithm Group of Experts
SAK	select acknowledge
SAM	secure application module *(see glossary)*
SAT	SIM Application Toolkit *(see glossary)*
SC	security conditions
SC	smart card
SCC	smart card controller
SCMS	smart card management system
SCOPE	Smart Card Open Platform Environment *(see glossary)*
SCP	Smart Card Platform

SCQL	structured card query language
SCSUG	Smart Card Security Users Group
SDL	specification and description language
SDMA	space division multiple access *(see glossary)*
SE	security environment *(see glossary)*
SECCOS	Secure Chip Card Operating System *(see glossary)*
SEIS	Secured Electronic Information in Society
SEL	select code
SELECT	select command
SEMPER	Secure Electronic Marketplace for Europe (EU project)
SEPP	secure electronic payment protocol
SET	secure electronic transaction *(see glossary)*
SFGI	start-up frame guard time integer
SFGT	start-up frame guard time
SFI	short file identifier *(see glossary)*
SGSN	serving GPRS support node
S-HTTP	secure hypertext transfer protocol
SigG	*Signaturgesetz* [German electronic signature act] *(see glossary)*
SigV	*Signaturverordnung* [German electronic signature ordinance] *(see glossary)*
SIM	subscriber identity module *(see glossary)*
SIMEG	Subscriber Identity Module Expert Group *(see glossary)*
SKDF	secret key directory file
SM	secure messaging
SM	security mechanism
SMD	surface mounted device *(see glossary)*
SMG9	Special Mobile Group 9 *(see glossary)*
SMIME	Secure Multipurpose Internet Mail Extensions
SMS	short message service *(see glossary)*
SMSC	short message service center
SMS-PP	short message service point to point
SOF	start of frame
SPA	simple power analysis *(see glossary)*
SQL	structured query language
SQUID	superconducting quantum interference device
SRAM	static random-access memory *(see glossary)*
SRES	signed response
SS	supplementary service
SSC	send sequence counter
SSL	secure socket layer
SSO	single sign-on *(see glossary)*
STARCOS	Smart Card Chip Operating System (product of G+D)
STC	sub technical committee
STK	SIM Application Toolkit *(see glossary)*
STT	secure transaction technology
SVC	stored value card (product of Visa International)

SW	software
SW1, SW2	status word 1, 2
SWIFT	Society for Worldwide Interbank Financial Telecommunications
T	tag
TAB	tape-automated bonding
TACS	Total Access Communication System
TAL	terminal application layer
TAN	transaction number *(see glossary)*
TAR	toolkit application reference
tbd	to be defined
TC	trust center *(see glossary)*
TC	technical committee
TC	thermochrome
TCOS	Telesec Card Operating System
TCP	transport control protocol
TCP/IP	Transmission Control Protocol / Internet Protocol
TCSEC	Trusted Computer System Evaluation Criteria *(see glossary)*
TD/CDMA	time division / code division multiple access *(see glossary)*
TDES	triple DES *(see glossary)*
TDMA	time division multiple access *(see glossary)*
TETRA	Trans-European Trunked Radio *(see glossary)*
TLS	transport layer security
TLV	tag, length & value *(see glossary)*
TMSI	temporary mobile subscriber identity
TOE	target of evaluation *(see glossary)*
TPDU	transmission protocol data unit *(see glossary)*
TRNG	true random number generator *(see glossary)*
TS	technical specification
TTCN	tree-and-tabular combined notation
TTL	terminal transport layer
TTL	transistor-transistor logic
TTP	trusted third party *(see glossary)*
UART	universal asynchronous receiver/transmitter *(see glossary)*
UATK	UIM Application Toolkit
UCS	Universal Character Set *(see glossary)*
UI	user interface
UICC	universal integrated circuit card *(see glossary)*
UID	unique identifier
UIM	user identity module *(see glossary)*
UML	unified modeling language *(see glossary)*
UMTS	Universal Mobile Telecommunication System *(see glossary)*
URL	uniform resource locator *(see glossary)*
USAT	USIM application toolkit *(see glossary)*
USB	universal serial bus
USIM	universal subscriber identity module *(see glossary)*

USSD	unstructured supplementary services data
UTF	UCS transformation format
UTRAN	UMTS radio access network
VAS	value-added services *(see glossary)*
Vcc	supply voltage
VCD	vicinity coupling device
VEE	Visa Easy Entry *(see glossary)*
VKNR	*Versichertenkartennummer* [subscriber card number for German medical insurance]
VLSI	very large scale integration
VM	virtual machine *(see glossary)*
VOP	Visa Open Platform *(see glossary)*
Vpp	programming voltage
VSI	vertical system integration
W3C	World Wide Web Consortium
WAE	wireless application environment
WAN	wide-area network
WAP	wireless application protocol *(see glossary)*
WCDMA	wideband code division multiple access *(see glossary)*
WDP	wireless datagram protocol
WfSC	Windows for Smart Cards
WG	working group
WIG	wireless Internet gateway
WIM	wireless identification module *(see glossary)*
WML	wireless markup language *(see glossary)*
WORM	write once, read multiple
WSC	Windows for Smart Cards
WSP	wafer-scale package
WSP	wireless session protocol
WTAI	wireless telephony application interface
WTLS	wireless transport layer security
WTP	wireless transport protocol
WTX	waiting time extension
WTXM	waiting time extension multiplier
WUPA	wakeup command, type A
WUPB	wakeup command, type B
WWW	World Wide Web *(see glossary)*
XML	extensible markup language *(see glossary)*
XOR	logical exclusive-OR operation
Y	year
ZKA	*Zentraler Kreditausschuss* [Central Loans Committee] *(see glossary)*

1 Introduction

This book has been written for students, engineers and technically minded persons who want to learn more about smart cards. It attempts to cover this broad topic as completely as possible, in order to provide the reader with a general understanding of the fundamentals and the current state of the technology.

We have put great emphasis on a practical approach. The wealth of pictures, tables and references to real applications is intended to help the reader become familiar with the subject rather more quickly than would be possible with a strictly technical presentation. This book is thus intended to be useful in practice, rather than technically complete. For this reason, descriptions have been kept as concrete as possible. In places where we were faced with a choice between technical accuracy and ease of understanding, we have tried to strike a happy medium. Whenever this proved to be impossible, we have always given preference to ease of understanding.

The book has been written so that it can be read in the usual way, from front to back. We have tried to avoid forward references as much as possible. The designs of the individual chapters, in terms of structure and content, allow them to be read individually without any loss of understanding. The comprehensive index and the glossary allow this book to be used as a reference work. If you want to know more about a specific topic, the references in the text and the annotated directory of standards will help you find the relevant documents.

Unfortunately, a large number of abbreviations have become established in smart card technology, as in so many other areas of technology and everyday life. This makes it particularly difficult for newcomers to become familiar with the subject. We have tried to minimize the use of these cryptic and frequently illogical abbreviations. Nevertheless, we have often had to choose a middle way between internationally accepted smart card terminology used by specialists and common terms more easily understood by laypersons. If we have not always succeeded, the extensive list of abbreviations at the front of the book should at least help overcome any barriers to understanding, which we hope will be short-lived. An extensive glossary in the final chapter of the book explains the most important technical concepts and supplements the list of abbreviations.

An important feature of smart cards is that their properties are strongly based on international standards. This is fundamentally important with regard to the usually compulsory need for interoperability. Unfortunately, these standards are often difficult to understand, and in

Smart Card Handbook, Third Edition. W. Rankl and W. Effing
© 2004 John Wiley & Sons, Ltd ISBN: 0-470-85668-8

some critical places they require outright interpretation. Sometimes only the members of the associated standardization group can explain the intention of certain sections. In such cases, the *Smart Card Handbook* attempts to present the understanding that is generally accepted in the smart card industry. Nevertheless, the relevant standards are still the ultimate authority, and in such cases they should always be consulted.

1.1 THE HISTORY OF SMART CARDS

The proliferation of plastic cards started in the USA in the early 1950s. The low price of the synthetic material PVC made it possible to produce robust, durable plastic cards that were much more suitable for everyday use than the paper and cardboard cards previously used, which could not adequately withstand mechanical stresses and climatic effects.

The first all-plastic payment card for general use was issued by the Diners Club in 1950. It was intended for an exclusive class of individual, and thus also served as a status symbol, allowing the holder to pay with his or her ‘good name’ instead of cash. Initially, only the more select restaurants and hotels accepted these cards, so this type of card came to be known as a ‘travel and entertainment’ card.

The entry of Visa and MasterCard into the field led to a very rapid proliferation of ‘plastic money’ in the form of credit cards. This occurred first in the USA, with Europe and the rest of the world following a few years later. Today, credit cards allow travelers to shop without cash everywhere in the world. A cardholder is never at a loss for means of payment, yet he or she avoids exposure to the risk of loss due to theft or other unpredictable hazards, particularly while traveling. Using a credit card also eliminates the tedious task of exchanging currency when traveling abroad. These unique advantages helped credit cards become rapidly established throughout the world. Many hundreds of millions of cards are produced and issued annually.

At first, the functions of these cards were quite simple. They served as data storage media that were secure against forgery and tampering. General information, such as the card issuer’s name, was printed on the surface, while personal data elements, such as the cardholder’s name and the card number, were embossed. Many cards also had a signature panel where the cardholder could sign his or her name for reference. In these first-generation cards, protection against forgery was provided by visual features, such as security printing and the signature panel. Consequently, the system’s security depended quite fundamentally on the quality and conscientiousness of the persons responsible for accepting the cards. However, this did not represent an overwhelming problem, due to the card’s initial exclusivity. With the increasing proliferation of card use, these rather rudimentary features no longer proved sufficient, particularly since threats from organized criminals were growing apace.

Increasing handling costs for merchants and banks made a machine-readable card necessary, while at the same time, losses suffered by card issuers as the result of customer insolvency and fraud grew from year to year. It became apparent that the security features for protection against fraud and manipulation, as well as the basic functions of the card, had to be expanded and improved.

The first improvement consisted of a magnetic stripe on the back of the card, which allowed digital data to be stored on the card in machine-readable form as a supplement to the visual information. This made it possible to minimize the use of paper receipts, which were previously essential, although the customer’s signature on a paper receipt was still required in traditional credit card applications as a form of personal identification. However, new approaches that did not require paper receipts could also be devised. This made it possible to finally achieve the long-standing objective of replacing paper-based transactions by electronic data processing. This required a different method to be used for user identification, which previously employed the user's signature. The method that has come into widespread general use involves a secret personal identification number (PIN) that is compared with a reference number. The reader is surely familiar with this method from using bank machines (automated teller machines). Embossed cards with magnetic stripes are still the most commonly used types of cards for financial transactions.

However, magnetic-stripe technology has a crucial weakness, which is that the data stored on the stripe can be read, deleted and rewritten at will by anyone with access to the necessary equipment. It is thus unsuitable for storing confidential data. Additional techniques must be used to ensure confidentiality of the data and prevent manipulation of the data. For example, the reference value for the PIN could be stored in the terminal or host system in a secure environment, instead of on the magnetic stripe. Most systems that employ magnetic-stripe cards thus use online connections to the system's host computer for reasons of security, even though this generates significant costs for the necessary data transmissions. In order to reduce costs, it is necessary to find solutions that allow card transactions to be executed offline without endangering the security of the system.

The development of the smart card, combined with the expansion of electronic data-processing systems, has created completely new possibilities for devising such solutions. Enormous progress in microelectronics in the 1970s made it possible to integrate data storage and processing logic on a single silicon chip measuring a few square millimetres. The idea of incorporating such an integrated circuit into an identification card was contained in a patent application filed by the German inventors Jürgen Dethloff and Helmut Grötrupp as early as 1968. This was followed in 1970 by a similar patent application by Kunitaka Arimura in Japan. However, the first real progress in the development of smart cards came when Roland Moreno registered his smart card patents in France in 1974. It was only then that the semiconductor industry was able to supply the necessary integrated circuits at acceptable prices. Nevertheless, many technical problems still had to be solved before the first prototypes, some of which contained several integrated circuit chips, could be transformed into reliable products that could be manufactured in large numbers with adequate quality at a reasonable cost. Since the basic inventions in smart card technology originated in Germany and France, it is not surprising that these countries played the leading roles in the development and marketing of smart cards.

The great breakthrough was achieved in 1984, when the French PTT (postal and telecommunications services agency) successfully carried out a field trial with telephone cards. In this field trial, smart cards immediately proved to meet all expectations with regard to high reliability and protection against manipulation. Significantly, this breakthrough for smart cards did not come in an area where traditional cards were already used, but in a new application. Introducing a new technology in a new application has the great advantage that compatibility with existing systems does not have to be taken into account, so the capabilities of the new technology can be fully exploited.

A pilot project was conducted in Germany in 1984–85, using telephone cards based on several technologies. Magnetic-stripe cards, optical-storage (holographic) cards and smart cards were used in comparative tests. Smart cards proved to be the winners in this pilot study. In addition to a high degree of reliability and security against manipulation, smart card technology promised the greatest degree of flexibility for future applications. Although the older but less expensive EPROM technology was used in the French telephone card chips, more recent EEPROM chips were used from the start in the German telephone cards. The latter type of chip does not need an external programming voltage. An unfortunate consequence is that the French and German telephone cards are mutually incompatible. It appears that even after the introduction of the euro, French and German telephone cards will remain unusable in each other's country of origin for at least a while.

Further developments followed the successful trials of telephone cards, first in France and then in Germany, with breathtaking speed. By 1986, several million 'smart' telephone cards were in circulation in France alone. The total rose to nearly 60 million in 1990, and to several hundred million worldwide in 1997. Germany experienced similar progress, with a time lag of about three years. These systems were marketed throughout the world after the successful introduction of the smart card public telephone in France and Germany. Telephone cards incorporating chips are currently used in more than 50 countries.

The integrated circuits used in telephone cards are relatively small, simple and inexpensive memory chips with specific security logic that allows the card balance to be reduced while protecting it against manipulation. Microprocessor chips, which are significantly larger and more complex, were first used in large numbers in telecommunications applications, specifically for mobile telecommunications. In 1988, the German Post Office acted as a pioneer in this area by introducing a modern microprocessor card using EEPROM technology as an authorization card for the analog mobile telephone network (C-Netz). The reason for introducing such cards was an increasing incidence of fraud with the magnetic-stripe cards used up to that time. For technical reasons, the analog mobile telephone network was limited to a relatively small number of subscribers (around one million), so it was not a true mass market for microprocessor cards. However, the positive experience gained from using smart cards in the analog mobile telephone system was decisive for the introduction of smart cards into the digital GSM network. This network was put into service in 1991 in various European countries and has presently expanded over the entire world, with over 600 million subscribers in more than 170 countries.

Progress was significantly slower in the field of bank cards, in part due to their greater complexity compared with telephone cards. These differences are described in detail in the following chapters. Here we would just like to remark that the development of modern cryptography has been just as crucial for the proliferation of bank cards as developments in semiconductor technology.

With the general expansion of electronic data processing in the 1960s, the discipline of cryptography experienced a sort of quantum leap. Modern hardware and software made it possible to implement complex, sophisticated mathematical algorithms that allowed previously unparalleled levels of security to be achieved. Moreover, this new technology was available to everyone, in contrast to the previous situation in which cryptography was a covert science in the private reserve of the military and secret services. With these modern cryptographic procedures, the strength of the security mechanisms in electronic data-processing systems could be mathematically calculated. It was no longer necessary to rely on a highly subjective assessment of conventional techniques, whose security essentially rests on the secrecy of the procedures used.

The smart card proved to be an ideal medium. It made a high level of security (based on cryptography) available to everyone, since it could safely store secret keys and execute cryptographic algorithms. In addition, smart cards are so small and easy to handle that they can be carried and used everywhere by everybody in everyday life. It was a natural idea to attempt to use these new security features for bank cards, in order to come to grips with the security risks arising from the increasing use of magnetic-stripe cards.

The French banks were the first to introduce this fascinating technology in 1984, following a trial with 60,000 cards in 1982–83. It took another 10 years before all French bank cards incorporated chips. In Germany, the first field trials took place in 1984–85, using a multifunctional payment card incorporating a chip. However, the *Zentrale Kreditausschuss* (ZKA), which is the coordinating committee of the leading German banks, did not manage to issue a specification for multifunctional Eurocheque cards incorporating chips until 1996. In 1997, all German savings associations and many banks issued the new smart cards. In the previous year, multifunctional smart cards with POS functions, an electronic purse and optional value-added services were issued in all of Austria. This made Austria the first country in the world to have a nationwide electronic purse system.

An important milestone for the future worldwide use of smart cards for making payments was the completion of the EMV specification, which was a product of the joint efforts of Europay, MasterCard and Visa. The first version of this specification was published in 1994. It contained detailed descriptions of credit cards incorporating microprocessor chips, and it guaranteed the mutual compatibility of the future smart cards of the three largest credit card organizations.

Electronic purse systems have proven to be another major factor in promoting the international use of smart cards for financial transactions. The first such system, called Danmønt, was put into operation in Denmark in 1992. There are currently more than 20 national systems in use in Europe alone, many of which are based on the European EN 1546 standard. The use of such systems is also increasing outside of Europe. In the USA, where smart-card systems have had a hard time becoming established, Visa experimented with a smart-card purse during the 1996 Olympic Summer Games in Atlanta. Payments via the Internet offer a new and promising application area for electronic purses. However, the problems associated with making small payments securely but anonymously throughout the world via the public Internet have not yet been solved in a satisfactory manner. Smart cards could play a decisive role in providing an answer to these problems. Besides this, smart cards could plan an important role in introducing electronic signatures. Several European countries have initiated the introduction of electronic signature systems after a legal basis for the use of electronic signatures was provided by approval of a European directive regarding electronic signatures in 1999.

As the result of another application, almost every German citizen now possesses a smart card. When health insurance cards incorporating chips were introduced, more than 70 million smart cards were issued to all persons enrolled in the national health insurance plan. Presently, smart cards are being used in the health-care sector in many countries.

The smart card's high degree of functional flexibility, which even allows programs for new applications to be added to a card already in use, has opened up completely new application areas extending beyond the boundaries of traditional card uses.

Smart cards are also being used as 'electronic tickets' for local public transport in many cities throughout the world. Contactless smart cards are usually used for such applications, since they are particularly convenient and user friendly.

1.2 APPLICATION AREAS

As can be seen from the historical summary, the potential applications for smart cards are extremely diverse. With the steadily increasing storage and processing capacities of available integrated circuits, the range of potential applications is constantly being expanded. Since it is impossible to describe all of these applications in detail within the confines of this book, a few typical examples must serve to illustrate the basic properties of smart cards. This introductory chapter is only meant to provide an initial overview of the functional versatility of these cards. Several typical applications are described in detail in Chapters 12, 13 and 14.

To make this overview easier to follow, it is helpful to divide smart cards into two categories: memory cards and microprocessor cards.

1.2.1 Memory cards

The first smart cards used in large quantities were memory cards for telephone applications. These cards are prepaid, with the value stored electronically in the chip being decreased by the amount of the call charge each time the card is used. Naturally, it is necessary to prevent the user from subsequently increasing the stored value, which could easily be done with a magnetic-stripe card. With such a card, all the user would have to do is record the data stored at the time of purchase and rewrite them to the magnetic stripe after using the card. The card would then have its original value and could be reused. This type of manipulation, known as 'buffering', is prevented in smart phone cards by security logic in the chip that makes it impossible to erase a memory cell once it has been written. The reduction of the card balance by the number of charge units used is thus irreversible.

This type of smart card can naturally be used not only for telephone calls, but also whenever goods or services are to be sold against prior payment without the use of cash. Examples of possible uses include local public transport, vending machines of all types, cafeterias, swimming pools, car parks and so on. The advantage of this type of card lies in its simple technology (the surface area of the chip is typically only a few square millimeters), and hence its low cost. The disadvantage is that the card cannot be reused once it is empty, but must be discarded as waste – unless it ends up in a card collection.

Another typical application of memory cards is the German health insurance card, which has been issued since 1994 to all persons enrolled in the national health insurance plan. The information previously written on the patient's card is now stored in the chip and printed or laser-engraved on the card. Using a chip for data storage makes the cards machine-readable using simple equipment.

In summary, memory-type smart cards have limited functionality. Their integrated security logic makes it possible to protect stored data against manipulation. They are suitable for use as prepaid cards or identification cards in systems where low cost is a primary consideration.

1.2.2 Microprocessor cards

As already noted, microprocessor cards were first used in the form of bank cards in France. Their ability to securely store private keys and execute modern cryptographic algorithms made it possible to implement highly secure offline payment systems.

Since the microprocessor built into the card is freely programmable, the functionality of microprocessor cards is restricted only by the available storage space and the capacity of the processor. The only limits to the designer's imagination when implementing smart card systems are thus technological, and they are extended enormously with each new generation of integrated circuits.

Following a drastic reduction in the cost of smart cards in the early 1990s due to mass production, new applications have been introduced year after year. The use of smart cards with mobile telephones has been especially important for their international proliferation. After being successfully tested in the German national C-Netz (analog mobile telephone network) for use in mobile telephones, smart cards were prescribed as the access medium for the European digital mobile telephone system (GSM). In part, this was because smart cards allowed a high degree of security to be achieved for accessing the mobile telephone network. At the same time, they provided new possibilities and thus major advantages in marketing mobile telephones, since they made it possible for network operators and service providers to sell telephones and services separately. Without the smart card, mobile telephones would certainly not have spread so quickly across Europe or developed into a worldwide industry standard.

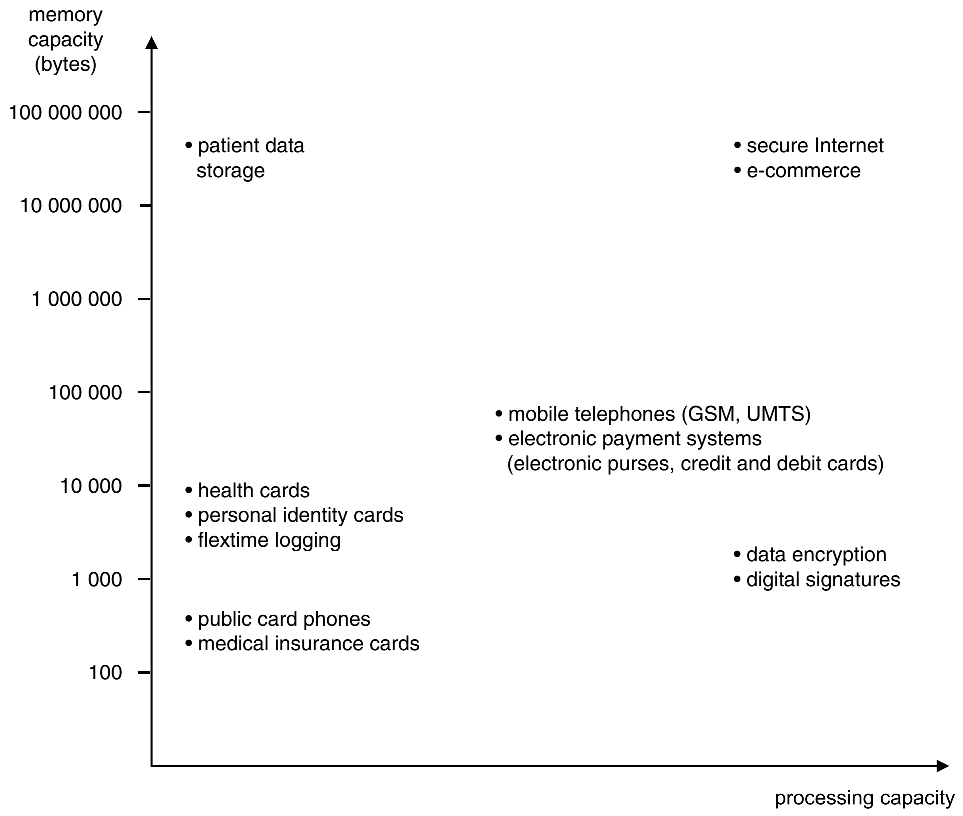


Figure 1.1 Typical smart card application areas, showing the required storage capacity and arithmetic processing capacity

Possible applications for microprocessor cards include identification, access control systems for restricted areas and computers, secure data storage, electronic signatures and electronic purses, as well as multifunctional cards incorporating several applications in a single card. Modern smart-card operating systems also allow new applications to be loaded into a card after it has already been issued to the user, without compromising the security of the various applications. This new flexibility opens up completely new application areas. For example, personal security modules are indispensable if Internet commerce and payments are to be made trustworthy. Such security modules could securely store personal keys and execute high-performance cryptographic algorithms. These tasks can be performed in an elegant manner by a microprocessor with a cryptographic coprocessor. Specifications for secure Internet applications using smart cards are currently being developed throughout the world. Within a few years, we can expect to see every PC equipped with a smart-card interface.

In summary, the essential advantages of microprocessor cards are large storage capacity, the ability to securely store confidential data and the ability to execute cryptographic algorithms. These advantages make a wide range of new applications possible, in addition to the traditional bank card application. The potential of smart cards is by no means yet exhausted, and furthermore, it is constantly being expanded by progress in semiconductor technology.

1.2.3 Contactless cards

Contactless cards, in which energy and data are transferred without any electrical contact between the card and the terminal, have achieved the status of commercial products in the last few years. Presently, both memory cards and microprocessor cards are available as contactless cards. Although contactless microprocessor cards can usually work at a distance of only a few centimeters from the terminal, contactless memory cards can be used up to a meter away from the terminal. This means that such cards do not necessarily have to be held in the user's hand during use, but can remain in the user's purse or wallet.

Contactless cards are thus particularly suitable for applications in which persons or objects should be quickly identified. Sample applications are:

- access control,
- local public transportation,
- ski passes,
- airline tickets,
- baggage identification.

However, there are also applications where operation over a long distance could cause problems and should thus be prevented. A typical example is an electronic purse. A declaration of intent on the part of the cardholder is normally required to complete a financial transaction. This confirms the amount of the payment and the cardholder's agreement to pay. With a contactless card, this declaration takes the form of inserting the card in the terminal and confirming the indicated amount using the keypad. If contactless payments over relatively long distances were possible, a 'con artist' could remove money from the electronic purse without the knowledge of the cardholder. Dual-interface cards (sometimes called 'combicards') offer a possible solution to this problem. These cards combine contact and contactless interfaces in a single card. Such a card can communicate with the terminal via either its contact interface or its contactless interface, according to what is desired.

There is great interest in contactless cards in the field of local public transportation. If the smart cards presently used in payment systems, which are generally contact-type cards, can have their functionality extended to include acting as electronic tickets with contactless interfaces, transportation operators could use the existing infrastructure and cards of the credit card industry.

1.3 STANDARDIZATION

The prerequisite for the worldwide penetration of smart cards into everyday life, such as their current use in Germany in the form of telephone cards, health insurance cards and bank cards, has been the creation of national and international standards. Due to the special significance of such standards, in this book we repeatedly refer to currently applicable standards and those that are in preparation. Why are standards so important for expanding the use of smart cards?

A smart card is normally one component of a complex system. This means that the interfaces between the card and the rest of the system must be precisely specified and matched to teach other. Of course, this could be done for each system on a case-by-case basis, without regard to other systems. However, this would mean that a different type of smart card would be needed for each system. Users would thus have to carry a separate card for each application. In order to avoid this, an attempt has been made to generate application-independent standards that allow multifunctional cards to be developed.

Since the smart card is usually the only component of the system that the user holds in his or her hand, it is enormously important with regard to the recognition and acceptance of the entire system. Nonetheless, from a technical and organizational perspective, the smart card is usually only the tip of the iceberg, since complex systems (which are usually networked) are quite often hidden behind the card terminal, and it is these systems that make the services possible in the first place.

Let us take telephone cards as an example. In technical terms, they are fairly simple objects. By themselves, they are almost worthless, except perhaps as collector's items. Their true function, which is to allow public telephones to be used without coins, can be realized only after umpteen thousand card phones have been installed throughout a region and connected to a network. The large investments required for this can only be justified if the long-term viability of the system is ensured by appropriate standards and specifications. Standards are also an indispensable prerequisite for multifunctional smart cards used for several applications, such as telephony, electronic purses, electronic tickets and so on.

What are standards

This question is not as trivial as it may appear at first glance, since the terms 'standard' and 'specification' are often used fairly indiscriminately. To make things clear, let us consider the ISO/IEC definition:

Standard: a document that is produced by consensus and adopted by a recognized organization, and which, for general and recurring applications, defines rules, guidelines or features for activities or their results, with the objective of achieving an optimum degree of regulation in a given context.

Note: standards should be based on the established results of science, technology and experience, and their objective should be the promotion of optimized benefits for society.

International standards should thus help make life easier and increase the reliability and usefulness of products and services. In order to avoid confusion, ISO/IEC have also defined the term 'consensus' as follows:

Consensus: general agreement, characterized by the absence of continuing objections to essential elements on the part of any significant portion of the interested parties, and achieved by a procedure that attempts to consider the views of all relevant parties and address all counter-arguments.

Note: consensus does not necessarily mean unanimity.

Although unanimity is not required for consensus, the democratic process naturally takes time. This is in particular due to the fact that it is necessary to consider not only the views of the technical specialists, but also the views of all relevant parties, since the objective of a standard is the promotion of optimum benefits for the whole of society. Hence, the preparation of an ISO or CEN standard usually takes several years. A frequent consequence of the slowness of this process is that a small group of interested parties, such as commercial firms, generates its own specification ('industry standard') in order to hasten the development of new systems. This is particularly true in the field of information technology, which is characterized by especially fast development and correspondingly short innovation cycles. Although industry standards and specifications have the advantage that they can be developed significantly faster than 'true' standards, they carry the risk of ignoring the interests of the parties that are not involved in their development. For this reason, ISO attempts to create possibilities for retroactively incorporating significant publicly accessible specifications into international standards.

What does ISO/IEC mean?

The ISO/IEC standards are especially significant for smart cards, since they define the basic properties of smart cards. What lies behind the abbreviations 'ISO' and 'IEC'? 'ISO' stands for the International Organization for Standardization, while 'IEC' stands for the International Electrotechnical Commission.

The International Organization for Standardization (ISO) is a worldwide association of around 100 national standards agencies, with one per country. ISO was founded in 1948 and is a non-national organization. Its task is to promote the development of standards throughout the world, with the objective of simplifying the international exchange of goods and services and developing cooperation in the fields of science, technology and economy. The results of the activities of ISO are agreements that are published as ISO standards.

Incidentally, 'ISO' is not an abbreviation (the abbreviation of the official name would of course be 'IOS'). The name 'ISO' is derived from the Greek word *isos*, which means 'equal' or 'the same'. The prefix *iso-*, derived from the Greek *isos*, is commonly used in the three official languages of ISO (English, French and Russian), as well as in many other languages.

As already noted, the members of ISO are the national standards bodies of the individual countries, and only one such body per country is allowed to be a member. The member organizations have four basic tasks, as follows:

- to inform potentially interested parties in their own countries about relevant activities and possibilities of international standardization,
- to form national opinions on a democratic basis and represent these opinions in international negotiations,
- to set up a secretariat for ISO committees in which the country has a particular interest,
- to pay the country's financial contribution in support of the central ISO organization.

The IEC is a standardization organization whose scope of activity covers the areas of electrical technology and electronics. The first card standards were published by the IEC. After the introduction of smart cards, a difference of focus arose between the ISO and the IEC. In order to avoid duplication of effort, standards are developed in joint technical committees and published as ISO/IEC standards.

How is an ISO standard generated?

The need for a standard is usually reported to a national standards organization by an industrial sector. The national organization then proposes this to ISO as a new working theme. If the proposal is accepted by the responsible working group, which consists of technical experts from countries that are interested in the theme, the first thing that is done is to define the objective of the future standard.

After agreement has been reached with regard to the technical aspects to be considered in the standard, the detailed specifications of the standard are discussed and negotiated among the various countries. This is the second phase in the development of the standard. The objective of this phase is to arrive at a consensus of all participating countries, if possible. The outcome of this phase is a 'Draft International Standard' (DIS).

The final phase consists of a formal vote on the proposed standard. Acceptance of a standard requires the approval of two-thirds of the ISO members that actively participated in drafting the standard, as well as three-quarters of all members participating in the vote. Once the text has been accepted, it is published as an ISO standard.

To prevent standards from becoming outdated as the result of ongoing development, ISO rules state that standards should be reviewed, and if necessary revised, after an interval of at most five years.

Cooperation with the IEC and the CEN

ISO is not the only international standards organization. In order to avoid duplication of effort, ISO cooperates closely with the IEC (International Electrotechnical Commission). The areas of responsibility are defined as follows: the IEC covers the fields of electrical technology and electronics, while ISO covers all other fields. Combined working groups are formed to deal with themes of common interest, and these groups produce combined ISO/IEC standards. Most standards for smart cards belong to this category.

ISO and the European standardization committee CEN (*Comité Européen de Normalisation*) also agree on rules for the development of standards that are recognized as both European and international standards. This leads to time and cost savings.

International standardization of smart cards

International standards for smart cards are developed under the auspices of ISO/IEC, and on the European level by the CEN. The major industrial countries are represented in all relevant committees, and they generally also maintain 'mirror' committees in the form of national working groups and voting committees. In Germany, this responsibility is borne by the DIN. Figure 1.2 shows an overview of the structure of the relevant ISO and IEC working groups and the standards for which they are responsible.

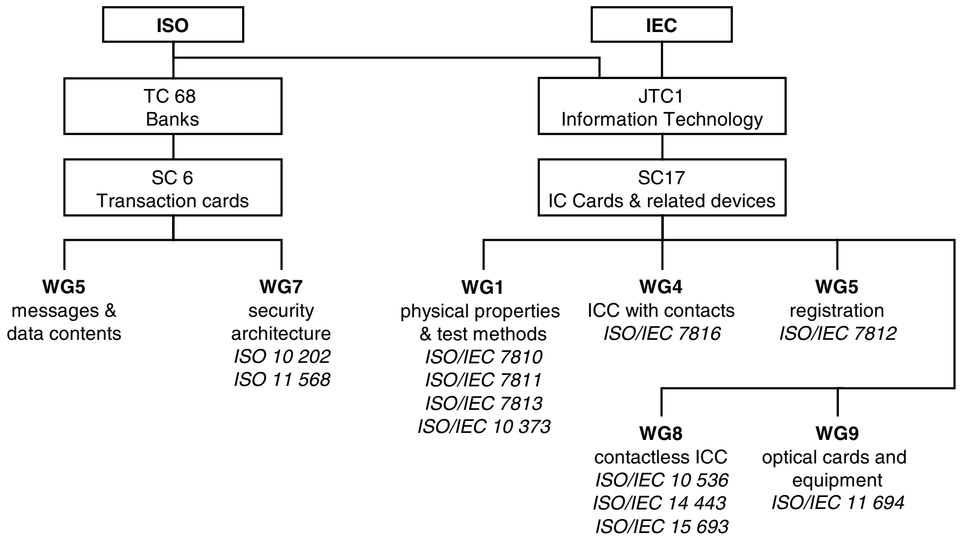

ISO
IEC
TC 68 Banks
JTC1 Information Technology
SC 6 Transaction cards
SC17 IC Cards & related devices
WG5 messages & data contents
WG7 security architecture ISO 10 202 ISO 11 568
WG1 physical properties & test methods ISO/IEC 7810 ISO/IEC 7811 ISO/IEC 7813 ISO/IEC 10 373
WG4 ICC with contacts ISO/IEC 7816
WG5 registration ISO/IEC 7812
WG8 contactless ICC ISO/IEC 10 536 ISO/IEC 14 443 ISO/IEC 15 693
WG9 optical cards and equipment ISO/IEC 11 694

Figure 1.2 Overview and organization of the working groups for international smart card standards

As can be seen, there are two technical committees that are concerned with the standardization of smart cards. The first is ISO TC68/SC6, which is responsible for the standardization of cards used in the financial transaction area, while the second is ISO/IEC JTC1/SC17, which is responsible for general applications. This division has historical roots, since the first international applications were for identification cards used for financial transactions. The number of applications has naturally increased enormously since then, so the general standards, which are looked after by the SC17 committee, have taken on greater significance. The standards specifically related to financial transactions can thus be regarded as a subset of the general standards. Brief descriptions of the standards listed in Figure 1.2, including their current status, can be found in Chapter 16, 'Appendix'.

Within CEN, the general subject of smart cards is dealt with by the TC224 committee ('Machine-readable Cards, Related Device Interfaces and Procedures'). The activities of CEN complement those of ISO. ISO standards are adopted as CEN standards where possible, which means they must be translated into the three official CEN languages (English, French and German). They may also be enlarged or reduced as necessary to comply with specific European conditions. The CEN working groups also produce application-specific standards, which would not be possible as such within ISO.

An additional European standardization body, the European Telecommunications Standards Institute (ETSI), has made a significant contribution to the widespread international use of smart cards. ETSI is the standardization body of the European telecommunications companies and telecommunication industry. The GSM 11.11 family of standards specifies the interface between the smart card (referred to as the 'subscriber identity module' (SIM) in the GSM system) and the mobile telephone. This family of standards is based on the ISO/IEC standards. With the international proliferation of GSM systems beyond the boundaries of Europe, the ETSI standards have become highly important for the smart card industry.

After more than 20 years of standardization effort, the most important basic ISO standards for smart cards are now complete. They form the basis for further, application-specific standards, which are currently being prepared by ISO and CEN. These standards are based on prior ISO standards in the 7810, 7811, 7812 and 7813 families, which define the properties of identification cards in the ID-1 format. These standards include embossed cards and cards with magnetic stripes, which we all know in the form of credit cards.

Compatibility with these existing standards was a criterion from the very beginning in the development of standards for smart cards (which are called 'integrated circuit(s) cards', or 'ICC', in the ISO standards), in order to provide a smooth transition from embossed cards and magnetic-stripe cards to smart cards. Such a transition is possible because all functional components, such as embossing, magnetic stripes, contacts and interface components for contactless interfaces, can be integrated into a single card. Of course, a consequence of this is that the integrated circuits, which are sensitive electronic components, are exposed to high stresses during the embossing process and recurrent impact stresses when the embossed characters are printed onto paper. This makes heavy demands on the packaging of the integrated circuits and the manner in which they are embedded in the card.

A summary of the currently available standards, with brief descriptions of their contents, can be found in the Appendix.[1]

In the last few years, an increasing number of specifications have been prepared and published by industrial organizations and other non-public groups, with no attempt being made to incorporate them into the standardization activities of ISO. The argument most commonly offered for this manner of working is that the way ISO operates is too slow to keep pace with the short innovation cycles of the informatics and telecommunication industries. Since

[1] See Section 16.4, 'Annotated Directory of Standards and Specifications'

frequently only a few companies are involved in drafting these ‘industry standards’, there is a large risk that the interests of smaller companies, and especially the interests of the general public, will be ignored in the process. It is a major challenge to the future of ISO to devise a working method that can safeguard general interests without hampering the pace of innovation.

2 Types of Cards

As already mentioned in the Introduction, smart cards are the youngest member of the family of identification cards using the ID-1 format defined in ISO standard 7810, 'Identification Cards – Physical Characteristics'. This standard specifies the physical properties of identification cards, such as flexibility and temperature resistance, as well as the dimensions of three different card formats: ID-1, ID-2 and ID-3. The smart card standards (ISO 7816-1 ff) are based on ID-1 cards, millions of which are used nowadays for financial transactions.

This chapter provides an overview of various types of cards in the ID-1 format, since a combination of various functions is of particular interest for many applications, especially when the cards currently used in an existing system (such as magnetic-stripe cards) are to be replaced by smart cards. In such cases, it is usually not possible to replace the existing infrastructure (such as magnetic-stripe card terminals) by a new technology overnight.

The solution to this problem generally consists of issuing cards with both magnetic stripes and chips, for use during a transition period. Such cards can be used with both types of terminals (old and new). Naturally, new functions that are only possible with a chip cannot be used with a terminal that only supports magnetic-stripe cards.

2.1 EMBOSSED CARDS

Embossing is the oldest technique for adding machine-readable features to identification cards. The embossed characters on the card can be transferred to paper using simple, inexpensive devices, and they can also be easily read visually (by humans). The nature and location of the embossing are specified in the ISO 7811 standard ('Identification Cards – Recording Technique'). This standard, which is divided into five parts, deals with magnetic stripes as well as embossing.

ISO 7811 Part 1 specifies the requirements for embossed characters, including their form, size and embossing height. Part 3 defines the precise positioning of the characters on the card and defines two separate regions, as shown in Figure 2.1. Region 1 is reserved for the card's identification number, which identifies the card issuer as well as cardholder. Region 2 is reserved for additional data relating to the cardholder, such as his or her name and address.

Smart Card Handbook, Third Edition. W. Rankl and W. Effing
© 2004 John Wiley & Sons, Ltd ISBN: 0-470-85668-8

At first glance, transferring information by printing from embossed characters may appear quite primitive. However, the simplicity of this technique has made worldwide proliferation of credit cards possible, even in developing countries. The exploitation of this technology requires neither electrical energy nor a connection to a telephone network.

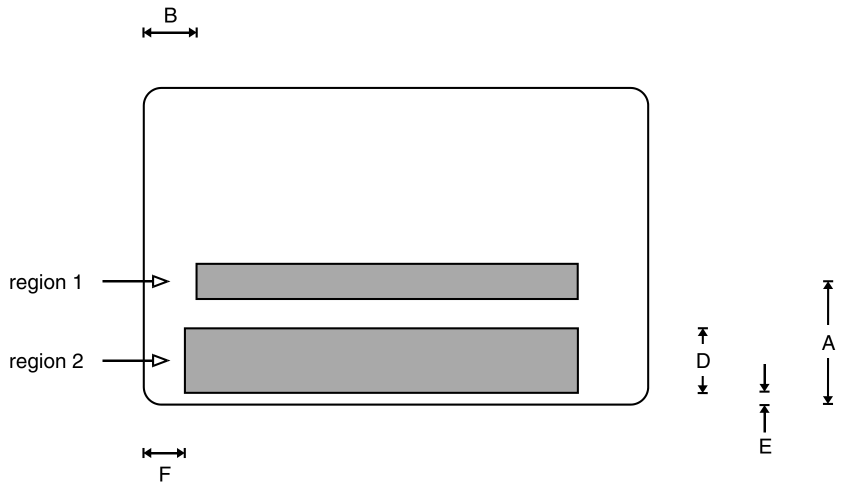


Figure 2.1 Embossing locations according to ISO 7811-3. Region 1 is reserved for the ID number (19 characters), and region 2 is reserved for the cardholder's name and address (4 × 27 characters). A = 21.42 ± 0.12 mm, B = 10.18 ± 0.25 mm, D = 14.53 mm, E = 2.41−3.30 mm, F = 7.65 ± 0.25 mm

2.2 MAGNETIC-STRIPE CARDS

The fundamental disadvantage of embossed cards is that their use creates a flood of paper receipts, which are expensive to process. One remedy for this problem is to digitally encode the card data on a magnetic stripe located on the back of the card. The magnetic stripe is read by pulling it across a read head, either manually or automatically, with the data being read and stored electronically. No paper is required to process the data.

Parts 2, 4 and 5 of ISO standard 7811 specify the properties of the magnetic stripe, the coding technique and the locations of the magnetic tracks. The magnetic stripe may contain up to three tracks. Tracks 1 and 2 are specified to be read-only tracks, while track 3 may also be written to.

Although the storage capacity of the magnetic stripe is only about 1000 bits, which is not very much, it is nevertheless more than sufficient for storing the information contained in the embossing. Additional data can be read and written on track 3, such as the most recent transaction data in the case of a credit card.

The main drawback of magnetic-stripe technology is that the stored data can be altered very easily. Manipulating embossed characters requires at least a certain amount of manual dexterity, and such manipulations be easily detected by a trained eye. By contrast, the data recorded on the magnetic stripe can be altered relatively easily using a standard read/write device, and it is difficult to afterwards prove that the data have been altered. Furthermore, magnetic-stripe cards are often used in automated equipment in which visual inspection is not possible, such as cash dispensers. A potential criminal, having obtained valid card data, can easily use duplicated cards in such unattended machines without having to forge the visual security features of the cards.

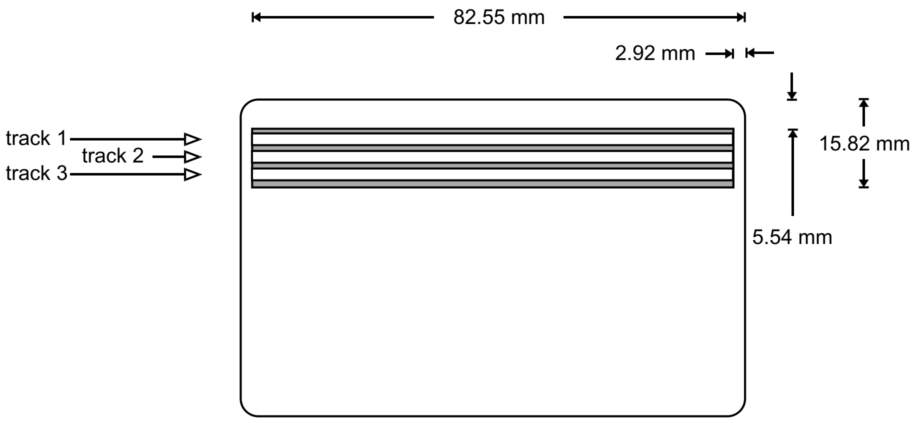


Figure 2.2 Location of the magnetic stripe on an ID-1 card. The data region of the magnetic stripe is intentionally not extended to the edges of the card, since the use of hand-operated card readers causes rapid wear at the ends of the stripe

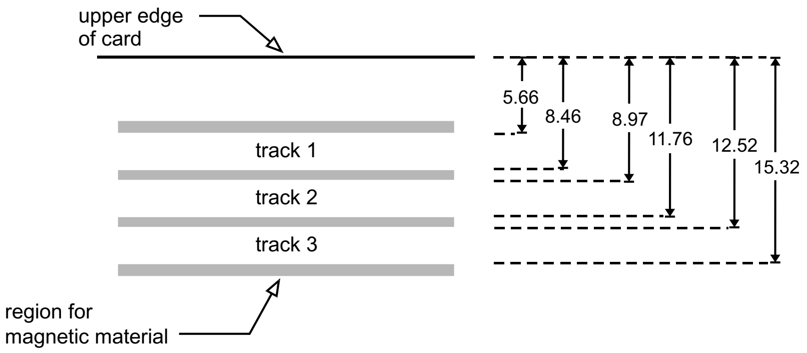


Figure 2.3 Locations of the individual tracks on an ID-1 card (all dimensions in mm)

Manufacturers of magnetic-stripe cards have developed various means to protect the data recorded on the magnetic stripe against forgery and duplication. For example, German Eurocheque cards contain an invisible, unalterable code in the body of the card, which effectively makes it impossible to alter or duplicate the data on the magnetic stripe. However, such techniques require a special sensor in the card terminal, which considerably increases the cost of the terminal. For this reason, none of these techniques has so far succeeded in becoming internationally established.

Table 2.1 Standard features of the three tracks on a magnetic-stripe card, as specified in ISO 7811

Feature	Track 1	Track 2	Track 3
Amount of data	79 characters max	40 characters max	107 characters max
Data coding	6-bit alphanumeric	4-bit BCD	4-bit BCD
Data density	210 bpi (8.3 bit/mm)	75 bpi (3 bit/mm)	210 bpi (8.3 bit/mm)
Writing	not allowed	not allowed	allowed

2.3 SMART CARDS

The smart card is the youngest and cleverest member of the family of identification cards in the ID-1 format. Its characteristic feature is an integrated circuit embedded in the card, which has components for transmitting, storing and processing data. The data can be transmitted using either contacts on the surface of the card or electromagnetic fields, without any contacts.

Smart cards offer several advantages compared with magnetic-stripe cards. For instance, the maximum storage capacity of a smart card is many times greater than that of a magnetic-stripe card. Chips with more than 256 kB of memory are currently available, and this figure will multiply with each new chip generation. Only optical memory cards, which are described in the next section, have greater capacities.

However, one of the most important advantages of smart cards is that their stored data can be protected against unauthorized access and manipulation. Since the data can only be accessed via a serial interface that is controlled by an operating system and security logic, confidential data can be written to the card and stored in a manner that prevents them from ever being read from outside the card. Such confidential data can be processed only internally by the chip's

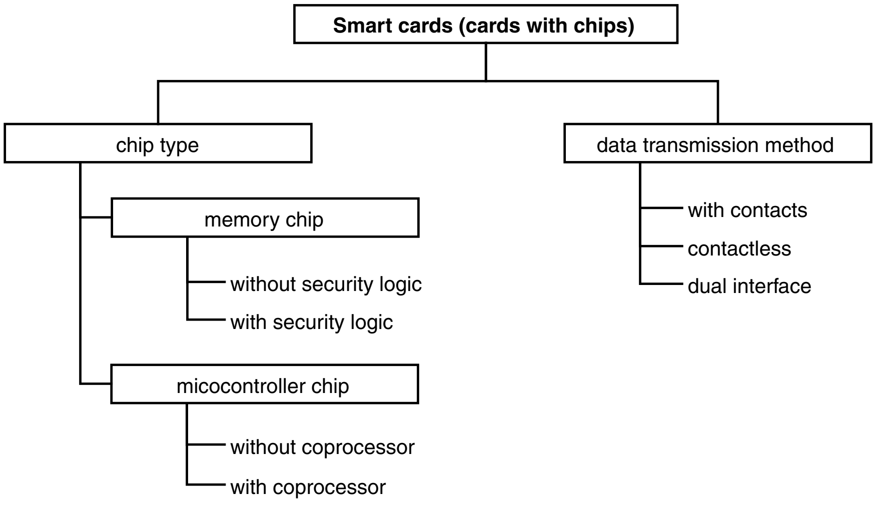


Figure 2.4 Classification chart for cards containing chips according to the type of chip used and the method used for data transmission

processing unit. In principle, both hardware and software mechanisms can be used to restrict the use of the storage functions of writing, erasing and reading data and tie them to specific conditions. This makes it possible to construct a variety of security mechanisms, which can also be tailored to the specific requirements of a particular application. In combination with the ability to compute cryptographic algorithms, this allows smart cards to be used to implement convenient security modules that can be carried by users at all times, for example in a purse or wallet. Some additional advantages of smart cards are their high level of reliability and long life compared with magnetic-stripe cards, whose useful life is generally limited to one or two years.

The fundamental characteristics and functions of smart cards are specified in the ISO 7816 family of standards, which are described in detail in the following chapters.

Smart cards can be divided into two groups, which differ in both functionality and price: *memory cards* and *microprocessor cards*.

2.3.1 Memory cards

Figures 2.5 and 2.6 show architectural block diagrams of memory cards.

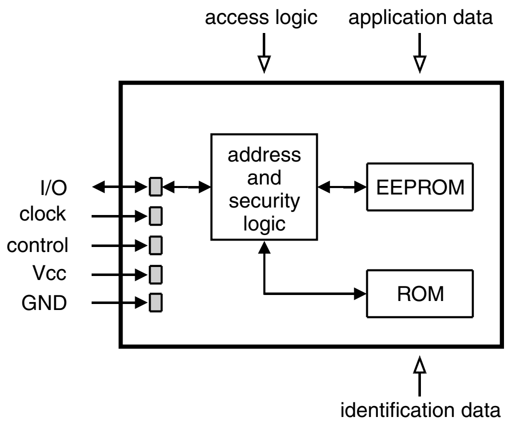


Figure 2.5 Typical architecture of a contact-type memory card with security logic. The figure shows only basic energy and data flows and is not a detailed schematic diagram

The data needed by the application are stored in the memory, which is usually EEPROM. Access to the memory is controlled by the security logic, which in the simplest case consists only of write protection or erase protection for the memory or certain memory regions. However, there are also memory chips with more complex security logic that can also perform simple encryption. Data are transferred to and from the card via the I/O port. Part 3 of the ISO 7816 standard defines a special synchronous transfer protocol that allows the chip implementation to be particularly simple and inexpensive. However, some smart cards use the I^2C bus, which is commonly used for serial-access memories.

The functionality of memory cards is usually optimized for a particular application. Although this severely restricts the flexibility of the cards, it makes them quite inexpensive. Memory cards are typically used for prepaid telephone cards and health insurance cards.

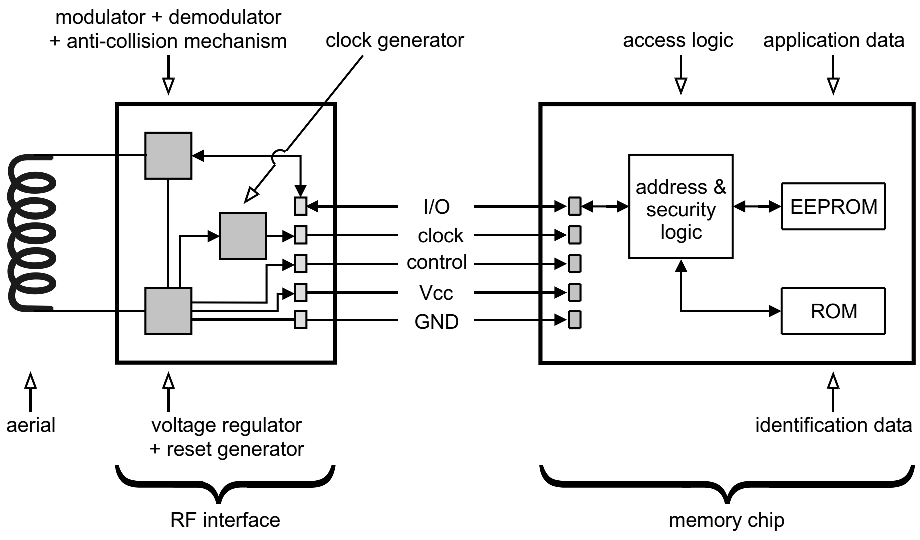


Figure 2.6 Typical architecture of a memory card with security logic and a contactless interface. The figure shows only basic energy and data flows and is not a detailed schematic diagram

2.3.2 Microprocessor cards

The heart of the chip in a microprocessor card, as the name suggests, is a processor, which is usually surrounded by four additional functional blocks: mask ROM, EEPROM, RAM and an I/O port. Figure 2.7 shows the architecture of a typical device of this type.

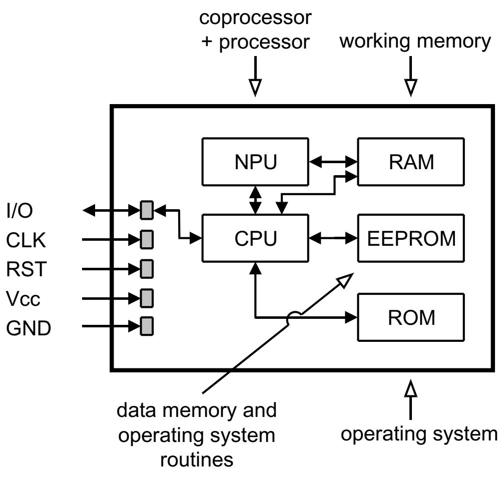


Figure 2.7 Typical architecture of a contact-type microprocessor card with a coprocessor. The figure shows only basic energy and data flows and is not a detailed schematic diagram

The mask ROM contains the chip's operating system, which is 'burned in' when the chip is manufactured. The content of the ROM is thus identical for all the chips of a production run, and it cannot be changed during the chip's lifetime. The EEPROM is the chip's non-volatile memory. Data and program code can be written to and read from the EEPROM under the control of the operating system. The RAM is the processor's working memory. This memory is volatile, so all the data stored in it are lost when the chip's power is switched off. The serial I/O interface usually consists only of a single register, via which data are transferred bit by bit.

Microprocessor cards are very flexible in use. In the simplest case, they contain a program optimized for a single application, so they can only be used for this particular application. However, modern smart card operating systems allow several different applications to be integrated into a single card. In this case, the ROM contains only the basic components of the operating system, with the application-specific part of the operating system being loaded into the EEPROM only after the card has been manufactured. Recent developments even allow application programs to be loaded into a card after it has already been personalized and issued to the cardholder. Special hardware and software measures are used to prevent the security conditions of the individual applications from being violated by this capability. Special microprocessor chips with high processing capacities and large memory capacities, which are optimized for such use, are now available.

2.3.3 Contactless smart cards

Electrical connections with contact-type smart cards are made via the eight contacts specified in the ISO 7816 Part 1 standard. The reliability of contact-type smart cards has been steadily improved over the past years as the result of experience accumulated in manufacturing such cards. The failure rate of telephone cards within their one-year service life, for instance, is currently significantly less than one in a thousand. Nevertheless, contacts are one of the most frequent sources of failure in electromechanical systems. Disturbances can be caused by factors such as contamination and contact wear. In mobile equipment, vibrations can cause brief intermittent contacts. Since the contacts on the surface of the card are directly connected to the inputs of the integrated circuit chip embedded in card, there is a risk that the chip may be damaged or destroyed by electrostatic discharge. Static charges of several thousand volts are by no means rare.

These technical problems are elegantly avoided by contactless smart cards. In addition to its technical advantages, contactless-card technology offers card issuers and cardholders a range of new and attractive potential applications. For instance, contactless cards do not necessarily have to be inserted into a card reader, since there are systems available that work at a range of up to one meter. This is a great advantage in access-control systems where a door or turnstile has to be opened, since the access authorization of a person can be checked without requiring the card to be removed from a purse or pocket and inserted into a reader. One major application area for this technology is local public transport, which requires a large number of people to be identified in the shortest possible time.

However, contactless technology is also advantageous in systems that do require deliberate insertion of the card into a reader, since it does not matter how the card is inserted in the reader. This contrasts with magnetic cards or cards with contacts, which work only with a specific card

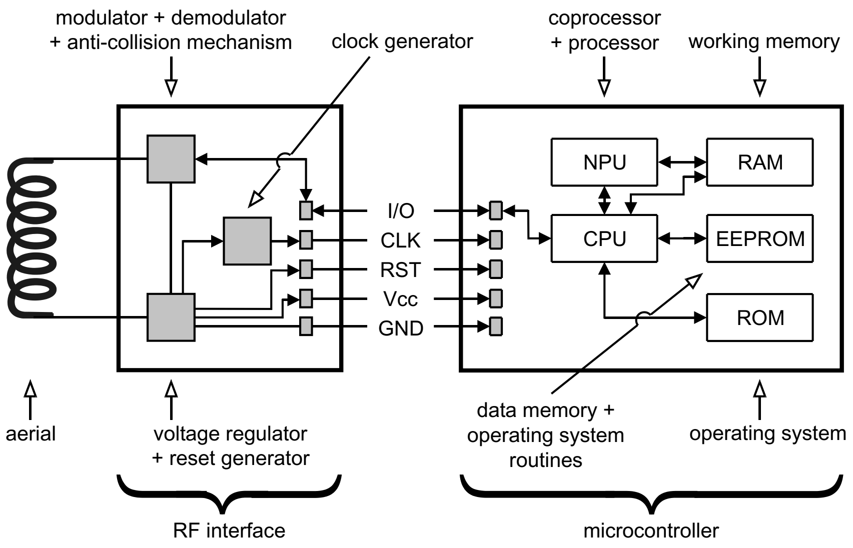


Figure 2.8 Typical architecture of a microprocessor card with a coprocessor and a contactless interface. The figure shows only basic energy and data flows and is not a detailed schematic diagram

orientation. Freedom from orientation restrictions simplifies use and thus increases customer acceptance.

A further interesting variation on using contactless cards involves a 'surface terminal'. In this case, the card is not inserted into a slot, but simply placed on a marked location on the surface of the card reader. In addition to simplicity of use, this solution is attractive because it significantly reduces the risk of vandalism (for example, forcing chewing gum or superglue into the card slot).

For card marketing, contactless technology offers the advantage that no technical components are visible on the card surface, so visual design is not constrained by magnetic stripes or contacts. However, this advantage comes at the price of more complex terminals with correspondingly higher prices. Another disadvantage is that several different systems for contactless smart cards have been standardized and marketed, further increasing the complexity of terminals that must be compatible with all standardized cards.

Manufacturing technology for the mass production of contactless cards has matured to the point that high-quality products are available at prices that do not significantly differ from those of comparable contact-type cards. Up to now, contactless cards have predominantly been used in local public transportation systems, in which they serve as electronic tickets in modern electronic-fare systems. Most of the systems presently in use employ single-function cards containing inexpensive chips with hard-wired security logic. However, there is a growing demand for incorporating value-added services in electronic tickets. Multifunction cards with integrated microprocessors are thus being used increasingly often, with the payment function most commonly being implemented using the conventional contact-based technique in order to utilize existing infrastructures, such as electronic purse systems. These new multifunction

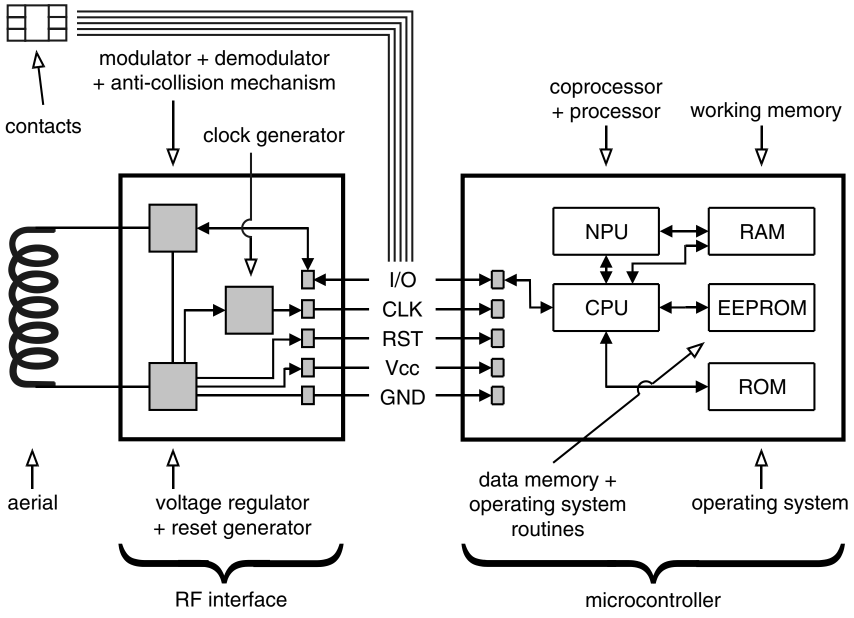


Figure 2.9 Typical architecture of a microprocessor card with a coprocessor and both contactless and contact interfaces. The figure shows only basic energy and data flows and is not a detailed schematic diagram

cards have both contact and contactless coupling elements and are called 'dual-interface cards' or 'combicards'.

The technology and operating principles of contactless smart cards are described in detail in Section 3.6, 'Contactless Smart Cards'.

2.4 OPTICAL MEMORY CARDS

For applications where the storage capacity of smart cards is insufficient, optical cards that can store several megabytes of data are available. However, with current technology these cards can be written only once and cannot be erased.

The ISO/IEC 11 693 and 11 694 standards define the physical characteristics of optical memory cards and the linear data-recording technique used with such cards.

Combining the large storage capacity of optical memory cards with the intelligence of smart cards leads to interesting new possibilities. For example, data can be written in encrypted form to the optical memory, with the key being securely stored in the private memory of the chip. This protects the optically stored data against unauthorized access.

Figure 2.11 shows the typical layout of an optical smart card with contacts, a magnetic stripe and an optical storage region. It can be seen that the area available for optical storage

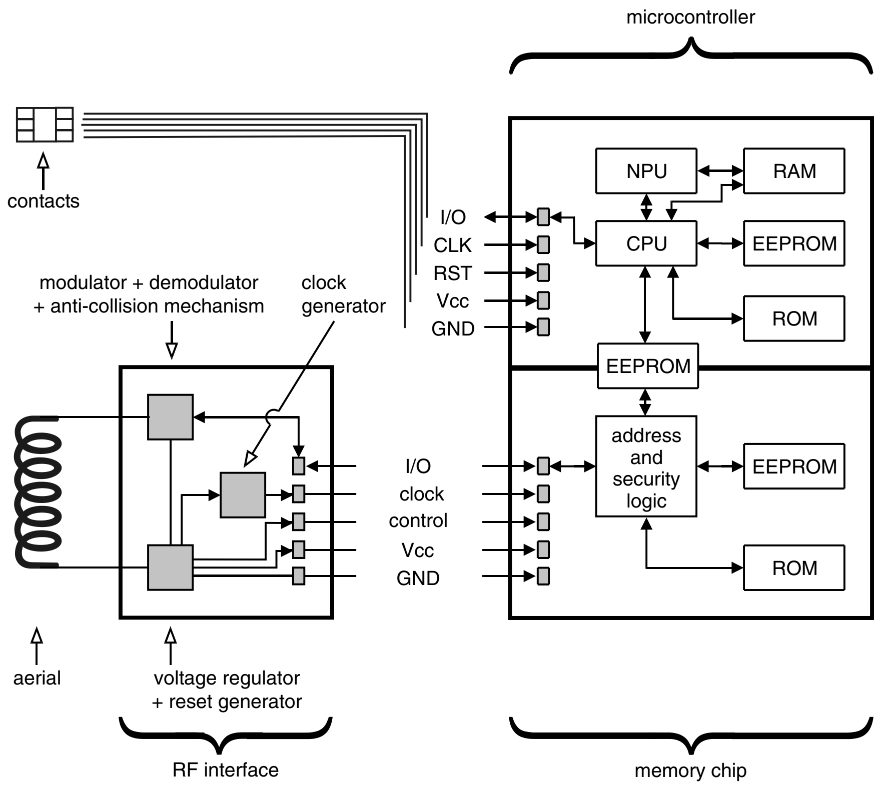


Figure 2.10 Typical architecture of a dual-interface card, which is a combination of a contactless memory card and a contact-type microprocessor card. The figure shows only basic energy and data flows and is not a detailed schematic diagram

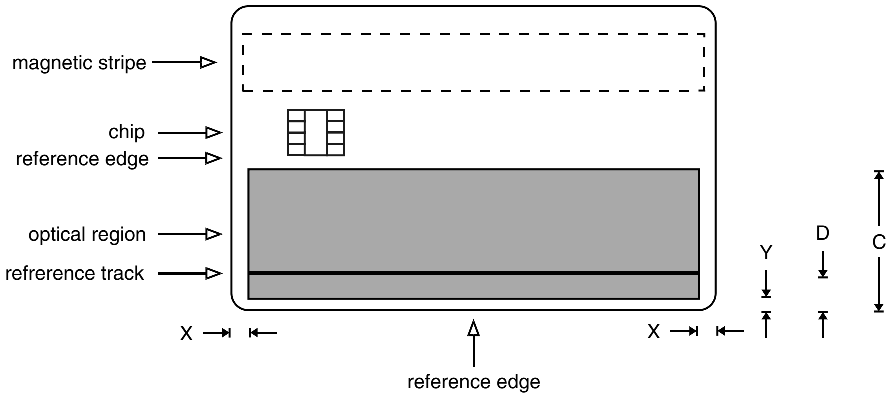


Figure 2.11 Location of the optical storage area on an ID-1 card according to ISO/IEC 11 694-2. C = 9.5–49.2 mm, D = 5.8 ± 0.7 mm, X = 3 mm max with PWM or 1 mm max with PPM, Y = 1 mm min with PWM (Y < D) or 4.5 mm max with PPM (PWM = pulse-width modulation, PPM = pulse-position modulation)

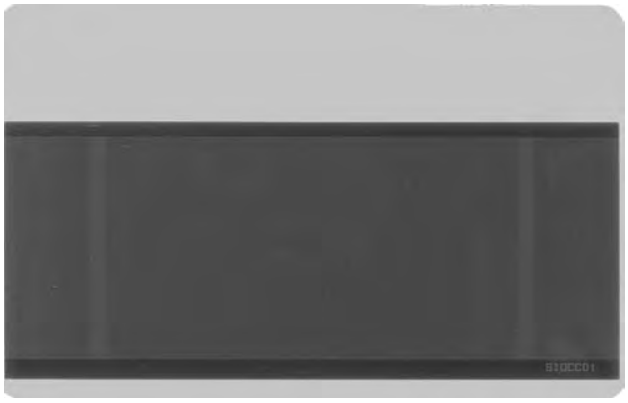


Figure 2.12 A typical optical memory card with a net storage capacity (with error correction) of approximately 4 MB. The raw capacity (without error correction) is approximately 6 MB

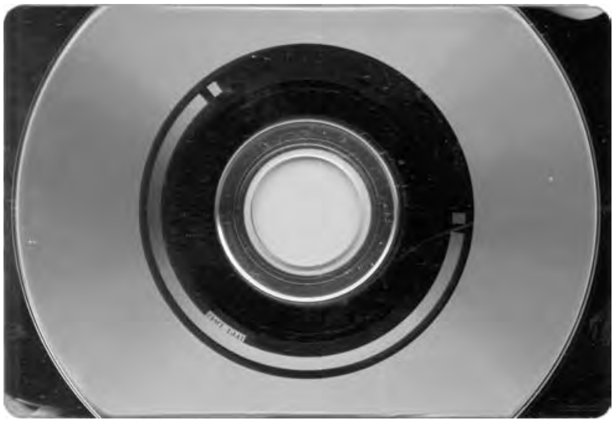


Figure 2.13 An optical memory card with a storage capacity of approximately 32 MB that can be read by a CD-ROM drive

is limited by the contacts for the chip, which naturally reduces the total storage capacity. The magnetic stripe is located on the rear of the card.

Up to now, use of optical memory cards has been severely limited by the high cost of equipment for reading and writing this type of card. One application for optical memory cards is recording patient data in the medical sector, since their large storage capacity allows even X-ray images to be stored on a card.

3 Physical and Electrical Properties

The card body of a smart card inherits its fundamental properties from its predecessor, the familiar embossed card, which still dominates the market in the credit card sector. Technically speaking, such cards are simple plastic structures that are personalized by being embossed with a variety of user features, such as the name and number of the cardholder. Later versions of these cards were provided with a magnetic stripe to enable simple machine processing. When the idea of implanting a chip in the card first arose, this existing type of card was used as the basis and a microcontroller was embedded in the body of the card. Many standards relating to the card's physical properties are thus not specific to smart cards, but apply equally well to magnetic-stripe and embossed cards.

3.1 PHYSICAL PROPERTIES

If you hold a smart card in your hand, the first thing you notice is its format. After this, you might see that it has set of contacts, although a contactless smart card may not have any visible electrical interface. The next feature to catch your eye might be a magnetic stripe, embossing or a hologram. All these features and functional components form part of the physical characteristics of a smart card.

Most of the physical characteristics are actually purely mechanical in nature, such as the format of the card and its resistance to bending or twisting. These characteristics are familiar to every user from personal experience. In practice, however, physical properties such as sensitivity to temperature or light and resistance to moisture are also important.

The interaction between the body of the card and the implanted chip must also be considered, since only the combination of the two components makes a functional card. For instance, a card body designed for use at high ambient temperatures is of little benefit if its embedded microcontroller does not share this property. These two components must individually and collectively meet all of the relevant requirements, since otherwise high failure rates can be expected in use.

Smart Card Handbook, Third Edition. W. Rankl and W. Effing
© 2004 John Wiley & Sons, Ltd ISBN: 0-470-85668-8

3.1.1 Card formats

Small cards with the typical smart card dimensions of 85.6 mm by 54 mm have been in use for a very long time. Almost all smart cards are produced in this format, which is certainly the most familiar. It is designated ID-1, and its size is specified in the ISO 7810 standard. This standard originated in 1985 and thus has nothing to do with smart cards as we know them today, as can easily be seen from the abbreviation 'ID', which stands for 'identification'. This standard simply describes an embossed plastic card with a magnetic stripe that is intended to be used for the identification of a person. When it was written, no one had thought of putting a chip into the card. The presence of a chip and location of its contacts on the card were only defined several years later in other standards.

With the variety of cards available today, which are used for all possible purposes and have a wide range of dimensions, it is often difficult to determine whether a particular card is actually an ID-1 smart card. In addition to the embedded chip, one of the best identifying features is the thickness of the card. If this measures 0.76 mm and the card contains a microcontroller, it can be considered to be a smart card in the sense of the ISO standard.

The conventional ID-1 format has the advantage of being very easy to handle. The card's format is specified such that it is not too large to be carried in a wallet, but not so small that it is easily lost. In addition, the card's flexibility makes it more convenient than a rigid object.

Nevertheless, this format does not always meet the demands of modern miniaturization. Some mobile telephones weigh only 200 g and are not much bigger than a packet of tissues. It thus became necessary to define a smaller format in addition to the ID-1 format, in order to address the needs of small terminal devices. The card used in such devices can be very small, since it is usually inserted into the device only once and remains there for good. The ID-000 format was defined for this purpose, and it bears the descriptive name 'plug-in'. This format is presently only used with GSM mobile telephones, which have very little room for a card and do not require the card to be frequently exchanged.

However, the fact that cards in the ID-000 format are inconvenient to handle, both in production and by end users, led to the development of an additional format. This format is designated ID-00, or 'mini-card'. Its dimensions are approximately halfway between those of ID-1 and ID-000 cards. This type of card is more convenient to handle and cheaper to produce, for instance because it is easier to print. However, the ID-00 definition is fairly new, and this format has not yet become established either nationally or internationally.

The formats are defined in the relevant standards in a way that simplifies measuring the card dimensions. For instance, the height and width of an ID-1 card must be such that it fits between two concentric rectangles as shown in Figure 3.1 (ignoring the rounded corners), which have the following dimensions:

• external rectangle:	width:	85.72 mm (3.375 inches)
	height:	54.03 mm (2.127 inches)
• internal rectangle:	width:	85.46 mm (3.365 inches)
	height:	53.92 mm (2.123 inches)

The thickness must be 0.76 mm (0.03 inch), with a tolerance of ±0.08 mm (±0.003 inch). The corner radii and the card's thickness are defined conventionally. Based on these definitions, an ID-1 card's dimensions can be represented as shown in Figure 3.2.

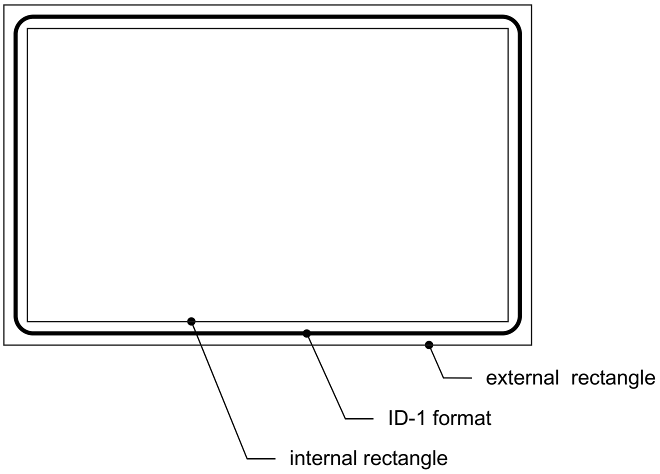


Figure 3.1 Definition of the dimensions of an ID-1 format card

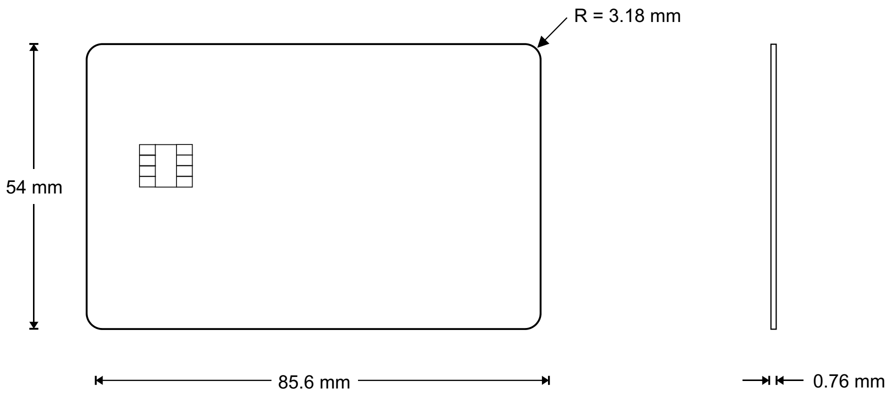


Figure 3.2 The ID-1 format. Thickness: 0.76 mm ± 0.08 mm; corner radius: 3.18 mm ± 0.30 mm. The dimensions shown indicate the size of the card excluding tolerances

The ID-000 format is also defined using two concentric rectangles. Since this format originated in Europe (based on the GSM mobile telephone system), the basic dimensions are metric. The bottom right-hand corner of a plug-in card is cut off at an angle of 45°, as shown in Figure 3.3, in order to facilitate correct insertion of the card into the card reader.

The dimensions of the two rectangles for the ID-000 format are:

• external rectangle:	width:	25.10 mm
	height:	15.10 mm
• internal rectangle:	width:	24.90 mm
	height:	14.90 mm

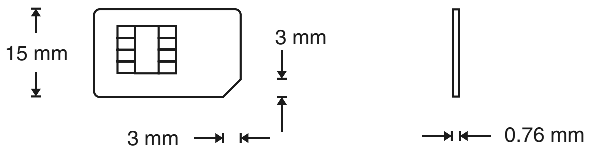


Figure 3.3 The ID-000 format. Thickness: 0.76 mm ± 0.08 mm; corner radius: 1 mm ± 0.10 mm; corner: 3 mm ± 0.03 mm. The dimensions shown indicate the size of the card excluding tolerances

The ID-00 format is also based on metric measurements. Its maximum and minimum dimensions are defined by two concentric rectangles with the following dimensions:

• external rectangle:	width:	66.10 mm
	height:	33.10 mm
• internal rectangle:	width:	65.90 mm
	height:	32.90 mm

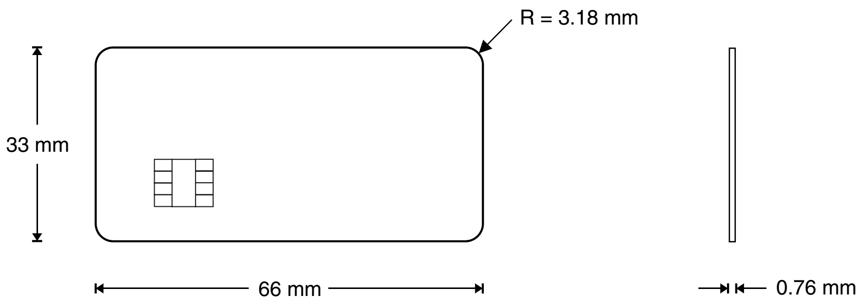


Figure 3.4 The ID-00 format. Thickness: 0.76 mm ± 0.08 mm; corner radius: 3.18 mm ± 0.30 mm. The dimensions shown indicate the size of the card excluding tolerances

The relative sizes of the ID-1, ID-00 and ID-000 formats are shown in Figure 3.5. Cards in the smaller formats can be produced from the larger versions by punching them from the body of a larger-format card. This is especially important for card manufacturers, since it allows the production process to be optimized and made more economical using a uniform ID-1 format. For instance, card manufacturers commonly produce card blanks in only one format (preferably ID-1), embed modules in them and fully personalize them. Depending on the specific application of the cards so produced, such cards can be trimmed to the desired format in a subsequent production step.

Alternatively, the format may be modified later by the customer. This has become common practice with cards for mobile telephones. The customer receives an ID-1 card that is prepunched such that it can be converted into an ID-000 card by breaking a small card free from the larger card body. In another technique, the ID-000 card is completely punched free from the ID-1 body and attached to the surrounding portion using single-sided tape on the side without contacts. The customer can thus 'produce' a card whose format fits his or her equipment, while the manufacturer only has to produce and supply one card format.

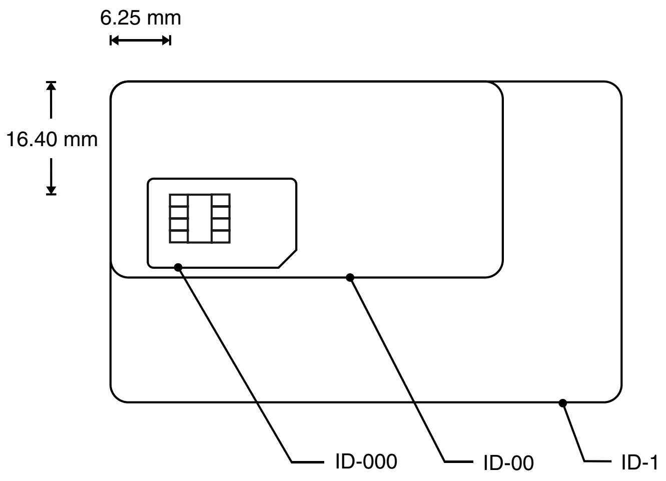


Figure 3.5 Relative sizes of the ID-1, ID-00 and ID-000 formats

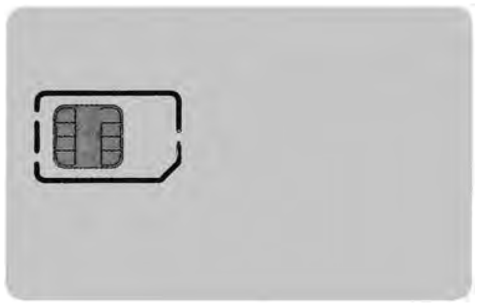


Figure 3.6 Example of a mobile telephone card in ID-1 format, which the user can convert into an ID-000 card if necessary by pressing out the smaller-format card

However, the usual card format still has some disadvantages for some applications. In such situations, other form factors can be used, such as a USB plug with an integrated smart card microcontroller. The logical behavior of such smart card variants is generally fully equivalent to that of the usual forms.

3.1.2 Card components and security features

Since smart cards are primarily used to provide authorization for specific actions or identify cardholders, security features on the card body are often needed in addition to the embedded chip. Since the authenticity of the card may be verified by humans as well as by machines, many security features are based on visual features. However, some security features employ a modified smart card microcontroller and thus can only be verified by a computer. In contrast to the security features used with microcontrollers, the usual features for human verification of

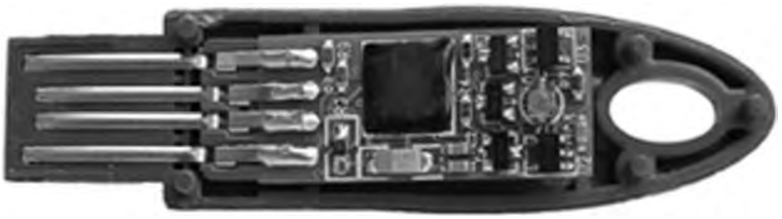


Figure 3.7 Example of a smart card with a different form factor. This photo shows a USB plug with a soldered-in smart card microcontroller and the necessary interface components, which has been opened up to reveal its internal components

the authenticity of a card are not based on cryptographic procedures (such as mutual authentication). Instead, they are primarily based on using secret materials and production processes or using processes whose mastery requires a large amount of effort or considerable expertise, or that are technically difficult.

Particularly in the area of new card components, there is considerable potential for new developments in the near future with regard to the integration of additional components such as keypads, displays, solar cells and batteries.

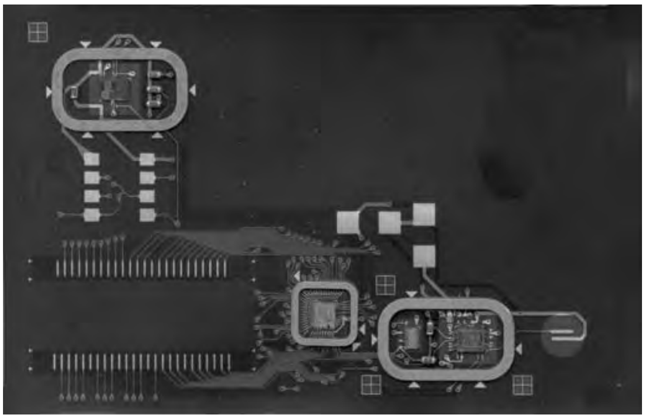


Figure 3.8 Inlay foil for a super smart card. The actual smart card microcontroller can be seen at the left, connected to the contact below it. The contact pads for a display are located at the lower left. The driver IC for the display is located to the right of the contact pads, with four large contacts for the battery above it. The contacts for a pushbutton switch can be seen at the lower right, with various components for the interface adapter located to the left (*Source:* Giesecke & Devrient)

Signature panels

A very simple way to identify the cardholder is to use a signature panel attached to the card, as is common with credit cards. Once such a panel has been signed, it cannot be altered, so it

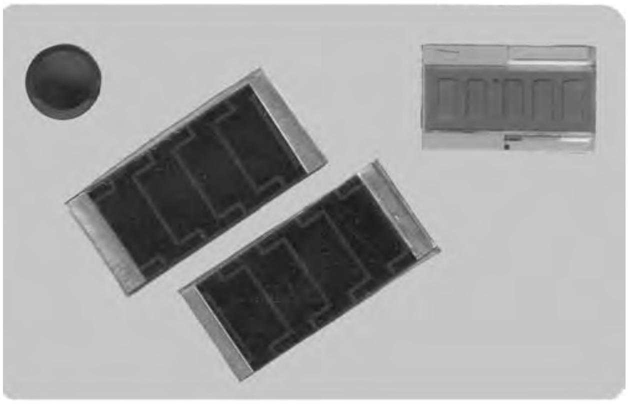


Figure 3.9 Early laboratory prototype of a super smart card for an electronic purse system using contactless cards. A pushbutton switch for confirming transactions can be seen at the upper left, with two solar cells in the middle to supply power. A five-digit display for showing the purse balance and other data is located at the upper right (*Source:* Giesecke & Devrient)

is erasure-proof. A very fine colored pattern printed on the panel makes any attempt to cover the panel immediately apparent. The signature panel is permanently bonded to the card body by using a hot-gluing process to attach a printed paper strip to the card. Alternatively, the signature panel may be part of the top layer of the card, which is laminated into the card when it is assembled.

Guilloche patterns

A somewhat more complicated technique is to place a foil printed with guilloche patterns under the transparent outer layer of the card. Guilloche patterns are decorative patterns consisting of very fine interwoven lines, usually round or oval, such as are found on some bank notes and share certificates. These patterns have such fine structures that they can presently only be produced by printing processes, and are thus difficult to copy.

Microtext

Another technique that is based on the security provided by fine printed line structures is using microtext lines. These appear be plain lines to the naked eye, but they can be recognized as text using a loupe. Like guilloche patterns, microtext cannot be photocopied.

Ultraviolet text

In order not to affect the visible layout of the card, control characters or control numbers can be printed on the card using ink that is only visible under ultraviolet light. However, this technique provides only relatively limited protection against forgery.

Barcodes

For storing a small amount of data, a barcode can be printed on the surface of the card using laser engraving or thermal-transfer printing. The advantage of barcodes is that they can be automatically read at close range using optical equipment. The barcodes used on smart cards include not only the widely use one-dimensional type, but also two-dimensional barcodes in the form of stacked or matrix barcodes. A two-dimensional matrix barcode, such as PDF 417 for example, can easily encode up to 1000 bytes, and if an integrated Reed–Solomon code is used for error correction, the data can be recovered even when up to 25 percent of the barcode area is unreadable.

Holograms

A hologram integrated into the card is a security feature that by now is familiar to all card users. The security of holograms is primarily based on the fact that they are produced by only a few companies in the world and that they are not readily available.

The holograms used for smart cards are called 'embossed' holograms. Since they can be viewed using diffuse reflected daylight, they are also referred to as 'white-light reflection holograms'. By contrast, a conventional transmission hologram must be viewed using coherent laser light. Supplementary security features that can only be seen with laser light are sometimes integrated into the hologram as well. In order to produce an embossed hologram, it is necessary to first generate a master hologram using the conventional holographic technique. A master embossing stamp is then prepared from the master hologram using a transfer process. The embossing stamp contains the microstructures that will produce the subsequent embossed holograms. Daughter stamps are prepared from the master stamp using electroplating processes, and these daughter stamps are used to emboss the hologram structure in plastic films. These films are then coated with a layer of vaporized aluminum to produce the well-known white-light reflection holograms.

The hologram is permanently bonded to the card body, so it cannot be removed without destroying it. This can be done using either a lamination process or the 'roll-on' process. In the latter process, a hologram located on a carrier film is pressed onto the card by a heated roller. The carrier film is then pulled off, and the hologram remains permanently welded to the plastic card body. A third process that is used is the 'hot-stamping' process, which is similar to the roll-on process except that a heated stamp is used instead of a heated roller.

Kinegrams

Kinegrams, which are popularly called '3-D images', are made in the same way as holograms. The viewer sees an image that changes abruptly when the viewing angle is changed. Kinegrams are just as hard to forge as holograms, and they have the advantage that they are more quickly recognized by the viewer and thus can be verified more quickly.

Multiple laser image (MLI)

A multiple laser image is a sort of kinegram that is very similar to a simple hologram. It uses an array of lenses pressed into the surface of the card, some of which have been blackened by a laser. The main difference between an MLI and a hologram is that card-specific information is shown in the small MLI image. For instance, this technique can be used to mark the name of the cardholder on an individual card in the form of a kinegram.

Laser engraving

Darkening a special plastic layer by heating it with a laser beam is called laser engraving, or simply 'lasing'. In contrast to embossing, this is a secure way to write data on an individual card, such as the cardholder's name and the card number. It is secure because the necessary equipment and the knowledge of how to use it are not readily available.

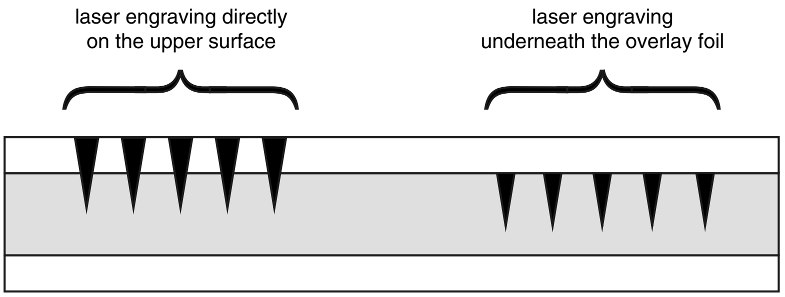

laser engraving directly on the upper surface
laser engraving underneath the overlay foil

Figure 3.10 Cross-section of laser engraving in a card (not to scale). Laser engraving can take place either on the surface of the card or in an internal layer below a cover foil that is transparent to the laser light

Two different methods are used for laser engraving: vector engraving and raster engraving. In the vector method, the laser beam is directed along its path without interruption. This is very well suited to writing characters and has the advantage of being quick. In the raster technique, by contrast, a large number of adjacent points are blackened to produce an image, similar to the operation of an ink-jet or dot-matrix printer. This method is primarily used to place a picture on the card. Although it has the advantage of high resolution, which allows details to be reproduced well, it has the disadvantage of being very time-consuming. For instance, it takes approximately 10 seconds to laser-engrave a standard-quality passport photograph.

Embossing

Another way to add user data to a card is to emboss characters onto the card. This is done by hammering metal letter punches against the card. In principle, this process works the same way as a mechanical typewriter. Nowadays, the only benefit of embossing is that the embossed characters can easily be transferred to preprinted forms using carbon paper. However, this is very important in practical use, since this is still the most widely used method of paying with a credit card.

It is very easy to manipulate embossed characters, since the plastic can easily be flattened by moderately heating the embossed characters (using an iron, for example). In order to counter this, one of the embossed characters is often placed on top of the hologram, which will be destroyed if it is heated.

Thermochrome displays

There are certain applications in which it is desirable to occasionally change the text and image(s) printed on the card. A good example is a student identification card in the form of a smart card that must be renewed twice a year. Ideally, it should be possible to visually read the expiry date without having to use any special equipment. This means that it must be printed on the card, rather than just being stored in the chip. A similar example is an electronic purse smart card, which requires using a card reader to show the current balance of the 'money' stored in the card.

Smart cards with microcontroller-driven displays are currently technically possible, but they are still too expensive for large-scale use. A thermochrome display (TC display) is a simple alternative that has some drawbacks compared with 'real' display, but is inexpensive and already available. A TC display is a supplementary card component on which characters and images can be reversibly printed (printed and subsequently reprinted) using a special card reader.

The technical operating principle is relatively simple. The thermochrome strip consists of a thin film (10–15 μm) of a temperature-sensitive material laminated to the card. This material darkens when it is heated to 120 °C. A printing head with a resolution of 200 or 300 dpi, such as is used in thermal-transfer and dye-sublimation printers, is used to heat individual points on the thermochrome strip to form characters or an image. This darkened material can be changed back to a nearly transparent state by heating the entire strip, which amounts to erasing the strip.

The thermochrome process is currently the only economical manner to present time-varying information to the user on the surface of the card such that it can be read without using any special equipment. Its major disadvantages are that it is subject to fraud and that it requires a special card reader with a built-in thermochrome printer.

The MM technique

In 1979, the German banking industry decided to include a machine-readable security feature in all German Eurocheque (EC) cards. After various potential methods were tested, the MM technique (developed by the firm GAO) was selected as the security process for these cards. This security feature is still used in all German Eurocheque cards, even though they are now equipped with microcontroller chips. The objective of this security feature was, and still is, to prevent unauthorized copying or modification of the magnetic-stripe data.

The MM technique is a typical example of a secret and very effective security feature. It has been used for two decades in millions of cards. Its basic structure is summarized in an article by Siegfried Otto [Otto 82].

The name 'MM technique' comes from the German term *moduliertes Merkmal* (modulated feature), which can be understood to refer to a machine-readable substance that is incorporated in the interior of the card body [Mayer 96]. A card is verified by reading its MM code using a special sensor and passing the code to a security module called the 'MM box'. The MM box

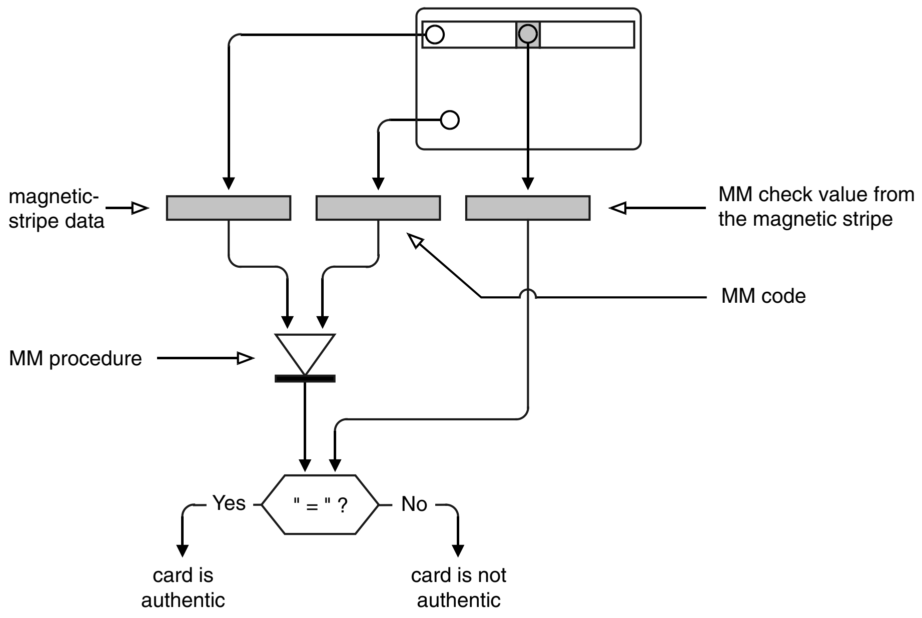


Figure 3.11 Operating principle for verifying the genuineness of a German Eurocheque card using the MM technique. The security module, or ‘MM box’, protects the MM procedure and the subsequent comparison, so only a yes/no result is reported to the higher level system

also receives the complete content of the magnetic stripe, in particular the MM check value, which is also stored on the magnetic stripe. Inside the MM box, a one-way function based on the DES algorithm is used to calculate a value from the magnetic-stripe data and the MM code. If the result of this calculation is the same as the MM check value, it can be concluded that the magnetic-stripe data matches the card.

If a valid set of magnetic-stripe data is written onto a blank card, this will be detected by the fact that the blank card does not have any MM feature. Copying the magnetic-stripe data from one EC card to another EC card will also be detected, since the MM check value will be incorrect. The MM feature is invisible, and the details of how it works and exactly where it is located in the card are secret. In addition, it is produced using materials and technology that are not commercially available.

A MM box is built into every German bank machine (ATM), as well as some POS terminals. These devices can thus check whether the magnetic-stripe data matches the card. The technique itself is not defined in any standard, and it is used only in Germany. Thanks to it, the magnetic stripes of German Eurocheque cards are protected against copying, which nowadays does not otherwise present any technical difficulties.

Security features

A large number of visual security features were developed in the period between the massive use of cards without chips and the introduction of smart cards. During this period, such features were the only way to verify the genuineness of the cards. The embedded microcontroller in the new type of card, and the cryptographic procedures that it makes possible, have diminished the importance of these features. They are nevertheless still very important whenever the genuineness of a card must be verified by a person instead of a machine, since a person cannot access the chip without special equipment.

Here we can only describe the most essential and best-known security features used with cards in a highly condensed form. There are many other types of features, such as invisible markings that can only be seen with IR or UV illumination, magnetic codes and special printing processes using rainbow-colored inks. These features are technically very interesting, but unfortunately there is not enough room to describe all of them.

In the future, security features will be found not only on the cards but also in the chips. It is conceivable that 'security' chips could could be used in the same way that bank-note paper is now used. Genuine bank notes cannot be printed without using real bank-note paper, which has specific features to show that it is genuine. In order to incorporate similar security features into chips, special chips with specifically modified hardware are necessary. A terminal can then measure the modification, which constitutes the 'feature' of the chip, and judge the genuineness of the chips from the result.

As an example of a hardware feature, suppose that computation of a fast cryptographic algorithm is implemented in supplementary hardware in a certain chip. The time required to compute a particular value could be made so short, due to the hardware implementation of the algorithm, that it would not be possible to perform the same computation using a software emulation in a different chip in an equally short time. A terminal could thus distinguish this chip from other chips by making a simple timing measurement.

There are now chips available with hardware features similar or identical to what has just been described. Naturally, they are not freely available, just as bank-note paper is not freely available. Of course, such hardware features are only suitable for very large-scale applications, due to the high cost of developing chip-specific hardware. The consequence of this, which is that such chips are almost invariably available from only one manufacturer with no possibility of an alternate source, is difficult for many card producers to accept. However, hardware-based security is an important component of the security architecture of a smart card system, and it is unfortunately not available for free.

3.2 THE CARD BODY

The materials, construction and production of the body of the card are effectively determined by the card's functional components, as well as by the stresses to which it is subjected during use. Typical functional components include:

- magnetic stripe
- signature panel
- embossing
- imprinting of personal data via laser beam (text, photo, fingerprint)
- hologram

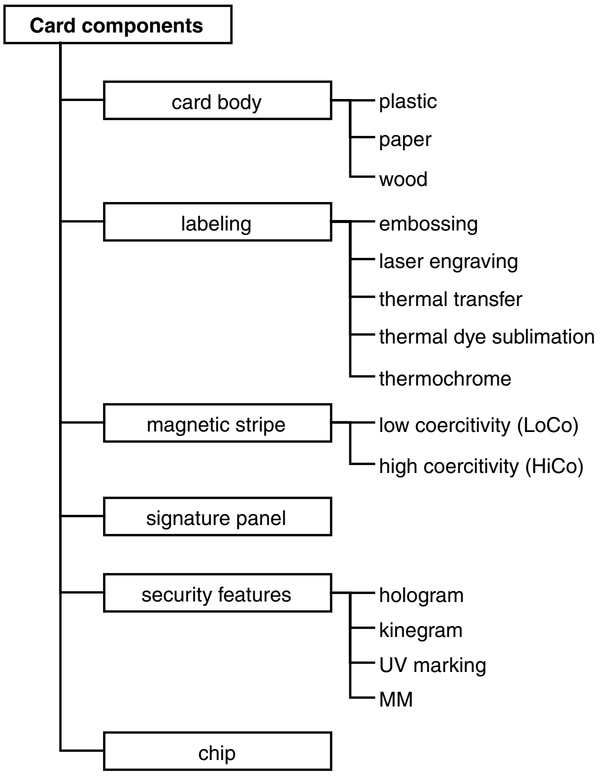


Figure 3.12 Classification scheme for card components

- security printing
- invisible authentication features (e.g. fluorescence)
- chip with contacts or other coupling elements

Clearly, even a relatively small card, only 0.76 mm thick, must sometimes contain a large number of functional components. This places extreme demands on the quality of the materials used and the manufacturing process. The minimum requirements relating to card robustness are specified in ISO standards 7810, 7813 and 7816 Part 1. The requirements essentially relate to the following areas:

- ultraviolet radiation
- X-ray radiation
- surface profile of the card
- mechanical robustness of the card and contacts

- electromagnetic susceptibility
- electrostatic discharges
- temperature resistance

The ISO/IEC 10373 standard specifies test methods for many of these requirements, to enable users and card manufacturers to objectively test card quality. The bending and twisting tests are particularly important for smart cards, since the chip, which is as fragile and brittle as glass, is a delicate foreign object in the elastic card. Special structural features are required to protect it against the mechanical stresses produced by bending and twisting the card. Chapter 9 contains a detailed list of tests and the methods used to perform them.

3.2.1 Card materials

The first material employed for ID cards, which is still widely used, is polyvinyl chloride (PVC), an amorphous thermoplastic material. It is the least expensive of all the available materials, easy to process and suitable for a wide range of applications. It is used throughout the world for credit cards. Its drawbacks are a limited lifetime, due to physical deterioration, and limited resistance to heat and cold. PVC is used in sheet form to manufacture cards, since injection molding is not possible. The worldwide production of PVC was around 13 million metric tons in 1996, of which 35,000 metric tons (0.27 %) were used for cards. PVC is considered to be environmentally hazardous, since the feedstock, vinyl chloride, is a known carcinogen. In addition, if it is burned, hydrochloric acid and (under unfavorable conditions) possibly dioxins are released. In addition, heavy-metal compounds are often used as stabilizers. Nonetheless, PVC is still by far the most widely used material for cards. This is primarily due to its low price and good processing characteristics. However, it used less and less each year due to its undesirable environmental properties. Many card issuers have decided not to use PVC for reasons of environmental policy.

To avoid the drawbacks of PVC, acrylonitrile butadiene styrene (ABS) has been used for some time to make cards. It is also an amorphous thermoplastic that is distinguished by its stability and resistance to temperature extremes. Consequently, it is often used for cards for mobile telephones, which for obvious reasons may be subjected to relatively high temperatures. ABS can be processed both in sheet form and by injection molding. Its major drawbacks are limited ink acceptance and low weathering resistance. Although the feedstock for ABS production, benzene, is a carcinogen, ABS has no other known environmental drawbacks.

For applications in which extreme stability and durability are required, polycarbonate (PC) is used. It is typically used for identity cards, and it is incidentally the base material for compact discs and DVDs. Due to its high thermal stability, relatively high temperatures are needed to apply holograms or magnetic stripes using the hot-stamp process. This can easily cause problems, due to the limited thermal stability of the materials being applied. The main drawbacks of polycarbonate are its low degree of resistance to scratching and very high cost compared with other card materials. A further drawback is that phosgene and chlorine are needed for the production of polycarbonate, and both of these materials are environmentally

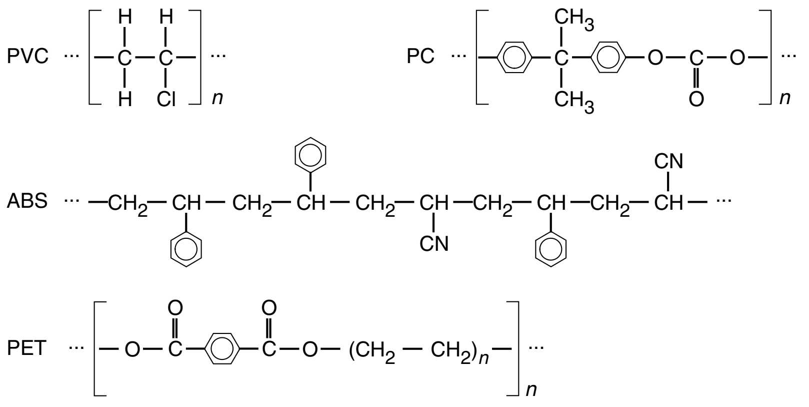


Figure 3.13 Structural formulae of the most important materials used for card bodies

problematical. Polycarbonate cards can be easily recognized by the characteristic 'tinny' sound they produce when dropped on a hard surface.

An environmentally friendly material that is mainly used as a PVC substitute is polyethylene terephthalate (PET), which has been used for a relatively long time to make packaging materials. It is commonly known as polyester. This thermoplastic material is used in smart cards in both its amorphous form (A-PET) and its crystalline form (PETP). Both types are suitable for processing in sheet form, as well as by injection molding. However, PETP is difficult to laminate, which makes additional processing steps necessary in the manufacturing process.

Numerous attempts have been made to find new or better materials for card bodies besides the usual materials (PVC, ABS, PC and PET). One example is cellulose acetate, which although having good environmental properties, has up to now proven to be poorly suited to the mass production of cards. Truly different materials, such as paper, have been frequently discussed, but as yet they have never been used in any significant quantity. The requirements imposed on cards, in terms of cost, durability and quality, are after all very high, and they can presently only be met by plastics.

In 1996 and 1997, Danmønt[1] conducted a field trial using around 600 cards that, while they did not represent a real alternative to plastic card bodies, were at least an interesting (or amusing) idea. The cards were laminated from eight layers of birch wood, each 0.1 mm thick. These cards did not meet the requirements of the various tests specified in ISO 10 373, such as those for bending and twisting, and they were naturally not suitable for embossing. However, around 90 % of their users expressed a positive reaction and said that they experienced no problems with their cards. Unfortunately, a birchwood card is not especially innovative from an environmental perspective, since the layers must be laminated using a plastic adhesive and the usual printing processes are required.

[1] See [a la Card 97]

Table 3.1 Summary of the characteristics of the standard materials for card bodies.[2] The relative cost is based on the cost of PVC

Characteristic	PVC	ABS	PC	PET
Primary use	credit cards	cellular-telephone cards	identification cards	health-insurance cards
Principal feature	inexpensive	thermally stable	durable	environmentally friendly
Temperature range	65–95 °C	75–100 °C	<160 °C	<80 °C
Cold tolerance	moderate	high	moderate	moderate
Mechanical stability	good	good	good	very good
Embossing	good	poor	good	good
Printing	good	moderate	moderate	moderate
Hot stamping (e.g. holograms)	good	good	difficult	good
Laser engraving	yes	poor	good	good
Typical lifetime	≈2years	≈3years	≈5years	≈3years
Share of worldwide card production (1998)	85 %	8 %	5 %	2 %
Relative cost	1	2	7	2.5
Environmental aspects	stabilizers contain heavy metals burning may release dioxins	base material benzene is a carcinogen burning may release prussic acid	phosgene and chlorine needed for production burning does not release dangerous materials	the most environmentally friendly material burning does not release dangerous materials
Special features	negative public image		low resistance to scratching	

3.2.2 Chip modules

The most important component of a smart card is naturally the chip. Of course, this very fragile component cannot be simply laminated to the surface of the card like a magnetic stripe. Instead, it needs a sort of enclosure to protect it from the rough everyday life of the card. This enclosure is the called the chip module. In addition to protection from ambient conditions,

[2] Based in part on [Houdeau 97, Grün 96]

chips for contact-type smart cards need six or eight contacts, which provide power to the chip and allow data communications with the terminal. A portion of the module's surface serves to provide these electrical contacts to the outside world. Naturally, the chip module should be as inexpensive as possible.

A wide variety of module designs have been devised in the course of the development of smart cards in order to meet these two technical requirements – protection of the fragile semiconductor chip and provision of contact surfaces. The most important of these are shown in Figures 3.14 and 3.15.

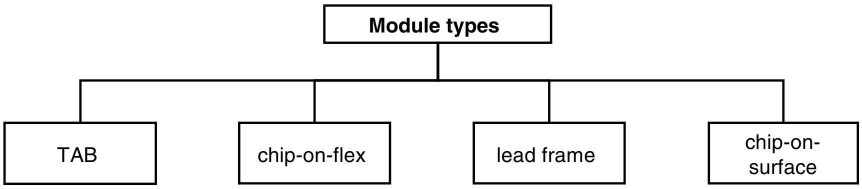


Figure 3.14 Classification of the various types of chip modules

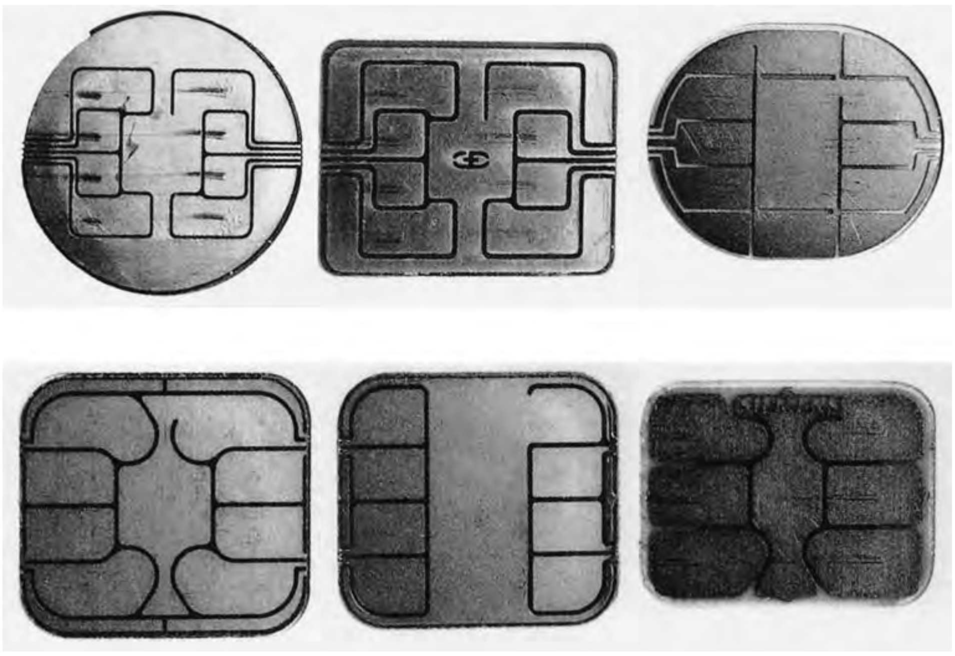


Figure 3.15 These examples illustrate the evolution of the chip-on-flex process, starting with one of the first eight-contact chip-on-flex modules at the upper left and proceeding to contemporary modules with six or eight contacts

3.2.2.1 Electrical connections between the chip and the module

Electrical connections are required between the chip inside the module and the contacts on the outside of the module. Presently, two processes are primarily used for this. In the wire-bonding process, an automatic bonding machine attaches gold wires with a diameter of only a few micrometers between the chip and the rear surfaces of the contacts. The wires are electrically attached to the chip and the module using ultrasonic welding. With this process, the contact arrangement on the top surface of the chip is always opposite that of module. This has been a standard process in the semiconductor industry for some time, and it can be readily used for mass-producing chip modules. However, each chip must be electrically connected to the module by five wires, which naturally costs time and money.

The die-bonding process was developed to further reduce the cost of fitting chips into modules. In this process, the electrical connections between the chip and module are not made with wires. Instead, the connections are made by mechanically attaching the chip to the rear surface of the module.

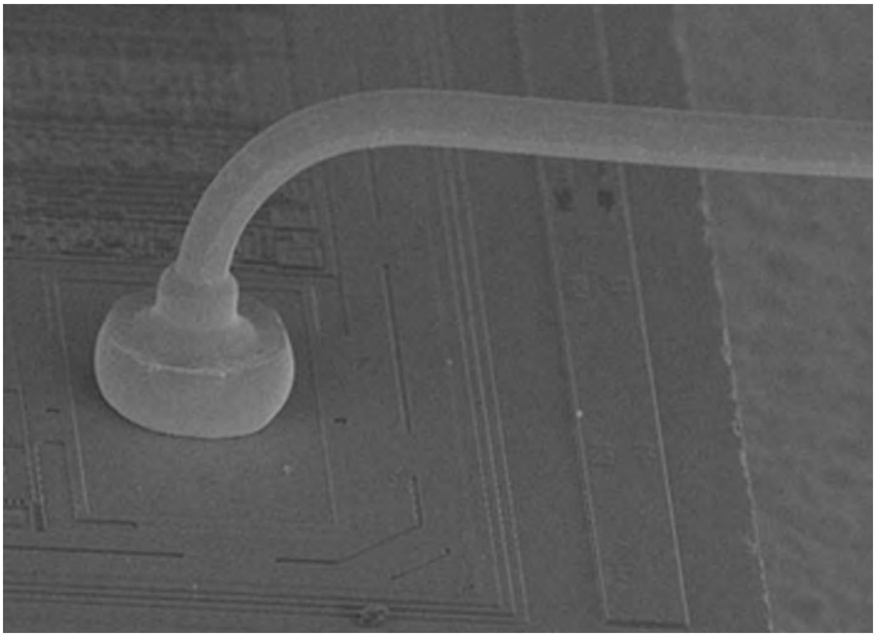


Figure 3.16 Photograph of the contact zone between a bonding wire and a bonding pad of a smart card microcontroller, magnified 1000 times (*Source:* Giesecke & Devrient)

3.2.2.2 TAB modules

Tape-automated bonding (TAB) was a standard process for large-volume chip packaging at the beginning of the 1990s, but it is presently not commonly used, since it has become technically obsolescent and too expensive. It is described here primarily for the sake of completeness.

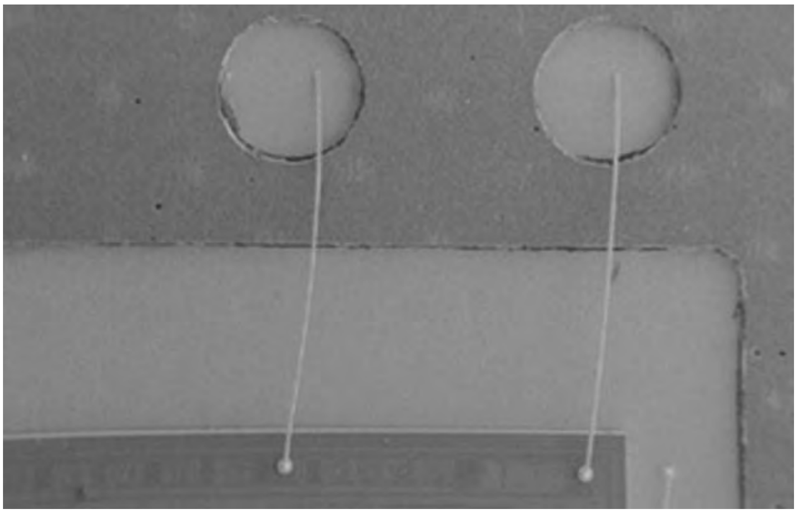


Figure 3.17 View of the electrical connections between a smart card microcontroller (bottom) and the chip module (top), magnified 400 times (*Source:* Giesecke & Devrient)

A chip module produced using the TAB process is shown in Figure 3.18. The special feature of this process is that metallic bumps are first electrically attached to the pads of the chip, and the leads of the carrier film are then soldered to these bumps. The solder connections are so sturdy that no additional support is required for the chip, which hangs from its leads. The active surface of the chip is protected against ambient conditions by an encapsulation material. The advantages of the TAB process are the mechanical strength of the connections to the chip and the low profile of the module. However, these advantages come at the price of higher costs compared with other module preparation processes.

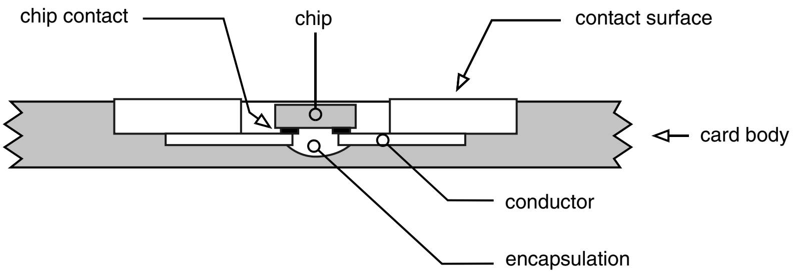


Figure 3.18 Cross-section of a chip module using the TAB process

Fitting a TAB module into a smart card is not easy, since the module must be taken into account in preparing the lamination foils for the card. Before the layers are laminated, suitable openings are punched in them, and the chip module is then inserted. The chip module is subsequently welded to the body of the card during the lamination process. This process provides a highly reliable bond between the chip module and the card body. It is nearly impossible to remove the chip from the card without destroying the card.

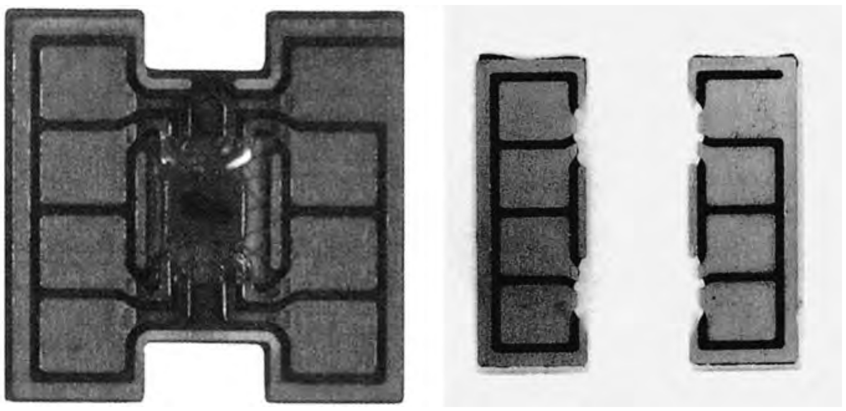


Figure 3.19 A TAB module ready for embedding in a smart card (left), and a TAB module fitted in a smart card (right)

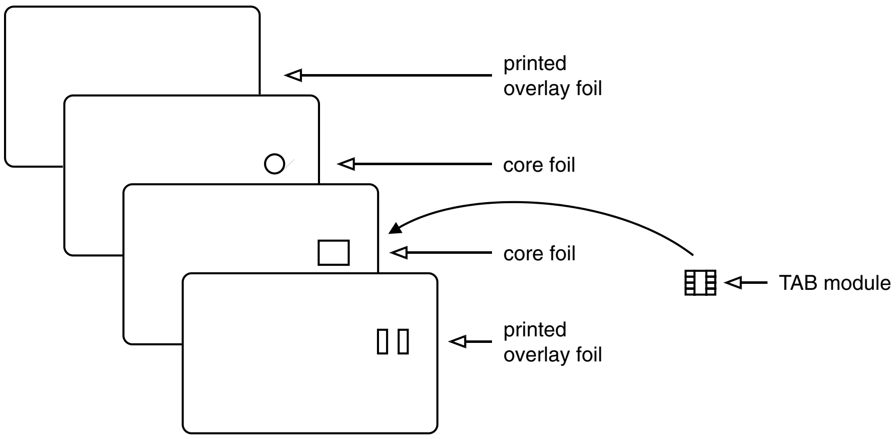


Figure 3.20 Inserting a TAB module during the lamination process

3.2.2.3 Chip-on-flex modules

Currently, the chip-on-flex module with wire-bonded contacts is the most widely used type of module. The construction of such a module is shown in cross-section in Figure 3.21. With this process, an opening into which the chip module can be glued is milled into the finished card body.

The carrier material is a flexible circuit board made of fiberglass-reinforced epoxy resin with a thickness of 120 μm. The contacts are formed from a layer of copper laminated onto the carrier, with a thickness of 35 or 75 μm. The contact surfaces are electroplated with gold in a later process step to protect them against processes that could adversely affect their electrical conductivity, such as oxidation. Holes are punched into the carrier to receive the chips and wire bonds. The chips, which are around 200 μm thick, are taken from the sawn wafer by a pick-and-place robot and fitted into the openings in the circuit board. Next, the chip contacts are connected to the rear surfaces of the contacts using bonding wires a few micrometers in diameter. Finally, the chip and the bonding wires are encapsulated in a blob of synthetic resin to protect them against ambient conditions. The total thickness of the finished module is typically around 600 μm.

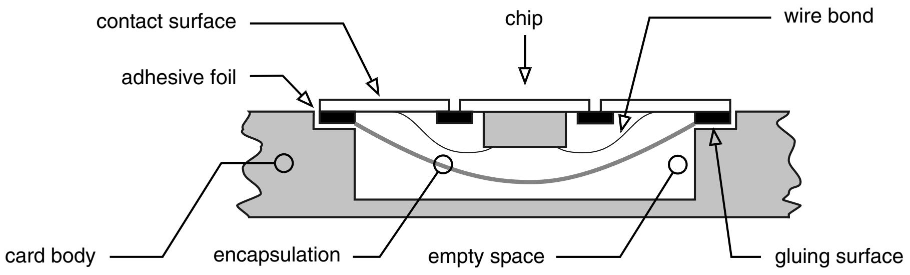


Figure 3.21 Cross-section of a chip-on-flex chip module

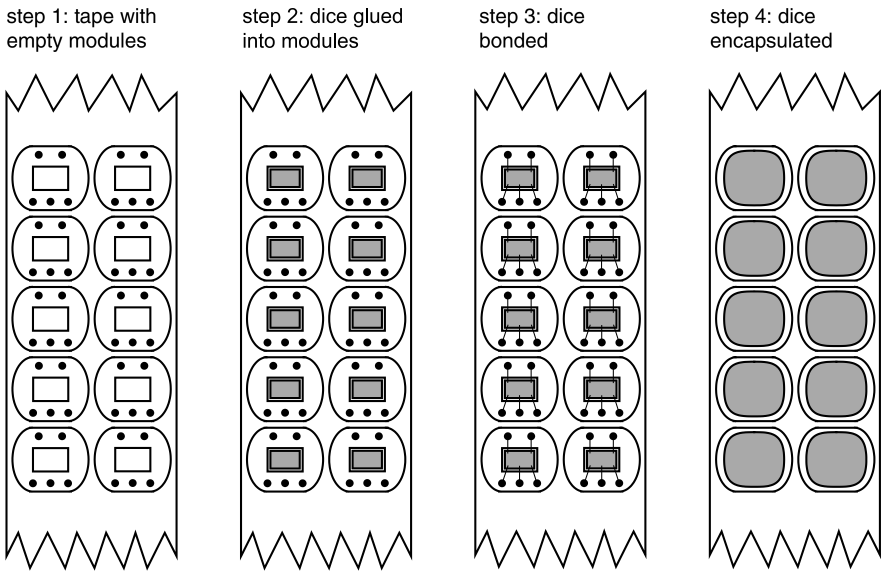

step 1: tape with empty modules
step 2: dice glued into modules
step 3: dice bonded
step 4: dice encapsulated

Figure 3.22 The four main process steps in the production of chip-on-flex modules

The advantage of this process is that it is largely based on a standard process used in the semiconductor industry for fitting chips in standard packages. It does not require as much specialized experience as the TAB process, so it less expensive. This process also lends itself well to producing very complex card bodies with many active components. This is because defective card bodies can be separated from the rest before the expensive chip modules have been fitted. The disadvantage of this process is that the thickness and the surface dimensions of the chip module are significantly greater that those of a TAB module, since not only the chip but also the bonding wires must be covered by the protective encapsulation. This is particularly disadvantageous, in that the standard smart card thickness of 0.76 mm does not leave a lot of room for overly thick modules.

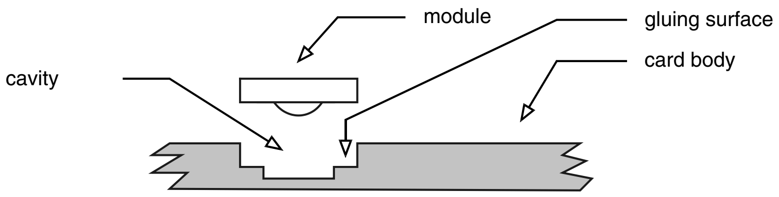


Figure 3.23 Inserting the chip module in a milled opening in the card body

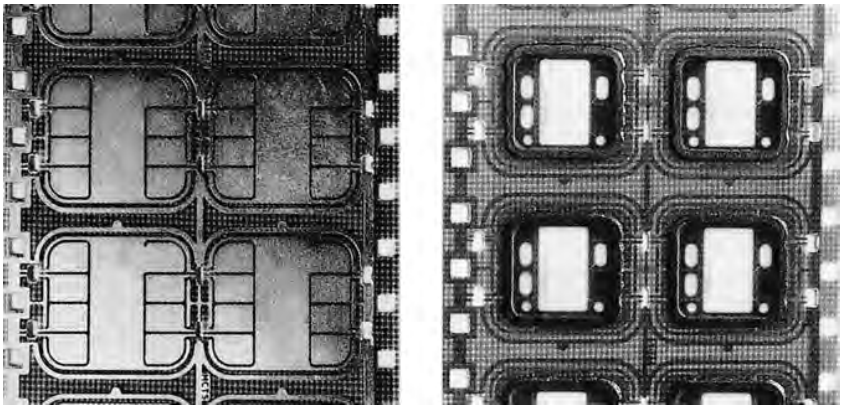


Figure 3.24 Front and rear views of chip-on-flex modules on 35-mm tape. The five openings in the carrier circuit board, for the bonding wires that make the electrical connections to the chip, can be clearly seen in the rear view

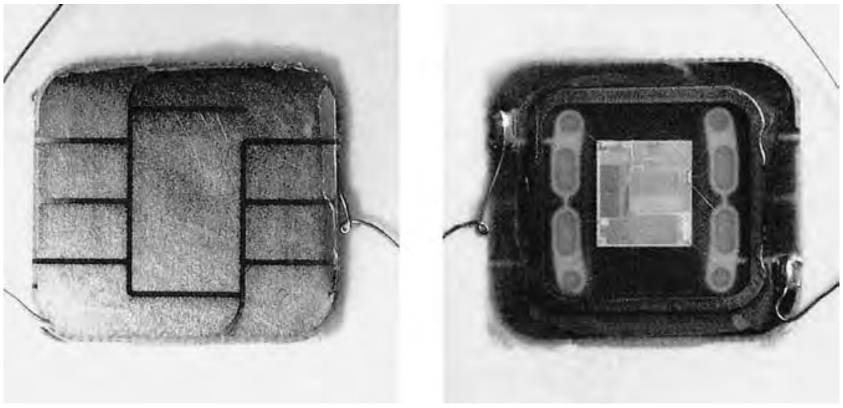


Figure 3.25 Front and rear views of a chip-on-flex module for a dual-interface card

3.2.2.4 Lead-frame modules

Technically, the TAB and chip-on-flex processes leave something to be desired, since they both provide little scope for reducing production costs. In the TAB process, producing the card body is very costly due to the characteristics of the module, while in the chip-on-flex process, the complexity of the module and the use of wire bonding lead to unfavorable production costs. These problems led to the development of a new type of module, the lead-frame module, which is mechanically just as robust as TAB and chip-on-flex modules but has lower production costs. The structure of a lead-frame module is relatively simple. The contacts, which are stamped from a gold-plated copper alloy, are held together by a plastic mold body. The chip is placed onto the lead frame by a pick-and-place robot and then connected to the backs of the contacts using wire bonding. Next, the chip is covered by a protective blob of opaque epoxy resin, usually black. The lead-frame process is currently one of the least expensive processes for making chip modules, without any accompanying reduction in the mechanical robustness of the modules.

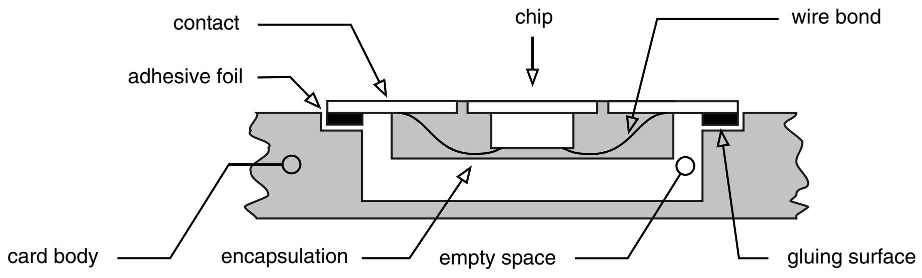


Figure 3.26 Cross-section through a lead-frame chip module

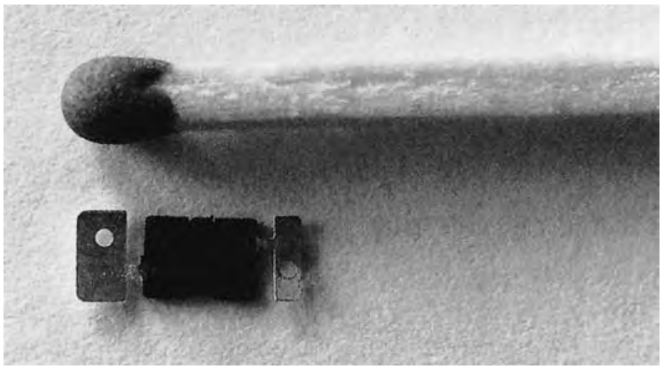


Figure 3.27 Stamped-out lead-frame module with the two coil connections for a contactless smart card, with a match for comparison

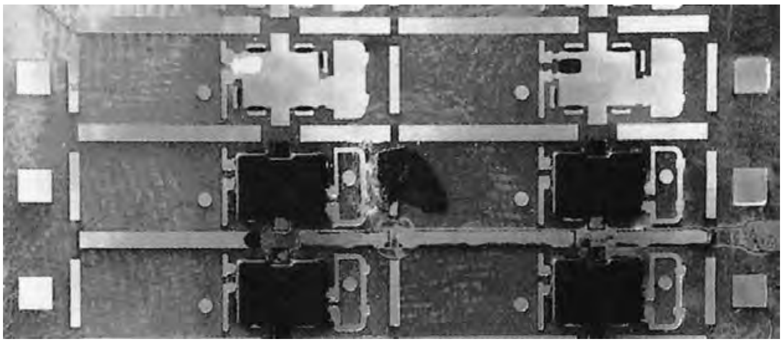


Figure 3.28 Lead-frame modules for contactless smart cards, arranged in pairs on a 35-mm tape. The two empty locations for modules that have been stamped out can be seen at the top

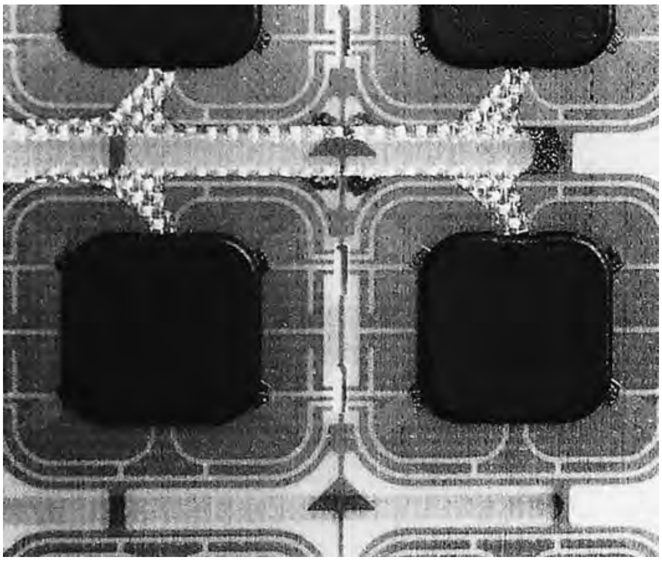


Figure 3.29 Lead-frame modules for smart cards with contacts, arranged in pairs on a 35-mm tape

3.2.2.5 The chip-on-surface process

For chips with relatively small surface areas, a process available since the mid-1990s offers a technically very interesting alternative to the usual process of fitting chips into modules. With the MOSAIC (Microchip on Surface and in Card) process, developed by Soliac [Sligos], no module is needed for the chip, since it is located directly in the card body.

The MOSAIC process is suitable for chips whose surface area is around 1 mm^2. This presently limits its application to pure memory chips, since microcontrollers are still too large for this process. The process works as follows: first, a laser is used to remove material from the location where the chip is to be placed, and then the chip is glued into this recess. In the next step, a conductive silver paste is silk-screened onto the surface of the chip and the card body, thus forming contact surfaces and connecting them to the chip at the same time. In the final step, the chip and the leads to the contacts are covered with a non-conductive lacquer. This provides electrical insulation and protects them against external ambient conditions.

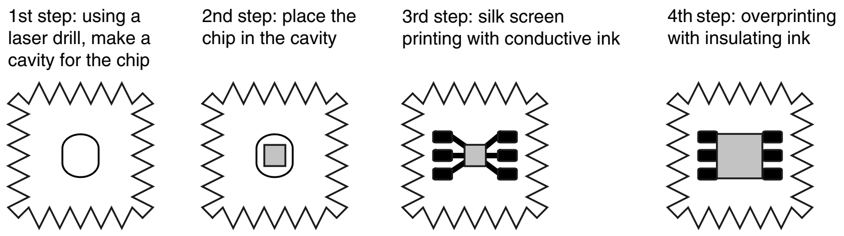

1st step: using a laser drill, make a cavity for the chip
2nd step: place the chip in the cavity
3rd step: silk screen printing with conductive ink
4th step: overprinting with insulating ink

Figure 3.30 The four stages in the production of a smart card using the chip-on-surface process

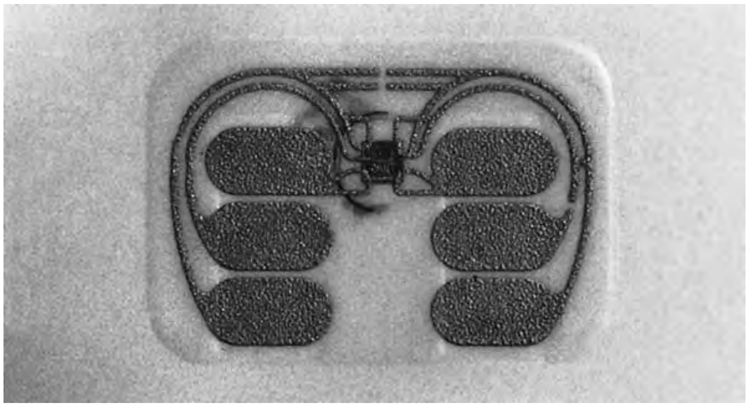


Figure 3.31 A memory chip with an edge length of 0.5 mm (0.25 mm^2 area) and ISO/IEC 7816-3 contact surfaces, fitted to a telephone card along with its contacts using the chip-on-surface process

As can clearly be seen from the figure, the chip-on-surface process is highly suitable for mass production of large numbers of cards, since it essentially consists of only a brief laser milling of the card body and two printing processes. However, this process requires an extremely precise printing process to ensure that the contacts for the chip are located correctly. Up to now, the card body has been primarily made of polycarbonate, which is especially suitable for the chip-on-surface process. The production capacity for finished cards lies in the region of 5000 pieces per hour per machine.

Another process is the flip-chip process in which the chip is placed with its face against the rear surface of the module and electrically bonded, after which the assembled module is filled with a casting resin. This type of low-cost module is usually referred to as FCOS (flip-chip on substrate).

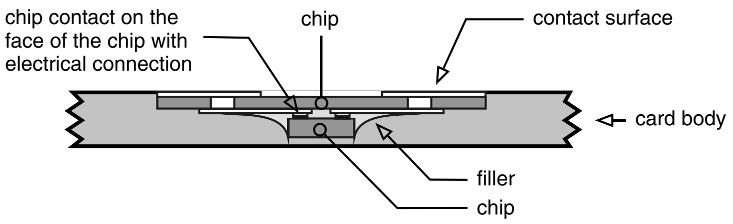

chip contact on the face of the chip with electrical connection
chip
contact surface
card body
filler
chip

Figure 3.32 Cross-section of a chip module made using the flip-chip process

3.3 ELECTRICAL PROPERTIES

The electrical properties of smart cards depend solely on the embedded microcontroller, since it is the only component of the card with an electrical circuit. This situation will undoubtedly change in the future with the addition of other components to cards, such as displays, keypads and the like, but it will take some time before these new types of smart cards are widely used.

The application that from the very beginning has imposed many rigid requirements on the electrical properties of smart cards is mobile telecommunications using the GSM system. This system, which features an extremely large variety of technically different types of terminal devices made by an equally large variety of manufacturers, which must work with a variety of card types that is at least as large, has for a long time imposed extremely severe requirements. Due to the large number of smart cards used in the GSM system, the electrical characteristics specified for GSM cards have become general guidelines for all manufacturers of smart card microcontrollers. It can be assumed that nearly all new microcontrollers for smart cards will comply with the general electrical parameters of the relevant GSM specifications, since they otherwise would be practically unsellable in the telecommunications market.

In the early days of smart card technology, quite often the primary consideration was that the implanted microcontroller was functional, and less attention was paid to its general electrical properties, such as current consumption. At that time, the applications were almost exclusively closed, and they used a single type of card with a terminal specifically designed to match that type of card. The electrical properties of the smart card were relevant only in the sense that they had to be constant, since the terminal was designed to work with a particular type of microcontroller. However, the present situation is completely different. With current large-scale applications, in which various types of smart cards must work together with many different types of terminals, it is an unavoidable requirement that all of the cards that are used are either electrically identical or at least behave uniformly within clearly defined electrical regions.

The general international basis for the electrical properties of smart cards is the ISO/IEC 7816-3 standard and its associated amendment (Amd. 1). The amendment will be incorporated into the standard in the next major revision and thus disappear as a separate document. This standard specifies all of the fundamental electrical requirements for smart cards, such as the voltage ranges, maximum current consumption and the activation and deactivation sequences.

As usual with international standards, ISO/IEC 7816-1 provides a range of options that in many cases is too extensive for practical use. This allowed supplementary industry standards

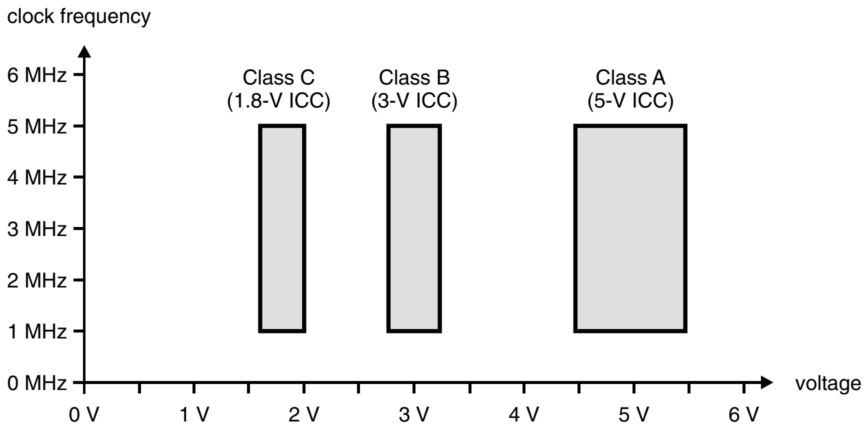


Figure 3.33 Comparison of the three possible classes of fixed voltage and clock frequency ranges as specified in ISO/IEC 7816-3 and ISO/IEC 7816-3 Amd. 1

to become established in the form of EMV 2000 for financial transactions and GSM 11.11, GSM 11.12, GSM 11.18 and TS 102.221 for telecommunications applications. These industry standards by no means compete with the ISO/IEC 7816-1 standard, but instead complement it with meaningful restrictions arising from practical smart card applications using millions of issued cards.

The distinction between smart cards used for financial transactions and smart cards used for telecommunications came about because certain requirements proved to have fundamentally different natures in these two application areas. For example, in the financial transactions area the current consumption and voltage range parameters are fully non-critical, since the terminals used for such applications are connected to the public power network. The situation in the telecommunications area is completely different, since every milliwatt counts when the objective is to achieve the longest possible operating time for a battery-powered mobile telephone. Consequently, the requirements for the least possible current consumption and low supply voltages are highly important in this application area.

Table 3.2 provides a summary of the most important electrical requirements of the essential international standards and industry standards. More detailed information is provided in the following sections.

3.3.1 Electrical connections

Smart cards have either six or eight contacts on the front side, which form the electrical interface between the terminal and the microcontroller in the card. All electrical signals are passed via these contacts. However, according to ISO/IEC 7816-2, two of the eight contacts (C4 and C8) are reserved for the auxiliary contacts AUX1 and AUX2, which can be used in the future for interfaces such as USB. Presently, some smart card modules have only six contacts, since this

Table 3.2 Summary of three electrical parameters (voltage, current and clock rate) for the most important international and industrial standards for smart cards. The tolerance ranges for the maximum current can be found in the relevant standards

Standard and class	Voltage	Clock rate	Maximum current
ISO/IEC 7816-3 and ISO/IEC 7816-3 Amd. 1			
Class A	5 V ± 10 % ⇒ 4.5–5.5 V	1–5 MHz	60 mA at 5 MHz
Class B	3 V ± 10 % ⇒ 2.7–3.3 V	1–5 MHz	50 mA at 4 MHz
Class C	1.8 V ± 10 % ⇒ 1.62–1.98 V	1–5 MHz	30 mA at 4 MHz
Class A, B and C, with clock stopped			0.5 mA (clock stop)
EMV 2000	5 V ± 10 % ⇒ 4.5–5.5 V	1–5 MHz	50 mA for all clock rates
GSM 11.11 5-V SIM	5 V ± 10 % ⇒ 4.5–5.5 V	1–5 MHz	10 mA (operating) 200 μA at 1 MHz (idle) 200 μA (clock stop)
GSM 11.12 3-V SIM	3 V ± 10 % ⇒ 2.7–3.3 V	1–5 MHz	6 mA at 3.3 V / 5 MHz (operating) 200 μA at 1 MHz (idle) 100 μA (clock stop)
GSM 11.18 1.8-V SIM	1.8 V ± 10 % ⇒ 1.62–1.98 V	1–5 MHz	4 mA at 1.8 V / 5 MHz (operating) 200 μA at 1 MHz (idle) 100 μA (clock stop)
TS 102.221			
Class A, from reset to application selection	5 V ± 10 % ⇒ 4.5–5.5 V	1–5 MHz	10 mA at 5 MHz (operating) 200 μA at 1 MHz (idle)
Class A, during an application-specific session			60 mA at 5 MHz
Class B, from reset to application selection	3 V ± 10 % ⇒ 2.7–3.3 V	1–5 MHz	7.5 mA at 5 MHz or 6 mA at 4 MHz (operating) 200 μA at 1 MHz (idle)
Class B, during an application-specific session			50 mA at 5 MHz
Class C, from reset to application selection	1.8 V ± 10 % ⇒ 1.62–1.98 V	1–5 MHz	5 mA at 5 MHz (operating) 4 mA at 4 MHz (operating) 200 μA at 1 MHz (idle)
Class C, during an application-specific session			30 mA at 5 MHz

slightly reduces manufacturing costs. However, they have the same functionality as modules with eight contacts.

The contacts are numbered sequentially from top left to bottom right. Figure 3.34 shows the ISO designations and electrical assignments of the eight defined contacts.

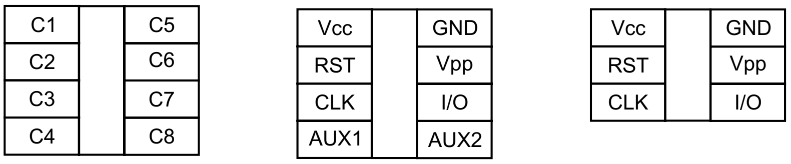


Figure 3.34 Electrical assignments and numbering of smart card contacts, per ISO 7816-2

Until the late 1980s, it was necessary to apply an external voltage to program and erase the EEPROM, since the microcontrollers then in use did not have charge pumps. Contact C6 was reserved for this purpose. However, since the early 1990s it has been standard practice to generate this voltage directly in the chip using a charge pump, so this contact is no longer used. Nevertheless, it cannot be employed for some other function, as this would conflict with the ISO standard. Thus, every smart card has a contact that has no real function, but which must still be present. Since the programming voltage contact lies between two others that are necessary for the operation of the card, it cannot simply be eliminated. This somewhat reduces the drawback of having a superfluous contact.

Table 3.3 Contact designations and functions according to ISO 7816-2

Contact	Designation	Function
C1	Vcc	Supply voltage
C2	RST	Reset input
C3	CLK	Clock input
C4	AUX1	Supplementary contact (Auxiliary 1)
C5	GND	Ground
C6	Vpp	Programming voltage (long since no longer used)
C7	I/O	Input/output for serial communications
C8	AUX2	Supplementary contact (Auxiliary 2)

3.3.2 Supply voltage

The supply voltage for smart cards was originally 5 volts, with a maximum tolerance of $\pm 10\,\%$. This voltage, which is the same as that used for conventional TTL circuits, was the standard value for all commercial smart cards and all applications.

As with other semiconductor components, increasingly smaller structure widths of semiconductor components and the need for reduced current consumption has made it necessary to markedly reduce the operating voltage range. This has been given extra impetus by the mobile telephone sector. The market-driven demand for reducing the weight of mobile telephones required changing from 6-V batteries to 3-V types, and since all other components for mobile telephones were available in 3-V technology, for a while the smart card was the only component in a mobile telephone that still needed 5 V. Consequently, an expensive voltage converter was needed to provide electrical power to the smart card, resulting in an avoidable extra cost.

Consequently, in the international standards the voltage range for smart cards was first extended to 3–5 V with a tolerance of ±10 %. This yields an effective range of 2.7 V to 5.5 V. However, it can already be foreseen that this extension will be insufficient. Consequently, the revised version of ISO/IEC 7816-3 and ISO/IEC 7816-3 Amd. 1 will again be revised in the relatively near future to permit smart cards using a supply voltage of 1.8 V with a tolerance of ±10 %.

The extended voltage range does not pose a problem for the microprocessor or most types of memory, particularly since the core voltage for semiconductors built with 0.13-μm technology is usually only 1.8 V. However, EEPROMs are also integrated into smart card microcontrollers. These EEPROMs and their associated charge pumps form the greatest obstacle to low-voltage smart cards. Nevertheless, with a certain amount of technical ingenuity it is certainly possible to integrate EEPROMs and their charge pumps into microcontrollers that can work over a supply voltage range of 1.62 to 5.5 V.

The ISO/IEC 7816-3 standard and its amendment define three classes for characterizing the voltage ranges of smart cards. Class A covers the voltage range of 5 V ±10 %, Class B covers the range of 3 V ±10 % and Class C covers the range of 1.8 V ±10 %. All three classes can be used individually or in any desired combination. For instance, if a smart card meets the requirements for both Class A and Class B, it can be used with both 5-V and 3-V supply voltages. However, it must be borne in mind that the range between 3.3 V and 4.5 V lies outside the specified ranges, so the smart card need not necessarily be able to operate in this range. Nevertheless, smart cards can usually be used without any problems between the upper and lower limits of the specified voltage ranges.

The ISO/IEC 7816-3 standard imposes yet another equally important requirement, which is that under no circumstances may the microcontroller of a smart card be damaged if the card is powered from voltage not supported by the microcontroller. This is an essential requirement for ensuring the upward compatibility of new types of smart cards with older types of terminals. The objective is to eliminate the possibility that using a 3-V card in a 5-V terminal, for example, could destroy the IC in the card.

The three possible voltage ranges defined by ISO/IEC 7816-3 and ISO/IEC 7816-3 Amd. 1 have not yet been used in the area of smart cards for financial transactions. In this area, the original operating voltage of 5 V ± 10 % still prevails, since stationary terminals can easily provide a voltage of 5 V without additional technical components.

The situation for smart cards used in the telecommunications area is completely different. In this area, smart cards that can be used only with 5-V supplies ('5-V-only cards') have fully disappeared. Since the end of the 1990s, 3 V has become the standard operating voltage for GSM devices. In the area of the new UMTS mobile telephone network, it is already clear that there will not be any mobile telephones that support 5 V, with the 3-V supply voltage being supported only for reasons of compatibility. In the medium term, the future standard operating voltage will be 1.8 V.

The ISO/IEC 7816-3 standard specifies a particular procedure for selecting the supply voltage, which essentially amounts to trying the voltages for each of the three classes in turn. As soon as the terminal can receive an ATR, it analyzes the ATR to see whether the smart card prefers a particular class. If so, the terminal initiates a new activation sequence using the desired class. If the ATR does not include any information about the voltage range, the smart

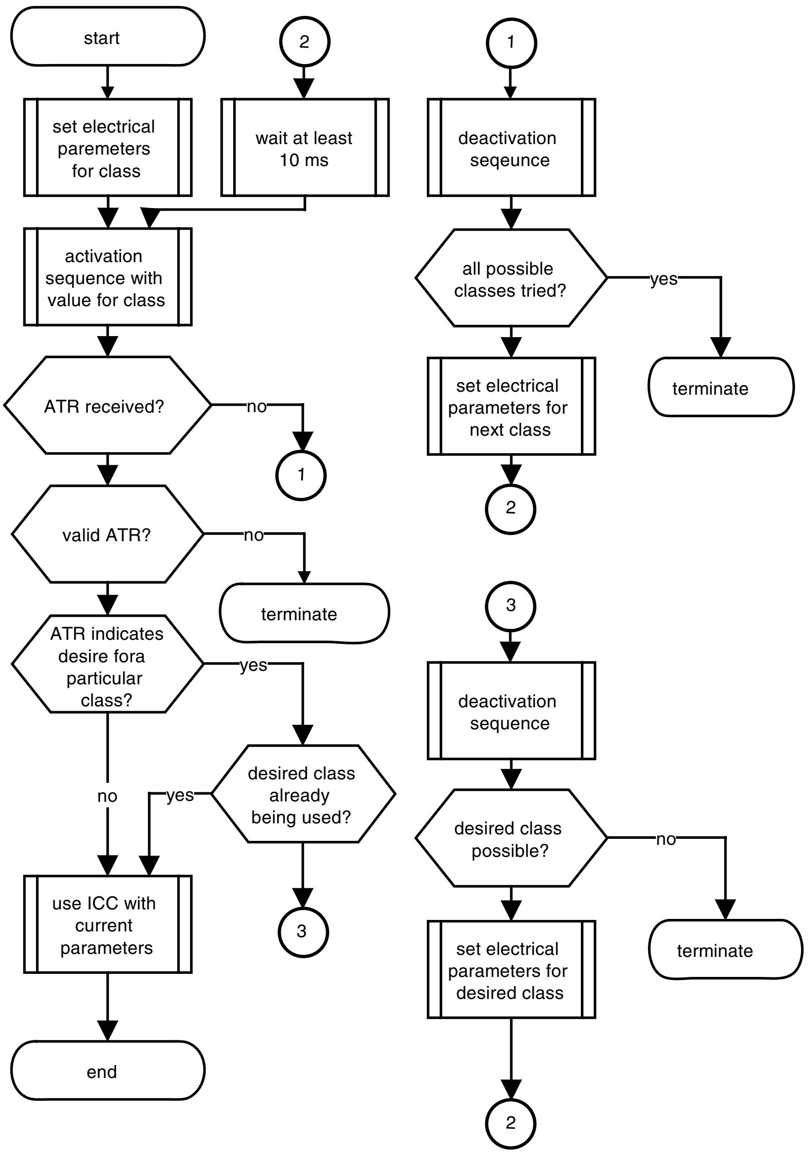

start
2
1
set electrical paremeters for class
wait at least 10 ms
deactivation seqeunce
activation sequence with value for class
all possible classes tried?
yes
ATR received?
no
set electrical parameters for next class
terminate
1
2
valid ATR?
no
terminate
3
ATR indicates desire fora particular class?
yes
deactivation sequence
no
yes
desired class already being used?
desired class possible?
no
use ICC with current parameters
3
set electrical parameters for desired class
terminate
end
2

Figure 3.35 Flow chart showing the actions taken by a terminal when selecting the operating voltage based on the classes specified in ISO/IEC 7816-3 and ISO/IEC 7816-3 Amd. 1. With mobile end-user equipment, the process is usually started using the lowest supply voltage

card will be used with the voltage with which the first ATR could be received. The selection procedure is shown in the form of a flow chart in Figure 3.35.

It is already certain that an additional voltage class will be introduced in the future to allow a supply voltage of 1.2 V. Particularly in the light of the steadily decreasing structure widths of semiconductor devices, in the medium term this could become a typical operating voltage for microcontrollers.

3.3.3 Supply current

The card's microcontroller obtains its supply voltage, and thus its supply current, via contact C1. According to the GSM 11.11 specification, this current may not exceed 10 mA. The current must lie below a certain value to allow the hardware in the terminal to be designed to supply a corresponding maximum current. The first version of the ISO/IEC 7816-3 standard in 1989 specified a maximum current of 200 mA with a 5-V supply voltage and 5-MHz clock, but even then that was too much. Since that time, the values have been significantly reduced and made dependent on the various supply voltage classes.

The most important factor is that the current consumption of a microcontroller is directly proportional to both the applied clock frequency and the supply voltage. It is also somewhat dependent on the temperature of the microcontroller. The current version of ISO/IEC 7816-3 specifies a maximum current of 60 mA for voltage class A (5 V) at a maximum clock frequency of 5 MHz and a maximum ambient temperature of 50 °C.

With regard to smart cards for financial transactions, in the EMV 2000 specification the ISO/IEC 7816-3 value for the maximum current is reduced from 60 mA to 50 mA, but there are no other significant supplementary restrictions. In the telecommunications sector, current consumption has been a critical factor since the very beginning. Consequently, in this sector are there is a complicated set of rules specifying the maximum current as a function of the clock rate and operating state of the smart card. A detailed presentation of the maximum current for the various voltage classes is given in Table 3.2.

A technically interesting innovation has been introduced for smart cards that conform to the USIM specification. Here two different operating states are specified with regard to current consumption. The first state encompasses the time from the reset until selection of the application, while the second state comprises the subsequent application-specific session. The maximum allowable current consumption in the first state is significantly lower than in the second state. Furthermore, the mobile telephone can use an application-specific data object to determine the current demand of the currently selected application. This is because the current consumption of a smart card is considerably higher when its numeric coprocessor or internal frequency multiplier is enabled in order to achieve higher performance. With this mechanism, it would at least be theoretically possible to have a mobile telephone use only those 'smart applications' whose current consumption it can adequately support. Unfortunately, the corresponding USIM specification does not include any procedure to allow a mobile telephone and a smart card to negotiate the maximum available current, as with PPS. With the current version of the specification, the only option available to a mobile telephone if its smart card demands too much current is to deactivate the smart card.

Modern microcontrollers for smart cards have current consumptions on the order of 350 μA per megahertz of clock frequency. Using this value, we can write the following formula for the current consumption of a microcontroller as a function of the applied clock frequency or the clock frequency generated inside the chip:

$$I = \frac{f}{2.875} \bullet \frac{\text{mA}}{\text{MHz}}$$

This formula is useful for making initial estimates, but it must be remembered that the current consumption depends not only on the clock frequency, but also on the supply voltage, the temperature and of course the type of chip.

With a supply voltage of 5 V and an assumed current consumption of 60 mA, a smart card has a power consumption of 300 mW. This value is so low that there is no need to be concerned about overheating of the chip while it is operating, even though this amount of power is dissipated over an area of approximately 20 mm^2.

All smart card microcontrollers have one or more special power-saving modes. The operating principle of such modes is based on disabling all of the functional components of the chip that are not being used. In principle, only the interrupt logic of the I/O interface, the processor registers and the RAM need to remain energized in order to save the current operating state. In practice, the processor often remains energized as well, but the ROM and EEPROM are switched off. When the microcontroller is in this sleep mode, or idle state, its current consumption drops dramatically, since most parts of the chip are isolated from the supply voltage. In addition to this sleep mode, many smart card microcontrollers support another mode in which the applied clock can be switched off, called the ‘clock stop mode’. The main purpose of this mode is to allow the hardware components in the terminal that generate the clock to be switched off, which makes this mode particularly attractive for battery-operated terminal devices. According to ISO/IEC 7816-3, the maximum allowable current in the sleep mode with the clock stopped is 500 μA for all three classes. Even this value is too high for the mobile telecommunications area. For instance, GSM 11.11 specifies an upper limit of 200 μA for 5-V smart cards at a clock frequency of 1 MHz.

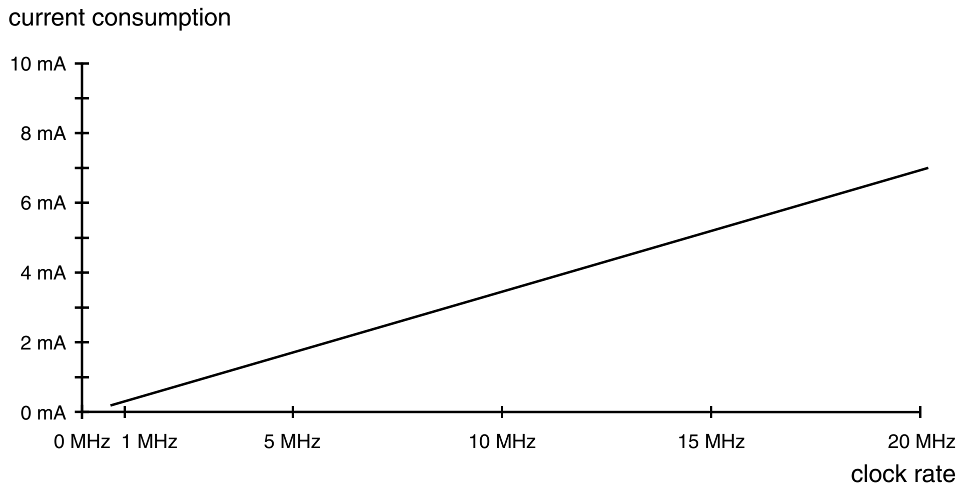


Figure 3.36 Microcontroller current consumption versus clock frequency in the normal operating mode (not the sleep mode). The current consumption in the sleep mode with the clock applied is also linearly dependent on the clock frequency and is approximately 50 μA at 5 MHz, depending on the microcontroller type

Another important detail regarding the supply current causes severe headaches for terminal manufacturers who choose to ignore it. All current microcontrollers employ CMOS technology. Under certain conditions, large short-circuit current can occur briefly during transistor switching processes. These produce current spikes that are many times greater than the nominal operating current, with durations in the nanosecond range. These spikes can also occur when the EEPROM charge pump switches on. If the terminal cannot supply such large currents during these short intervals, the supply voltage will drop below the permitted value. This can produce a write error in the EEPROM or trigger the undervoltage detector in the chip.

For this reason, references to such spikes can now be found in practically every relevant standard and specification. For instance, ISO/IEC 7816-3 requires power sources for class-A (5-V) cards to be able to handle spikes with a maximum duration of 400 ns and a maximum amplitude of 100 mA. Assuming a triangular spike, this amounts to a charge of 20 nA–s that must be supplied. This requirement can be met in a simple manner by connecting a 100-nF ceramic capacitor between circuit ground and the supply voltage line very close to the contacts for the card.

3.3.4 External clock

Smart card processors usually do not have internal clock generators. An externally supplied clock is therefore necessary. This clock also provides the reference for data transmission rates. According to ISO/IEC 7816-3 and most other standards and specifications, the duty factor of the clock must be 50 %. The usual tolerance is a duty factor range of 40 to 60 %.

The clock signal applied to the contact is not necessarily the same as the internal clock provided to the processor. Some microcontrollers have a clock multiplier or divider that may optionally be inserted between the external and internal clocks. The clock divider frequently has a division factor of 2, so the internal clock rate is only half of the external clock rate. This is partly due to the characteristics of the chip hardware and partly because it allow oscillators already present in terminals to be used as the source of the clock signal for the chip.

Most smart card microcontrollers allow the clock signal to be switched off when the CPU is in the sleep mode. In this case, switching off the clock means holding the clock line at a defined level. Depending on the preference of the chip manufacturer, the ‘off’ level may be either high or low.

Since smart cards draw only a few microamperes from the clock line, switching off the clock may at first glance appear somewhat curious. Nevertheless, the amount of power saved within the terminal is substantial, so it can be worthwhile in certain applications.

3.3.5 Data transmission

If an error occurs during data transmission, it may happen that the terminal and the card attempt to send data at the same time. This results in a data collision on the connecting I/O line. Quite apart from the problems this causes at the application level, at the physical level it could produce currents in the I/O line that might be large enough to destroy the interface components. To prevent damage to the semiconductors in such an event, the I/O line in the terminal is tied to the +5-V level via a 20-kΩ pull-up resistor, as shown in Figure 3.37. In combination with the agreed convention of never sending an active 5-V level, this avoids any problems that might occur if the two parties attempted to drive the data line to two different levels as the result of a communications error. Whenever the I/O line has to be set to a +5-V level during communications, the party in question simply switches its output to a high-impedance state (tri-state level), and the line is raised to the +5-V level by the pull-up resistor alone.

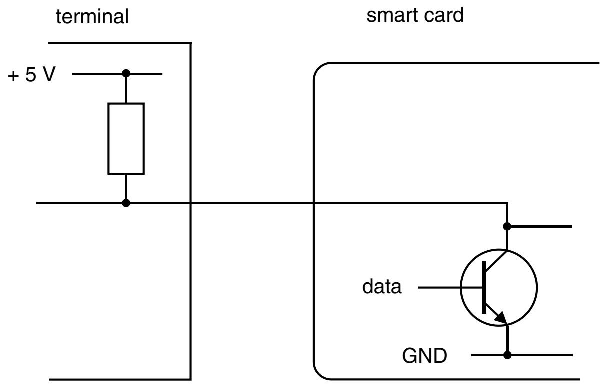


Figure 3.37 The circuit of the I/O channel between the terminal and the smart card

3.3.6 Activation and deactivation sequences

All smart card microcontrollers are protected against electrostatic charges and potentials on the contacts. In order to avoid undefined states, precisely specified activation and deactivation sequences are prescribed, and they must be strictly adhered to. This is also reflected in the relevant part of ISO/IEC 7816-3. These sequences define the electrical aspects of activating and deactivating the card and have nothing to do with the sequence of establishing mechanical contact with the card, which is anyhow not specified. Nevertheless, mechanical contact is first made with the ground contact of the card as an intelligent precaution in order to ensure well-defined electrical connection and disconnection.

As shown in Figure 3.38, the electrical ground connection must be made first, followed by the supply voltage connection. After this comes the clock connection. If an attempt were made to connect the clock before the supply voltage, for example, the microcontroller would try to draw its entire supply current via the clock line. This could irreversibly damage the chip, causing complete functional failure. A faulty deactivation sequence could also have similar effects on the microcontroller.

When the microcontroller is operating, it can be reset via the reset line. This requires a low level to be first applied to this line, with the actual reset being initiated by the subsequent rising edge. Such a reset during operation is called a warm reset, as in other computer systems. By contrast, a cold reset is one that occurs when all the supply lines are switched as specified in the ISO standard.

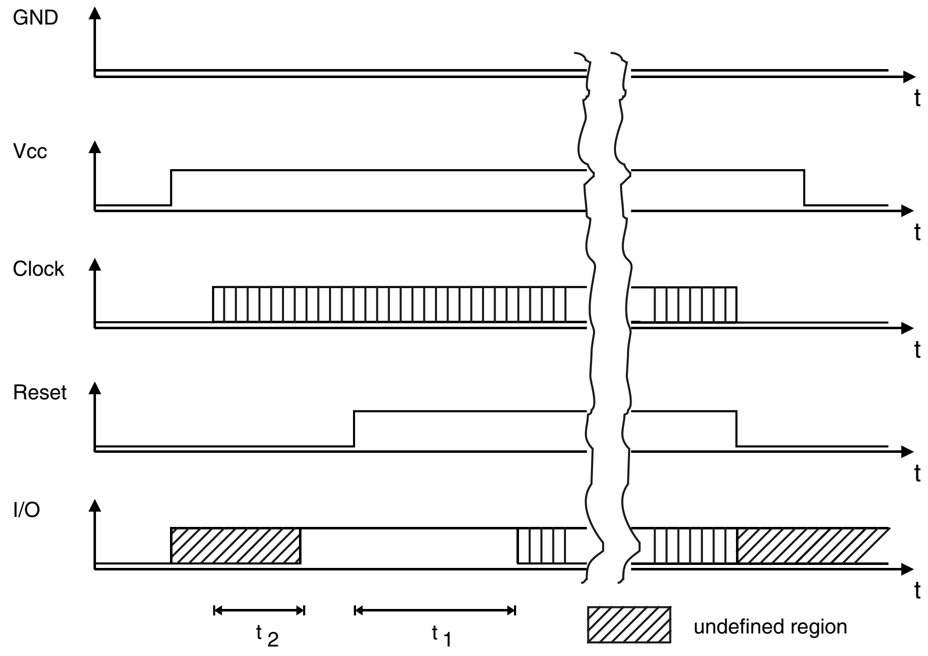


Figure 3.38 Smart card activation and deactivation sequences according to ISO/IEC 7816-3. The intervals t_1 and t_2 lie in the ranges $400/f \leq t_1 \leq 40,000/f$ and $t_2 \leq 200/f$

3.4 SMART CARD MICROCONTROLLERS

From an informatics perspective, the central component of a smart card is the microcontroller embedded under the contacts. It controls, initiates and monitors all of the card's activities. The microcontrollers that have been specially designed and developed for this purpose are complete computers in their own right. This means that they contain processors, memory and interfaces to the outside world.

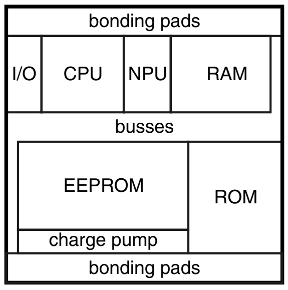


Figure 3.39 Possible arrangement of the essential functional components on the die of a simple smart card microcontroller

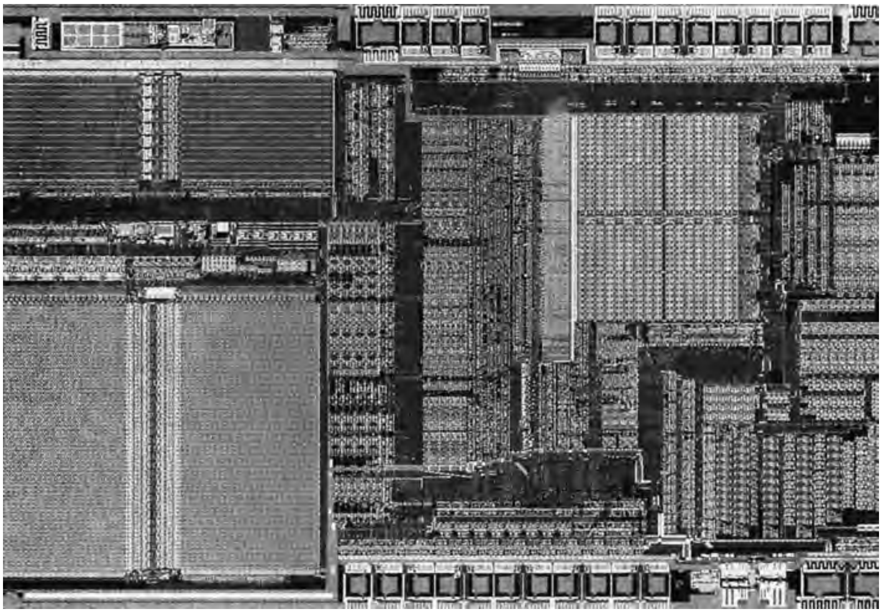


Figure 3.40 Photograph of a PC 83C852 smart card microcontroller with the following functional components (from top left to bottom right): ROM, EEPROM, processor with coprocessor and RAM. This chip has an area of 22.3 mm^2 and contains 183,000 transistors. Although this chip is no longer produced, it clearly shows the arrangement of the functional components on the die (*Source:* Philips)

The most important functional components of a typical smart card microcontroller are the processor, the address and data buses and the three types of memory (RAM, ROM and EEPROM). The chip also has an interface unit that provides serial communication with the outside world. This interface should not be imagined to be a complex functional unit that can independently transmit and receive data. In the simplest case, the serial interface is just a location that can be addressed by the CPU and is connected to the I/O contact.

In addition, some manufacturers provide special processors on the chip that act as a sort of mathematical coprocessor, although the functions provided by these components are limited to exponential and modulus operations on integers. Both of these operations are fundamental and necessary elements of public-key encryption procedures, such as the RSA algorithm.

The semiconductor technologies that are presently commonly used to produce smart card microcontrollers work with structure widths of around 0.25 µm, 0.18 µm and 0.13 µm, which definitely lie in the range of the smallest currently achievable structure widths.

Microcontrollers used in smart cards are not standard, widely available components. Instead, they have been specifically developed for this purpose, and they are not used in other applications. There are several important reasons for this, which are described below.

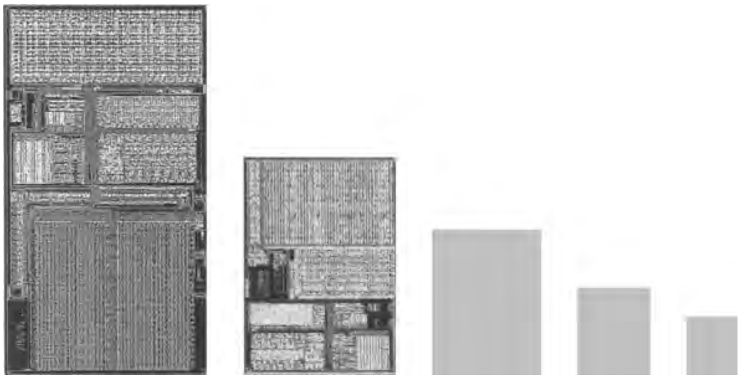


Figure 3.41 Relative sizes of functionally identical smart card microcontrollers before and after chip-area reduction ('shrink processing'). At the far left is an SLE 44C80 in 1-μm technology, with a surface area of 21.7 mm^2. To its right is an SLE 44C80S in 0.8-μm technology, with a surface area of 10 mm^2. The three gray rectangles show the relative sizes this smart card microcontroller would have if fabricated in 0.5-μm, 0.35-μm and 0.13-μm technologies (*Photos:* Infineon Technologies)

Manufacturing costs

The surface area of the microcontroller on the silicon wafer is one of the decisive factors with regard to manufacturing costs. A large chip area leads to more complicated packaging in the module, and thus increased costs. The chip area is thus kept as small as possible.

Furthermore, many commercially available standard devices include functions that are not needed in smart cards. Since these functions take up extra space on the wafer, they can be deleted from chips designed for smart cards. Although only a small reduction in the manufacturing cost per chip is achieved by such efforts to minimize the chip size, these small savings add up to a significant amount when a large number of chips are produced. This justifies the modifications to the chip design.

Functionality

Due to the need to integrate all the functional components of a computer into a single silicon chip, the available number of suitable semiconductor devices is extremely limited. Given the requirements of a minimum chip area, 5-V or 3-V supply voltage and a serial interface on the chip, all standard devices are effectively ruled out. In addition, the chip must contain a memory that can be written and erased but that does not require a permanent power supply for data retention (EEPROM or Flash EEPROM).

Security

Since smart cards are primarily used in security-related areas that require both passive and active security features in the chip, developing chips specially designed for this purpose is an unavoidable necessity.

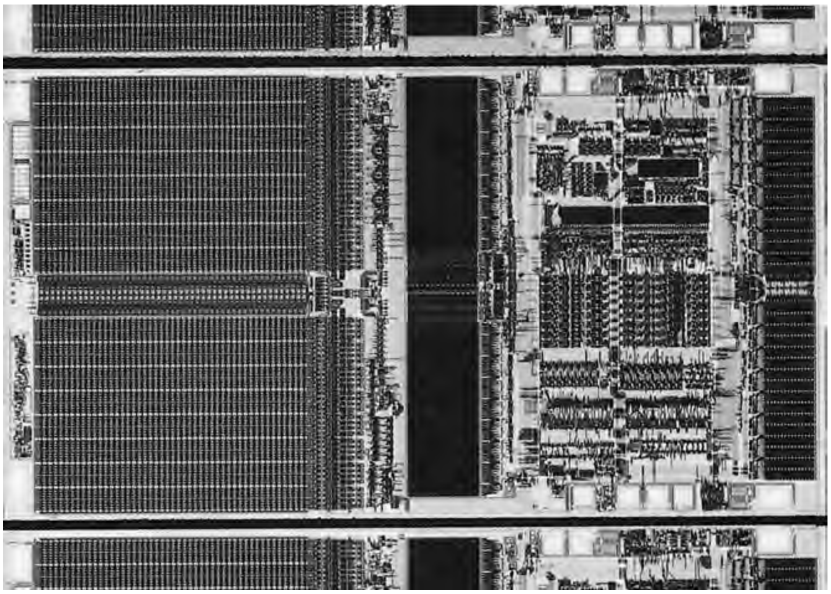


Figure 3.42 An ST16623 smart card microcontroller with the following functional components (from left to right): ROM, EEPROM, CPU and RAM. Although this chip is no longer manufactured, it clearly shows the arrangement of the individual functional components on the die
(*Source:* ST Microelectronics)

Chip area

The size of the microcontroller die strongly affects the fragility of the chip. A larger die is more likely to break when the card is bent or twisted. Consider a telephone card carried in a wallet, for instance: the bending stresses on the card and the embedded chip are enormous. Even the finest hairline crack in the chip is sufficient to render it useless. Therefore, most card manufacturers impose an upper limit of about 25 mm^2 on the chip area and demand that the layout be as nearly possible square, in order to minimize the risk of fracture.

Availability

The security policy of many card manufacturers is that the microcontrollers they use are not available on the open market. This makes it considerably more difficult to analyze the chip's hardware, since a potential attacker normally does not have access it.

However, this position has been seriously weakened by the general availability of programmable smart cards, such as Java Card types, and it is generally no longer defensible for standard applications.

Limiting the types of microcontrollers used to only a few specialized types (and consequently only a few manufacturers) has the disadvantage that the card manufacturer is highly dependent

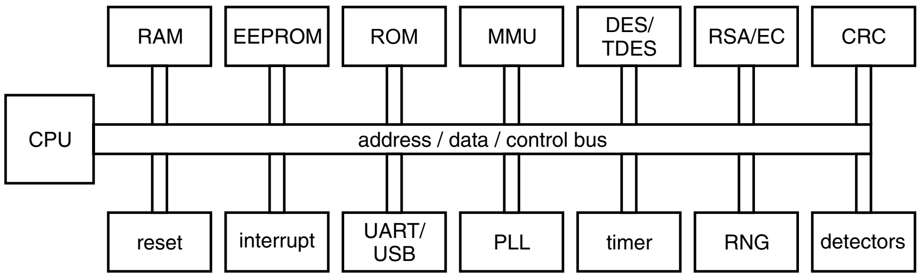


Figure 3.43 The essential functional logic units of a high-performance smart card microcontroller

on the chip suppliers. If a semiconductor manufacturer experiences production problems, it is not possible to quickly switch to a different device.

3.4.1 Processor types

The processors used in smart cards are not special designs, but instead proven devices that have been used in other areas for a long time. In this industry, it is not usual to develop new processors for special application areas, since this is generally too expensive. In addition, it would yield a completely unfamiliar processor, for which no suitable function libraries and development tools would be available from producers of operating systems.

Additionally, smart card processors must be extremely reliable. It is therefore better to rely on older type processors that have been proven in practice, rather than experimenting with the latest developments of semiconductor manufacturers. The aerospace industry, which is very interested in functional security, uses only components that are one or two generations behind the current state of the art, for the same reasons.

The transistor was invented at the Bell labs in 1947. Intel brought out the first microprocessor in 1971, with the type designation ‘4004’. It contained 2300 transistors and had a clock frequency of 108 kHz, 45 machine instructions, 604 bytes of address space and a 4-bit data bus, yielding a processing capacity of 0.06 MIPS. Since then, the development of integrated microprocessor components has made huge strides. This is clearly shown by recent products, such as the 32-bit Pentium IV processor with 42 million transistors on 217 mm^2 in 0.13-μm technology and a 2-GHz clock rate. However, the most technologically advanced processors are not used in smart cards, for the reasons that have just been stated. A total of 200,000 transistors is typical for an IC with a midrange processing capacity.

Smart card microcontrollers at the lower end of the performance scale usually have an addressable memory in the range of 6 KB to 30 KB. Under these conditions, using an 8-bit memory bus does not impose any significant restrictions. The processors used generally have a CISC (complex-instruction-set computer) architecture, which means that they require several clock cycles to execute machine instructions and usually have very large instruction sets. The address range of the 8-bit processors is most often 16 bits, which allows up to 65,536 bytes to be addressed. The processor instruction sets are based on either the Motorola 6805 or Intel 8051 architecture. The semiconductor manufacturer may add supplementary instructions to the standard instruction set. Such instructions usually involve additional options for 16-bit memory addressing, which exists only in the most rudimentary form in the two instruction sets that form the basis for the instruction sets of smart card processors.

Processors in the 8-bit families are also available with extensions that allow them to address additional memory banks, in order to surmount the 64-kB limit. Access to such memory banks is controlled via special registers that map the memory banks into a specific memory region, where they can be accessed by the processor. However, this type of non-linear memory addressing has significant drawbacks. For instance, the relatively complex distribution of program code over several memory banks significantly complicates the software, thus increasing the likelihood of errors. Additional memory is also needed for bank switching functionality. Consequently, memory space expansion using memory banks is primarily a makeshift remedy that is used while awaiting the transition to processors with larger bit widths.

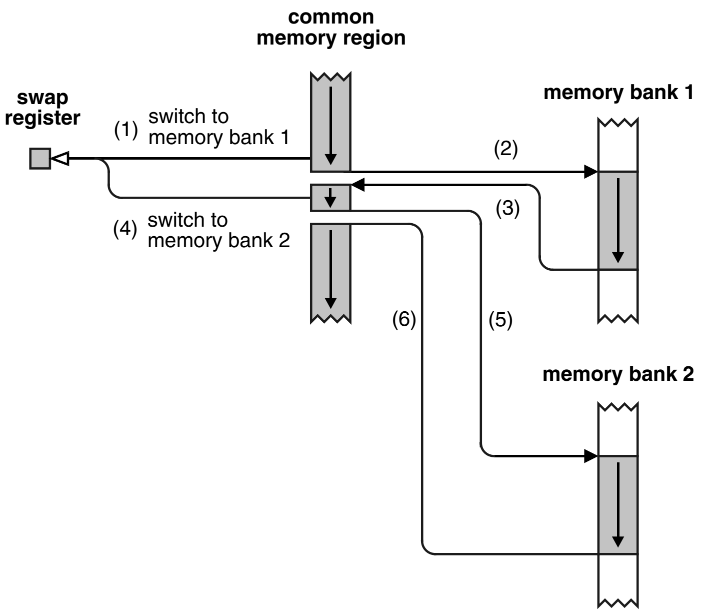


Figure 3.44 Basic arrangement of a memory divided into several banks and the associated program flow. This example shows two subroutines located in two different memory banks being called from a common memory region. The numbers in parentheses indicate the sequence of events in the process

The 16-bit processors include several derivatives of existing 8051 architectures, as well as company-specific designs, such as the Renesas H8, Philips XA and Samsung CALM devices. Incidentally, for a long time the H8 was the only 16-bit processor for smart card microcontrollers with a RISC-like architecture and a corresponding instruction set ('RISC' stands for 'reduced-instruction-set computer').

At the upper end of the performance scale for smart card microcontrollers, both 16-bit and 32-bit types are now available. The development trend is quite clearly heading in the direction of 32-bit processors. Such processors are urgently needed in this performance class, in order to handle large memories (exceeding the 64-kB boundary) and above all to satisfy the enormous processing appetites of modern interpreter-based smart card operating systems, such as Java

45

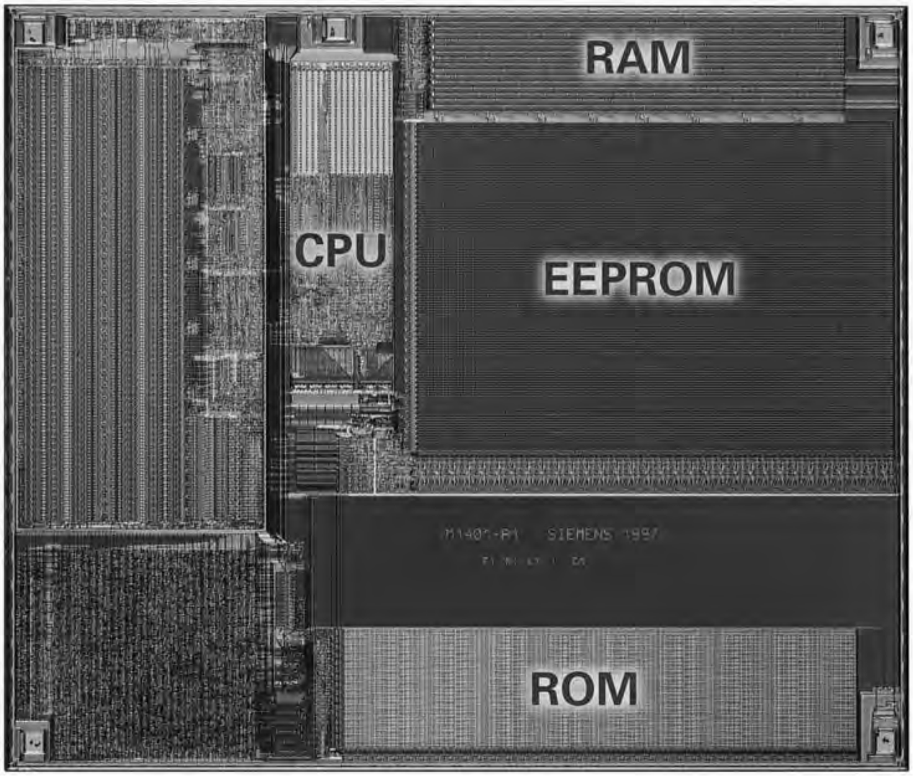


Figure 3.45 An SLE 66CX160S smart card microcontroller with an area of 21 mm^2, fabricated in 0.6-µm technology with 32 kB of ROM, 16 kB of EEPROM and 1280 bytes of RAM. The two unlabeled regions on the left-hand side of the chip are the numeric coprocessor and the peripheral components (timer, random-number generator and CRC coprocessor). The five bonding pads for the electrical connections to the module contacts can be clearly seen in the photo (*Photo*: Infineon Technologies)

Card. The key selection criteria for processors include code density, power dissipation and resistance to attacks.

Among the 32-bit processors, company-specific types are presently becoming established, but two processor cores used in typical microcontroller markets have also gained an entry into the smart card realm. They are the MIPS [MIPS] and ARM [ARM] processors.

The first step into the 32-bit league was taken in 1993 with the European CASCADE ('Chip Architecture for Smart Card and portable intelligent Devices') project. One objective of this project was to provide a high-performance processor for smart cards. The project selected the ARM 7M RISC processor, which is often used in portable equipment such as video cameras and PDAs.[3] It has a 32-bit architecture and can run at up to 20 MHz with a 3-V supply voltage,

[3] Based on [Peyet 97]

while drawing only 40 mA. In 0.8-μm technology, the ARM 7 core has an area of 5.9 mm^2 (3.12 × 1.9 mm), with the associated arithmetic processor for the usual public-key algorithms (RSA, DSA and EC) occupying an additional 2 mm^2. The processor has both supervisor and user modes, and thus supports partitioning of the operating-system code and application code. Since originally being selected, the ARM 7 has been optimized for smart card applications, and in the form of the SC 100 ('secure core') it has been built into smart card microcontrollers by various semiconductor manufacturers. The next evolutionary step is a dedicated smart card variant of the ARM 9, with the type designation SC 200. A similar situation exists with the MIPS processors, which were also originally developed for other application areas.

Although 32-bit processors take up significantly more die space than 8-bit processors using the same technology, due to their wider buses and more complex internal structures, they will be used in increasing numbers in future smart card applications. The processing power that they offer is indispensable for these applications, so the disadvantages of greater power consumption and increased chip area can be accepted as the price of progress. Of course, 8-bit processors will not die out in the foreseeable future, since they provide a solid basis for inexpensive chips at the lower end of the performance scale.

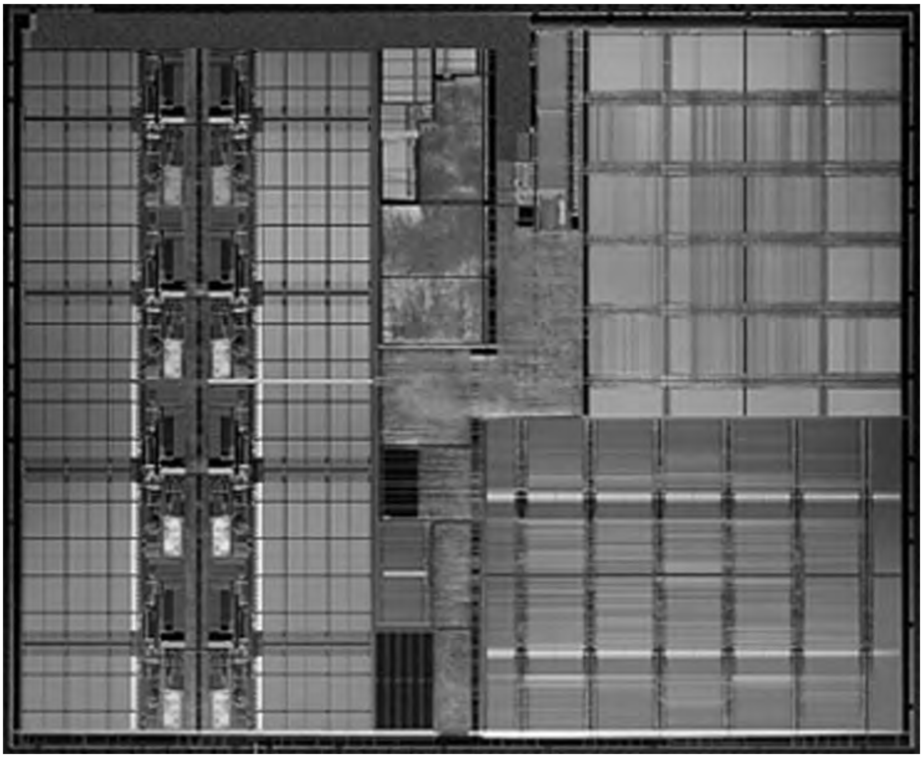


Figure 3.46 Bond-out version of an SLE88CX720P 32-bit smart card microcontroller without final chip shielding. This microcontroller was fabricated in 0.22-μm technology and has 240 kB of ROM, 80 kB of EEPROM and 8 kB of RAM (*Photo:* Infineon Technologies)

3.4.2 Memory types

Besides the processor, the most important components of a microcontroller are various types of memory, which serve to store program code and data. Since smart card microcontrollers must be complete computers, they exhibit a characteristic division of memory into RAM, ROM and EEPROM. The exact division depends very strongly on the chip's ultimate application area. In any case, an effort is always made to keep the RAM and EEPROM as small as possible, since they require the most space per bit.

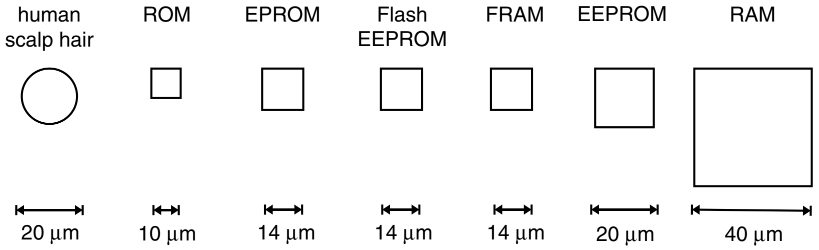


Figure 3.47 Comparison of the die area of a single bit cell for various types of memory. The dimensions shown here are approximate and relate to 0.8-µm technology. For comparison, the diameter of the first planar transistor in 1959 was 764 µm [Buchmann 96, Stix 96], the diameter of the dot at the end of each sentence in this book is 400 µm, the resolution limit of the human eye is 40 µm, the size of a bacterium is 0.4–2 µm and the size of a DNA double helix is 0.1 µm

In the case of multiapplication smart cards, which can manage several applications at the same time, the most commonly used chips have a ROM capacity that is roughly twice as large as that of the EEPROM, in order to provide enough room to store the complex operating system code. For single-application smart cards, microcontrollers are selected whose EEPROM capacity is only slightly larger than the volume of the application data. All variable application data, along with some parts of the operating system, can thus be stored in EEPROM in order to make optimum use of the EEPROM, which takes up a relatively large amount of space on the die and is thus expensive.

Integrating three different types of semiconductor memory into a single silicon die is a technically difficult task requiring a significant number of production steps and exposure masks. The different types of memory also occupy markedly different areas, due to their different structures and operating principles. For example, a RAM cell occupies about four times as much space as an EEPROM cell, which in turn occupies four times as much space as a ROM cell. This is why smart card microcontrollers have so little RAM, with 4 kB of RAM already considered to be large. If you consider that 16 kB of EEPROM or 64 kB of ROM can be put into the same area, you can understand why.

A new type of memory technology for smart cards has become available relatively recently. This is called 'Flash' EEPROM, and it permits write and erase access times that are much shorter than with previously available types of EEPROM. The cell size is approximately half of that of a conventional EEPROM, depending on the particular design.

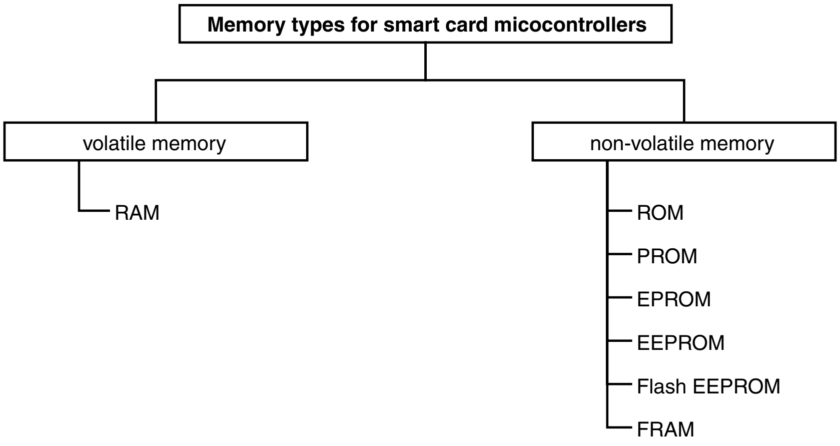


Figure 3.48 Classification chart for smart card microcontroller memories. Generally speaking, PROM and EPROM are no longer used in modern microcontrollers. FRAM is only starting to be used in smart cards

Table 3.4 Memory types used in smart card microcontrollers. For comparison, the area of the dot at the end of each sentence in this book is 125,660 μm^2

Type of memory	Number of possible write/erase cycles	Write time per memory cell	Typical cell size with 0.8-μm technology
Volatile memory			
RAM	unlimited	≈ 70 ns	≅ 1700 μm^2
Non-volatile memory			
EEPROM	100,000–1,000,000	3–10 ms	≈400 μm^2
EPROM	1 (not UV-erasable)	≈50 ms	≈200 μm^2
Flash EEPROM	≈10, 000	≈10 μs	≈200 μm^2
FRAM	≈10^{10}	≈100 ns	≈200 μm^2
PROM	1	≈100 ms	—
ROM	0	—	≈100 μm^2

The following three numeric examples illustrate these size relationships:

- A simple laser printer works at a resolution of 600 dpi (dots per inch), which means that the minimum possible dot size is 42.6 μm. Also, the dot at the end of this sentence has a diameter of 400 μm. If you wanted to print with a resolution equal to a 0.8-μm structure width, which is still used in semiconductor technology, you need a printer with a resolution of 32,000 dpi!
- High-capacity hard disk drives can store up to 11.6 billion bits per square inch. Under the idealized assumption that each bit occupies a square area, this yields an edge length of 0.24 μm for each bit cell. A ROM cell of a smart card microcontroller made using 0.8 μm technology requires 1700 times as much area for a single bit!

- With a CD-ROM, the situation is different. In this case, the storage density is 7.3 MB/cm^2. This corresponds to an edge length of 1.4 μm for single bit cell, assuming square cells. This is around 80 times less than the area occupied by a ROM cell in 0.8μm technology. With DVDs (digital versatile disks), a density of 50.5 MB/cm^2 is possible. A single bit thus occupies the area of a square with an edge length of 0.5 μm, which is 400 times smaller than a single ROM bit cell in 0.8-μm technology.

Table 3.5 Typical surface area distribution for a smart card microcontroller

Type of component or memory	Surface area
CPU and NPU	20 %
ROM	10 %
EEPROM	45 %
RAM	15 %
Miscellaneous	10 %

ROM (read-only memory)

As the name implies, this type of memory can only be read and cannot be written. No supply voltage is needed to retain data, since the data are 'hard-wired' in the memory.

A smart card's ROM contains most of the operating system routines, as well as various test and diagnostic functions. These programs are built into the chip by its manufacturer when it is made. This is done by preparing a ROM mask from the program code and then using this mask to 'burn' the program into the chip using lithographic processes. In this case the data, which are the same for all chips of a production run, can only be entered into the ROM during manufacturing.

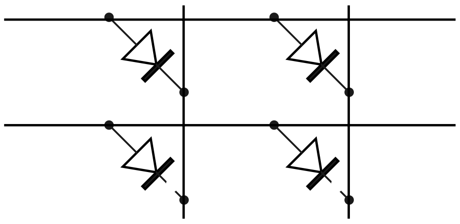


Figure 3.49 Basic functional structure of a ROM

PROM (programmable read-only memory)

PROM is not used in smart card microcontrollers, though it could offer several advantages. In contrast to ROM, PROM need not be programmed during manufacturing, but can be written shortly before the chip is fitted into its module. PROM also does not need any supply voltage to retain data. The main reason for not using PROM is that programming a PROM requires access to the address, data and control buses. This is precisely what should not be possible with smart cards, because it would allow data to not only be written but also read out. Since the memory holds confidential data, using PROM is strictly prohibited.

EPROM (erasable programmable read-only memory)

EPROM was often used in the early years of smart card technology, since at that time it was the only type of memory that could retain data without a supply voltage and could also be written (although only once per bit). However, since an EPROM can only be erased using UV light, it cannot be erased in a smart card. This is why EPROM no longer has any practical significance.

The only meaningful use for EPROM is to irreversibly store a chip number during semiconductor production, but this can now be realized using a special type of non-erasable EEPROM.

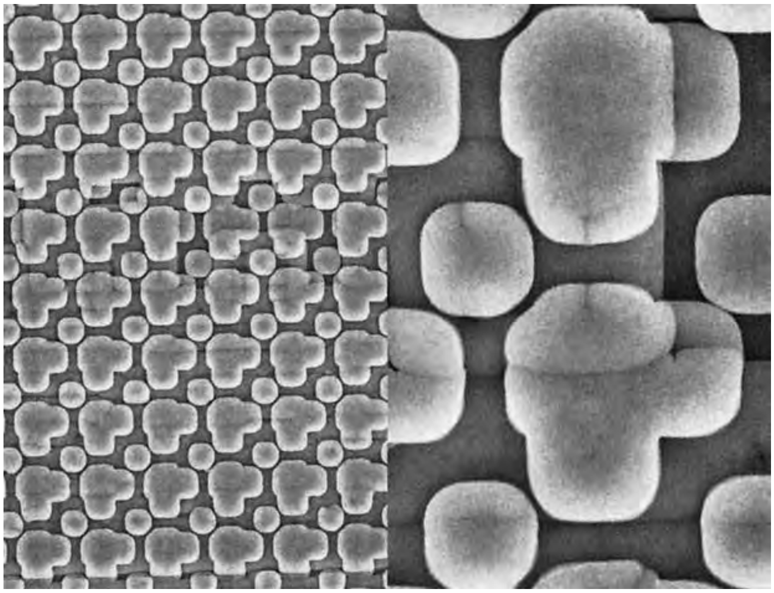


Figure 3.50 Photo of a ROM cell at 1000× enlargement (left) and 11,200× enlargement (right) (*Source:* Giesecke & Devrient)

EEPROM (electrically erasable programmable read-only memory)

EEPROM, which is technically more complex than ROM or RAM, is used in smart cards for all data and programs that need to be modified or erased at some time. Functionally, an EEPROM corresponds to the hard disk of a PC, since it retains data in the absence of power and the data can be altered as necessary. EEPROM is thus non-volatile memory.

In principle, an EEPROM cell is a tiny capacitor that can be charged or discharged. The charge state can be interrogated by sensing logic. A charged capacitor represents a logic 1, while a discharged capacitor represents a logic 0. In order to store one data byte, eight of these small capacitors are needed, along with suitable sensing circuitry.

The erased state of the EEPROM cell is the critical factor with regard to writing to the cell. In most types of EEPROM, the erased state is ˚1˚. An EEPROM has the property that an individual cell can only be programmed from its erased state to its unerased state, which in this example is ˚0˚. If an EEPROM cell is already in the ˚0˚ state, an entire EEPROM page must be erased in order to restore that bit to the ˚1˚ state. The algorithm that is usually used for an EEPROM write routine is described in Listing 3.1.

Listing 3.1 Pseudocode of a routine for writing complete EEPROM pages. If multiple pages or only part of a page is to be written, this routine should be nested in a higher-level routine. A similar procedure should be used if a write retry routine must be called in the event of an error. Here the erased state of the EEPROM is 'FF', and the written state is '00'

UpdateEEPROM: // *NewData*: data to be written // *StoredData*: stored data	Entry point for writing data to an EEPROM page

IF (*NewData* = *StoredData*) THEN
   (GOTO UpdateEEPROM_Exit)
	If the data already stored in the EEPROM page are the same as the new data, exit the function.

WorkData := *NewData* XOR *StoredData*	Following the XOR operation, the differences between the stored data and the new data can be seen as set bits in the variable *WorkData*.
WorkData := *WorkData* AND *NewData*	The AND operation causes the variable *WorkData* to be non-zero if the EEPROM page must be erased before the write process.

IF (*WorkData* <> 0) THEN
   (Erase EEPROM_Page
   IF (*StoredData* <> 'FF') THEN
      (GOTO UpdateEEPROM_Errror_Exit))
	If the variable *WorkData* is non-zero, the EEPROM page must be erased before the write operation. After this operation, a test is made to see if the EEPROM page was successfully erased.

Write EEPROM_Page with *NewData*
   IF (*StoredData* <> *NewData*) THEN
      (GOTO UpdateEEPROM_Errror_Exit)
	The EEPROM page can now be written. Afterwards, a check is made to see whether the data were successfully written to the EEPROM.

Update EEPROM_Exit: RETURN	The function has completed successfully.
Update EEPROM_Error_Exit: RETURN	An error occurred during execution of the function.

Figure 3.52 shows the cross-section of an EEPROM cell. The actual structure is somewhat more complicated, but this simplified diagram is a very useful aid to comprehension.

In order to understand how an EEPROM cell works, you need to understand its semiconductor background. In its simplest form, an EEPROM cell is essentially a modified field-effect transistor (MOSFET) built on top of a silicon substrate. A MOSFET is formed by first creating a source and a drain in the substrate and then placing a control gate between them. The current

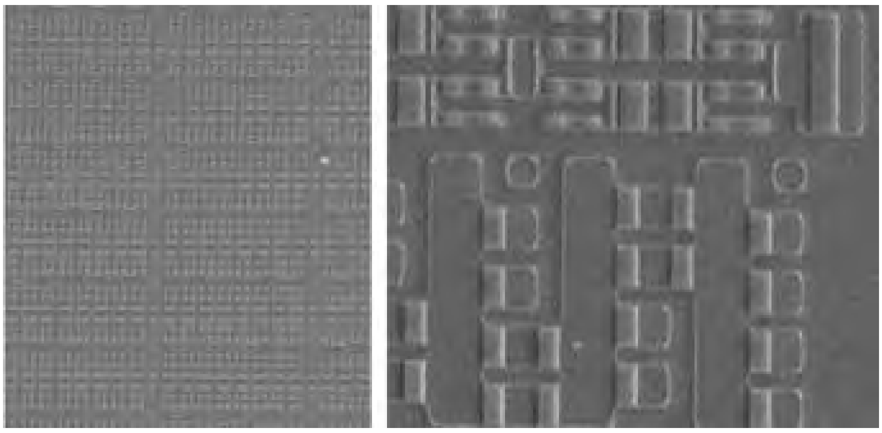


Figure 3.51 Photo of an EEPROM cell at 1000× enlargement (left) and 4000× enlargement (right) (*Source:* Giesecke & Devrient)

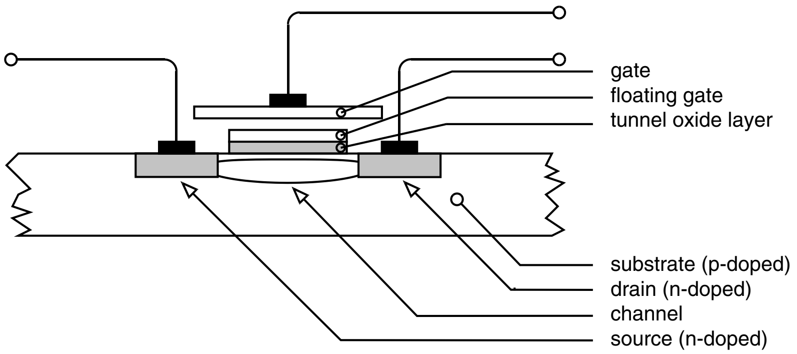


Figure 3.52 Cross-section of the semiconductor structure of an EEPROM cell

flowing from the source to the drain can be controlled by applying a potential to this gate. As long as no potential is present on the gate, no current can flow, since there are two diode junctions (n–p and p–n) between the source and the drain. If a positive potential is applied to the gate, electrons are drawn towards it from the substrate, forming an electrically conducting channel between the source and the drain. The FET is then conductive, and a current can flow.

In an EEPROM cell, an additional 'floating' gate is located between the control gate and the substrate. It is not connected to any external voltage source, and the separation between it and the substrate is very small, on the order of 10 nm. The floating gate can be charged or discharged via the substrate using the tunnel effect (Fowler–Nordheim effect), which allows charge carriers to penetrate thin oxide layers that act as insulators. This requires a sufficiently large potential difference across this oxide layer, which is called the tunnel-oxide layer. Current flow from the source to the drain is controlled by the charge on the floating gate. This means that the state of this gate can be interpreted as a logic 0 or a logic 1 according to whether a current can flow through the gate.

To charge the floating gate, a high positive voltage is applied to the control gate. This creates a large potential difference between the substrate and the floating gate, which in turn causes electrons to tunnel through the oxide layer to the floating gate, with a current that can be measured in picoamperes. The negative charge on the floating gate produces a high threshold voltage between the source and the drain, which means that the field-effect transistor is blocked. No current can flow between the source and the drain. Storing electrons in the floating gate is thus equivalent to storing data.

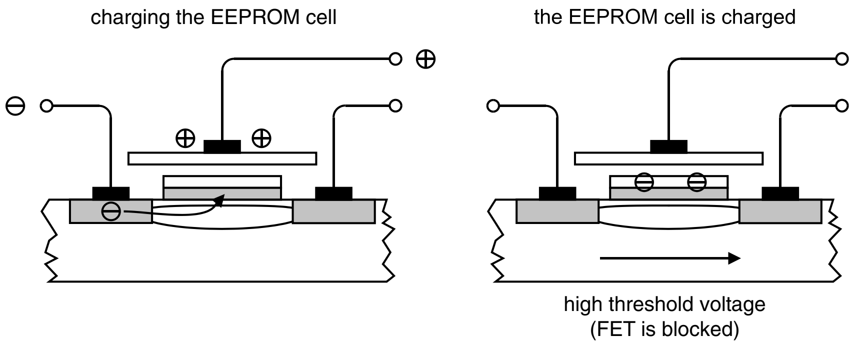


Figure 3.53 Charging an EEPROM cell

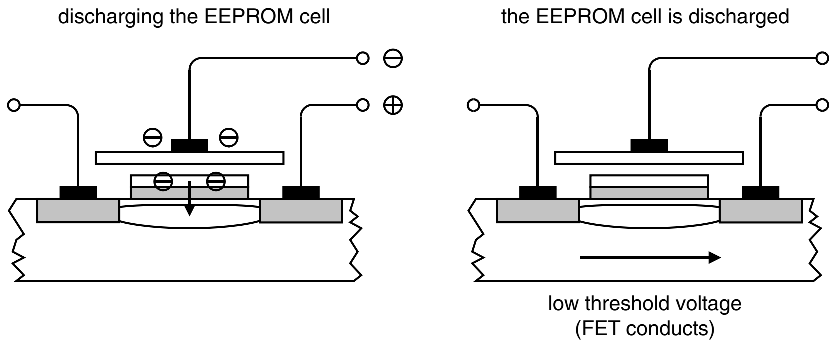


Figure 3.54 Discharging an EEPROM cell

The potential needed to charge the EEPROM cell is about 17 V at the control gate, which is reduced to about 12 V at the floating gate by capacitive coupling. However, since smart card microcontrollers work with a supply voltage of only 1.8–5 V, a charge pump is needed to produce the necessary voltage. In principle, the charge pump is a cascaded voltage-multiplier circuit. It generates an output voltage of about 25 V from the low input voltage, which yields a voltage close to the necessary level of 17 V after stabilization. Depending on the structure of the cell, charging an EEPROM cell requires from 2 to 10 ms per memory page (1–32 bytes).

To erase an EEPROM cell, a negative voltage is applied to the control gate. This causes the electrons to leave the floating gate and return to the substrate. The EEPROM cell is then

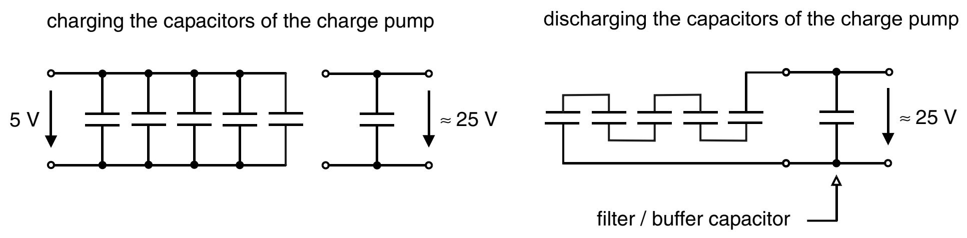

charging the capacitors of the charge pump
5 V
≈ 25 V
discharging the capacitors of the charge pump
≈ 25 V
filter / buffer capacitor

Figure 3.55 This schematic diagram shows the operating principle of a charge pump circuit during charging (left) and discharging (right). These processes are repeated at a high frequency, causing the charge pump to produces a slightly pulsating DC voltage at its output.

discharged and the threshold voltage between the source and the drain is low, so the FET conducts.

The floating gate can also be discharged by heat or energetic radiation (such as X-rays or UV light), which causes it to return to its 'secure' state. This state is of fundamental significance in the design of smart card operating systems, since security barriers can be breached by deliberately altering ambient conditions if the secure state of the EEPROM is not used to store critical data. Depending on the technical implementation of an EEPROM cell, the secure state can correspond to a logic 0 or a logic 1. This is specific to each type of smart card microcontroller, and it should be confirmed with the manufacturer if necessary.

EEPROM is one of the few types of semiconductor memory having a limited number of access cycles. It can be read any number of times, but it can be programmed only a limited number of times. The reason for this limitation can be found in its semiconductor structure. The life expectancy of an EEPROM depends strongly on the nature, thickness and quality of the tunnel-oxide layer between the floating gate and the substrate. Since this layer must be produced very early in the fabrication process, it is exposed to strong thermal stresses in subsequent fabrication steps. This may cause damage to the oxide layer, which in turn affects the useful life of the EEPROM cell. During fabrication, and every time the cell is written, the tunnel-oxide layer absorbs electrons that are not subsequently released. These 'trapped' electrons are located close to the channel between the source and the drain, and once they reach a certain number they have a stronger effect on the threshold potential than the charge stored in the floating gate. When this happens, the EEPROM cell has reached the end of its useful life. Although it can still be written, the charge on the floating gate has only a minimal effect on the characteristics of the channel between the source and the drain, so the threshold potential always remains the same. The number of possible write/erase cycles varies greatly, depending on structural details. Typical values range from 100,000 to 1,000,000 cycles over the entire range of operating temperature and voltage. At room temperature and using an optimum supply voltage, values that are 10 to 50 times greater can be achieved.

When an EEPROM cell is approaching the end of its life, its data retention time decreases. The retention time can range from hours to minutes or even seconds. The more exhausted the EEPROM becomes, i.e., the more electrons that have been absorbed by the tunnel oxide layer, the shorter is the retention time.

A charged floating gate loses charge over time, due to insulation losses and quantum-mechanical effects. The time required for this to become noticeable can range from 10 to

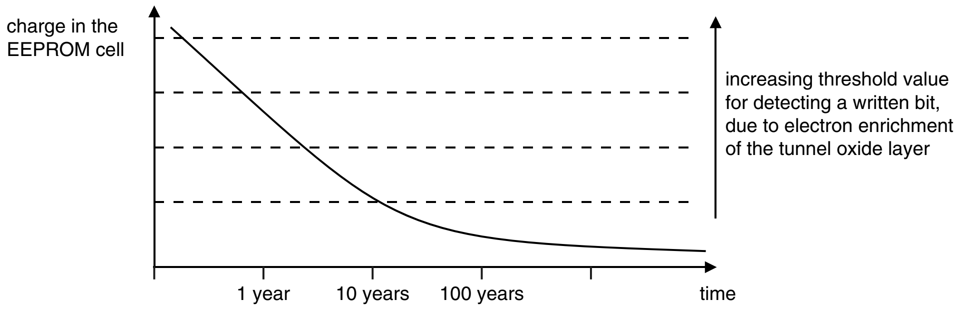


Figure 3.56 Displacement of the discharge curve of an EEPROM cell as a function of the number of executed program/erase cycles

100 years. In this regard, it is interesting to note that a charged floating gate holds 100,000 to 1,000,000 electrons, depending on the implementation. Currently, all semiconductor manufacturers guarantee data retention for 10 years. In order to increase this value, the contents of EEPROM cells can be periodically refreshed by reprogramming. However, this is only worthwhile when the data must be stored for a long time.

Flash EEPROM (Flash electrically erasable programmable read-only memory)

Flash EEPROM, which is often simply called 'Flash memory', shares the property of nonvolatility with regular EEPROM. This means that it retains data in the absence of a supply voltage. It is very similar to EEPROM in its construction and operation. The basic difference between a Flash EEPROM and a normal EEPROM is in the writing process, which is based on hot-electron injection instead of the Fowler–Nordheim (tunneling) effect. 'Hot' electrons are fast electrons produced by a high potential difference between the source and the drain. Some of these electrons penetrate the tunnel-oxide layer, due to the influence of a positively charged control gate, and are stored in the floating gate. This reduces the writing time to around 10 μs, which is a considerable improvement on the value of 2–10 ms for a regular EEPROM. The name 'Flash' comes from this extremely short programming time. Another advantage is that the programming voltage is only 12 V, compared with 17 V for EEPROMs.

There are several smart card microcontrollers with Flash EEPROM, which is primarily used in smart card microcontrollers as a replacement for mask-programmed ROM. Using a microcontroller with Flash EEPROM can reduce the development time of a smart card project by several months, since this eliminates the need to generate ROM masks.

Unfortunately, it is extremely difficult to make semiconductor devices having EEPROM and Flash EEPROM on the same chip. Consequently, in practice a microcontroller with Flash EEPROM usually does not contain any regular EEPROM. Instead, the EEPROM is replaced by a Flash EEPROM of around 8 kB, which has the smallest possible page size in order to minimize the impact on the smart card operating system. The page size of the Flash memory used to replace the ROM is generally significantly larger (e.g., 64–128 bytes), since the routines stored in this memory are written only rarely. When the chip is fabricated, a boot loader is

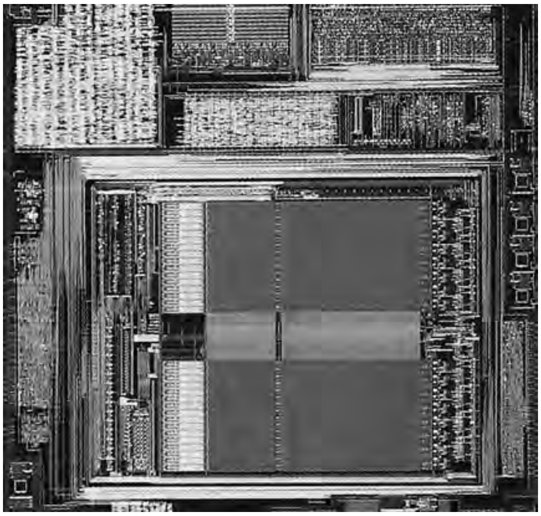


Figure 3.57 An AT89SC168 smart card microcontroller with Flash EEPROM. The functional components at the top are (from left to right) the logic unit, RAM and CPU. The EEPROM charge pump and Flash EEPROM can be seen at the bottom (from left to right) (*Source:* Atmel)

stored in a small ROM to allow the smart card manufacturer to load program code and data into the Flash EEPROM.

Current Flash EEPROM cells have a guaranteed data retention period of at least 10 years, at least 100,000 write/erase cycles and typical page sizes of 8–128 bytes.

There are a few isolated smart card microcontrollers that have unusually large memories, frequently on the order of 1–2 MB. They are always fabricated using Flash memories with page sizes of up to 64 kB. This yields significant area savings with regard to the address and control lines, so memories of this size can be realized in chips having the maximum possible area of 25 mm^2.

FRAM (ferroelectric random-access memory)

FRAM is a new development in semiconductor technology. Despite its name, FRAM is not volatile like RAM, but instead retains its content without a supply voltage. This type of memory exploits the properties of ferroelectric materials in order to store data. Its cell structure is similar to that of EEPROM, but with a ferroelectric material located between the control gate and the floating gate.

FRAM is potentially ideal for smart card memory, since it has very desirable properties as a data storage medium. Only 5 V is needed for programming, the programming time is around 100 ns and the maximum number of programming cycles is around one trillion. The integration density is similar to that of Flash EEPROM. However, FRAM has two disadvantages. The first is a limited number of read cycles, which makes a type of refresh cycle necessary. The second, which is more significant, is that producing FRAMs involves processing steps that

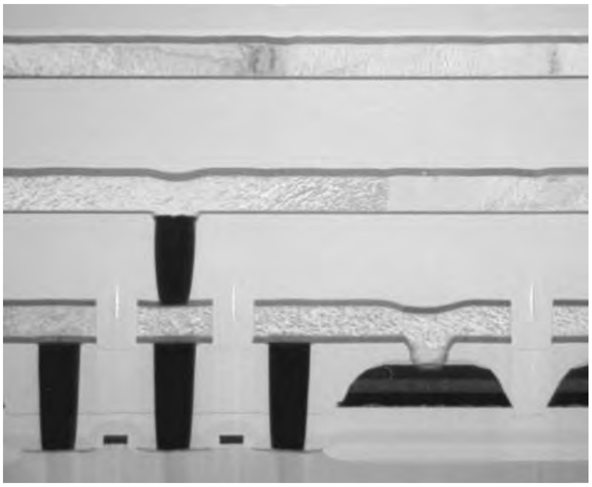


Figure 3.58 Cross-section of a FRAM cell in 0.35-µm technology. The light horizontal bands are aluminum metallization layers, and the dark vertical bars are interconnections ('vias') between the layers. The trapezopidal horizontal area at the lower right is the actual FRAM cell. The width of the cell is approximately 1.5 µm (*Source:* Fujitsu)

are difficult to master. Up to now, little effort has been made to use this technology in smart card microcontrollers. However, this could change in a few years, since FRAM technology possesses all the features needed to allow it to completely supplant EEPROMs, which are presently used almost exclusively.

RAM (random-access memory)

In smart cards, RAM is the memory used to hold data that are stored or altered during a session. The number of accesses is unlimited. RAM needs a power supply in order to operate. If power is switched off or fails temporarily, the content of the RAM is undefined.

A RAM cell consists of several transistors, connected such that they work as a bistable multivibrator. The state of this multivibrator represents the stored value of one bit in the RAM. The RAM used in smart cards is static (SRAM), which means that its contents do not have to be periodically refreshed. It is thus not dependent on an external clock, in contrast to dynamic RAM (DRAM). It is important for the RAM to be static, since it must be possible to stop the clock signal to a smart card. With dynamic RAM, this would cause the stored information to be lost.

3.4.3 Supplementary hardware

There are some requirements specific to smart cards that cannot be fully satisfied using software and thus must be satisfied by supplementary hardware, since they cannot be satisfied using the hardware of conventional microcontrollers. Consequently, the various manufacturers of smart card microcontrollers offer a wide range of supplementary functions in the form of on-chip hardware.

The most commonly used components for supplementary functions are described below. These components do not necessarily have to all be present in any particular microcontroller. The components that are present depend strongly on the target application, among other things. For example, it would be economically unreasonable to integrate an RSA coprocessor into a microcontroller whose target application uses only symmetric cryptographic algorithms. Nevertheless, there are a few commercially available microcontrollers that include nearly all of the components described below.

Another aspect of supplementary functionality with regard to smart card microcontrollers relates to the general subject of security. Chapter 8, ‘Security Techniques’, contains extensive descriptions of supplementary functions implemented in hardware that are primarily intended to counter possible attacks. Consequently, here we describe only those components whose primary purpose is not enhancing security against attacks.

Hardware-based data transmission (UARTs)

The only communications between a smart card and the outside world take place via a bidirectional serial interface. Originally, data transmission and reception via this interface were controlled exclusively by operating system software, without any hardware support. This requires very complex software, and it creates additional potential sources of software errors. However, the main problem is that the speed of software-based data transmission is limited, since the speed of the processor is limited. With current processors, the upper limit is represented by a divider value (clock rate conversion factor) of around 30, which yields a data transmission rate of approximately 115 kbit/s with a 3.5-MHz clock.

If higher communication speeds are desired or required, it is necessary to use either internal clock multiplication or a UART (universal asynchronous receiver–transmitter) component. As the name suggests, such a component is a general-purpose component for transmitting and receiving data independent of the processor. It is not limited by the speed of the processor, nor does it need software for communicating at the byte level. Of course, the upper layers of the data transmission protocol still must be present in the smart card in the form of software, but the lowest layer is implemented in hardware in the UART.

Current UARTs can generally work with divider values smaller than 372, in line with ISO/IEC 7816-3, and some of them can transmit and receive data with divider values as small as 1. There is a wide range of implementations of this function. Some UARTs can transmit and receive only single bytes, and only support byte retransmission according to the $T = 1$ protocol in the event of a transmission error. With such UARTs, all the processor has to do is to supply the necessary data to the UART on time and read it from the UART on time. Reception of a complete byte can be signaled to the processor by a flag or interrupt. All more advanced UARTs can transmit or receive multiple bytes in succession. The highest level of technical capability is presently provided by UARTs that can directly transmit data from the RAM or store received data in the RAM using direct memory access (DMA), without the intervention of the processor and in parallel with the other activities of the processor.

It has been technically feasible to implement UARTs in smart card microcontrollers since the origin of smart cards, but until the end of the 1990s, transmission and reception routines implemented in ROM software required less physical area on the silicon than a functionally equivalent UART component. Since the total surface area is a decisive cost factor for smart card microcontrollers, for a long time nearly all semiconductor manufacturers rejected the hardware approach. However, conditions have changed with increasing integration density. UARTs with a wide range of capabilities are now standard components of all smart card microcontrollers.

Many new types of microcontrollers also allow a USB interface to be placed on the chip as an optional component, in addition to a UART. With such an interface, it would be possible to exchange data with a terminal using the USB protocol with hardware support. Unfortunately, up to now it is effectively not possible to use this USB hardware extension, since USB on smart cards is not yet covered by any standard, which means that it is impossible to guarantee the mutual compatibility of smart cards and terminals.

Timers and watchdogs

Timers in smart card microcontrollers are connected to the internal processor clock or UART clock (which counts etu's) via a configurable divider. They usually have a counting range of 16 or (more rarely) 32 bits. Using a timer, the number of clock pulses from the Start command to the End command can be measured without involving the processor. Most timers can also be used in the reloadable mode, in which they count down from a predefined value and trigger an interrupt when the count reaches zero, after which the counter is automatically reset to the initial value and continues counting.

A watchdog is also often present in the microcontroller. In principle, a watchdog is a timer that must be regularly reset by an explicit processor instruction, in order to prevent it from timing out after a present interval and triggering a reset. A watchdog allows the processor to be reset to a defined state after a definable maximum interval if it becomes trapped in an endless loop. The primary typical application for watchdogs is in the autonomous controller environment, where they are highly useful. However, they are not of much use for smart cards, partly because the software is (hopefully) extremely reliable and partly because the terminal can always interrupt the processor if it lands in an endless loop. Consequently, watchdogs are generally not used in smart card microcontrollers.

Internal clock multiplication and generation

The demands on the processing power of smart cards are constantly increasing. This applies to the processor as well as components that support cryptographic algorithms. One way to meet these demands is to simply increase the frequency of the clock applied to the microcontroller, since processing power is proportional to the clock rate. Doubling the clock rate thus doubles the performance of the processor. However, for reasons of compatibility with applicable standards, it is generally not possible to increase clock rate above 5 MHz.

To get around this restriction, the internal clock frequency of the microcontroller can be increased by using a clock multiplier. This is technically realized using a phase-locked loop (PLL) circuit, which is a well-proven technique, or an RC oscillator. For instance, a smart card connected to an external 5-MHz clock can be operated internally at 20 MHz, which provides significant benefits with regard to computation times for complex cryptographic algorithms or running a Java virtual machine.

Nevertheless, when clock multiplication or an internal clock generator is used, it must be remembered that a higher clock rate causes a proportional increase in current consumption. As a rule, the relationship between clock frequency and current consumption is linear, which means that quadrupling the clock frequency (for example) also quadruples the current consumption. Particularly with battery-operated terminals, increased current consumption is not desirable.

An elegant solution to this problem is provided by 'intelligent power management' in the microcontroller, which involves communicating the maximum allowable current consumption to the control logic of the PLL. This logic then adjusts the PLL to operate in a frequency range that avoids exceeding the prescribed maximum current consumption, without any involvement by the processor. For instance, if the power-hungry NPU is switched into the processing loop, the internal clock frequency will be automatically reduced to prevent the current consumption from rising above the permissible value. Unfortunately, there is a small difficulty with this solution, which is that the specifications for smart cards for GSM and UMTS telecommunications (presently) prohibit the use of free-running oscillators in smart card microcontrollers. This prohibition is based on fear of possible interference with the other circuitry in the mobile telephone. As long as these portions of the specifications continue to exist, it is not possible to use either a continuously adjustable internal clock frequency or an oscillator that is completely independent of the applied clock signal. However, these specifications do allow clock rate multipliers to be used, as long as the internal and external clock rates have a fixed relationship governed by predefined multiplication factors.

Processor speed is not the only bottleneck in smart cards. Data transmission rates, which are specified in the standards, and EEPROM write and erase times do not benefit from increased clock rates. This somewhat limits the advantage of increasing the clock rate. Nevertheless, it can be highly beneficial to use a smart card with an elevated internal clock rate for certain applications, particularly considering that the amount of additional circuitry (and thus area) required on the chip is small. For this reason, nearly all new types of smart card microcontrollers have internal clock multiplication capability.

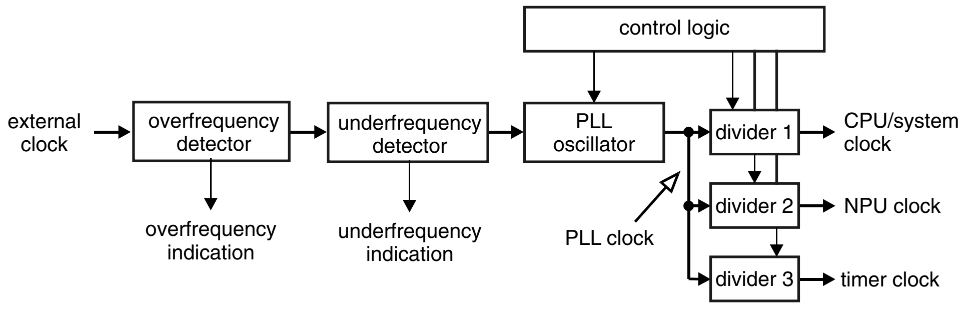


Figure 3.59 Block diagram of a possible internal clock multiplication circuit using a PLL oscillator followed by a divider to supply clock signals to the various microcontroller components. The clock for the NPU is often taken directly from the PLL without any division. If there are several timers in the microcontroller, each one usually has its own individually configurable predivider

DMA (direct memory access)

DMA components have been used for a long time in the PC realm. DMA makes it possible to copy or exchange data between two or more memory regions at high speed, independent of the processor and in part in parallel with the other activities of the processor. It is often also possible to independently fill a certain memory region with a predefined value. The main effect of a DMA unit is to offload the processor and thus allow certain routines to be fashioned more simply. Up to now, high-performance DMA components have been sporadically available in smart card microcontrollers.

Hardware-based memory management, firewalls and memory management units (MMUs)

The latest smart card operating systems allow executable machine code to be downloaded directly to the card.[4] This code, which can then be run using a special command, can be used for purposes such as executing a cryptographic function only known to the card issuer. However, it is in principle not possible to prevent such downloaded code from including a function for reading out secret data from the memory. Operating system manufacturers have been very careful to maintain the confidentiality of their system architectures and program code. The same is also true of secret keys and algorithms in various applications in the card. The public availability of such confidential information would have fatal consequences for an application provider. One administrative solution is to have every new program tested by an independent organization. However, even this cannot guarantee complete security, since a program that is not the same as the one that was tested could later be substituted for the certified program, or the program might be so secret that nobody other than the application provider is allowed to know about it.

One acceptable solution to this impasse is to equip the smart card microcontroller with a memory management unit (MMU). Such a unit monitors the memory boundaries of the current application program while it is running. The permitted memory region is defined by an operating system routine before the application is called, and it cannot be altered by the application program while it is running. This ensures that the application is completely encapsulated and cannot access memory areas forbidden to it. The barriers formed in this manner are called ‘firewalls’, in analogy to walls used for fire protection in buildings. If an application attempts to access another memory region from within a region demarcated by firewalls, it will fundamentally be prevented from doing so, and in addition any such attempt usually triggers an interrupt so the violation can be immediately detected.

Presently, very few smart card microcontrollers have MMUs, although they have been used for years in many other areas. Nonetheless, the importance of this supplementary hardware will greatly increase in the future, since it is the only practical way to securely isolate several applications sharing a single smart card.

Another aspect of MMUs is their ability to relocate physically addressable memory regions to any desired location within the logical memory space of the processor. To a certain extent,

[4] See also Section 5.10, ‘Smart Card Operating Systems with Downloadable Program Code’

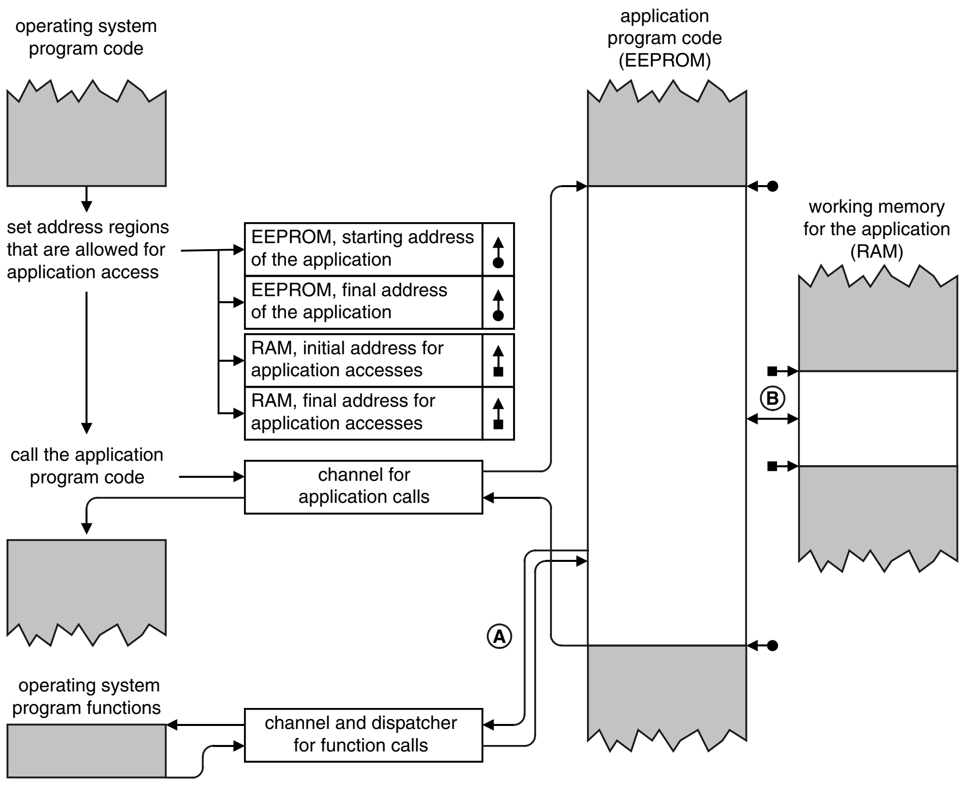

operating system program code
application program code (EEPROM)
set address regions that are allowed for application access
EEPROM, starting address of the application
EEPROM, final address of the application
RAM, initial address for application accesses
RAM, final address for application accesses
working memory for the application (RAM)
B
call the application program code
channel for application calls
A
operating system program functions
channel and dispatcher for function calls

Figure 3.60 Schematic representation of the operating principles of a hardware-based memory management unit (MMU) in a smart card microcontroller. Process 'A' shows a call to an operating-system function that is channeled via the MMU and controlled by a task dispatcher. 'B' is an example of a write–read access to an application memory area demarcated by the MMU

this considerably simplifies the memory management function of the smart card operating system, as well as making it possible to enforce strict isolation of applications with regard to memory space. Furthermore, if downloadable native code is used, the MMU can be used to relocate it to a suitable memory area, thus eliminating the need to use the operating system to manually relocate the executable code.

There is a critical factor that must be considered when using an MMU in a smart card operating system. With the current state of the technology, all MMUs used in various types of microcontrollers are specifically designed for the microcontroller in question. Although this allows the operation and space requirements of the MMU to be optimized, it comes at the price of portability of the object code. In practice, the particular type of MMU that is used has significantly greater consequences for the operating system than the type of processor that is used. Consequently, MMUs are used only very reluctantly in combination with smart card operating systems for large-scale applications, which must of necessity support several different hardware platforms.

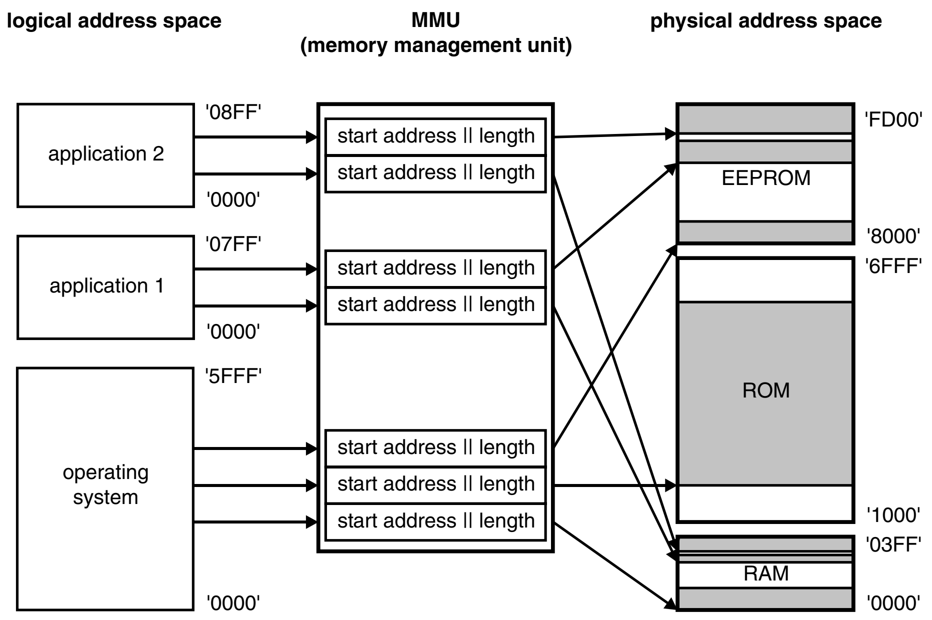


Figure 3.61 Schematic representation of the operating principles of hardware-based memory management (MMU) in a smart card microcontroller with regard to the arrangement of logical and physical address spaces. This example shows an operating system and two applications that share the physically available memory via the MMU. For each of these software components, the MMU translates its physical address space into to a logical address space starting at '0000'

CRC (cyclic redundancy check) calculation unit

CRC codes are still frequently used to secure data or programs by means of an error detection code. Calculating a CRC in software is relatively slow, due to the large number of bit manipulations required, and the calculation can be readily implemented in hardware on the silicon of the microcontroller. For this reason, there are microcontrollers for smart cards that have hardware-based CRC calculation units. Naturally, with such units it must be possible to select the usual generator polynomials and seed values.

Random number generator (RNG)

Random numbers are frequently needed in smart cards for generating keys and authenticating smart cards and terminals. For reasons of security, they should be genuine random numbers rather than pseudo-random numbers, as are commonly produced by typical software-based random number generators. All new smart card microcontrollers have hardware random number generators that produce true random numbers.

However, the quality of the numbers produced by such generators must be immune to being adversely affected by external influences, such as temperature or supply voltage. The hardware

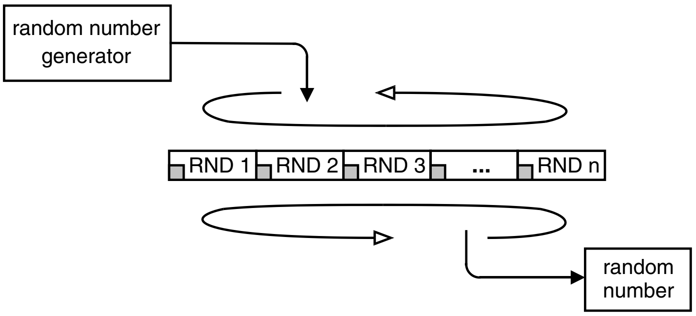


Figure 3.62 Example of a random number generator whose outputs are constantly written to a ring buffer, from which they can be requested as necessary. The gray rectangles mark random numbers that have already been read once and cannot be requested again. This sort of buffer arrangement is used in some types of low-speed random number generators

may use such external influences to assist in generating random numbers, but it must not be possible to predict the generated random numbers by purposefully manipulating one or more of these parameters.

This is very difficult to implement in silicon, so a different approach is taken. The random number generator takes various logic states of the processor, such as the clock signal and the contents of the memory, and applies them to a linear feedback shift register (LFSR) clocked by a signal that is also generated using several different parameters. In some cases, this clock can have a frequency several times that of the processor. If the CPU reads the content of this random number generator, it obtains a relatively good random number that cannot be ascertained from outside in a deterministic manner. The quality of the random number so obtained can be improved by supplementary procedures and algorithms. However, what is important here is that the hardware-based random number generator must basically provide good random numbers that can withstand the usual tests[5] (e.g., FIPS 140–2).

Java accelerator

Within only two years, Java Card has established itself as an industry standard for executable program code in smart cards. However, since the Java VM must interpret the bytecode rather than directly execute it, there is an unavoidable loss of execution speed compared with native machine instructions, which can be directly executed by the processor. However, the widespread use of Java in smart cards makes it attractive for semiconductor manufacturers to devise remedies for this processing speed problem. Presently, two different approaches are being pursued.

In the first approach, large portions of the Java VM are incorporated into the smart card microcontroller as dedicated hardware components that supplement the actual processor. This technique thus goes in the direction of picoJava, which means in the direction of a real IC that

[5] The quality of random numbers is treated in overview in Section 4.10.2, 'Testing random numbers'

can directly process Java bytecode. This solution has two drawbacks, which are that the Java processor takes up additional space, besides that occupied by the regular processor, and that a full implementation of a Java VM is relatively costly. The advantage of this solution is its high execution speed.

In the second approach, the instruction set of the processor is extended to include typical Java machine instructions. This allows bytecodes supplied by the software VM to be immediately processed by the extended processor. This variant is implemented using a processor lookup table containing CPU microinstruction sequences corresponding to the bytecodes to be emulated. The advantage of this solution relative to the first one is that it requires less additional space on the chip, although its execution speed is somewhat lower.

Coprocessors for symmetric cryptographic algorithms

Up to now, DES has been used as the standard cryptographic algorithm for financial transaction systems and telecommunications applications. This large market potential made it worthwhile for semiconductor manufacturers to fit smart card microcontrollers with their own DES calculation units. In principle, this is not particularly difficult, since DES was originally designed to be primarily implemented in hardware. The largest problem in marketing DES calculation units in microcontrollers is not technical, but instead relates to export restrictions, since in many countries components with fast, hardware-based DES encryption are subject to a variety of export regulations.

The advantages of DES calculation units for smart card microcontrollers can be clearly seen by examining their performance figures. At 3.5 MHz, they can achieve times on the order of 75 μs for a simple DES operation and 150 μs for a triple-DES operation with two keys. The calculation time decreases linearly as the clock rate is increased. Besides this, a DES calculation unit does not require significantly more chip area than that occupied by the ROM code for a software DES implementation, so it does not increase the size of the die.

In the future, besides DES coprocessors there will also be special coprocessors for AES in smart card microcontrollers, usually supporting all three possible key lengths (128, 196 and 256 bytes). This is technically just as feasible as a DES coprocessor, since the AES algorithm is also relatively easy to implement in hardware.

Coprocessors for asymmetric cryptographic algorithms

For calculations in the realm of public-key algorithms, such as RSA and elliptic-curve algorithms, there are specially developed arithmetic units that are placed on the silicon along with the usual functional components of a smart card microcontroller. These arithmetic units are only capable of performing several basic calculations that are necessary for these types of algorithms, namely exponentiation and modulo calculations using large numbers. The speed of these components, which are optimized for these two arithmetic operations, is due to their very broad architectures (up to 140 bits). In their particular application area, some of them can even outperform a powerful PC.

The arithmetic unit is called by the processor, which either passes the data directly or passes a pointer to the data and then issues an instruction to start the processing. After the task has been completed and the result has been stored in RAM, control of the chip is returned to the processor. In general, these coprocessors can process all key lengths up to 1024 bits for the RSA algorithm, and in the medium term this will increase to 2048 bits. For elliptic curves, the usual capacity is up to 160 bits, with 210 bits to come in the future.

Error detection and correction in EEPROM

The essential limitation on the useful life of a smart card is imposed by the EEPROM, with its technically limited number of possible write/erase cycles. One way to relax this limitation is to use software to calculate error correction codes for certain heavily used regions of EEPROM, so that errors can be corrected. It is also possible to implement error correction codes using hardware circuitry on the chip. In this way, EEPROM errors can be detected and corrected (as long as they are not too extensive) in a manner that is transparent to the software.

Naturally, additional EEPROM is necessary to store the codes. Since good error correction codes take up a relatively large amount of memory, the designer is confronted with a strategic decision: good error detection demands extra memory – up to 50 % of the memory to be protected. What's more, the memory for the error correction mechanism can only be used for this purpose. Although lower performance error correction requires less additional memory, its usefulness is highly questionable.

There are a few microcontrollers on the market that have EEPROM error detection and correction implemented in hardware, but they may require extra memory amounting to as much as half the volume of the memory to be protected to be used for the protection codes. As a result, the amount of EEPROM available to the user may not be particularly large. However, the useful life of an EEPROM secured using this mechanism is several times the usual value.

Chip hardware extensions

If the chip hardware must be extended for a particular reason, considerable expenditures of development effort and costs are required on the part of the manufacturer. There are only two ways to implement customer-specific hardware: it can be built in silicon on the basis of an existing chip family, or it can be built as a two-chip system, with all of the associated drawbacks.

There is an acceptable solution to this problem in the form of a compromise that incorporates elements of both of these options. A chip with the new hardware unit can be glued directly to the existing chip and electrically connected to it by bonding wires. This solution benefits from the fact that most smart card microcontrollers have several I/O ports, and one of these ports can be used to communicate with the additional chip. The thickness of the resulting sandwich construction is not significantly greater than that of a normal chip, since the silicon substrates can be ground away more than usual to make them thinner. A sandwich chip can thus be built into a standard module without additional effort or costs.

This technique is ideal for satisfying customer-specific needs for additional hardware without expensive redesign. An existing chip can be combined with a new unit, which may for instance have a special serial interface for testing the security features of other chips. It is also possible to fit a special ASIC containing a secret cryptographic algorithm into the card.

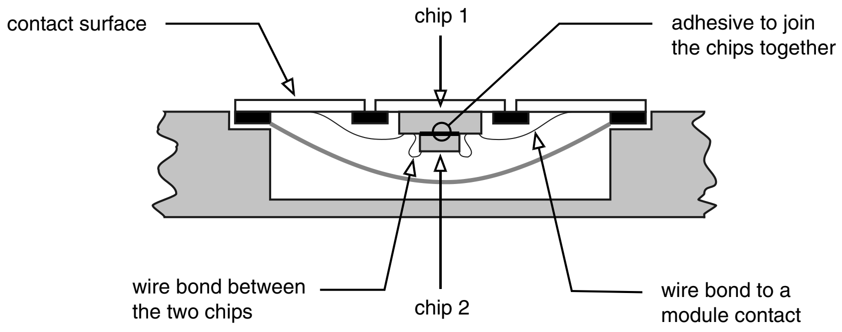

contact surface
chip 1
adhesive to join the chips together
wire bond between the two chips
chip 2
wire bond to a module contact

Figure 3.63 Cross section of a chip module containing two different chips electrically interconnected via bonding wires

This method is not cost-effective for large production quantities (in the range of millions of pieces), since in such cases it is worthwhile to develop special chips. However, for small to medium piece counts, sandwich chips are a very effective solution for prototype series or special applications, such as security modules for terminals or smart cards for pay-TV decoders.[6]

Vertical system integration (VSI) and face-to-face

Another technique used to extend chip hardware by combining semiconductor technologies that are incompatible on a single chip is vertical system integration (VSI), in which two or

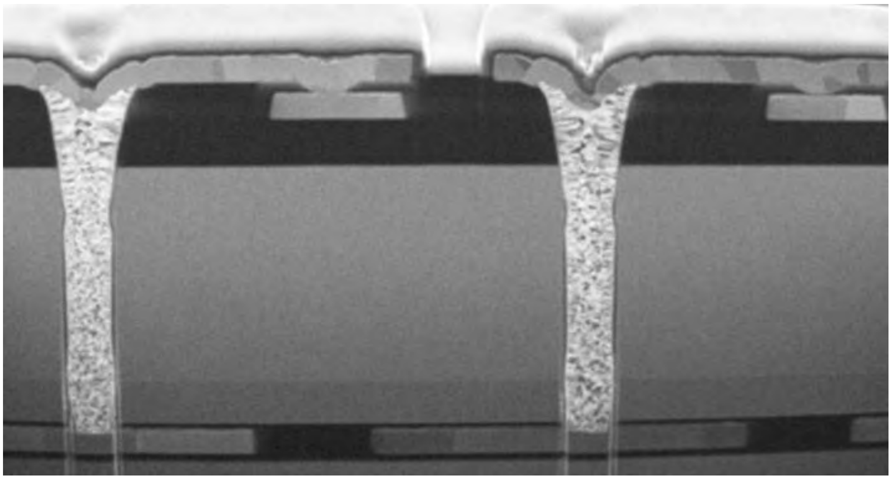


Figure 3.64 Cross-sectional photograph of a VSI stack. The two through contacts between the two stacked dies can be easily recognized (*Source:* Giesecke & Devrient)

[6] See also [Kuhn]

more dies that have been ground thin are bonded together mechanically to form a stack, with the individual dies also being electrically interconnected by through contacts ('vias') formed using semiconductor fabrication processes. Using VSI, the available chip area can be increased in units of the original area in a very elegant manner. With two stacked dice, twice the original area is available, and with four dice there is four times as much area. It is possible to achieve not only a significant increase in the amount of available memory, but also a considerable improvement in chip security. This is because it is presently effectively impossible to access a chip sandwiched between two other dice using analytical equipment of the type used in the semiconductor industry without destroying the surrounding chips.

A simpler variant of VSI, which in principle can be scaled up as much as desired, is the face-to-face arrangement of two chips. Here the electrical connections are made by extremely precise positioning of the two chips, with their upper surfaces (faces) touching.

VSI and face-to-face chip bonding both allow significantly better extensions of chip hardware to be realized than what can be achieved by interconnecting two chips using wire bonds.

3.5 CONTACT-TYPE CARDS

The main difference between a smart card and other types of cards is the embedded microcontroller. If contacts are used for the power supply and data transmission functions, electrical

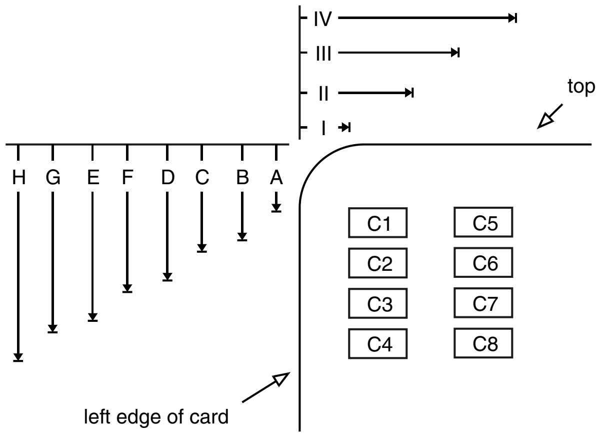


Figure 3.65 The positions of the contacts relative to the card body outline (drawing not to scale)

I	10.25 mm maximum	A	19.23 mm minimum
II	12.25 mm minimum	B	20.93 mm minimum
III	17.87 mm maximum	C	21.77 mm maximum
IV	19.87 mm minimum	D	23.47 mm minimum
		E	24.31 mm maximum
		F	26.01 mm minimum
		G	26.85 mm maximum
		H	28.55 mm minimum

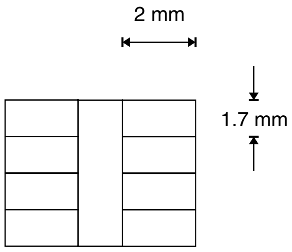


Figure 3.66 Minimum contact dimensions as specified in ISO 7816-2

connections are required. These consist of six or eight gold-plated contacts, which can be seen on every standard smart card. The locations of these contacts with respect to the card body, and their sizes, are specified by the ISO 7816-2 standard.

In France, a national standard generated by AFNOR was already in use long before ISO 7816-2 was issued. It specifies a slightly higher location for the contacts than the ISO standard. This location is also included in the ISO standard as a 'transitional contacts location', but the standard recommends that this location not be used in the future. However, since there are still many cards in France that use the 'transitional' position, it is not likely that it will disappear quickly.

The absolute position of the contact field is in the upper left corner of the card body. The is clearly shown in dimensioned drawing shown in Figure 3.65. The minimum dimensions of any contact are 1.7 mm by 2 mm (height by width). The maximum dimensions of any contact are not specified, but they are of course limited by the fact that the individual contacts must be electrically isolated from each other.

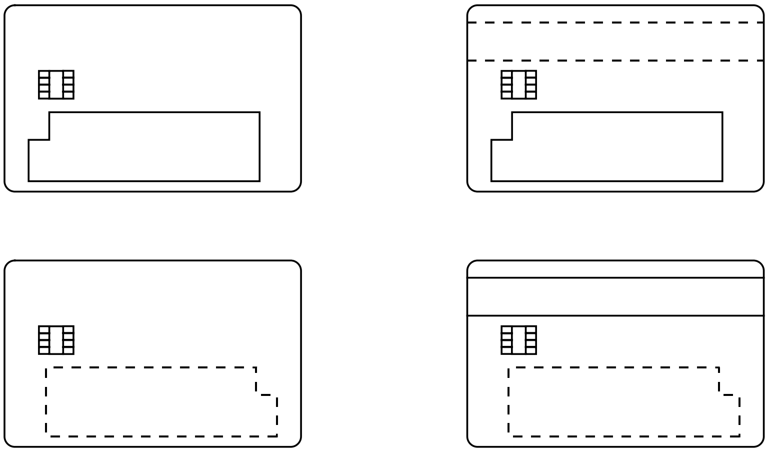


Figure 3.67 The various possible arrangements for the chip, embossing field and magnetic stripe, according to the ISO 7816-2 standard

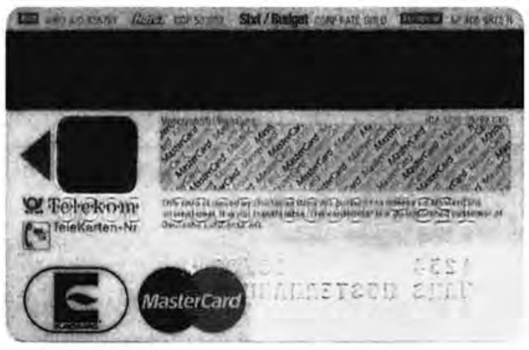


Figure 3.68 An example of a card with a chip, magnetic stripe, signature area and embossing (*Source:* Giesecke & Devrient)

The position of the module within the card body is specified in the standard. The locations of the magnetic stripe area and the area reserved for embossing are also exactly specified (see ISO 7811). All three of these components may be present on a single card. However, in this case the following mutual relationships must be taken into account: (a) if only a chip and an embossing field are present, they may be located on the same side or on opposite sides of the card; (b) if a magnetic stripe is also present, it and the embossing area must be located on opposite sides of the card.

3.6 CONTACTLESS CARDS

As already described in Section 2.3.3, contactless cards do not require any electrical connection between the smart card and the card terminal in order to transfer energy and data over a short distance. The most important advantages of the contactless card technology are described in Section 2.3.3. In this section we examine the technology and operating principles of contactless cards in more detail. The techniques used with contactless cards for transferring energy and data are not new. They have been common knowledge for many years in radio-frequency identification (RFID) systems, which have been used for a variety of applications, such as animal implants and transponders for electronic anti-start systems for vehicles. There are many techniques for identifying persons or objects at short or even long distances based on radio techniques, and in particular on radar techniques. Among the large variety of technical possibilities, only a small number are suitable for use in smart cards in the ID-1 format (to which we restrict our attention), since all of the functional components must be housed in a flexible card that is only 0.76 mm thick. For instance, fitting flexible batteries into the card body remains an unsolved problem for mass-produced cards. Although flexible batteries with suitable thickness are now available, there is no experience with using such batteries in the field or in mass production. Consequently, we are still limited to passive techniques in which the energy to power the card must be extracted from the electromagnetic field of the card terminal. This limits the useful range to around 1 m.

To make it easier to understand the variety of techniques used, they can be classified according to various parameters. One possibility is to classify them according to the method used to transfer energy and data. The most commonly used methods are transmission using radio waves or microwaves, optical transmission, capacitive coupling and inductive coupling. Capacitive and inductive coupling are best suited to the flat shape of a smart card lacking an internal source of power. The systems presently available on the market utilize these methods exclusively, which are also the only ones considered in the relevant group of ISO/IEC standards (10 536, 11 443 and 15 693). Consequently, in this book we limit ourselves to these methods.

Just as with contact-type smart cards, a system using contactless cards consists of at least two components, namely a card and a compatible terminal. The terminal can act as a reader or a reader/writer, according to the technology used. As a rule, the terminal includes an additional interface, via which it can communicate with a background system.

The following four functions are necessary to allow a contactless card to communicate with a terminal:

- energy transfer to the card for powering the integrated circuit
- clock signal transfer
- data transfer to the smart card
- data transfer from the smart card.

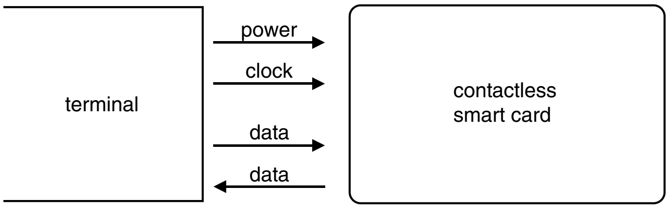


Figure 3.69 The necessary energy and data transfers between a terminal and a contactless smart card

Many different concepts based on experience with RFID systems have been developed to satisfy these requirements. Most of them are specifically designed for particular applications. For instance, there is a considerable difference between systems where the cards are only a few millimeters away from the terminal in normal use and systems where the cards can by up to a meter away from the terminal. Naturally, when many different solutions specifically designed and optimized for particular applications are developed, they are inevitably mutually incompatible.

Inductive coupling

Inductive coupling is presently the most widely used technique for contactless smart cards. It can be used to transfer both energy and data. Various requirements and constraints, such as radio licensing regulations, have resulted in a variety of actual implementations.

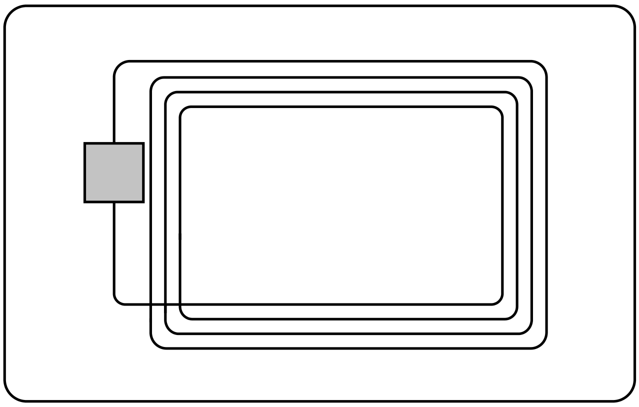


Figure 3.70 Basic construction of a contactless smart card with inductive coupling

With some applications, such as access control, it is sufficient to only be able to read the data stored in the cards, which makes technically simple solutions possible. Due to their low power consumption (a few tens of microwatts), the usable range of such cards extends to approximately one meter. Their memory capacity is usually only several hundred bits. If data must also be written, the power consumption rises to more than 100 µW. As a consequence, the range is limited to around 10 cm in the writing mode, since licensing restrictions prevent the emitted power of the writing equipment from being arbitrarily increased. The power consumption of microprocessor cards is even greater and is typically 100 mW. The distance from the terminal is thus even more restricted.

Figure 3.71 Inlay foil for a contactless smart card with inductive coupling using an etched coil

Independent of their range and power consumption, all cards that employ inductive coupling work on the same principle. One or more coils (usually with large enclosed areas) are incorporated into the card body to act as coupling components for energy and data transfers, along with one or more chips.

Energy transfer

Almost without exception, contactless smart cards are used passively. This means that all of the energy needed for operating the chip in the smart card must be transferred from the reader to the card.

This energy transfer is based on the principle of a loosely coupled transformer. A strong high-frequency magnetic field is generated by a coil in the terminal in order to transfer the energy. The most commonly used frequencies are <135 kHz and 13.56 MHz, which correspond to wavelengths of 2400 m and 22 m, respectively. The wavelengths of the electromagnetic fields are thus several times greater than the distance from the card to the terminal, which means that the card is located in the near field of the terminal. This allows the loosely coupled transformer model to be used. If a contactless card is brought close to the terminal, a portion of the terminal's magnetic field passes through the coil in the card and induces a voltage U_i in this coil. This voltage is rectified to provide power to the chip. Since the coupling between the coils in the terminal and the card is very weak, the efficiency of this arrangement is very low. A high current level is thus required in the terminal coil to achieve the necessary field strength. This is achieved by connecting a capacitor C_T in parallel with the coil L_T, with the value of the capacitor chosen such that the coil and capacitor form a parallel-resonant network whose resonant frequency matches the frequency of the transfer signal.

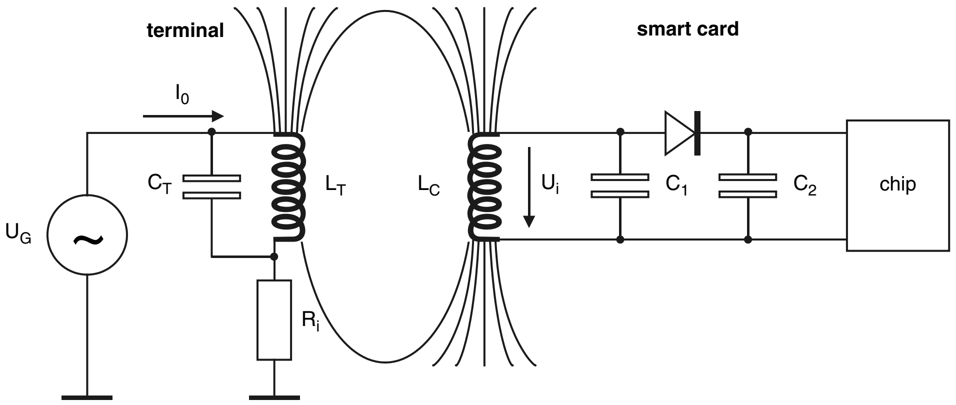


Figure 3.72 Using inductive coupling to supply energy to a smart card

Coil L_C and capacitor C_1 in the card also form a resonant circuit with the same resonant frequency. The voltage induced in the card is proportional to the signal frequency, the number of windings of coil L_C and the enclosed area of the coil. This means that the number of turns needed for the coil drops with increasing signal frequency. At 125 kHz, it is 100 to 1000 turns, while at 13.56 MHz it is only 3 to 10.

Data transfer

For transferring data from the terminal to the card, all known digital modulation techniques can be used. The most commonly used techniques are:

- ASK (amplitude-shift keying)
- FSK (frequency-shift keying)
- PSK (phase-shift keying).

ASK and PSK are usually used, since these are especially easy to demodulate.

In the other direction, from the smart card to the terminal, a type of amplitude modulation is used. It is generated by using the data signal to digitally alter a load in the card (load modulation). If a smart card tuned to the resonant frequency of the terminal is brought into the near field of the terminal, it draws energy from this field as previously described. This causes the current I_0 in the coupling coil of the terminal to increase, which can be detected as an increased voltage drop across an internal resistor R_i. The smart card can thus vary (amplitude modulate) the voltage U_0 in the terminal by varying the load on its coil, for example by switching the load resistor R_2 into and out of the circuit as shown in Figure 3.73. If the switching of resistor R_2 is controlled by the data signal, the data can be detected and evaluated in the terminal.

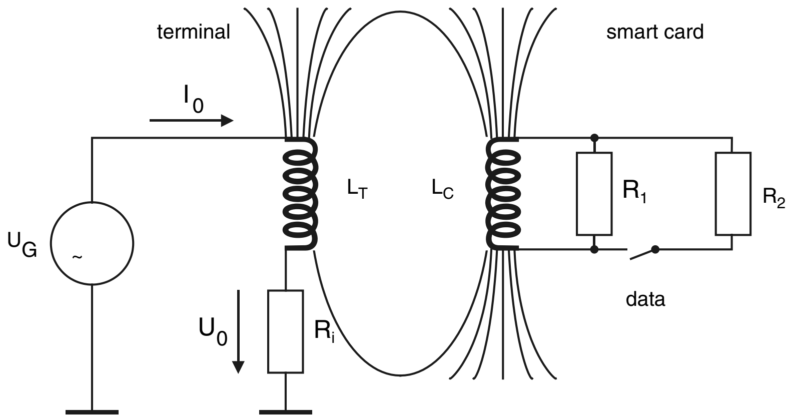


Figure 3.73 A sample circuit illustrating the principle of load modulation, which is used in contactless smart cards for data transmission

Due to the low degree of coupling between the coils in the terminal and the card, the voltage variations induced in the terminal by load modulation are very small. In practice, the amplitude of the usable signal is only a few millivolts. This can only be detected using sophisticated circuitry, since it is overlaid by the significantly larger signal (around 80 dB) transmitted by the terminal. However, if a subcarrier frequency is employed with a frequency of f_s, the received data signal appears in the terminal as two sidebands at the frequencies $f_c \pm f_s$. These can be isolated from the significantly stronger terminal signal by filtering with a bandpass filter and then amplified. After this, they can readily be demodulated. The

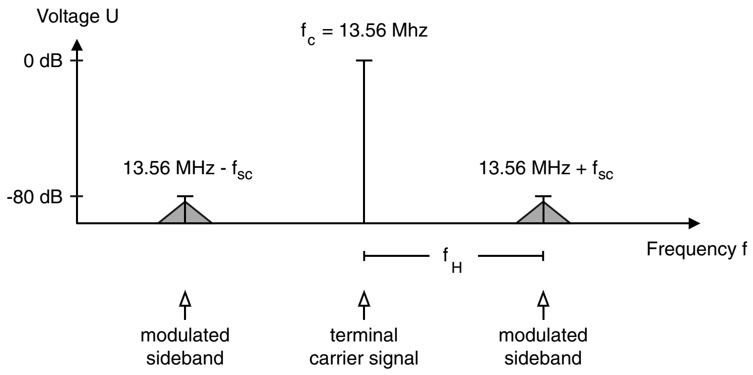


Figure 3.74 Load modulation using a subcarrier produces two sidebands separated from the transmission frequency of the terminal by the value of the subcarrier frequency f_s. The information is contained in the sidebands of the two subcarrier sidebands, which are produced by modulation of the subcarrier (based on Klaus Finkenzeller [Finkenzeller 02])

disadvantage of modulation with a subcarrier is that it requires significantly more bandwidth than direct modulation. It can thus only be used in a limited number of frequency bands.

Capacitive coupling

If the distance between the card and the terminal is very small, it is possible to transfer data using capacitive coupling. With this type of coupling, conductive surfaces are incorporated

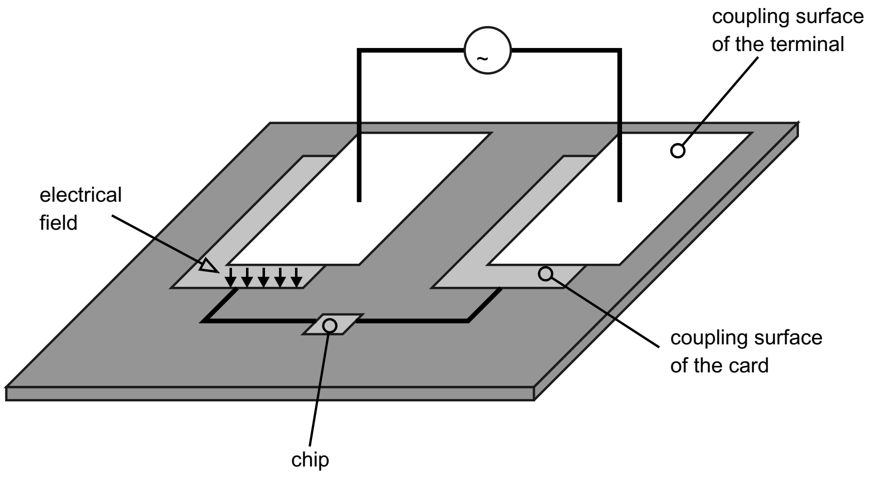

coupling surface of the terminal
electrical field
coupling surface of the card
chip
~

Figure 3.75 Operating principle of capacitive coupling. The coupling arises from the alternating electrical field between two parallel, electrically conductive surfaces in the card and terminal

into the card body and the terminal such that they act as the plates of a capacitor when the card is inserted in the terminal or placed on the terminal. The capacitance that can be obtained essentially depends on the sizes of the coupling surfaces and their separation. The maximum size is thus limited by the dimensions of the card, while the minimum separation is determined by the insulation required between the coupling surfaces. With an acceptable level of cost and effort, a usable capacitance of several tens of picofarads can be obtained. This is insufficient for transferring enough energy to power a microprocessor. Consequently, this method is used only for data transmission, with the operating power being transferred inductively. This mixed method has been standardized in ISO/IEC 10 536 for 'close coupling cards', and it is fully described in Section 3.6.1. As its name says, this method is limited to small coupling distances.

Collision avoidance

When contactless cards are used, there is always a possibility that two or more cards may be located in the range of a terminal at the same time. This is especially true for systems with large effective ranges, but it can even happen with systems with relatively small ranges – for instance, two cards might be lying on top of each other and thus be activated concurrently by the terminal. All cards within range of a particular terminal will attempt to respond to commands from the terminal. However, simultaneous data transmissions will unavoidably cause interference and loss of data if suitable countermeasures are not taken. The technical methods used to ensure interference-free data exchanges with multiple cards within the effective range of a card terminal are called collision-avoidance methods or anticollision methods.

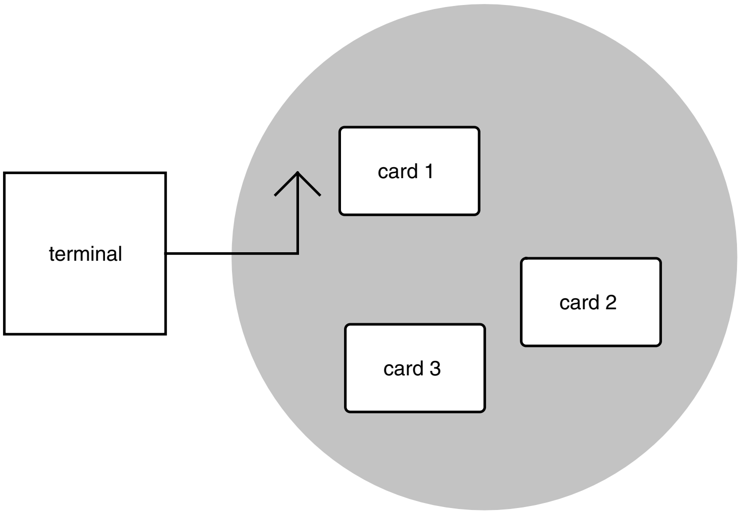


Figure 3.76 Concurrent operation of several cards within the range of a terminal (multiple access) requires using an anticollision method to ensure interference-free data exchanges

Exchanging data between many mobile units and a base station is a frequently encountered situation in communications engineering, and it is referred to as 'multiple access'. A typical

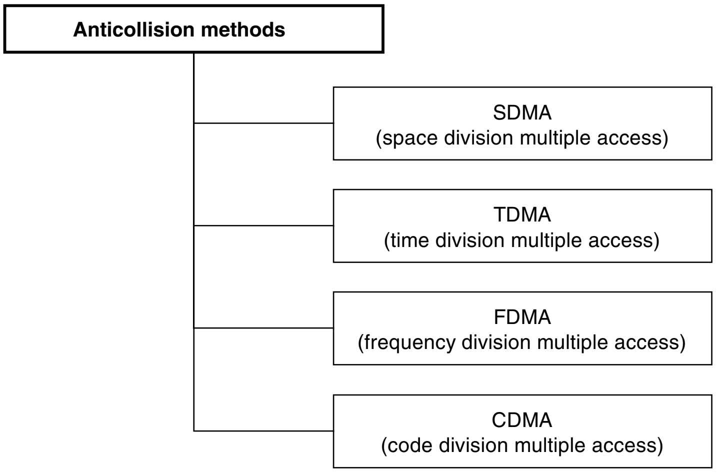

Anticollision methods
SDMA
(space division multiple access)
TDMA
(time division multiple access)
FDMA
(frequency division multiple access)
CDMA
(code division multiple access)

Figure 3.77 The four types of anticollision methods

example is a mobile telephone network, in which all users located in a particular radio cell concurrently access a single base station. Numerous methods have been developed to allow the signals of the individual users to be distinguished from each other. These anticollision methods can be classified into four types, as shown in Figure 3.77.

Space division multiple access (SDMA) attempts to limit or scan the operational area of a terminal in such a way that only one card can be acquired at any given time. Since this method requires very complicated and correspondingly expensive aerials, it is not used for contactless cards.

With time division multiple access (TDMA), measures are taken to ensure that the individual cards have different timing behavior so that they can be separately identified and individually addressed by the terminal. This is the most commonly used method, and it has many variants. Two of them, which are standardized in ISO/IEC 14 433-3 for 'proximity cards', are described extensively in Section 3.6.3.

With frequency division multiple access (FDMA), different carrier frequencies are provided concurrently for multiple transmission channels. However, this technique is technically complicated and thus expensive. Consequently, it is not used for contactless cards. The same considerations also apply to code division multiple access (CDMA).

The present state of standardization

Given the many different techniques used by various manufacturers, standardization (which was initiated in 1988 by ISO/IEC) proved to be difficult and time consuming, as was expected. The responsible working group had the task of defining a standard for contactless cards that is largely compatible with other standards for identification cards. This means that a contactless card can also have other functional components, such as a magnetic stripe, embossing and chip contacts. This allows contactless cards to also be used in existing systems that employ other technologies. As already described, the technical options for transferring energy and data without using contacts essentially depend on the desired distance between the card and the terminal for reading and writing data. It was therefore not possible to create a single standard that provides a single technical solution to all the requirements arising from various applications.

Presently, three different standards describing three different reading ranges have been completed. Each of these standards in turn permits various technical solutions, since the members of the standardization committee could not agree on a single solution. In order to achieve interoperability among the various options, card terminals must support all of these options.

Table 3.6 Completed ISO/IEC standards for contactless smart cards

Standard	Type of contactless smart card	Range
ISO/IEC 10 536	Close-coupling card	Up to approx. 1 cm
ISO/IEC 14 443	Proximity coupling card (PICC)	Up to approx. 10 cm
ISO/IEC 15 693	Vicinity coupling card (VICC)	Up to approx. 1 m

Standardization started with ‘close-coupling’ cards (ISO/IEC 10536), since the microprocessors available at that time had relatively high power consumption, making energy transfer over a relatively large distance impossible. The essential parts of this standard have been completed and approved and are described in the following section. In use, this type of card offers only minimal advantages compared with normal contact-type cards, since it must be inserted into a terminal or at least precisely placed on a surface of a card terminal. Furthermore, the structure of the card is complex, which results in high manufacturing costs. Consequently, up to now this type of system has hardly established a significant position in the market.

3.6.1 Close-coupling cards: ISO/IEC 10536

In the ISO/IEC 10536 standard for close-coupling cards, this application is designated as ‘slot or surface operation’, which expresses the fact that in use the card must be inserted into a slot or laid on a marked surface of the terminal. The ISO/IEC 10536 standard , which bears the title ‘Identification Cards – Contactless Integrated Circuit(s) Cards’, consists of four parts:

- Part 1: Physical characteristics
- Part 2: Dimension and location of coupling areas
- Part 3: Electronic signals and reset procedures
- Part 4: Answer to reset and transmission protocols.

Parts 1 through 3 have already become international standards, while Part 4 is still in preparation. The important ingoing requirements for these standards were the following:

- extensive compatibility with ISO 7816
- operation with arbitrary orientation of the card to the reader

- transfer carrier frequency between 3 and 5 MHz
- bidirectional data transmission with inductive or capacitive coupling
- card power consumption less than 150 mW (adequate for microprocessor chips).

Part 1 of the standard defines the physical characteristics of the card. Essentially the same requirements are imposed as for contact-type smart cards, particularly with regard to bending and twisting. One difference is in the tolerance to electrostatic discharge. Since a contactless card does not require any conductive path between the card surface and the integrated circuit embedded in the card body, it is largely insensitive to damage from ESD. A test voltage of 10 kV is thus specified in the standard, compared with 1.5 kV for cards with contacts.

Part 2 of the standard specifies the locations and dimensions of the coupling components. Since it was not possible to agree on a single method, both capacitive and inductive coupling

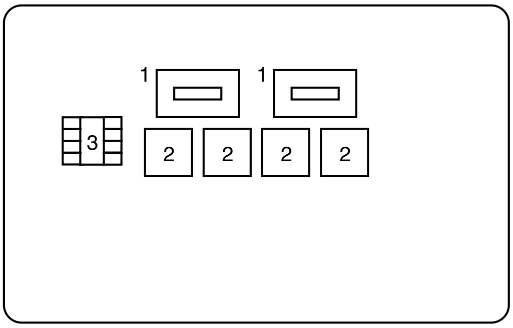


Figure 3.78 The arrangement of the coupling components of a contactless smart card: (1) coupling coils in the card body, (2) capacitive coupling surfaces in the card body and (3) a set of contacts for the chip

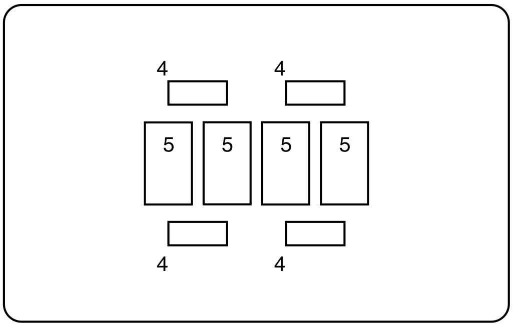


Figure 3.79 The arrangement of the coupling components in a terminal for contactless smart cards: (4) coupling coils in the terminal, (5) capacitive coupling surfaces in the terminal

components are defined in such a way that both can be implemented together in a single card or terminal. Examples of this are shown in Figures 3.78 and 3.79. The chosen arrangement is intended to ensure orientation independence with suitable excitation in the terminal.

Part 3 of the standard, published in 1996, is the most important part to date. It describes the modulation methods to be used for capacitive and inductive data transmission, since agreement on a single method could not be achieved. A terminal that complies with the standard must therefore support both methods, and both methods may be implemented in a single card.

Energy transfer

Energy is transferred by a sinusoidal alternating magnetic field with a frequency of 4.9152 MHz, which passes through one or more inductive coupling surfaces, depending on how many coupling coils are present in the card. The terminal must generate all four fields.

Alternating magnetic fields *F1* and *F2*, which pass through areas *H1* and *H2*, have a mutual phase difference of 180 degrees, as do fields *F3* and *F4*, which pass through areas *H3* and *H4*. The phase difference between fields *F1* and *F3* and between *F2* and *F4* is 90 degrees. Each magnetic field is strong enough to transfer at least 150 mW to the card. However, the card should not consume more than 200 mW. This complicated definition of the magnetic fields is necessary to achieve the same data transfer characteristics for four different card orientations, as explained below.

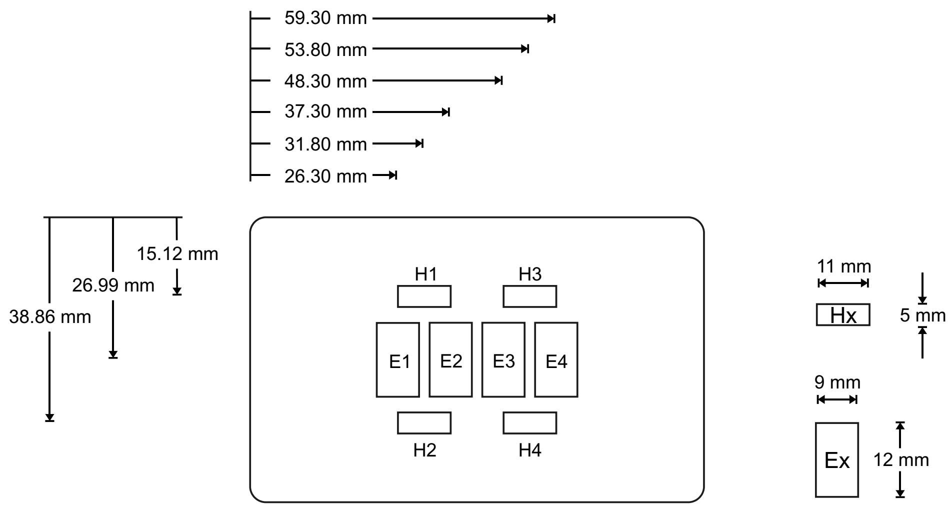


Figure 3.80 Locations and sizes of the coupling areas in the contactless card and the terminal

3.6.1.1 Inductive data transfer

Different types of modulation are used for data transmission in the two directions.

Data transmission from the card to the terminal

For data transmission from the card to the terminal, a 307.2 kHz subcarrier is first generated using load modulation (see Figure 3.81), with a load variation of at least 10 %. Data modulation is achieved by switching the phase of the subcarrier by 180 degrees, producing two phase states that can be interpreted as logic 1 and logic 0. The initial state after the magnetic field has been established is defined to be logic 1. This initial state (interval t_3 in Figure 3.84) remains stable for at least 2 ms. Following this, every subcarrier phase shift represents a reversal of the logic state, yielding non-return to zero (NRZ) coding. The data transmission rate, at least for the ATR, is 9600 bits per second.

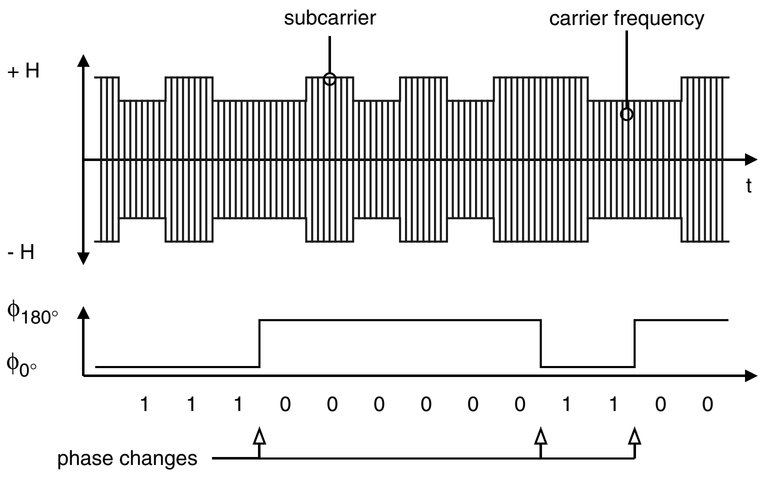


Figure 3.81 Operating principle of phase modulation for data transmission with a contactless smart card. The upper diagram shows the alternating magnetic field, and the associated phase states are shown in the lower diagram. The carrier frequency is 4.9152 MHz, and the subcarrier frequency is 307.2 kHz

Data transmission from the terminal to the card

To transfer data from the terminal to the card, the four alternating magnetic fields *F1* through *F4*, which pass through coupling surfaces *H1* through *H4*, are phase modulated using phase-shift keying (PSK). This causes the phase of all four fields to simultaneously shift by 90 degrees. In this way, two phase states A and A' are defined. Depending on the orientation of the card relative to the terminal, this yields two different constellations of phase states, as shown in Figures 3.82 and 3.83.

Since the card must work in all four possible orientations with respect to the terminal, the initial state (intervals t_2 and t_3 in Figure 3.84) is interpreted as a logic 1, regardless of which

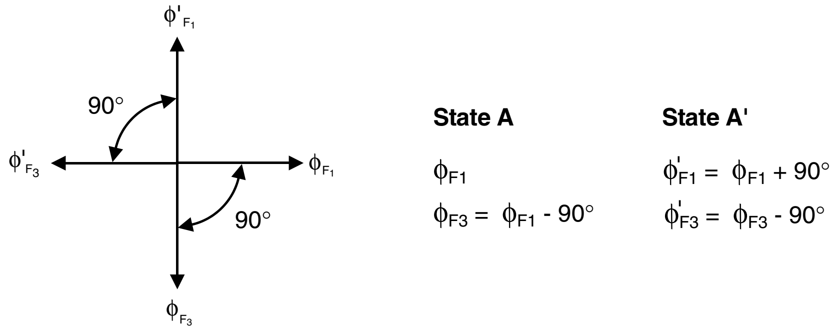


Figure 3.82 The first phase modulation variant for data transmission with a contactless chip card. The four arrows represent phase vectors

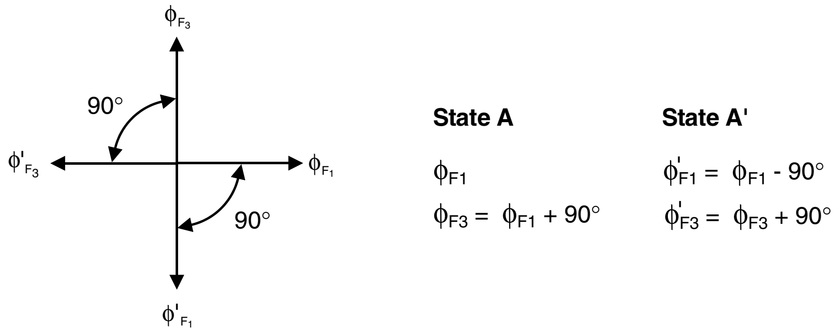


Figure 3.83 The second phase modulation variant for data transmission with a contactless chip card. The four arrows represent phase vectors

of the indicated alternatives is actually present. Following this, every phase change represents a reversal of the logic state, which again produces an NRZ encoding.

3.6.1.2 Capacitive data transfer

For capacitive data transmission from the card to the terminal, one pair of coupling surfaces is used, depending on the orientation of the card relative to the terminal – either *E1* and *E2* or *E3* and *E4* as shown in Figure 3.80. The other pair of coupling surfaces can be used for data transmission in the opposite direction. Since the card sends the ATR via one particular pair of coupling surfaces, the terminal can recognize the relative orientation of the card. The maximum potential difference between a pair of coupling surfaces is limited to 10 V, but it must at least exceed the minimum differential voltage of the receiver (± 300 mV). Differential NRZ encoding is used for data transmission. The transmitter generates the encoding by reversing the voltage between surfaces *E1* and *E2* or *E3* and *E4*. The state representing a logic 1 is again established in interval t_3 (see Figure 3.84). Following this, every polarity reversal represents a change in the logic state.

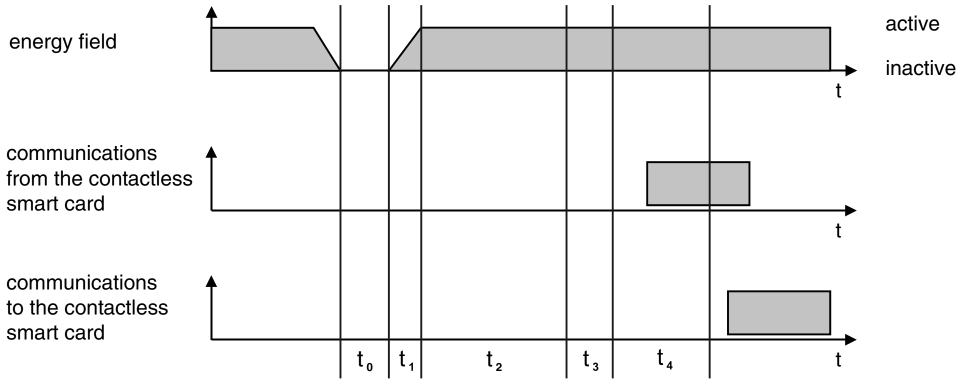

energy field
active
inactive
t
communications from the contactless smart card
t
communications to the contactless smart card
t
t_0
t_1
t_2
t_3
t_4

Figure 3.84 Timing diagram for data transmission with a contactless smart card according to ISO/IEC 10536 3. Here $t_0 \geq 8$ ms, $t_1 \leq 0.2$ ms, $t_2 = 8$ ms, $t_3 = 2$ ms and $t_4 \leq 30$ ms

Initial state and answer to reset (ATR)

In order for the terminal to unambiguously determine the type of data transmission and the orientation of the card at the beginning of a data exchange, certain time intervals must be defined for initiating energy and data transfers. Figure 3.84 shows the constraints and values for the reset recovery time t_0, power-up time t_1, initialization time t_2, stable logic state time t_3 and answer to reset time t_4.

Minimum reset recovery time: t_0

If a reset is to be produced by switching the energy-transfer field off and back on, the time between switching the field off and then on again, during which no energy is transferred, must be equal to or greater than 8 ms.

Maximum power-up time: t_1

The time required for the energy-transfer field produced by the terminal to be established must be less than or equal to 0.2 ms.

Initialization time: t_2

The initialization time, which is the time allowed for the card to attain a stable operating state, is 8 ms.

Stable logic state time: t_3

Prior to the Answer to Reset, the logic state is held at the logic 1 level for 2 ms. During this interval, the card and the terminal are set to logic 1 for inductive data transmission.

Maximum response time for ATR: t_4

The card must start sending the ATR before 30 ms have elapsed. The card can use the ATR to indicate that the conditions for subsequent operation must be changed with regard to energy level, data transmission rate or the frequency of the fields. The ‘maneuvering room’ provided here can be utilized according to the requirements of the application. For instance, a significantly higher data transmission rate can be selected for a time-critical application.

3.6.2 Remote-coupling cards

The term ‘cards with remote coupling’ encompasses smart cards that can transmit data over a distance of a few centimeters to approximately one meter from the terminal. This capability is of great significance for all applications in which data should be exchanged between the card and the terminal without requiring the card user to take the card in his or her hand and insert it into a terminal. Some sample applications are:

- access control
- vehicle identification
- electronic driver’s licenses
- ski passes
- airline tickets
- electronic purses
- baggage identification.

The variety of applications suggests that there are a large number of possible technical implementations. In the preparation of the standards, an attempt was made to limit the number

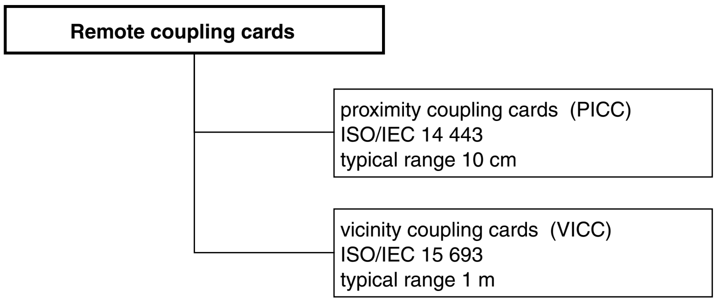


Figure 3.85 For remote-coupling cards, the ISO/IEC standards distinguish between proximity cards and vicinity cards

of technical variants, with only mixed success. International standards ISO/IEC 14 443 and ISO/IEC 15 633 cover the ranges of up to 10 cm and 1 m, respectively.

3.6.3 Proximity integrated circuit(s) cards: ISO/IEC 14 443

The ISO/IEC 14 443 standard, which is titled 'Identification cards – Contactless integrated circuit(s) cards – Proximity cards', describes the properties and operation modes of contactless smart cards with a range of approximately 10 cm. The amount of energy that can be transferred over this range is sufficient to power a microprocessor. In order to allow this type of card to be used with existing infrastructures for contact-type cards, they often have contacts in addition to the coupling components for contactless operation, so that they can be used with or without contacts as desired. This type of card is called a 'dual-interface card' or 'combicard'.

The ISO/IEC 14 443 standard consists of the following parts:

- Part 1: Physical characteristics
- Part 2: Radio frequency power and signal interface
- Part 3: Initialization and anticollision
- Part 4: Transmission protocol.

Physical characteristics

The physical characteristics of proximity cards, which are defined in Part 1 of the ISO/IEC standard for proximity integrated circuit cards (PICCs), essentially correspond to the requirements specified for smart cards with contacts. It is to be expected that in use, proximity cards will be exposed to electromagnetic fields corresponding to those intended to be used for the operation of other types of cards that comply with standards such as ISO/IEC 10 536 or ISO/IEC 15 693. The cards must not suffer permanent damage as the result of exposure to such fields or the environmental stress of normal ambient electromagnetic fields. In order to ensure this, the standard specifies maximum values for stresses due to alternating electric and magnetic fields that the cards must withstand without damage. It is the task of the semiconductor manufacturer to design the chips such that they meet these requirements.

Radio-frequency power and signal interface

Proximity cards work on the principle of inductive coupling. Operating power and data are both transferred using an alternating magnetic field generated by the card terminal. In the ISO/IEC 14 443 standard, the card terminal is called a 'proximity coupling device' (PCD). For the sake of readability, in the following description the more general term 'terminal' and 'PCD' are used interchangeably.

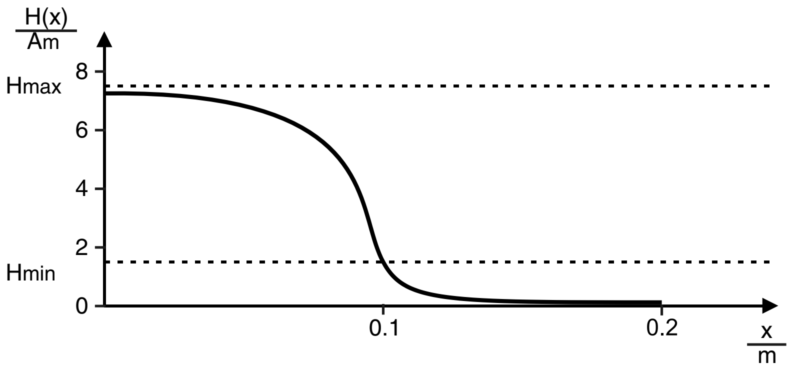


Figure 3.86 Typical magnetic field strength characteristic of a terminal for proximity cards (PCD)

The transmission frequency of the PCD is set to $f_C = 13.56$ MHz ± 7 kHz, with a magnetic field strength H of at least 1.5 A/m and at most 7.5 A/m (effective value). The typical field strength versus distance is shown in Figure 3.86.

The range of the system can be estimated from the field strength of the PCD and the activation field strength of a proximity card (PICC). With the typical field strength curve shown in Figure 3.86 and an assumed PICC activation field strength of 1.5 W/m, we obtain a range of approximately 10 cm.

Signal and communication interface

Two different communication interfaces are defined in the ISO/IEC 14 443 standard, which are designated Type A and Type B. The reason for standardizing two different methods was not technical, but rather that at the time that ISO/IEC 14 443 was being prepared, various designs from different manufacturers were already in existence. As is often the case with standardization, the differing interests of the persons involved made it impossible for them to agree on a single method, although that would have been technically desirable. The two methods mentioned above were agreed on as a compromise, and they were published as an international standard in June 2001. Even with the already existing methods, it is necessary for card terminals to support both methods in order to achieve full interoperability with all cards meeting the ISO/IEC 14 433 standard, since the cards generally support only one of the two standard methods.

While a terminal is waiting to detect a proximity card, it must periodically switch back and forth between the two communications methods. This allows it to recognize both Type-A and Type-B cards. Once the PCD has recognized a card, it continues to use the appropriate communications method until the card is deactivated by the terminal or leaves the working range of the terminal.

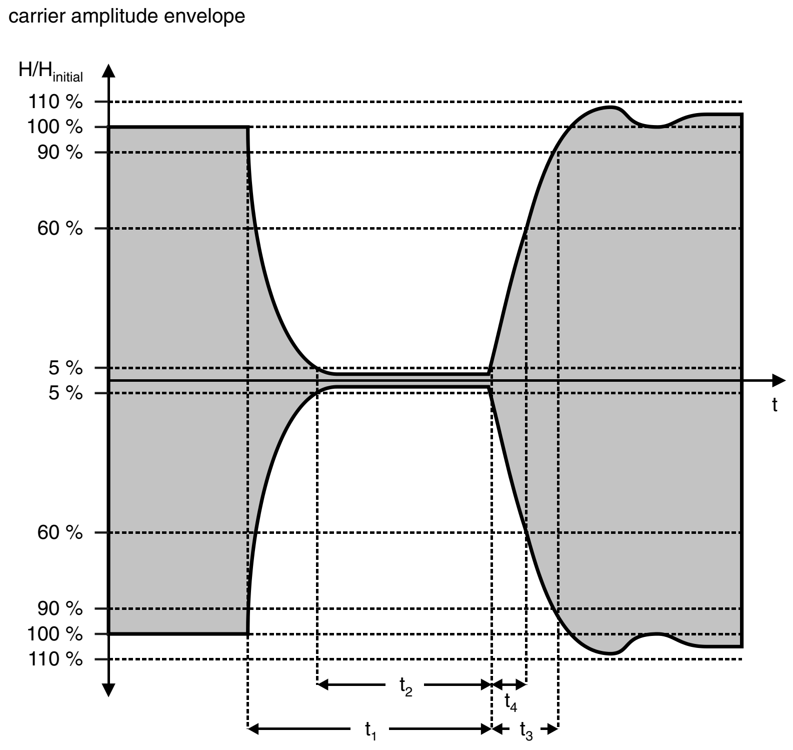


Figure 3.87 Specification of the blanking interval (gap) according to ISO/IEC 14 433-2. The maximum duration of the gap is limited to 3 μs in order to interrupt the energy supply to the card as briefly as possible. Here $2.0\ \mu s \leq t_1 \leq 3.0\ \mu s$; $0.5\ \mu s \leq t_2 \leq t_1$ if $t_1 > 2.5\ \mu s$, or $0.7\ \mu s \leq t_2 \leq t_1$ if $t_1 \leq 2.5\ \mu s$; and $0\ \mu s \leq t_4 \leq 0.4\ \mu s$

3.6.3.1 Type-A communications interface

With Type-A cards, data transmission takes place in both directions at a bit rate of $f_C/128$ ($\approx$106 kbit/s).

Data transmission from the terminal to the card

Digital amplitude modulation (100 % ASK) with modified Miller coding is used for data transmission from the PCD to the card, with the length of the blanking interval (gap) being limited to 3 μs. This relatively short blanking interval makes it easier to provide a steady supply of energy to the card. The exact specification of the length of the blanking interval and its rise and decay characteristics are shown in Figure 3.87.

The card recognizes the end of the pause during interval t_4, which means after the magnetic field has reached 5 % of H_{INITIAL} and before it exceeds 60 % of H_{INITIAL}. Overshoots must be limited to $H_{\text{INITIAL}} \pm 10\,\%$.

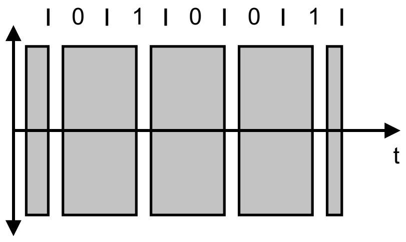


Figure 3.88 Coding of a bit sequence transmitted from the terminal to the card for a Type-A communications interface with 100 % ASK and modified Miller coding at 106 kbit/s. The figure shows the voltage at the terminal aerial

An example of the coding of a bit sequence using modified Miller coding is shown in Figure 3.88. The following coding rules apply here:

• logic 1:	blanking interval after half the bit interval
• logic 0:	no blanking, with the following exceptions: • if there are two or more logic 0 states in succession, there is a blanking interval at the start of the bit interval • if the first bit of a protocol frame is a 0, it is represented by a blanking interval at the start of the bit interval
• start of a message:	blanking interval at the start of a bit interval
• end of a message:	logic 0 followed by one bit with no blanking interval
• no data:	no blanking interval for the duration of at least two bits.

Data transmission from the card to the terminal

The bit rate for data transmission from the card to the terminal is also $f_C/128$ ($\approx$106 kbit/s). Load modulation with a subcarrier is used, which means that the subcarrier is generated by switching a load inside the card. The frequency of the subcarrier is specified to be $f_S = f_C/16$ ($\approx$847 kHz). The subcarrier is modulated by switching the subcarrier on and off (on–off keying, or OOK) using Manchester coding. An example of the coding of a bit sequence is shown in Figure 3.89.

The coding is defined as follows:

• logic 1:	the carrier is modulated by the subcarrier during the first half of the bit interval
• logic 0:	the carrier is modulated by the subcarrier during the second half of the bit interval
• start of a message:	the carrier is modulated by the subcarrier during the first half of the bit interval
• end of a message:	the carrier is not modulated for one-bit interval
• no data:	no subcarrier modulation.

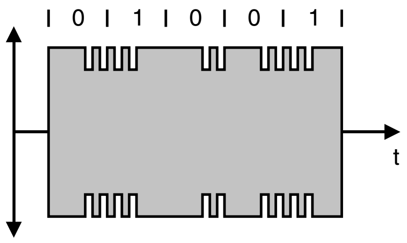


Figure 3.89 Load modulation for data transmission from the card to the terminal using a subcarrier at a frequency of $f_C/16 (\approx 847$ kHz) and Manchester coding with a bit rate of 106 kbit/s and OOK. The figure shows the voltage on the card coil

3.6.3.2 *Type-B communications interface*

Data transmission from the terminal to the card

With Type-B cards, ASK modulation with a modulation index of 10 % (–2 %, +4 %) is used for data transmission from the PCD to the card. In contrast to the Type-A method, in which continuity of the energy supply is assured by very short blanking intervals, with the Type-B method it is assured by the small modulation index, which is defined such that at least 86 % of the carrier field is always available. The bit rate is again $f_C/128$ ($\approx$106 kbit/s). The exact form of Type-B modulation is shown in Figure 3.90.

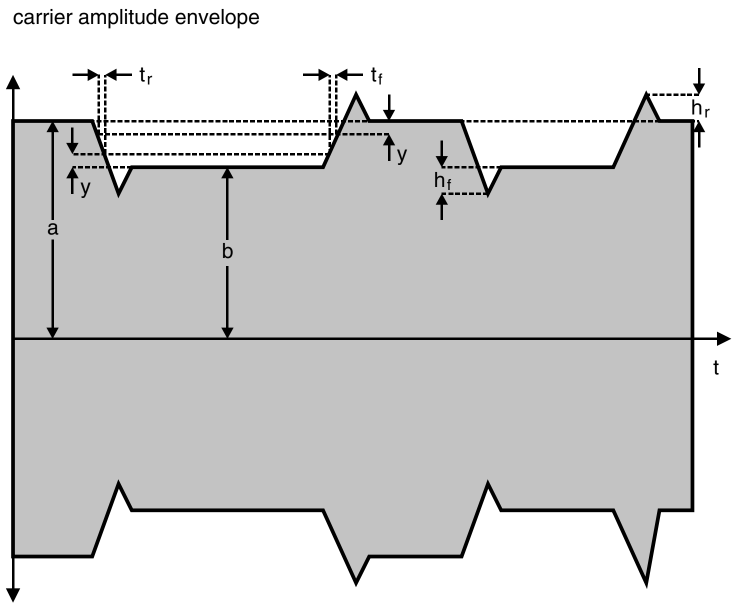


Figure 3.90 Type-B carrier modulation. A continuous supply of energy is made possible by the small modulation index (10 %). Here t_f, $t_r \leq 2$ µs, y $= 0.1 \cdot$ (a – b) and h_f, $h_r \leq 0.1 \cdot$ (a – b)

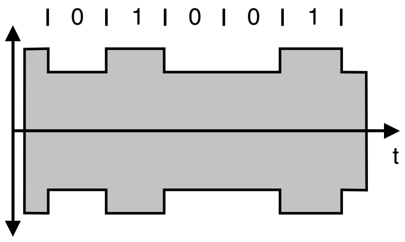


Figure 3.91 Coding of a bit sequence from the terminal to the card for a Type-B communications interface with 10 % ASK, NRZ coding and a bit rate of 106 kbit/s. The figure shows the voltage on the terminal aerial

Simple non-return to zero (NRZ) bit coding is used, with the following coding rules:

- logic 1: high carrier amplitude
- logic 0: low carrier amplitude.

Data transmission from the card to the terminal

With the Type-B method, load modulation with a subcarrier is also used for data transmission from the card to the terminal. The frequency of the subcarrier is again $f_C/16$ ($\approx$847 kHz). In contrast to Type A, the subcarrier is modulated by shifting the phase by 180 degrees (binary phase-shift keying, or BPSK), again using a bit rate of $f_C/128$ ($\approx$106 kbit/s) and NRZ coding. In order to have an unambiguous initial state, the following sequence must be observed at the start of each protocol frame:

- No subcarrier is generated during a guard time interval TRO $> 64/f_S$ following reception of data from the terminal.
- After the guard time, the card generated the subcarrier with no phase shifting for a synchronization time interval TR1 $> 80/f_{S.}$ The phase during this interval is defined to be the reference phase ϕ_0.

The initial phase ϕ_0 is defined to be logic 1, so the first phase shift means a change from a logic 1 to a logic 0. The following rules apply to the remainder of the data transmission:

- logic 1: ϕ_0
- logic 0: $\phi_0 + 180^\circ$

3.6.3.3 Initialization and anticollision (ISO/IEC 14 433-3)

When a proximity card comes within the working range of a terminal, communications between the card and the terminal must first be established. It may happen that the terminal is already communicating with another card, or that several cards are concurrently present

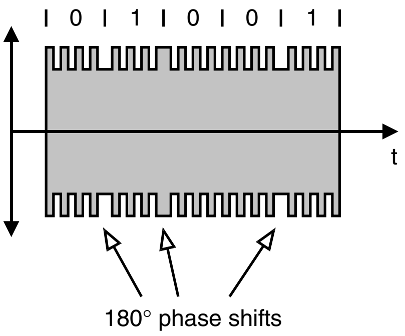


Figure 3.92 Coding of a bit sequence from the card to the terminal for a Type-B communications interface with a subcarrier of $f_C/16$ (≈847 kHz), BPSK, NRZ coding and a bit rate of 106 kbit/s. The figure shows the voltage on the coil in the card

within the working range of the terminal. Means must be provided to allow interference-free communication with a single card or a specific group of cards to occur under such conditions.

Establishing communications between a card and a terminal and the anticollision methods to be used for selecting a individual card are described in Part 3 of ISO/IEC 14 433. Due to the use of different modulation methods, Type-A and Type-B cards also have different protocol frames and anticollision methods.

Type-A initialization and anticollision

A dynamic binary search algorithm is used to initialize and select Type-A cards. With this method, it is necessary for the terminal to be able to recognize a data collision at the bit level. As explained below, the Manchester coding used here makes bitwise collision detection possible (see Figure 3.97). However, this requires all cards within the working range of the terminal to transmit their data synchronously.

If a proximity card comes into the field of a terminal, the microprocessor in the card is supplied with power, and following the power-on reset the card enters the Idle state. In this state, the card is only allowed to respond to a REQA (Request Type-A) command or a WUPA (Wake-up Type-A) command. All other commands transmitted by the terminal for communicating with any other Type-A or Type-B cards already present within the working range of the terminal must be ignored in order to avoid interfering with these communications. The state diagram shown in Figure 3.93 shows all possible states that can be assumed by a Type-A card during the initialization and anticollision phase.

As already mentioned, the card enters the Idle state after the power-on reset. The standard requires the card to enter the Idle state within 5 ms after it receives adequate operating power from the terminal's field. In the Idle state, the card awaits further commands. It changes to the Ready state when it recognizes a REQA or WUPA command, but it ignores all other commands.

In order to ensure a high level of reliability for recognizing the REQA and WUPA commands, they are transferred using special short frames. All other commands except anticollision commands are transmitted using standard frames. Special frames called 'bit-oriented anticollision frames' are defined for the anticollision commands.

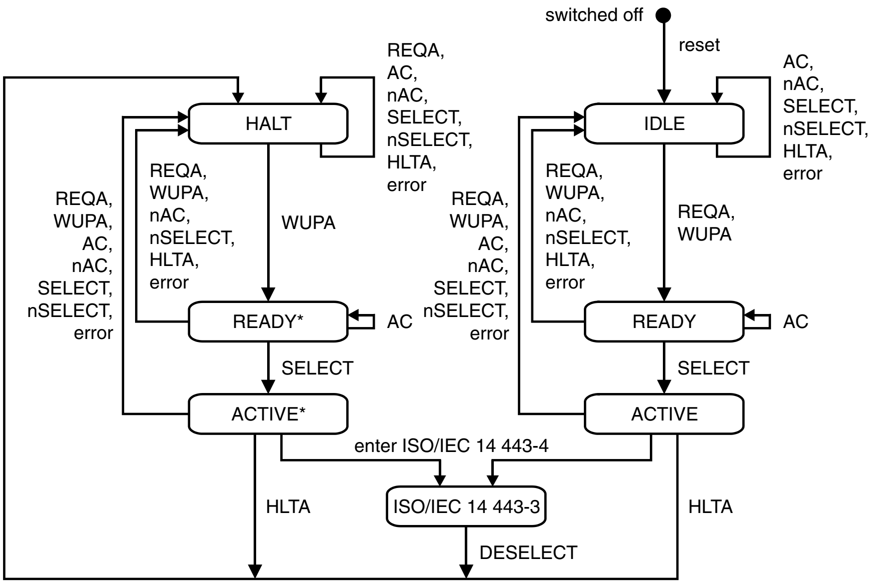

switched off
reset
HALT
IDLE
READY*
READY
ACTIVE*
ACTIVE
ISO/IEC 14 443-3
REQA, AC, nAC, SELECT, nSELECT, HLTA, error
AC, nAC, SELECT, nSELECT, HLTA, error
REQA, WUPA, AC, nAC, SELECT, nSELECT, error
REQA, WUPA, nAC, nSELECT, HLTA, error
WUPA
REQA, WUPA
AC
SELECT
enter ISO/IEC 14 443-4
HLTA
DESELECT

Figure 3.93 State diagram of a Type-A PICC during the initialization and anticollision phase. The abbreviations are explained in the text

Short frames

As already mentioned, short frames are only used for the initialization commands. A short frame consists of nine bits in the following sequence:

- one message start bit
- seven data bits starting with the least significant bit (lsb first)
- one message stop bit.

The coding rules for the start and end bits and the data bits are described in Section 3.6.3.1.

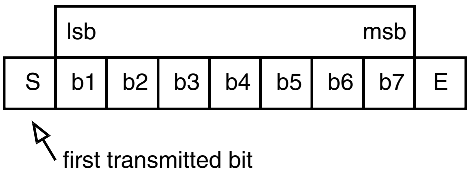


Figure 3.94 Structure of a short frame

Table 3.7 shows the coding of the REQA and WUPA commands, which are the only types of commands transmitted using short frames.

Table 3.7 Coding of the REQA and WUPA commands, which use the short-frame format with seven data bits

b7	b6	b5	b4	b3	b2	b1	Meaning
0	1	0	0	1	1	0	'26' = REQA
1	0	1	0	0	1	0	'52' = WUPA
0	1	1	0	1	0	1	'35' = optional timeslot method
1	0	0	x	x	x	x	'40' – '4F' = proprietary
1	1	1	1	x	x	x	'78' – '7F' = proprietary
			all other values				RFU

The REQA and WUPA commands are transmitted by the terminal to determine whether any cards are present within the working range of the terminal (see Figure 3.93).

Standard frames

Standard frames are used for regular data exchanges. A standard frame consists of:

- start of message
- $n \times$ (8 data bits + odd-parity bit), with $n \geq 1$
- end of message.

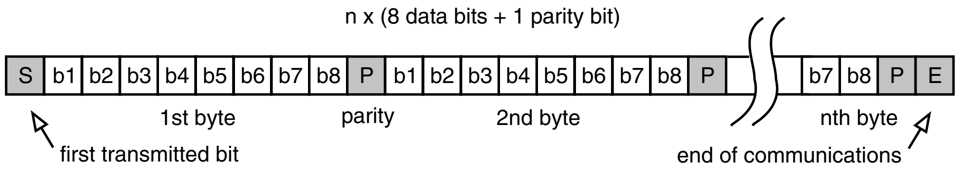


Figure 3.95 Structure of a standard frame

When the card changes to the Ready state, it transmits an Answer to Request, Type A (ATQA) after a precisely defined frame delay time (see Figure 3.98). An ATQA consists of two bytes, and due to the uniquely specified frame delay time, all ATQA messages are sent synchronously by all addressed cards. Figure 3.96 show the coding of the ATQA message.

When the terminal receives an ATQA, it recognizes that at least one card is present within its working range. It then initiates the anticollision procedure, which also allows it to read the Type-A unique identifier (UID), by transmitting a SELECT command. If the terminal is able to determine the complete identifier, it transmits a SELECT command containing this identifier. The card with the corresponding identifier confirms this command by transmitting a SELECT Acknowledge (SAK) message and changes to the Active state. In the Active state, the card can communicate using higher level protocols (such as those defined in ISO/IEC 14 443-4).

The card can be put into the Halt state by transmitting a HLTA command (Halt Command Type A). The card can also be put into the Halt state by means of special commands belonging to

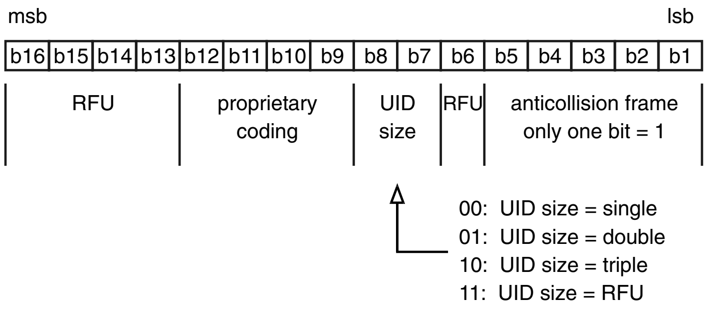


Figure 3.96 Coding of ATQA. All RFU bits must be set to 0. Bits 9–12 can be used to indicate other, non-standardized methods. One of bits b1–b5 must be set to 1. Bits b7 and b8 indicate the size of the UID

higher level protocols. In the Halt state, the card only responds to a WUPA (Wake-Up Type A) command, to which it responds by transmitting an ATQA (Answer to Request, Type A) and changing to the Ready* state. The Ready* state is similar to the Ready state. The conditions for changing to the Active* state are shown in Figure 3.93.

In detail, the procedure used for collision avoidance and determining the identifier works as follows. If two or more cards are concurrently in the Ready state and located within the working range of a terminal, they react simultaneously to a SELECT command from the terminal by each transmitting a portion of their different identifiers. This is done using a special bit-oriented frame, which allows the direction of data transmission between the terminal and the cards to be reversed after an arbitrary number of data bits have been transmitted. If several cards transmitting different data are present, the terminal will receive the data superimposed on each other, and it can detect a collision by the fact that this superimposition will cause the carrier to be modulated by the subcarrier for the full duration of one or more of the bit intervals. This is an irregular state, since the Manchester coding used requires a pulse edge to always occur within each bit interval. Figure 3.97 illustrates how this irregular state is produced.

In order for the terminal to be able to detect a collision at the bit level, all cards in the Ready state that are located within the working range of the terminal must respond to an ANTICOLLISION command at exactly the same time. To ensure this, the timing requirements imposed on the terminal and the card for exchanging frames are precisely specified in ISO/IEC 14 433-3.

Frame delay time (FDT)

The time between the end of the final pause transmitted by the terminal at the end of a message and the leading edge of the modulation pulse for the start bit transmitted by the card is designated the ‘frame delay time PCD to PICC’, which is abbreviated as FDT. This interval is defined in Figure 3.98. There are two different cases, depending on whether the final data bit transmitted by the terminal is a logic 1 or a logic 0.

For the REQA/WUPA, ANTICOLLISION and SELECT commands, the value of n is set to 9, which means that FDT is $1236/f_C$ or $1172/f_C$. This causes all cards within the working range of the terminal to respond synchronously to these commands, which are used in the

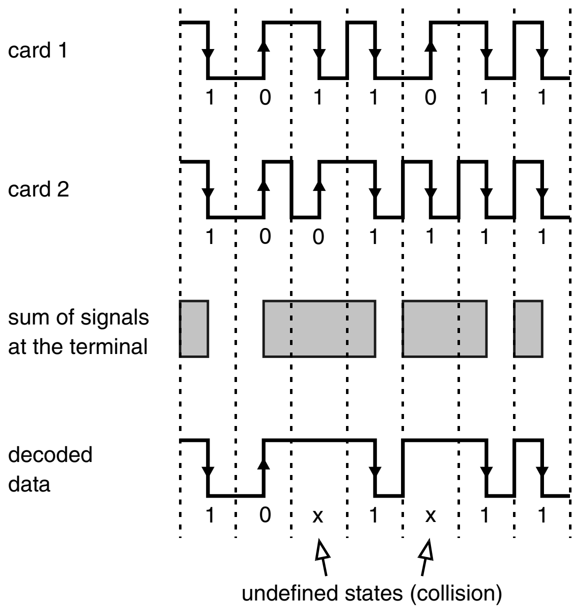


Figure 3.97 Collision of two bit sequences with Manchester coding (Type A). With an interference-free transmission, the carrier is always modulated by the subcarrier during only one half of each bit interval. If different bits are superimposed, modulation is present for the entire duration of the bit interval, allowing the terminal to detect a collision

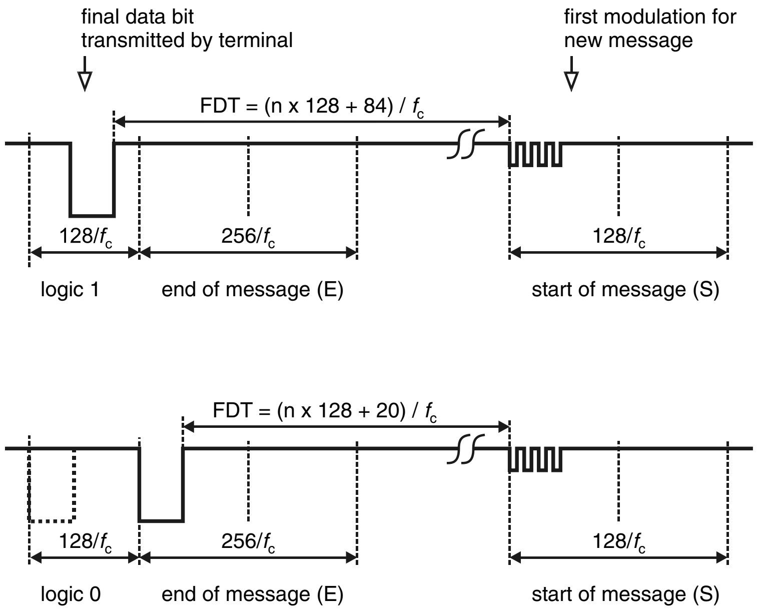


Figure 3.98 Frame delay time PCD to PICC (FDT)

anticollision loop. This makes it possible to detect collisions at the bit level. For all other commands, $n \geq 9$.

The time between the final modulation pulse transmitted by the card and the first pause signal transmitted by the terminal is designated the 'frame delay time PICC to PCD'. This time is not significant for collision detection at the bit level. Consequently, it is not precisely specified in the standard. The only requirement is that it must be equal to or greater than $1172/f_C$.

In addition, the minimum time between two successive REQA commands is specified to be $7000/f_C$. This time is designated the 'request guard time'.

As can be seen in the following sections, if there are several PICCs within the working range of the terminal, it may be necessary for the terminal to execute the anticollision loop many times before it can address a specific card. In order to avoid spending an excessive amount of time executing this loop, a special bit-oriented frame is used for the anticollision (AC) command, which is a special form of the SELECT command. This type of frame, which is called a 'bit-oriented anticollision frame', is described below.

Bit-oriented anticollision frame

If the terminal (PCD) detects a collision, it responds by transmitting bit-oriented anticollision frames, which have the same structure as standard seven-byte frames but are divided into two parts. The first part is used to transfer data from the PCD to the PICC, while the second part is used to transfer data in the opposite direction, from the PICC to the PCD. The relative lengths of the two parts vary in successive executions of the anticollision loop. The sum of the number of data bits in the two parts is always 56. The following rules apply to the lengths of the two parts:

- Rule 1: the total number of data bits is 56.
- Rule 2: the minimum length of Part 1 is 16 data bits.
- Rule 3: the maximum length of Part 1 is 55 data bits.

Consequently, Part 2 has a minimum length of one data bit and a maximum length of 40 data bits.

The frame can be divided between the two parts at any desired location, which means that the division may also be located within a data byte. In this case, no parity bit is appended to the first part of the divided byte, and the parity bit of the second part of the byte is ignored by the PCD. Two examples of anticollision frames are shown in Figure 3.99.

Commands used in the anticollision loop

As already indicated by the state diagram in Figure 3.93, the following commands may be used during initialization and in the anticollision loop:

- REQA (Request command, Type A)
- WUPA (Wake-Up command, Type A)

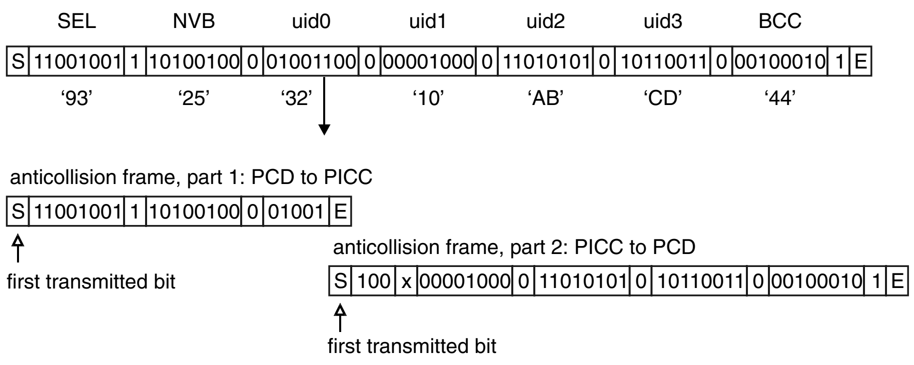

SEL
NVB
uid0
uid1
uid2
uid3
BCC
S 11001001 1 10100100 0 01001100 0 00001000 0 11010101 0 10110011 0 00100010 1 E
'93'
'25'
'32'
'10'
'AB'
'CD'
'44'
anticollision frame, part 1: PCD to PICC
S 11001001 1 10100100 0 01001 E
first transmitted bit
anticollision frame, part 2: PICC to PCD
S 100 x 00001000 0 11010101 0 10110011 0 00100010 1 E
first transmitted bit

Figure 3.99 Two examples of bit-oriented anticollision frames. In the first example, the frame is divided after a full byte, while in the second example it is divided after the fifth bit of a data byte

- ANTICOLLISION
- SELECT
- HLTA (Halt command, Type A).

The ANTICOLLISION and SELECT commands

The ANTICOLLISION and SELECT commands are used in the anticollision loop. As can be seen in Figure 3.99, these commands consist of the following data fields:

- SEL (select code; 1 byte)
- NVB (number of valid bits; 1 byte)
- 0–40 data bits for the identifier (UID), depending on the value of NVB.

The unique identifier (UID) may consist of 4 bytes (single size), 7 bytes (double size) or 10 bytes (triple size).

Table 3.8 Possible sizes for the unique identifier (UID)

Cascade level	UID size	Number of UID bytes
1	Single	4
2	Double	7
3	Triple	10

The UID may be a fixed number or a random number generated by the card. With double- and triple-size identifiers, a cascade tag is transmitted as the first byte for cascade levels 1 and 2. The cascade tag is coded as '88'. In a single-size UID, the first byte is not allowed to have the value '88'.

Table 3.9 Coding of the SEL, which shows the current cascade level of the terminal

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	0	0	1	0	0	1	1	'93': cascade level 1 selected
1	0	0	1	0	1	0	1	'95': cascade level 2 selected
1	0	0	1	0	1	1	1	'97': cascade level 3 selected
1	0	0	1	all values other than those listed above				RFU

At most four data bytes can be transmitted from the PCD within an ANTICOLLISION or SELECT command (see Figure 3.99). If the card has a double- or triple-size identifier, it indicates this fact to the terminal in the SAK (Select Acknowledge) by setting the cascade bit, and it remains in the Ready state. The terminal will then start the anticollision procedure anew in order to ascertain bytes 5 though 7 of the UID. With a triple-size identifier, the anticollision procedure must be executed a third time to ascertain bytes 8 through 10. The terminal uses the SELECT command to inform the card of its current cascade level, in order to indicate which part of the UID is being requested (see Table 3.9).

This process is illustrated in Figure 3.100 in the form of a flow chart.

If the NVB byte does not specify 40 bits, the command is an ANTICOLLISION command and the card remains in the Ready or Ready* state. Once the terminal has ascertained the full UID, it sends a SELECT command with NVB = '70', which means that 40 data bits are specified. A CRC_A checksum formed in accordance with ISO/IEC 13 239 is appended to this command. The computation of this checksum is described in the informational Annex A of ISO/IEC 14 433-3 by means of an example. If the card is addressed using a SELECT command with its full identifier, it changes from the Ready state to the Active state, or from the Ready* state to the Active* state (see Figure 3.93), and then transmits SAK (Select Acknowledge) to indicate that the UID is complete. The codings of NVB (number of valid bits) and SAK are shown in Tables 3.10 and 3.11.

The procedure executed in the anticollision loop is shown in Figure 3.102 in the form of a flow chart for the terminal. This loop must be executed at every cascade level as long as the full UID is not known to the terminal.

To illustrate the selection process, Figure 3.103 shows an example in which two cards are within range of the terminal. In this example, card 1 (PICC 1) has a single-size UID with a value of '10' for uid0, while card 2 (PICC 2) has a double-size UID.

The terminal initiates the selection process by transmitting a REQA command. All cards within range of the terminal respond to this command. In this example, PICC 1 responds with an ATQA in which an anticollision bit frame is indicated by bit b1 being set and a single-size UID is indicated by bits b7 and b8 being cleared. PICC 2 also indicates an anticollision bit frame by setting bit b1 in its ATQA and indicates that it has a double-size UID by setting bit b7.

In the next step, the actual anticollision loop starts at cascade level 1. The terminal transmits an ANTICOLLISION command with a Select code of '93', which indicates an anticollision frame for cascade level 1. The value of '20' for NVB (number of valid bits) means that the terminal is not sending any portion of the cascade level 1 UID. Each card within range of

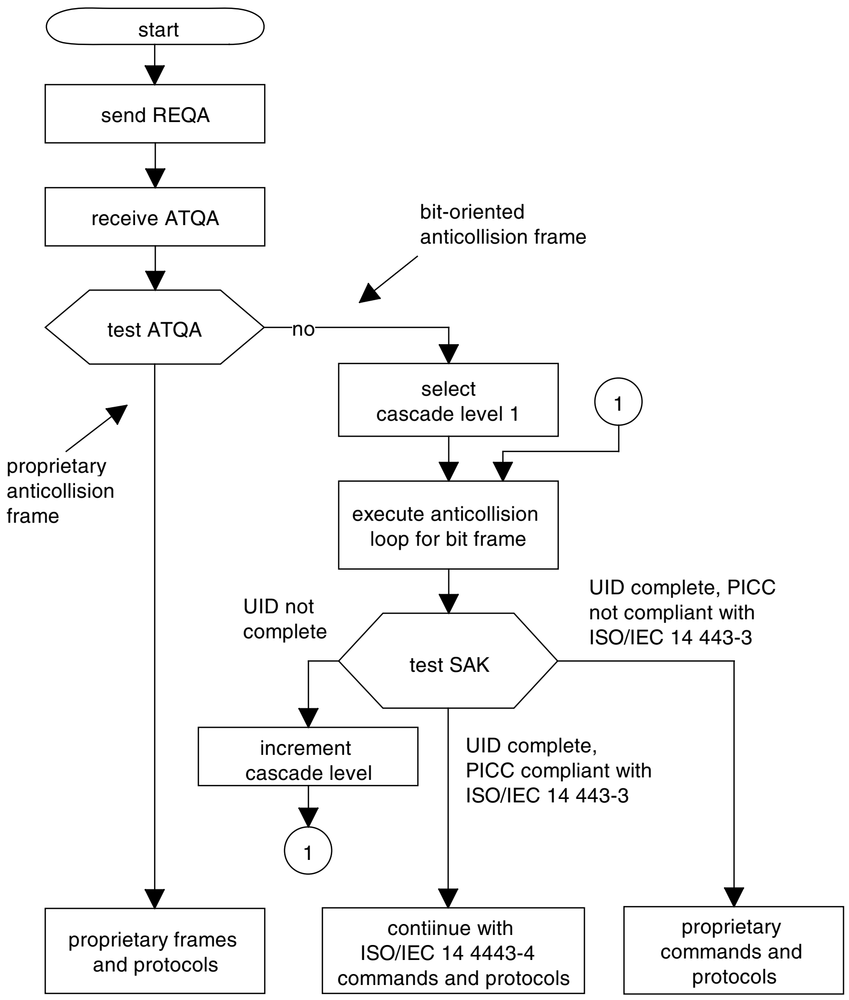

start
send REQA
receive ATQA
bit-oriented anticollision frame
test ATQA
no
select cascade level 1
1
proprietary anticollision frame
execute anticollision loop for bit frame
UID complete, PICC not compliant with ISO/IEC 14 443-3
UID not complete
test SAK
increment cascade level
UID complete, PICC compliant with ISO/IEC 14 443-3
1
proprietary frames and protocols
contiinue with ISO/IEC 14 4443-4 commands and protocols
proprietary commands and protocols

Figure 3.100 Flow chart for the process of selecting a particular card. After the terminal has transmitted a SELECT command, all cards in the Idle state respond simultaneously with ATQA

1st byte	2nd & 3rd bytes
SAK	CRC_A

msb lsb msb lsb

Figure 3.101 Format of SAK (Select Acknowledge). SAK is transmitted by the card when NVB indicates 40 data bits and these bits match the UID of the card

Table 3.10 Coding of NVB (number of valid bits). The upper four bits are called the ‘byte count’ and indicate the number of complete data bytes (including SEL and NVB) transmitted by the terminal. The lower four bits are called the ‘bit count’ and indicate the number of bits of any incomplete byte transmitted by the terminal

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	0	1	0	x	x	x	x	byte count = 2
0	0	1	1	x	x	x	x	byte count = 3
0	1	0	0	x	x	x	x	byte count = 4
0	1	0	1	x	x	x	x	byte count = 5
0	1	1	0	x	x	x	x	byte count = 6
0	1	1	1	x	x	x	x	byte count = 7
0	1	0	0	x	x	x	x	byte count = 8
x	x	x	x	0	0	0	0	bit count = 0
x	x	x	x	0	0	0	1	bit count = 1
x	x	x	x	0	0	1	0	bit count = 2
x	x	x	x	0	0	1	1	bit count = 3
x	x	x	x	0	1	0	0	bit count = 4
x	x	x	x	0	1	0	1	bit count = 5
x	x	x	x	0	1	1	0	bit count = 6
x	x	x	x	0	1	1	1	bit count = 7

Table 3.11 Coding of SAK (Select Acknowledge)

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
x	x	x	x	x	1	x	x	Cascade bit set:: UID incomplete
x	x	1	x	x	0	x	x	UID complete, PICC complies with ISO/IEC 14 443-4
x	x	0	x	x	0	x	x	UID complete, PICC does not comply with ISO/IEC 14 443-4

the terminal thus responds with its full cascade level 1 UID. The first bit collision occurs at bit b4. The terminal recognizes this bit collision and sends a new ANTICOLLISION command, this time containing the first three bits of the CL1 UID, which were received without errors, followed by a ˚1˚, and consequently with a value of '24' assigned to NVB. The first four bits now match the first four bits of the CL1 UID of card 2, but not the first four bits of the CL1 UID of card 1. Consequently, only card 2 responds with the remaining 36 bits of its CL1 UID. Since the terminal now knows the complete CL1 UID of card 2, it transmits a SELECT command for card 2, which responds with a SAK (Select Acknowledge) with the cascade bit (b3) set. From this, the terminal recognizes that the UID is not yet complete, so it increments the cascade level.

The terminal now sends a new ANTICOLLISION command with a Select code (SEL) indicating an anticollision bit frame and cascade level 2. NVB is set to '20' in order to request

Table 3.12 The steps of the anticollision algorithm. The numbering corresponds to Figure 3.102

Step 1	The terminal specifies the select code (SEL) for the cascade level.
Step 2	The terminal sets the NVB byte (number of valid bits) to '20'. This indicates that the terminal will not transmit any part of the UID, which means that every card within the working range of the terminal will transmit its complete CL*n* UID (the portion of the UID for cascade level *n*).
Step 3	The terminal transmits an ANTICOLLISION command containing SEL and NVB.
Step 4	All cards within range of the terminal respond with their CL*n* UIDs.
Step 5	If several cards with different identifiers are within range of the terminal, a collision will occur. If no collision occurs, steps 6 through 10 are skipped.
Step 6	The terminal determines the bit position of the first collision.
Step 7	The terminal assigns a value to NVB that indicates the number of valid bits of the CL*n* UID. The valid bits are the portion of the CL*n* UID that was received before the collision occurred, followed by a ˚0˚ or ˚1˚.
Step 8	The terminal transmits an ANTICOLLISION command containing SEL, NVB and the valid bits.
Step 9	All cards whose corresponding portion of their CL*n* UID matches the valid bits transmitted by the terminal transmit the remaining bits of their CL*n* UIDs.
Step 10	If a collision occurs again, steps 6 through 9 are repeated.
Step 11	When no more collisions occur, the terminal sets NVB to '70'. This means that the terminal will transmit the complete CL*n* UID.
Step 12	The terminal transmits a SELECT command containing SEL, NVB and the complete CL*n* UID, followed by the CRC_A checksum.
Step 13	The card whose identifier matches the CL*n* UID responds with SAK.
Step 14	If the UID is complete, the card transmits a SAK with the cascade bit cleared and changes from the Ready state to the Active state, or from the Ready* state to the Active* state.
Step 15	The terminal checks whether the cascade bit is set in order to decide whether an additional anticollision loop must be executed at a higher cascade level.

the complete CL2 UID from the card. Card 2 responds to this command with all 40 bits of its CL2 UID. The terminal can now transmit a SELECT command containing all of the bits of the CL2 UID, to which the card responds with a SAK in which cascade bit b3 is not set, indicating that its UID is complete. Card 2 now changes from the Ready state to the Active state, in which it can receive commands for higher level protocols.

Type-B initialization and anticollision

Type-B proximity cards use a 'dynamic slotted ALOHA procedure' for selection. In this procedure, cards within the working range of a terminal transmit their data to the terminal in predefined time slots. The time slots are selected randomly, and the number of slots can be determined by the terminal. If only a few slots are provided and significantly more cards are present in the working range of the terminal than the number of available slots, the probability that a card can transmit its data without interference is small. In such situations, interference-free data transmission can generally only occur if each time slot is used by only one card, or in

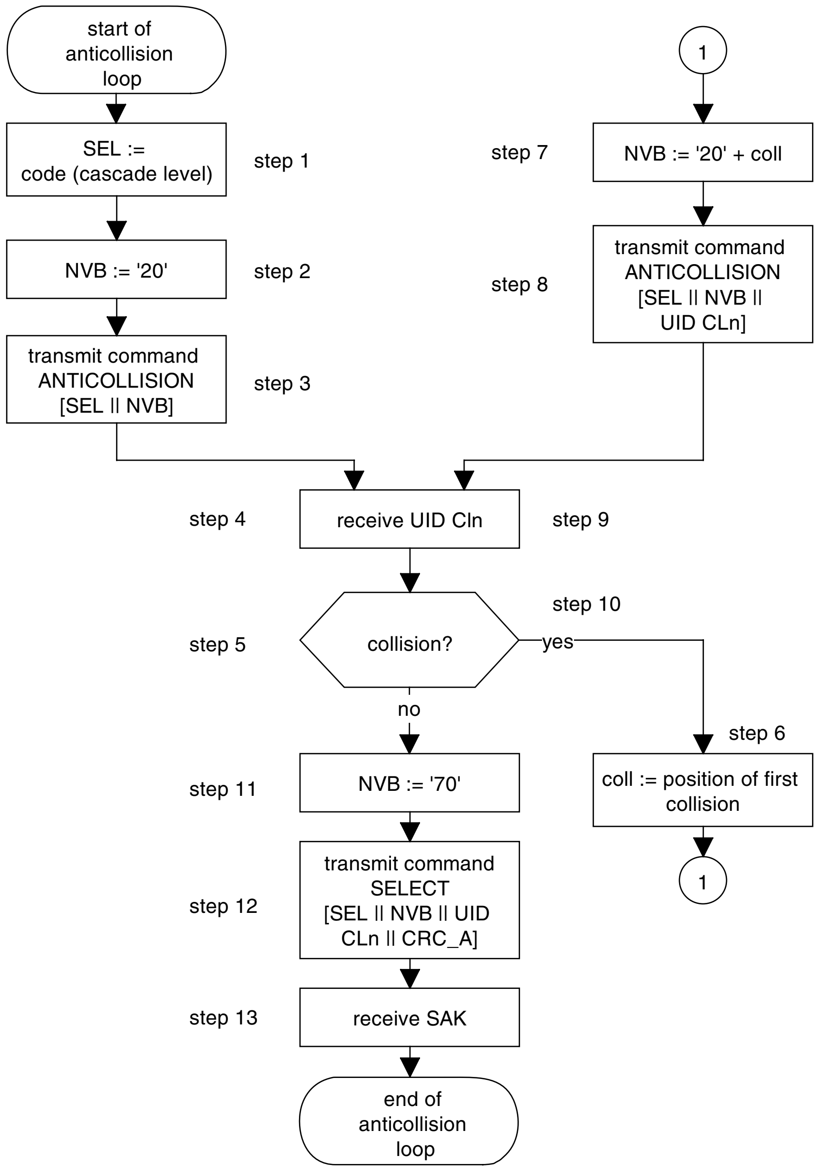

start of anticollision loop
SEL := code (cascade level)
step 1
NVB := '20'
step 2
transmit command ANTICOLLISION [SEL || NVB]
step 3
1
step 7
NVB := '20' + coll
step 8
transmit command ANTICOLLISION [SEL || NVB || UID CLn]
step 4
receive UID Cln
step 9
step 10
step 5
collision?
yes
no
step 6
coll := position of first collision
1
step 11
NVB := '70'
step 12
transmit command SELECT [SEL || NVB || UID CLn || CRC_A]
step 13
receive SAK
end of anticollision loop

Figure 3.102 Flow chart of the anticollision loop as seen by the terminal. The step numbers ('Step 1', Step 2' etc.) refer to the algorithm steps listed in Table 3.12

the event that a slot is used by more than one card, if one of these cards is significantly closer to the terminal than the others, so the signal levels for its data prevail at the terminal. Although increasing the number of slots increases the probability of interference-free transmission, it has the disadvantage of increasing the amount of time required to execute the polling loop,

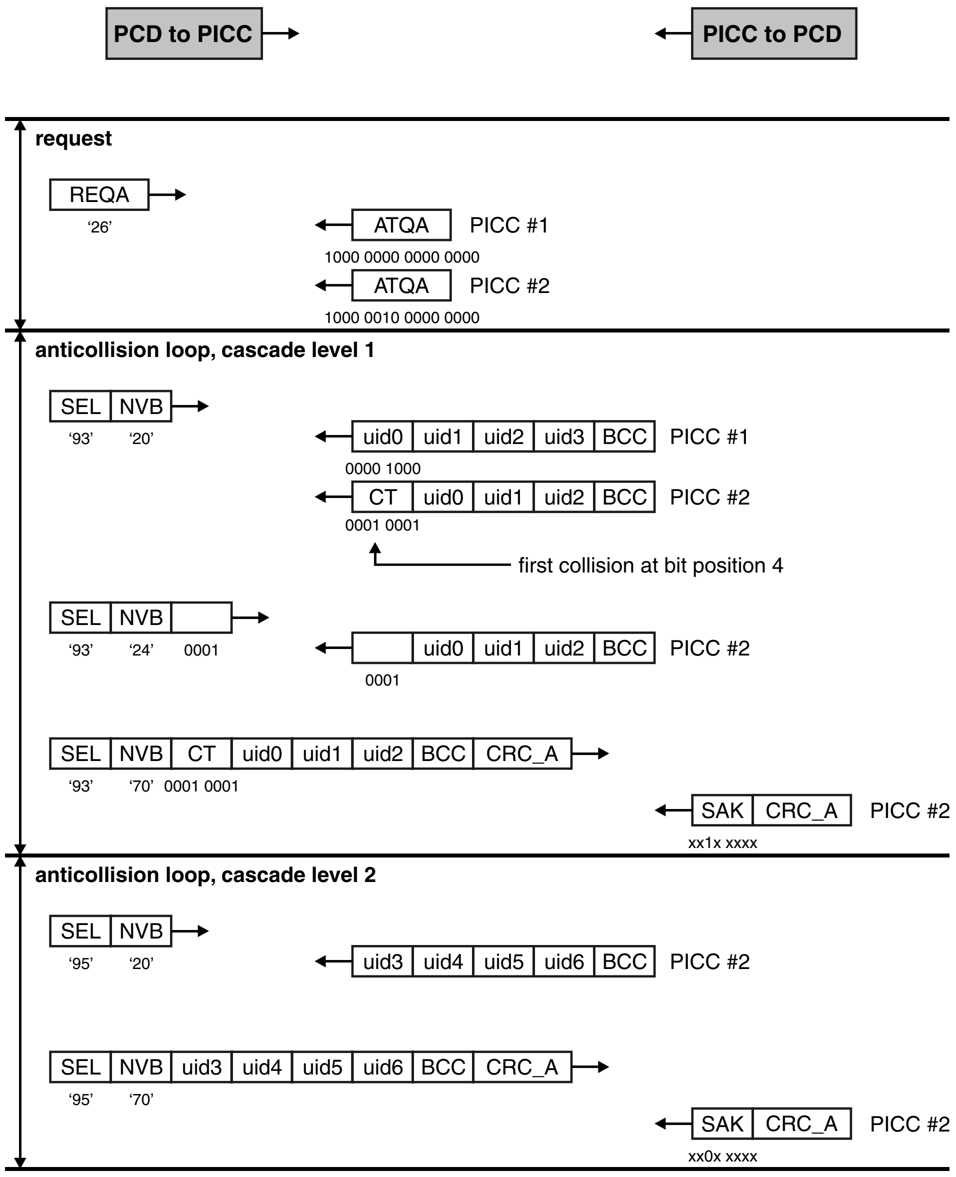


Figure 3.103 Sample initialization and anticollision sequence for Type-A cards. For the sake of simplicity, only the essential elements of the communications are shown

since each card must wait for the duration of all of the time slots provided for cards that might be within range of the terminal.

To make it easier to understand this procedure, which is specified in ISO/IEC 14 433-3, we will first explain it using a simplified example before describing it in detail. The terminal needs two commands to execute this algorithm:

• REQUEST	This command causes every card within the working range of the terminal to transmit its identifier in a subsequent time slot. In our example, four time slots will be provided.
• SELECT	This command transmits a previously determined identifier to cards within the range of the terminal, causing the card with the matching identifier to be activated. Cards having other numbers remain passive and respond only to REQUEST or SELECT commands with matching identifiers.

When the terminal is in the operational state, it periodically transmits REQUEST commands. Here we assume that six cards having identifiers 1 through 6 are concurrently brought within the working range of the terminal. All six cards recognize the next REQUEST command, and they select time slots at random for transmitting their identifiers to the terminal (see Figure 3.104). In this example, collisions occur in time slots 1 and 3, while identifiers 2 and 3 are transmitted without interference in time slots 2 and 4. The terminal can now select one of these two cards using a SELECT command, and then communicate with the selected card without any further collisions.

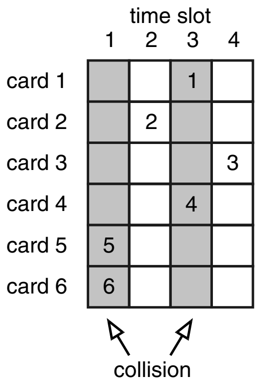


Figure 3.104 Example anticollision process for Type-B contactless smart cards. Collisions occur in time slots 1 and 3, so only identifiers 2 and 3 are transmitted without interference. This example is explained in detail in the text

When communication with the selected card is terminated, the terminal can search for other cards by again transmitting REQUEST commands. If it does not receive any error-free identifiers on the first attempt, it repeats the REQUEST command until it receives a valid identifier.

Now that the basic principle of this anticollision procedure has been explained, we can give our detailed attention to the commands and the timing behavior of the cards and the terminal, as specified in ISO/IEC 14 433-3 for Type-B cards. This standard defines a set of commands that allow various types of anticollision loops to be implemented. This gives users a certain amount of freedom for system optimization.

Formats and timing specifications for characters and frames

The terminal and the cards transmit data bytes in the form of characters, with several characters grouped into frames. For error detection, a 2-byte CRC checksum (CRC_B) computed according to ISOIEC 13 239 is appended to the characters in the frame. An example of the computation of CRC_B is given in Annex B of ISOIEC 14 443-3.

Each character consists of one start bit (logic 0) followed by eight data bits (with the least significant bit first) and one stop bit (logic 1).

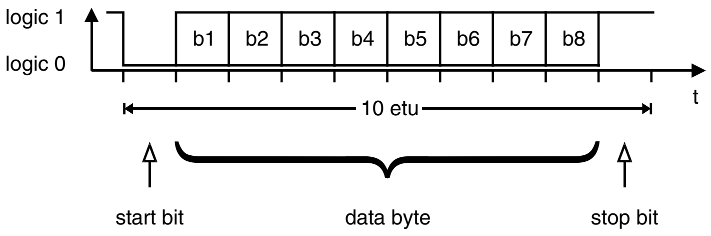


Figure 3.105 Character format

Each pair of characters is separated by a gap called the 'extra guard time' (EGT), which allows the transmitter and receiver to prepare for the next character. For data transmission to the card, this guard time ranges from 0 to 57 μs, while in the opposite direction it ranges from 0 to 19 μs.

Several characters are grouped together to form frames for transmission in each direction. A frame starts with a start of frame (SOF) character, followed by the characters to be transmitted and ending with an end of frame (EOF) character. The SOF character consists of a single falling edge followed by 10 etu of logic 0, with a rising edge in the eleventh etu, followed by a logic 1 for at least 2 and at most 3 etu.

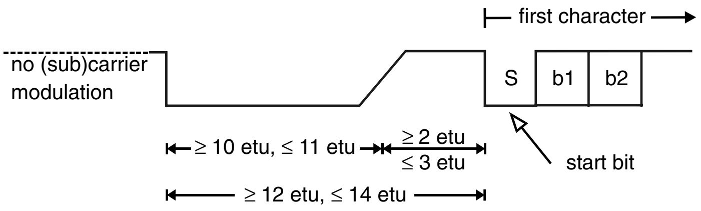


Figure 3.106 Timing of the start of frame (SOF) character

The EOF character also begins with a falling edge, followed by 10 etu of logic 0 and a rising edge within the eleventh etu.

To prevent the transmission protocol from 'hanging' in the event of an error, and to provide the card with defined minimum and maximum times for its internal activities, the times between two frames transmitted in opposite directions are specified in the standard. After the card has recognized the EOF of a frame transmitted by the terminal, it waits for the duration of the guard time (TR0) before generating the unmodulated subcarrier. After waiting for the duration of the synchronization time (TR1), the card starts to modulate the subcarrier. The minimum

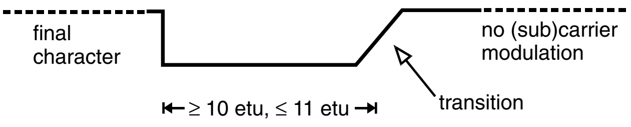


Figure 3.107 Timing of the end of frame (EOF) character

value for TR0 is $64/f_S$, while the maximum value in an anticollision loop (for an ATQB) is $256/f_S$. For all other types of frames, the maximum value is calculated using the formula

$$\mathrm{TR0}_{\max} = (256/f_S) \times 2^{\mathrm{FWI}}$$

The value of FWI ranges from 0 to 14 and is supplied to the terminal in the ATQB. The minimum value of the synchronization time (TR1) is $80/f_S$, and the maximum value is $200/f_S$.

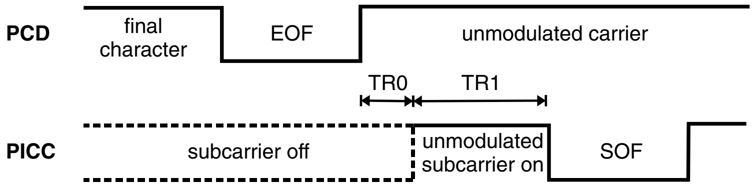


Figure 3.108 Definition of the guard time (TR0) and synchronization time (TR1)

Once the terminal recognizes the end of a frame transmitted by the card, it waits for an interval of at least $32/f_S$ before starting to send a new frame. During this interval, the card switches off the subcarrier within 2 etu of the end of the transmitted frame.

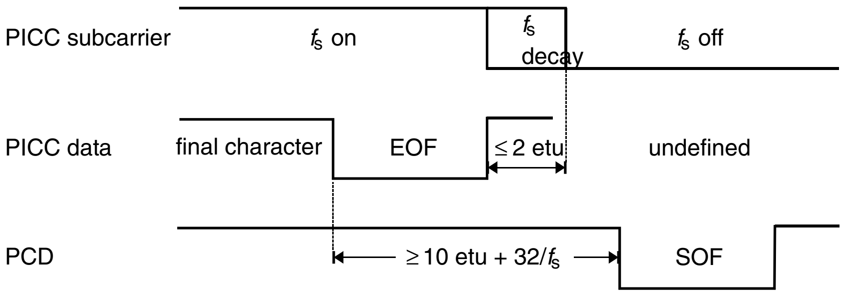


Figure 3.109 Definition of the waiting time between a frame transmitted by the card and the following frame transmitted by the terminal. The card switches off the subcarrier only after the end of the EOF character

Card selection procedure

If a Type-B card enters the working range of a terminal, the microprocessor in the card is supplied with power and enters the Idle state following the power-on reset. Just like Type-A

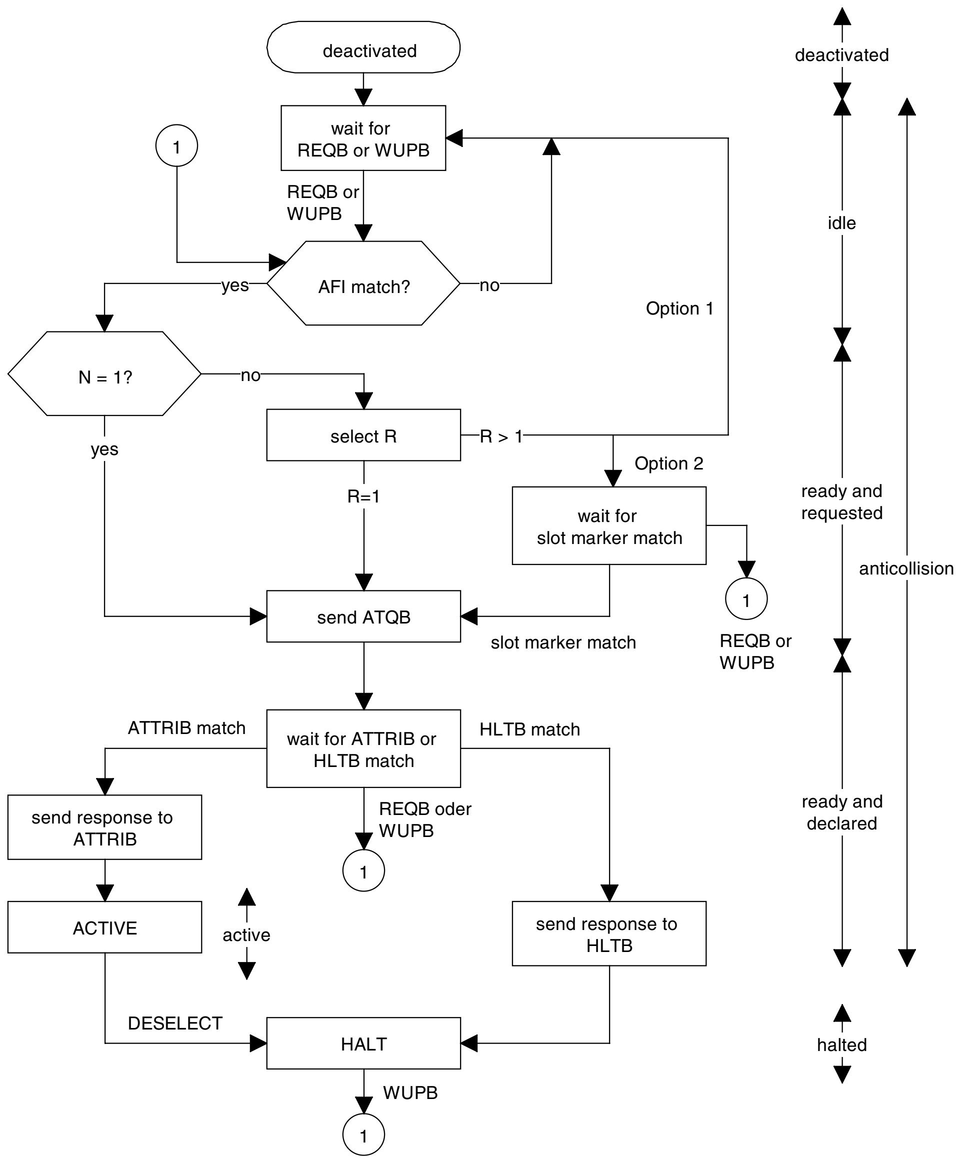


Figure 3.110 State diagram of a Type-B smart card according to ISO/IEC 14 443

cards, Type-B cards must enter the Idle state within 5 ms of receiving sufficient operating power from the field of the terminal. In the Idle state, the card is not allowed to send any frames. Instead, it must wait to receive a valid REQB or WUPB command. These commands contain a parameter indicating the number of time slots used by the terminal, along with an application family identifier (AFI) that can be used to prescribe a specific application group.

This produces a type of preselection, since only those cards whose applications belong to the indicated 'application family' will respond.

The REQB / WUPB command

The terminal transmits the REQB / WUPB command to determine whether any Type-B cards are present within its working range. The WUPB command is specifically used to wake up any cards that may be in the Halt state. A REQB / WUPB command consists of an anticollision prefix (APf) with a value of '05', followed by the AFI byte, a parameter byte (PARAM) indicating the number of available time slots and two CRC-B bytes.

APf	**AFI**	**PARAM**	**CRC_B**
1 byte	1 byte	1 byte	2 bytes

Figure 3.111 Format of the REQB/WUPB command

Table 3.13 Coding of the Application Family Identifier (AFI) byte

AFI upper nibble	AFI lower nibble	Meaning	Comments and sample applications
'0'	'0'	All families and subfamilies	No application preselection
X	'0'	All subfamilies of family X	Application preselection
X	Y	Only subfamily Y of family X	
'0'	Y	Only proprietary subfamily Y	
'1'	'0', Y	Transportation	Local public transport, busses, airlines, etc.
'2'	'0', Y	Financial sector	Electronic purses, banks, retail trade etc.
'3'	'0', Y	Identification	Access control, etc.
'4'	'0', Y	Telecommunications	Public telephone network, GSM, etc.
'5'	'0', Y	Medical	
'6'	'0', Y	Multimedia	Internet services, etc.
'7'	'0', Y	Games	
'8'	'0', Y	Data storage	Portable files, etc.
'9' – 'F'	'0', Y	RFU	

After a card has received a valid REQB command, it checks to see whether it supports the applications identified by the AFI. If it does, it evaluates the PARAM byte in order to obtain the value of N, which specifies the number of available slots. The card is now in the Ready Requested state.

If $N = 1$, the card transmits an ATQB (Answer to Request, Type B) and switches to the Ready Declared state. If $N > 1$, the card generates a random number R with a uniform distribution over the range of 1 to N. If $R = 1$, the card transmits an ATQB (Answer to

Table 3.14 Coding of the PARAM byte. Bit b4 = 0 marks a REQB command (all cards in the Idle or Ready states respond to this command). Bit b4 = 1 marks a WUPB command (all cards in the Idle, Ready or Halt states respond to this command)

b8	b7	b6	b5	b4	b3	b2	b1
RFU				REQB/WUPB	*N* (number of slots)		

Table 3.15 Coding of *N* (number of slots). The probability that a card responds in the first time slot is $1/N$. If only one time slot is used, the probability that a card responds in this slot depends on the value of *N*

b3	b2	b1	N
0	0	0	$2^0 = 1$
0	0	1	$2^1 = 2$
0	1	0	$2^2 = 4$
0	1	1	$2^3 = 8$
1	0	0	$2^4 = 16$
1	0	1	RFU
1	1	X	RFU

Request, Type B) and switches to the Ready Declared state. If $R > 1$, two different options are provided in the standard in order to support two different algorithms:

Option 1: This option is for cards that do not support selecting specific time slots. At this point, the card returns to the Idle state. It cycles through this loop repeatedly until $R = 1$ occurs by chance ('probabilistic approach') or the terminal sets the value of N to 1. This option does not actually use a time-slot method in the true sense, since only one time slot is used. This option is easy to implement and adequate for systems in which only a few cards are concurrently present within the working range of the terminal.

Option 2: This option is for cards that support time slot selection. In this case, the card waits until it receives a Slot Marker command with a matching time slot number (slot number = R) before transmitting an ATQB and switching to the Ready Declared state.

The Slot Marker command

The terminal sends a Slot Marker command at the start of each time slot. The format of this command is shown in Figure 3.112.

The coding of the 'anticollision prefix byte' (APn) is APn = 'n5', where n is the number of the following slot. Slot Marker commands may be transmitted in any desired order; they do not have to be transmitted in increasing order of slot number.

After the Slot Number command has been transmitted, the terminal waits for an interval of $TR0_{max} = 256/f_S$ before checking to see whether a card has started to transmit an ATQB.

AP*n*	**CRC_B**
1 byte	2 bytes

Figure 3.112 Format of the Slot Marker command

Table 3.16 Coding of the slot number in anticollision prefix byte *n* (AP*n*)

n	Slot number
0001	2
0010	3
0011	4
...	...
1110	15
1111	16

If the terminal does not detect an ATQB, it can immediately transmit the next Slot Marker command.

ATQB (Answer to Request, Type B)

The format of ATQB, which is sent in response a REQB/WUPB or Slot Marker command, is shown in Figure 3.113.

APa	**PUPI**	**Application Data**	**Protocol Info**	**CRC_B**
'50'	4 bytes	4 bytes	3 bytes	2 bytes

Figure 3.113 Format of ATQB (Answer to Request, Type B)

ATQB contains information regarding important parameters of the smart card, which the terminal needs in order to select the card. The pseudo-unique PICC identifier (PUPI) is the identification number of the PICC for the anticollision loop. This may be a number that is permanently assigned to the card, or it may be a random number generated by the card following power-on reset.

The Application Data field contains information about the applications present in the card. This information allows the terminal to select the desired card if several cards are present within its working range. The meaning of the application data parameter depends on the content of the application data coding (ADC) parameter in the Protocol Info field (described below), which specifies whether the 'CRC_B compressing method' or proprietary coding is used.

If the CRC_B compressing method is used, the Application Data field is formatted as shown in Figure 3.114.

The Protocol Info field shows important parameters supported by the card. These parameters allow the terminal to optimally adapt itself to the performance capacity of the card for the

AFI	CRC_B (AID)	Number of applications
1 byte	2 bytes	1 byte

Figure 3.114 Format of the Application Data field. The coding of the AFI parameter is given in Table 3.13. It indicates the family of the application for a multiapplication card. CRC_B (AID) is the ISO/IEC 7816-5-compliant CRC_B checksum of the application identifier (AID) of an application in the card corresponding to the AFI sent in the REQB / WUPB command. The Number of Applications field indicates the number of additional applications present in the card. The upper nibble of this byte indicates the number of applications corresponding to the AFI, where '0' means ‘no applications’ and 'F' means ‘15 or more applications’. The lower nibble indicates the total number of applications present in the card, with the same meaning ('0' = ‘no applications’, 'F' = ‘15 or more applications’)

subsequent application protocol or adapt itself to cards that do not meet all the requirements of the standard.

Byte 1	Byte 2				
Bit_Rate_capability	Max_Frame_Size	Protocol_Type	FWI	ADC	FO
8 bits	4 bits	4 bits	4 bits	2 bits	2 bits

Figure 3.115 Format of the Protocol Info field

Tables 3.17 and 3.18 show the coding of the individual fields of the Protocol Info field.

Table 3.17 Coding of the FO (frame option) parameter

b2	b1	Meaning
1	x	PICC supports NAD
x	1	PICC supports CID

Table 3.18 Coding of the ADC (application data coding) parameter

b4	b3	Meaning
0	0	Proprietary application data coding
0	1	Application coded as described in the text

The frame waiting time integer (FWI) specifies the maximum amount of time needed by the card to start transmitting a response after it has fully received a command from the terminal. If a card does not respond within this interval, the terminal can assume that communications with the card have been interrupted. The frame waiting time (FWT) is calculated using the following formula:

$$\mathrm{FWT} = (256 \times 16/f_C) \times 2^{\mathrm{FWI}}$$

The value of FWI lies between 0 and 14, with 15 being reserved for future use (RFU). The following minimum and maximum values for the frame waiting time can be calculated using this formula:

- minimum value (FWI = 0): $\mathrm{FWT_{min}} \approx 302\ \mu s$
- minimum value (FWI = 14): $\mathrm{FWT_{max}} = 4949$ ms

The Protocol_Type field indicates whether the card supports the ISO/IEC 14 443-4 transmission protocol. The coding of this field is shown in Table 3.19.

Table 3.19 Protocols supports by the card. All other values are reserved for future use (RFU)

b4	b3	b2	b1	Meaning
0	0	0	1	PICC supports ISO/IEC 14 443-4
0	0	0	0	PICC does not support ISO/IEC 14 443-4

In the Max_Frame_Size field, the card indicates the maximum frame size it can receive. This is limited by the size of the receive buffer in the card's RAM. Inexpensive chips typically have only a small amount of RAM, so they can only receive small frames. The Bit_Rate_capability field indicates the data transmission rates supported by the card, as shown in Table 3.21.

Table 3.20 Maximum frame size capacity of the card

Max_Frame_Size code in ATQB	0	1	2	3	4	5	6	7	8	9–F
Maximum frame size (bytes)	16	24	32	40	48	64	96	128	256	RFU>256

Table 3.21 Bit rates supported by the card. All other values are reserved for future use (RFU)

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	0	0	0	0	0	0	0	PICC supports only 106 kbit/s in both directions
1	X	X	X	0	X	X	X	The same bit rate in both directions
X	X	X	1	0	X	X	X	PICC to PCD: 1 etu = $64/f_C$, 212 kbit/s supported
X	X	1	X	0	X	X	X	PICC to PCD: 1 etu = $32/f_C$, 424 kbit/s supported
X	1	X	X	0	X	X	X	PICC to PCD: 1 etu = $16/f_C$, 847 kbit/s supported
X	X	X	X	0	X	X	1	PCD to PICC: 1 etu = $64/f_C$, 212 kbit/s supported
X	X	X	X	0	X	1	X	PCD to PICC: 1 etu = $32/f_C$, 424 kbit/s supported
X	X	X	X	0	1	X	X	PCD to PICC: 1 etu = $16/f_C$, 847 kbit/s supported

As already mentioned, the card changes to the Ready Declared state after it transmits its ATQB (see Figure 3.111). In this state, the card responds only to the REQB/WUPB, ATTRIB and HLTB commands. It responds to a REQB/WUPB command in the same way as when it is in the Idle state.

If the card recognizes a valid ATTRIB command in which the PUPI matches the PUPI of the card, it transmits an Answer to ATTRIB frame and changes to the Active state. If the PUPI parameters do not match, the card remains in the Ready Declared state and waits for an ATTRIB command with the proper PUPI. The card responds to an appropriate HALTB command (containing the proper PUPI) by transmitting an Answer to HALTB and changing to the Halt state.

In the Active state, the card has a card identifier (CID) that is assigned to it by the ATTRIB command. As a result, it is in a higher protocol layer and responds to suitable application commands having the proper CID and correct CRC_B checksum. Special commands belonging to this higher protocol layer can put the card into the Idle or Halt state. When it is in the Active state, the card is not allowed to respond to REQB/WUPB, Slot Marker and ATTRIB commands.

In the Halt state, the card is passive and can only be reset to the Idle state by a valid WUPB command with the proper PUPI.

Format and coding of the ATTRIB command

The ATTRIB command is transmitted by the terminal to the card and contains information needed to select a card. It also contains information regarding the parameters supported by the terminal for subsequent communications and those required by the card for error-free communications. This includes parameters such as the minimum value of the guard time (TR0), the minimum value of the synchronization time (TR1), whether the card can suppress SOF and/or EOF to accelerate communications, the maximum frame size and selection of the optimum bit rate.

'1D'	**Identifier (PUPI)**	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Higher Layer Inf.**	**CRC_B**
1 byte	4 bytes	1 byte	1 byte	1 byte	1 byte	0 or more bytes	2 bytes

Figure 3.116 Format of the ATTRIB command. 'Identifier' contains the value of the PUPI, which is sent by the card in the ATQB. The format of Param 1 is shown in Figure 3.117

b8	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**
Minimum TR0		Minimum TR1		EOF	SOF	RFU	

Figure 3.117 Format of Parameter 1

The value of the 'Minimum TR0' parameter defines the minimum time that the card must wait before responding to a command received from the terminal. This is the time needed by the terminal to switch from transmit mode to receive mode, which depends on the performance of the terminal.

Table 3.22 Coding of the Minimum TR0 parameter

b8	b7	Minimum TR0
0	0	Default value
0	1	$48/f_S$
1	0	$16/f_S$
1	1	RFU

The value of the ‘Minimum TR1’ parameter defines the minimum delay between the activation of the subcarrier and the start of data transmission (see Figure 3.107). The terminal needs this time for synchronization with the card.

Table 3.23 Coding of the Minimum TR1 parameter

b6	b5	Minimum TR1
0	0	Default value
0	1	$64/f_S$
1	0	$16/f_S$
1	1	RFU

Bits b3 and b4 indicate to the terminal whether the card supports suppression of EOF and/or SOF from the card to the terminal in order to reduce communications overhead. This capability is optional for the card.

Table 3.24 SOF utilization

b3	SOF required
0	yes
1	no

Table 3.25 EOF utilization

b4	EOF required
0	yes
1	no

The lower nibble of Parameter 2 (bits b4–b1) specifies the maximum size of a frame that can be received from the terminal. The upper nibble is used to select the bit rates in both directions. The terminal can make this choice, since it already knows the bit rates supported by the card from the ATQB.

The lower nibble of Parameter 3 is used to confirm the protocol type. The coding corresponds to that shown in Table 3.19. The upper nibble is set to '0'. All other values are reserved for future use.

Table 3.26 Coding of bits b4–b1 in Parameter 2, which specify the maximum frame size

Max_Frame_Size code in ATTRIB	0	1	2	3	4	5	6	7	8	9–F
Maximum frame size (bytes)	16	24	32	40	48	64	96	128	256	RFU>256

Table 3.27 Coding of bits b8–b5 in Parameter 2, which select the transmission bit rate

b8	b7	b6	b5	Meaning
0	0	x	x	PICC to PCD: 1 etu = $128/f_C$, bit rate is 106 kbit/s
0	1	x	x	PICC to PCD: 1 etu = $64/f_C$, bit rate is 212 kbit/s
1	0	x	x	PICC to PCD: 1 etu = $32/f_C$, bit rate is 424 kbit/s
1	1	x	x	PICC to PCD: 1 etu = $16/f_C$, bit rate is 847 kbit/s
x	x	0	0	PCD to PICC: 1 etu = $128/f_C$, bit rate is 106 kbit/s
x	x	0	1	PCD to PICC: 1 etu = $64/f_C$, bit rate is 212 kbit/s
x	x	1	0	PCD to PICC: 1 etu = $32/f_C$, bit rate is 424 kbit/s
x	x	1	1	PCD to PICC: 1 etu = $16/f_C$, bit rate is 847 kbit/s

Parameter 4 also consists of two parts. The lower nibble is called the ‘card identifier’ (CID) and defines the logical number of the addressed card, with a range of 0 to 14. The value 15 is reserved for future use. The card identifier is specified by the terminal and is unique for each active card. If the card does not support CID, a value of '0' is used. The upper nibble is set to '0'. All other values are reserved for future use.

The ‘Higher-Layer Inf’ field can be used to transfer any desired higher level commands. The ability to process such commands is optional for the card.

Response to the ATTRIB command

The card responds to every valid ATTRIB command (having the proper PUPI and correct CRC_B checksum) as shown in Figure 3.118.

Byte 1		**Bytes 2–n**	
MBLI	CID	Higher-layer response	CRC_B
1 byte		Optionally 0 or more bytes	2 bytes

Figure 3.118 Format of the response to an ATTRIB command

If the terminal receives a valid response to an ATTRIB command (one having the same CID and a correct CRC_B checksum), it knows that card selection was successful. The lower nibble of the first byte in the response (bits b4–b1) contains the CID. The upper nibble of the first byte (bits b8–b5) is called the ‘maximum buffer length index’ (MBLI). The card uses the MBLI to tell the terminal the maximum size of its input buffer. This allows the terminal to avoid causing the input buffer of the card to overflow by sending too many chained frames. If

MBLI is set to 0, the card does not provide any information about the size of its internal buffer. If MBLI is greater than 0, the maximum internal buffer length (MBL) can be calculated using the following formula:

$$\text{MBL} = (\text{PICC maximum frame size}) \times 2^{(\text{MBL}-1)}$$

The card sends its maximum frame size parameter to the terminal in the ATQB. When the terminal transmits chained frames, it must ensure that the cumulative length does not exceed the value of MBL.

The HLTB command

The HLTB command is used to place a card in the Halt state, so that it no longer responds to REQB commands. After responding to this command, the card ignores all subsequent commands except WUPB.

'50'	Identifier	CRC_B
1 byte	4 bytes	2 bytes

Figure 3.119 Format of the HLTB command

The Identifier parameter contains the PUPI of the card to be placed in the Halt state. The format of the card's response to a valid HLTB command is shown in Figure 3.120.

'00'	CRC_B
1 byte	2 bytes

Figure 3.120 Format of the response to a HLTB command

Example of an anticollision sequence with three Type-B cards

The standard gives the developer the freedom to implement various types of anticollision strategies. This corresponds to the basic function of a standard, which is to make interoperability possible while providing as much latitude as possible for implementation in order to avoid hindering technical progress. An example of an anticollision sequence is shown in Figure 3.121. This example, which is also contained in the annex to the standard, serves to illustrate the processes and commands described in the previous section. It makes no claim to being a technically superior implementation.

3.6.3.4 Data transmission protocol (ISO/IEC 14 433-4)

A half-duplex block transmission protocol that is tailored to the specific requirements of a contactless system is defined in Part 4 of ISO/IEC 14 433-4. This protocol is largely based on

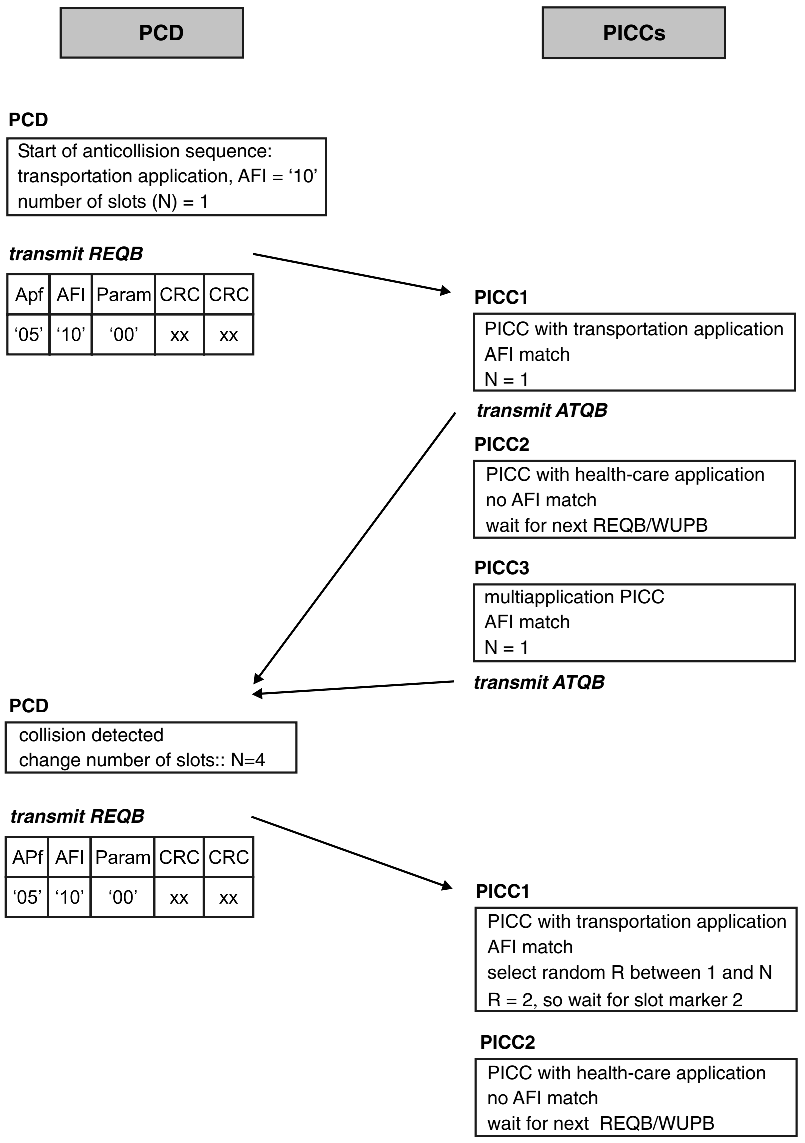

PCD
PICCs
PCD
Start of anticollision sequence:
transportation application, AFI = '10'
number of slots (N) = 1
transmit REQB
Apf | AFI | Param | CRC | CRC
'05' | '10' | '00' | xx | xx
PICC1
PICC with transportation application
AFI match
N = 1
transmit ATQB
PICC2
PICC with health-care application
no AFI match
wait for next REQB/WUPB
PICC3
multiapplication PICC
AFI match
N = 1
transmit ATQB
PCD
collision detected
change number of slots:: N=4
transmit REQB
APf | AFI | Param | CRC | CRC
'05' | '10' | '00' | xx | xx
PICC1
PICC with transportation application
AFI match
select random R between 1 and N
R = 2, so wait for slot marker 2
PICC2
PICC with health-care application
no AFI match
wait for next REQB/WUPB

Figure 3.121 Example of an anticollision sequence with three Type-B cards (Part 1 of 2)

the T = 1 protocol defined in the ISO/IEC 7816-3 standard, which is widely used throughout the world. This simplifies the implementation of dual-interface cards, since they anyhow must support a contact-type transmission protocol.

For Type-A cards, the standard defines an activation sequence that must be executed before starting the actual protocol. During this sequence, parameters for the subsequent data transmission are specified and exchanged between the terminal and the card. These parameters

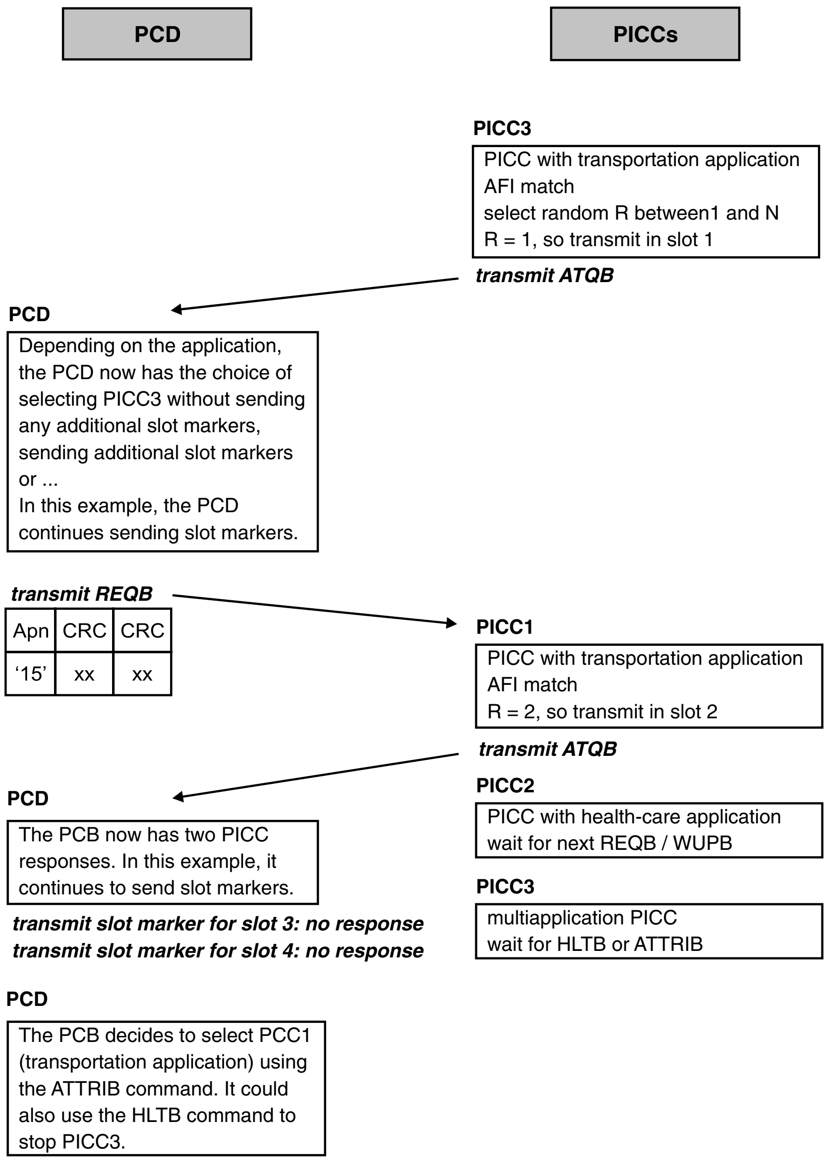

PCD
PICCs
PICC3
PICC with transportation application
AFI match
select random R between1 and N
R = 1, so transmit in slot 1
transmit ATQB
PCD
Depending on the application, the PCD now has the choice of selecting PICC3 without sending any additional slot markers, sending additional slot markers or ...
In this example, the PCD continues sending slot markers.
transmit REQB
Apn CRC CRC
'15' xx xx
PICC1
PICC with transportation application
AFI match
R = 2, so transmit in slot 2
transmit ATQB
PCD
The PCB now has two PICC responses. In this example, it continues to send slot markers.
PICC2
PICC with health-care application
wait for next REQB / WUPB
PICC3
multiapplication PICC
wait for HLTB or ATTRIB
transmit slot marker for slot 3: no response
transmit slot marker for slot 4: no response
PCD
The PCB decides to select PCC1 (transportation application) using the ATTRIB command. It could also use the HLTB command to stop PICC3.

Figure 3.121 Example of an anticollision sequence with three Type-B cards (Part 2 of 2)

relate to things such as the bit rate in each direction and the required waiting times between frames.

Type-B cards do not need any special activation sequence. They can immediately initiate the actual data transmission protocol after being selected. With such cards, the necessary parameters for data transmission are specified and exchanged during the initialization and selection processes, as described in the previous section.

Protocol activation for Type-A cards

After a Type-A PICC has been successfully selected, as previously described, the terminal executes an activation sequence as illustrated by the flow chart shown in Figure 3.122.

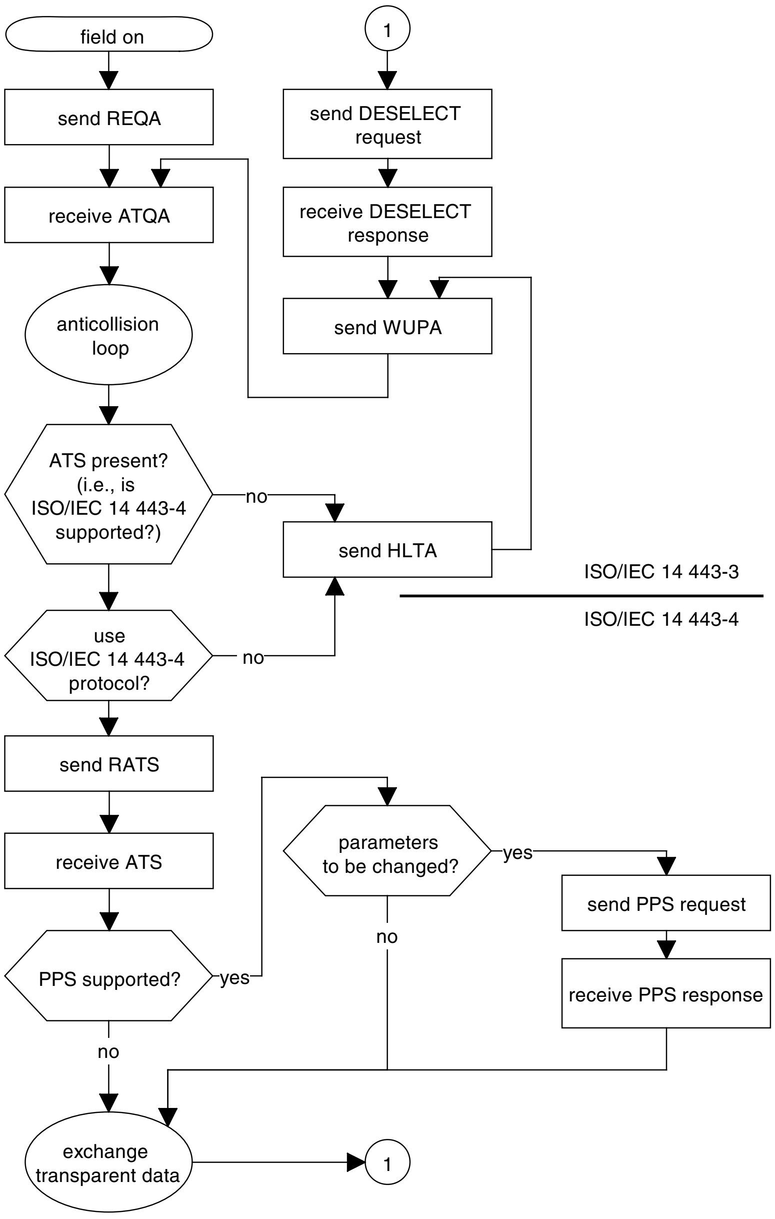


Figure 3.122 Activation of a Type-A card by a terminal

From the SAK (Select Acknowledge) transmitted by the card at the end of the anticollision loop, the terminal can recognize whether the card supports the standard data transmission protocol. If it does not, the terminal issues a HLTA command to place the card in the Halt state. If the protocol defined in ISO/IEC 14 433-4 is supported, the terminal sends a RATS (request for answer to select) command to the card. The RATS command and the ATS (answer to reset) returned by the card are used to exchange data and parameters in order to determine the data transmission options supported by the card and the terminal. Following this, the values of the modifiable parameters can be selected using PPS (protocol and parameter selection) to make optimum use of the capabilities of the card and the terminal. In order to make inexpensive, technically simple implementations possible, default values are defined for the modifiable parameters. In the simplest case, the card supports only these default values. In this case, the PPS sequence is unnecessary, and the terminal can immediately start exchanging data using the block protocol after receiving the ATS.

Request for answer to select (RATS)

The RATS command contains a parameter byte specifying the maximum frame size that the terminal can receive ('frame size for proximity coupling card', or FSDI) and the card identifier (CID) assigned to the card for the duration of its active state. Starting with the reception of the RATS command, the card uses this CID as its logical identifier.

'E0'	Parameter	CRC_A
1 byte	1 byte	2 bytes

Figure 3.123 Format of the RATS (Request for Answer to Select) command

b8	b7	b6	b5	b4	b3	b2	b1
FSDI				CID			

Figure 3.124 Format of the Parameter byte of the RATS command. The CID defines the logical number of the addressed card and has a range of 0 through 14; 15 is reserved for future use. FSDI codes the maximum frame size (FSD) that the terminal can receive

Table 3.28 Coding of bits b8–b5 of the Parameter byte of the RATS command

FSDI	0	1	2	3	4	5	6	7	8	9–F
FSD (bytes)	16	24	32	40	48	64	96	128	256	RFU>256

Answer to Select (ATS)

A Type-A card responds to a RATS command with an Answer to Select (ATS). The ATS identifies the set of parameters supported by the card. These parameters may include:

- the maximum frame size
- the bit rates in both directions supported by the card
- the waiting time between frames
- the specific frame guard time
- support for NAD and CID.

Default values are specified for cards that do not offer any selection of parameters. In the simplest case, which is when only the default values are supported, the ATS consists of only the length byte and the CRC bytes.

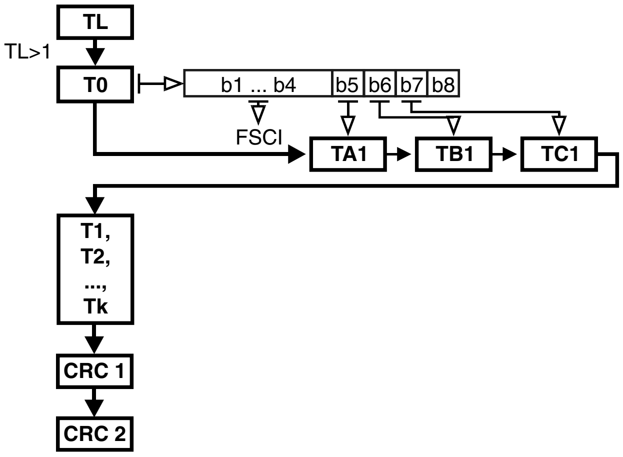


Figure 3.125 Format of ATS

Table 3.29 The data elements of the ATS and their designations according to ISO/IEC 14 433-4

Data element	Designation
TL	Length byte
T0	Format byte
TA1, TB1, TC1	Interface bytes
T1, T2, . . . Tk	Historical bytes
CRC1, CRC2	Cyclic redundancy check

Length byte The length byte (TL) indicates the number of bytes transmitted in the ATS, including the TL byte but excluding the two CRC bytes. The length of the ATS is not allowed to exceed the maximum frame length (FSD) given in the RATS command. This means that the maximum value of TL is not allowed to be greater than FSD – 2.

Format byte If the length given in TL is greater than 1, the format byte (T0) is sent next. T0 consists of three parts, as follows:

- The most significant bit (b8) has a value of 0. The value 1 is reserved for future use.
- Bits b7–b5 indicate the presence of subsequent interface bytes TC1, TC2 and TC3.
- The lower nibble (b4–b1) is called the 'frame size for proximity card integer' (FSCI). It codes the value of 'frame size for proximity card' (FSC), which is the maximum frame size that can be received by the card. The default value for FSCI is 2, which corresponds to a maximum frame size of 32 bytes.

Table 3.30 Coding of the format byte (T0)

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	...	...	...	...	...	...	...	Set to 0 (1 is RFU)
0	1	X	X	...	...	...	...	TC1 transmitted
0	X	1	X	...	...	...	...	TB1 transmitted
0	X	X	1	...	...	...	...	TA1 transmitted
0	...	...	...	X	X	X	X	Maximum frame size (FSCI)

Table 3.31 Coding of bits b4–b1 of the FSCI parameter

FSCI	0	1	2	3	4	5	6	7	8	9–F
FSC (bytes)	16	24	32	40	48	64	96	128	256	RFU>256

TA1 interface byte The TA1 interface byte consists of four parts, as follows:

- The most significant bit (b8) indicates whether different divider values can be used for the two transmission directions. The value of the etu (elementary time unit, equal to the duration of one bit) is determined by the divider D according to the formula 1 etu $= 128 \div (D \times f_C)$. The initial value of D is 1, which yields an etu value of $128/f_C$.
- Bits b7–b5, which are called 'divisor send' (DS), specify the bit rates supported by the card for data transmission from the card to the terminal.
- Bit b4 is set to 0. The value 1 is RFU.
- Bits b3–b1, which are called 'divisor receive' (DR), specify the bit rates supported by the card for data transmission from the terminal to the card.

The divider values are selected by the terminal in the subsequent PPS command.

Table 3.32 Coding of the TA1 interface byte

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	...	...	...	0	...	...	...	Different values of D are supported for data transmissions in the two directions
1	...	...	...	0	...	...	...	Only one value of D is supported for data transmissions in the two directions
...	1	...	...	0	...	...	...	DS = 8 is supported
...	...	1	...	0	...	...	...	DS = 4 is supported
...	...	...	1	0	...	...	...	DS = 2 is supported
...	...	...	...	0	1	...	...	DR = 8 is supported
...	...	...	...	0	...	1	...	DR = 4 is supported
...	...	...	...	0	...	...	1	DR = 2 is supported

TB1 interface byte The TB1 interface byte is used to transfer parameters that define the frame waiting time and the start-up frame guard time. Consequently, the TB1 interface byte consists of two parts, as follows:

- The upper nibble (b8–b5) is called the 'frame waiting time integer' (FWI) and determines the frame waiting time (FWT). The meaning and calculation of the frame waiting time are described in the next section.
- The lower nibble (b4–b1) is called the 'start-up frame guard time integer' (SFGI) and is used to calculate the start-up frame guard time (SFGT). The SFGT is the time needed by the card after transmission of the ATS before it is ready to receive the next frame. The value 15 is RFU. The value 0 means that the card does not need any particular SFGT. With a value ranging from 1 through 14, the SFGT is calculated using the formula:

$$\mathrm{SFGT} = (256 \times 16/f_{\mathrm{C}}) \times 2^{\mathrm{SFGI}}$$

The default value of SFGI is 0, and its maximum value ($\mathrm{SFGT}_{\mathrm{MAX}}$) is approximately 4949 ms.

b8	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**
FWI				SFGI			

Figure 3.126 Format of the TB1 interface byte

TC1 interface byte The TC1 interface byte indicates special protocol options. It consists of the following parts:

- The six most significant bits (b8–b3) are set to '0'. All other values are RFU.
- Bits b2 and b1 indicate which optional fields in the prologe field are supported by the card (see the next section). The terminal is not obliged to actually transmit all fields that are supported, so it may omit one or more fields. However, fields that are not supported by

the card may not be transmitted by the terminal to the card under any circumstances. The default value for bit b2 is 1, and the default value for b1 is 0. These values mean that CID is supported and NAD is not supported.

Table 3.33 Coding of the TC1 interface byte

b8	b7	b6	b5	b4	b3	b2	b1
0	0	0	0	0	0	b2 = 1 means that CID (card identifier) is supported	b1 = 1 means that NAD (node address) is supported

Historical bytes The historical bytes (T1 through Tk) are optional. Their contents are defined in ISO/IEC 7816-4 (see also Section 6.2, 'Answer to Reset (ATR)'). The maximum possible number of historical bytes can be determined from the maximum length of the ATS.

Protocol and parameter selection

If the card indicates in the ATS that it supports modifiable parameters, the terminal can change the parameters for the subsequent protocol by using the PPS (protocol and parameter selection) command. If the card does not support any modifiable parameters, it is not required to support the PPS command (see Figure 3.122). In this case, the protocol will continue with unaltered parameters.

If we assume that the card has indicated in the ATS that it supports modifiable parameters, the terminal can now evaluate whether it wants to utilize the modification options indicated by the card. If so, it transmits a PPS Request command having the structure shown in Figure 3.127.

Byte 1	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**
PPSS	PPS0	PPS1	CRC1	CRC2
start byte	Parameter 0 indicates whether PPS1 is present	Parameter 1 codes DRI and DSI		

Figure 3.127 Format of the PPS Request (protocol and parameter selection request) command

Start byte The start byte (PPSS) consists of two parts:

- The upper nibble (b8–b5) is set to 'D' to identify the PPS. All other values are RFU.
- The lower nibble (b4–b1), which is called the 'card identifier' (CID), defines the logical number of the addressed card.

Parameter 1 Parameter 1 indicates which of the possible values of DS and DR in the TA1 interface byte have been selected. This determines the transmission rates for subsequent data transmissions.

Table 3.34 Coding of Parameter 1 (PPS1)

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	0	0	0	DSI		DRI		DRI and DSI code the divisor *D*

Table 3.35 Coding of *D* by DRI and DRS

DRI / DSI	0	0	0	1	1	0	1	1
D	1		2		4		8	

Protocol and parameter selection response

The card confirms correct reception of the protocol and parameter selection request by means of a protocol and parameter selection response, which consists of only the PPSS start byte and the two checksum bytes (CRC 1 and CRC 2). After this, the terminal and the card use only the selected parameters for data transmission.

Activation frame waiting time

In order to avoid unnecessarily long waiting times in case of transmissions errors, ISO/IEC 14 433-4 specifies the maximum time between when a Type-A card receives the end of a frame and when it sends a response. This time is called the ‘activation frame waiting time’, and it is set to $65{,}536/f_C$ ($\approx$4833 µs). If a card does not respond within this interval, the terminal knows that communications with the card are impaired.

Error detection and correction

In a system using contactless cards, it must be expected that errors will occur more often during data transmission than is usual with contact-type cards. For example, with contact-type cards it is relatively uncommon for a card to be removed from a terminal while it is communicating with the terminal. With contactless cards, interruptions to communications can occur more often, since the cards are free to move within the working range of the terminal during use and can unintentionally leave the working range. It is thus important to have methods available that allow such interruptions to be recognized as quickly as possible and allow communications to be resumed in a state that is as well defined as possible.

In order to avoid having every type of error completely interrupt communications and force re-initiation of the entire process, ISO/IEC 14 433-4 defines rules for error detection and correction during the protocol activation phase for Type-A cards. Rather than describe these rules in detail, we refer you to the appropriate section of the standard, where they are presented in an illustrative manner.

Half-duplex protocol in accordance with ISO/IEC 14 433-4

The transmission protocol defined in Part 4 of ISO/IEC 14 433-4 takes into account the special requirements imposed by the use of contactless cards. For instance, it allows a terminal to communicate with several cards in parallel.

Like the T = 1 protocol for contact-type cards, this protocol supports a clean separation of layers in accordance with the OSI reference model. Separation of layers means that data belonging to a higher level layer are transmitted completely transparently with respect to lower lever layers. This protocol is based on the frames defined for both types of cards (Type A and Type B) in Part 3 of the standard. Four layers are distinguished:

- the physical layer (byte transmission in accordance with 14 433-3)
- the data link layer (for exchanging data blocks in accordance with 14 433-3)
- the session layer (coupled to the data link layer in order to minimize overhead)
- the application layer (where the commands are processed).

This block-oriented protocol starts after the activation sequence has been completed. The terminal is entitled to make the first transmission. This means that after the activation sequence, the card must wait until it receives a block from the terminal. The card responds to each block that it receives by transmitting a response block within a defined frame waiting time. After the response block has been sent, the terminal again holds transmit authorization and the card switches back to reception mode. Communications continue in this manner, with transmit authorization alternately held by the terminal and the card.

The protocol allows several cards to be concurrently activated by a terminal. It can switch back and forth among several cards without having to spend time deactivating a card and activating another card each time it switches cards.

As already mentioned, the probability of errors in data transmissions is higher in systems using contactless cards than in comparable systems using contact-type cards. It is thus especially important for communications between the terminal and the card to take place in the shortest possible time. The data transmission rate, which has a default value of 106 kbit/s, is relatively high compared with the default value of 9.6 kbit/s for the T = 0 and T = 1 transmission protocols.

Block structure

Each transmission block consists of a leading prolog field, an optional information field and a trailing epilog field. The information field contains data for the application layer.

There are three different types of blocks:

- Information blocks, which are used for the transparent exchange of data belonging to the application layer.
- Reception confirmation blocks (R blocks), which do not contain information fields and serve to indicate positive or negative confirmation of reception.

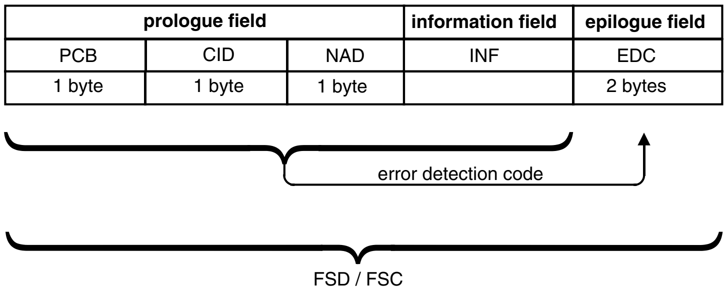


Figure 3.128 Block structure

- System blocks (S blocks), which are used to exchange control data for the protocol. Two types of S blocks are defined: WTX, which is an S block for extending the frame waiting time and has a single-byte information field, and DESELECT, which is an S block with no information field that is used to place the card in the Halt state.

Prologue field The prologue field consists of the protocol control byte (PCB), an optional card identifier (CID) and an optional node address (NAD). The coding of the protocol control byte is shown in Table 3.36.

Table 3.36 Coding of a the protocol control byte (PCB)

Bit	I-block PCB	R-block PCB	S-block PCB	
			DESELECT	WTX
b8	0	1	1	1
b7	0	0	1	1
b6	0 (1 is RFU)	1	0	1
b5	1 = chaining	0 = ACK, 1 = NAK	0	1
b4	1 = CID follows	1 = CID follows	1 = CID follows	
b3	1 = NAD follows	0 (no NAD)	0 (no NAD)	
b2	1	1 (0 is RFU)	1 (0 is RFU)	
b1	block number	block number	0 (1 is RFU)	

Card identifier (CID) The card identifier field is used to identify a particular card, and it also contains information about the power supplied to the card. The most significant two bits (b8 and b7) indicate the power level provided to the card by the terminal. Bits b6 and b5 are set to 0, with 1 being reserved for future use. Bits b4–b1 code the card identifier.

b8	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**
Power level indication		0 (1 = RFU)	0 (1 = RFU)	CID			

Figure 3.129 Format of the card identifier field

The coding of CID is defined in the PPS command for Type-A cards and in the ATTRIB command for Type-B cards. The following rules apply to evaluating the CID field:

- A card that does not support CID ignores all blocks containing a CID.
- A card that supports CID responds to blocks containing its CID by sending back its CID. It ignores blocks that contain other CIDs, and it responds to any block containing a CID of 0 by returning a block with no CID.

These rules allow the terminal to communicate concurrently with several active cards without having to deactivate cards that are not being addressed.

Table 3.37 Coding of the power level indication

b8	b7	Meaning
0	0	The card does not support power level indication
0	1	The amount of power is insufficient for full functionality
1	0	Adequate power for full functionality
1	1	More than adequate power for full functionality

Node address (NAD) The third byte of the prologue field is called the 'node address'. The node address is used to establish and address certain types of logical connections between the card and the terminal. Node addresses are used in the same manner as for contact-type cards. They are defined in ISO/IEC 7816-3 and described in Chapter 6.

Information field (INF) In I blocks, the information field acts as a container for data for the application layer. The content of this field is transferred fully transparently. In S blocks, the information field is used to control extending the frame waiting time.

Frame waiting time (FWT)

In order to achieve a defined termination of communications with a non-responding card in the shortest possible time, a 'frame waiting time' (FWT) is defined. It corresponds to the block waiting time of the $T = 1$ protocol for contact-type cards. The frame waiting time is the maximum interval between the end of a frame transmitted by the terminal and the start of the response frame from the card. If this interval expires without a response from the card, the terminal assumes that there is a malfunction in the card and reacquires transmit authorization in order to initiate error-detection mechanisms. As previously described, the ATS for Type-A cards contains the value of the frame waiting time integer (FWI) in the TB1 interface byte. The frame waiting time can be calculated from the FWI using the following formula:

$$\text{FWT} = (256 \times 16 \div f_\text{C}) \times 2^{\text{FWI}}$$

For Type-B cards, the FWI is defined in the ATQB (Answer to Request, Type B). The minimum value of FWT, which is obtained when the value of FWI is 0, is approximately 302 μs. The maximum value of FWT (FWT_MAX), which is obtained when the value of FWI is 14, is approximately 4949 ms. The normal command processing time of the card must be taken into account when selecting the frame waiting time. If the value of the frame waiting time is set too large, it will take longer for the terminal to detect that a card is not responding. In practice, cards can respond relatively quickly to most commands, but a few commands, such as commands involving the computation of a cryptographic algorithm, require significantly more time to execute. In order to avoid having to use a long frame waiting time for all commands just to accommodate such cases, there is a mechanism for extending the waiting time. This allows the card to request an extension of the frame waiting time for an individual command.

Extending the frame waiting time

To request an extension of the FWT, the card transmits a special S block called 'S(STX) request'. It receives a corresponding 'S(WTX) response' from the terminal to confirm the request. The terminal is not allowed to deny such a request.

The length of the extension to the frame waiting time is sent to the terminal using one byte in the information field of the S(WTX) S block. The new, temporary frame waiting time for processing the current command is obtained by multiplying this value by the frame waiting time.

b8	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**
Power level indication		WXTM (1–59; 0 and 60–63 are RFU)					

Figure 3.130 Format of the INF field of an S(WTX) request

b8	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**
		WXTM					

Figure 3.131 Format of the INF field of an S(WTX) response

The temporary frame waiting time ($\mathrm{FWT}_{\mathrm{TEMP}}$) is measured starting with the end of the S(WTX) response sent by the terminal. It is calculated using the following formula:

$$\mathrm{FWT}_{\mathrm{TEMP}} = \mathrm{FWT} \times \mathrm{WTXM}$$

If the formula yields a result greater than $\mathrm{FWT}_{\mathrm{MAX}}$ ($\approx$4949 ms), the value of $\mathrm{FWT}_{\mathrm{MAX}}$ must be used instead of the calculated value of $\mathrm{FWT}_{\mathrm{TEMP}}$.

Block chaining

The chaining function allows either one of the communicating parties to transmit data blocks that are too big to fit within a single frame by partitioning the data into several I frames sent in succession. Each of these chained I blocks has a length that is less than or equal to the frame length specified by FSC or FSD, as appropriate.

When block chaining is used, the sender sets the chaining bit in the protocol control byte (PCD) of the first block of the chain. This indicates to the recipient that the block chaining function is being used and that the subsequent block contains chained data. If the recipient receives the first block of the chain correctly, it indicates correct reception and its readiness to receive the next block by returning an R block with the same block number as the block it just received. The next block can then be sent. This back-and-forth exchange of I blocks and R blocks continues until the sender transmits an I block in which the chaining bit is not set. On receiving this block, the recipient knows that all of the application-layer data have been received, so it can process this data block and send the associated response.

Deactivating a card

When data transmission between the terminal and the card is completely finished, the terminal places the card in the Halt state by sending it a DESELECT command, which is transmitted using an S block. The card responds to this command with an S(DESELECT) response block and enters the Halt state.

Error handling

The block transmission protocol has error-detection mechanisms that are similar to those of the T = 1 protocol and allow resynchronization at various levels in case of transmission errors. The exact rules for the protocol processes can be found in Part 4 of ISO/IEC 14 433. Extensive examples of error-free protocol processes and error handling can also be found in the Annex to the standard.

3.6.4 Vicinity integrated circuits cards (ISO/IEC 15 693)

The ISO/IEC 15 693 standard, whose exact title is ‘Identification cards – Contactless integrated circuit(s) cards - Vicinity cards’, describes the properties and operating modes of contactless smart cards having a range up to 1 m. This type of card is preferred for applications such as access control, since a range of around 1 m means that it is not necessary for the card to be held in the user’s hand. Instead, it can remain in the user’s pocket, purse or other location. Up to now, this standard has not found widespread use in smart card systems, so we omit providing an extensive description of it here.

3.6.5 Test methods for contactless smart cards

The ISO/IEC 10 373 standard contains a compilation of all test methods for ID cards with and without chips. ISO/IEC 10 373 consists of seven parts, as shown below. As can be seen from this following list, three parts of this standard contain special test methods for contactless cards.

- Part 1: General characteristics tests
- Part 2: Cards with magnetic stripes
- Part 3: Integrated circuit(s) cards with contacts and related devices
- Part 4: Close-coupled cards

- Part 5: Optical memory cards
- Part 6: Proximity cards
- Part 7: Vicinity cards.

3.6.5.1 Part 4: Test methods for close-coupling smart cards

This part of the standard describes methods for testing the physical interfaces of contactless close-coupling smart cards compliant with ISO/IEC 10 536. Test aids in the form of reference coils and capacitive coupling surfaces are defined for use in measuring energy and data transfers between the terminal and the smart card and verifying conformance with the values specified in the standard (ISO/IEC 10 536).

3.6.5.2 Part 6: Test methods for proximity-coupling smart cards

Part 6 of the standard describes methods for testing the physical interfaces of contactless proximity-coupling smart cards compliant with ISO/IEC 14 433. The test aids necessary for this purpose, which consist of a calibration coil, a test setup for measuring load modulation and a reference card, are defined in the standard. Test methods for the following properties of the card or terminal are described in the standard:

- the resistance of the card to damage by electrostatic discharge
- the amplitude of the load modulation and the functionality of the card within its defined modulation region, as described in the basic standard
- the strength of the field generated by the terminal
- the modulation index and transient behavior (rise and fall times, overshoots etc.) of the signal generated by the terminal.

It must be noted that the small amplitude of the load modulation signal means it is difficult to make accurate and reproducible measurements of this signal, and it is to be hoped that suitable measurement equipment will soon be commercially available. Until such time, it is advisable to request assistance from suppliers of cards and/or terminals and agree on terms and conditions of delivery at an early date.

3.6.5.3 Part 7: Test methods for vicinity-coupling smart cards

Part 7 of the standard describes methods for testing the physical interfaces of vicinity-coupling smart cards compliant with ISO/IEC 15 693. The test aids and methods largely correspond to those in Part 6 of the standard. The only difference is in the construction of the reference card, due to the different subcarrier frequency.

4 Informatic Foundations

'A smart card is a small computer in credit-card format with no man–machine interface'. This statement expresses an essential fact, which is that in contrast to all other types of cards, the specific properties of smart cards are determined by the microcontroller integrated into the card.

The primary function of the plastic body of the card is to carry the microcontroller. Of course, other components may be present in addition to the microcontroller, but they are not essential to the actual smart card functions. A basic understanding of certain aspects of informatics is necessary to understand the characteristics of these small computers and the IT mechanisms based on them.

Our intention here is not to convey expert knowledge. That is anyhow not necessary for understanding the basic features of the procedures and techniques used with smart cards. The basic knowledge that this chapter offers provides a fully adequate level of comprehension. Consequently, the information presented here delves into the technical details only to the extent necessary to understand fundamental relationships.

Nearly half of this chapter is dedicated to cryptographic procedures used in the field of smart cards. Until a few years ago, the subject of cryptography was surrounded by a veil of secrecy and ignorance. However, this situation has changed dramatically in recent years, and there is now an extensive literature on this subject. As with the sections of this chapter that deal with the general aspects of information technology, here we provide only the basic information necessary for understanding cryptographic algorithms and protocols. For more detailed information, we refer you to the well-known books on this subject, such as those by Bruce Schneier [Schneier 96] and Alfred Menezes [Menezes 97]. A further rich source of information on cryptography is the World Wide Web, where you will find the home pages of several research institutes (such as [GMD]), standards organizations (such as [ETSE, IEC, ISO]), agencies (such as [BSI, NSA]), companies (such as [Certicom, Counterpane, R3, RSA]), associations (such as [CCC, Teletrust]) and individuals with an interest in the subject (such as [Gutmann]).

Smart Card Handbook, Third Edition. W. Rankl and W. Effing
© 2004 John Wiley & Sons, Ltd ISBN: 0-470-85668-8

4.1 STRUCTURING DATA

Storing or transmitting data unavoidably requires an exact definition of the data in question and their structure. Only then is it possible to subsequently recognize and interpret the data elements. Fixed-length data structures with non-modifiable sequences regularly cause systems to 'collapse'. The best example of this is the conversion of the many different European currencies to the euro. All systems and data structures with fixed currency definitions had to be upgraded at considerable cost. The same difficulties manifest themselves in many smart card applications. Fixed data structures that need to be extended or shortened sooner or later give rise to considerable effort and expense.

However, the problem of structuring data has been around for a long time, and there is an adequate choice of methods that can be used to solve the problem. One method that is very popular in the world of smart cards, and which is coming into more general use in informatics, comes from the field of data transmission. It is called Abstract Syntax Notation 1, or ASN.1 for short. This is a coding-independent description of data objects, originally developed for transmitting data between different computer systems. An alternative to ASN.1 would be using extensible markup language (XML) to structure data, but up to now this method has not gained a foothold in real applications in the smart card world.

In principle, ASN.1 is a sort of artificial language that is suitable for describing data and data structures, rather than programs. The syntax is standardized in ISO/IEC 8824, and the coding rules are defined by ISO/IEC 8825. Both of these standards were developed from Recommendation X.409 of the CCITT.

Describing ASN.1 in detail would require a book on its own, so here we only address a few essential aspects in order to give a general ideal of how it works. For further information, we suggest you consult the relevant literature, such as Walter Gora [Gora 98].

ASN.1 has a number of elementary ('primitive') data types and composite ('constructed') data types. It is also possible to extend the syntax of ASN.1 using macros in order to obtain any desired enhancements to ASN.1. Listings 4.1 through 4.3 show some simple examples of how ASN.1 can be used, including defining and coding data.

Table 4.1 Some of the data types used in ASN.1

Data type	Sort	Meaning
BOOLEAN	Primitive	Boolean value: yes/no
INTEGER	Primitive	Negative and positive integers
OCTET STRING	Primitive	Byte sequence (one byte = one 8-bit octet)
BIT STRING	Primitive	Bit sequence
SEQUENCE	Constructed	Several components combined to form a new data type

The basic idea of coding data using ASN.1 is to prefix each data object with a unique label and information about its length. The rather complex syntax of the description language also allows users to define their own data types and nest data objects. The original idea, which was to create a generally valid syntax that could form the basis for data exchange between fundamentally different computer systems, is scarcely used in smart cards. Currently, only a very small part of the available syntax is used in this area, mainly due to the very limited memory capacity of smart cards.

Listing 4.1 A simple example of data type definition using ASN.1

`SC_Controller ::= SEQUENCE {`	Definition of a new data type for SC_Controller.
`  Name IA5String,`	The name of the microcontroller is an ASCII string.
`  CPUType CPUPower,`	CPUType refers to the definition of CPUPower.
`  NPU BOOLEAN,`	Boolean value as a yes/no assertion regarding whether a coprocessor (NPU) is present.
`  EEPROMSize INTEGER,`	The size of the EEPROM is an integer value.
`  RAMSize INTEGER,`	The size of the RAM is an integer value.
`  ROMSize INTEGER}`	The size of the ROM is an integer value.
`CPUPower ::= ENUMERATED {`	Definition of a new data type for CPUPower as an enumerated type.
`  8Bit (8),`	Possible selection value for the 8-bit CPU type.
`  16Bit (16),`	Possible selection value for the 16-bit CPU type.
`  32Bit (32)}`	Possible selection value for the 32-bit CPU type.

Listing 4.2 The data definitions from Listing 4.1, filled with data for a particular microcontroller

`SuperXS SC_Controller ::= {`	Specific instance of the SC-Controller data type with the data for SuperXS.
`Name "XS 8 Bit",`	The name of the microcontroller is 'XS 8 Bit'.
`CPUType 8,`	This is an 8-bit CPU.
`NPU true,`	No coprocessor (NPU) is present.
`EEPROMSize 1024,`	The size of the EEPROM is 1024 bytes.
`RAMSize 256,`	The size of the RAM is 256 bytes.
`ROMSize 8192}`	The size of the ROM is 8192 bytes.

Listing 4.3 The data for a particular microcontroller from Listing 4.2, coded using the ASN.1 BER

`'30 1C'`	Tag '30' for a string with a length of 28 bytes ('1C').
`  '16 08 58 53 20 38 20 42 69 74'`	Tag '16' for an IA5 string with a length of 8 bytes ('08') and a content of '58 53 20 38 20 42 69 74' (= "XS 8 Bit").
`  '0A 01 08'`	Tag '0A' for an enumerated data type with a length of 1 byte ('01') and a content of '08'.
`  '01 01 FF'`	Tag '01' for a Boolean data type with a length of 1 byte ('01') and a content of 'FF', which corresponds to the value 'true'.
`  '02 02 04 00'`	Tag '02' for an integer data type with a length of 2 bytes ('02') and a content of '04 00' (1024).
`  '02 02 01 00'`	Tag '02' for an integer data type with a length of 2 bytes ('02') and a content of '01 00' (256).
`  '02 02 20 00'`	Tag '02' for an integer data type with a length of 2 bytes ('02') and a content of '02 00' (8192).

The Basic Encoding Rules (BER) for ASN.1 are defined in the ISO/IEC 8825 standard. Data objects created according to these rules are called BER-TLV-coded data objects. A BER-coded data object has a label (called a 'tag'), a length field and the actual data part, with an optional end marker. Certain bits in the tag are predefined by the coding rules. The actual structure is shown in Figure 4.1. The Distinguished Encoding Rules (DER) form a subset of the BER. These coding rules specify, among other things, the coding of the length information, which may be one, two or three bytes long. A basic summary of the BER and DER can be found in Burton Kaliski [Kaliski 93].

ASN.1 objects are coded using the classic TLV structure, in which 'T' (tag) denotes the object's label, 'L' (length) refers to its length and 'V' (value) is the actual data. The first field of a TLV structure is the tag for the data object in the following V field. To avoid the need for each user to define his or her own tags, which would open the door to incompatibility, there are standards that define tags for various, frequently used data structures. ISO/IEC 7816-6, for example, defines tags for objects used in general industrial applications, ISO/IEC 7816-4 defines tags for secure messaging, and EMV also defines several other tags. It is by no means the case that a given tag is universally used for the same type of data element, but a process of standardization is essentially taking place.

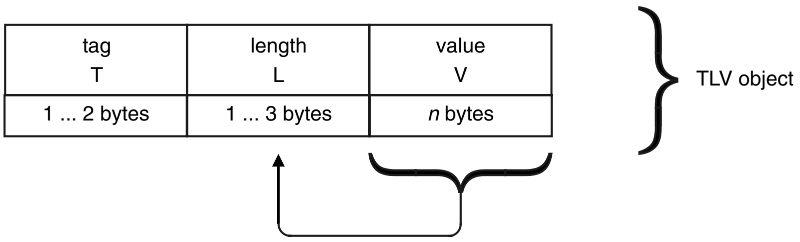


Figure 4.1 The principle of BER-based TLV coding according to ANS.1

The two most significant bits of the tag encode the class of the following data object. The class indicates the general type of the data object. The *universal* class indicates general data objects, such as an integers and character strings. The *application* class indicates that the data object belongs to a particular application or standard (e.g. ISO/IEC 7816-6). The other two classes, *context-specific* and *private*, fall under the heading of non-standardized applications.

The bit following the two class bits indicates whether the tagged object is constructed from other data objects. The five least-significant bits are the actual label. Since this can have a value of only 0 through 30, due to its limited address space, it is possible to point to the following byte by setting all five bits to 1. All values from 31 to 127 are allowed in the second byte. Bit 8 of the second byte is a pointer that is reserved for future use, so it cannot presently be used. The required number of length bytes is shown in Table 4.3.

The standard also defines the term 'template'. A template is a data object that serves as a container for other data objects. ISO/IEC 7816-6 defines the tags for possible data objects in the domain of industry-wide applications of smart cards. ISO 9992-2 covers the domain of smart card financial transactions.

This method of data encoding has several characteristics that are particularly useful in the field of smart cards. Since the available memory space is generally never enough, using data objects based on ASN.1 can produce considerable space savings. TLV encoding makes it possible to transfer and store variable-length data without a lot of complications. This allows

Table 4.2 ASN.1 tag coding

Byte 1	b8	b7	b6	b5	b4	b3	b2	b1	Meaning
	0	0	...	...	...	...	...	...	Universal class
	0	1	...	...	...	...	...	...	Application class
	1	0	...	...	...	...	...	...	Context-specific class
	1	1	...	...	...	...	...	...	Private class
	...	...	0	...	...	...	...	...	Primitive data object
	...	...	1	...	...	...	...	...	Constructed data object
	...	....	...	X	X	X	X	X	Tag code (0–30)
	...	...	...	1	1	1	1	1	Pointer to the following byte (byte 2), which specifies the tag code
Byte 2	b8	b7–b1							Meaning
	0	31–127							Tag code

Table 4.3 Structure of the DER length field in ASN.1

Byte 1	Byte 2	Byte 3	Meaning
0–127	—	—	One byte is needed for these length values
'81'	128–255		Two bytes are needed for these length values
'82'	256–65,535		Three bytes are needed for these length values

memory to be used very economically. This is illustrated in Figure 4.2, which shows the TLV encoding of a name.

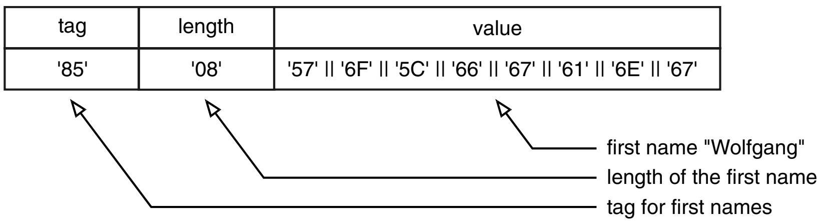


Figure 4.2 TLV encoding of the name 'Wolfgang'

Subsequent extensions to data structures can be undertaken very easily with ASN.1, since all that is necessary is to insert additional TLV-coded data objects into the existing data structure. Full compatibility with the previous version is retained as long as the previous TLV objects are not deleted. The same is true of new versions of data structures in which changes have been made with respect to the previous coding. This is a straightforward process that only requires modifications to the tags. It is equally simple to represent the same data using different codings. Collectively, these advantages explain why the ASN.1 syntax, based on TLV coding, is particularly popular in the smart card industry.

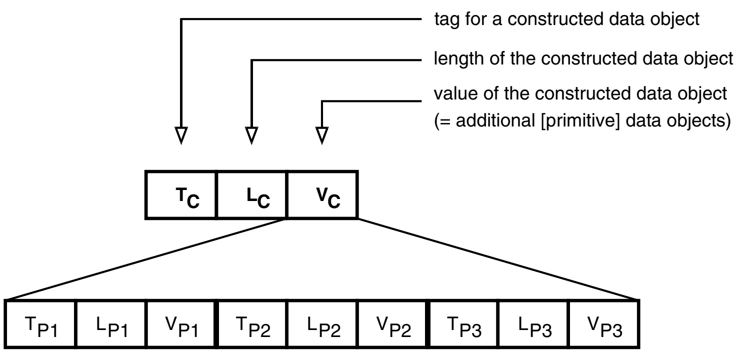

tag for a constructed data object
length of the constructed data object
value of the constructed data object
(= additional [primitive] data objects)
T_C
L_C
V_C
T_{P1}
L_{P1}
V_{P1}
T_{P2}
L_{P2}
V_{P2}
T_{P3}
L_{P3}
V_{P3}

Figure 4.3 Basic scheme for forming constructed TLV-coded data structures from several primitive TLV-coded data objects. The indices ‘C’ and ‘P’ stand for ‘constructed’ and ‘primitive’

The main disadvantage of ASN.1 data objects is that the administrative data overhead is rather high if the volume of user data is small. For example, if the user data is only one byte, two additional bytes (tag and length) are still needed for its administrative data. However, the larger the volume of the user data, the more favorable is the relationship. The ASN.1 structured data in the German health insurance card form a good example of this. There are between 70 and 212 bytes of user data. The administrative data amount to 36 bytes, which means that the administrative overhead ranges from 17 to 51 %.

We can recapitulate all the above with a further example. Suppose we wish to store surnames, given names and titles in a file with a transparent data structure. Irrespective of the proper ASN.1 description, the TLV-coded data will have the structure shown in Figure 4.4. The tags used in this example have been freely chosen and thus do not correspond to any relevant standard.

	T	L	V	T	L	V	T	L	V
version 1	'85'	'07'	"Manfred"	'87'	'05'	"Meier"	'84'	'04'	"Ing."

	T	L	V	T	L	V	T	L	V
version 2	'84'	'04'	"Ing."	'85'	'07'	"Manfred"	'87'	'05'	"Meier"

	T	L	V	T	L	V	T	L	V
version 3	'87'	'05'	"Meier"	'85'	'07'	"Manfred"	'84'	'04'	"Ing."

Figure 4.4 An example of sequence independence within a TLV structure

When evaluating this data structure, the computer compares the first tag with all tags known to it. If it finds a match, then it recognizes the first object as a given name. It reads the length of this object from the next byte. The subsequent bytes are then the actual object, i.e. the given name. This is followed by the next TLV object, whose first byte is the tag for a surname. The computer recognizes this using exactly the same process as for the first object.

If it becomes necessary to extend the data structure, e.g. by adding a title, a new type of data object can simply be inserted into the existing structure. The insertion point is unimportant. The extended structure remains fully compatible with the previous version, since the new type of data object receives its own tag and is thus unambiguously identified. Programs that only know the old tags will not be upset by the new one, since they do not recognize it and thus automatically skip it. Other programs that do know the new tag can evaluate it, but even if the old structure is used, they will not experience any problems.

4.2 CODING ALPHANUMERIC DATA

The alphanumeric data in the files and data objects of smart cards can be stored in a wide variety of formats. In part, this is a result of intensive memory space optimization measures and a lack of general agreement among the various applications and specifications, and in part it is due to the triumphant progress of smart cards in countries outside of Western Europe, which have their own alphabets. In such situations, the original 7- and 8-bit character sets must be replaced by more powerful coding schemes for alphanumeric data.

4.2.1 7-bit code

A total of 128 (2^7) characters can be represented using a 7-bit code. The most widely used international 7-bit code, which is commonly known as the ASCII (American Standard Code for Information Interchange) code, is specified in ISO/IEC 646. The importance of ASCII has been steadily decreasing for many years, since the number of characters it can represent is much too small.

4.2.2 8-bit code

The most commonly used 8-bit code ($2^8 = 256$ characters) is derived from the 7-bit ASCII code and is standardized in ISO/IEC 8859. It consists of two 7-bit code tables specifying control characters and printable characters. The lower order table is identical to the 7-bit ASCII table and is always the same. The higher order table can vary to accommodate a country-specific character set. Probably the best-known higher-order code table is Latin 1, which contains the characters specific to the countries of Western Europe. Latin 2, by contrast, contains the special characters for the East European countries. ISO/IEC 8859 consists of 16 parts in total, which define a series of higher order code tables for the character sets of various languages.

The characters of the Latin 1 code table in ISO/IEC 8869 are also found in a slightly modified coding in DOS as ‘Code Table 850’ according to the IBM register, in the form of ‘PC ASCII’ and as ‘ANSI code’ under Windows.

EBCDIC (‘extended binary coded decimal interchange code’), which is widely used in mainframe computers, is not used with smart cards.

Table 4.4 The 7-bit standard character table specified for the GSM system by GSM 03.38, which is based on the ASCII character set. Each character is shown at the top center of each cell, with the 7-bit code in decimal notation at the bottom left and hexadecimal notation at the bottom right. The following abbreviations are used: CR = carriage return, LF = line feed, SP = space

@ 0 '00'	Δ 16 '10'	SP 32 '20'	0 48 '30'	¡ 64 '40'	P 80 '50'	¿ 96 '60'	p 112 '70'
£ 1 '01'	– 17 '11'	! 33 '21'	1 49 '31'	A 65 '41'	Q 81 '51'	a 97 '61'	q 113 '71'
$ 2 '02'	Φ 18 '12'	" 34 '22'	2 50 '32'	B 66 '42'	R 82 '52'	b 98 '62'	r 114 '72'
¥ 3 '03'	Γ 19 '13'	# 35 '23'	3 51 '33'	C 67 '43'	S 83 '53'	c 99 '63'	s 115 '73'
è 4 '04'	Λ 20 '14'	¤ 36 '24'	4 52 '34'	D 68 '44'	T 84 '54'	d 100 '64'	t 116 '74'
é 5 '05'	Ω 21 '15'	% 37 '25'	5 53 '35'	E 69 '45'	U 85 '55'	e 101 '65'	u 117 '75'
ù 6 '06'	Π 22 '16'	& 38 '26'	6 54 '36'	F 70 '46'	V 86 '56'	f 102 '66'	v 118 '76'
ì 7 '07'	Ψ 23 '17'	' 39 '27'	7 55 '37'	G 71 '47'	W 87 '57'	g 103 '67'	w 119 '77'
ò 8 '08'	Σ 24 '18'	(40 '28'	8 56 '38'	H 72 '48'	X 88 '58'	h 104 '68'	x 120 '78'
Ç 9 '09'	Θ 25 '19'	) 41 '29'	9 57 '39'	I 73 '49'	Y 89 '59'	i 105 '69'	y 121 '79'
LF 10 '0A'	Ξ 26 '1A'	* 42 '2A'	: 58 '3A'	J 74 '4A'	Z 90 '5A'	j 106 '6A'	z 122 '7A'

Table 4.4 (*Cont.*)

Ø 11 '0B'	1) 27 '1B'	+ 43 '2B'	; 59 '3B'	K 75 '4B'	Ä 91 '5B'	k 107 '6B'	ä 123 '7B'
ø 12 '0C'	Æ 28 '1C'	, 44 '2C'	< 60 '3C'	L 76 '4C'	Ö 92 '5C'	l 108 '6C'	ö 124 '7C'
CR 13 '0D'	æ 29 '1D'	- 45 '2D'	= 61 '3D'	M 77 '4D'	Ñ 93 '5D'	m 109 '6D'	ñ 125 '7D'
Å 14 '0E'	ß 30 '1E'	. 46 '2E'	> 62 '3E'	N 78 '4E'	Ü 94 '5E'	n 110 '6E'	ü 126 '7E'
å 15 '0F'	É 31 '1F'	/ 47 '2F'	? 63 '3F'	O 79 '4F'	§ 95 '5F'	o 111 '6F'	à 127 '7F'

4.2.3 16-bit code (Unicode)

Codes with a width of 16 bits allow 65,546 (2^{16}) characters to be represented. The only example of such a code is Unicode, which was developed as a private initiative by the Unicode Consortium [Unicode] as an industry standard.

The first 256 Unicode characters are identical to ISO/IEC 8859 Latin 1, so there is at least upward compatibility in this part of the character coding. Although the number of characters that can be represented with a 16-bit code is sufficient to represent the characters of the most important living languages, it is unfortunately not sufficient to represent all existing characters.

To compensate for this, a sort of escape sequence ('surrogate pairs') has been incorporated into the 16-bit character code in the current version of Unicode (3.0). This allows a supplementary byte to be used, so that up to one million characters can be represented.

4.2.4 32-bit code (UCS)

Unicode was originally limited to 65,536 characters. Although this limitation does not cause problems in everyday use, it can be avoided by using an extended character coding scheme. ISO/IEC 10 646 specifies a 32-bit code called the 'Universal Character Set' (UCS), which allows 4,294,967,296 (2^{32}) characters to be represented, although only half of the available codes (2,147,483,648) are actually used.

The four bytes of the UCS are called (in decreasing order of significance) *group*, *level*, *row* and *cell*. USC thus consists of 256 groups of 256 levels, each of which has 256 rows of 256 cells. A level thus specifies 65,546 characters. The lowest level, which is Group 0, Level 0, is called the 'basic multilingual plane' (BMP) and is identical to Unicode. The lowest row, which is Group 0, Level 0, Row 0, thus automatically corresponds to the character set of ISO/IEC 8859 Latin 1, and the first 128 characters are thus identical to the ASCII code.

This can be illustrated using a brief example. The letter "A" is coded as '30' in 7-bit ASCII and 8-bit ISO/IEC 8859 Latin 1. Since the first 256 characters of Unicode are identical to ISO/IEC 8859 Latin 1, the letter "A" is coded as '00 30' in 16-bit Unicode and as '00 00 00 30' in 32-bit UCS. UCS is the only character coding scheme that allows all characters of all living and dead languages to be coded using unique numerical values. Consequently, UCS is the most important coding scheme for future use, despite its memory requirement of four bytes per character.

There are three commonly used schemes, called 'UCS transition formats' (UTFs), for translating the codes of 32-bit UCS and 16-bit Unicode characters. UTF-8 translates characters into variable-length byte strings whose least-significant seven bits correspond to ASCII. UTF-16 uses 16 bits for coding and thus corresponds to the BMP of UCS, which also uses two bytes for coding. UTF-16 is also referred to as 'UTF-2', since it uses two bytes for coding. UTF-32 corresponds to the usual four-byte representation of UCS, for which reason it is also referred to as 'UCS-4'.

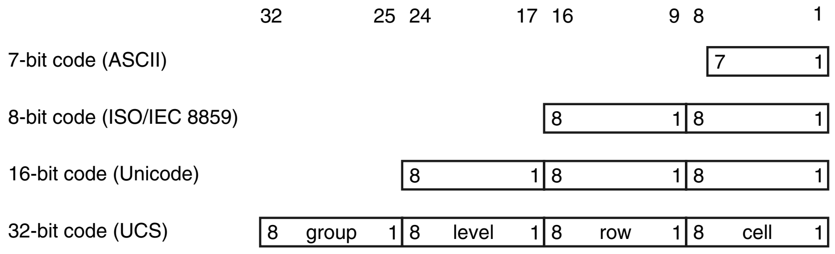


Figure 4.5 The relationships between the internationally most commonly used 7-, 8-, 16- and 32-bit codes for alphanumeric characters

4.3 SDL NOTATION

This book uses SDL notation to describe states and state transitions. For some years, this approach has been used ever more frequently in the smart card domain to describe state-oriented mechanisms, such as those used for communication protocols. 'SDL' stands for 'Specification and Description Language', and it is described in detail in CCITT Recommendation Z.100.

SDL notation is similar to the notation used in standard flowcharts. However, it does not describe program flows, but instead states and state transitions. SDL diagrams are constructed using standardized individual symbols interconnected by lines. The flow is always from top left to bottom right, so the lines connecting individual symbols do not need arrowheads to identify their start and end points.[1]

In simplified form, the notation can be regarded as a description of a system consisting of a certain number of processes, where each process is a state machine. If a state machine is in a stable state, it can receive a signal from outside. Depending on the data it receives, the

[1] For a detailed example of an SDL diagram, see Section 6.4.2, 'The T = 0 transmission protocol'

machine may then attain a specific new state. Additional actions may occur between the initial and final states, such as receiving and transmitting data or computing a value.

Figure 4.6 shows the 10 symbols used in this book. They are a selection from a much larger set defined in Z.100, but they suffice as a basic set for use with smart cards. The Start symbol (1) denotes the beginning of a process. Most SDL diagrams begin with this symbol. The Task symbol (2) indicates a specific activity, which is described by text within the box. With this symbol, there is no additional detailed description in the form of a subroutine. The Decision symbol (3) allows a query during a state transition, to which the answer may be 'yes' or 'no'. The Label symbol (4) marks a link to another SDL diagram and is primarily used to divide large diagrams into several smaller diagrams.

The Input (5) and Output (6) symbols represent interfaces to the outside world. The exact input and output parameters are described inside the symbol. The State symbol (7) is used to describe a state. The state attained at each stage is indicated by this symbol.

The final three symbols describe subroutines. The Subroutine symbol (8) indicates that the content of this box is described in more detail elsewhere. The Subroutine start (9) and Subroutine end (10) symbols delimit the detailed description of a subroutine.

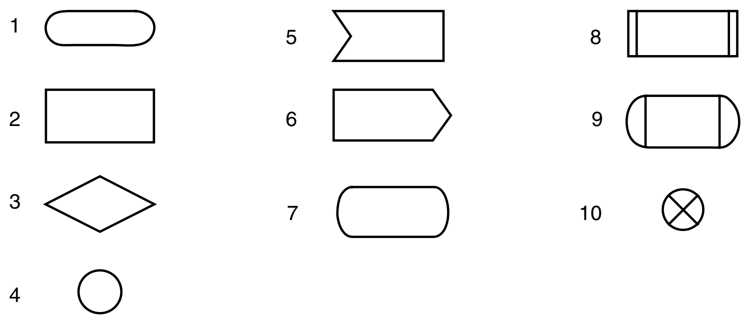


Figure 4.6 The SDL notation symbols used in this book, which are compliant with CCITT Z.100:

1 – Start	5 – Input	8 – Subroutine
2 – Task	6 – Output	9 – Subroutine start
3 – Decision	7 – State	10 – Subroutine end
4 – Label		

4.4 STATE MACHINES

A state machine is a type of automaton. A common example of an automaton is a vending machine, into which you insert a coin and then press a button. After this, you can open a compartment and remove your selection. In slightly more abstract terms, this automaton defines a chain of events involving various state transitions. In the initial state, the automaton waits for money to be inserted. Any other action, such as pressing a button, will not cause anything to happen. Only the insertion of a coin causes a transition from the initial state to the 'money inserted' state. The next transition occurs as a result of pressing the button, following which the automaton allows a compartment to be opened.

In informatics, state machines can be very effectively visualized using graphs or Petri networks. These are not only useful for modeling state machines; they can also be used to investigate certain properties of the systems they describe. The objectives are to identify any deadlocks that may occur in the process and ensure correct command processing.

4.4.1 Basic theory of state machines

The objective of this section is to provide an introduction to state diagrams, which are used to describe smart card applications, and a general explanation of how to interpret them.

A state diagram is a type of graph that represents a set of states and the interrelationships of these states. The states are shown as nodes, and their relationships are shown as lines. If a line indicates a direction, which means that it has an arrowhead at one end, it is called a 'directed line' and the graph is a 'directed graph'. The arrow indicates the direction in which a state transition can take place. The actual placement of the nodes and lines in the graph plays no part in the interpretation of the diagram. A sequence of nodes connected by lines is called a path. If the first and last nodes are the same and there is more than one node, the path is called a loop.

This is only a very small part of graph theory, but it is essentially all we need to be able to describe states and their associated state machines in smart card applications.

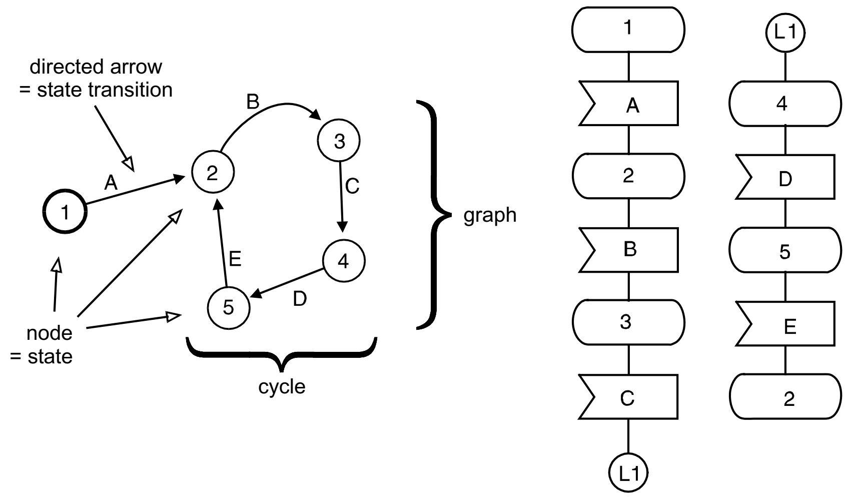


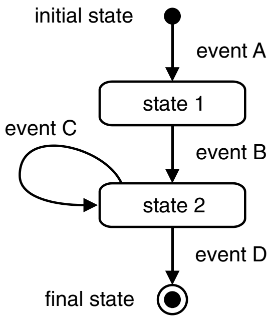
Figure 4.7 Examples of two different representations of state diagrams. On the left is a directed state diagram, and on the right an equivalent SDL diagram

4.4.2 Practical applications

An additional advantage of microprocessor cards compared with simple memory cards is that the command sequences can be specified in advance. It is thus possible to precisely specify all commands in terms of their parameters and sequence. In combination with object-oriented access authorization for files, this provides additional protection against unauthorized access.

Table 4.5 Description of the state diagram of Figure 4.6 in tabular form

State after transition	State before transition				
	1	2	3	4	5
1	—	—	—	—	—
2	A	—	—	—	E
3	—	B	—	—	—
4	—	—	C	—	—
5	—	—	—	D	—



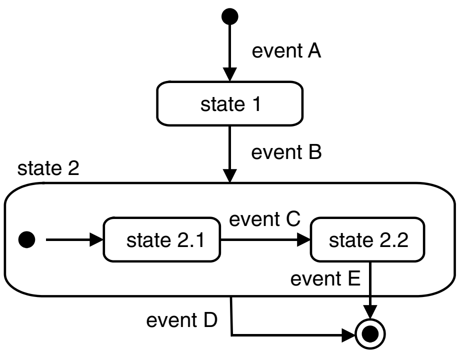


Figure 4.8 Two sample state diagrams using UML notation. The diagram on the left shows a simple sequential state flow, while the diagram on the right shows a state diagram with substates

However, the possibilities offered by smart cards in this respect vary greatly. Simple operating systems usually cannot manage state machines, while with modern operating systems it is even possible to define application-specific state machines that work with command parameters.

A typical example of a simple state machine is provided by the two commands needed to authenticate a terminal. The first command asks the card for a random number. This activates a state machine that accepts only an authentication command as the next command. If the card receives this command, the process completes and all other types of command are allowed. If the card receives any command other than an authentication command, the state machine generates an error message and the process is aborted. The command sequence must then be restarted from the beginning.

Such simple state machines have several major advantages in smart cards. Since they are limited to very few commands in a rigidly defined sequence, they require little memory space and program overhead. In many applications, it is sufficient to protect file contents using object-oriented access mechanisms, without imposing any other restrictions on command sequences. Only a few procedures, such as authentication, must follow prescribed sequences. This can be implemented with very little memory using simple state machines.

These simple state machines can be extended to verify all commands, along with all of their parameters, within a defined graph before they are executed. Depending on how the state machine is constructed, under certain conditions it may be possible to dispense with

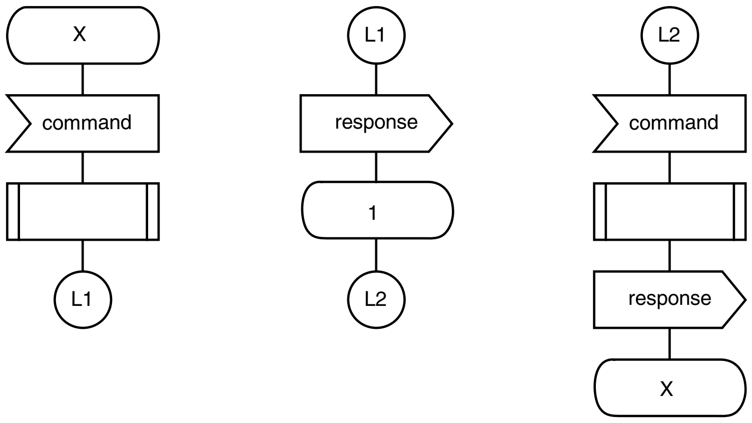


Figure 4.9 An example of a simple smart card state machine with two states, 'X' and '1'

object-oriented file access protection, since the state machine can perform all the necessary checks before a command is actually executed. Of course, an error in the state diagram could have fatal consequences for the security of the system. As it is very difficult to verify the complete absence of errors in the state diagrams of complex state machines, file access protection is still used in practice. Correctly describing all of the processes and commands present on a smart card is very time-consuming, so it is often necessary to do this empirically to a certain extent.

Now that we have described the advantages of state machines, we must mention their drawbacks. Implementing a state machine with the required capabilities is very time-consuming in terms of both design and subsequent programming. Since a state machine is controlled by the stored representation of a graph, a considerable amount of program memory is needed just to hold a state machine, since the graph must be stored in memory in addition to the actual state machine. The amount of memory space naturally depends on the complexity of the graph to be executed. The amount of information contained in a graph having many states and a corresponding number of transitions can be very large relative to typical smart card memory capacities.

State machines for smart cards are addressed by the ISO/IEC 7816-9 standard. It describes 'access control descriptors' (ACDs), which define the commands that are permitted in a specific state, along with their associated parameters. A smart card operating system can monitor hard-coded state machines using these ACDs.

In order to illustrate the capabilities of a state machine in summary form, Figure 4.10 shows the state diagram for a small application. Its operation is described below.

After a reset, the smart card is in the initial state, denoted by 1. In this state, every file in the directory may be selected using SELECT FILE; this does not cause a state transition. All other commands except PIN verification (VERIFY) are prohibited, and the card responds to such commands with an error message. After successful verification of the PIN, the state machine changes to state 2.

Two commands are permitted in state 2. The first path leads via SELECT FILE to state 3, where the selected file may be read. The second path originating from state 2 leads to state 4 after the terminal requests a random number from the card (ASK RANDOM). From here, any command other than EXTERNAL AUTHENTICATE leads back to the initial state (1). When

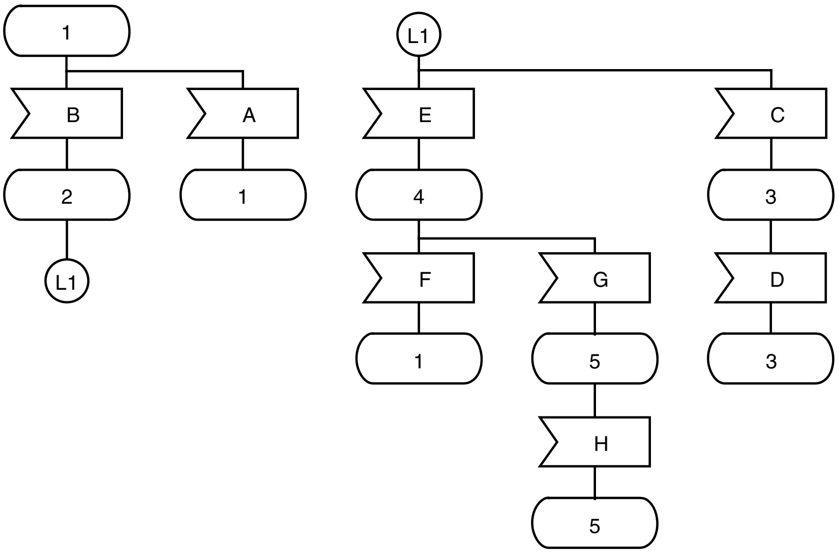


Figure 4.10 Example of a smart card state machine with the following states and transitions:

1 – initial state	2 & 4 – intermediate states	3 & 5 – final states

A – SELECT FILE	E – ASK RANDOM
B – VERIFY	F – all commands except G
C – SELECT FILE	G – EXTERNAL AUTHENTICATE
D – READ BINARY	H – SELECT FILE / UPDATE BINARY

the terminal has been successfully authenticated, the card reaches state 5. In this state, according to the diagram, files may be selected and written using SELECT FILE and UPDATE BINARY.

In this diagram, states 3 and 5 cannot be exited during a session, so they represent the two end states. A transition to state 1 is only possible via a card reset. This is not shown in the diagram, since the 'awareness' of every state machine is limited to its current session. No information at all is transferred from one session to the next within the state machine.

4.5 ERROR DETECTION AND CORRECTION CODES

Whenever data are transmitted or stored, it should be possible to detect any changes to the data. In particular, stored programs must be protected against corruption, since a single altered bit of program code could ruin the program or modify its execution to such an extent that it no longer provides the required functions. The EEPROM memory used in smart cards is especially sensitive to external influences, such as heat and voltage fluctuations. Consequently, the sections that perform security-related functions must be protected so that undesired changes can be detected by the operating system and their negative effects can be avoided.

Very sensitive file contents, such as program code, keys, access conditions, pointer structures and the like, must be protected against alteration. Error detection codes (EDCs) are used for this purpose. The probability of detecting a change in a memory region protected by an EDC depends on the type of code used. Error correction codes (ECCs) are an extension of error detection. They make it possible to not only detect errors in the protected data, but to also correct errors to a limited extent

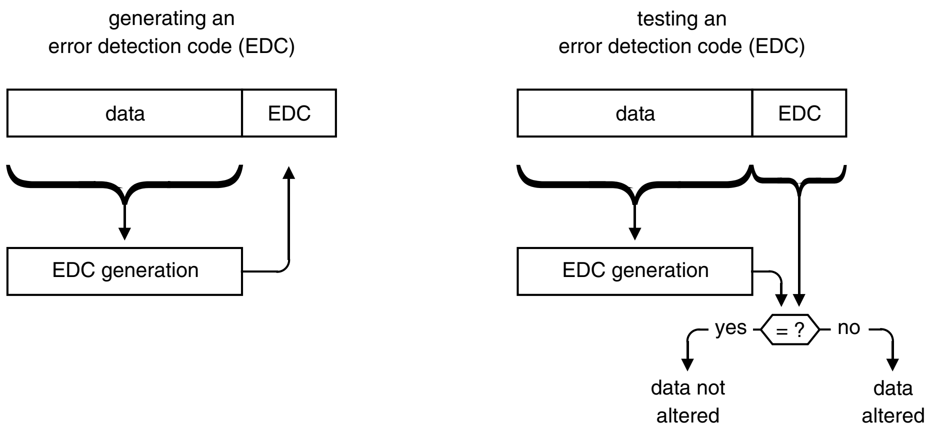


Figure 4.11 The basic processes for generating and checking an error detection code (EDC)

All of these codes work on the principle of assigning a checksum to the protected data. The checksum is usually stored along with the protected data. It is computed using a generally known algorithm, not a secret one. The data can be checked for changes as necessary by using the EDC. This is done by comparing the stored checksum with one computed anew.

A particular aspect of error detection and correction is that it utilizes a wide variety of mathematical procedures. Some of these provide a higher degree of protection for the more significant bits, in order to reduce adverse effects on numerical values as much as possible. In most cases, however, using such algorithms enormously increases the complexity and size of the program code. It is thus more common to use procedures in which error detection does not distinguish between the upper and lower parts of a byte, but instead operates on the byte as a whole.

Error detection and correction codes are very similar to message authentication codes (MACs) and cryptographic checksums (CCSs). However, there is a fundamental difference. EDC and ECC checksums can be computed and checked by anyone. In contrast, the computation of a MAC or CCS requires a secret key, since these codes are designed to protect against manipulation of the data instead of against accidental corruption.

The most widely known type of error detection code is doubtless the parity bit, which is appended to each byte in many data transmission protocols and some types of memory modules. Before computing a parity bit, you must decide whether to use even or odd parity. With even parity, the parity bit is chosen such that the total number of '1' bits in the data byte plus the parity bit is an even number. With odd parity, this total is an odd number.

If two bits in a byte are simultaneously wrong, the parity will not change and no error will be detected. Another drawback of parity-based error detection is the relatively large overhead of one parity bit for every eight data bits. This represents an additional memory load of 12.5 %. Furthermore, it is very difficult to work with supplementary parity bits when the memory is

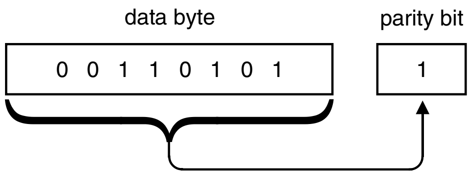


Figure 4.12 Example of error detection using a supplementary odd parity bit

organized in bytes, since this requires significant program overhead. This is why parity bits are not used for error detection in smart card memories. Other methods, such as XOR and CRC checksums, are better suited to this task.

4.5.1 XOR checksums

An XOR checksum, which is also known as a longitudinal redundancy check (LRC) due to how it is computed, can be obtained very simply and very quickly. These are both important criteria for error detection codes used in smart cards. In addition, the algorithm can be implemented extremely easily. Besides protecting data stored in memory, XOR checksums are typically used for data transmission (e.g., ATR with the T = 1 transmission protocol). An XOR checksum is computed by performing consecutive logical XOR operations on all data bytes. In other words, byte 1 is XOR-ed with byte 2, the result of this is XOR-ed with byte 3 and so on.

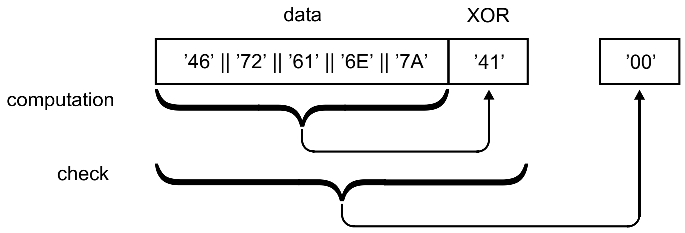


Figure 4.13 Computing and checking an XOR checksum

If the checksum is placed directly after the data and a new checksum is computed using both the data and the first checksum, the result is '00'. This is the simplest way to verify that the data and the checksum still have their original values and thus are uncorrupted.

The primary advantage of XOR checksums is that they can be computed quickly using a simple algorithm. The algorithm is so simple that its assembler code is only 10 to 20 bytes long. One reason for this is that the XOR operation is directly available in all processors as a machine instruction. In addition, an algorithm for XOR checksum computation must be implemented in almost every smart card operating system, due to the requirements of various ISO standards relating to data transmission using the T = 1 protocol. This algorithm can be used for other purposes without any additional overhead.

Unfortunately, XOR checksums also suffer from several serious drawbacks, which considerably limit their practical application. They are in principle not very secure. For example, they do not allow the interchange of two bytes within the overall data to be detected. Also, multiple errors can occur at the same position in several bytes and cancel each other out. The consequence of all this is that XOR checksums are mainly used for data transmission; they are used only to a very limited extent to verify the consistency of memory contents. XOR checksums can be computed very quickly, since specific machine instructions for this purpose are generally available. With a typical 8-bit smart card microcontroller clocked at 3.5 MHz, a rate of 1 µs/byte can be achieved, which corresponds to a throughput of 1 MB/s.

4.5.2 CRC checksums

The CRC method (cyclic redundancy check) also comes from the field of data communications, but it is significantly better than the XOR method. Still, a CRC checksum is also only an error detection code, so it cannot be used for error correction. The CRC method has been used for a long time in data transmission protocols, such as Xmodem, Zmodem and Kermit, and it is widely used in hard disk drive controllers in a hardware implementation. It is based on the CCITT V.41 recommendation. An additional standard for CRC checksums is ISO/IEC 13 329.

A CRC-16 checksum is generated by a 16-bit cyclic feedback shift register, while a 32-bit shift register is used to generate a CRC-32 checksum. The following description refers only to the CRC-16 checksum, since this is the most commonly used CRC checksum method for smart cards. The feedback in the shift register is determined by a generating polynomial. In mathematical terms, the data to be checked are represented as a large number, which is divided by the generating polynomial. The remainder from this division is the checksum. The CRC-16 method (that is, a 16-bit CRC) should only be used with data volumes up to 4 kB, since the error detection probability drops sharply beyond this point. However, this restriction can easily be circumvented by dividing the data into blocks that are no larger than 4 kB. Alternatively, a CRC-32 method (32-bit CRC checksum) can be used, which allows single-bit errors to be detected in up to 4 GB of data.

Table 4.6 Commonly used generator polynomials for CRC-16 computation

Designation	Generator polynomial
CRC CCITT V.41, ISO/IEC 3309	$G(x) = x^{16} + x^{12} + x^5 + 1$
CRC-16	$G(x) = x^{16} + x^{15} + x^2 + 1$
CRC-12	$G(x) = x^{12} + x^{11} + x^3 + x^2 + x + 1$

With a CRC checksum, it is always necessary to know the generating polynomial as well as the initial value for the shift register, since otherwise the computation cannot be reproduced. In the overwhelming majority of cases (e.g. ISO 3309), the initial value for the shift register is zero, but several data transmission protocols (such as CCITT Recommendation X.25) set all bits to 1.

The computation of a CRC checksum, as illustrated in Figure 4.14, proceeds as follows: (a) the 16-bit CRC register is set to its initial value; (b) the data bits are fed into the feedback shift register one after the other, starting with the least-significant bit; and (c) the feedback (which represents the polynomial division) takes place via bitwise logical XOR operations on

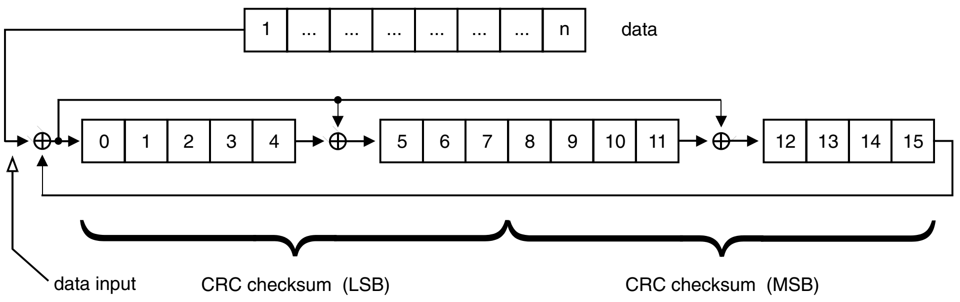


Figure 4.14 Calculating a CRC-16 checksum with a generator polynomial $G(x) = x^{16} + x^{12} + x^5 + 1$. The data and the CRC register are both shown as bits

the CRC bits. After all data bits have been fed into the register, the computation is complete and the content of the 16 bits is the desired CRC checksum.

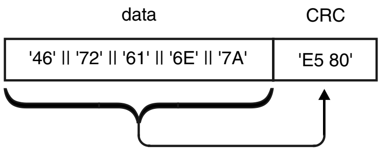


Figure 4.15 Sample computation of a CRC checksum using the generator polynomial $G(x) = x^{16} + x^{12} + x^5 + 1$ and an initial value of '0000'

A CRC checksum can be verified by again calculating the CRC checksum of the data and comparing the result with the checksum provided with the data. If they are the same, it follows that the data and the checksum have not been altered.

The major advantage of CRC checksums is that they provide reliable error detection, even with multiple errors. Only very few methods can achieve this. In addition, in contrast to the XOR method, CRC allows interchanged data bytes to be detected, since byte order definitely plays a role in checksum generation via the feedback shift register. However, it is very difficult to specify exact detection probabilities for such errors, since they are very dependent on the locations of the errors within the bytes in question.

The CRC algorithm is relatively simple, and the amount of code needed to implement it thus matches the needs of small smart card memories. Its greatest drawback is the slowness of the computation, since the algorithm requires the data to be shifted bit by bit. The CRC checksum algorithm was originally designed for hardware implementation, and this has a strong detrimental effect when it is implemented in software. The throughput of a CRC-16 routine is lower than that of an XOR checksum routine by a factor of around 200. A typical figure is 0.2 ms/byte at a 3.5-MHz clock frequency, which corresponds to a throughput of 5000 byte/s. Computing a CRC checksum for a 10-kB smart card ROM would thus require around 2 seconds.

Many types of microcontrollers have a special component for hardware-assisted generation of CRC checksums for definable memory regions. The rates that can thereby be achieved can be as high as one byte per clock cycle. For a microcontroller clocked at 5 MHz without an internal clock divider, this would yield a throughput of 5 MB/s.

4.5.3 Reed–Solomon codes

In 1960, the mathematicians Irving S. Reed and Gustave Solomon published a paper with the title 'Polynomial Codes over Certain Finite Fields', which forms the basis for what has become one of the most widely used methods for error detection and correction. Reed–Solomon codes (RS codes), which are named after their inventors, are used for error detection and correction for data storage (e.g. bar codes, DAT, CD and DVD) and data transmission (e.g. DSL, satellite communications and space probes).

Reed–Solomon codes are block-oriented error correction codes (ECCs) that can also correct burst errors. They are computed using the arithmetic of finite bodies (Galois fields, or GF). The characteristics of a Reed–Solomon code, in terms of the number of detectable and correctable errors for a specific data length, can be adapted to a particular application via the choice of generator polynomial.

For several years now, Reed–Solomon codes have been used by various smart card operating systems for error detection, and quite rarely for error correction, in order to secure data stored in EEPROM. The generator polynomials used are matched to the properties of EEPROM storage with regard to the occurrence of burst errors, thus yielding significantly more secure error detection than what is possible using the CRC method.

For example, with an RS code using a 2^8 GF and two supplementary bytes in addition to the data to be protected, it is possible to detect two incorrect bytes or correct one incorrect byte. With three supplementary bytes, three incorrect bytes can be detected or one incorrect byte can be corrected, and with four supplementary bytes, four incorrect bytes can be detected or two incorrect bytes can be corrected. The size of the executable code in 8051 assembler is approximately 100 bytes, and the computation rate is approximately 10 ms/byte at 3.57 MHz. RS codes are also quite suitable for implementation in hardware.

4.5.4 Error correction

If it is necessary to not only detect changes in memory regions, but also to correct them if possible if they result from errors, error correction codes must be used. Since computing such codes is costly in terms of program code, using them to protect smart card memory is problematic. Furthermore, the algorithms in question are usually designed to correct only low error rates. Since EEPROM memory in smart cards is page-oriented, a whole page usually fails in the event of an error, so only methods that are capable of correcting burst errors are worth considering. Consequently, a different approach is taken for error correction.

The technically simplest solution is to store the data in multiple, physically separate memory pages and use a majority-vote procedure when reading the data. Triple storage is commonly used, together with a 2-of-3 vote. A less memory-intensive variation of this method is to store the data in two locations with EDC checksum protection for each location. The occurrence of a memory error can be detected by checking the two EDC values. This also allows the memory region where the error occurred to be identified. The region with no detected error must then contain the valid data, which can be restored to the faulty region.

Of course, a significant amount of extra memory is needed for these error correction methods, but for small quantities of data it is still well within acceptable limits. The main advantage is that no complicated, code-intensive algorithm is needed to evaluate the data.

As an alternative to protection by multiple data storage, it is possible to use an error correction algorithm, such as the Reed–Solomon algorithm. This is particularly suitable for use with clustered errors, such as may occur in smart cards due to page failures. The algorithm occupies a few hundred bytes of memory when programmed in assembly language, and the size of the ECC data depends primarily on probability with which errors must be detected and/or properly corrected.

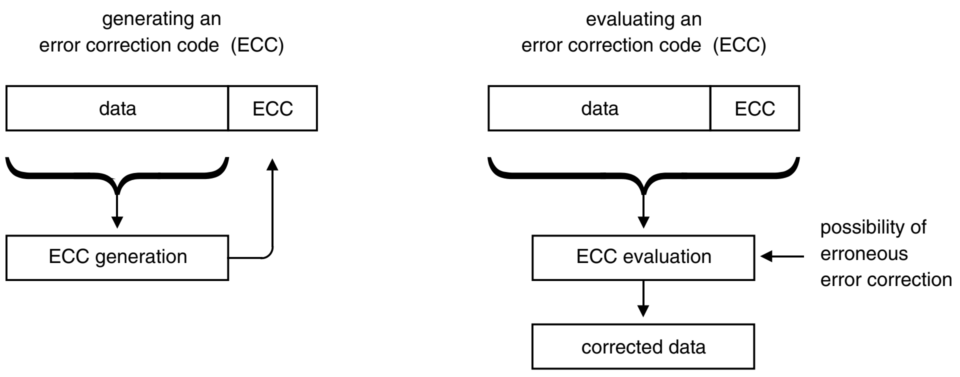


Figure 4.16 The basic principle of using an error correction code (ECC)

Several basic remarks are in order here with regard to using error correction methods in smart cards. At first glance, it may be tempting to use these methods to correct errors that occur in the EEPROM. However, the presumed data security is bought at the price of several serious drawbacks. The required amount of memory space is enormous, and the time required to write data to memory also increases considerably, since data must be stored in multiple locations. Algorithms that can correct clustered errors on the scale at which they typically occur with page-based EEPROMs are complicated and require a large amount of memory space for the EDCs. However, there is an even more serious fundamental drawback. Even when an error correction algorithm is used, errors can in principle still be present in the corrected data, since the algorithm works properly only up to a certain number of errors. If an operating system corrects memory errors automatically, in principle it is never possible to be certain that the correction was made properly.

For example, suppose that automatic error correction is applied to the balance in an electronic purse. The system operator can never be sure of what will happen to the credited amounts in the event of an error. The balance may be corrected properly, but there is a certain probability that it will be too high or too low after the correction. In this regard, it must be remembered that smart cards are inexpensive mass-produced articles, which can simply be replaced if they are faulty.

As a rule, when problems occur with data contents, a higher level system that allows human intervention must decide what to do. For example, on the first instance of an error occurring in a smart card purse, the cardholder's balance will most likely be manually restored. However, if the error recurs repeatedly, the system operator will be much less forthcoming with regard to the cardholder, since there is a possibility that the EEPROM has been fraudulently manipulated. This cannot be handled by an error correction code in the card; instead, the system administrator must intervene.

4.6 DATA COMPRESSION

As is well known, the amount of memory available in a smart card is severely limited. Consequently, the desire to improve this situation by using data compression repeatedly arises among application providers.

There are certain hurdles that must be overcome before data compression can be used. The algorithm must not take up too much memory space, and in particular it must require very little RAM. In addition, an acceptable compression speed should be achieved. The compression factor is not all that important, since the data volume is always only a few hundred bytes at most. The methods that are frequently used for smart cards are run-length encoding and variable-length encoding.

With run-length encoding, a contiguous string of identical data objects is replaced by the combination of a repetition count and the object (such as a character) to be repeated. With variable-length encoding, the frequency of occurrence of characters having a fixed length (e.g., one byte) is analyzed, and the most frequently occurring characters are replaced by characters with shorter lengths (Huffman algorithm). Less frequently occurring characters are encoded using longer codes.

With static variable-length encoding, replacements are made using a previously defined table. The dynamic version of variable-length encoding first analyzes the frequency distribution of the characters in the original data and then constructs a replacement table based on the results of this analysis. A third variation is adaptive variable-length encoding, in which the replacement table is continuously updated during the compression process to achieve optimum compression.

Both dynamic and adaptive variable-length encoding are out of the question for smart cards, due to the complexity of their algorithms and their large memory requirements. Run-length encoding and static variable-length encoding are thus the only real alternatives for use in smart cards. The algorithm for run-length encoding does not need much program code, but it has the drawback that it can only be used with repetitive data. Image data, for example, are particularly suitable, since images often contain large areas with the same value. Keys for symmetric cryptographic algorithms would be completely unsuitable for compression with this algorithm, since they have the characteristics of random numbers.

Static variable-length encoding is the second compression method used with smart cards. It is quite suitable for files containing telephone directory information, for instance, since the structure of the stored data is known and the replacement table can be permanently built into the algorithm. Telephone numbers consist of only the numerals 0 through 9 and a few special characters, such as ‘*’ and ‘#’. If only capital letters are allowed for names, the replacement table only has to accommodate the 26 characters of the alphabet. Furthermore, certain letters occur significantly less often in names than do others, which also affects the encoding. With telephone directories, a memory space reduction of 30 % (compared with the uncompressed data) can certainly be achieved, although this does not take into account the memory occupied by the compression algorithm.

However, certain things must be considered with regard to data compression for smart cards. Ideally, data compression should be performed in the operating system in a manner that is fully transparent to the outside world, such that uncompressed data can be read and written in the usual way using standard commands. Compression can also only be applied to certain types of data. The results of attempting to compress program code and keys are usually unsatisfactory. This must be taken into account in the design of the application, since otherwise the anticipated

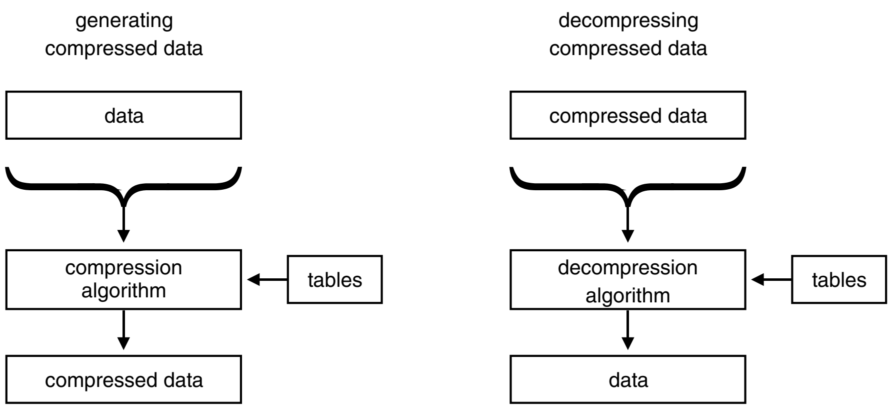


Figure 4.17 The basic principle of data compression for stored data

reduction in memory space can (in the worst case) turn into a need for even more memory as the result of 'compression'.

For all of these reasons, data compression has been used only sparingly in smart cards up to now. In special applications, such as telephone directories in cards used in the telecommunications sector, compression algorithms are sometimes used. With general-purpose operating systems and applications in which the structure of the data is not known in advance, using data compression does not produce satisfactory results. It should thus be avoided, due to the additional memory space required by the compression algorithm.

4.7 CRYPTOLOGY

In addition to their function as data storage media, smart cards are also used as authorization media and encryption modules. As a result, cryptography achieved a central significance in the early days of smart cards. Nowadays, the procedures and methods of this discipline are firmly established components of smart card technology.

Cryptology can be split into two areas of activity, namely cryptography and cryptanalysis. Cryptography is the study of the methods used for encrypting and decrypting data, while cryptanalysis is concerned with attempting to break existing cryptographic systems.

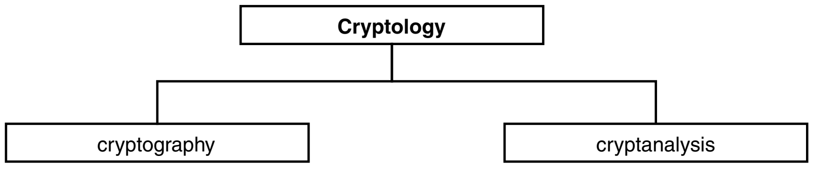


Figure 4.18 The two subdivisions of cryptology are cryptography and cryptanalysis

In the smart card realm, the practical use of existing cryptographic procedures and methods represents the principal task and primary application area with regard to cryptography. Consequently, here we concentrate more on the practical aspects of cryptography than on the theoretical aspects. However, we do not entirely neglect the application of the procedures and the basic features of their theoretical foundations.

The four objectives of cryptography are maintaining the secrecy of messages (*confidentiality*), ensuring the *integrity* and the *authenticity* of messages and ensuring the binding force (*non-repudiation*) of messages. These objectives are mutually independent, and they place different demands on the system in question. Confidentiality means that only the intended recipient of a message can decrypt its contents. Authenticity means that the recipient can verify that the received message has not been altered in the course of being transmitted. Non-repudiation means that the sender can verify that a certain recipient has received a particular message, which means that the message has binding force.

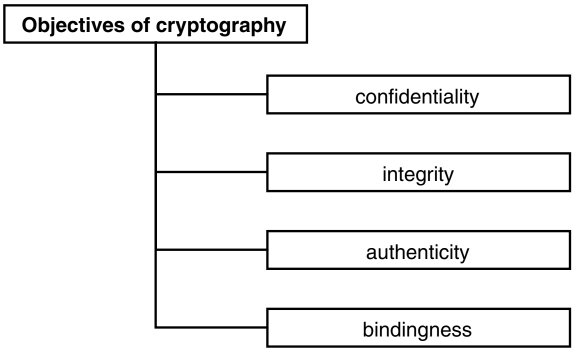


Figure 4.19 Classification of the four independent objectives of cryptography

The notation used in this book for cryptographic procedures is illustrated in Figures 4.20 and 4.21. The terms and principles described below form the basis for cryptology and are a prerequisite for understanding the procedures described in the rest of this section.

In simplified terms, there are three types of data in encryption technology. The first is *plaintext*, which is unencrypted data. Encrypted data is referred to as *ciphertext*. Finally there is a *key*, one or more of which is required for encryption and decryption. These three types of data are processed by an encryption algorithm. The algorithms that are currently used in smart cards are generally block-oriented, which means that the plaintext and ciphertext can only be processed in packets with fixed lengths (such as 8 bytes with DES).

Modern cryptographic algorithms are generally based on Kerckhoff's principle. This principle, which is named after Auguste Kerckhoff (1835–1903), says that the entire security of an algorithm should be based only on the secrecy of the key, and not on the secrecy of the cryptographic algorithm. The consequence of this generally known but often-disregarded principle is that many algorithms used in the civil sector have been published, and in part also standardized.

The opposite of Kerckhoff's principle is the principle of security by concealment. With this principle, the security of a system is based on the idea that a would-be attacker does not know how the system works. This principle is very old, and it is still frequently used even today. However, you should take care not to develop a cryptographic system (or any other system) based on this principle alone. Up to now, every system based on this principle alone has been broken, usually in a very short time. In our information society, it is generally not possible to

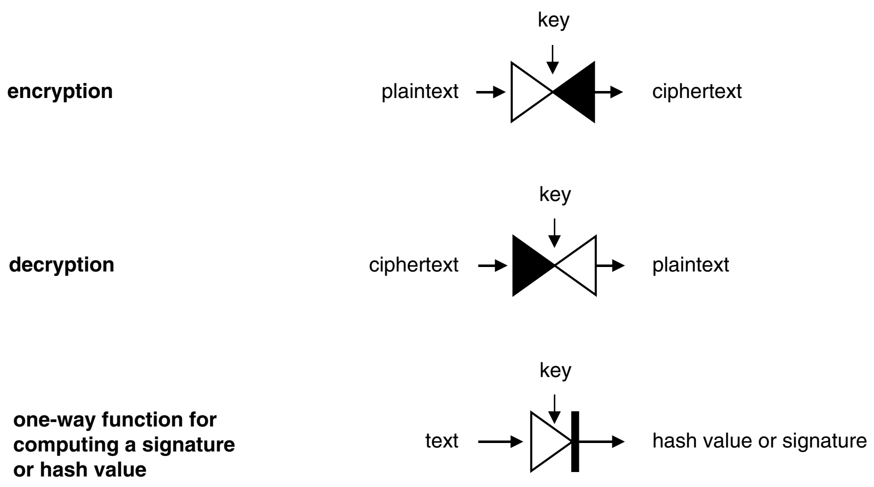


Figure 4.20 The symbols used in this book for cryptographic algorithms

keep the technical details of a system secret for a long time, and that is precisely what this principle requires.

Of course, the consequences of the unintentional interception of messages can most certainly be limited by using concealment. This principle is thus repeatedly used in combination with Kerckhoff's principle. In many large systems, it is also incorporated as a supplementary security level. Since the security of modern, published cryptographic algorithms is primarily based only on the limiting processing capacity of current computers, concealing the procedure that is used increases the level of protection against attacks.

If you rely only on the protection provided by the assumption that a potential attacker does not have access to sufficient processing power, you may be quickly overtaken by the rapid pace of technical progress. Statements such as 'it would take a thousand years to break this cryptographic system' are unreliable, since they are based on currently available processing capacities and algorithms. They cannot take future developments into account, since such developments are generally unknown. The arithmetic processing capacities of processors double around every 18 months, which means that the capacity per processor has increased by a factor of approximately 25,000 over the last 25 years.

Recently, the increased degree of networking of computers has created another option for mounting serious attacks on keys or cryptographic systems. For instance, a request to help break a DSS key, if posted on the Internet, would be forwarded to millions of users by the snowball effect. If only 1 % of all current users[2] participated in such an action, the potential attacker would have access to a parallel computer composed of 300,000 individual computers.

Cryptographic algorithms are divided into two types: *symmetric* and *asymmetric*. This division is based on the key that is used. Here 'symmetric' means that the algorithm uses the same key for encryption and decryption. By contrast, asymmetric algorithms (which were

[2] In the summer of 2001, it was assumed that there were around 500 million users of the Internet

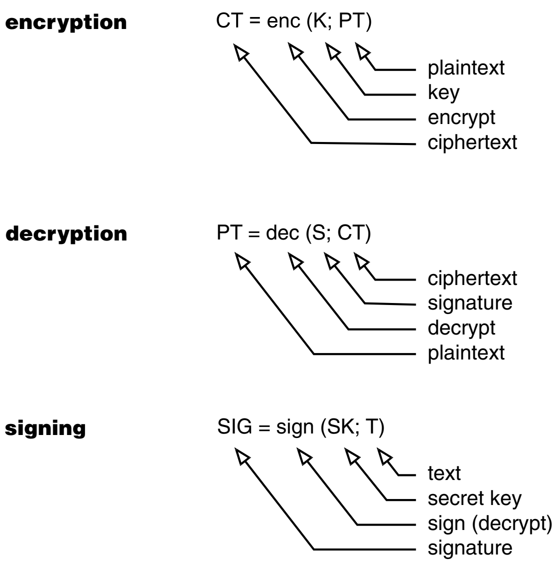


Figure 4.21 The notation used in this book for cryptographic procedures

postulated in 1976 by Whitfield Diffie and Martin E. Hellman) use different keys for encryption and decryption.

A term that often comes up in connection with cryptographic algorithms is the magnitude of the key space. This refers to the number of possible keys that can be used with a particular cryptographic algorithm. A large key space is one of several criteria for a secure cryptographic algorithm.

A requirement that has only recently become prominent with regard to the technical implementation of cryptographic algorithms in smart cards is freedom from noise. In this context, this means that the execution time of the algorithm must not depend on the key or the plaintext and ciphertext. If this requirement is not met, it could be possible to discover the key in a relatively short time, which would mean that the entire cryptographic system was broken.

In cryptology, there is a strong distinction between the theoretical and practical security of a system or an algorithm. A system is *theoretically* secure if an attacker, given unlimited time and technical resources, cannot break the system. This means that even if an attacker would need 100 years and the aid of several supercomputers to break a system, it could not be considered to be theoretically secure. If a system cannot be broken when the attacker has only a limited amount of time and technical resources, it is considered to be *practically* secure.

A cryptographic system can assure the confidentiality and/or authenticity of a message. If the system has been broken, this means that confidentiality and/or authenticity are no longer guaranteed. If an attacker can discover the secret key of an encryption algorithm, for example, he can then decrypt data that have been protected by being encrypted, in order to learn their content and modify them as desired.

Several different methods of attack can be used to break the key of a cryptographic algorithm. In a 'ciphertext-only' attack, the attacker knows only the ciphertext and attempts to determine

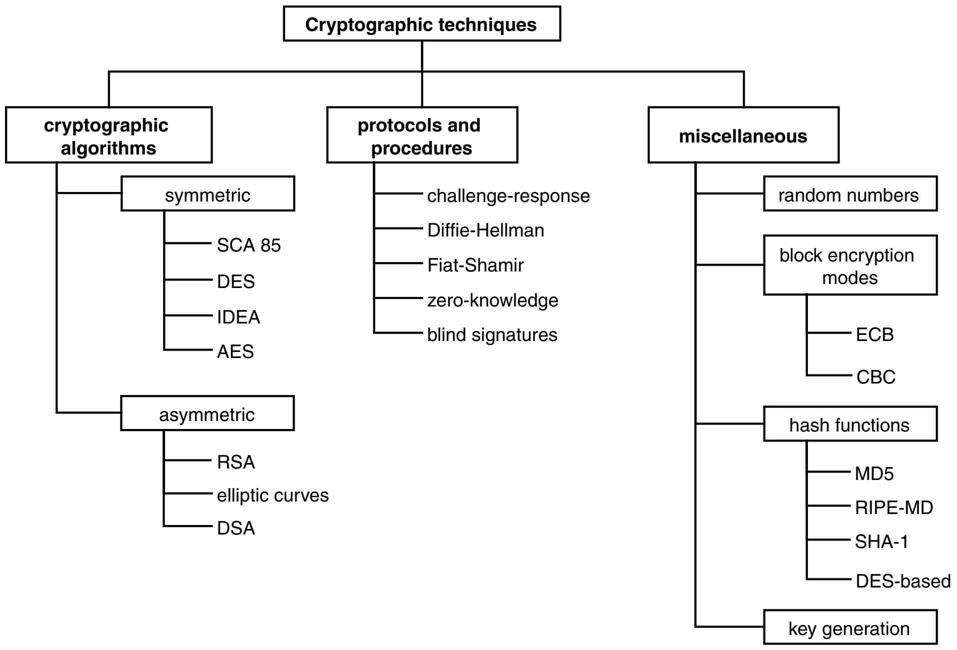


Figure 4.22 Classification chart for cryptographic techniques used in the smart card area

the key or plaintext from the ciphertext. A more promising method of attack is the 'known-plaintext' attack, which involves the attacker knowing several plaintext–ciphertext pairs for a secret key. The 'chosen-plaintext' and 'chosen-ciphertext' attacks require the attacker to be able to generate his own plaintext–ciphertext pairs. If this is possible, the likelihood of success is improved, since the secret key can be discovered experimentally.

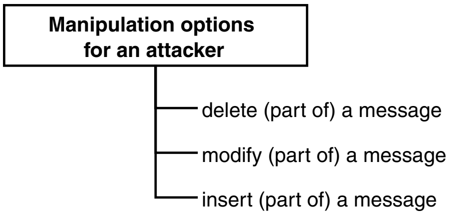

Manipulation options for an attacker
delete (part of) a message
modify (part of) a message
insert (part of) a message

Figure 4.23 Classification of the various manipulation options available to an attacker

Discovering a key by trial and error (a 'brute-force' attack) is naturally the least sophisticated method of attack. With this method, an attempt is made to find out the correct key by employing a large amount of processing capacity to test all possible keys with a know plaintext–ciphertext pair. Obviously, a processing capacity in the supercomputer range is normally a prerequisite for this method. Statistically seen, on average half of the possible keys must be tested before the right one is found. Naturally, a large key space considerably increases the difficulty of such an attack.

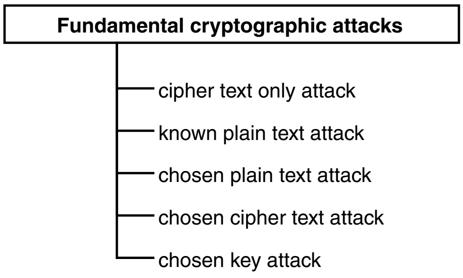


Figure 4.24 Classification of the basic methods of cryptographic attack

4.7.1 Symmetric cryptographic algorithms

Symmetric cryptographic algorithms are based on the principle of performing encryption and decryption using the same secret key – hence the designation 'symmetric'.

The DES algorithm

The best-known and most widely used type of symmetric cryptographic algorithm is the Data Encryption Algorithm (DEA). It was developed by IBM in collaboration with the NBS (US National Bureau of Standards) and published in 1977 as the US FIPS 46 standard. The standard that describes the DEA is generally referred to as the DES (Data Encryption Standard). Consequently, the Data Encryption Algorithm is often (but not entirely correctly) called the DES algorithm.

Since this algorithm is of course fashioned according to Kerckhoff's principle, it could be published without losing any of its security. However, even today not all of the development criteria have been made known, which repeatedly leads to assumptions regarding possible methods of attack and the possible presence of 'trap doors'. However, up to now all attempts to break the algorithm on this basis have failed.

Two important principles for a good encryption algorithm were incorporated into the design of the DES algorithm. These are the principles of *confusion* and *diffusion*, as first proposed by C. Shannon. The confusion principle states that the statistics of the ciphertext should affect the statistics of the plaintext in a manner that is so complex that an attacker can derive no advantage from them. The diffusion principle states that each bit of the plaintext and each bit of the key should affect as many bits of the ciphertext as possible.

DES, which is a symmetric block encryption algorithm, does not expand the ciphertext. This means that the plaintext and ciphertext blocks have the same size. The block size is 64 bits (8 bytes), which is also the length of the key, although only 56 of these bits are used as the actual key. The key contains eight parity bits, which reduces the available key space. The 64 bits of the key are numbered consecutively from left (msb) to right (lsb). Bits 8, 16, 24, ..., 64 are the parity bits. The parity is always odd. Due to the parity bits, the key space is 2^{56}, which means that there are approximately 7.2×10^{16} possible keys. At first glance, a key space with 72,057,594,037,927,936 possible keys may appear very large, but the limited size of its

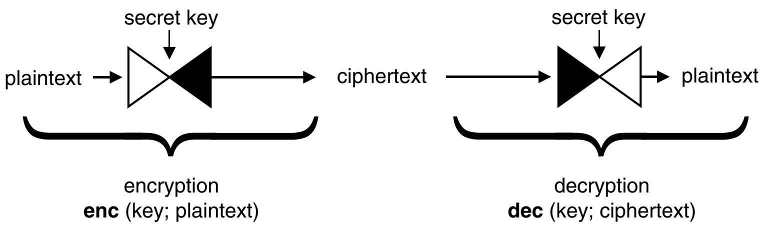


Figure 4.25 Operating principle of a symmetric cryptographic algorithm for encrypting and decrypting data. The DES is a typical example of this type of cryptographic algorithm

key space is actually the main weakness of the DES algorithm.[3] Given the steadily increasing processing capacity of modern computers, a key space of this size is already considered to be too small for a secure cryptographic algorithm. If a plaintext–ciphertext pair is available, it is easy to test all possible keys if the key space is too small.

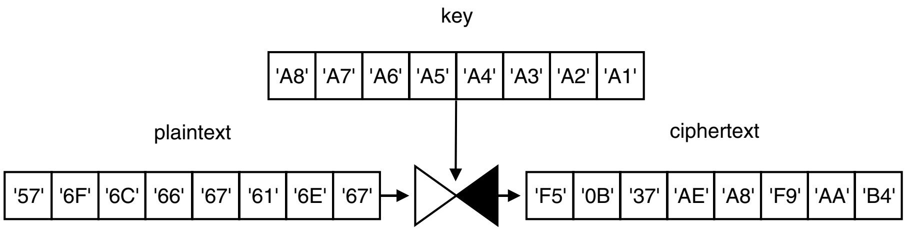

key
'A8' 'A7' 'A6' 'A5' 'A4' 'A3' 'A2' 'A1'
plaintext
'57' '6F' '6C' '66' '67' '61' '6E' '67'
ciphertext
'F5' '0B' '37' 'AE' 'A8' 'F9' 'AA' 'B4'

Figure 4.26 The operation of the DES algorithm for encrypting data

If a plaintext–ciphertext pair is obtained by tapping the communications between a terminal and a smart card, a brute-force attack can be mounted by encrypting the plaintext using all possible keys. The correct key can be determined by comparing all of the resulting ciphertexts with the previously obtained ciphertext. This procedure can very easily be executed using several parallel processors. Each of the processors tests only the relatively small portion of the key space assigned to it. A sample calculation can illustrate the amount of time needed for such a brute-force attack. The fastest currently available DES components require 64 ns for a complete block encryption.[4] If 10,000 such computation modules are assembled in parallel, each one can independently test a small part of the key space. Assuming that on average only half of the key space must be searched to find the correct key, the processing time can be calculated as follows:

$$((2^{56} \times 64\ \text{ns}) \div 10{,}000) \times 0.5 \approx 64\ \text{h}$$

In 1993, Michael Wiener published plans for a million-dollar computer that could test all the DES keys of a given plaintext–ciphertext pair within seven hours [Wiener 93]. In 1998, a

[3] To get a feeling for the size of this large number, you might enjoy considering the following comparison. The mass of the earth is roughly 5.974 $\times$ 10^{27} grams. Based on this value, the number of electrons, protons and neutrons that make up the earth can be taken to be around 10^{52}. If only a single elementary particle were needed to store a bit, then at most 10^{52} bits could be stored in a memory with the mass of the earth

[4] Using a DEC gate array built with gallium-arsenide technology that operates in the ECB and CBC modes

similar concept, called ‘DES Cracker’, was implemented by the Electronic Frontier Foundation (EFF) at a cost of approximately US $ 250,000. The DES Cracker can test approximately 88.8 billion keys per second, and in a public test it took 56 hours to determine an unknown key [EFF 98]. The biggest problem with the use of DES is that its key space has become too small over time. Consequently, the triple-DES algorithm is used almost exclusively for new applications.

A full description of the exact implementation of the DES algorithm is far beyond the scope of this book, and it is not necessary for understanding the subject. If you are interested in having more detailed information, consult FIPS Publication 46, Carl Meyer [Meyer 82] or Bruce Schneier [Schneier 96]. However, one significant detail must be mentioned. The DES was designed as an encryption algorithm that can easily be implemented in hardware. There are presently many smart card microcontrollers with a DES hardware module. However, if it is necessary to implement DES in a smart card in software, it will occupy around 1 kB of assembler code, even in a highly optimized version. The size of a DPA-resistant version is approximately 2 kB. Its computational speed is consequently rather low.

Typical computation times for encryption and decryption using a smart card, with comparison values for using a PC and a hardware integrated circuit, are shown in Table 4.7. These numbers can vary depending on the actual implementation, and they take into account only the pure processing time for DES encryption or decryption of an 8-byte block, assuming that all registers are already loaded. As a general rule, it can be assumed that DES implemented in hardware is approximately 150 times faster than DES in software.

Keys for the DES algorithm can be generated using a random number generator that produces an 8-byte random number, which is then checked against the four weak and 12 semi-weak keys. If the computed value does not match any of these easily broken keys, the parity bits are computed and the result is a DES key.

Table 4.7 Typical DES computation times (8-byte block)

Implementation	Computation time / throughput
Smart card, 3.5-MHz clock, software implementation	17.0 ms / 3.8 kbit/s
Smart card, 3.5-MHz clock and DES processing unit	112 µs / 571 kbit/s
Smart card, 3.5-MHz clock and triple-DES processing unit	130 µs / 492 kbit/s
Smart card, 4.9-MHz clock, software implementation	12.0 ms / 5.3 kbit/s
Smart card, 4.9-MHz clock and DES processing unit	80 µs / 800 kbit/s
Smart card, 4.9-MHz clock and triple-DES processing unit	93 µs / 688 kbit/s
PC (80486, 33 MHz)	30 µs / 2.1 MB/s
PC (Pentium, 90 MHz)	16 µs / 4 MB/s
PC (Pentium, 200 MHz)	4 µs / 16 MB/s
DES integrated circuit	64 ns / 100 MB/s

IDEA algorithm

There are many other symmetric cryptographic algorithms besides DES. Here we consider only the International Data Encryption Algorithm (IDEA) as a representative example. It was developed by Xuejia Lai and James L. Massey, and it was published in 1990 as the ‘proposed encryption standard' (PES). It was improved in 1991, and for a short while the improved version was known as the 'improved proposed encryption standard' (IPES), but nowadays it is commonly known as IDEA. The development criteria and internal construction of this algorithm have been fully published, so Kerckhoff's principle is satisfied. However, IDEA is subject to patent restrictions, which was formerly also the case with the RSA algorithm.

IDEA, like DES, is a block-oriented cryptographic algorithm, and it also uses 8-byte plaintext and ciphertext blocks. In contrast to the DES algorithm, the key length is 16 bytes (2×8 bytes). This provides a significantly larger key space, with a size of $2^{128} \approx 3.4 \times 10^{38}$. In regular decimal notation, the number of possible keys for the IDEA is exactly 340,282,366,920,938,463,463,374,607,431,768,211,456.

Due to its structure, IDEA is compatible with DES except for its extended key length. It is also compatible with triple-DES systems, which use keys that are 2×56 bits long, which means that changing the algorithm used does not affect the lengths of the keys or the input and output data blocks. Of course, compatibility in this regard does not mean that DES-encrypted data can be decrypted using the IDEA. In general, IDEA is considered to be a very good cryptographic algorithm. It has also been widely distributed in the form of the public-domain program PGP (Pretty Good Privacy) from Philip Zimmermann, which is used for secure data transmission.

There are only a few smart card implementations of IDEA. The amount of memory space needed for the program is around 1000 bytes. Typical computation times for encryption and decryption are somewhat less than for DES. However, in the development of IDEA, it was assumed that the computations would be executed by a 16-bit processor. Since most smart cards still have 8-bit processors, the speed advantage in comparison with DES is not as great as might be expected. Table 4.8 lists typical values for IDEA operations on an 8-byte block, assuming that previously computed keys are available.

Table 4.8 Typical IDEA computation times (8-byte block)

Implementation	Computation time
Smart card with 3.5-MHz clock, software implementation	12.3 ms
Smart card with 4.9-MHz clock, software implementation	8.8 ms
PC (80386, 33 MHz)	70 μs
PC (Pentium Pro, 180 MHz)	4 μs
IDEA integrated circuit	370 ns

AES algorithm

The amount of processing capacity available at the end of the 1990s, as the result of technical progress and the international networking of computers, was sufficient to allow even ambitious private individuals with organizational abilities to mount successful brute-force attacks on the DES algorithm. As a result, the future viability of DES became so questionable that the relevant national authorities increasingly devoted their attention to specifying a new symmetrical cryptographic algorithm as the official successor to DES. In light of the never fully extinguished doubts about the integrity of DES arising from the fact that not all of the design criteria were made available for public inspection, in 1997 the US National Institute of Standards and Technology (NIST) requested the international cryptographic community to submit possible successors to DES, accompanied by all associated documentation, to the NIST for a sort of competitive evaluation. In 1998, the NIST announced that there were 15 candidates for the Advanced Encryption Standard (AES) and published all of the documentation so that it could be studied with regard to informatics and cryptanalytical aspects by the US National Security Agency (NSA), research institutes, experts and interested parties. Following detailed investigations of these methods and broad public discussion of the results, a candidate for the successor to the DES algorithm was announced by the NIST in 2000.[5] This algorithm was developed by the Belgian cryptologists Joan Daemen and Vincent Rijmen and was already known as the Rijndael algorithm prior to the announcement of the competition.

AES is a symmetric block encryption algorithm with a block length of 128 bits (16 bytes) that can be used with three different key lengths: 128 bits (16 bytes), 192 bits (24 bytes) and 256 bits (32 bytes). It is thus referred to as AES-128, AES-192 or AES-256, depending on the key length. AES can be implemented in hardware logic, and good software implementations are also possible using relatively low-performance 8-bit processors as well as high-performance 16- and 32-bit processors. Furthermore, it can be used throughout the world free of licensing fees, and according to the official pronouncement its anticipated useful life is more than 20 years. AES is standardized by FIPS 197, which is available free of charge via the Internet [NIST].

The size of the key space of the AES algorithm with a 128-bit key is approximately 3.4×10^{38} (2^{128}), which makes it a factor of 4.7×10^{21} greater than the key space of DES with a 56-bit key. The large key space obtained with a key length of only 128 bits enormously increases the difficulty of attacks involving successively testing all possible keys. The software implementation of AES in a DPA-resistant form in a smart card occupies approximately 4 kB of ROM.

Table 4.9 Typical computation times for AES using a 128-bit key (16-byte block)

Implementation	Computation time
Smart card, 16-bit CPU, 4.9-MHz clock, software implementation; encryption	20 ms
Smart card, 16-bit CPU, 4.9-MHz clock, software implementation; decryption	25 ms

Operating modes for block-oriented encryption algorithms

The DES algorithm, like every block-oriented encryption algorithm, can be used in four different operating modes that are standardized in ISO 8372. Two of these operating modes, the CFB and OFB modes, are especially suitable for sequential text with no block structure. The other two, the ECB and CBC modes, are based on a block size of 8 bytes. These two block-oriented modes are most commonly used in smart card applications.

The basic operating mode of the DES algorithm is designated as the ECB (electronic code book) mode. In this mode, 8-byte plaintext blocks are independently encrypted using a single key. This is the DES algorithm in its pure form, without extensions.

[5] A good summary of the selection process and selection criteria can be found in [Nechvatal 00]

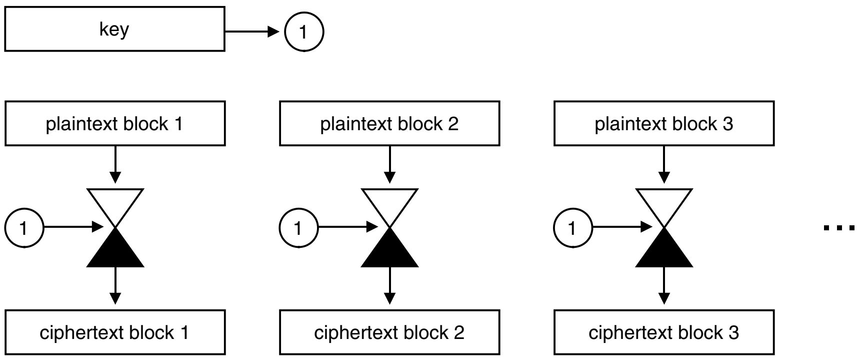


Figure 4.27 Data encryption with a block-oriented encryption algorithm operating in the ECB mode. Decryption is performed in a similar manner

The second block-oriented mode is known as the CBC (cipher block chaining) mode. In this mode, a data string consisting of several blocks is chained using XOR operations during the encryption such that each block becomes dependent on the block preceding it. This makes it possible to reliably detect interchanging, adding or deleting encrypted blocks. This is not possible with the ECB mode. If the plaintext blocks are suitably structured (with sequence counters in their headers or initialization vectors), a consequence of CBC chaining is that even identical plaintext blocks are converted into non-identical ciphertext blocks. This makes the cryptanalysis of intercepted data much more difficult, since codebook analysis (for example) is not possible.

The first plaintext block is XOR-ed with an initialization vector (often called an IV), and then encrypted with the DES algorithm. The result is the ciphertext, which is in turn XOR-ed with following plaintext block. The process continues in this manner with the following blocks. As a rule, the initialization vector is preset to null. However, in some systems, a session-specific random number is written to the initialization vector as a substitute for a temporary key. This number must naturally be known when the data are subsequently deciphered.

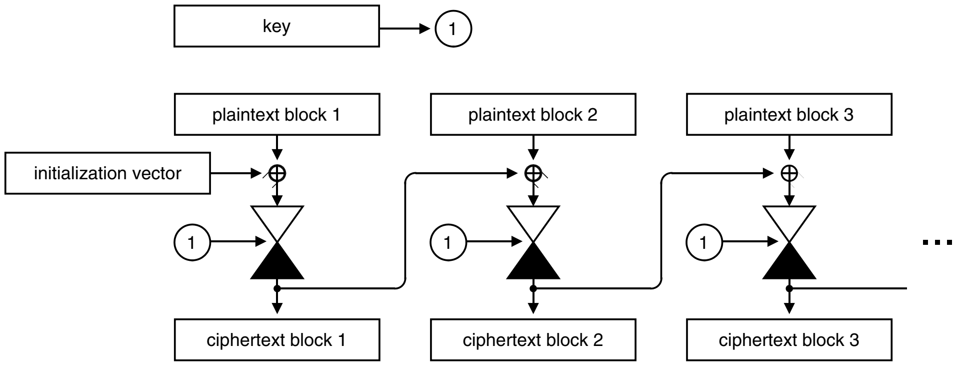


Figure 4.28 Data encryption with a block-oriented encryption algorithm operating in the CBC mode. Decryption is performed in a similar manner

Multiple encryption

In addition to the four operating modes of a block-oriented encryption algorithm, another variation is also used to improve security. However, it is actually only used with the DES algorithm, due to the small key space of this algorithm. In principle, it can be used with any block-oriented encryption algorithm that is not a group. If an encryption method has the group property, double encryption using two different keys will not increase the level of security, since the same result could be obtained by single encryption using a third key. The size of the key space is not increased by double encryption using an algorithm that has the group property, since an attacker would only have to discover the third key in order to obtain the result previously obtained by double encryption using two different keys:

$$\text{ciphertext} = \textbf{enc}\ (\text{key } 2;\ (\textbf{enc}\ (\text{key } 1;\ \text{plaintext}))$$

Since DES is not a group, the result of double DES encryption using two different keys fundamentally cannot be duplicated by single encryption using a third key.

However, in 1981 Ralph C. Merkle and Martin E. Hellman published a method of attack called the 'meet-in-the-middle' attack [Merkle 81], which can be used very successfully against every type of double encryption using a block-oriented encryption algorithm. It presupposes that the attacker knows several plaintext–ciphertext pairs. The operating principle of this attack is based on computing all possible encryptions of the plaintext using the first of the two keys, followed by decrypting the known result (the ciphertext) with every possible second key. The set of results from the first process is then compared with the set of results from the second process. If a match can be found, there is a certain probability that the two keys have been discovered. The level of confidence that the correct key has been found can be increased by making the same comparisons using additional known plaintext–ciphertext pairs. As can be seen, the amount of effort required for this attack is not significantly greater that the amount of effort needed for a normal attack that requires the entire key space to be searched. Consequently, cascaded double encryption is not used with the DES algorithm.

The process that is used instead is called triple-DES. In this mode, three sequential CBC-mode DES operations are performed using alternating encryption and decryption. Blocks that have been encrypted in this manner are decrypted by reversing the order of the operations (in other words: decryption, encryption and then decryption). If all three keys are the same, the result of the alternating encryption and decryption operations is the same as that obtained by a single encryption. This is the reason for not using a sequence of three encryption operations.

If the three DES operations using three keys are applied directly to each plaintext block in turn, the process is referred to as 'DES in the inner CBC mode'. If instead the plaintext is first completely encrypted using the first key and the result is then further encrypted in a similar manner, the process is referred to as 'DES in the outer CBC mode'. The outer CBC mode is more resistant to attack and is therefore generally recommended [Schneier 96].

The triple-DES algorithm has several other names, such as TDES, DES-3, 3-DES and 2-DES. Actually, the terms 'triple-DES' and '3-DES' only mean that three 56-bit keys are used. If the first and third keys are the same, this is called 2-DES, but if the three keys are all different, the designation 3-DES is often used. The key length must always be stated with triple DES in order to unambiguously specify the algorithm.

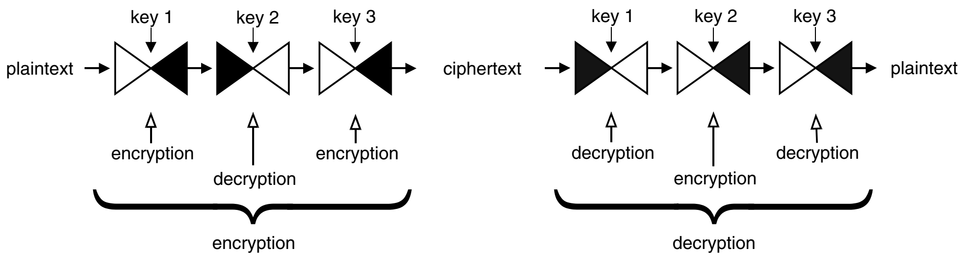


Figure 4.29 Operating principle of data encryption using triple-DES in the outer CBC mode. Key 1 is normally chosen to be the same as key 3, so the effective key length is 2 × 56 bits (112 bits). Less commonly, three independent keys are used, yielding a key length of 3 × 56 bits (168 bits)

The triple-DES algorithm is significantly more secure than sequential double encryption using two different keys, since the meet-in-the-middle attack is not effective against this method. Three 56-bit keys are needed instead of only one, but the first and third keys are usually the same. This yields a key length of 2 × 56 bits. This means that this procedure is data-compatible with the normal DES algorithm and that it imposes no additional costs except for the doubled key size. This in particular is one of the main reasons for the widespread use of triple-DES in smart cards. It is primarily used for deriving keys and protecting very sensitive data, such as when transferring keys, due to its improved level of security compared with single encryption.

4.7.2 Asymmetric cryptographic algorithms

In 1976, Whitfield Diffie and Martin E. Hellman described the idea of developing an encryption algorithm based on two different keys [Diffie 76]. One of these keys was to be public, the other secret or ‘private’. This would allow persons to encrypt a message using the public key, with only the owner of the private key being able to decrypt it. This in turn would eliminate the problems associated with exchanging and distributing secret symmetric keys. In addition, it would for the first time make certain other processes possible, such as generating digital signatures that can be verified by everyone.

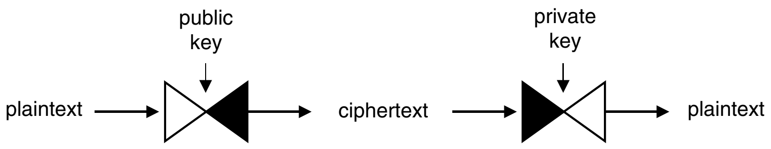


Figure 4.30 Encryption and decryption using a public-key algorithm

The RSA algorithm

Two years later, Ronald L. Rivest, Adi Shamir and Leonard Adleman presented an algorithm that met the above-mentioned criteria [Rivest 78]. This algorithm, which is called the RSA

Table 4.10 Summary of symmetric and asymmetric cryptographic algorithms used in smart cards, with their input and output parameters

Name	Type	Plaintext length	Ciphertext length	Key length
DES	Symmetric	8 bytes	8 bytes	56 bits
Triple DES (with two keys)	Symmetric	8 bytes	8 bytes	2×56 bits (112 bits)
Triple DES (with three keys)	Symmetric	8 bytes	8 bytes	3×56 bits (168 bits)
IDEA	Symmetric	8 bytes	8 bytes	128 bits
AES	Symmetric	8 bytes	16 bytes	128 bits (16 bytes) 192 bits (24 bytes) 256 bits (32 bytes)
RSA	Asymmetric	512 bits (64 bytes) 768 bits (96 bytes) 1024 bits (128 bytes) 2048 bits (256 bytes) 4096 bits (512 bytes)	(512 bytes) (64 bytes) 768 bits (96 bytes) 1024 bits (128 bytes) 2048 bits (256 bytes) 4096 bits (512 bytes)	512 bits (64 bytes) 768 bits (96 bytes) 1024 bits (128 bytes) 2048 bits (256 bytes) 4096 bits (512 bytes)
DSS (512 bits)	Asymmetric	20 bytes	20 bytes	$(64 + 20)$ bytes

algorithm after the initials of its inventors, is the best-known and most versatile asymmetric encryption algorithm presently in use. Its very simple operating principle is based on the arithmetic of large integers. The two keys are generated from two large prime numbers.[6]

The encryption and decryption processes can be expressed mathematically as follows:

$$\text{encryption:} \quad y = x^e \textbf{ mod } n$$
$$\text{decryption:} \quad x = y^d \textbf{ mod } n$$

$$\begin{aligned} \text{where } x &= \text{plaintext} \\ y &= \text{ciphertext} \\ e &= \text{public key} \end{aligned}$$

[6] Whitfield Diffie and Martin E. Hellman discovered the principle of public-key algorithms, while Ronald L. Rivest, Adi Shamir and Leonard Adleman produced the first implementation. However, public-key algorithms had already been discovered several years earlier in the field of military cryptography. In the period 1969–1973, James Ellis, Clifford Cocks and Malcolm Williamson, working at the British Government Communication Headquarters (GCHQ), developed both the principle of public-key algorithms and the asymmetric encryption algorithm now known as the RSA algorithm, several years before they were discovered by civilians. Unfortunately, these cryptologists were pledged to secrecy and were not allowed to publish anything about their work until after 1997 [Levy 99]

$$\begin{aligned} d &= \text{private key} \\ n &= \text{public modulus} = p \cdot q \\ p, q &= \text{secret prime numbers} \end{aligned}$$

Before being encoded, the plaintext block must be padded to the appropriate block size, which varies in the RSA algorithm according to the length of the key used. Encryption itself is performed by exponentiation of the plaintext followed by a modulo operation. The result of this process is the ciphertext. This can only be decoded if the private key is known. The decryption process is analogous to the encryption process.

The security of the algorithm is thus based on the difficulty of factoring large numbers. It is quite easy to compute the public modulus from the two prime numbers by multiplication, but it is very difficult to decompose the modulus into its two prime factors, since there is no effective algorithm for this operation.

The RAM capacity of smart cards is not sufficient for performing exponential operations on large numbers as required for encryption and decryption, since the numbers become very large before being subjected to the modulo operation. For this reason, modular exponentiation is used, which means that the intermediate result of the calculation never exceeds the value of the modulus. For example, if the value of x^2 **mod** n must be calculated, the expression $(x \cdot x)$ **mod** n is not evaluated directly, since the intermediate result $(x \cdot x)$ would be excessively large before being reduced by the modulo operation. Instead, the expression $((x$ **mod** $n) \cdot (x$ **mod** $n))$ **mod** n is evaluated, which yields the same mathematical result. The advantage of this is that it requires significantly fewer calculations and less memory, since the intermediate results are immediately reduced in size.

An additional way to increase the speed of the RSA algorithm is to use the Chinese remainder theorem for the calculations.[7] Of course, a prerequisite for using the Chinese remainder theorem is that both of the secret prime numbers p and q are known, which means that it can only be used for decryption (which means for signing).

The private key should be as long as possible, since this impedes attempts to break the code. Public and private keys may have different lengths, and in fact this is usually the case, since the time required to verify a digital signature can be considerably reduced by making the public key as short as possible. The fourth Fermat number is frequently used as a public key. This prime number has the value of $2^{16} + 1 = 65537$, and due to its small size it is very well suited to quickly verifying digital signatures. The numbers 7 and 17 are likewise used.

Table 4.11 Typical public keys for the RSA algorithm

Public key	Remarks
$2 = {}^\circ10^\circ$	The only even prime number; used for the Rabin procedure
$3 = (2^1 + 1) = {}^\circ11^\circ$	The smallest odd prime number
$17 = (2^4 + 1) = {}^\circ1\ 0001^\circ$	—
$65537 = (2^{16} + 1) = {}^\circ1\ 0000\ 0000\ 0000\ 0001^\circ$	The fourth Fermat number F_4

[7] For additional information regarding both procedures, see [Simmons 92, Schneier 96]

If an attacker succeeds in factoring the public modulus into its two prime components, he could then reproduce the entire encryption process. With a small value, such as 33, it is easy to factor the modulus, but there is presently no fast algorithm that can be used to factor large numbers. If the values of both prime factors can be found, the system is thereby broken, since the private key is then known.[8]

Consequently, a requirement for RSA keys is that they are sufficiently long. A length of 512 bits (64 bytes) is presently considered to be the lower limit. In any case, keys with a length of 768 bits (96 bytes) and 1024 bits (128 bytes) are presently used. In the coming years, the first 2048-bit (256-byte) key will come into use. The amount of computational effort needed for encryption and decryption increases with the key length. This increase is not linear, but instead approximately exponential.

Smart card microcontrollers, with their 8-bit CPUs, are normally not capable of performing an RSA computation in less than a few minutes. However, there are now microcontrollers with arithmetic coprocessors that have been specially developed for fast exponentiation. With such coprocessors, it is possible to perform RSA computations in an acceptable length of time with reasonable software overhead. The size of the code for a hardware-supported 512-bit RSA algorithm is around 300 bytes. Around 1 kB of assembler code in the smart card is needed for 768-bit and 1024-bit keys. As can be seen from Table 4.12, even with a 512-bit key the number of possible prime numbers is so large that collisions between two different key pairs will never occur.[9]

Table 4.12 Typical RSA key lengths with characteristic parameters. The ratio NS/PN indicates the relationship between the number of non-prime numbers and the number of prime numbers. The reciprocal of this is the probability that a random number in the number space is a prime. This is very important with regard to the length of time that is required to generate an RSA key

Key length	Maximum number of digits	Size of the number space (NS)	Number of prime numbers in the number space (PN)	NS/PN
8 bits (1 byte)	3	256	54	≈ 4.7
40 bits (5 bytes)	13	$\approx 1.1 \times 10^{12}$	$\approx 3.9 \times 10^{10}$	≈ 28
512 bits (64 bytes)	155	$\approx 1.3 \times 10^{154}$	$\approx 3.8 \times 10^{151}$	≈ 342
768 bits (96 bytes)	232	$\approx 1.6 \times 10^{231}$	$\approx 2.9 \times 10^{228}$	≈ 552
1024 bits (128 bytes)	309	$\approx 1.8 \times 10^{308}$	$\approx 2.5 \times 10^{305}$	≈ 720
2048 bits (256 bytes)	617	$\approx 3.2 \times 10^{616}$	$\approx 2.3 \times 10^{613}$	≈ 1391
4096 bits (512 bytes)	1234	$\approx 1.0 \times 10^{1233}$	$\approx 2.5 \times 10^{1229}$	≈ 4000

However, one of the strengths of the RSA algorithm is that it is not limited to a particular key length, in contrast to algorithms such as DES. If increased security is needed, longer keys

[8] As early as the summer of 1994, a 426-bit key was broken in eight months using approximately 1600 computers connected via the Internet. The total amount of computation amounted to 5000 MIPS-years. A 100-MHz Pentium processor, for comparison, has a capacity of 50 MIPS. In 19991, a 512-bit key was successfully factored in seven months using 292 networked computers

[9] The largest number that can be represented with 512 bits is $2^{512} - 1$, or in full: 13,407,807,929,942,597, 099,574,024,998,205,846,127,479,365,820,592,393,377,723,561,443,721,764,030,073,546,976,801,874,298,166, 903,427,690,031,858,186,486,050,853,753,882,811,946,569,946,433, 649,006,084,095

can be used without modifying the algorithm. The RSA algorithm is thus scaleable. However, computation time and amount of memory space needed must be kept in mind, since even 768-bit keys are presently still considered to be secure. With current factoring algorithms, a good rule of thumb is that increasing the key length by 15 bits doubles the effort of computing the factors.[10] Andrew Odlyzko [Odlyzko 95] provides an excellent summary of the internationally available and required processing capacity for factoring integers.

Although the RSA algorithm is very secure, it is rarely used to encrypt data, due to its long computation time. It is primarily used in the realm of digital signatures, where the benefits of an asymmetric procedure can be fully realized. The greatest drawback of the RSA algorithm with regard to smart cards is the amount of memory space required for the key. The complexity of the key generation process also causes problems in certain cases.

Widespread use of the RSA algorithm is restricted by patent claims that have been made in several countries and by major import and export restrictions imposed on equipment that employs this algorithm. Smart cards with RSA coprocessors fall under these restrictions, which considerably hinders their use internationally.

Table 4.13 Sample computation times for RSA encryption and decryption as a function of key length. The indicated values are in part subject to considerable variation, since they are strongly dependent on the microcomputer used, the bit structure of the key and the use of the Chinese remainder algorithm (which can only be used for signing)

Implementation	Mode	512 bits	768 bits	1024 bits	2048 bits
Smart card without NPU, 8-bit CPU, 3.5 MHz clock	Signing	20 min	—	—	—
Smart card without NPU, 8-bit CPU, 3.5 MHz clock (with Chinese remainder theorem)	Signing	6 min	—	—	—
Smart card with NPU, 3.5 MHz clock	Signing	308 ms	910 ms	2.0 s	—
Smart card with NPU, 3.5 MHz clock (with Chinese remainder theorem)	Signing	84 ms	259 ms	560 ms	—
Smart card with NPU, 4.9 MHz clock	Signing	220 ms	650 ms	1.4 s	—
Smart card with NPU, 3.5 MHz clock	Verifying	—	—	1.04 s	—
Smart card with NPU, 4.9 MHz clock (with Chinese remainder theorem)	Signing	60 ms	185 ms	400 ms	—
Smart card with NPU and PLL	Verifying	60 ms	185 ms	400 ms	—
PC (Pentium, 200 MHz)	Signing	12 ms	46 ms	60 ms	
PC (Pentium, 200 MHz)	Verifying	2 ms	4 ms	6 ms	
RSA integrated circuit	Signing	1.6 ms	—	—	

[10] As of January 1998, the largest known prime number had 909,256 digits and a value of $2^{3,402,377} - 1$

Generating RSA keys

Keys for the RSA algorithm are generated using a simple process. The following is a small worked-through example:

1. First, select two prime numbers p and q:	$p = 3; q = 11$
2. Next, calculate the public modulus:	$n = p \cdot q = 33$
3. Calculate the temporary variable z for use during key generation:	$z = (p - 1) \cdot (q - 1)$
4. Calculate a public key e which satisfies the conditions $e < z$ and **gcd** $(z, e) = 1$ (that is, the greatest common denominator of z and e is 1). Since there are several numbers that meet these conditions, select one of them:	$e = 7$
5. Calculate a private key d that satisfies the condition $(d \cdot e)$ **mod** $z = 1$:	$d = 3$

This completes the computation of the keys. The public and private keys can now be tested for encryption and decryption using the RSA algorithm, as illustrated in the following numeric example:

1. Use the number '4' as the plaintext x ($x < n$):	$x = 4$
2. Encrypt the text:	$y = 4^7 \textbf{ mod } 33 = 16$
2. The result of the calculation is the ciphertext y:	$y = 16$
3 Decrypt the ciphertext:	$x = 16^3 \textbf{ mod } 33 = 4$

The result of decrypting the ciphertext is again the original plaintext, as expected.

In actual practice, key generation is more laborious, since it is very difficult to test large numbers to determine if they are prime. The well-known sieve of Eratosthenes cannot be used here, since it requires prior knowledge of all prime numbers smaller than the number being tested. This is practically impossible for numbers as large as 512 bits. Consequently, probabilistic tests are used to determine the likelihood that the selected number is a prime number. The Miller–Rabin test and the Solovay–Strassen test[11] are typical examples of such tests. To avoid having to use these time-consuming tests more than necessary, randomly generated candidate numbers are first tested to see if they have any small prime factors. If the randomly generated number can be exactly divided by a small prime number, such as 2, 3, 5 or 7, it obviously cannot be a prime number. Once it has been determined that the number to be tested does not have any small prime factors, a prime number test such as the Miller–Rabin test can be used. The principle of this test is illustrated in Figure 4.31 and described in detail in the appendix of the IEEE 1363 standard.[12]

[11] The procedure and the algorithm are described by Alfred Menezes [Menezes 97]

[12] Many tips and criteria that must be taken into account for the generation of prime numbers can be found in an article by Robert Silverman [Silverman 97]

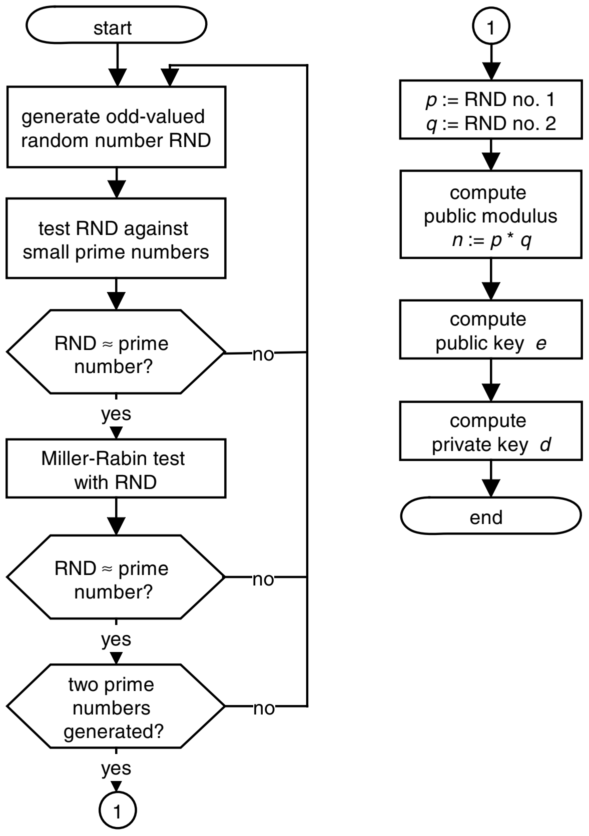

start
generate odd-valued random number RND
test RND against small prime numbers
RND ≈ prime number?
no
yes
Miller-Rabin test with RND
RND ≈ prime number?
no
yes
two prime numbers generated?
no
yes
1
1
p := RND no. 1
q := RND no. 2
compute public modulus n := p * q
compute public key e
compute private key d
end

Figure 4.31 Basic procedure for generating RSA keys for use in smart cards

The algorithms for generating RSA keys have a special feature, which is that the time required to generate a key pair (a public key together with a private key) is only statistically predictable. This means that it is only possible to say that there is a certain probability that key generation will take a given amount of time. A definitive statement such as '. . . will take *x* seconds' is not possible, due to the need to run the prime number test on the random number. The time required to perform this test is not deterministically predictable.

The DSS algorithm

In mid-1991, the NIST (US National Institute of Standards and Technology) published the design of a cryptographic algorithm for adding signatures to messages. This algorithm, which has since been standardized in the US (FIPS 186), has been named the Digital Signature Algorithm (DSA), and the standard that describes it is called the Digital Signature Standard (DSS). The DSA and RSA algorithms are the two most widely used procedures for generating digital signatures. The DSA algorithm is a modification of the El Gamal procedure. The background for the standardization of this algorithm is that a procedure was wanted that could be used to generate signatures but not to encrypt data. For this reason, the DSA algorithm is

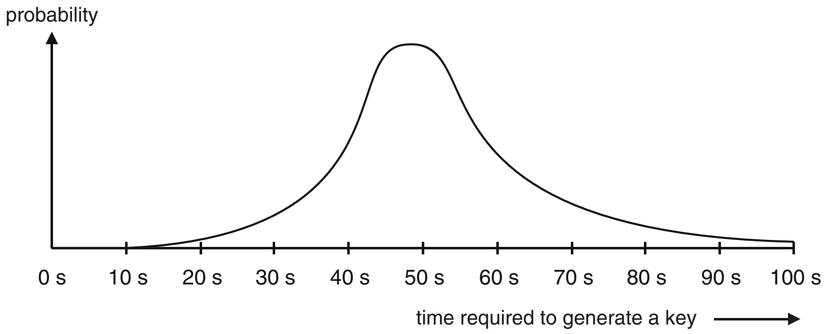


Figure 4.32 Typical time behavior of a probabilistic algorithm for generating key pairs for the RSA algorithm. The vertical axis shows the probability that a given amount of time will be required to generate a 1024-bit key in a smart card. Consequently, key generation takes 50 s on average in this example. The total area under the curve has a probability of 1

Table 4.14 Examples of the time required to generate a pair of public and private keys for the asymmetric RSA cryptographic algorithm. Exact times cannot be given, since the duration of the key generation process depends on whether the generated random numbers are prime, among other things

Generating a public/private key pair for the RSA algorithm	Typical time	Possible time
Smart card, 512-bit key, 3.5 MHz	6 s	$\approx$ 1 s to $\approx$ 20 s
Smart card, 1024-bit key, 3.5 MHz	14 s	$\approx$ 6 s to $\approx$ 40 s
Smart card, 2048-bit key, 3.5 MHz	80 s	$\approx$ 6 s to $\approx$ 40 s
PC (Pentium, 200 MHz), 512-bit key	0.5 s	—
PC (Pentium, 200 MHz), 1024-bit key	2 s	—
PC (Pentium, 200 MHz), 2048-bit key	36 s	—

more complicated than the RSA algorithm. However, it has been shown that it is possible to encrypt data using this algorithm [Simmons 98].

In contrast to the RSA algorithm, the security of the DSS algorithm does not depend on the problem of factoring large numbers, but rather on the discrete logarithm problem. The expression $y = a^x \textbf{ mod } p$ can be computed quickly, even with large numbers. However, the reverse process, which is calculating the value of x for given values of y, a and p, requires a very large amount of computational effort.

With all signature algorithms, the message to be signed must first be reduced to a predefined length using a hash algorithm. The NIST therefore published a suitable algorithm for use with the DSS algorithm. This is named SHA-1 (Secure Hash Algorithm).[13] This variant of the MD5 hash algorithm generates a 160-bit hash value from a message of any arbitrary length. Computations for the DSS algorithm, like those for the RSA algorithm, are performed using only integers.

[13] See Section 4.9, 'Hash Functions'

To compute a signature with the DSA algorithm, the following global values must first be determined:

p (public):	prime number with a length of 512 to 1024 bits, evenly divisible by 64
q (public):	160-bit prime factor of $(p - 1)$
g (public):	$g = h^{(p-1)/q}$ where h is an integer satisfying the conditions $h < p - 1$ and $g > 1$

The private key x must satisfy the following condition:

$$x < q$$

The public key y is computed as follows:

$$y = g^x \textbf{ mod } p$$

Once all of the necessary keys and numbers have been determined, the message m can be signed as follows:

Generate a random number k, where $k < q$:	k
Compute the hash value of m:	$H(m)$
Calculate r:	$r = (g^k \textbf{ mod } p) \textbf{ mod } q$
Calculate s:	$s = k^{-1}\ (H(m) + x \cdot r) \textbf{ mod } q$

The two values r and s are the digital signature of the message. With the DSS algorithm, the signature consists of two numbers, instead of only one number as with the RSA algorithm.

The signature is verified as follows:

Calculate w:	$w = s^{-1} \textbf{ mod } q$
Calculate $u1$:	$u1 = (H(m) \cdot w) \textbf{ mod } q$
Calculate $u2$:	$u2 = (r \cdot w) \textbf{ mod } q$
Calculate v:	$v = ((g^{u1} \cdot y^{u2}) \textbf{ mod } p) \textbf{ mod } q$

If the condition $v = s$ is satisfied, the message m has not been altered and the digital signature is authentic.

In practice, the RSA algorithm has achieved more widespread use than the DSS algorithm, which up to now has seen only very limited use. The original idea of standardizing a signature algorithm that cannot be used for encryption, which led to the DSS algorithm, has largely come to nothing. The complexity of this algorithm also discourages its widespread use. Nonetheless, for many institutions the fact that the standard exists and the political pressure to generate signatures using the DSS and SHS represent strong arguments in its favor.

Table 4.15 Examples of computation times for the DSA algorithm as a function of the clock rate, divided into the times required for verifying (encrypting) and generating (decrypting) a signature. These values are subject to considerable variation, since they depend strongly on the bit structure of the key. The computation time can be reduced by precomputation

Implementation	Verifying a 512-bit signature	Generating a 512-bit signature
Smart card with 3.5-MHz clock	130 ms	70 ms
Smart card with 4.9-MHz clock	90 ms	50 ms
PC (80386, 33 MHz)	16 s	35 ms

Using elliptic curves as asymmetric cryptographic algorithms

In addition to the two well-known asymmetric cryptographic algorithms, RSA and DSA, there is a third type of cryptography that is used for digital signatures and key exchanges in the realm of smart cards. It is based on elliptic curves (EC).

In 1985, Victor Miller and Neal Koblitz independently proposed the use of elliptic curves for constructing asymmetric cryptographic algorithms. The properties of elliptic curves are well suited to such applications, and in the course of the following years, practical cryptographic systems based on these proposals were developed. In general, they are usually referred to as elliptic curve cryptosystems (ECC).

Elliptic curves are sets of smooth curves that satisfy the equation $y^2 = x^3 + ax + b$ within a finite three-dimensional space. No point is allowed to be a singularity. This means, for instance, that $4a^2 + 27b^2 \neq 0$. In the realm of cryptography, the finite spaces GF(p), GF(2^n) and GF(p^n) are used, where p is a prime number and n is a positive integer greater than 1.

The mathematics of cryptographic systems based on elliptic curves are relatively difficult. For this reason, you are referred to the book by Alfred Menezes on the subject [Menezes 93]. The very comprehensive IEEE 1363 public-key cryptography standard and the ISO/IEC 15946 series of standards dealing with elliptic curves also provide good synopses of elliptic curves and other asymmetric cryptographic techniques.

The major advantages of asymmetric cryptographic systems based on elliptic curves are that they require much less computational capacity than systems such as RSA (for instance), and that the same level of cryptographic strength can be attained with significantly shorter keys. For example, roughly the same amount of computation is required to break an ECC algorithm with a 160-bit key as an RSA algorithm with a 1024-bit key. Similarly, an ECC algorithm with a 256-bit key corresponds to an RSA algorithm with a 2048-bit key, while an ECC algorithm with a 320-bit key roughly corresponds to an RSA algorithm with a 5120-bit key. This cryptographic strength and the relatively small size of the keys are precisely the reasons why ECC systems are found in the realm of smart cards.

The arithmetic processing components of modern-day smart card microcontrollers generally support ECC, which means that a relatively high computational speed is available. As with the RSA algorithm, the key length is an important characteristic of these asymmetric cryptographic algorithms.

Interestingly enough, cryptographic systems based on elliptic curves require so little processing capacity that they can even be implemented in microcontrollers lacking coprocessors.

Some typical times for generating and verifying signatures are shown in Table 4.16. An 8-bit microcontroller clocked at 3.5 MHz without a coprocessor requires approximately one second to generate a 160-bit ECC key pair using a look-up table approximately 10 kB in size. This time can be reduced to 200 ns using a coprocessor.

Table 4.16 Sample processing times for cryptographic algorithms based on elliptic curves in GF(p). The remarkably good times for smart cards without coprocessors are achieved using table look-up to accelerate certain time-intensive computations (table size approximately 10 kB)

Implementation	Generating a 135-bit signature	Verifying a 135-bit signature
Smart card, 3.5-MHz clock and 8-bit processor	1 s	4 s
Smart card, 3.5-MHz clock and numeric coprocessor	150 ms	450 ms
PC (Pentium III, 500 MHz)	10 ms	20 ms

One factor limiting the use of elliptic curves for asymmetric cryptographic algorithms is that they are regarded as a relatively new discovery in the cryptographic world, even though they have been known for a long time. It will no doubt take some time until the use of ECC systems becomes commonplace in the cautious world of cryptographers and smart card application designers, despite the fact that cryptographic systems based on elliptic curves presently offer the highest level of security per bit relative to all other asymmetric methods.

4.7.3 Padding

In smart cards, the DES algorithm is primarily used in the two block-oriented modes (ECB and CBC). However, since the data communicated to the card do not always fit exactly into a certain number of blocks, it is occasionally necessary to fill up a block. Filling up a data block so that its length is an exact multiple of a given block size is called *padding*.

The recipient of a padded data block has a problem after the data have been decrypted, since he does not know where the actual data stop and the padding bytes start. One solution to this would be to state the length of the message at the beginning of the message, but this would change the structure of the message, which is generally undesirable. It would also be especially onerous with data that do not always have to be encrypted, since in this case no padding would be needed and thus no length as well. In many cases, therefore, the structure of the message may not be changed.

This means that a different method must be used to identify the padding bytes. The algorithm defined in the ISO/IEC 9797 standard is described here in detail as an example, although there are a variety of other methods available. The most significant bit (msb) of the first padding byte following the useful data is set to 1. This byte thus has the hexadecimal value '80'. If additional padding bytes are needed, they have the value '00'. The recipient of the padded message thus searches from the beginning to the end of the message for a byte with the msb set to 1, or for the value '80'. If such a byte is found, the recipient knows that this byte and all subsequent bytes are padding bytes and not part of the message.

In this regard, it is important for the recipient to know whether messages are always padded or padded only if necessary. If padding only takes place when the length of the data to be

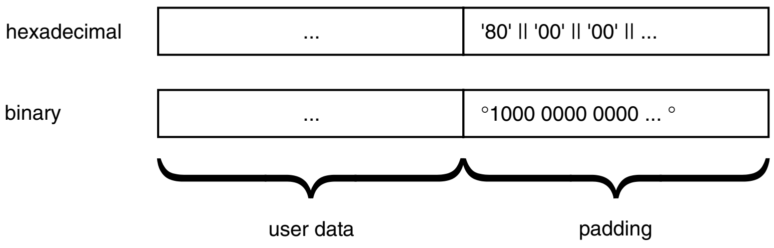


Figure 4.33 Data padding according to ISO/IEC 9797, Method 2

encrypted is not an integer multiple of the block length, the recipient must take this into account. Consequently, there is often an implicit understanding that padding always takes place, which of course has the disadvantage that occasionally an unnecessary block of padding data must be encrypted, transferred and decrypted.

In some applications, only the value '00' is used for padding. This is because this value is normally used for padding in MAC computations, and using only one padding algorithm reduces the size of the program code. Of course, in this case the application must know the exact structure of the data to allow it to distinguish between user data and padding.

Table 4.17 Typical padding methods using in the smart card realm. The data to be padded are designated as 'data'

Padding format	Description
ISO/IEC 9797	This padding format is used for generating MACs and for encryption Method 1: the data to be padded are padded using '00' Formal representation: data || n × '00' Method 2: '80' is appended to the data to be padded, which are then padded using '00' Formal representation: data || '80' || n × '00'
ISO/IEC 9796-2	This padding method is used for digital signatures. The data to be padded are appended to a bit sequence starting with °11° and ending with °1°, with a number of °0° characters in between as needed for padding, and the tag 'BC' is appended to the data. In addition, a random number can be integrated into the padding sequence in order to individualize the data to be padded Formal representation with bytewise padding: '60' || n × '00' || '01' || data || 'BC' Formal representation with bytewise padding and individualized data: '60' || n × '00' || '01' || RND || data || 'BC'
PKCS #1	The Type 1 version of this padding format is used for digital signatures, while the Type 2 version is used for generating MACs and encryption. The data to be padded are preceded by a tag and a fixed value or random number having the length necessary for the padding Formal representation, Type 1: '00' || '01' || n × 'FF' || '00' || data Formal representation, Type 2: '00' || '02' || n × RND || '00' || data

4.7.4 Message authentication code and cryptographic checksum

The authenticity of a message is far more important than its confidentiality. The term 'authenticity' means that the message has not been altered or manipulated, and is thus genuine. To ensure authenticity, a 'message authentication code' (MAC) is computed and appended to the message before it is sent to the recipient. The recipient can then compute the MAC for the message and compare it with the received MAC. If the two values match, the message has not been altered during its journey.

message
MAC

Figure 4.34 The usual arrangement of the message and the message authentication code (MAC)

A cryptographic algorithm with a secret key is used to generate a MAC. This key must be known to both parties to the communication. In principle, a MAC is a sort of error detection code (EDC), which can naturally only be verified if the associated secret key is known. For this reason, the term 'cryptographic checksum' (CCS) is also used (as well as some other terms), but technically a CCS is fully identical to a MAC. In general, the difference between the two terms is that 'MAC' is used for data transmission and 'CCS' is used for all other applications. The term 'signature' is often encountered as an equivalent to 'MAC'. However, this is not the same as a 'digital signature', since the latter is generated using an asymmetric cryptographic algorithm.

In principle, any cryptographic algorithm can be used to compute a MAC. In practice, however, the DES algorithm is used almost exclusively. This algorithm is used here to demonstrate the process (see Figure 4.35).

If the message is encrypted using the DES algorithm in the CBC mode, each block is linked to its previous block. This means that the final block depends on all previous blocks. This final block, or a portion of it, represents the MAC of the message. However, the actual message remains in plaintext, rather than being transmitted in encrypted form.

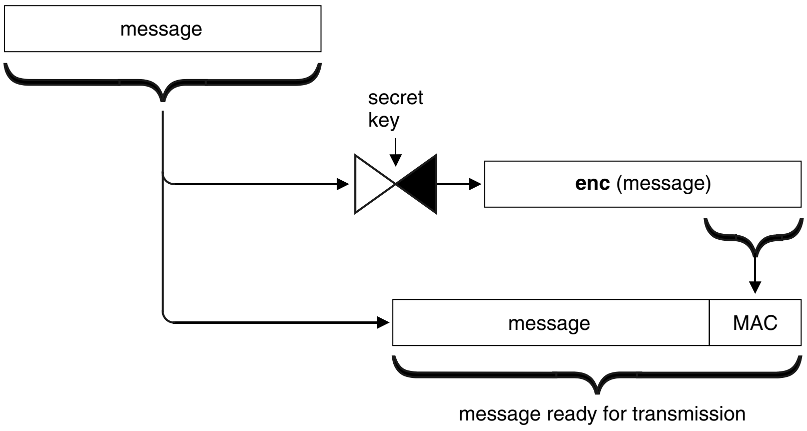


Figure 4.35 Example of a MAC computation process

There are a few important conditions relating to generating a MAC using the DES algorithm. If the length of the message is not an exact multiple of eight bytes, it must always be extended, which generally involves padding. However, in most cases only the value '00' is used for padding (in line with ANSI X.99 – Message Authentication). This is allowed in this case because there must be prior agreement regarding the length and location of the MAC within the message. The actual MAC consists of the left-most (most significant) four bytes of the final block produced by CBC-mode encryption. However, the padding bytes are not sent when the message is transmitted. This limits the data to be transmitted to the protected data and the appended MAC.

4.8 KEY MANAGEMENT

The sole objective of all administrative principles relating to keys for cryptographic algorithms is to minimize the consequences to the system and the smart card application if one or more secret keys become known to unauthorized persons. If it could be guaranteed that the keys would always remain secret, a single secret key for all smart cards would be sufficient. However, it is impossible to guarantee such secrecy.

Using the security-enhancing principles described here for keys used with cryptographic algorithms causes the number of keys to increase dramatically. If all of the principles and methods described in this section are implemented in a single smart card, the keys will usually take up more than half of the memory available for application data.

However, it is not always necessary to use every possible principle and method, depending on the application. For example, there is no need to support multiple generations of keys if the card is valid for only a limited length of time, since the additional administrative effort and memory space cannot be justified.

4.8.1 Derived keys

Since smart cards, in contrast to terminals, can be taken home by anyone and possibly subjected to thorough and painstaking analysis, they are naturally exposed to the most severe attacks. If no master key is present in the card, the consequences of a successful attempt to read out the card contents can be minimized. Consequently, the keys that are found in the card are only those that have been derived from a master key.

Derived keys are generated using a cryptographic algorithm. The input values are a card-specific feature and a master key. The triple-DES or AES algorithm is usually used. For the sake of simplicity, the card number is usually used as the specific feature. This number, which is generated when the card is manufactured, is unique in the entire system and can be used throughout the system to identify the card.

Derived keys are thus unique. One function that can be used to generate derived keys, as illustrated in Figure 4.36, is:

$$\textit{derived key} = \textbf{enc}\,(\textit{master key; card number})$$

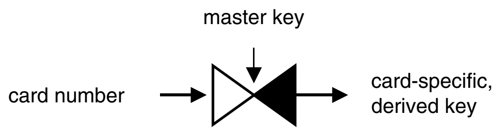


Figure 4.36 A possible method for generating a derived, card-specific symmetric key from the card number and a master key

4.8.2 Key diversification

In order to minimize the consequences of a key being compromised, a separate key is often used for each cryptographic algorithm. For example, different keys can be used for signatures, secure data transmission, authentication and data encrypting. For each type of key, there must be a separate master key from which the individual keys can be derived.

4.8.3 Key versions

It is normally not adequate to employ only one key generation for the full lifetime of a smart card. For example, suppose that a master key could be computed as the result of a successful attack. In this case, all application vendors would have to shut down their systems and card issuers would have to replace all their cards. The resulting loss would be enormous. Consequently, all modern systems include the possibility of switching to a new key generation.

Switching to a new generation of keys may be forced by the fact that a key has been compromised, but it can also take place routinely at a fixed or variable interval. The result of a switch is that all of the keys in the system are replaced by new ones, without any need for the cards to be recalled. Since the master keys are located in the terminals and the higher level parts of the system, a secure data exchange is all that is needed to provide new, confidential keys to the terminals.

4.8.4 Dynamic keys

In many applications, and in particular in the area of data transmission security, it is common practice to use dynamic keys. Such keys are also called ‘temporary keys’ or ‘session keys’. To generate a dynamic key, one of the two communicating parties first generates a random number, or some other value for use in a specific session, and passes it to the other party. The further course of the process depends on whether cryptographic algorithms used are only symmetric or also asymmetric.

Dynamic keys with symmetric cryptographic algorithms

For procedures that use only symmetric cryptographic algorithms, the random number generated by one of the two parties is sent as plaintext to the other party. The smart card and the terminal then encrypt this number using a derived key. The result, as shown in Figure 4.37, is a key that is valid only for one particular session.

$$\textit{dynamic key} = \textbf{enc}\ (\textit{derived key; random number})$$

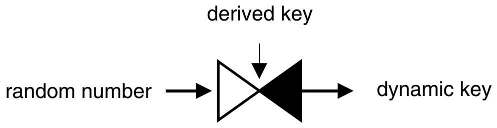


Figure 4.37 A possible way to generate a dynamic key using a random number and a derived key

The main advantage of dynamic keys is that they are different for each session, which makes attacks significantly more difficult. However, care must be taken when a dynamic key is used to generate a signature, since the dynamic key will also be needed to verify the signature. This key can only be generated using the same random number as was used when the signature was created. This means that whenever a dynamic key is used for a signature, the random number used to generate the key must be retained for use in verification, which means it must be stored.

The ANSI X 9.17 standard proposes a different method for generating derived and dynamic keys. Although it is somewhat more complicated than the previously described method, it is widely used in financial transaction systems. This method requires two inputs: a value T_i that is independent of the time or session and a key Key_{Gen} that is reserved for generating new keys. The resulting initial key Key_i can be used to compute as many additional keys as desired. This key generation method has the additional advantage that it cannot be computed in reverse; in other words, it is a one-way function:

$$Key_{i+1} = \textbf{enc}\ (Key_{\text{Gen}};\ \textbf{enc}\ (Key_{\text{Gen}};\ (T_i \text{ XOR } Key_i)))$$

Exchanging dynamic keys using an asymmetric cryptographic algorithm

Figures 4.38 and 4.39 show procedures for generating and subsequently exchanging a symmetric dynamic key for message encryption. An asymmetric cryptographic algorithm, such as RSA or DES, is used for key exchange. A similar process is used in PGP, for example, which uses the IDEA and RSA algorithms. The basic advantage of this hybrid process is that the actual encryption of large volumes of data can be performed using a symmetric cryptographic algorithm, which has significantly higher throughput than an asymmetric algorithm.

4.8.5 Key parameters

A mechanism that is as simple as possible is needed to allow the key stored in the card to be externally addressed. The smart card operating system must also always ensure that the key can only be used for its intended purpose. For instance, it must prevent an authentication key from being used for encrypting data. Besides the intended use, the key number must be known

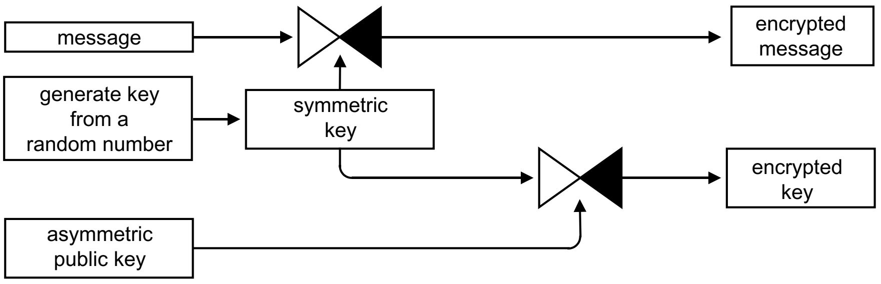


Figure 4.38 Sample procedure for key exchange using a combination of symmetric and asymmetric cryptographic algorithms. An encrypted dynamic symmetric key is first generated and then exchanged between two parties using an asymmetric cryptographic algorithm. The generation and exchange of the key pair for the asymmetric cryptographic algorithm, which takes place separately and in advance, is not shown

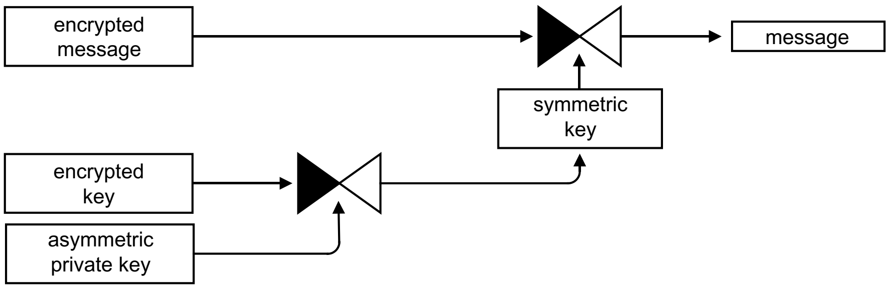


Figure 4.39 Sample procedure for key exchange using a combination of symmetric and asymmetric cryptographic algorithms. A previously encrypted dynamic symmetric key is recovered using an asymmetric cryptographic algorithm. The generation and exchange of the key pair for the asymmetric cryptographic algorithm, which takes place separately and in advance, is not shown

for it to be addressed. This number is the actual reference to the key. In addition, the version number is also needed to address a specific key.

Some smart card operating systems cause a retry counter associated with the key to be incremented each time a failure occurs in some activity that uses the key, such as an authentication. This can be used to quite reliably prevent the key value from being fished out by repeated trials, although this type of an attack does not represent a serious risk due to the long processing times in the card. If the retry count reaches its maximum value, the key is blocked and cannot be further used. The retry counter is reset to zero if the attempt to use the key is successful. Such a mechanism must always be used with great care, since an incorrect master key in a terminal could easily lead to massive card failures. A retry counter can normally only be reset using a special terminal, and the identity of the cardholder must be verified before this is done.

Some systems prohibit the reuse of old versions of keys. This is accomplished by providing the key with a ‘disable’ field that is activated as soon as a new key with the same key number is addressed.

Table 4.18 Typical key parameters stored in a smart card

Data object	Remarks
Key number	Key reference number; unique within the key file.
Version number	Version number of the key; which may affect key derivation.
Application purpose	Identifies the cryptographic algorithms and the procedures with which the key may be used.
Disable	Allows the key to be temporarily or permanently disabled.
Retry counter	This counter keeps track of non-successful attempts to use the key with a cryptographic procedure.
Maximum retry count	If the retry count reaches the maximum count, the key is blocked.
Key length	—
Key	The actual key.

4.8.6 Key management example

Here we would like to describe an example of key management for a system based on smart cards. The objective is to further illustrate the previously described principles by means of an easily understood general example. Compared with this example, large real systems frequently have arrangements that are much more complex, with several structural layers. Small systems often have no key hierarchy at all, since a secret global key is used for all cards. The system presented here occupies a middle position between systems with very simple structures and large systems, and thus represents a good example.

In the example shown in Figure 4.40, the keys for loading and paying can be used with an electronic purse. They use symmetric cryptographic procedures. These keys are evidently important within the system, since they are relatively well protected by the described key hierarchy. The individual derivation functions are not shown in detail here, but the DES or triple-DES algorithm could always be used for them. The lengths of the keys are also not dealt with in detail, but they certainly may vary. The keys at the top of the hierarchy are normally derived using more powerful cryptographic functions than those used at the lower level keys, for reasons of security.

The key at the top of the hierarchy is called the *general master key*. There is only one such key for an entire generation of keys. A generation could remain valid for a year, for example, and be replaced in the following year by a new generation, which means a new generation of the general master key. The general master key is the most sensitive key of the system with regard to security. If it becomes known, all of the keys of its generation can be computed, and the system is broken for one generation. The general master key may be generated from a random number. It is also conceivable to base the general master key on the values shown by dice thrown by several independent persons, each of whom consequently knows only part of the value of the key. The general master key should never be completely known by any single person, and its generation must under no circumstances be reproducible.

A master key for each function is separately derived from the general master key. These keys may be used for loading or paying with an electronic purse, for example. A one-way function, such as a modified triple-DES algorithm, is used in our example to derive the separate master keys for the various functions. This makes it impossible to compute the general master key from

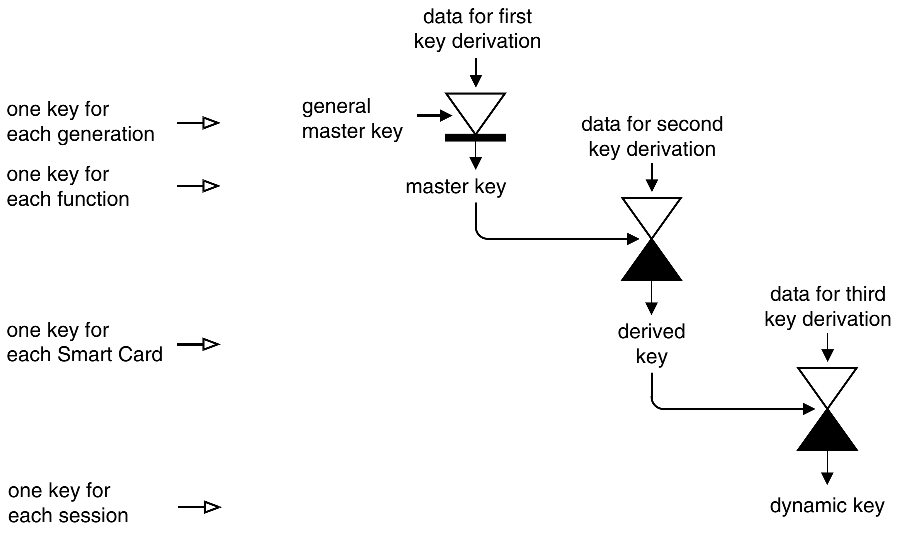

one key for each generation
one key for each function
one key for each Smart Card
one key for each session
data for first key derivation
general master key
master key
data for second key derivation
derived key
data for third key derivation
dynamic key

Figure 4.40 An example of a key hierarchy in a system based on smart cards and using symmetric cryptographic algorithms

a master key by applying the procedure in reverse. If a one-way function is not used to derive the master keys, the general master key could be computed if, despite all security measures, a master key becomes known and the derivation parameters are also known. A one-way function is used here because it is assumed that in this imaginary purse system, the master keys will be located in the security modules of local terminals. This means that with regard to system security, they are much more endangered than the general master key, which never leaves the background system.

The derived keys form the next level in the key hierarchy. These are the keys that are located in the smart cards. Each card contains a set of derived keys, which are classified according to their functions and generations. If such a card is used at a terminal, the terminal can compute the derived key for itself, based on the parameters used to derive the key in question. Naturally, the terminal first reads the derivation parameters from the card. Once the derived key is available, the following step is to compute the dynamic key, which is specific to a particular session. This key is valid only for the duration of a single session. The duration of a session ranges from a few hundred milliseconds to a few seconds in most smart card applications. A dynamic key is no longer used after the end of the session.

This example system may appear complicated at first glance, but it is relatively simple compared with real systems. The objective of the example is to show exactly how all the keys in a system can be generated. It also implicitly shows what measures must be taken if a key becomes known. If the general master key becomes known, a switch to a new generation must be made if the system is to continue to be used without concerns about security risks. By contrast, if a derived key becomes known, all that is necessary is to block the card in question; any other key management changes would surely be inappropriate. Of course, all of these measures presume that the reason why one or more keys have become known can be determined, so that it can be prevented in the future.

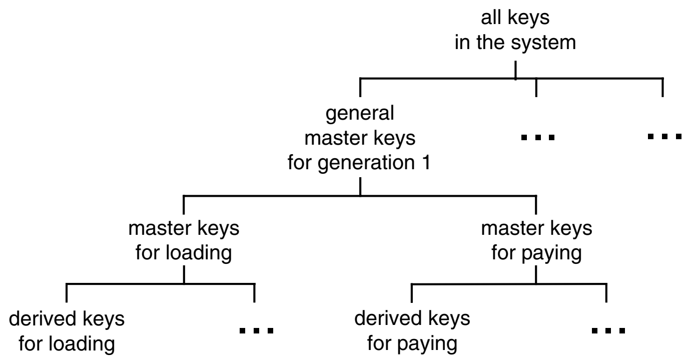


Figure 4.41 Examples of keys for an electronic purse system with two functions: loading and paying. Only the stored keys are shown here; keys that are dynamically generated for individual sessions have been omitted to simplify the diagram

Given this key hierarchy, it is evident that very many keys must be generated and stored in the smart cards. Of course, it is always possible to assign several functions to a single key in order to save memory space. It is also quite conceivable to use a different structure for the key hierarchy, which naturally strongly depends on the system for which the key management system is developed.

4.9 HASH FUNCTIONS

Even powerful computers require a great deal of time to compute a digital signature. In addition, large documents would need many signatures, since the document to be signed cannot be arbitrarily long. A trick is therefore used. The document is first compressed to a much shorter fixed length, and then the signature of the compressed data is computed. It does not matter whether the compression can be reversed, since the signature can always be reproduced from the original document. The functions used for this type of computation are called one-way hash functions.

Generally speaking, a one-way hash function is a function that derives a fixed-length value from a variable-length document in a manner such that this value represents the original content of the document in a compressed form and cannot be used to reconstruct the original document. In the smart card domain, these functions are used exclusively to compute the input values for digital signatures. If the length of the document is not a multiple of the block length used by the hash function, it must be padded appropriately.

For a hash function to be effective, it must exhibit certain properties. The result must have a fixed length, so that it can be readily used by signature algorithms. Since large quantities of data normally have to be processed, the hash function must have a high throughput. It must also be easy to compute the hash value. By contrast, it should be difficult, or better yet impossible, to derive the original document from a known hash value. Finally, the hash function must be collision-resistant. This means that for a given document, it should not be easy to find a second document that yields the same hash value. Nevertheless, there certainly will be other documents with the same hash value. This is only natural, since all possible messages, ranging in length from null to infinity, are represented by a set of hash values having the same fixed length. An unavoidable consequence of this is that collisions will occur. That is why the term 'collision-resistant' is used, rather than 'collision-free'.

What is the effect of a collision? There will be two different documents with the same hash value, and thus the same digital signature. This will have the fatal consequence of making the signature worthless, since it would be possible to alter the document without anyone being able to detect the fact. This is precisely what is involved in one of the two typical attacks on hash functions, which consists of systematically searching for a second document that has the same hash value as the original document. If the content of this document makes sense, the digital signature derived from the hash value is discredited. Since the two documents are interchangeable, the signature is worthless. After all, it makes an enormous difference whether a house purchase contract is for €10,000 or €750,000.

The second type of attack on a hash value is somewhat subtler. In this case, two documents with identical hash values but different contents are prepared in advance. This is not particularly difficult, considering all the special symbols and extensions available in the character set. The result is that a single digital signature is valid for both documents, and it is impossible to prove which document was originally signed.

Finding two documents with the same hash value is not as difficult as it might seem. It is possible to exploit the birthday paradox, which is well known in statistical theory. This paradox involves two questions. The first question is: how many people must be in a room for the probability to be greater than 50 % that one of them has the same birthday as the person asking the question. The answer can be easily found, since it is only necessary to compare the birthday of the questioner with the birthday of everyone else in the room. There must be at least 183 ($365 \div 2$) people in the room.

The second question reveals the paradox, or better, the surprising result of this comparison. This question is: how many people must be in a room for the probability to be greater than 50 % that two people in the room have the same birthday. The answer is only 23 people. The reason is that although only 23 people are present, this represents a total of 253 pairs for comparing birthdays. The probability that two people have the same birthday is based on these pairs.

Precisely this paradox is utilized in attacking a hash function. It is much easier to create two documents that have the same hash value than it is to modify a document until it yields a given hash value. The consequence is that the results of hash functions must be large enough to successfully foil both types of attack. Most hash functions thus produce values that are at least 128 bits long, which is presently generally considered to be adequate with regard to the two types of attack just described.

Many different hash functions have been published up to now, and some of them are also defined in standards. However, these functions are frequently modified as a consequence of the discovery of a successful form of attack. Table 4.19 provides a short summary of the hash functions currently in common use. Unfortunately, a description of their internal operation is beyond the scope of this book.

The ISO/IEC 10118-2 standard specifies a hash function based on an n-bit block-encryption algorithm (e.g. DES). With this algorithm, the length of the hash value may be n or $2n$ bits. The MD4 (message digest 4) hash function (presently rarely used) and its successor MD5 were published by Ronald L. Rivest in 1990–1991. They are based on a standalone algorithm, and both functions generate a 128-bit hash value. In 1992, the NIST published a hash function

Table 4.19 Summary of commonly used hash functions

Name	Input block size	Hash value length
ISO/IEC 10118-2	n bits (e.g. 64 or 128 bits)	n or $2n$ bits (e.g., 64 or 128 bits)
MD4	512 bits	128 bits
MD5	512 bits	128 bits
MDC-4	64 bits	128 bits
RIPEMD-128	512 bits	128 bits
RIPEMD-160	512 bits	160 bits
SHA-1	512 bits	160 bits

for the DSS algorithm that is known as SHA. After the discovery of certain weaknesses, it was modified, and the resulting function has been known since mid-1995 as SHA-1. It is also standardized under the name FIPS 180–1.

Since data transmission to smart cards is generally slow, the hash function is performed in the terminal or in a computer connected to the terminal. This drawback is balanced by the fact that this makes the hash function interchangeable. Besides, in most cases, memory limitations prevent hash functions from being stored in the cards. The program size is in almost every case around 4 kB of assembler code. The throughput of typical hash functions is very high relative to the demands placed on them. With an 80386 computer running at 33 MHz, it is usually at least 300 kB/s, and it lies in the range of 4 to 8 MB/s with a 200-MHz Pentium PC.

4.10 RANDOM NUMBERS

Random numbers are repeatedly needed in connection with cryptographic procedures. In the field of smart cards, they are typically used to ensure the uniqueness of a session during authentication, as padding for data encryption and as initial values for send sequence counters. The length of the random number needed for these functions usually lies in the range of 2 to 8 bytes. The maximum length naturally comes from the block size of the DES algorithm.

The security of all these procedures is based on random numbers that cannot be predicted or externally influenced. The ideal solution would be a hardware-based random number generator in the card's microcontroller. However, this would have to be completely independent of external influences, such as temperature, supply voltage, radiation and so on, since otherwise it could be manipulated. That would make it possible to compromise certain procedures whose security relies on the randomness of the random numbers. Current random number generators in smart card microcontrollers are generally based on linear feedback shift registers (LFSRs) driven by voltage-controlled oscillators.

Even with the current level of technological development, it is difficult to construct a random number generator immune to external influences (a 'true random-number generator', or TRNG) in silicon on a microcontroller die. Consequently, operating system designers frequently take recourse to software implementations. These yield pseudo-random number generators (PRNGs), most of which produce very good (that is, random) random numbers. Nevertheless, they do not generate truly random numbers, since the numbers are computed using strictly deterministic algorithms and thus can be predicted if the algorithm and its input values are known. This is why they are called 'pseudo-random numbers'.

It is also very important to ensure that the cards of a production batch generate different sequences of random numbers, so that the random numbers produced by one card cannot be inferred from those produced by another card from the same batch. This is achieved by entering a random number as the seed number (starting value) for the random number generator when the operating system is completed in each card.

4.10.1 Generating random numbers

There are many different ways to generate random numbers using software. However, since the memory capacity of smart cards is very limited and the time needed to perform the computation should be as short as possible, the number of options is severely restricted. In practice, essentially only methods that utilize functions already present in the operating system are used, since they require very little additional program code.

Naturally, the quality of the random numbers must not be adversely affected if a session is interrupted by a reset or by removing the card from the terminal. In addition, the generator must be constructed such that the sequence of random numbers is not the same for every session. This may sound trivial, but it requires at least a write access to the EEPROM to store a new seed number for the next session. The RAM is not suitable for this purpose, since it needs power to retain its contents. One possible means of attack would be to repeatedly generate random numbers until the EEPROM cells holding the seed number fail. Theoretically, this would cause the same sequence of random numbers to then occur in every session, which would make them predictable and thus give the attacker an advantage. This type of attack can easily be averted by constructing the relevant part of the EEPROM as a ring buffer and blocking all further actions once a write error occurs.

Another very important consideration for a software random number generator is to ensure that it never runs in an endless loop. This would result in a markedly shorter repeat cycle for the random numbers. It would then be easy to predict the numbers, and the system would be broken.

Almost every smart card operating system includes an encryption algorithm for authentication. It is an obvious idea to use this as the basis for a random number generator. In this regard, it is important to realize that a good encryption algorithm mixes the plaintext as thoroughly as possible, so that the plaintext cannot be derived from the ciphertext without knowledge of the key. A principle known as the avalanche criterion says that, on average, changing one input bit should change half of the output bits. This property can be usefully exploited for generating random numbers. The exact structure of the generator depends on the specific implementation.

Figure 4.42 illustrates a possible arrangement. This generator uses the DES algorithm with a block length of 8 bytes, with the output value being fed back to the input. Naturally, any other encryption algorithm could also be used. The generator works essentially as follows. The value of a ring buffer element is encrypted by DES using a key unique to the card. The ciphertext so produced is the 8-byte random number. This number, when XORed with the previous plaintext, provides the new entry for the EEPROM ring buffer. The generator then moves to the following entry in the cyclic ring buffer. This relationship can be expressed mathematically as $\text{RND}_n := f(\mathit{key}, \text{RND}_{n-1})$.

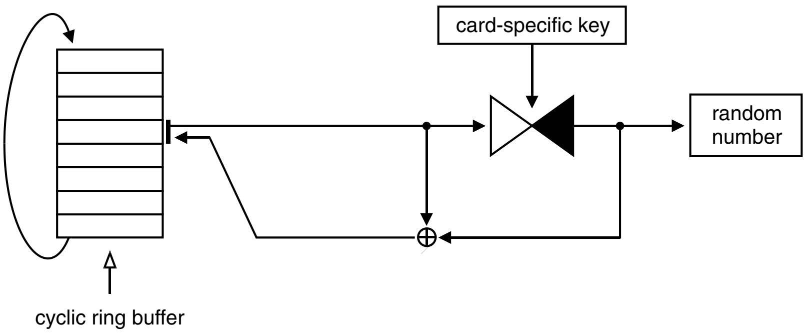


Figure 4.42 Sample architecture of a DES pseudo-random number generator for smart card operating systems. This generator is primarily designed to minimize the number of write accesses to the EEPROM

When the smart cards are completed, a card-specific DES key is stored in each card, and at the same time random seed numbers are entered into the ring buffer, which for example could be a 12 $\times$ 8 buffer. The seed numbers ensure that each card produces a unique sequence of random numbers. A 12-stage ring buffer increases the life span of the generator by a factor of 12. Assuming that the EEPROM is guaranteed to have 100,000 write cycles, this generator can produce at least 1,200,000 8-byte random numbers.

Erasing and writing eight bytes in the EEPROM takes about 14 ms (2 $\times$ 2 $\times$ 3.5 ms), and executing the DES algorithm takes about 17 ms at 3.5 MHz if it is implemented in software. The remaining processing time is negligible. The card thus needs around 31 ms to generate a random number. However, if the DES algorithm is computed in hardware (at a typical rate of 0.1 ms/block), a random number could be generated in only 14.4 ms using the described method.

Figure 4.43 shows another example of a pseudo-random number generator. This generator is initialized every time the card is reset, which is the only time a write access to the EEPROM occurs. Only RAM accesses are used for the subsequent generation of random numbers, which makes this generator relatively fast. However, the disadvantage of this is that the generator uses a few bytes of RAM for the duration of the session. The statistical quality of this pseudo-random number generator is not very good, but it is adequate for normal smart card authentication procedures. The primary consideration with such procedures is to avoid generating random numbers with short repeat cycles, since that would allow authentication to be compromised by replaying messages from previous sessions.

The FIPS 140-2 standard recommends that security modules check their built-in random number generators after every reset using statistical tests. Only after these tests have been successfully completed should the random number generator be released for further use. Current commonly used smart card operating systems rarely include such capability, since it is assumed that due to the deterministic nature of the pseudo-random number generator, the statistics of the generated random numbers will not change significantly.

The number of proposals, standards and designs for pseudo-random number generators is simply overwhelming. Some well-known examples are the generators in the X9.17 standard, FIPS 186, the proposals in the Internet RFC 1750 and the arrangements shown by Bruce Schneier [Schneier 96], Peter Gutmann [Gutmann 98a] and Benjamin Jun [Jun 99]. The guiding

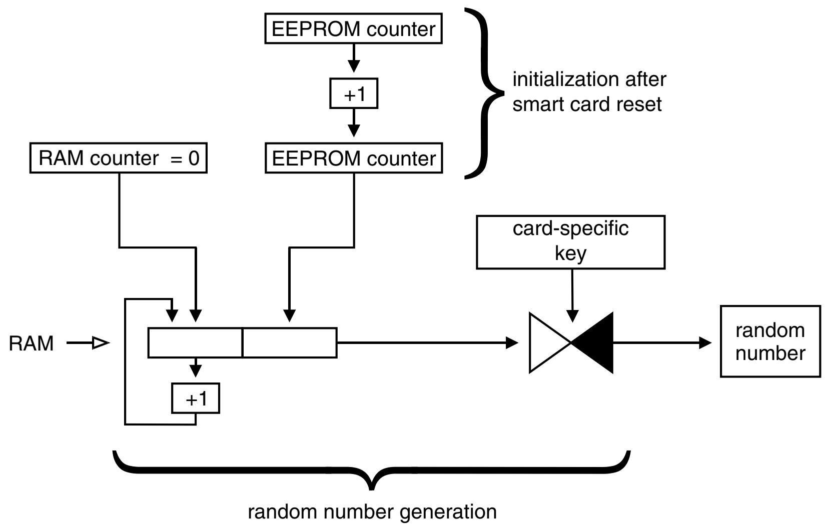


Figure 4.43 Sample architecture of a DES pseudo-random number generator for smart card operating systems. This generator is faster than the one shown in Figure 4.42, since only one EEPROM write cycle is needed per session. The quality of the random numbers it produces is adequate for normal smart card applications (authentication using the challenge–response procedure)

principle for a random number generator should always be to keep it as simple and easily understandable as possible. Only then is it possible to assess its characteristics and thus determine its quality.

4.10.2 Testing random numbers

After a random number generator has been implemented, it is generally necessary to test the quality of the numbers it produces. Fundamentally, there should be a nearly equal number of ones and zeros in the generated random numbers. However, it is not enough to simply print out a few numbers and compare them. Random numbers can be mathematically tested using standard statistical procedures. It is self-evident that a large number of 8-bit random numbers will be needed for such testing. Between 10,000 and 100,000 random numbers should be generated and analyzed in order to arrive at reasonably reliable results. The only way to test this many numbers is to use computerized testing programs.

When evaluating the quality of the random numbers, it is also necessary to investigate the distribution of the generated numbers. If this is very uneven, with certain values strongly favored, then exactly these regions can be used for purposes of prediction. This means that Bernoulli's theorem should be satisfied as closely as possible. This theorem states that the occurrence of a particular number, independent of what has come before it, depends only on the probability of occurrence of the number itself. For example, the probability that a 4 appears when a die is thrown is always 1/6, independent of whatever number appeared on the previous throw. This is also referred to as 'event independence'.

The period of the random numbers, which is the number of random numbers generated before the series repeats itself, is also very important. It must naturally be as long as possible, and in any case longer than the lifetime of the random number generator. In this way, the possibility of attacking the system by recording all random numbers generated for a complete period can be excluded in a quite simple and reliable manner.

There are many statistical tests for investigating the randomness of events, but in practice, we can limit ourselves to a few simple tests whose results are easily interpreted. There are also many publications on the subject of testing for randomness [Knuth 97, Menezes 97], as well as corresponding standards [FIPS 141-2, RFC 1750]. One test that is simple to set up and easy to interpret is to count the number of times that each byte value occurs in a large number of random numbers. If the results are displayed graphically as shown in Figure 4.44, they give a good indication of the distribution of the numbers.

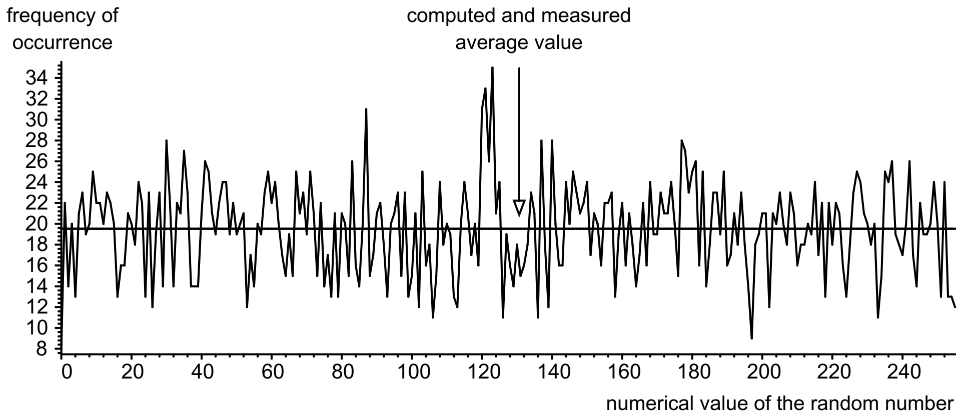


Figure 4.44 Statistical distribution of a series of 5000 single-byte random numbers. This is also referred to as the spectral distribution over one byte. These numbers were generated by a typical smart card pseudo-random number generator. Based on purely mathematical considerations, each of the possible values (in the range of 0–255) should occur 19.5 times

If such a diagram is used to investigate 8-byte random numbers, the values plotted on the horizontal axis must still be single-byte or at most two-byte numbers, since the number of samples needed for a statistical analysis would otherwise become extremely large. A good guideline is that every random number should occur approximately four to 10 times for each value in order to obtain reasonably reliable results. In this way, it is possible to quickly see whether the random numbers that have been generated fully exploit the possible bandwidth of the byte. If certain values are strongly favored, this offers an attacker a possible starting point.

Unfortunately, this test does not say anything about the order in which the random numbers occur, but only something about their distribution. For example, it would be possible for a 'random number' generator to output numbers cyclically from 0 to 255. This would yield an outstandingly uniform distribution, but the numbers would be completely predictable. Other tests must be used to assess this quality criterion for random numbers.

Another practical test that yields a simple and quick estimate of the quality of a series of random numbers is to compress the series using a file-compression program. According to

Shannon, the degree of compression that is possible is inversely related to the randomness of the set of generated numbers.

A significantly more robust test is the very well-known χ^2 test. Although it tests the same aspect as the previously described graphic test for even-statistical distribution, it is significantly more exact because it is performed using a mathematical procedure [Bronstein 96]. If the random numbers are assumed to be evenly distributed, the median value and standard deviation can be calculated. The deviation from a normal distribution can then be determined based on a χ^2 distribution. From this, it is possible to state a numerical value for the distribution of the random numbers.

However, this test cannot be used to draw any conclusions regarding the sequence in which the random numbers occur. Other statistical tests can be used to verify the randomness with which the numbers occur [Knuth 97], such as the Serial Test, which analyzes the periods of patterns that occur in the random numbers. Similarly, the Gap Test analyzes the intervals over which patterns do not occur. The Poker Test should also be used to evaluate the χ^2 distribution of patterns that do occur, and the Coupon Collector Test should be used to evaluate the χ^2 distribution of patterns that do not occur.

The Spectral Test, which investigates the relationship between each random number and the next following number, also has a certain amount of relevance [Knuth 97]. In the two-dimensional version of this test, random numbers and their immediate successors are plotted in an *X*–*Y* coordinate system, as shown in Figure 4.45. The three-dimensional version requires the successor to the successor number in addition, as well as a third axis (the Z axis). N-dimensional spectral tests can be performed in a similar manner, but for understandable reasons, they must dispense with graphical representation.

At a minimum, the above-mentioned tests must be performed and analyzed in order to achieve a reliable and definitive evaluation of a random number generator. Additional calculations and tests can be used to confirm the results so obtained. Only in this way is it possible to make a reasonably correct assessment of the quality of a set of random numbers.

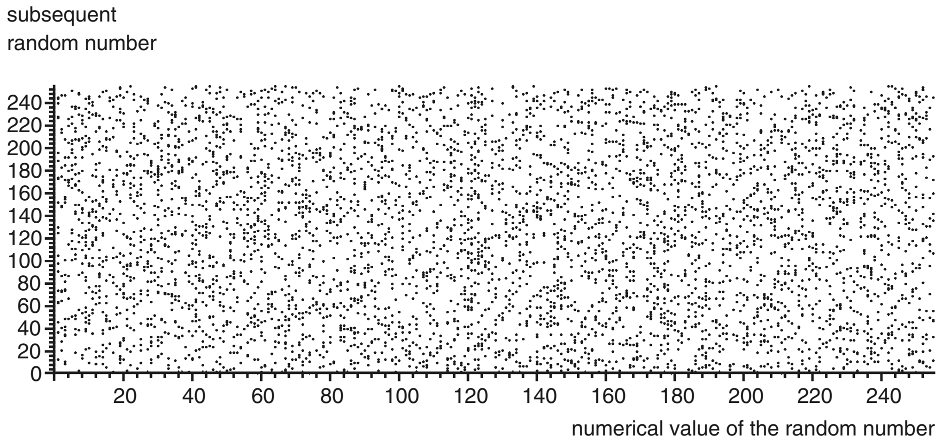


Figure 4.45 Graphic representation of the distribution of successor values of 5000 single-byte random numbers, corresponding to a spectral test. The nearly uniform distribution of the successor values can be seen at a glance from the regular pattern. The numbers were generated by a typical smart card pseudo-random number generator

Of course, considering the areas in which random numbers are used in smart card applications, an overly sophisticated random number generator is usually not justified. For instance, the effect on security of being able to predict the random numbers used for authentication would be very slight, since no attack is possible without knowledge of the private key used to encrypt the random number.

A more serious problem would, however, arise if it were possible to manipulate the random number generator, for example so that it would always generate the same sequence of random numbers. In this case, an attack based on replaying the numbers would be not only possible but also successful. This would also be true if the period of the random numbers were very short. In each individual case, the primary conditions that the random numbers must satisfy must be carefully considered, since this naturally affects the random number generator. Although a supreme effort here may lead to very high-quality random numbers, it also usually results in increased use of memory space, which is particularly limited in smart cards.

Table 4.20 Summary of standard statistical tests for random numbers

Test and reference	Remarks
Coupon collector test [Knuth 97] Poker test [Menezes 97]	χ^2 distribution of the non-occurrence of patterns in a series of random numbers.
Frequency test [Knuth 97, Menezes 97]	Counting the number of ones in a series of random numbers.
Gap test [Knuth 97]	Investigating the patterns that do not occur in a series of random numbers.
Long run test per FIPS 140-2	Investigating whether a series of ones and zeros with a length of 34 bits occurs in a series of random numbers that is 20,000 bits long.
Monobit test per FIPS 140-2	Counting the number of ones in a series of random numbers that is 20,000 bits long.
Poker test [Knuth 97]	χ^2 distribution of the occurrence of patterns in a series of random numbers.
Poker test per FIPS 140–1 Serial test [Menezes 97]	Counting 4-bit patterns in a series of random numbers that is 20,000 bits long.
Runs test per FIPS 140-1	Investigating maximum length of a series of all ones or all zeros in a series of random numbers that is 20,000 bits long.
Serial test [Knuth 97]	Investigating the patterns that occur in a series of random numbers.
Spectral test [Knuth 97]	Investigating the distribution of successor values of random numbers.

4.11 AUTHENTICATION

The purpose of authentication is to verify the identity and genuineness of a communications partner. Translated into the world of smart cards, this means that the card or the terminal determines whether its communications partner is a genuine terminal or a genuine smart card, respectively. For the sake of clarity, the term ‘identification’ is consistently used in this book to refer to verifying the authenticity of persons, although in principle it falls under the general concept of authentication.

Authentication requires the communicating parties to share a common secret that can be verified by means of an authentication procedure. Such a procedure is significantly more secure than a pure identification procedure, such as a PIN test. In the latter case, all that happens is that a secret (the PIN) is sent to the card, which confirms its genuineness if it is correct. The drawback of this procedure is that the secret is sent as plaintext to the card, which means that an attacker could easily come to know the secret (the PIN).

By contrast, with an authentication procedure it is not possible to discover the common secret by tapping the communications channel, since the secret does not have to be sent openly via the interface. A distinction is also made between static and dynamic authentication. In a static procedure, the same (static) data are always used for the authentication. A dynamic procedure, by contrast, is constructed such that it is protected against being attacked by re-entering data recorded during a previous session. This is because each authentication is based on different data when dynamic authentication is used.

There is also a fundamental difference between unilateral and mutual authentication procedures. A unilateral authentication, if it is successful, establishes the authenticity of one of the two communications partners. Mutual authentication, when successful, establishes the authenticity of both of the communications partners.

Authentication procedures based on cryptographic algorithms and used with smart cards can be further classified into symmetric and asymmetric procedures. Currently, the procedures used with smart cards are almost exclusively symmetric. Due to their slow execution speeds, asymmetric procedures, which means those based on the RSA algorithm or similar algorithms, do not yet have any practical significance with regard to smart cards systems. However, it can be foreseen that this will change in the future. In any case, the operating principle of asymmetric procedures is the same as that of symmetric procedures.

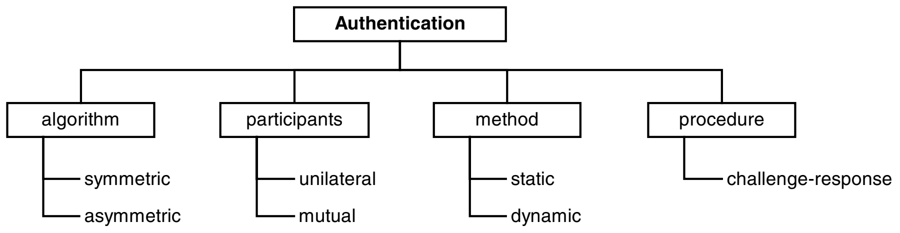


Figure 4.46 Classification of authentication procedures used with smart card systems

There are several standards relating to the authentication of equipment. The ISO/IEC 9798 standard is the most prominent of these. Part 2 of this standard describes symmetric procedures, while Part 3 describes asymmetric procedures. Fundamentally, the five parts of the ISO/IEC 9798 standard form an outstanding compilation of the commonly used authentication procedures, including symmetric, asymmetric, MAC-based and zero-knowledge-based procedures.

The principle of authentication in the field of smart cards is always based on a challenge–response procedure. In this procedure, one of the communications partners first asks the other one a randomly generated question (the challenge). The second partner computes an answer using an algorithm and sends the answer (the response) back to the first one. Naturally, the algorithm is preferably an encryption using a shared secret key that represents the common secret of the two communications partners.

4.11.1 Symmetric unilateral authentication

A unilateral authentication serves to assure one party of the trustworthiness of the other party to a communication. For it to be possible, both parties must have a shared secret, the knowledge of which is verified by the authentication procedure. This secret is the key for an encryption algorithm, and the entire security of the authentication procedure depends on this key. If the key should become known, an attacker could authenticate himself just as readily as a genuine communications partner.

The principle of unilateral authentication with a symmetric cryptographic algorithm is illustrated in Figure 4.47. For the sake of clarity, it is assumed that the terminal authenticates a smart card. This means that the terminal determines whether the smart card is trustworthy.

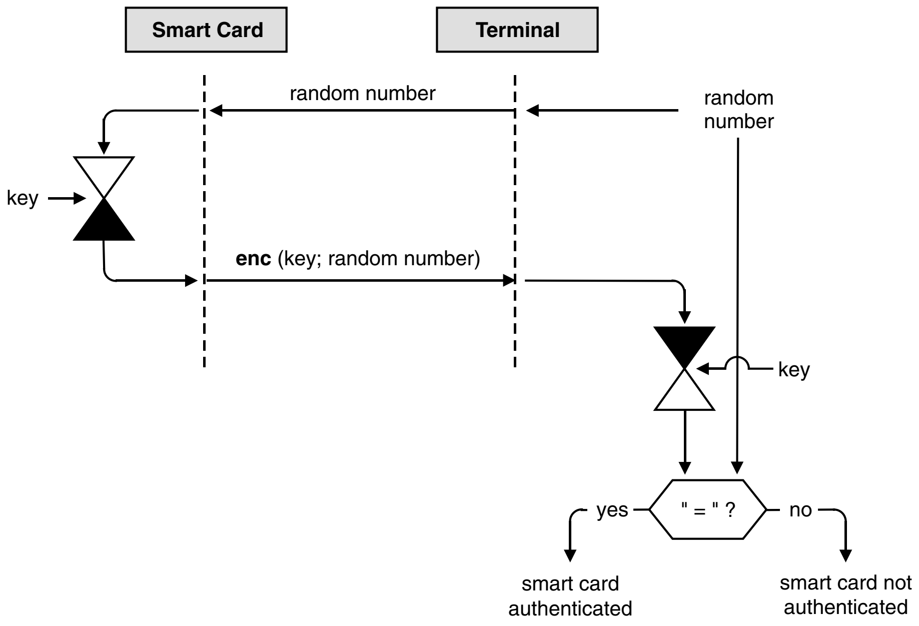


Figure 4.47 Working principle of unilateral authentication with a symmetric cryptographic algorithm. This example shows the authentication of a smart card by a terminal, which can be implemented using the INTERNAL AUTHENTICATE command of the ISO/IEC 7816-4 standard

The terminal generates a random number and sends it to the smart card. This is the challenge. The smart card encrypts the random number it receives, using a key known to both the card and the terminal. The security of the procedure depends on this key, since only the possessor of the secret key can generate the correct response to be sent to the terminal. The card then returns the result of the encryption to the terminal. This is the response to the challenge. The terminal uses the secret key to decrypt the encrypted random number it has received, and then compares the result with the random number it originally sent. If the two numbers match, the terminal knows that the smart card is authentic.

This procedure cannot be attacked by replaying a challenge or response that has been intercepted from an earlier session, since a different random number is generated for each session. The only type of attack with a moderately good chance of success would be to systematically search for the secret key. Since the challenge and response are simply a plaintext–ciphertext pair, the secret key could be discovered using a brute-force attack.

If all the cards for a given application have the same key and this key becomes known, the entire system will be discredited. In order to avoid exactly this possibility, in practice only card-specific keys are used as a matter of principle. This means that every card has an individual key, which may be derived from a non-secret feature of the card. This specific feature can be the serial number of the chip, which is written to the chip when it is manufactured, or some other number that is specific to each card.

In this case, the terminal requests the chip number from the smart card in order to compute the card-specific key. The chip number is specific to the card and unique within the system, so there is no other card in the system that matches this card. The value of the card-specific secret key is a function of the card number and the master key, which is known to the terminal. In practice, a portion of the card number is encrypted using the master key, and the result is used as the card-specific authentication key. A DES or triple-DES algorithm can be used for the encryption.

It must of course be borne in mind that if the master key (which is known only to the terminal) becomes compromised, the entire system will be compromised, since all card-specific authentication keys can be computed using the master key. The master key must therefore be securely stored in the terminal (in a security module, for example), and, if possible, it should be actively erasable in case of an attack.

Once the terminal has computed the necessary authentication key for the card, the usual challenge–response procedure occurs. The smart card receives a random number, encrypts it using its individual key and returns the result to the terminal. The terminal executes the reverse function of the computation performed by the card and compares the two results. If they match, the terminal and the smart card have a common secret, which is the secret card-specific key, and the smart card has been authenticated by the terminal.

In this case, the authentication process is somewhat time-intensive due the use of the DES algorithm (to the extent that it is implemented in software) and the data transmission from and to the card. This can cause problems in some applications. Given certain assumptions, we can roughly calculate the time required to perform a unilateral authentication. We assume that the smart card has a 3.5-MHz clock, uses the $T = 1$ transmission protocol, has a divisor of 372 and uses a DES algorithm that takes 17 ms per block. Without going into details, we assume that the internal routines in the smart card take 9 ms. This simplifies the calculation without significantly distorting the result, which is shown in Table 4.21. As can clearly be seen from this calculation, a single authentication takes around 65 ms. This will not usually cause any time-related problems in an application.

4.11.2 Symmetric mutual authentication

The principle of mutual authentication is based on dual unilateral authentication. In principle, two successive unilateral authentications could also be used, one for each of the communicating

Table 4.21 Calculation of the processing time within a smart card for a unilateral authentication, taking data transmission times into account

Command	Data transmission		Computation	Total
INTERNAL AUTHENTICATE (DES in software)	38.75 ms	+	26 ms	= 64.75 ms
INTERNAL AUTHENTICATE (DES in hardware)	38.75 ms	+	0.08 ms	= 38.83 ms

parties, in order to achieve mutual authentication. However, since the communications overhead must be kept as low as possible to minimize the time required by the process, a procedure that interleaves the two unilateral authentication processes has been defined. This also increases the security of the procedure, since it is much more difficult for an attacker to intervene in the communications process.

Before the terminal can compute the card-specific authentication key from the card number, it first needs the card number. After the terminal has received this number, it computes the specific authentication key for this card. It then requests a random number from the card, and at the same time it generates a random number itself. The terminal then swaps the two random numbers and concatenates them, after which it encrypts the resulting number using the authentication key. Finally, it sends the resulting ciphertext to the card. The objective of reversing the random numbers is to allow the challenge and response to be distinguished from each other.

The card can decrypt the received block and check whether the random number it previously sent to the terminal matches the number it received in return. If this is the case, the smart card knows that the terminal possesses the secret key. This authenticates the terminal with respect to the card. Next, the smart card swaps the two random numbers, encrypts the resulting number using the secret key and sends the resulting ciphertext block back to the terminal.

The terminal decrypts the received block and compares the random number it previously sent to the card with the one it has received in return. If they match, the smart card has been authenticated with respect to the terminal. This completes the mutual authentication process, and the terminal and the smart card both know that the other is trustworthy.

To minimize the communications time, the smart card can return the random number together with its card number. This is particularly attractive when mutual authentication takes place between a smart card and a background system. In this case, the card is directly addressed by the background system, with the terminal being 'transparent'. The data transmission rate in such situations is often very low, so the communications process must be streamlined as much as possible.

In order to illustrate the considerable amount of time required for a mutual authentication compared with a unilateral authentication, we can again make a sample calculation. The basic assumptions are the same as for the calculation of the time required for unilateral authentication (see Table 4.21). The results are shown in Tables 4.22 (for software implementations) and 4.23 (for hardware implementations). As can be seen, mutual authentication takes nearly three times as long as unilateral authentication.

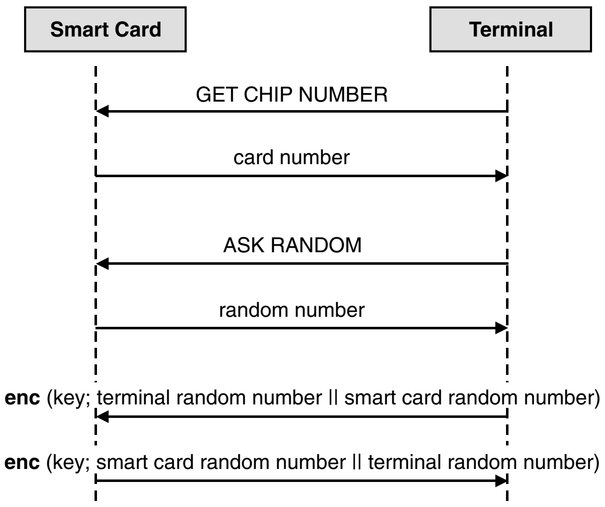

Smart Card
Terminal
GET CHIP NUMBER
card number
ASK RANDOM
random number
enc (key; terminal random number || smart card random number)
enc (key; smart card random number || terminal random number)

Figure 4.48 Mutual authentication using a card-specific key and a symmetric cryptographic algorithm. The illustrated procedure corresponds to a mutual authentication of a smart card and a terminal as implemented in the ISO/IEC 7816-8 AUTHENTICATE command

Table 4.22 Estimated time required for a smart card to perform mutual authentication if random number generation and DES computation are implemented in software, including data transmission time. It is assumed that derived keys are not used, so GET CHIP NUMBER is not necessary

Command	Data transmission		Computation	Total
ASK RANDOM	28.75 ms		26 ms	
MUTUAL AUTHENTICATE	68.75 ms		95 ms	
	97.50 ms	+	121 ms	= 218.50 ms

Table 4.23 Estimated time required for a smart card to perform mutual authentication if random number generation and DES computation are implemented in hardware, including data transmission time. It is assumed that derived keys are not used, so GET CHIP NUMBER is not necessary

Command	Data transmission		Computation	Total
ASK RANDOM	28.75 ms		0.08 ms	
MUTUAL AUTHENTICATE	68.75 ms		0.16 ms	
	97.50 ms	+	0.24 ms	= 97.74 ms

4.11.3 Static asymmetric authentication

Only a few smart card microcontrollers have arithmetic processing units that can be used to execute the RSA algorithm. This is mainly because such capability would take up additional space on the chip, which would increase its price.

However, the fact that a supplementary asymmetric authentication procedure would offer increased protection, since it requires an attacker to break two cryptographic algorithms instead of only one, often makes its use attractive. The problem presented by the absence of a suitable arithmetic processing unit on the card can be dealt with by the expedient of using static authentication of the card by the terminal. This only requires verification within the terminal, and an additional security module in the terminal does not significantly increase its overall cost. This solution is thus much more economical than the use of special smart card microcontrollers. In addition, this procedure is mush faster, since only one asymmetric encryption is required, as opposed to two in the case of dynamic asymmetric authentication.

The price of this compromise is reduced security of the authentication procedure. With a static procedure, there is naturally no protection against replaying previous data. This is why it is used only as a supplementary verification of the authenticity of the card, which has already been verified using a dynamic symmetric procedure.

The procedure works essentially as follows. When each smart card is personalized, card-specific information is entered into the card. This can for example be a card number, as well as the name and address of the cardholder. This information does not change during the lifetime of the card. As part of the personalization of the card, the digital signature of this information is computed using a secret key. This key is used globally in the system. When the card is used at a terminal, the terminal reads the signature and the signed data from a file in the card. The terminal has the public key, which is valid for all cards in the system, and it can use this key to encrypt the signature it has read and then compare the result with the data it has read from the card. If these two values match, the card has been authenticated by the terminal.

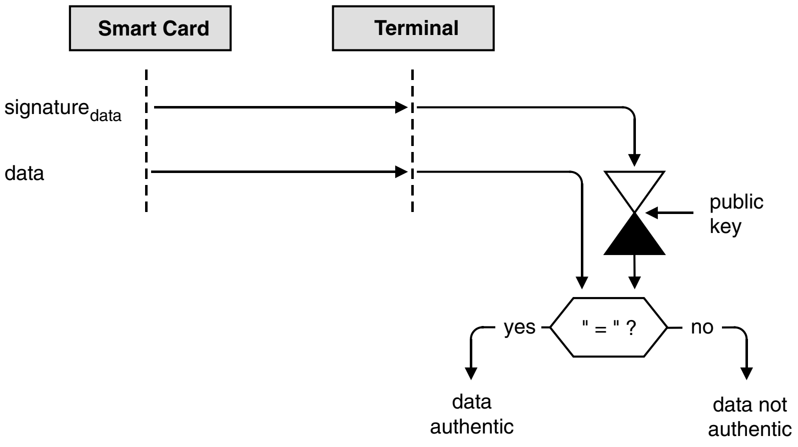


Figure 4.49 Operating principle of static, asymmetric unilateral authentication of a smart card by a terminal using a global key

The procedure illustrated in Figure 4.49, in addition to lacking protection against replaying data, has yet another drawback, which is that a global key is used to generate and verify the signature. Although the key in the terminal does not need to be protected, since it is public, global keys (which are the same for all cards) should fundamentally not be used in a large system. If such a key is broken, or if it becomes known for any other reason, authentication is rendered worthless in the entire system. This means that it is necessary to introduce card-specific key pairs for static authentication.

However, this presents a problem with the memory capacity of the terminals, since each terminal must hold all available public keys for signature verification. Even in a medium-sized system, such as one with one million smart cards, this would require each terminal to have 128 MB of memory for key storage, assuming 1024-bit RSA keys. This would increase the price of the terminals to a level that would not be acceptable to system operators.

When symmetric methods are used, it is quite easy to derive the card-specific keys from a master key.[14] This is not possible with asymmetric methods, due to the way the keys are generated. Consequently, a different approach is taken when card-specific keys are required. The public key for the verification of the signature is stored in the card, along with the signature. In the system of the previous example, the amount of memory needed to store the public keys is still 128 MB, but this is now distributed in 128-byte packets over one million cards. The terminal thus reads the public key from a file in the smart card and can then use it to verify the signature. This avoids the problem of having to store all the public keys of the system in every terminal.

However, an attacker could now generate a key pair and use these keys to sign the information in a counterfeit card. The terminal would read the public key and conclude that the card was genuine. A refinement of the procedure just described is therefore required. This consists of signing the combination of the public key and the card-specific key stored in each card, using a global secret key. This signature is then stored in each card.

The terminal now works as follows. It first reads the public and card-specific keys from the card and then tests the authenticity of the card-specific key using the global public key. If the card-specific key is authentic, the terminal then reads the actual data and verifies them using the public key stored in the smart card. This procedure is shown in Figure 4.50.

These two procedures are already used in some systems, and they will certainly be used increasingly in the coming years. However, as soon as the inclusion of an arithmetic processing unit for asymmetric cryptographic algorithms does not significantly increase the price of a smart card microcontroller, these two procedures will lose a lot of their significance. Their biggest disadvantage is the absence of protection against replaying data from earlier sessions. Although this can be partially compensated by the use of various tricks, such as reusing signed data in subsequent symmetric cryptographic algorithms, it is still not possible to match the level of protection provided by dynamic authentication procedures.

4.11.4 Dynamic asymmetric authentication

All of the previously described static asymmetric procedures have certain disadvantages. These can be eliminated by making the authentication dynamic, which provides protection against

[14] See also Section 4.8.1, 'Derived keys'

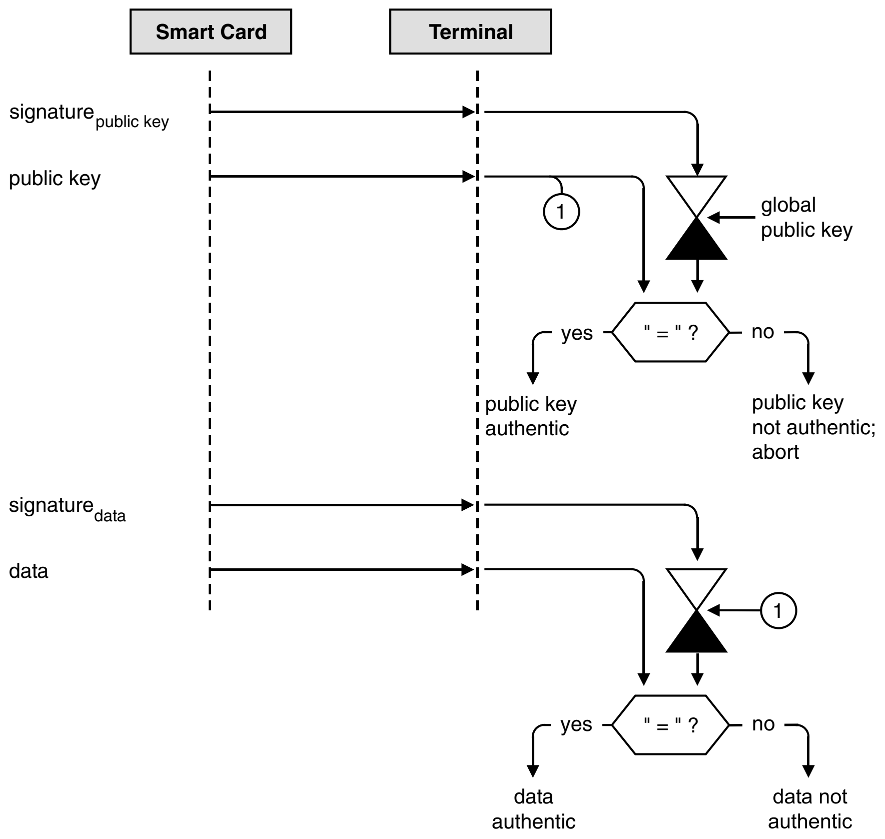


Figure 4.50 Operating principle of static, asymmetric unilateral authentication of a smart card by a terminal using a card-specific key

the re-entry of data intercepted from earlier sessions. The usual practice is to use a random number as the input value for a cryptographic algorithm. Of course, this means that the card must contain an arithmetic processing unit that can execute the asymmetric cryptographic algorithm.

Figure 4.51 illustrates a unilateral authentication using a global public key. If card-specific authentication keys are required, the procedure for the storage and authentication of card-specific public keys described in Section 4.11.3 is necessary.

As with symmetric authentication, the terminal generates a random number and sends this to the smart card. The card decrypts the random number using the secret key[15] and then sends the result back to the terminal. The terminal holds the global public key, and it uses this key to

[15] Using encryption to generate signatures comes from the convention that with an asymmetric cryptographic algorithm, the secret key is always used for decryption and the public key for encryption. The convention of using the public key for encryption goes back to the origins of the RSA procedure. One of the ideas at that time was to allow the RSA procedure to be used by agents operating in hostile territory to encrypt information to be kept secret. All that is needed to allow agents' reports to be sent back to headquarters in encrypted form is the RSA algorithm and

encrypt the random number that it has received. If the result of this computation is the same as the random number that was previously sent to the card, the card has been authenticated by the terminal.

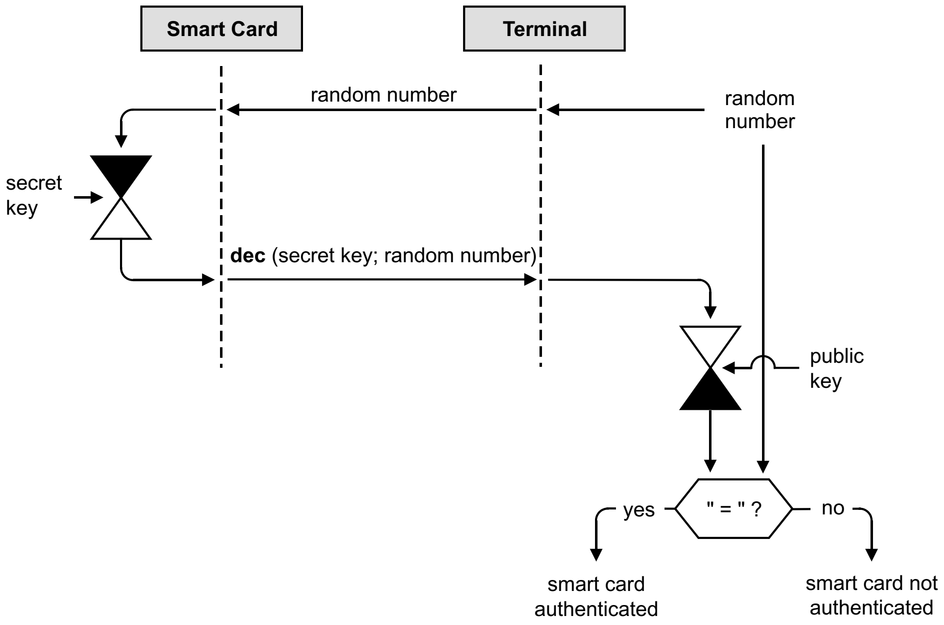


Figure 4.51 Operating principle of dynamic, asymmetric unilateral authentication of a smart card by a terminal

The basic features of a mutual authentication procedure for the smart card and the terminal are analogous to the unilateral procedure that has just been described. However, a mutual authentication requires a relatively long time, due to the large amount of data that must be exchanged and the time-consuming asymmetric encryption algorithm. Consequently, it is presently used very rarely.

4.12 DIGITAL SIGNATURES

Digital signatures, which are often referred to as 'electronic signatures', are used to establish the authenticity of electronically transmitted messages or electronic documents. Verification of the signature can be used to determine whether the message or document has been altered.

a public key. The messages can then be decrypted at headquarters in friendly territory using the private key. The main advantage of this arrangement is the ease of key distribution, since in principle agents can be given their keys without employing security measures. Even if a key becomes known, nobody would be able to decrypt messages that had been encrypted using the key, since this requires the private key. This initial strongly military application of the RSA algorithm forms the historical basis for the still valid convention that the public key is used for encryption and the private key for decryption

A signature has the property that it can be correctly produced by only one single individual, but it can be verified by anyone who receives the message – or at least, by any recipient who has previously seen the signature, or who has a copy available for comparison. This is also the essential characteristic of a digital signature. Only one person or one smart card can 'sign' a document, but everyone can verify whether the signature is genuine. Given this required characteristic, asymmetric cryptographic techniques represent the ideal starting point.

The message or document to be signed is usually at least several thousand bytes long. In order to keep the computation time for generating the cryptographic checksum within acceptable limits, the checksum is not computed over the entire data string. Instead, a hash value for the data string is first produced. Hash functions[16] are, simply stated, one-way functions for data compression. This compression is not reversible, which means that the original data cannot be reconstructed from the compressed data. Since the computation of a hash value is very fast, hash functions are an ideal aid for computing digital signatures.

The term 'digital signature' is usually only used in connection with asymmetric cryptographic algorithms, since the separation of the public and private keys makes such algorithms very suitable for use with digital signatures. Nonetheless, 'signatures' based on symmetric cryptographic methods are often used in practice. However, with such signatures it is only possible to verify the authenticity of a document if the secret key used to generate the signature is known. Such a 'signature' is thus actually not a signature in the true sense of the word, but it is often referred to as such in practice. The term 'digital' is in this case omitted, to indicate the type of procedure used.

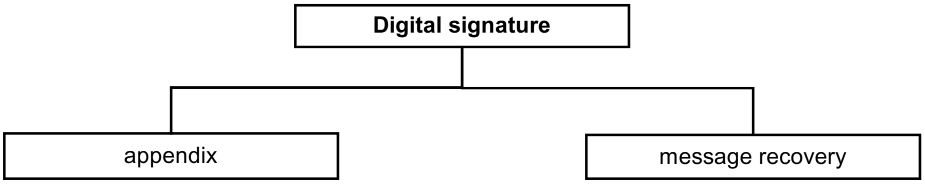


Figure 4.52 Classification of the two basic digital signature formats

From an informatics perspective, there are two different ways to attach a signature to a message. The first is a form of cryptographic checksum for a given data string, similar to a MAC (message authentication code), with the signature appended to the actual message (digital signature with appendix). This has the advantage that the message can be completely read without requiring prior verification of the signature. However, the drawback is that the size of the message is increased by the length of the signature, which can certainly be a consideration in the case of smart cards. This drawback can be avoided by using the second method for attaching a digital signature to a message, which is called 'digital signature with message recovery'. In this method, the hash value of the actual message is first appended to the message, following which an input block for the digital signature algorithm is formed starting at the end of the resulting data string. This means that the digitally signed message is increased in size only by the length of the hash value, but it cannot be completely read until the digital signature has been verified.

[16] See also Section 4.9, 'Hash functions'

The procedure for generating a digital signature with appendix can be quite easily portrayed. First, a hash algorithm is used to form a hash value from the content of the message, which may for example be a file produced by any arbitrary word-processing program. This hash value is decrypted using an asymmetric cryptographic algorithm, such as RSA in the example shown in Figure 4.53. The result of this computation is the actual signature, which is appended to the message.

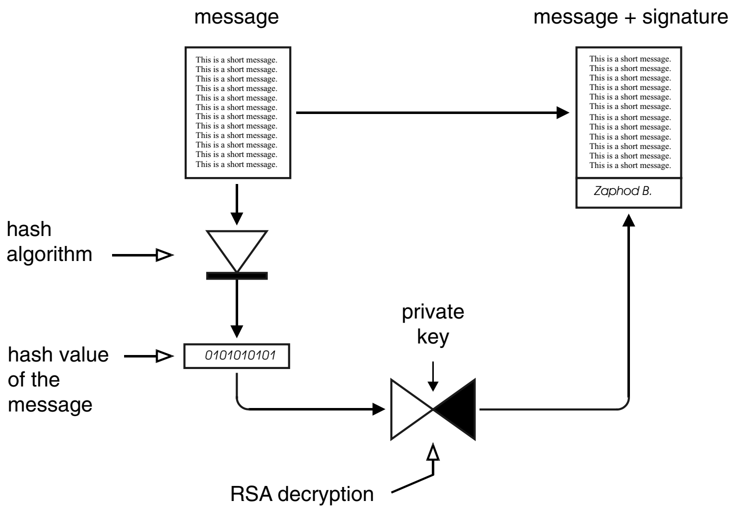


Figure 4.53 Signing a message with the RSA algorithm by appending the generated signature to the message (digital signature with appendix)

The signed message can now be sent via a non-secure path to the recipient. The recipient separates the signature from the message and then compresses the message using the same hash algorithm. The digital signature is encrypted using the public key of the RSA algorithm, and the result is compared with the result of the hash computation. If both values are the same, the message has not been altered while underway; otherwise, either the message or the signature has been altered during transmission. In the latter case, authenticity is no longer assured, and it cannot be assumed that the content of the message is unaltered.

The task of the smart card in this scenario is very simple. It stores at minimum the private RSA key, and it decrypts the hash value formed from the message, which means that it generates the signature. Everything else, such as generating the hash value or subsequently verifying the signature, can in principle be performed equally well by a PC.

Still, the ideal situation would be for the smart card to receive the message via its interface, compute the hash value and then send the signed message back to the terminal. Verification of the signature could also be performed by the smart card. This procedure is naturally no more secure than just computing the signature, but it is significantly more 'application-friendly'. This is because hash algorithms and RSA keys can be changed by simply exchanging the smart card, without any need to alter programs or data in the PC.

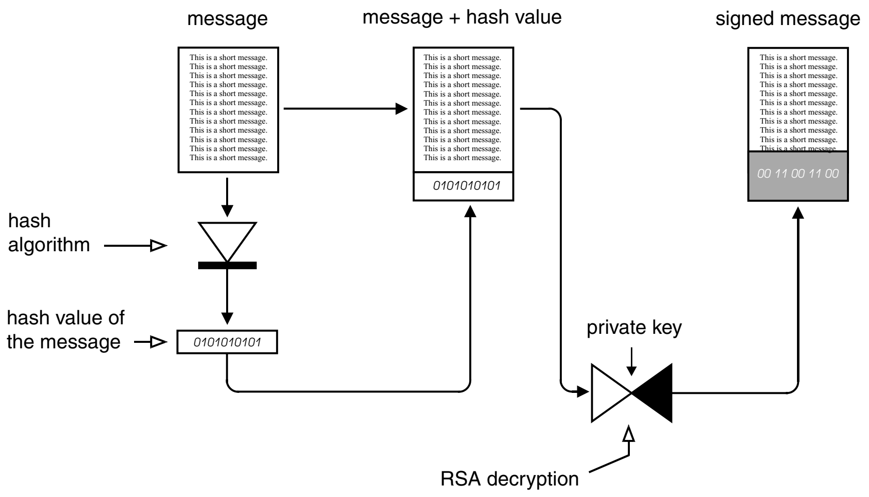


Figure 4.54 Signing a message with the RSA algorithm by incorporating the message and a hash value formed from the message in the signature (digital signature with message recovery)

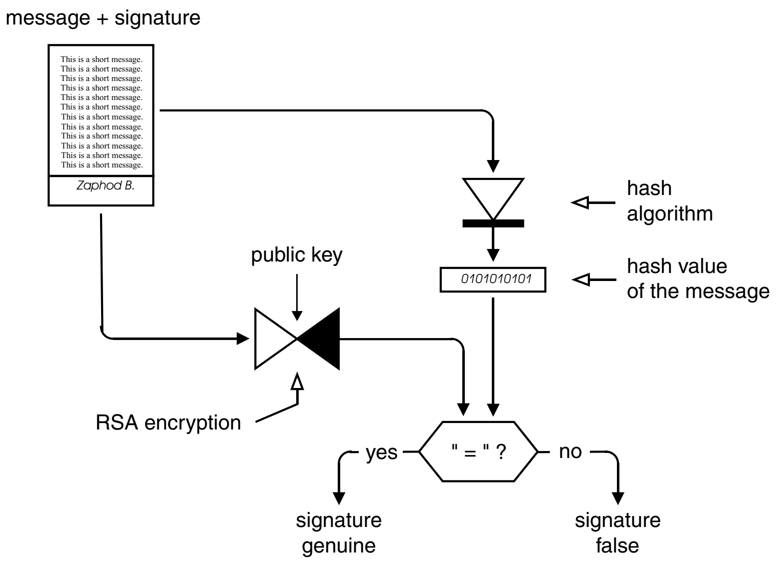


Figure 4.55 Verifying a message that has been signed using the RSA algorithm with the signature appended to the message (digital signature with appendix)

In these two examples, the keys used to generate and verify digital signatures are global, which means that they are the same for every smart card in a particular system. If a different arrangement should be used for security reasons, so that each card has its own key for digital signatures, a scheme such as that described in Section 4.11.3 must be used.

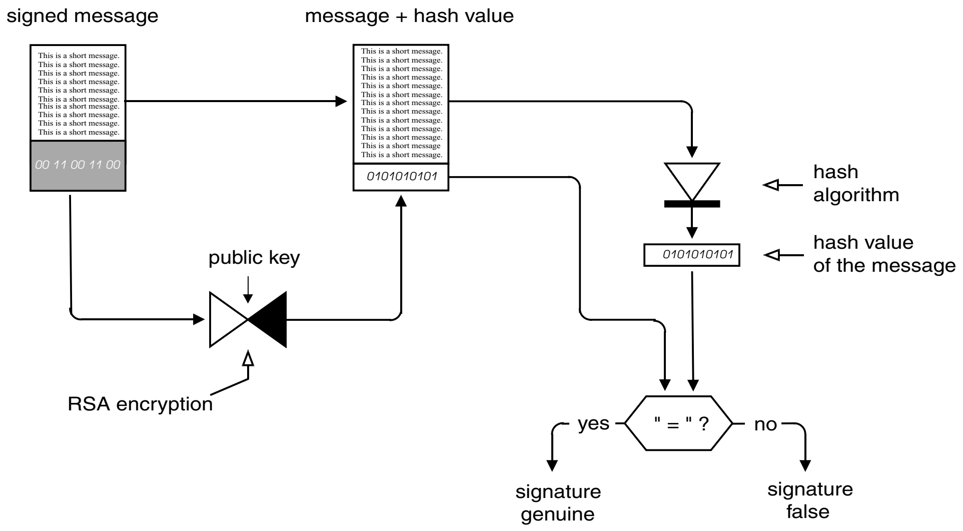


Figure 4.56 Verifying a message that has been signed using the RSA algorithm, in which a part of the message is used for the signature (digital signature with message recovery)

The RSA algorithm is not the only one that can be used to produce digital signatures. There is also a cryptographic procedure that has been specially developed for this application, namely the DSA (Digital Signature Algorithm). It was first proposed by the NIST (US National Institute of Standards and Technology) in 1991. Signatures can be both generated and verified using the DSA. In contrast to the RSA algorithm, it is largely designed such that it cannot be used for data encryption and decryption, although this has now been shown to not be true. Compared with the strong export restrictions applicable to the RSA algorithm and algorithms based on elliptic curves, such a feature would represent a major advantage for international use and/or export.

4.13 CERTIFICATES

With regard to the use of digital signatures, one is rather quickly confronted with a problem that must not be underestimated. Anyone who wants to verify the digital signature of a message needs the appropriate public key. However, the public key cannot be simply sent without any protection, since otherwise the recipient cannot verify the authenticity of the key. The public key must therefore be signed by a trustworthy body so that its authenticity can be verified. This body is called a certification authority (CA). The combination of a public key that has been signed by a certification authority, the accompanying digital signature and certain additional parameters is called a certificate.

There is also another body involved in this process, which is the trust center (TC). A trust center generates and manages certificates and associated blacklists, and it can optionally generate keys for digital signature cards. As a rule, a trust center also maintains a public directory of certificates, so that anyone who wants to verify a signed message can request the associated signed public key from the center, for example via the Internet.

A certificate contains not only the signed public key, but also a large number of additional parameters and options, since it must be possible to verify the public key of a certificate without any further information. From this, it follows that among other things, the algorithms used to generate the hash value and the signature must be clearly specified. In principle, everyone who signs documents could specify his or her own certificate structure. Of course, this would make it impossible to exchange certificates. This would generally rob such certificates of their meaning, since exchangeability is an essential characteristic of a certificate.

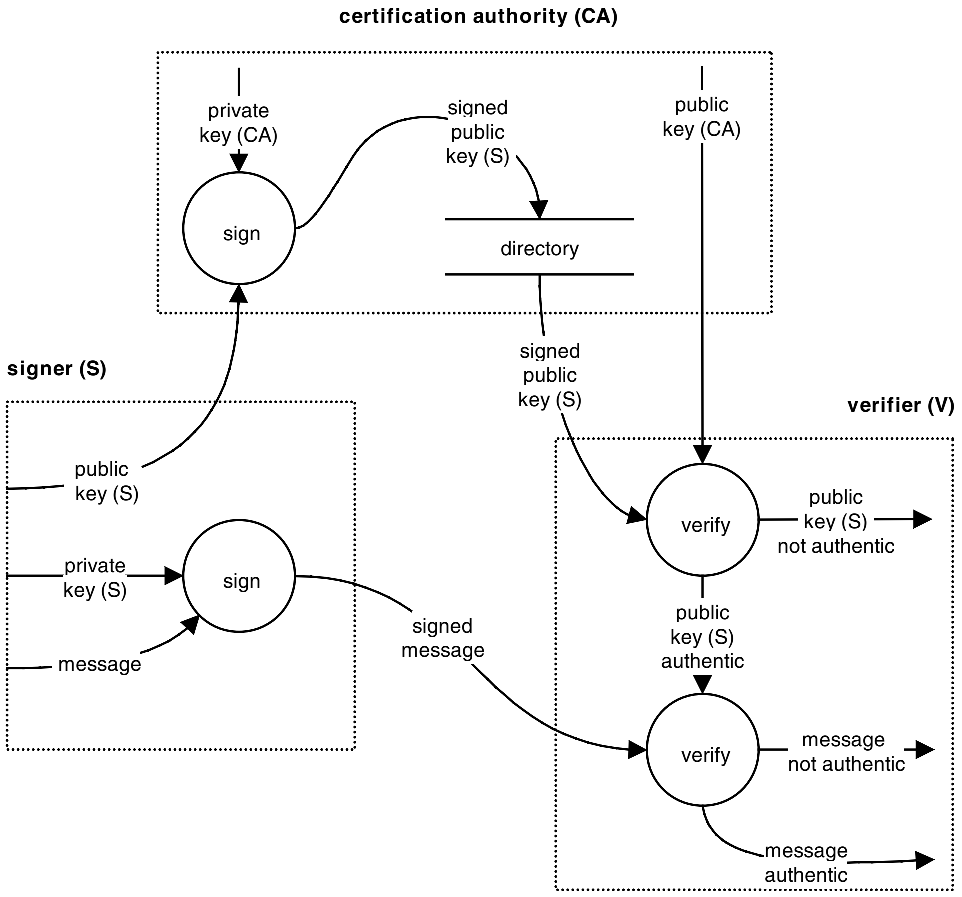


Figure 4.57 Data flow diagram of the basic processes for generating and verifying a transmitted message using a certificate. The certificate, which is generated by a certification authority, contains the public key of the signer and the signature of the certification authority

In order to insure that this sort of cooperation actually can take place, there are standards that specify the structure of certificates. The best known of the relevant standards is X.509, which specifies the structure and coding of certificates. It has also entered the ranks of ISO/IEC standards as ISO/IEC 9594-8

The comprehensive X.509 standard is a framework in which the structure of certificates is defined in unambiguous terms. It forms the basis for many digital signature applications. Some examples are the Secure Socket Layer (SSL) Internet security mechanism and the Privacy Enhanced Mail (PEM), Secure Multipurpose Internet Mail Extensions (SMIME) and Secure Electronic Transaction (SET) applications.

ASN.1 is consistently used in X.509 to describe certificates, and the widely known TLV coding scheme is used in accordance with the Distinguished Encoding Rules (DER) for the actual coding.[17] Some of the objects that may be included in a certificate are listed in Table 4.24. A brief introduction and summary of the subject of X.509 certificates can be found in a paper by Peter Gutmann [Gutmann 98b].

Table 4.24 Typical content of an X.509 certificate

Data element and X.509 designation	Explanation
Version	Identifies the version of X.509 that defines the data elements of this certificate. This is usually version 3.
Serial number	The serial number of the certificate. This must be issued by the issuer of the certificate, so that it is unique.
Signature algorithm identifier	Identifies the cryptographic algorithm used for the digital signature.
Issuer name	The name of the issuer of the certificate. The spelling of this name is unique, according to the X.500 standard.
Term of validity	The period for which the certificate is valid.
Subject name	The name of the entity whose public key should be recognized as authentic based on this certificate. According to the X.500 standard, the spelling of this name is unique
Public key	The subject entity's public key, which should be recognized as authentic based on this certificate.
Signature	The digital signature formed from the data of the certificate.

Many optional data fields for a wide variety of applications are defined in the X.509 standard. For example, it is easily possible to include several public keys in a single certificate and have them signed by different certification agencies. This can result in certificates that contain several kilobytes of data, which can cause problems if a smart card is to be used to verify the certificate. Of course, this scheme can also be used to generate items such as complementary certificates and tree-structured certificate hierarchies. A typical X.509 certificate in a smart card normally has a size of approximately 1 kB.

[17] See also Section 4.1, 'Structuring Data'

5
Smart Card Operating Systems

It may seem presumptuous to refer to the few thousand or ten thousand bytes of program code in a smart card microprocessor as an operating system, but the name is fully justified. According to the German DIN 44300 standard, an operating system is no more and no less than 'the programs of a digital computer system that together with the properties of the computing system form the basis for the possible operating modes of the digital computing system, and which in particular control and monitor program execution'.

The term 'operating system' is thus not automatically limited to enormous programs and data volumes. Instead, it is completely independent of size, since it only refers to functionality. You should not automatically associate the term 'operating system' exclusively with multi-megabyte programs for PCs and Unix computers. These operating systems are designed just as specifically for a particular man–machine interface, which uses a monitor, keyboard and mouse, as smart card operating systems are designed to work with the bidirectional serial interface to the terminal.

Ultimately, the decisive factor for an operating system is its functionality, which results from the interaction of mutually compatible and interdependent library routines. The fact that an operating system provides an interface between the computer hardware and the actual application software is also important, since it makes it unnecessary for the application software to directly address the hardware. This is a significant benefit, since it provides the application software with a certain amount of portability, even though this is often very limited.

At the beginning of the 1990s, there were very few true smart card operating systems. This was in part due to the very limited memory capacity of smart card microcontrollers at the time. The usual situation then was not so much an operating system as a well-structured collection of library routines in ROM, which were used as necessary for a particular application when the card was completed. The structures of these systems were largely monolithic and could be modified only at considerable expense. The next generation was already built in the form of a layered operating system, and present-day systems still have this structure, with innumerable refinements.

One of the first true smart card operating systems was STARCOS, which was developed by Giesecke & Devrient [GD] and the *Gesellschaft für Mathematik und Datanverarbeitung*. This operating system, whose development began in 1990, allowed several applications to be stored, used and managed independently in a single smart card, even at that relatively early date.

Smart Card Handbook, Third Edition. W. Rankl and W. Effing
© 2004 John Wiley & Sons, Ltd ISBN: 0-470-85668-8

In the course of time, the term COS (card operating system) has become accepted throughout the world as a designation for a smart card operating system. It often forms part of the name of the operating system, as with ‘STARCOS’ and ‘MPCOS’. Presently, there are more than a thousand companies that produce general-purpose and application-specific smart card operating systems. A number of examples are listed in Table 5.1.

Table 5.1 Some examples of smart card operating systems from various producers, with references to their WWW addresses. This is only an incomplete selection

Operating system name	Producer
GemXplore, GPK, MPCOS	Gemplus [Gemplus]
STARCOS, STARSIM, STARDC	Giesecke & Devrient [GD]
Multos	Maosco [Maosco]
AuthentIC, SIMphonic	Oberthur [Oberthur]
Micardo	Orga [Orga]
Cyberflex, Multiflex, Payflex	Schlumberger [Schlumberger]
CardOS	Siemens [Siemens]
TCOS	Telesec [Telesec]

It is conceivable that a consolidation of smart card operating systems, with their various features and functions, could occur in the next few years. This would have the same effect as with PCs, which nowadays all use a ‘uniform’ operating system. Whether this will actually happen with smart card operating systems remains to be seen, since in this case the external conditions are less favorable to a uniform solution. Extremely severe requirements with regard to security and software quality, a shortage of memory capacity and the demand for confidentiality of the operating system software, taken together, certainly have the potential to make it impossible to produce a universal smart card operating system that satisfies everyone’s wishes, at least in the foreseeable future.

In this chapter, we attempt to shed some light on the features and varieties of modern smart card operating systems, based on various specifications, standards and descriptions of software for smart cards.

5.1 HISTORICAL EVOLUTION OF SMART CARD OPERATING SYSTEMS

The evolution of operating systems for smart cards has passed through the same stages as for all other computer systems. The original special-purpose programs for single applications were repeatedly generalized and extended. The end result was a structured, general-purpose operating system that is easy to use.

From a modern perspective, the programs for smart card microcontrollers that existed at the beginning of the 1980s, at the start of this evolution, cannot be considered to be true operating systems. They were nothing more than application programs embedded in the ROM of a chip. However, since manufacturing mask-programmed microcontrollers is expensive and time-consuming, the demand for general-purpose kernel routines grew very quickly. Special-purpose application programs stored in EEPROM could then be built on top of these routines, using them as necessary. However, this also increased the demand for memory, which led some companies to move in the opposite direction, back to special software for specific applications.

However, a steadily increasing market demand for individually tailored solutions, which continues to exist, has more or less forced producers of operating systems to design their programs to meet this demand. Presently, the individualized approach with specially developed ROM software is only used for applications involving a very large number of cards. General-purpose operating systems based on standard commands are the norm, since in principle they can be used for any type of application. If this is not possible for some reason, these operating systems are at least designed so that they can be modified to meet the requirements of any particular application with a minimum of time and effort.

The 'historical' development of smart card operating systems from 1980 to the present can be very well illustrated by the smart cards used in the German mobile telephone networks. The smart cards used since 1987 in the C-Netz (the German precursor of the GSM, or D-Netz) have an operating system that is optimized for this application. The modifications include a custom transmission protocol, special commands and a file structure specially adapted to the application. All in all, this type of card certainly has a complete operating system, but it is totally tailored for use in the C-Netz. The essential components of the application based on this operating system were tailored to the special requirements of the library-oriented operating system and its underlying hardware.

The next step was the transition from a special solution to a somewhat more open operating system architecture. One representative of this is the first GSM card, whose design is significantly more open and multifunctional. When GSM smart cards were specified, there were already proposed standards for the command sets and data structures of smart cards, which meant that the cornerstone had been laid for compatibility among various operating systems. Starting from this basis, further developments came step by step. Applications based on this operating system are largely independent of the hardware and are based on the various interfaces of the operating system.

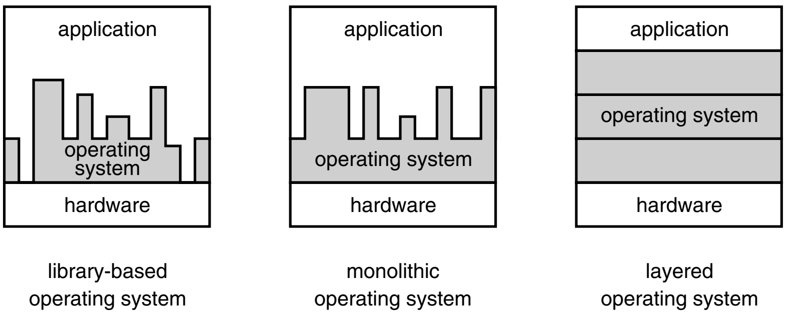


Figure 5.1 Schematic representation of the historical development of smart card operating systems. Up to around 1991, newly developed operating systems were library-based. They were gradually replaced by monolithic operating systems up until around 1998. Monolithic operating systems have now been replaced by layered operating systems. For the sake of simplicity, each of the diagrams shows only one application

Modern operating systems for GSM have functions such as memory management, multiple file trees and state machines, which brings them very close to the features provided by multiapplication operating systems. They can manage several applications independently while preventing interaction among the applications. Most of them also have very complex state machines and a large command set, and some of them support several data transmission protocols.

Even these operating systems do not represent the end of the evolution. Smart cards for mobile telephones are taking on more and more of the functions of the telephone itself, such as driving the display and polling the keypad. In order to provide the maximum possible flexibility, it is necessary to break with what has up to now been a rigid fundamental principle in the smart card world. A modern smart card operating system must be able to run third-party program code in the card. With smart card operating systems, it goes without saying that this will not have any detrimental effect on the functionality or security of any other applications in the card. All modern smart card operating systems have a layered structure, with only the bottom

Table 5.2 Source code size and characteristic code metrics of typical smart card operating systems for the GSM telecommunications application. Adding Java Card capability to the operating system dating from 2002 would increase the size of the source code by approximately 180,000 lines

Type of smart card operating system	OS for GSM circa 1990, in assembler	OS for GSM circa 1997, in assembler	OS for GSM circa 1996, in C	OS for GSM circa 2002, in C
Functionality	GSM 11.11, administrative commands	GSM 11.11, administrative commands, OTA	GSM 11.11, administrative commands	GSM 11.11, GSM 11.14, WIM, 3 microbrowsers, administrative commands, OTA
Microcontroller specifications	6 kB ROM 3 kB EEPROM 128 bytes RAM	16 kB ROM 8 kB EEPROM 256 bytes RAM	16 kB ROM 8 kB EEPROM 256 bytes RAM	196 kB ROM 68 kB EEPROM 4096 bytes RAM
Total size of linked object code (ROM + EEPROM)	10 kB + 0.3 kB	16 kB + 0.8 kB	16 kB + 0.8 kB	190 kB + 4.9 kB
Lines of source code [1] (including comments)	≈10,000	≈22,000	≈12,000	≈120,000
Number of source code files	22	14	37	184
Number of subroutine calls (assembler) or functions (C)	≈470	≈930	≈115	≈900
Number of returns	≈95	≈35	≈205	≈1500
Number of constants	≈360	≈250	≈100	≈150
Number of branches (assembler) or IF statements (C)	≈200	≈560	≈1100	≈3100

[1]In 2000, the largest known software program was Windows 2000, with 29 million lines of source code.

layers being dependent on the hardware. The hardware is increasingly abstracted in the layers built on top of these lower layers.

In the foreseeable future, it is certainly possible for the evolution of smart card operating systems to lead (via several intermediate steps) to an international quasi-standard for general-purpose operating systems, as has already occurred with many other types of operating systems. It may take a while, but sooner or later a certain industry standard comes to predominate, and competitors in the market must support it as the ‘least common denominator’ if they wish to continue to operate successfully. Such a standard does not yet exist in the smart card world, but the first signs of one can already be seen. The basis for this, in contrast to the PC world for example, is formed by international standards and specifications. These are primarily the ISO/IEC 7816 family of standards, the UICC specifications (TS 102.221) and the EMV specifications.

5.2 FUNDAMENTALS

Smart card operating systems, in contrast to commonly known operating systems, do not include user interfaces or the possibility of accessing external memory media. This is because they are optimized for completely different functionality. Security during program execution and protected access to data have the highest priority. Due to the limitations imposed by the amount of available memory, smart card operating systems have a very small amount of program code, which normally lies in the range of 3–250 kB. The lower limit relates to special applications, while the upper limit relates to multiapplication operations systems, some of which have interpreters for downloadable program code. However, the average memory requirement is around 64 kB for operating systems that do not support downloadable third-party program code.

The program modules are written as ROM code. This limits the programming techniques that can be used, since many techniques typically used with RAM program code (such as self-modifying code) are not possible with ROM code. The fact that the code is in ROM also explains why no changes at all are possible once the microcontroller has been programmed and manufactured. Correcting an error is thus extremely expensive and takes 10 to 12 weeks. If the smart cards have already been issued, errors can only be corrected by a large-scale recall campaign, which could destroy the reputation of a system using smart cards. ‘Quick and dirty’ programming is thus clearly out of the question. Consequently, the amount of time spent on testing and quality assurance is usually significantly greater than the amount of time spent on programming.

These operating systems must not only have a very small number of errors, they must also be very reliable and robust. They must not allow their functionality, and above all their security, to be impaired by any command coming from the outside world. System crashes or unpredictable responses to an erroneous command or the failure of a page of EEPROM must never occur under any circumstance.

From the perspective of operating system design, it is an unfortunate fact that the implementation of certain mechanisms is influenced by the hardware that is used. Above all, the secure state of the EEPROM has a small but noticeable effect on the design of the operating system. For example, all retry counters must be designed such that their maximum value corresponds to the erased state of the EEPROM. If this were not the case, it would be possible to reset a counter to its initial value by (for example) intentionally switching off power to the card while it is writing a new value to the retry counter. This is because the EEPROM must be erased before certain types of write operations. If the power can be switched off exactly at the time between erasing the EEPROM and writing the new value, the portion of the EEPROM used for the retry counter would be in the erased state. If the operating system were designed incorrectly, this would amount to resetting the retry counter to its initial value. This type of attack can be countered by properly coding the counter, as just described, or by making the process of writing the retry counter an atomic operation. The situation with regard to the retry counter and the secure (lowest energy) state of the EEPROM is similar. The retry counter must be coded such that its maximum count corresponds to the secure state of the EEPROM. If this were not the case, it would be possible to reset the retry counter (by locally heating certain EEPROM cells, for example). These are only two examples of hardware dependences in the design of a smart card operating system; there are many others. For reasons of security, a smart card operating system must be closely coupled to the hardware of the microcontroller used. Consequently, it can never be fully hardware-independent.

There is also another aspect to the concept of a secure operating system. ‘Trap doors’ and other types of hidden access points for system programmers are frequently found in large operating systems, in which they are perfectly normal features. However, they must be totally excluded in smart card operating systems. There must not be any possibility that someone could use some mechanism to bypass the operating system and (for example) obtain unauthorized read access to a file.

Another aspect that should not be underestimated is the required processing capacity. The cryptographic functions present in the operating system must execute in very short lengths of time. It is thus common to expend weeks of painstaking effort to optimize the algorithms in question in assembler code.

Due to the hardware platforms used and the required level of reliability, it is obvious that there is no place for multitasking capability in smart card operating systems. However, the limitation to a single executing task also restricts the use of security processes that monitor operating system components with regard to process execution and constraints.

In summary, the primary tasks of a smart card operating system are the following:

- transferring data to and from the smart card
- controlling the execution of commands
- managing files
- managing and executing cryptographic algorithms
- managing and executing program code.

Command processing

Command processing in smart cards that do not support downloadable program code is typically organized as shown in Figure 5.2. The smart card receives each command via the serial I/O interface. The I/O manager performs error detection and correction procedures as necessary, fully independent of the other, higher level layers. After a command has been completely received without errors, the secure messaging manager must decrypt the message or test its integrity. If secure data transmission is not used, this manager is completely transparent to both the command and the response.

After this processing, the next higher level, which is the command interpreter, attempts to decode the command. If this is not possible, the return code manager is called. It generates a suitable return code and sends it back to the terminal via the I/O manager. It may be necessary to design the return code manager in an application-specific fashion, since the return codes are not necessarily the same for all applications. If the command can be decoded, the logical channel manager determines which channel has been selected, switches to the state of this channel and invokes the state machine if no error occurs.

The state machine checks whether the command, in combination with its accompanying parameters, is permitted in the current state of the smart card. If it is, the program code of the application command that processes the received command is executed. If the command is prohibited in the current state, or if its parameter values are not allowed, the terminal receives a message to this effect via the return code manager and the I/O manager.

If it is necessary to access a file while processing the command, this occurs only via the file manager, which converts all logical addresses into physical addresses within the chip. The file manager also monitors all addresses with regard to region boundaries and tests access conditions for the file in question.

The file manager utilizes a lower level memory manager, which performs all management functions for the physically addressed EEPROM. This ensures that this program module is the only one that uses physical addresses, which significantly increases the portability and security of the entire operating system.

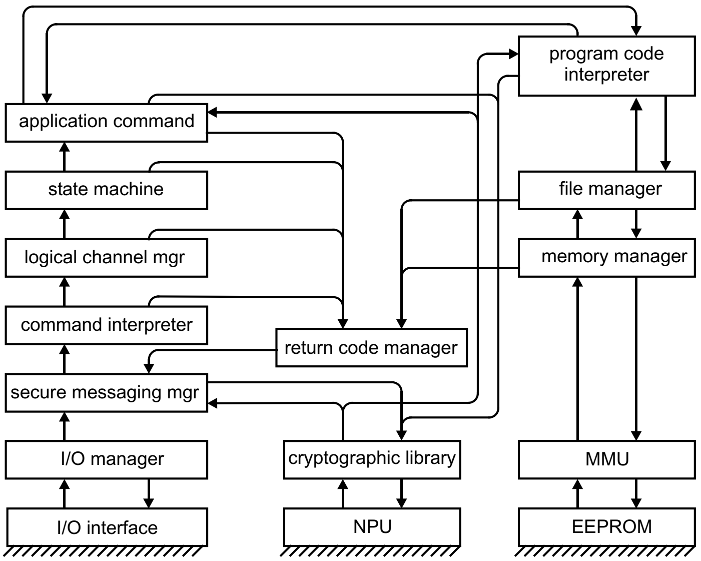


Figure 5.2 Command processing within a smart card operating system. The level of hardware abstraction increases from the bottom to the top in this figure. The program code interpreter is optional

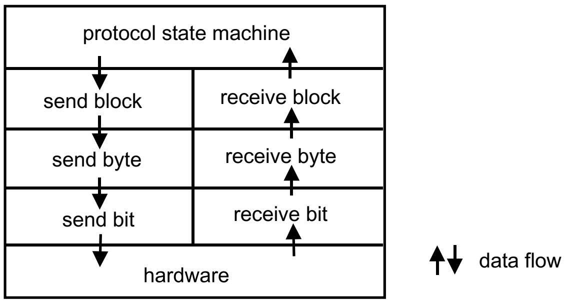


Figure 5.3 The portion of the command processing sequence performed within the I/O manager of a smart card operating system

A central return code manager is responsible for generating the return code. It always produces the complete response sent to the calling routine. This level is responsible for managing and generating the return codes used in all other parts of the operating system.

Since smart card operating systems usually utilize cryptographic functions, there is normally also a dedicated library of cryptographic functions separate from the rest of the operating system. This library serves all other modules as a central point of departure for using cryptographic functions.

In addition to these levels, an interpreter or verification routine for executable files may be present in the region above the application command level. It monitors the programs contained in these files and runs or interprets them. The exact design and implementation depends on whether there are actually any files with executable code present, and on whether the stored code is machine code for the processor or code to be interpreted. This subject is described in detail in Section 5.10, 'Smart Card Operating Systems with Downloadable Program Code'.

Smart card profiles

In contrast to the realm of PC operating systems, the memory space of smart cards is so severely limited that in many cases not all of the standard commands and file structures are implemented. Consequently, 'profiles' for smart cards are included in the two relevant standards for general-purpose operating systems (EN 726-3 and ISO/IEC 7816-4). Each profile defines a subset of the commands and file structures of the standard in question.

A smart card that matches a certain profile must at least incorporate the defined subset for that profile. However, the descriptions of the profiles are contained in appendices in both standards and are designated 'informational' instead of normative. They thus represent only recommendations to operating system designers. The five profiles defined in the ISO/IEC 7816-4 standard are summarized in Table 5.3.

Commercial smart card operating systems normally support various types of smart card microcontrollers with various amounts of memory. Consequently, there are in practice also operating system profiles that specify certain ranges of functions, depending on the chip type. These profiles within an operating system are normally designed such that applications can migrate relatively easily, at least from a smaller memory to a larger one, without any changes to commands or file structures.

Table 5.3 Brief descriptions of the smart card profiles defined in the ISO/IEC 7816-4 standard. The listed file structures and commands represent the minimum requirement for each profile

Profile	Description	
M	File structures:	• transparent • linear fixed
	Commands:	• READ BINARY, UPDATE BINARY without implicit selection; maximum length 256 bytes • READ RECORD, UPDATE RECORD without implicit selection • SELECT FILE with explicit specification of the FID • VERIFY • INTERNAL AUTHENTICATE
N	as Profile M, with the supplementary use of a DF name for SELECT FILE	
O	File structures:	• transparent • linear fixed • linear variable • cyclic
	Commands:	• READ BINARY, UPDATE BINARY without implicit selection; maximum length 256 bytes • READ RECORD, UPDATE RECORD without implicit selection • APPEND RECORD • SELECT FILE • VERIFY • INTERNAL AUTHENTICATE • EXTERNAL AUTHENTICATE • GET CHALLENGE
P	File structures:	• transparent
	Commands:	• READ BINARY, UPDATE BINARY without implicit selection; maximum length 64 bytes • SELECT FILE with explicit specification of the DF name • VERIFY • INTERNAL AUTHENTICATE
Q	Data transmission:	• secure messaging
	File structures:	• —
	Commands:	• GET DATA • PUT DATA • SELECT FILE with explicit specification of the DF name • VERIFY • INTERNAL AUTHENTICATE • EXTERNAL AUTHENTICATE • GET CHALLENGE

5.3 DESIGN AND IMPLEMENTATION PRINCIPLES

As is well known, design errors first manifest themselves in the implementation stage, where they result in costs that are several times greater than those for a better design with fewer errors. However, errors are a fact of life in all software projects. In order to minimize such errors, it is recommended to observe several principles during the design and implementation of a smart card operating system.

Due to its functional specification, a smart card operating system is a secure operating system designed to manage information and above all keep information confidential. In addition, it is normally not possible to make any changes at all to the software once it is in use. The first principle follows directly from these considerations. A smart card operating system must be extremely reliable, which means it must have an extremely small number of errors. Total elimination of errors is in reality never achievable, since even the smallest smart card operating systems are too large for all of the capabilities of their internal processes to be completely tested.

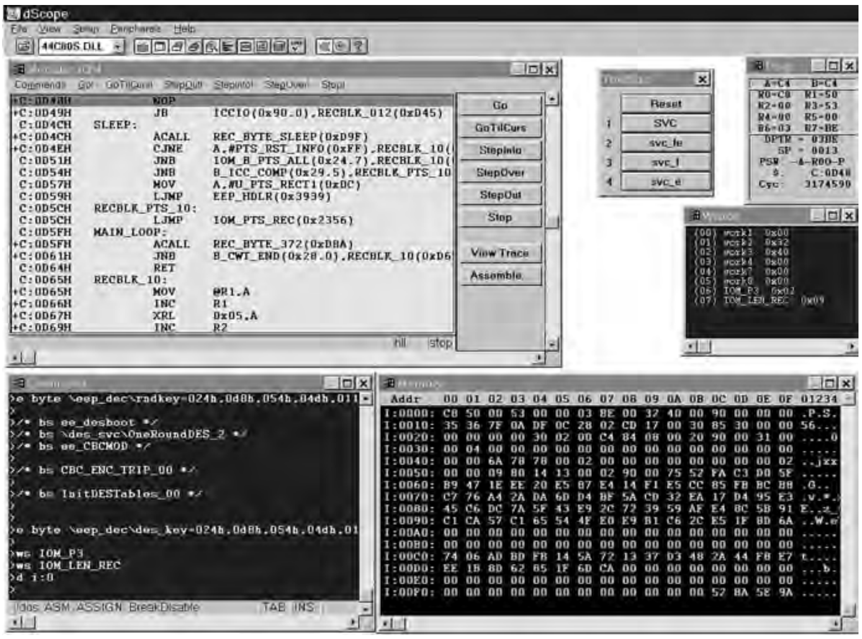
However, strict modular design makes a decisive contribution to the discovery and elimination of any errors that may be present at the implementation stage. This modularity, which strongly increases the reliability of the operating system, need not necessarily result in large increase in the volume of program code. An additional benefit of a modular design is that any system crashes that may occur generally do not affect the system's security as strongly as with highly optimized program code occupying less memory. With a modular design, the consequences of any errors remain localized, and the operating system as a whole is more robust and more stable.

The fact that the software most often must be implemented in assembler makes it more prone to errors. A design based on individual, fully testable modules strongly contributes to detecting programming errors in a timely manner and limiting the scope of their effects, due to use of defined interfaces. The layered operating system architecture shown in Figure 5.1 is a consequence of this modular approach. The increased amount of planning and programming effort that this requires is offset, at least financially, by the fact that testing and verification are significantly easier. For this reason, almost all current operating systems have an architecture essentially equivalent to the one described here.

The method usually used to design an operating system is based on the module–interface concept. In the design process, the tasks of the operating system and the application are broken down as much as possible into functions, and these functions are then incorporated into modules. Once the interfaces to the modules have been exactly defined, the individual modules can be programmed, possibly by several persons. In the ideal case, the first version of the implementation is platform-independent, which means that it does not yet depend on the characteristics of the microcontroller that is used. After this version has been completely tested, the necessary adaptations to the microcontroller can be made.

Since the amount of program code for a smart card operating system is relatively small, this very pragmatic approach can be used without any major problems. Its advantages, which are the relatively small amount of planning that is needed and the possibility of dividing the programming tasks among several persons, along with reusability of the program code, yield the maximum benefits here. The drawbacks of this approach, which are the difficulty of demonstrating the correctness of the system and the fact that changes to the system may strongly affect many modules, must be balanced against its benefits.

The development of software for smart card operating systems is moving away from programming entirely in assembler. Many recent projects have been executed from the very beginning in the high-level language C, which is relatively close to the hardware. However, the kernel of the operating system is still based on machine-dependent assembler routines, with all higher level modules, such as the file manager, the state machine and the command interpreter, being programmed in C. This significantly reduces the development time and makes the code more portable and more reusable. Above all, using a high-level language markedly improves the testability of the software. The improved, more easily understood program structure provided by a high-level language also yields a distinctly lower error rate.


dScope
File View Setup Peripherals Help
44C80S.DLL
Commands Go! GoTilCurs! StepOut! StepInto! StepOver! Stop!
Go
GoTilCurs
StepInto
StepOver
StepOut
Stop
View Trace
Assemble
hll stop
Reset
1 SVC
2 svc_fe
3 svc_f
4 svc_e
ldos ASM ASSIGN BreakDisable TAB INS

```
+C:0D48H            NOP
+C:0D49H            JB      ICCIO(0x90.0),RECBLK_012(0xD45)
 C:0D4CH   SLEEP:
+C:0D4CH            ACALL   REC_BYTE_SLEEP(0xD9F)
+C:0D4EH            CJNE    A,#PTS_RST_INFO(0xFF),RECBLK_10(
 C:0D51H            JNB     IOM_B_PTS_ALL(0x24.7),RECBLK_10(
 C:0D54H            JNB     B_ICC_COMP(0x29.5),RECBLK_PTS_10
 C:0D57H            MOV     A,#U_PTS_RECT1(0xDC)
 C:0D59H            LJMP    EEP_HDLR(0x3939)
 C:0D5CH   RECBLK_PTS_10:
 C:0D5CH            LJMP    IOM_PTS_REC(0x2356)
 C:0D5FH   MAIN_LOOP:
+C:0D5FH            ACALL   REC_BYTE_372(0xD8A)
+C:0D61H            JNB     B_CWT_END(0x28.0),RECBLK_10(0xD6
 C:0D64H            RET
 C:0D65H   RECBLK_10:
+C:0D65H            MOV     @R1,A
+C:0D66H            INC     R1
+C:0D67H            XRL     0x05,A
+C:0D69H            INC     R2
```

```
 A=C4   B=C4
R0=C8  R1=50
R2=00  R3=53
R4=00  R5=00
R6=03  R7=BE
 DPTR = 03BE
   SP = 0013
 PSW  -A-R00-P
   $:   C:0D48
 Cyc:   3174590
```

```
(00) work1  0x00
(01) work2  0x32
(02) work3  0x40
(03) work4  0x00
(04) work7  0x00
(05) work0  0x00
(06) IOM_P3  0x02
(07) IOM_LEN_REC  0x09
```

```
>e byte \eep_dec\rndkey=024h,0d8h,054h,04dh,01
>
>/* bs ee_desboot */
>/* bs \des_svc\OneRoundDES_2 */
>/* bs ee_CBCMOD */
>
>/* bs CBC_ENC_TRIP_00 */
>
>/* bs InitDESTables_00 */
>
>
>e byte \eep_dec\des_key=024h,0d8h,054h,04dh,01
>
>ws IOM_P3
>ws IOM_LEN_REC
>d i:0
>
```

```
 Addr    00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 01234
I:0000: C8 50 00 53 00 00 03 BE 00 32 40 00 90 00 00 00 .P.S.
I:0010: 35 36 7F 0A DF 0C 28 02 CD 17 00 30 85 30 00 00 56...
I:0020: 00 00 00 00 30 02 00 C4 84 08 00 20 90 00 31 00 ....0
I:0030: 00 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
I:0040: 00 00 6A 78 78 00 02 00 00 00 00 00 00 00 00 02 ..jxx
I:0050: 00 00 09 80 14 13 00 02 90 00 75 52 FA C3 D0 5F .....
I:0060: B9 47 1E EE 20 E5 87 E4 14 F1 E5 CC 85 FB BC B8 .G..
I:0070: C7 76 A4 2A DA 6D D4 BF 5A CD 32 EA 17 D4 95 E3 .v.*.
I:0080: 45 C6 DC 7A 5F 43 E9 2C 72 39 59 AF E4 8C 5B 91 E..z_
I:0090: C1 CA 57 C1 65 54 4F E0 E9 B1 C6 2C E5 1F 8D 6A ..W.e
I:00A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
I:00B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
I:00C0: 74 06 AD BD FB 14 5A 72 13 37 D3 48 2A 44 F8 E7 t....
I:00D0: EE 1B 8D 62 85 1F 6D CA 00 00 00 00 00 00 00 00 ...b.
I:00E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
I:00F0: 00 00 00 00 00 00 00 00 00 00 00 00 57 8A 5E 9A .....
```

Figure 5.4 An example of the simulation of a smart card microcontroller in a typical development environment for the C and assembler programming languages. The source code pane is at the top left, with various processor registers being shown at the top right. A memory map of the RAM is shown at the bottom right, and the command lines for controlling the simulator are shown at the bottom left. This simulator allows the software developer to monitor all the functions of the microcontroller and intervene at every stage of program execution (*Source:* Keil)

Unfortunately, the program code generated by a C compiler, even if it is highly optimized, occupies 20 to 40 % more space in ROM than equivalent code in assembler for the same functionality. In addition, the speed of a C implementation is marginally lower than an assembler implementation. However, this is only critical for cryptographic algorithms and the data transmission protocol, since the other parts of the smart card operating system software normally do not contain time-critical processes.

The greatest problem with C programming is not necessarily the extra memory space needed in the ROM or reduced execution speed, but the amount of RAM that is used. This type of memory is extremely limited in smart cards, and it has the further disadvantage that it takes up the most space per bit on the chip. This is the main reason why high-level languages have up to now been used rather sparingly in programming smart card operating systems.

Table 5.4 Some examples of typical memory usage for smart card operating system functions implemented in assembler

Function	Required program memory
CRC algorithm	≈50 bytes
File management (MF, 2 DF levels, EF and 4 EF structures)	≈1200 bytes
DES algorithm (not SPA/DPA-resistant)	≈1200 bytes
DES algorithm (SPA/DPA-resistant)	≈2000 bytes
EEPROM write	≈150 bytes
RSA algorithm (with NPU)	≈300 bytes
Data transmission protocol T = 0	≈500 bytes
Data transmission protocol T = 1	≈1200 bytes

Since smart cards are used in application areas in which security is a very important factor, the card issuer and/or application supplier must have considerable trust in the integrity of producer of the operating system. The latter has every opportunity to take unfair advantage of the entire system by means of deliberately introduced security gaps. For example, consider an electronic purse in a smart card whose load command has been manipulated such that under certain conditions the purse can be reloaded without authorization. Such scenarios are the reason why only a few operating system producers have become established up to now. The risk that an ostensibly secure operating system contains a Trojan horse is significantly greater if it comes from a small, unknown vendor than if it comes from one of the well-known firms in the field.

Nevertheless, there has recently been an increasing effort to evaluate smart card operating systems according to the ITSEC or its successor, Common Criteria, in order to achieve a higher level of comprehensibility and security with regard to such scenarios.[1] This occurs either on the initiative of the producer or in response to the demands of major card issuers, whose objective with such an evaluation is to achieve an increased level of confidence that there are no significant errors in the program code. Checking for deliberately introduced Trojan horses during an evaluation would probably be of limited use, since there is practically no limit to the number of ways a Trojan horse can be incorporated into a program.

Up to now, the usual evaluation levels for smart card operating systems have been ITSEC E3 and E4. Level E6 is also occasionally demanded or provided. It must be borne in mind that an evaluation of a complete smart card operating system at the E4 level can cost around €300,000. The obligation to repeat the evaluation after any modifications to the program code comes on top of this, although a re-evaluation is of course less costly than an initial evaluation. This is essentially why only relatively few smart card operating systems can boast ITSEC

[1] See also Section 9.3, 'Evaluating and Testing Software'

evaluations. More commonly, evaluations are performed by testing agencies without using the ITSEC. In such cases, testing is limited to a thorough review of the design criteria, the source code and the documentation. For example, this is a fundamental requirement for operating systems used in German Eurocheque cards.

5.4 COMPLETION

The life cycle of a smart card operating system passes through two major stages – that before completion and that after completion. In the stage before the card is completed, in which the microcontroller comes from the semiconductor plant with an empty EEPROM, all parts of the program run in ROM. No data are read from the EEPROM, nor is any code run from the EEPROM. If an error in the ROM code that makes completion impossible is discovered at this time, the entire batch of microcontrollers must be destroyed, since the chips are unusable.

To minimize the likelihood of such a situation, it would be possible to have the ROM contain only a small routine for loading the EEPROM, and load the actual operating system into the EEPROM. However, the chip area per bit is four times as much with EEPROM as with ROM, which means that such an approach would excessively increase the cost of the chip. For purely economic reasons, therefore, as much of the code as possible must be located in ROM. Consequently, all of the core operating system routines, as well as substantial parts of the rest of the operating system, are stored completely in the ROM. Only a few jumps to the EEPROM are provided for use in the completed version.

Some operating systems run completely in ROM even after completion, with only data being stored in the EEPROM, in order to keep the size of the relatively expensive EEPROM as small as possible. Of course, minimizing the area used by the memory comes at the price of major limitations on the flexibility of the operating system.

In the completion process, the code in the ROM is adapted to the actual application. The ROM code can be regarded as a large library that is interconnected and expanded to form a functional application by the code in the EEPROM. In addition, almost all operating systems allow program code for additional commands or special cryptographic algorithms to be loaded into the EEPROM during completion. This has nothing to do with any executable files that may be present, since the contents of such files can be downloaded at a later time, such as when the card is personalized. The routines that are entered into the card during completion are completely integrated into the operating system and can be utilized directly by the operating system.

In order to complete a smart card operating system in the EEPROM, unilateral or mutual authentication between the smart card and the outside world is necessary. There are many different methods that can be used for this. Figure 5.6 shows a typical example that uses a simple but relatively flexible approach. The producer of the smart card operating system incorporates a secret value in the ROM code for the microcontroller, which is thus specific to this ROM version of the operating system. During chip fabrication, the semiconductor manufacturer writes a secret key to the EEPROM, which may have a different value for each fabrication batch. After this, if the ROM and EEPROM keys are known, a cryptographic algorithm can be used to compute an authentication value for smart card completion. The operating system can be completed only after successful authentication.

This method allows batch-specific authentication to be performed. For instance, if an operating system producer provides only one authentication value to a card manufacturer, the latter

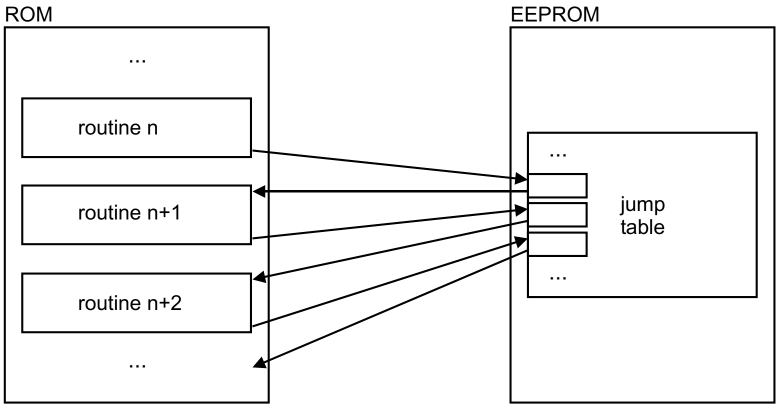


Figure 5.5 Routines in ROM can be interconnected by a link table that is stored in the EEPROM when the operating system is completed

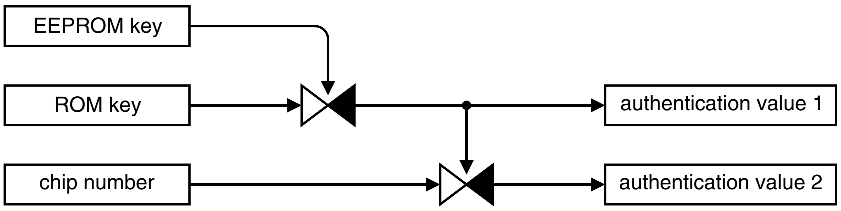


Figure 5.6 A possible procedure for dividing and deriving the secret key for authentication prior to completion of the smart card operating system. Authentication value 1 is specific to a batch of smart card microcontrollers, while authentication value 2 is different for each microcontroller. Triple DES or AES could be used as the cryptographic algorithm

can only use it to complete one particular batch of smart card microcontrollers. The advantage of this method is that the producer of the operating system only has to have one ROM mask, instead of generating a different version of the ROM for each card manufacturer.

This method can be further refined. For instance, a chip-specific authentication value can be generated using the microcontroller serial number. Not only does this increase the level of security if the chips fall into the wrong hands on their way to being completed, it also allows the producer of the operating system to control the completion process at the individual chip level. With such an approach, the completion agent receives a list of authentication values and can only complete the operating system in the corresponding microcontrollers. This method, or a similar method, is commonly used in practice.

Conceivably, a special smart card containing a function for reading out the contents of the memory could be smuggled into the smart card manufacturing process and be completed, initialized and personalized, after which the secret data loaded into memory during these steps could be read out. Although this would be rather difficult in a typical smart card manufacturing process, due to the strong security measures that are used, it is certainly not possible to fully exclude such a scenario. In order to provide inherent defense against this conceivable form of attack, the authenticity of the microcontroller can be verified prior to completion. This can be done in a large variety of manners, one of which is briefly illustrated in Figure 5.7.

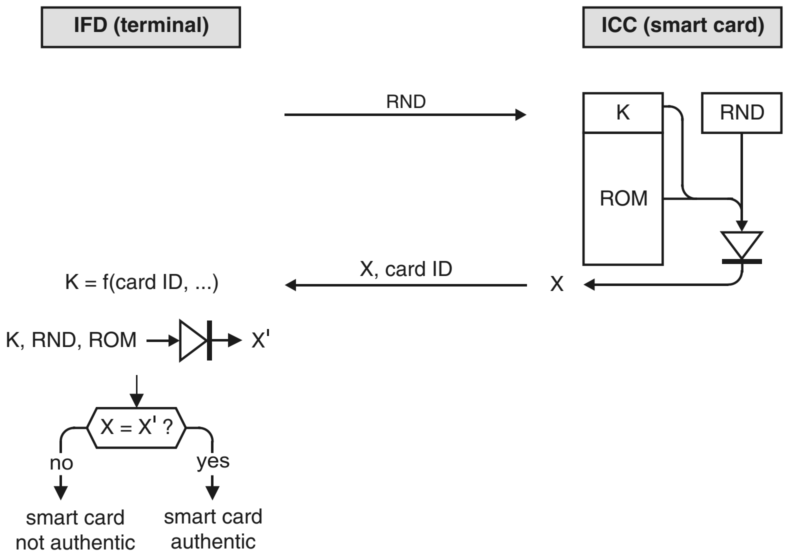


Figure 5.7 A possible method for ensuring that the ROM portion of a smart card operating system is authentic. This sort of verification is best performed prior to completing the operating system in EEPROM, in order to securely avoid loading genuine completion data into a counterfeit card

This method requires an individual key to be stored in each microcontroller by the semiconductor manufacturer. Prior to completion, a random number is sent to the smart card in order to prevent playback attacks. This random number, the individual card key and the content of the ROM are then converted into a hash value by a hash function in the smart card, and this hash value is conveyed to the outside world along with a card identifier. There the card-specific key can be computed using the card identifier. Since the ROM content and the random number are known to the outside world, the hash computation can be reproduced in the outside world, and the result can be compared with the hash value received from the smart card. If the two values match, the smart card is authentic. Although this method is simple, it is without doubt sufficiently effective. Certain aspects can be further refined, but our primary objective here is to present a conceivable approach to verifying authenticity prior to completion.

Hardware recognition

Most relatively recent operating systems can be used with microcontrollers having various amounts of EEPROM, although the size of the ROM and RAM are not allowed to vary. This allows the card issuer to always use the least expensive version of the microcontroller that meets his needs. For example, he could start with an inexpensive single-application card with 1 kB of EEPROM and then migrate to more expensive microcontrollers with 4, 16, 32 or 64 kB of EEPROM as needed for his multiapplication cards. To the extent that the microcontroller manufacturer supports such a range of chips, the smart card operating system must also have matching capabilities. This means that it must be able to automatically recognize the size of the EEPROM, and consequently set up its internal pointer structures for maximum free memory, file sizes and similar operational parameters. The technical implementation of this involves using an operating system routine to read the manufacturer's fabrication data and calculate the size of the available EEPROM from this data. This technique can only handle variable EEPROM sizes; current smart card operating systems cannot adapt themselves to various sizes of ROM or RAM.

The main advantage of this hardware recognition capability is that the producer of the smart card operating system does not have to match the program code to the size of the EEPROM. This eliminates a possible source of errors, and above all it means that the operating system does not have to be re-evaluated for every new hardware platform. The hardware recognition capability of modern operating systems saves considerable time in software development and can thus reduce product development time by several weeks.

Soft masks and hard masks

The terms 'soft mask' and 'hard mask' are often used in connection with field trials and smart card operating systems. Strictly speaking, both terms are nonsensical from a purely logical point of view, since a ROM mask – which means the program code located in the ROM – is always unalterable and thus 'hard'. In the jargon of the smart card world, however, the term 'soft mask' means something only roughly similar to a ROM mask. This term is used when part or all of the program code for a smart card operating system, or the commands for an application, is located in the EEPROM. This means that the code can be altered relatively easily, without the cost and time involved in generating a new ROM mask. This type of mask is thus alterable, or 'soft'. Soft masks are primarily used in testing and field trials, since they allow errors to be corrected and programs to be modified quickly and at minimal cost. The disadvantage of using a soft mask is that it requires using chips with large EEPROMs, which are more expensive than equivalent chips with program code in ROM. However, since field trials normally do not involve issuing millions of cards, the increased cost of using chips with large EEPROMs is entirely acceptable.

Once the test or field trial using a soft mask has been completed and the program code in the EEPROM is operational without any further modifications, it can be moved from the EEPROM to the ROM by generating a true ROM mask. This can be done with relatively little effort, and the result is called a 'hard mask', since it cannot be readily altered. Strictly speaking, the only advantage of a hard mask is that a given amount of memory occupies significantly less chip area in ROM than in EEPROM, thus allowing a smaller and less expensive microcontroller to be used for the same amount of program code.

This two-stage process using soft and hard masks for new smart card applications also provides flexibility and allows substantial changes to the program code to be made even shortly before the cards are issued to users. With a traditional process using a pure ROM mask, it is not possible to make significant changes to the program code once the mask has been given to the manufacturer. Since the two-stage process is superior to the traditional process in this regard, nowadays a soft mask is almost always used initially when introducing a new smart card application. Migration to a hard mask occurs only after any necessary modifications have been made to the program code.

Operating system APIs

Originally, smart card operating systems did not allow third-party software to be loaded into the card and executed as needed. Consequently, at that time operating systems did not have published programming interfaces that could be used by third parties to call operating system functions. More recent developments in smart card operating systems, such as MULTOS and operating systems that support Java (Java Card), allow third parties to load their own program code into smart cards. In order to eliminate the need to again program routines already present in the operating system, such operating systems include carefully conceived application programming interfaces (APIs) that provide access to the most important functions of the operating system. Naturally, practically all operating systems have their own internal APIs, but these APIs are not designed for external use and are usually confidential.

As an exception to the usual situation in the smart card world, there are no general standards relating to APIs for smart card operating systems. Instead, two industry standards have come into predominant use. One of them consists of the various Java Card APIs, while the other is the Multos API. The APIs described in the related specifications provide access to the essential functions of the operating system.[2] These functions include access to the file manager, calls to the available cryptographic functions and, naturally, transmitting and receiving data. The only essential difference between any of the well-known PC operating systems and a smart card operating system with a complete operating system API and provision for downloaded program code is the amount of memory available to the operating system.

5.5 MEMORY ORGANIZATION

The three different types of memory used in smart card microcontrollers have completely different properties. ROM can only be programmed using a mask during manufacturing, and it is all programmed at the same time. Its content is static for the life of the chip. Given the construction of ROM, the chance of an undesired change in the content of the ROM is practically zero.

In contrast to ROM, RAM retains its content only as long as power is applied to the smart card. A power failure causes total loss of all data stored in the RAM. However, data can be written to RAM at the full operating speed of the processor, and RAM can be erased an unlimited number of times. EEPROM, by contrast, can retain data without external power. However, it has three disadvantages, which are its limited lifetime, the fact that it is divided into pages and its relatively long write and erase times (around 1 ms/byte). With the exception of the interrupt vectors, which are prescribed by the microprocessor, the operating system program code stored in ROM does not have to comply with any particular structure. The individual routines can thus be linked to each other in any desired sequence, although an attempt is made to limit the length

[2] See also Section 5.14.1, ‘Java Card’, and Section 5.14.2, ‘Multos’

of jumps in order to save memory space. What is important is that the ROM is protected by error detection codes (EDCs), since it is certainly possible for errors to occasionally occur in the ROM. A scratch in the ROM region of the microcontroller, for example, or a fracture during the wire bonding process, can cause the data in the ROM to be incorrect. Interestingly enough, this does not necessarily mean that the operating system can no longer function; instead, it is certainly possible that only specific routines will run incorrectly. In order to prevent problems that might arise in such situations, the ROM is checked when the operating system starts up to verify that it is fundamentally free of errors.

Figure 5.8 shows the usual memory organization of a 256-byte RAM. It is divided into regions for the registers, the stack, general variables, workspace for cryptographic algorithms and the I/O buffer. If an I/O buffer of 256 bytes is required, for example, or if additional variables must be stored in RAM, the limits of the available memory can be reached very quickly. This problem is solved by having workspace in the EEPROM, which is thus used like RAM. The disadvantage of this is that it takes around 10,000 times as long to write data to EEPROM as to write data to RAM. An additional disadvantage is the limited lifetime of EEPROM cells, since unlike RAM cells, they cannot be written an unlimited number of times. However, moving the content of the RAM to the EEPROM is often the only solution, such as when an I/O buffer is needed that is larger than the entire amount of available RAM. For comparison, Figure 5.8 also shows the typical organization of a 6-kB RAM in a high-performance telecommunications smart card.

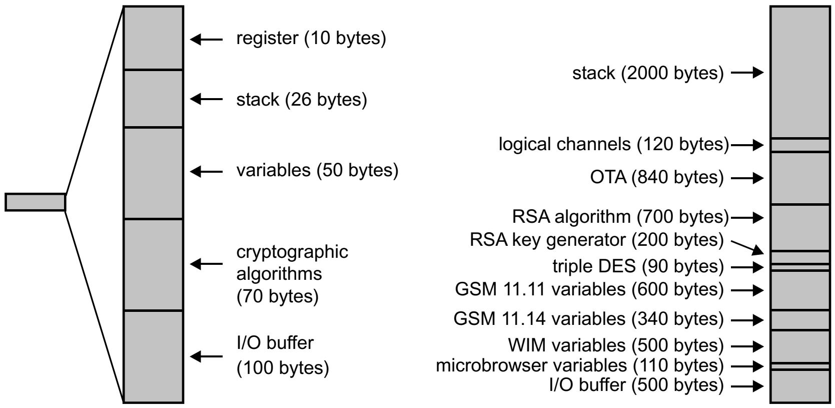

register (10 bytes)
stack (26 bytes)
variables (50 bytes)
cryptographic algorithms (70 bytes)
I/O buffer (100 bytes)
stack (2000 bytes)
logical channels (120 bytes)
OTA (840 bytes)
RSA algorithm (700 bytes)
RSA key generator (200 bytes)
triple DES (90 bytes)
GSM 11.11 variables (600 bytes)
GSM 11.14 variables (340 bytes)
WIM variables (500 bytes)
microbrowser variables (110 bytes)
I/O buffer (500 bytes)

Figure 5.8 The diagram on the left shows a typical partitioning of a 256-byte RAM for a simple smart card operating system programmed in assembler. The diagram on the right shows the memory organization of a 6-kB RAM for a high-performance telecommunications smart card whose operating system has been generated using the C language

The organization of the data stored in EEPROM is much more complicated and intricate than either of the other two types of memory. With modern operating systems, the basic partitioning of the EEPROM is essentially as follows (see Figure 5.9). First, many types of microcontrollers have a region at the beginning of the EEPROM with special hardware protection that can be used to store special manufacturing data, such as a number that is only used once and is thus unique to the chip. Many semiconductor manufacturers also record the chip type and the amount of EEPROM available to the operating system in this region. This region is usually designed for write once, read multiple (WORM) access, which means that it can be written only once, after which it can only be read. Technically, this is usually achieved by using regular EEPROM cells that have been modified so that they cannot be electrically erased.

Table 5.5 Examples of manufacturing data written by some semiconductor manufacturers to a WORM region in the EEPROM during chip fabrication. The first five manufacturing data elements, taken together, yield a unique 8-bit chip number. The major advantage of a chip number generated in this manner is that it does not require highly accurate time data synchronized over several production locations, but only data available to all production machines in every manufacturing facility

Data element	Size
Semiconductor manufacturer	1 byte
Production facility	1 byte
Semiconductor processing batch number	2 bytes
X coordinate on the wafer	2 bytes
Y coordinate on the wafer	2 bytes
Microcontroller type	2 bytes
Optional hardware components	6 bytes
RAM capacity	2 bytes
EEPROM capacity	3 bytes
ROM capacity	3 bytes

Above this region, which usually has a size of 16 to 32 bytes, come the tables and pointers for the operating system, which are loaded into the EEPROM when the card is completed. The combination of these tables and pointers and the routines stored in ROM yields the complete smart card operating system. In order to ensure that the operating system can always be used in a secure and stable state, this region is protected by an error detection code (EDC) that is re-computed and checked before the first EEPROM access or even before every EEPROM access. If a memory error is detected during EDC verification, the affected portion of the EEPROM must subsequently not be used, since this means that proper operation of the operating system cannot be assured.

Above the protected portion of the operating system, there is a region that contains additional application program code. If need be, this region may also be protected against alteration by using a checksum. Application-specific commands or algorithms that should not be located in the ROM or that are too large to fit in the ROM are placed in this region.

The next region contains all of the file structures, or in other words, the entire externally visible file tree. This region is not protected as a whole by a checksum, but instead usually has strong file-based protection. The internal structure of this region is shown in more detail in Figure 5.9.

At the top of the EEPROM, there is an optional free memory region that has its own memory manager. However, the free memory is often assigned to individual applications in the file region, where it can be used within the applications for the creation of new files.

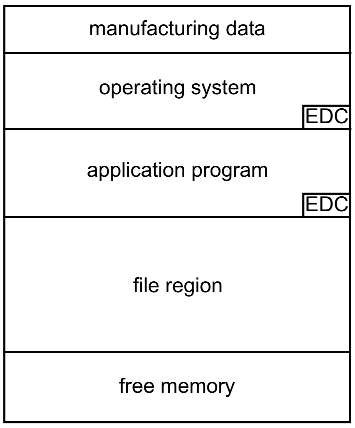


Figure 5.9 An example of EEPROM partitioning for a smart card operating system

Otherwise it belongs to the general file region and is available for all new applications that are loaded in their entirety.

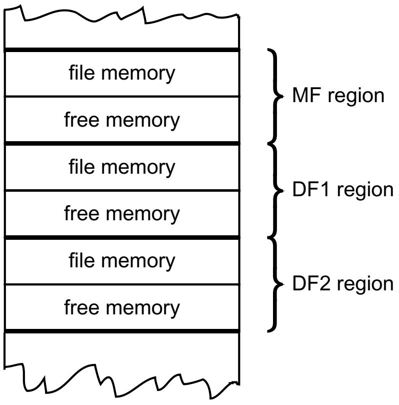


Figure 5.10 Sample partitioning of the file region of a smart card operating system that supports several independent and physically separated applications. Although this approach yields extremely rigid isolation of the individual applications, it is presently no longer used in operating systems with dynamic file management, since it is too inflexible with respect to free memory management

5.6 SMART CARD FILES

In addition to containing mechanisms for identification and authentication, smart cards are primarily data storage media. They have a decisive advantage relative to other storage media, such as diskettes, in that access to the data can be tied to certain conditions.

The first smart cards had only more or less directly addressable memory regions, which could be used for writing or reading data. The data were accessed by specifying physical memory addresses. Nowadays, all smart cards have complete hierarchical file management systems with symbolic, hardware-independent addressing.

Naturally, these file management systems have certain features that are specific to smart cards. The most obvious feature is that there is no man–machine interface. All files are addressed using hexadecimal codes, and all commands are strictly based on this addressing, since here communication only takes place between two computers. Equally typical of these file management systems is that they are designed to use a small amount of memory. Every redundant byte is avoided if possible. Since the ‘user’ in the terminal is a computer, this does not present any problems.

There is usually no form of sophisticated memory management, which also helps to keep memory usage as low as possible. If a file is deleted – and only a few operating systems have this capability – the space released does not necessarily become available for use by a newly created file. Normally, all files are created and loaded into the smart card when it is initialized or personalized. After this, changes to file contents are limited.

Naturally, the characteristics of the memory that is used also affect the nature of the file management system. The memory pages in an EEPROM cannot be written or erased an unlimited number of times, as can the hard disk of a PC. Consequently, there are special file attributes that allow information to be stored redundantly so it can be corrected if necessary.

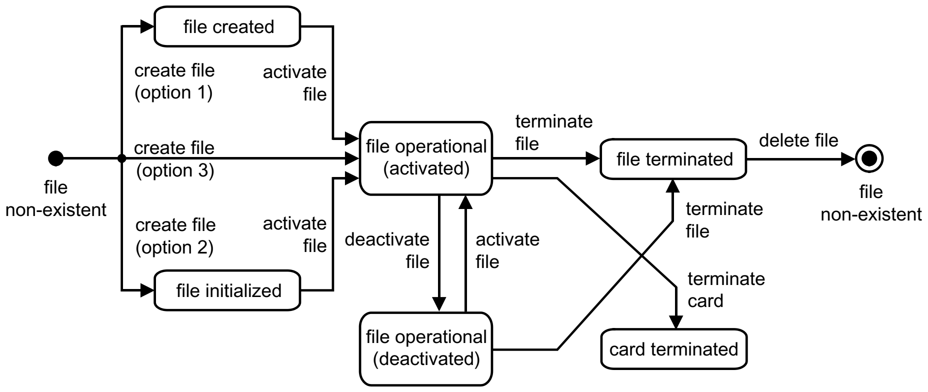


Figure 5.11 States and state transitions in the full life cycle of files in a smart card operating system compliant with ISO/IEC 7816-9. The various states following the creation of a file result from the possible parameters

Internal structures of files

Modern file management systems for smart cards have an object-oriented structure. This means that all information about a file is stored in the file itself. A further consequence of this principle is that a file must always be selected before any action can be performed. Files in such object-oriented systems are thus always divided into two parts. The first part, called the file header, contains information about the layout and structure of the file and its access conditions. The modifiable user data are stored in the second part, the file body, which is linked to the file header by a pointer.

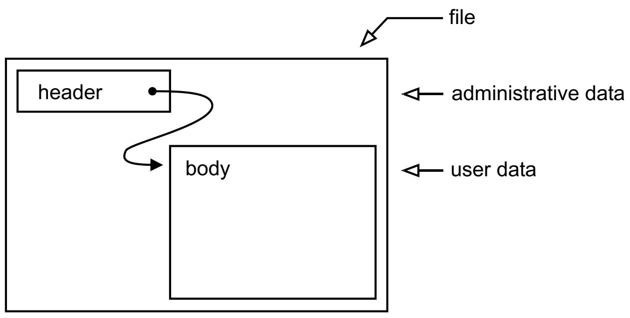


Figure 5.12 The internal structure of a file for a smart card file management system

In addition to providing improved data structuring, this scheme also has the advantage of providing better physical security for data items. The page-oriented EEPROM containing all the files allows only a limited number of write/erase cycles. The file header and the file body are always located on separate memory pages. The header, which is normally rarely altered, stores all of the access conditions. A write or erase error involving the file body thus cannot affect these conditions. If the header and the file body were located in the same page of memory, it would be possible to utilize deliberately generated write errors to alter the access conditions such that confidential information could be read from the file body.

Some smart card operating systems offer the option of addressing a file body from two different headers. These two headers are usually located in DFs belonging to two different applications. This allows data to be shared between two applications in a technically elegant manner. In this case, it is important for the access conditions specified in the two headers to be identical.

5.6.1 File types

The structure of a smart card file system, as specified in the ISO/IEC 7816-4 standard, is similar to that of a DOS or Unix system. The major difference is that smart cards do not contain any application-specific files, such as special types of files for a particular word processor. Only the standard file structures may be used in smart cards.

There are basically two categories of files for file smart cards. The first category is directory files, which are called ‘dedicated files’ (DFs). The second category consists of the files that hold the actual user data, which are called ‘elementary files’ (EFs). A DF acts as a sort of folder containing other, lower level DFs or EFs that logically belong together. EFs can be classified into those for the external world (working EFs) and those for the operating system (internal EFs). The various file types are described below, and their relationships are illustrated in Figure 5.14.

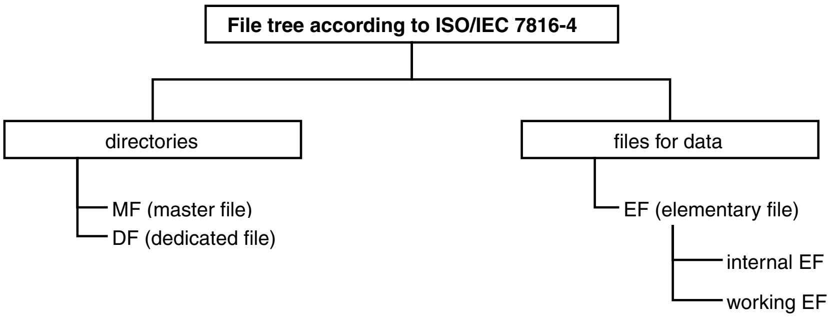


Figure 5.13 Classification of smart card file structures according to ISO/IEC 7816-4

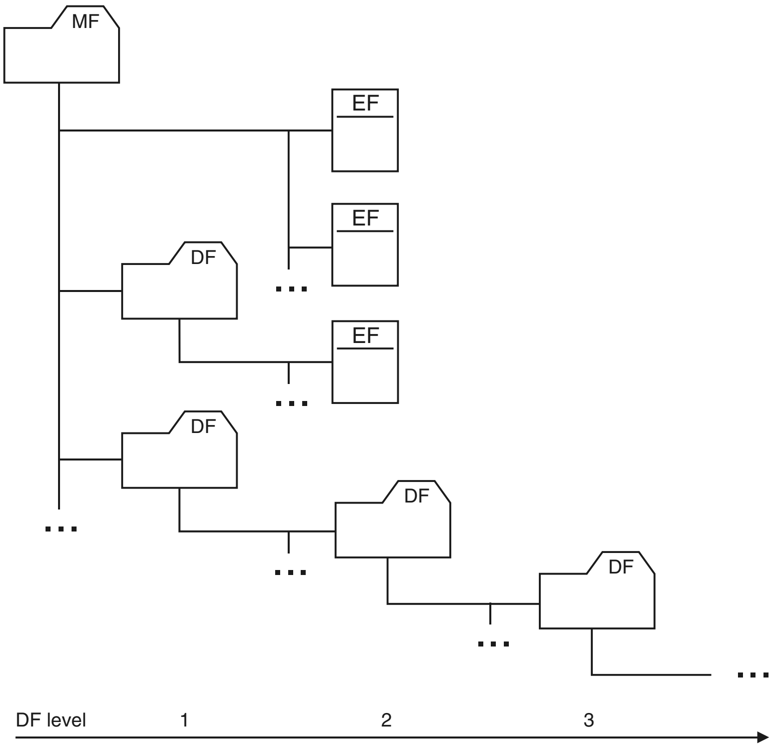


Figure 5.14 The various types of files in a smart card file tree

MF

The root directory is called the ‘master file’ (MF). It is implicitly selected after the smart card is reset. The MF contains all other directories and all files. It is a special type of DF, and it represents the entire extent of the smart card memory available for the file region. A master file must be present in every smart card.

DF

Dedicated files (DFs) may exist below the MF as needed. The term ‘directory files’ is frequently used for these files, although this does not comply with the official definition of the abbreviation ‘DF’ in the ISO/IEC 7816-4 standard. A DF is a directory in which other files (DFs and EFs) can be grouped together. A DF may contain other DFs. In principle, there is no limit on the number of levels of DFs. However, it is rare for there to be more than two levels of DFs under the MF, due to the limited amount of memory in smart cards.

With the specification for the UICC (TS 102.221), a special type of DF was introduced with the name ‘application dedicated file’ (ADF). This is a DF for applications, and it can be selected using an appropriate mechanism (a SELECT command with an AID), but it is not located below the MF. An ADF can thus be considered to be a type of MF.

EF

The user data needed for an application are located in EFs. ‘EF’ is the abbreviation of ‘elementary file’. EFs may be located directly below the MF or below a DF. To allow data to be stored in a minimum amount of memory and logically optimized data structures, an EF always has an internal file structure. This is the main difference between EFs and files in a PC, whose internal data structures are determined by applications (such as word processors) rather than by the operating system. EFs are classified into working EFs and internal EFs.

Working EFs

All application data that must be read by or written from the terminal, or in other words, all data that are intended for the external world (as seen by the smart card), are located in working EFs. The data contained in such files are not used by the operating system.

Internal EFs

In addition to EFs for applications, there are also internal system files that store data for the operating system itself, data for the execution of an application, secret keys and program code. Access to these data is specially protected by the operating system. According to ISO/IEC 7816-4, these system files can be integrated into the file management system in two different ways. The first option is to store these files in the relevant application DFs as invisible EFs. Such files cannot be selected, and they are managed fully transparently by the smart card operating system in a manner similar to how resource files are used in the Macintosh OS. With the second option, these system files are assigned regular file names (in other words, FIDs) and can be selected using these names. This is essentially the same principle as is used for file management under DOS. Each of these two approaches has an equal number of advantages and disadvantages, with both providing the same functionality in somewhat different manners.

Application files

According to convention, all files containing user data for a particular application (the EFs for that application) are always grouped together in a single DF. This produces a clear and easily understood structure, and it makes it easy to enter a new application into a smart card by creating the appropriate DF.

Since the MF is a special sort of DF, it goes without saying that in a single-application smart card, all the application files can be placed directly under the MF. In a typical single-application smart card, therefore, all the EFs can be located either directly under the MF or in a solitary DF. Smart cards with several applications have a corresponding number of DFs, in which the EFs belonging to the applications are located.

Additional DFs can be placed within such application DFs. For example, a DF placed directly under the MF could be dedicated to a 'Traffic Control' application. An additional level of DFs within the application DF could contain the files for the languages supported, such as 'English' and 'Deutsch'.

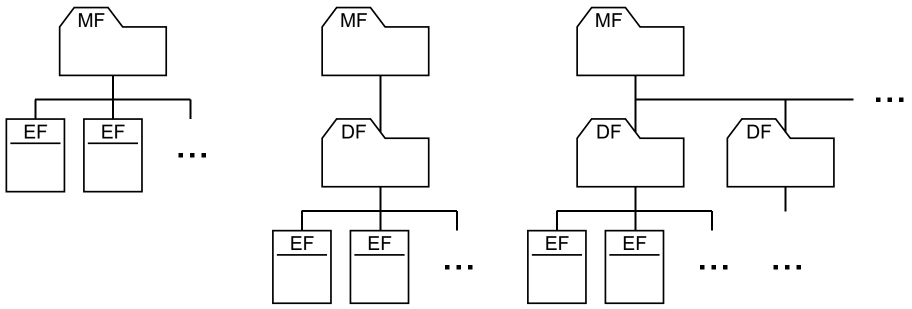


Figure 5.15 Differences between the file structure of a smart card with only one application and the file structure of a smart card with several applications. The two reasonable arrangements for cards with only one application are shown on the left and in the middle. The arrangement for a smart card with several applications is shown on the right

5.6.2 File names

In modern smart card operating systems, files are without exception addressed by logical names, rather than being accessed using direct physical addresses. The latter approach was perfectly normal in the smart card world earlier on, and there are a few smart card microcontrollers that still use it. With simple applications that occupy precisely defined memory regions, direct physical access can save a lot of memory space. It also does not result in any loss of user-friendliness, since all files are accessed by the computer in the terminal. However, direct physical access does not in any way match the criteria of modern software design, and it also creates major problems with regard to software enhancements and smart card microcontrollers with differing address spaces.

All schemes that use logical file names are significantly better, and above all much easier to extend. Without question, it can be assumed that in a few years file addressing using logical

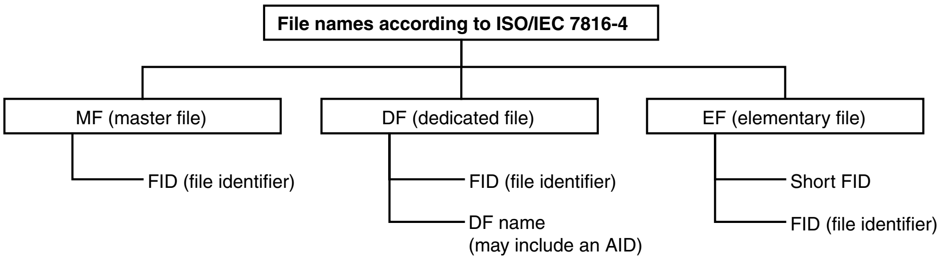


Figure 5.16 Classification chart of smart card operating system file names per ISO/IEC 7816-4

names will be the only type of file addressing used in smart cards with microcontrollers. With memory cards, on the other hand, physical file addressing will continue to be used for the foreseeable future.

File identifier (FID)

The system described here is based on the ISO/IEC 7816-4 standard, and it is in principle reflected in all other international smart card standards. Every file, including directory files, has a 2-byte file identifier (FID), which can be used to select the file.

For historical reasons, the FID of the MF is '3F00'. This FID is reserved for the MF within the entire logical address space. The logical file name 'FFFF' is reserved for future applications, and may not be used. There are also other FIDs that are reserved by the ISO standard and by other standards. They are listed in Table 5.6.

Table 5.6 FIDs reserved by the most important smart card standards

FID	Name and purpose	Standard
'2F00'	This FID is reserved for EF_{DIR} (directory) file, which is used to store application identifiers (AIDs) and the path names of the associated applications.	ISO/IEC 7816-4
'2F01'	This FID is reserved for EF_{ATR}, which contains extensions to the ATR.	ISO/IEC 7816-4
'3F00'	MF is the root directory for all files in a smart card.	ISO/IEC 7816-4, GSM 11.11, TS 102.221, EMV
'3FFF'	This FID is reserved for file selection using a path name.	ISO/IEC 7816-4
'FFFF'	This FID is reserved for future use by ISO/IEC.	ISO/IEC 7816-4

The GSM application is a typical example of the fact that various groups of FIDs cannot be freely used. In the GSM 11.11 specification, the more significant byte is determined by the location of the file within the directory structure (file tree). This coding has developed historically and originates from the first French smart cards. GSM DFs have a value of '7F' for the first (more significant) byte. The FIDs of EFs located directly below the MF have '2F' as the first byte, and the FIDs of EFs located underneath a DF start with '6F'. The less significant bytes are numbered sequentially. This specification applies only to the GSM application and is not a general standard. In other situations, the full 2-byte address region of the FID can be fully exploited and is not subject to any restrictions.

EF_{DIR} has an FID of '2F00'. It has a linear fixed structure and consists of at least one record. Each record is in turn a constructed data object containing data about a particular application in the smart card. This typically consists of the AID and a textual designation of the application in question. EF_{DIR} can also contain additional data, such as the path to the application. The purpose of EF_{DIR} is to display the applications present in a smart card to a terminal in a standardized format.

Table 5.7 Example of a typical structure of an EF_{DIR} file per ISO/IEC 7816-4 and ISO/IEC 7816-5, as specified for the UICC

EF_{DIR}	Directory EF	
Description:	This file contains information about the applications present in a smart card.	
File:	FID = '2F00'; structure: linear fixed, file size: n bytes; access: READ: always, UPDATE: application-dependent, but generally limited to the administrator	
Record coding:	byte 1:	'61' ('application template' tag)
	byte 2:	length of the application template (3–127)
	byte 3:	'4F' (AID tag)
	byte 4:	length of the AID (1–16)
	byte 5– n:	AID
	byte n + 1:	'50' ('application label' tag)
	byte n + 2:	length of the application label
	byte n + 3– m:	application label in ASCII (1–16)
Example:	'61 0F 4F 05 D2 76 00 00 60 50 05 52 61 6E 6B 6C'	
	'61'	⇒ 'application template' tag
	'0F'	⇒ length of the application template = 15 bytes
	'4F'	⇒ AID tag
	'05'	⇒ length of the AID = 5 bytes
	'D2 76 00 00 60'	⇒ AID
	'4F'	⇒ 'application label' tag
	'05'	⇒ length of the application label = 5 bytes
	'52 61 6E 6B 6C'	⇒ application label = "Rankl"

The FIDs in the file tree must be chosen such that the files can be unambiguously selected. It is thus prohibited for two different files within the same DF to have the same FID. A DF may also not have the same FID as an EF located directly underneath it, since this would mean that the operating system would have to decide whether to select the DF first or the EF. The following rules apply to the selection of unique FIDs:

Rule 1: all DFs and EFs within a single directory must have different FIDs.
Rule 2: nested directories (DFs) may not have the same FIDs.
Rule 3: an EF within a directory (MF or DF) may not have the same FID as the next higher or next lower directory.

Short file identifier (SFI)

Short file identifiers may be used for implicit file selection in the immediate context of a command. Short file identifiers are optional for EFs, so they do not necessarily have to be assigned. An SFI is passed as a command parameter for implicit selection of a file and is therefore only 5 bits long. It can thus take on values between 1 and 30, since a short file identifier of '0' addresses the current EF.

DF name

A DF consists of a collection of files used by an individual application. A DF is a sort of directory or folder, and it can contain both EFs and other DFs. In the future, the address space provided by the 2-byte FID could become too small. Consequently, each DF has a ‘DF name’ in addition to its FID. As specified in the ISO/IEC 7816-4 standard, the DF name has a length of 1 to 16 bytes. The DF name provides sufficient address space to allow every smart card application to be unambiguously identified throughout the world. Since DF names are freely chosen, it is possible for two different DFs to sometimes have the same DF name. Consequently, DF names are normally only used together with AIDs (application identifiers), as defined in the ISO/IEC 7816-5 standard. An AID may have a length between 5 and 16 bytes and is composed of two data elements defined by ISO. The AID is thus a subset of the DF name.

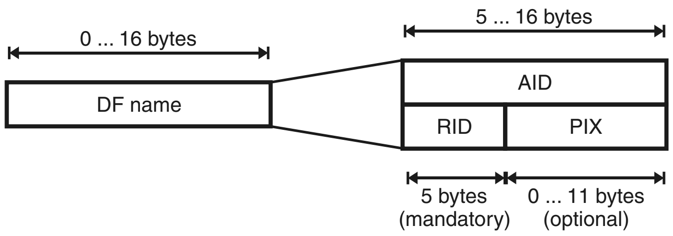


Figure 5.17 The DF name in relationship to the AID, which consists of the RID (registered identifier) and the PIX (proprietary application identifier extension)

Structure and coding of the application identifier (AID)

The application identifier (AID) consists of two data elements. The first data element is the registered identifier (RID), which has a fixed length of 5 bytes. It is assigned by a national or international registration authority and includes a country code, an application category and a number that refers to the application provider. This numerical coding means that each RID is assigned only once, so that it can be used worldwide to identify a particular application. Unfortunately, the lists of assigned RIDs are confidential, so they cannot be published. However, some examples of RIDs that have been made public can be found in Section 16.8, ‘Selected RIDs’. Addresses of national and international RID registration authorities are located in Section 16.7, ‘Registration Authorities for RIDs’.

If necessary, an application provider can place a proprietary application identifier extension (PIX) after the RID. The PIX, which may be up to 11 bytes long, is the optional second part of the AID. It may consist of a serial number and a version number, for example, which could be used for naming the application.

Table 5.8 Coding of the 5-byte (10-digit) registered identifier (RID)

RID			Meaning
D1	D2–D4	D5–D10	
X	—	—	Registration category: 'A'–international registration 'D'–national registration
—	X	—	Country code; coding per ISO 3166
—	—	X	Number of the application provider, which is assigned by a national or international registration authority

Table 5.9 Example of a nationally registered RID that complies with the ISO/IEC 7816-5 standard (in this case, the RID of Wolfgang Rankl)

RID			Meaning
D1	D2–D4	D5–D10	
‘D’	...	—	The registration category is ‘national’
...	'276'	—	The ISO 3166 country code for Germany
...	...	'00 00 60'	Application provider number assigned by the national registration authority

5.6.3 File selection

Object-oriented file management systems require that a file be selected before it can be accessed. File selection informs the operating system which file will subsequently be addressed. Successful selection of a new file causes the previous selection to become invalid. This means that only one file can be selected at any given time. Since FIDs may be freely chosen, certain limitations must be imposed on the free addressability of files. Otherwise, it could easily happen that several files with the same FID would be available in the file tree, and the operating system would then have to decide which file was meant. In order to avoid such ambiguity and thus be independent of the search algorithm used by the operating system file manager, the selection options for files are intentionally restricted.

Things would be different if all the FIDs used in the file tree were unique. In this case, it would be easy to select the desired file across several directory boundaries. However, this situation is exactly what cannot always be guaranteed. Consequently, selection is only possible within certain boundaries, since otherwise unambiguous selection of the desired file cannot be assured. The MF, however, can always be selected from anywhere within the file tree, since its FID is unique within the file tree. Selecting a DF located in the first level below the MF is only possible from a DF at the same level or from the MF. Figure 5.18 shows examples of various types of allowed and prohibited selections.

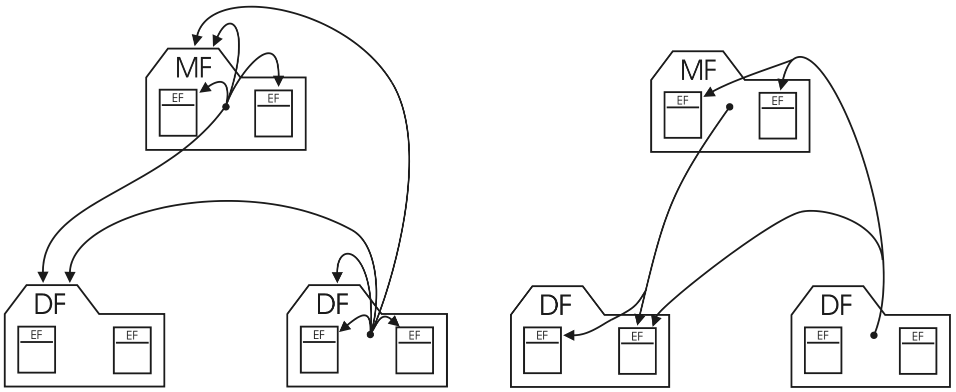


Figure 5.18 Examples of allowed selection options (left) and prohibited selection options (right) when a FID or DF name is used. Only direct selection without the path name is illustrated

Selecting directories (MF and DF)

The MF can be selected from anywhere within the file tree, either using a special selection option of the file selection command or by means of its FID ('3F00'), which only occurs once within the file tree. When the MF is selected, the selection state that exists immediately after the smart card is reset is restored, since the MF is implicitly selected by the operating system after a reset. DFs can be selected either via their FIDs or via their DF names, which contain registered and thus unique AIDs.

Explicit EF selection

There are basically two methods available for selecting EFs. With explicit selection, a specific command (SELECT FILE) is sent to the smart card before the actual access to the file takes place. This command includes a parameter holding the 2-byte FID of the file to be selected. After the file has been selected, it can be accessed by all subsequent commands.

Implicit EF selection

Implicit selection is the name given to the process in which a file is selected using a short file identifier passed as a parameter of a command that actually accesses the file. A number of restrictions apply to the use of implicit EF selection. It only works for EFs within the currently selected DF or MF. It is thus not possible to implicitly select a file across directory boundaries.

In addition, implicit selection is possible only with certain access commands that allow a short file identifier to be passed as a parameter (such as READ BINARY, UPDATE BINARY, READ RECORD and UPATE RECORD).

The major advantage of implicit selection is that it allows a file to be selected and accessed with a single command. This makes a SELECT FILE command unnecessary in many cases, which simplifies the sequence of commands. Due to the reduced need for communications, using implicit selection allows distinctly higher processing speeds to be achieved.

File selection using a path name

In addition to direct selection, the ISO/IEC standard allows two supplementary methods for explicit file selection using a path name. In the first method, the path from the currently selected file to the target file is passed to the operating system. The second method uses the path from the MF to the target file. Both methods are implemented in many smart card operating systems. Using these additional capabilities results in a measurable reduction in the time required to process command sequences.

5.6.4 EF file structures

In contrast to files in Windows systems, EFs in smart cards have internal structures. The structure can be individually selected for each EF according to the intended purpose of the file. This has major advantages for the outside world, since the internal structure allows data elements to be constructed such that they can be accessed very quickly and effectively.

Managing these data structures requires a significant amount of program code in the smart card. This is why the data structures are not all mutually symmetrical, but instead occur only in the forms often needed in practice.

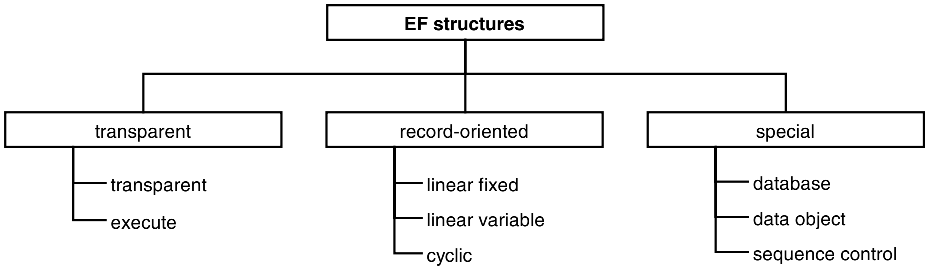


Figure 5.19 Classification of EF file structures for smart card operating systems

Transparent file structure

The transparent data structure is often referred to as a binary or amorphous structure; in other words, a transparent file has no internal structure. The data contained in the file can be accessed for reading or writing in bytes or blocks using an offset value. The READ BINARY, WRITE BINARY and UPDATE BINARY commands are used for this purpose.

The minimum size of a file with a transparent structure is one byte. No maximum size is explicitly specified in any standard. However, the maximum number of bytes that can be read in the short format (255) or the long format (65,536), combined with the maximum offset value (32,767), allows a maximum size of 65,791 bytes or 98,303 bytes, respectively. With the memory capacities of contemporary smart cards, these maximum sizes are slowly slipping into the realm of what is feasible, although in current practice transparent files are rarely larger than a few hundred bytes. Figure 5.20 illustrates the organization of the transparent file structure, and Figure 5.21 illustrates how six bytes can be accessed for reading from a 12-byte long file using an offset of 4 bytes.

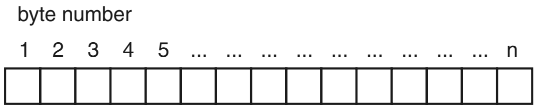


Figure 5.20 Transparent file structure

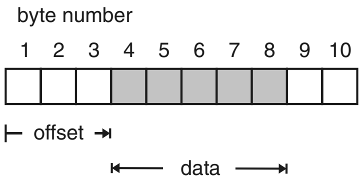


Figure 5.21 Reading 6 bytes from a transparent file using an offset of 4 bytes

A transparent file structure is primarily used for very small amounts of data or for data having no internal structure. An example of a typical use would be storing a digitized passport photograph that could be read from the smart card by the terminal. However, this linear, one-dimensional data structure can also be used to simulate other data structures if necessary. Of course, in such cases the terminal access takes a more complex form, since parameters related to the structure of the file must be stored inside the file itself.

Linear fixed file structure

The linear fixed data structure is based on chaining fixed-length records. A record consists of a series of individual bytes. Individual records within this data structure can be freely accessed. The smallest unit of access is one record, which means that it is not possible to access only part of a record. The commands READ RECORD, WRITE RECORD and UPDATE RECORD can be used for reading and writing within this data structure.

The first record is always numbered record 1. The largest allowed record number is 'FE', or 254 in decimal notation, since 'FF' is reserved for future extensions. The length of a single record is determined by the access commands, and can range from 1 to 254 bytes, but all records in the file must have the same length. Figure 5.22 illustrates the organization of the linear fixed file structure.

A typical application for this data structure is a telephone directory, in which the name comes first, followed by the associated telephone number in a fixed position.

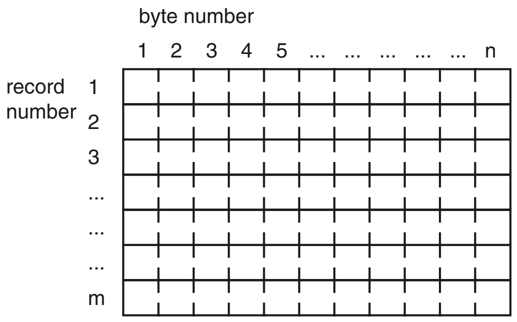


Figure 5.22 Linear fixed file structure

Linear variable file structure

The fact that all records in a linear fixed file structure have the same length means that memory space is often wasted when this structure is used, since many record-oriented data items have variable lengths. One example is the names in a telephone directory. The challenge of minimizing the amount of memory space is met by the linear variable structure, in which each record can have an individually defined length. The unavoidable consequence of this is that each record must have a supplementary field that contains information about its length. This structure is otherwise similar to the linear fixed structure, as can be seen from Figure 5.23.

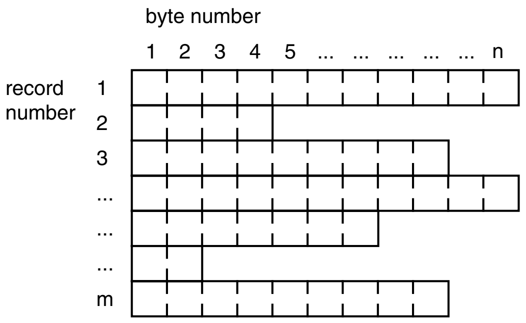


Figure 5.23 Linear variable file structure

The first record is numbered 1, and the maximum file length is 254 records. The length of an individual record is determined by the access commands, and can range from 1 to 254 bytes. The access commands for this structure are the same as for the linear fixed structure, namely READ RECORD, WRITE RECORD and UPDATE RECORD.

This file structure is preferably used when records with highly variable lengths are to be stored and the use of smart card memory must be minimized. For example, the previously mentioned telephone directory could be optimized by making each record exactly as long as the actual entry, so the records would not all have the same length. However, managing this file structure requires program code in the smart card operating system and extra memory space to store the record length information. For this reason, operating systems for microcontrollers with small memories often do not offer this file structure. Consequently, the ISO/IEC 7816-4 standard explicitly allows this limitation in some profiles.

Cyclic file structure

The cyclic structure is based on the linear fixed file structure, and thus consists of a certain number of records that all have the same length. In addition, the EF contains a pointer that always indicates the record that was last written. This record is always numbered record 1. If the pointer reaches the last record in the EF, the operating system automatically adjusts it to point to the first record in the EF when the next write access occurs. It thus behaves the same as the hour hand of an analog clock.

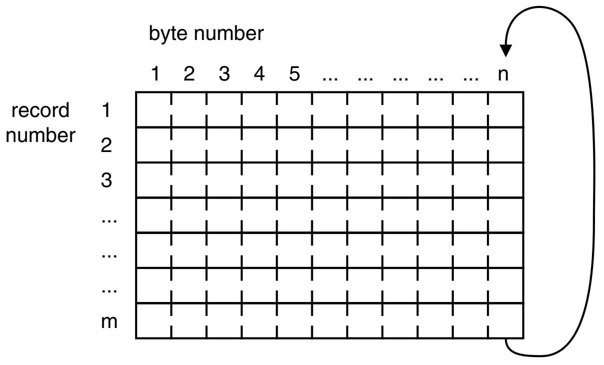


Figure 5.24 Cyclic file structures

If a cyclic file contains n records, the last one that was written is record number 1. The one that was written just before that is number 2, and the oldest record is number n. This file structure, like the other two record-oriented file structures, can also be accessed by addressing the first, last, next or previous record.[3]

The number and size of the records are fully analogous to those for the linear fixed structure. Due to the limitations of the write and read commands, a maximum of 254 records can be created, each with a maximum length of 254 bytes. This structure is typically used for log files within the smart card, in which the oldest entry is always overwritten by a new entry.

Execute file structure

The 'execute' structure is in principle not a separate structure, since it is based on the transparent structure. It is described in the EN 7826-3 standard and offers numerous extension options

[3] See also Section 7.2, 'Write and Read Commands'

within the operating system. The execute structure is not intended to be used for storing data, but rather for storing executable program code.[4] A file with execute structure can be accessed using the same commands as for transparent files. Of course, this structure creates a sort of ‘back door’, since anyone who can write to such a file can download his or her own program code into the card, possibly including a Trojan horse.

The maximum program size is the same as the maximum size of a transparent file, and is thus 65,791 bytes without offset addressing or 98,303 bytes with offset addressing, using the UPDATE BINARY command. No internal data structure is specified, but it is certainly possible for the program code contained in the file to introduce one, so that the executable program can define its own data regions in the execute file and access them internally.

Database file structure

The ISO/IEC 7816-7 standard defines a subset of SQL for smart cards with the designation SCQL (smart card query language). In order to store data in the file system of a smart card such that they can be read using SCQL commands, it is necessary to provide a suitable file structure. The layout of this structure is not standardized; instead, its design is left up to individual operating system producers. A database file stores the actual user data, various ‘views’ of the database, the access privileges and user profiles.

Data object file structure

The ISO 7816-4 commands PUT DATA and GET DATA are used to store TLV-coded data objects in smart cards and to read out such stored objects. This can be implemented either completely independent of the file management system, or within the file management system using a special structure for storing data objects. If the implementation is within the file management system, a slightly modified transparent or linear fixed file can be used as a storage location for data objects. In this case, the PUT DATA and GET DATA commands access these modified file structures via the file management system.

Sequence control file structure

If a smart card operating system has a command sequence controller, information regarding the commands that can be accepted must also be stored in memory. This is normally done using a file whose structure is specially adapted to this task. However, this is not standardized, so every operating system with sequence control has its own format, which is not compatible with that of any other operating system.

5.6.5 File access conditions

As part of their object-oriented design, all files contain information governing access to them within the context of the file management system. This information is always physically

[4] See also Section 5.12, ‘Downloadable Program Code’

coded in the header of each file. The entire security of smart card file management is based on managing file access privileges, since these privileges form the basis for controlling file access.

The access conditions are defined when a file is created, and they usually cannot be modified afterwards. There is a high degree of variation in the permitted file access conditions, depending on the commands present in the operating system. For example, there is no point in defining access conditions for a READ RECORD command if this command is not present in the smart card operating system.

For the MF and the DFs (in contrast to the EFs), there is no information stored with respect to data access (read or write privileges). Instead, the access conditions for creating new files are stored together with other information. Depending on the file type, other access conditions may also be stored. For EFs, these conditions relate to accesses to the file contents, and for the MF and DFs, they are the conditions that apply within these organizational structures.

In specifying access conditions, a distinction can be made between state-oriented and command-oriented access conditions. With state-oriented conditions, the current security state is compared with the corresponding access condition of the file using a definable logical comparison. There are two options for the current security state: the global security state and the local security state. The global security state is the security state of the MF, which means the state of the smart card as a whole. The local security state is the state of the currently selected directory, which means the state of the DF or the state of the directory above the DF. Both the global security state and the local security state can be altered by successful execution of identification and authentication commands using specific keys. When access to a file is requested, a logical comparison function is used to compare the current security state with the state specified in the file as the access condition. If this comparison is successful, the file may be accessed. For example, read access might be allowed in states 5 and above. In this case, read access would be prohibited in any state lower than state 5. Naturally, it is also possible to specify several different states for a particular type of access. For example, read access could be allowed in states 5, 8 and 9. As a rule, the retry counter (error counter) for the associated secret is reset to zero if the comparison is successful. If the comparison is not successful, the error counter is incremented, and if it reaches its maximum allowed value, the associated key is blocked.

At first glance, state-oriented access conditions may appear to be relatively complicated. However, they provide an enormous amount of freedom in the creation of applications, and in principle they can be used for any possible architecture. Their drawback is that they are relatively complex.

In contrast to state-oriented access conditions, command-oriented access conditions define the commands that must be correctly executed prior to the access. This primarily involves authentication and identification commands. Command-oriented access conditions are widely used in the smart card world, with the best-known example being smart cards for the GSM system. With command-oriented access conditions, the access table in the file contains information about the commands that must be successfully executed for each type of access. In many cases, the commands are further assigned to specific keys. In practice, for instance, the condition for read access to a file may require prior identification of the user by means of the VERIFY command and the user's No. 1 PIN. In this case, the file can be read only after this command has been successfully executed. The advantage of this type of access protection is its simple structure, which is generally suitable for the majority of applications. However, additional overhead is generally required if command-oriented access conditions are used, particularly with multiapplication smart cards. For instance, the access tables would have to be extended in the case of an operating system that supports downloadable program code, since explicit references to specific commands must be present in these tables. Consequently, this type of file access condition is somewhat inflexible in certain situations.

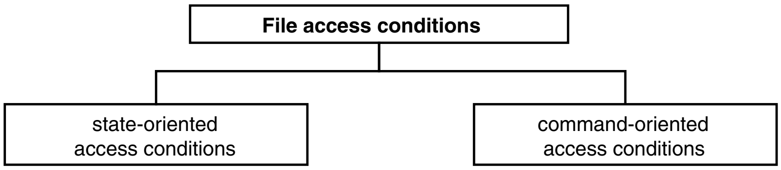

File access conditions
state-oriented access conditions
command-oriented access conditions

Figure 5.25 Classification of the two possible types of file access conditions

All possible types of access to an EF must be precisely governed by means of access privileges. The number of commands that this involves varies, depending on the operating system. Some of the most commonly used file access commands are the following:

APPEND	Enlarge a file
DELETE FILE	Delete a file
INCREASE/DECREASE	Computations within a file
INVALIDATE	Block a file
LOCK	Permanently block a file
READ/SEEK	Read or search within a file
REHABILITATE	Unblock a file
WRITE/UPDATE	Write within a file

The access conditions of a DF are fundamentally different from those of an EF. They specify the conditions under which specific commands may be executed within the directory in question. The three most important access commands are:

CREATE	Create a new file
DELETE FILE	Delete a file
REGISTER	Register a new file

State-oriented access conditions are primarily used in multifunctional smart card operating systems, such as STARCOS, since they are highly flexible and easily modified. However, most large smart card applications, such as GSM and the German Eurocheque system, use command-oriented access conditions to control file access. This approach requires slightly less program code and memory than state-oriented access conditions. Of course, this comes at the price of somewhat lower flexibility. Both approaches have their advantages and disadvantages, and most arguments in favor of one or the other are ultimately based on philosophical issues related to the design of operating systems. However, a method that can be used to flexibly fashion access conditions in a general form for controlling access to resources (including files) has now been specified in the ISO/IEC 7816-9 standard. This highly powerful concept is described in more detail further on.

5.6.6 File attributes

Within its object-oriented definition, every EF has special attributes that define supplementary properties of the file. However, this depends on the operating system and the application area of the smart card. These attributes define properties of EFs that are primarily related to the EEPROM, and they arise from the potential uncertainty of the file contents and the possibility of write errors in EEPROM operations. These attributes are defined when the file is created and usually cannot be changed afterwards.

WORM attribute

One of the attributes based on the EEPROM storage medium is called WORM (write once, read multiple). If a file has this attribute, data can be written to the file one time only, but they can be read an unlimited number of times. This attribute can be implemented either in the hardware of the EEPROM or as a software function. The WORM attribute can be used, for example, to write a serial number in a file once and forever. This attribute is also used with personalization, in which information such as the cardholder's name and the expiry date are permanently written to the card.

This attribute is intended to be used to protect sensitive data against being overwritten. The best possible protection is provided if WORM access is possible at the hardware level, which means that the EEPROM has hardware protection that allows data to be written only once. However, even a software implementation provides much better protection than other comparable mechanisms.

Frequent writing attribute ('high update activity')

An attribute that is primarily defined and used in the GMS realm is a flag for 'high update activity'. The only reason that this attribute exists is because an EEPROM has a limited number of write/erase cycles. A file having this attribute can be written very often without having its data content be affected by write errors. This can be achieved by storing multiple copies when writing the data and using a majority vote when reading the data. Triple parallel storage is commonly used for writing, with a 2-of-3 majority vote for reading. An alternative mechanism is to switch from one copy of a multiple data set to another copy if a read error or checksum error occurs, fully transparent to the outside world.

EDC utilization attribute

An attribute that provides special protection for the user data in a file by means of an error detection code (EDC) is used for particularly sensitive data. This allows the 'flipping' of bits in the EEPROM to at least be detected. If multiple storage is used together with EDC protection, it is also possible to correct flipped bits. This ECC (error correction code) attribute is primarily used for electronic purses. Here the flipping of a memory cell amounts to the actual loss of money, since the current amount of money in the purse is stored in the file. The EDC and ECC file attributes are thus used to minimize the effects of bit flipping.

Atomic write access attribute

Recent smart card operating systems often include a mechanism that ensures that when a file is accessed for writing, the writing operation is executed either completely or not at all.[5] Since this mechanism more than doubles the write access time for a file, it should in principle not be used for all files. A separate attribute allows this writing mechanism to be selectively applied to each file.

Concurrent access attribute

Smart card operating systems that support several logical channels often have a special file attribute for concurrent access. This attribute explicitly allows a file to be accessed for reading or writing by two or more commands at the same time if the smart card receives these commands via different logical channels that are concurrently open. It is important for this attribute to be specifically marked for the file, since with parallel access via different channels it is possible for data to be modified via one channel immediately before or after they are read via another channel. If the two processes are not synchronized, the data that are read will vary depending on when the commands reach the smart card. Consequently, concurrent access is generally not allowed, and access by any other channel is temporarily blocked when a file has been selected. Only after the file has been deselected is it possible for it to be accessed by another channel. The concurrent access attribute disables this block for a particular file. In this case, the relevant applications in the terminal are responsible for synchronizing parallel read and write processes. Of course, there is no problem if they only access the file for reading.

Data transmission selection attribute

The file management systems of smart cards that have both contact and contactless interfaces sometimes include a file attribute that determines which of the two interfaces may be used for accessing the file. This makes it possible to specify for each individual file whether commands may access a file via the contact interface and/or the contactless interface. With an electronic purse, for example, this attribute makes it very easy to allow purchases to be made only via the contact interface and the card to be loaded only via the contactless interface.

5.7 FILE MANAGEMENT

All files in a smart card are stored in the EEPROM. This is the only type of memory in the smart card that can retain stored data without power and that also allows data to be altered if necessary (when power is available). It also provides the only means to save information from one session to the next, since the contents of the RAM are lost when the smart card is deactivated, and the contents of the ROM cannot be altered after the chip has been manufactured.

[5] See also Section 5.10, 'Atomic Operations'

In earlier smart cards, files were directly accessed using physical addresses. Actually, there were no files in the true sense of the word. Instead, the entire memory was linearly addressable from the outside and could be accessed using write and read commands. However, this is not allowed in modern operating systems, for reasons related to security and applications. Object-oriented file management, with access condition information located directly in the files, is currently the standard. The organization and management of these files is the task of the file manager portion of the operating system.

With an objected-oriented structure, every file must have a file descriptor that contains all the information relevant to the file itself. In smart card technology, the file descriptor is referred to as the file header. The data content of a file, or in other words the user data, is located in the ‘body’ of the file.

The information contained in the file descriptor depends strongly on the capabilities of the file manager. However, the file descriptor must contain at least the following items:

• file name	(e.g. FID = '0001')
• file type	(e.g. EF)
• file structure	(e.g. linear fixed)
• file size	(e.g. 3 records of 5 bytes)
• access conditions	(e.g. READ = after PIN code has been entered)
• attribute	(e.g. WORM)
• link to the file tree	(e.g. directly under the MF).

With an EF or the MF, the file name is the two-byte file identifier (FID). With a DF, the application identifier (AID) also forms part of the file name. The file type, which may be MF, DF or EF, must also be indicated.

Depending on the file type, there may be an element in the header that describes the internal structure of the file (transparent, linear fixed, linear variable, cyclic or executable). All information relating to the length of the transparent data portion, or the number and length of the records, also depends on the file type.

Besides the basic attributes of the file, which have just been described, the operating system needs even more detailed information about the access conditions, which means which commands are allowed to access the file and what types of access are allowed. The access conditions must be individually specified for every possible command. Special file attributes, such as high update activity, WORM or EDC protection, can also be marked if they are supported by the file manager.

All of the above information relates to the file as an isolated object. In order to define the location of the file in the file tree, an additional pointer is needed to specify the exact position of the file within the MF or DF.

Pointer-based file management

In simple operating systems, the file headers have fixed lengths that depend on the file type (MF, DF or EF). This reduces the amount of administrative and computational overhead for internal file management. However, such an arrangement has the major disadvantage of being relatively inflexible with regard to extension. Another drawback is that it does not allow an unlimited number of different access conditions to be implemented for any given EF, or even a very large number. This is because the memory that would have to be reserved to accommodate a large number of access conditions in the header would not be fully used by most applications. Consequently, variable-length headers are very popular in smart card file management systems. Such headers can be automatically adapted to the needs of specific applications by the smart card operating system.

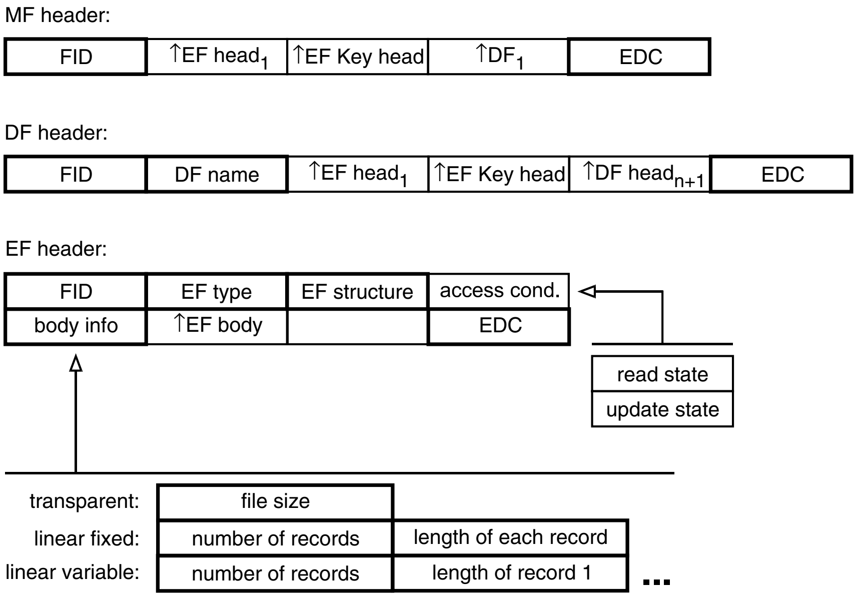


Figure 5.26 A possible file header structure for MF, DF and EF files (internal and working) in a pointer-based smart card file management system. Heavy borders indicate TLV-coded data objects that must be present, while light borders indicate optional data objects. The numbering is valid only within the directory in which the file is located, rather than globally for all files. This structure is based on the requirements for a simple file management system defined in Small-OS. Here 'access cond' stands for 'access condition' and 'head' for 'header'

FAT-based file management

A type of file management that is widely used for the hard disk drives of PCs is based on file allocations tables (FATs). This method can be adopted for use in the memory management of smart cards without any essential modification. With this method, the EEPROM to be managed is divided into many equal-sized pieces, which are called sectors. Ideally, the sizes and start addresses of the sectors should correspond to the EEPROM pages. This allows write and erase operations to be performed on integral EEPROM pages.

The FAT contains pointers to each of the sectors of the memory, with FAT entries also being linked to each other by means of pointers. Vacant FAT locations and defective sectors

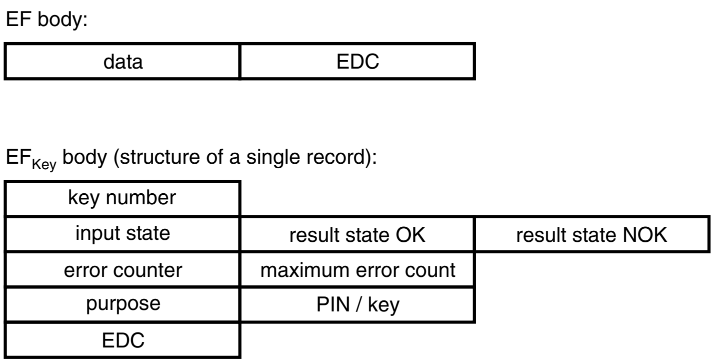


Figure 5.27 A possible file header organization for internal and working EFs in a smart card file management system. All TLV-coded data objects must be present. This structure is based on the requirements for a simple file management system defined in Small-OS. 'EDC' stands for 'error detection code'

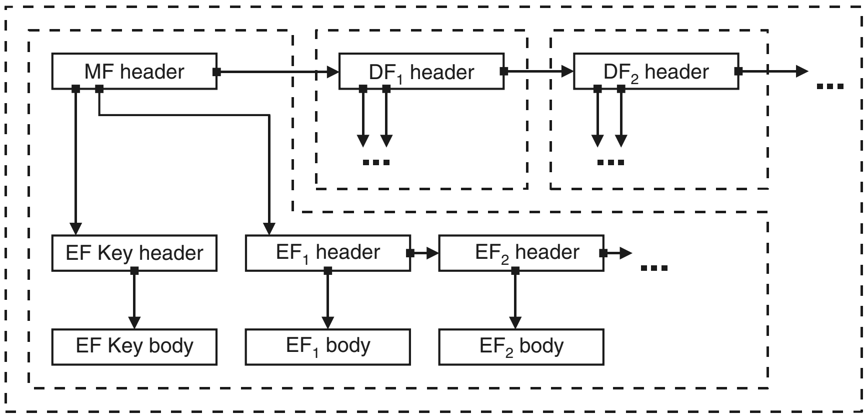


Figure 5.28 Outline of a possible architecture of the pointer and data structures in a smart card file management system. This is based on the requirements for a simple file management system defined in Small-OS. The dashed outlines demarcate the memory available to each directory

are marked by special entries. With regard to memory space, the FAT can be significantly compressed if there is a direct relationship between FAT locations and sectors. In this case, the sector pointers within the FAT are unnecessary.

Figure 5.29 shows a possible implementation of a FAT for file management in a smart card. The file descriptor, which contains the essential information about the structure of the file, contains a pointer to the initial entry in the FAT. In the FAT, a number of sectors corresponding to the size of the file are linked using internal FAT pointers. The final entry always contains an end-of-file (EOF) marker. There is a one-to-one relationship between the FAT entries and the sectors in the memory that contain the user data belonging to the file descriptor.

The question of whether file management is implemented using file headers and file bodies linked by pointers or using a FAT largely depends on various technical considerations and

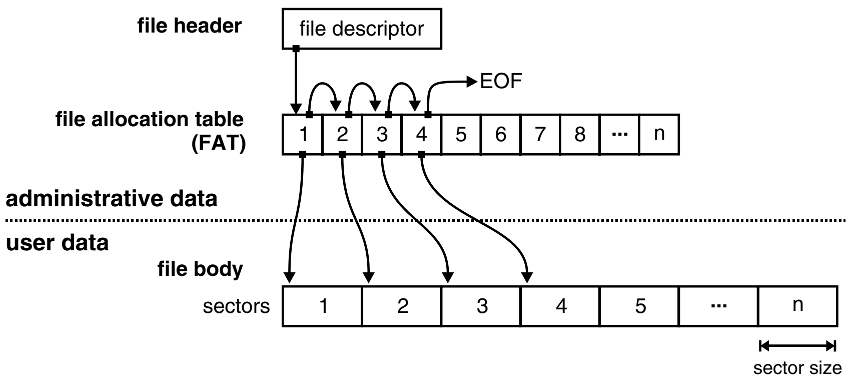


Figure 5.29 Basic structure of a file allocation table (FAT) for file management in a smart card. The end of the user data is indicated in the FAT by an EOF (end of file) marker

constraints. Both approaches have advantages as well as disadvantages. A FAT-based file management system requires memory for the FAT itself, and particularly in the case of small files, this is disproportionately larger than the amount required by a pointer-based system. However, memory fragmentation does not occur in a FAT-based system, since it fundamentally cannot occur in such a system.

Requesting and releasing memory within the memory management system of a smart card is implemented using a variety of services. At the lowest level, these services include requesting and releasing memory, increasing memory, reading and writing data and writing data as an atomic operation. The file management interface constructed on top of these services usually includes services to create and delete files, read data from files, write data to files, select the MF, select the higher level DF, select a file using its FID and select a file using its DF name.

Memory partitioning into pages

The limited number of write/erase cycles of the EEPROM and the partitioning of the EEPROM into pages create a general problem for file management. This has a considerable effect on the entire design of the file manager and the internal file structures. The sizes of both file headers and file bodies must be adapted to the predefined size of the memory page to prevent them from being split by page boundaries. File management information in the memory must also be strictly separated from the actual file data contents. If this were not the case, undesired side effects could occur between the management data in the header and the user data in the file body. These could destroy the entire internal security structure of the smart card operating system. This is briefly illustrated in the following example.

Suppose the access conditions for a file containing secret, non-readable keys are stored on the same memory page as the public, writeable data of another file. If a write operation to this file is interrupted, for example by pulling the card out of the terminal, this will affect the access conditions stored on the same page. In the worst case, no access conditions at all will remain, and the file containing the secret keys can be read by everybody. It is thus fundamentally important to store the internal file structures for file management and the user data on separate memory pages.

DF separation

One way of understanding the function of a DF is to see it as representing all of the memory provided to a particular application. Within this region, the application operator is fully responsible for his application and can essentially do and permit whatever he wishes. However, he must under no circumstances be able to access a memory region assigned to a different application from within his own region in any manner, nor should he be able to read or alter data stored in another region. Consequently, some smart card operating systems have special mechanisms that always test every memory access to see whether the physical address is located within the limits of the current DF. If this is not the case, the process is terminated and a severe internal error is reported.

This address monitoring is currently performed by suitable software routines in the operating system, due to the lack of hardware support. The security of this solution is naturally significantly lower than what could be achieved with suitable hardware, since it is more easily bypassed. In the future, smart card microcontrollers will probably have memory management units (MMUs), as do all current CPUs.[6] Such units can be used to achieve secure control over memory accesses within a DF. Until then, the only option is to employ suitable operating system routines to monitor the address boundaries of the DFs.

This principle of memory organization – storing all of the components of a DF within a single contiguous region of memory – had to be abandoned in the development of recent operating systems, since memory management would otherwise have been too inflexible.

Free memory management mechanisms

The small amount of available memory imposes major restrictions on smart card operating systems with regard to free memory management for the EEPROM. Only since the mid-1990s have operating systems been available that can create and subsequently delete files (DFs and EFs) after the card has been personalized. Given the secure nature of a smart card, this must naturally be protected in a cryptographically flawless manner. The ideal solution is to execute the appropriate commands in the secure messaging mode following an initial mutual authentication.

Free memory management must also take into account the fact that in the event of a sudden loss of power, which can for example occur if the card is pulled out of the terminal, the entire file tree must remain in a well-defined state. Particularly in such circumstances, the security of a smart card can completely collapse if file pointers suddenly become undefined. The satisfactory approaches to solving this problem once again involve atomic operations, although in this case, due to the large data volumes, such processes require a relatively large amount of time and correspondingly large memory buffers.

There are various realization strategies for memory management, and incidentally for all types of file management in smart cards, and they differ significantly in terms of software implementation. This is illustrated in Figure 5.30 and the associated description, using smart card memory management as an example.

[6] See also Section 3.4.3, ‘Supplementary hardware’

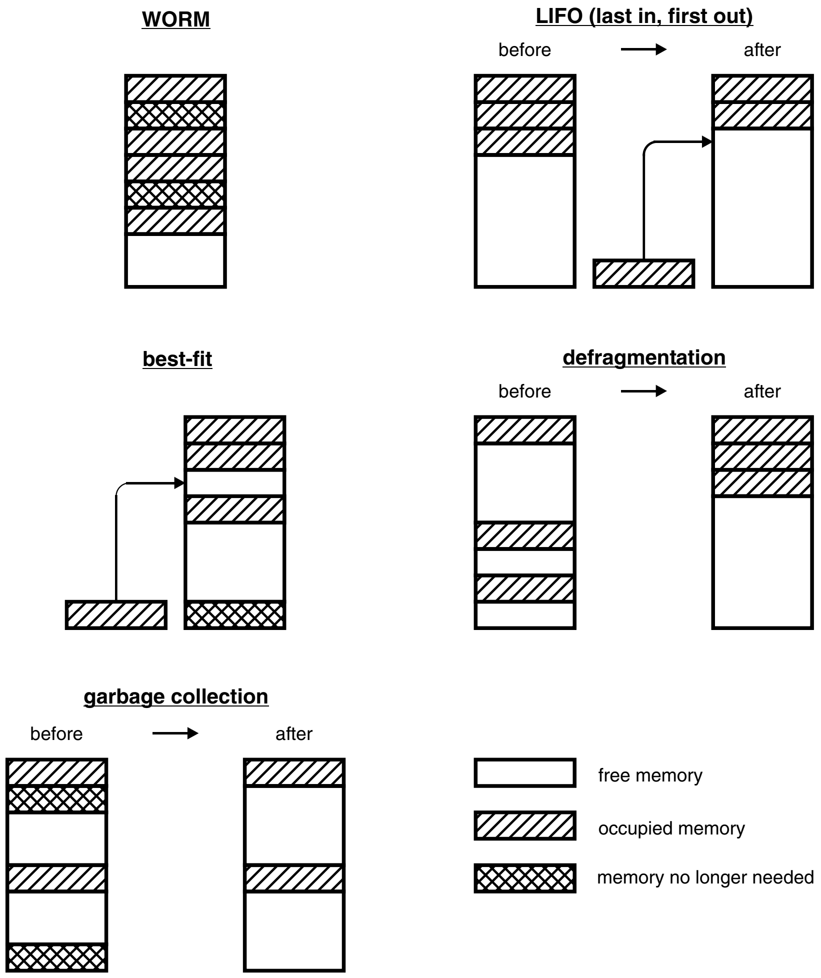


Figure 5.30 Typical memory management mechanisms for smart card operating systems. A detailed explanation is provided in the text

The simplest type of memory management is a sort of write once, read multiple (WORM) functionality. With such an approach, once memory space has been occupied by storing a file, it remains occupied even after the file has been (logically) deleted. The administration overhead with this approach is minimal.

A somewhat more elaborate method, from a software engineering point of view, is memory management based on the last-in, first-out (LIFO) principle. With this approach, the most recently created file can always be deleted, releasing the space it previously occupied. This method is often used in simple smart card operating systems. With the somewhat more sophisticated best-fit algorithm, the operating system always attempts to find the smallest suitable region of free memory when creating a new file. When the file is deleted, this region again becomes free and can be used by other files. However, if files of various sizes are frequently created and then deleted, strong fragmentation of the memory occurs relatively quickly. As a result, large files can no longer be created, since no single region of free memory will be large enough to hold the entire file, even if the total amount of free memory is sufficient to hold the file.

This is precisely where the defragmentation process comes into play. When the memory is heavily fragmented, this memory management process, which is relatively complex by smart card standards, repeatedly relocates the files in memory until it arrives at a situation in which all of the free memory forms a single contiguous block. The difficulty here comes from the fact that such an algorithm runs relatively slowly in smart cards, due to the need to perform many time-intensive EEPROM write accesses.

Garbage collection is a process that can be regarded as independent of the processes just described. Operating either on demand or periodically, the garbage collection process searches the entire memory for areas of memory that are no longer used. If an area that is no longer needed is found, the garbage collection process automatically allocates it the free memory pool. A defragmentation process can subsequently be used to combine all of these small memory blocks to form a large, contiguous region of free memory.

Data integrity

Another important consideration is ensuring data integrity. The file manager should always be able to test whether the data in the memory have accidentally changed, which could occur due to factors such as aging. To minimize the administrative overhead for this function, the level of data redundancy and/or the extent of the supervisory protective functions should match the importance of the data. There is thus no need to protect all data with checksums as a matter of principle. Several data elements, such as a complete file header, can be protected as a group, or particularly important data elements can be individually protected. This primarily depends on how often the data elements are altered in the EEPROM and how much memory space the designer of the operating system is willing to sacrifice to ensure data integrity.

Error detection codes are used to ensure data integrity. They are primarily used to protect critical data elements, such as data access privileges and file body pointers in file headers. Checksums based on CRCs are often used for this purpose, since they can be computed relatively quickly and do not require very much program code. However, Reed–Solomon codes are also often used to provide better protection against the typical failure mode of EEPROM cells. These codes are significantly better than CRC checksums for detecting the burst errors that typically occur when an entire EEPROM page has changed.

Cross-application access

Certain smart card functions are only enabled after the card user has entered a PIN code. Since the powers of memory of the average person are limited, it has become common to use only one PIN per smart card, even with cards that hold several applications. Every application in the card thus uses this common PIN. It could be stored separately for each application in an internal EF, but this would require each of the stored PINs to have its own retry counter. If there are five applications, for example, and each application allows three attempts, a total of 15 attempts to guess the PIN will be allowed. In many cases, this is not tolerable with regard to the design and security of the applications. Consequently, some operating systems allow cross-application access to PINs and keys.

This utilization of shared resources is in principle implemented in a manner similar to the alias mechanisms commonly used in PC operating systems. The main difference is that smart card operating systems only allow references to higher level DFs, with the MF being the highest level entity. It would thus not be possible to access a PIN located in an arbitrary DF, but only one located in a higher level of the file hierarchy, such as the MF.

In the case of the above example of a single PIN and retry counter that are shared by several applications, a possible implementation is as follows. The PIN is stored in an internal EF located immediately below the MF. In the application, which is located in a DF below the MF, a reference to the storage location of the PIN is stored in an internal EF. The states resulting from successful and unsuccessful PIN comparisons are naturally stored in the PIN record in the DF, since they apply to only one particular application. If a PIN comparison is triggered, the VERIFY command first accesses the PIN record in the current DF, from which it sees that the PIN and its associated retry counter are located at a different level. It then uses the indicated PIN for the PIN comparison. The state of the currently selected DF stored in the internal EF is set according to the result of the comparison.

This procedure is presently supported by many smart card operating systems in various forms. Particularly for the utilization of data that are shared by two or more applications, it provides a very elegant and cryptographically faultless solution. In addition to allowing PINs and keys to be used across several applications, some operating systems also offer an equivalent mechanism for EFs. This makes it possible to directly access global data in EFs located immediately below the MF without first deselecting the current DF.

5.8 SEQUENTIAL CONTROL

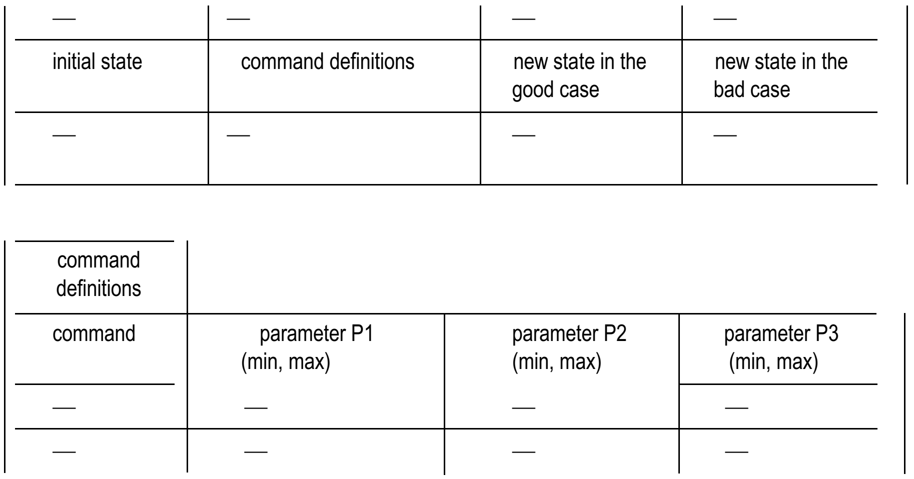
If a state machine must be implemented in an operating system, there are various ways in which it can be constructed. However, certain basic principles must be observed, independent of the operating system and its producer.

In the previously described layered model of the operating system, the state machine must be located after the command interpreter and before the actual execution of the command. The task of the state machine is to determine whether the received command may be executed in the present state. It does this using a table. A basic principle here, as is usual with smart cards, is to use as little memory as possible to provide the state information. In addition, this information must also be structured such that the actual state machine can be built using as little memory as possible.

The state machine needs a certain amount of information to analyze the command held in the I/O buffer. Figure 5.31 shows a possible structure for a smart card state table.

The first data element (initial state) contains the state the rest of data in the data structure is to be processed. This data element could contain a number that directly defines the state to which all the other information applies. This is followed by a subtable that identifies all commands that are allowed in the initial state. It must be possible to identify a single command, a group of commands, all commands or no commands in each subtable.

The allowed parameters for a command follow the command definition in the table structure. In these data elements, it must be possible to define both individual values and ranges of values for the parameters. For example, if the code for the READ BINARY command is in the command field, the P1 and P2 parameter fields could contain the minimum and maximum offset values for a read access to transparent data, while the P3 parameter field could contain



—	—	—	—
initial state	command definitions	new state in the good case	new state in the bad case
—	—	—	—

command definitions			
command	parameter P1 (min, max)	parameter P2 (min, max)	parameter P3 (min, max)
—	—	—	—
—	—	—	—

Figure 5.31 Example of data elements in a data structure for a state machine

both the lower and upper limits for the length. Since multiple entries may be present in this subtable for a given state, additional commands and their parameters could be defined after READ BINARY.

A table entry concludes with the new state that is to be assumed if the command is successfully executed, which means if command execution completes without any errors. The data structure of the example also allows a state to be defined that is to be assumed if command execution is not successful. In order to maintain a high degree of flexibility within the state machine, it must be possible to specify subsequent states either absolutely or relatively. Here 'relative' means that the new state is set by adding or subtracting a value to or from the value of the initial state, while 'absolute' means that the value of the new state is set directly, without reference to the value of the initial state.

In principle, there are no limits to how a state machine can be constructed. The data structure illustrated here is quite suitable for use in a relatively sophisticated operating system. In principle, every possible state machine diagram can be represented in a smart card using the described data structure and a corresponding state machine. Naturally, individual files also have their own supplementary protection against unauthorized reading or writing in the form of access conditions for commands. Nevertheless, sequential control for commands can provide an additional higher level mechanism that complements this object-oriented protection and that thus increases the security of the system. This is actually the primary benefit of using state machines in smart cards.

5.9 ACCESS TO RESOURCES IN ACCORDANCE WITH ISO/IEC 7816-9

The allowed types of access to files can be specified using state-oriented or command-oriented access conditions. With state-oriented access conditions, the current security state is compared to the relevant access condition by means of a logical comparison operation. There are two types of current security state, which are the global security state (the security state of the smart card as a whole) and the local security state (the security state of the currently selected directory). By contrast, with command-oriented access conditions the access table in the file contains information about the commands that must be successfully executed prior to each type of access.

Both types of access conditions (state-oriented and command-oriented) have been and will continue to be supported in various forms by commercial smart card operating systems. Until recently, the biggest problem has been the large variety of implementations and approaches that have been taken. The objective of the ISO/IEC 7816-9 standard is to define a uniform approach to accessing resources in smart cards, and it includes a section specifically devoted to this subject that specifies a very powerful model for access conditions for files as well as commands and data objects. Unfortunately, this model is also complicated. This universal access model unifies both state- and command-oriented access conditions, and to this it adds the possibility of specifying specific command sequences. Furthermore, ISO/IEC 7816-9 also allows the possibility of using specific data object tags to specify a state machine for the accesses. The concept is fully based on TLV-coded data objects, which as is well known, can be used very flexibly to create elegant IT structures.

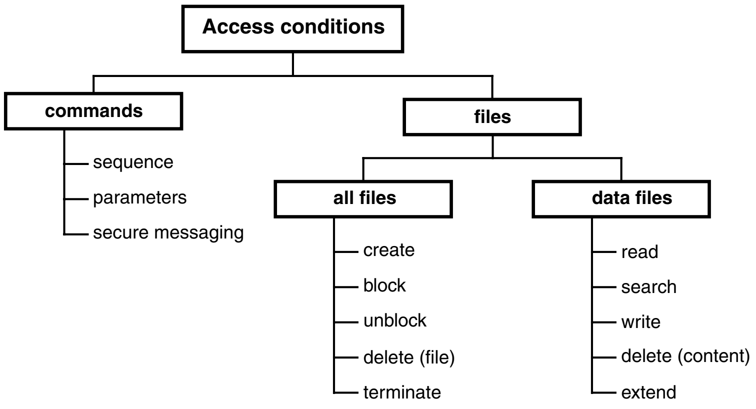


Figure 5.32 Classification of command and files access conditions according to ISO/IEC 7816-9

ISO/IEC 7816-9 defines ‘security attributes’ (SAs) that can be used to govern accesses and non-accesses and attain specific security states in the smart card. These security attributes control accesses to card resources such as files, commands and data objects, as well as SCQL tables and views.

The access control principle is relatively simple. The resource to be protected is assigned a reference (which may be explicit or implicit) to one or more security attributes. These attributes consist of one or more access rules (ARs), which in turn are composed of access modes (AMs) and security conditions (SCs). Each access mode specifies the type of access, such as read or write, while the security conditions specify the security mechanisms (SMs) needed to allow the access conditions to be satisfied. An additional object that may be incorporated into the security rules is the current security environment (SE).

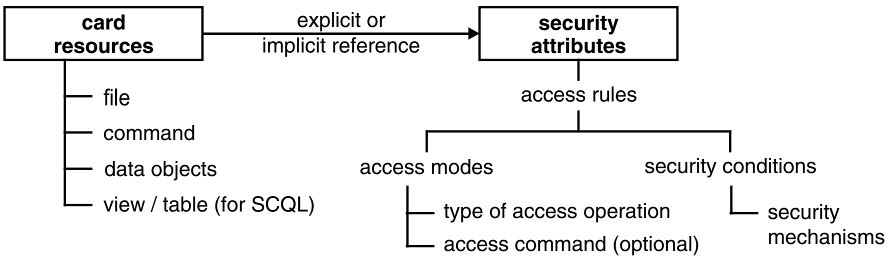


Figure 5.33 Classification of possible card resources and associated security attributes according to ISO/IEC 7816-9

All ISO/IEC 7816-9 security attributes are stored either in 'compact format' (to save memory space) or as regular TLV-coded data objects in the 'expanded format'. Each of these formats provides similar access protection functionality, but the expanded format offers significantly more flexibility. For instance, with this format it is possible to generate a detailed specification of commands and associated parameters for accessing resources.

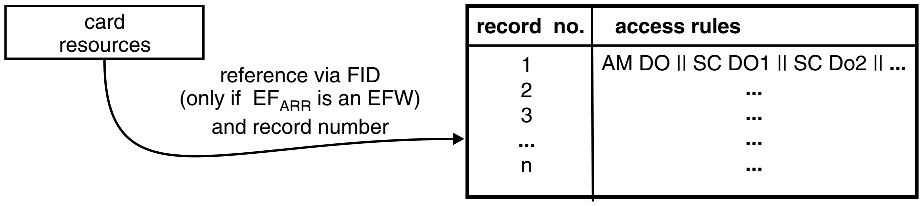

card resources
reference via FID (only if EF$_{ARR}$ is an EFW) and record number
record no.
access rules
1
AM DO || SC DO1 || SC Do2 || ...
2
...
3
...
...
...
n
...

Figure 5.34 Representation of the linking of arbitrary resources of a smart card to associated access rules, which are stored in an EF$_{ARR}$ file

The access rules are stored in one or more EFs with linear variable structure. Such EFs can be internal EFs (EFIs) or working EFs (EFWs), and they are given the name EF$_{ARR}$ (access rule reference EF). The file identifier (FID) of such an EF can be freely chosen. If an EFI is used to store access rules, an implicit access to this EF is made via the operating system every time an access condition occurs. In the case of an EFW, the card resource to be protected contains a reference to the FID of the EF$_{ARR}$. In both cases, the card resource to be protected references the number of the record in the EF$_{ARR}$ that contains the appropriate access rule. The advantage of a selectable EF$_{ARR}$ (that is, an EFW) is primarily that its content can be modified using normal commands and corresponding access conditions. This creates enormous flexibility, since the access conditions for the resources of the smart card can be modified whenever so desired.

It is important to mention that it is naturally not necessary to store a record in the EF$_{ARR}$ for every EF. It is fully sufficient to store a single record for all EFs having identical access conditions and then reference this record from these EFs. This considerably reduces the number of records needed in the EF$_{ARR}$.

The link between the EF and the EF$_{ARR}$ exists in only one direction, rather than possibly being bidirectional. It is therefore not possible to determine which EF or EFs reference a particular record in the EF$_{ARR}$ from within that record. This is important with regard to file

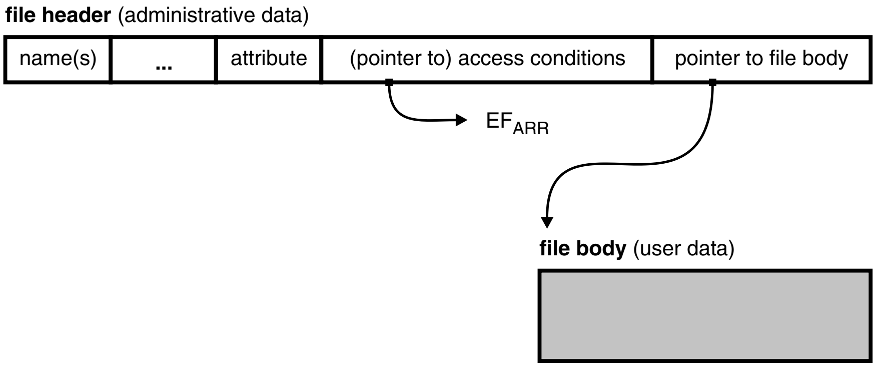


Figure 5.35 Linking an arbitrary file header in the file management system of a smart card to the associated access rules stored in an EF_{ARR} file

management, since when an EF is deleted, it is not allowed to also delete the associated record in the EF_{ARR}, as it may be referenced by one or more other EFs.

When a file is to be accessed by a command, the following procedure is used. First, the operating system tests whether the explicitly or implicitly referenced EF_{ARR} and the appropriate record are present. If they are not, access is denied. Next, the EF_{ARR} is searched for an access mode (AM) data object for the requested access. If this data object is found, the specified security condition (SC) is tested; otherwise access is again denied. If the security condition is met, access to the file with the corresponding command is allowed.

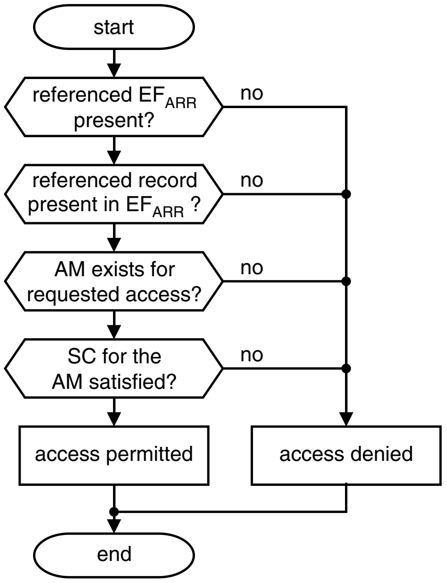

start
referenced EF_ARR present?
no
referenced record present in EF_ARR ?
no
AM exists for requested access?
no
SC for the AM satisfied?
no
access permitted
access denied
end

Figure 5.36 Flow chart showing the essential queries for testing for file access in accordance with the ISO\IEC 7816-9 access model

Table 5.10 Coding of the access mode (AM) byte for DFs as defined by ISO/IEC 7816-9

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	…	…	…	…	…	…	…	b7–b1 according to this table
1	…	…	…	…	…	…	…	b3–b1 according to this table and b7–b4 proprietary
…	1	…	…	…	…	…	…	DELETE FILE (this file)
…	…	1	…	…	…	…	…	TERMINATE CARD USAGE (MF), TERMINATE DF
…	…	…	1	…	…	…	…	ACTIVATE FILE
…	…	…	…	1	…	…	…	DEACTIVATE FILE
…	…	…	…	…	1	…	…	CREATE FILE (create a DF)
…	…	…	…	…	…	1	…	CREATE FILE (create an EF)
…	…	…	…	…	…	…	1	DELETE FILE (lower-level file)

Table 5.11 Coding of the access mode (AM) byte for EFs as defined by ISO/IEC 7816-9

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	…	…	…	…	…	…	…	b7–b1 according to this table
1	…	…	…	…	…	…	…	b3–b1 according to this table and b7–b4 proprietary
…	1	…	…	…	…	…	…	DELETE FILE
…	…	1	…	…	…	…	…	TERMINATE EF
…	…	…	1	…	…	…	…	ACTIVATE FILE
…	…	…	…	1	…	…	…	DEACTIVATE FILE
…	…	…	…	…	1	…	…	WRITE BINARY, WRITE RECORD, APPEND RECORD
…	…	…	…	…	…	1	…	UPDATE BINARY, UPDATE RECORD, ERASE BINARY
…	…	…	…	…	…	…	1	READ BINARY, READ RECORD, SEARCH BINARY, SEARCH RECORD

Table 5.12 Possible security conditions (SC) codes as defined by ISO/IEC 7816-9

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	0	0	0	0	0	0	0	Access always allowed
1	1	1	1	1	1	1	1	Access never allowed
…	…	…	…	0000				No reference to the security environment
…	…	…	…	0001 – 1110				Security environment number
…	…	…	…	1111				RFU
0	…	…	…	…	…	…	…	At least one condition must be satisfied
1	…	…	…	…	…	…	…	All conditions must be satisfied
…	1	…	…	…	…	…	…	Secure messaging
…	…	1	…	…	…	…	…	External authentication
…	…	…	1	…	…	…	…	User authentication (e.g. by means of PIN entry)

Table 5.13 Possible access mode data object (AM DO) codes as defined by ISO/IEC 7816-9. These codes are used in the extended format

Tag	Length	Meaning
'80'	1	AM byte from Tables 5.0 and 5.11
'81'–'8F'	X	Description of CLA || INS || P1 || P2 for specifying the parameters of access commands
'9C'	X	Specific (proprietary) description of a state machine

Table 5.14 Possible security condition data object (SC DO) codes as defined by ISO/IEC 7816-9. These codes are used in the extended format

Tag	Length	Meaning
'90'	0	Access always allowed
'97'	0	Access never allowed
'A4'	X	Control reference template (CRT) for authentication (external authentication or user authentication)
'B4', 'B6', 'B8'	X	Control reference template (CRT) for a command and/or response using secure messaging
'9E'	X	Security condition in accordance with Table 5.15
'A0'	X	The stated security conditions are to be combined in a logical OR manner
'AF'	X	The stated security conditions are to be combined in a logical AND manner

Table 5.15 Coding of the usage qualifier in a control reference template (CRT), as defined by ISO/IEC 7816-9. CRTs are used in the extended format. The tag for the usage qualifier is '95'

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	…	…	…	…	…	…	…	Verification, encryption and external authentication
…	1	…	…	…	…	…	…	Computation, decryption and internal authentication
…	…	1	…	…	…	…	…	Secure messaging response
…	…	…	1	…	…	…	…	Secure messaging command
…	…	…	…	1	…	…	…	Knowledge-based user authentication (e.g. PIN)
…	…	…	…	…	1	…	…	Biometric user authentication
…	…	…	…	…	…	X	X	RFU

To help clarify the information provided in the above tables, some typical examples of entries in an EF_{ARR} are shown in Tables 5.18 through 5.20, coded using both expanded and compact formats.

In summary, it can be noted with regard to access rules in accordance with ISO/IEC 7816-9 that although this system is quite powerful and highly flexible, it also demands a certain price from the smart card operating system in the form of additional memory space. Furthermore, in

Table 5.16 Possible file control parameter (FCP) codes as defined by ISO/IEC 7816-4 and ISO/IEC 7816-9. The tag for the FCP is '62'

Tag	Length	Meaning
'80'	2	Number of data bytes without structure information in a transparent EF
'81'	2	Number of data bytes with structure information
'83'	2	FID
'84'	1–16	DF name
'88'	1	SFI coded in bits 8–4; bits 3–1 set to 0
'8A'	1	Life-cycle status integer (LCSI)
'8C'	variable	Security attribute in compact format
'AB'	variable	Security attribute in expanded format

Table 5.17 Possible control reference data object (CR DO) codes as defined by ISO/IEC 7816-4

Tag	Meaning
'80'	Algorithm reference
'81'	File reference: FID path to a file
'82'	File reference: DF name
'83'	Key reference: for direct use
'84'	Key reference: for computing a session key

practice it is nearly impossible to manually code the access conditions for even simple smart card applications using these rule-based access conditions without the assistance of suitable software tools. Nevertheless, this concept for access to the resources of a smart card will come to prevail throughout the world, since the advantages of flexibility and standardization are highly important.

Table 5.18 Example of the content of an EF_{ARR} record coded in compact format. This record specifies the access conditions for UPDATE BINARY, . . . and READ BINARY, . . . for a file (EF). All other types of file access are automatically prohibited

Data item	Designation	Meaning
'8C'	tag	The tag '8C' identifies an access rule in compact format
'03'	length	The length of the following data is 3 bytes
'03'	AM	The following security conditions refer to UPDATE BINARY, . . . and READ BINARY, . . . , since '03' = °0000 0011°
'00'	SC	No security condition is specified for UPDATE BINARY, . . . which means that the EF may always be written
'00'	SC	No security condition is specified for READ BINARY, . . . which means that the EF may always be read

Table 5.19 Example of the content of an EF_{ARR} record coded in compact format. This record specifies the access conditions for ACTIVATE FILE, DEACTIVATE FILE and READ BINARY for a file (EF). All other types of file access are automatically prohibited

Data item	Designation	Meaning
'8C'	tag	The tag '8C' identifies an access rule in compact format
'04'	length	The length of the following data is 4 bytes
'19'	AM	The following security conditions refer to ACTIVATE FILE, DEACTIVATE FILE and READ BINARY, . . . , since '19' = °0001 1001°.
'90'	SC	Prior user authentication, such as PIN entry, is necessary as a security condition for ACTIVATE FILE since '90' = °1001 0000°. The PIN needed for this is implicitly known to the operating system
'90'	SC	Prior user authentication, such as PIN entry, is necessary as a security condition for DEACTIVATE FILE, since '90' = °1001 0000°. The PIN needed for this is implicitly known to the operating system
'00'	SC	No security condition is specified for READ BINARY, . . . , which means that the EF may always be read

Table 5.20 Example of the content of an EF_{ARR} record coded in expanded format as defined in ISO/IEC 7816-9. This specification allows access to the file in question using READ BINARY, . . . at all times. UPDATE BINARY is only possible after prior successful verification of PIN 1 or PIN 2. All other types of file access are automatically prohibited

Data item	Designation	Meaning
'AB'	tag	The tag 'AB' identifies an access rule in expanded format
'1A'	length	The length of the following data is 26 bytes
'80'	AM DO	The tag '80' indicates that the access mode data object (AM DO) contains a byte with the access conditions (AM byte)
'01'	length	The length of the following data is 1 byte
'02'	AM	The following security conditions refer to UPDATE BINARY, . . . , since '02' = °0000 0010°
'A0'	SC	The following SCs are to be combined in a logical OR fashion
'10'	length	The length of the following data is 16 bytes ('10')
'A4'	CRT DO	The tag 'A4' indicates that the following data contain information about necessary authorizations. These data form a control reference template (CRT)
'06'	length	The length of the following data is 6 bytes
'83'	key reference	The tag '83' indicates that this is a CRT data object for referencing a key
'01'	length	The length of the following data is 1 byte
'01'	key number	Key number 1 is to be used

Table 5.20 (*Cont.*)

'95'	usage qualifier DO	The tag '95' = °1001 0000° indicates that this is a CRT data object for a usage qualifier
'01'	length	The length of the following data is 1 byte
'08'	usage qualifier	Knowledge-based user authentication (i.e., a PIN) is specified as a usage qualifier, since '08' = °0000 1000°
'A4'	CRT DO	The tag 'A4' indicates that the following data contain information about necessary authorizations. These data form a control reference template (CRT)
'06'	length	The length of the following data is 6 bytes
'83'	key reference	The tag '83' indicates that this is a CRT data object for referencing a key
'01'	length	The length of the following data is 1 byte
'02'	key number	Key number 2 is to be used
'95'	usage qualifier DO	The tag '95' = °1001 0000° indicates that this is a CRT data object for a usage qualifier
'01'	length	The length of the following data is 1 byte
'08'	usage qualifier	Knowledge-based user authentication (i.e., a PIN) is specified as a usage qualifier, since '08' = °0000 1000°
'80'	AM DO	The tag '80' indicates that this is an access mode (AM) data object
'01'	length	The length of the following data is 1 byte
'01'	AM	The security condition refers to READ BINARY, . . . , since '01' = °0000 0001°
'90'	SC DO	The tag '90' indicates that accesses are always allowed
'00'	length	The length of the following data is 0 bytes. From this, it can be concluded that a security condition for READ BINARY . . . is not necessary, which means that the EF may always be read

5.10 ATOMIC OPERATIONS

A requirement that is frequently imposed on smart card microcontroller software is that certain parts of it must execute either completely or not at all. Operations that are indivisible and thus fulfill this requirement are called ‘atomic’ operations. They always occur in connection with EEPROM write routines.

Atomic operations are based on the idea of ensuring that when an EEPROM write access occurs, the data in question must never be written only partially. This could happen if, for example, the user pulls the card out of the terminal at the wrong instant or there is a sudden power failure. Since the smart card has no buffer for electrical energy, the software in the card would immediately lose its ability to do anything at all in such cases.

Particularly in the case of electronic purses in smart cards, it is essential to ensure that file contents are complete and correct at all times. For instance, it would be absolutely fatal if the balance of a purse were not completely changed to its new state if a card is suddenly pulled out of the terminal. Entries in log files must also always be complete. Since the hardware of a smart card does not support atomic operations, they must be implemented in software. The methods that are used for this are in principle not new. They have been used for a long time for databases and hard disk drives. The basic procedure of a method that is used in smart card operating systems is described here. This error recovery procedure is transparent to the outside world and thus does not require any changes to existing applications.

For purposes of demonstrating how this method works, let us assume that data destined for a particular file are sent to the smart card via its interface. This would be a typical process with an UPDATE BINARY command, for example. You can follow the exact process by referring to Figure 5.37 while reading the following description.

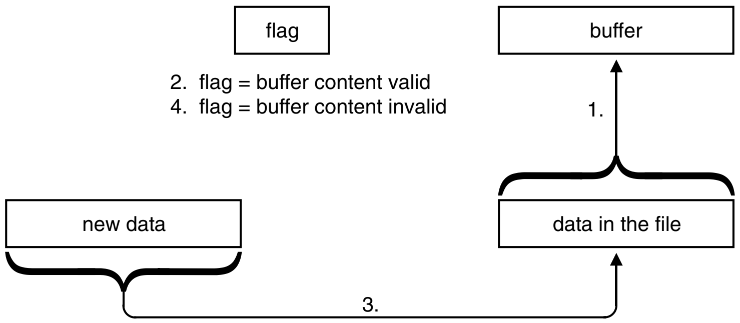

flag
buffer
2. flag = buffer content valid
4. flag = buffer content invalid
1.
new data
data in the file
3.

Figure 5.37 Example of a possible implementation of an atomic operation in a smart card operating system. This procedure can of course be extended to process multiple data elements in parallel

In the EEPROM portion of the operating system, a buffer is created that is large enough to accept all of the necessary data. This buffer has a status flag, which is also stored in the EEPROM. The state of the flag can be set to either 'data in buffer valid' or 'data in buffer not valid'. In addition to the buffer, there must also be suitable locations in memory for the target address and the current volume of the buffered data.

The procedure works as follows. In the first step, the data starting at the target address, for example in a file, are copied to the buffer according to the specified physical address and volume of the data. The buffer flag is then set to 'data in buffer valid'. In the following step, the operating system copies the new data to the desired address, and then changes the buffer flag back to 'data in buffer not valid'. Whenever the operating system starts up, it checks the buffer flag before sending the ATR. If the flag is set to 'data in buffer valid', the data in the buffer are automatically written to the memory area specified by the stored address and volume information.

This mechanism ensures that the data in the file are valid under all circumstances. If a routine is aborted at any time in the process of program execution, the data in the smart card EEPROM can always be restored. For example, if the cardholder pulls the card out of the terminal at the third step of the procedure – in which the new data are written to the EEPROM – the new data will be only partially present in the file. When the card is again activated in a subsequent session, the operating system notices that there are valid data in the buffer and copies them to the appropriate location. This restores the original status of the file, so the contents of all of the files in the EEPROM are consistent. The initial waiting time between the individual bytes of the ATR provides an excellent opportunity to make this correction.[7]

The procedure just described has two serious drawbacks. The first is that the buffer will have the heaviest write/erase stress of all of the EEPROM. Since the number of write/erase cycles for any given region of the EEPROM is limited, it is highly probable that this important buffer region will be the first part of the EEPROM in which write errors start to occur. Such errors would mean that the smart card could no longer be used, since the integrity of the data would no longer be assured. This problem can be made less severe by using a cyclic structure for the buffer, so that the same region is not written every time. Unfortunately, this forces the buffer to take up a relatively large amount of memory. The second disadvantage of this implementation of atomic operations is that it increases program execution time, due to the obligatory write access to the buffer. In the worst case, the file access can take three times as long with this procedure as it would if the data were written directly to the file in the EEPROM. It is thus common to limit such buffering to write accesses to certain files or data elements, rather than buffering all EEPROM accesses. This can be specified by an attribute in the header of each file.

This procedure can be very easily extended to writing not just a single data element into the buffer, but instead several data elements. If this is done, it is even possible to have write accesses to several different files or data elements be performed either completely or not at all. Practically all smart cards operating systems, such as Java Card, support atomic operations, and they may also allow relatively long sequences in the program flow to be marked as being atomic, thereby protecting them against power interruptions.

5.11 OPEN PLATFORM

Due to its activities as one of the largest card issuers, Visa International was confronted relatively early with the problem of managing manifold applications from a wide variety of sources in multiapplication smart cards. This led to the generation of the Visa Open Platform (VOP) specification, which defines an interface inside smart card operating systems for managing smart card applications. Since 1999, the publisher of this specification is the Global Platform Committee [Global Platform], whose function is to standardize technologies for multiapplication smart cards. The name of the specification was also changed to 'Open Platform' (OP) at that time. The OP specification is the most important international specification for application management in multiapplication smart cards, and it can be obtained free of charge from the Web server of the Global Platform Committee.

The OP specification is intentionally independent of any particular operating system, which allows it to be supported by all types of smart card operating systems, both proprietary and open (such as Multos and Java Card). In practice, however, the OP specification has primarily become the *de facto* standard for loading and managing Java-based applications with the Java Card operating system. For instance, the ETSI GSM 03.19 standard regards Open Platform as the standard interface for downloading applications.

The purpose of the OP specification is to provide card issuers with mechanisms for securely managing third-party applications in the smart cards they have issued. To this end, Open

[7] See also Section 6.2, 'Answer to Reset'

Platform defines the basic architecture of a multiapplication smart card. This is shown in Figure 5.38, and it is very important for understanding these mechanisms.

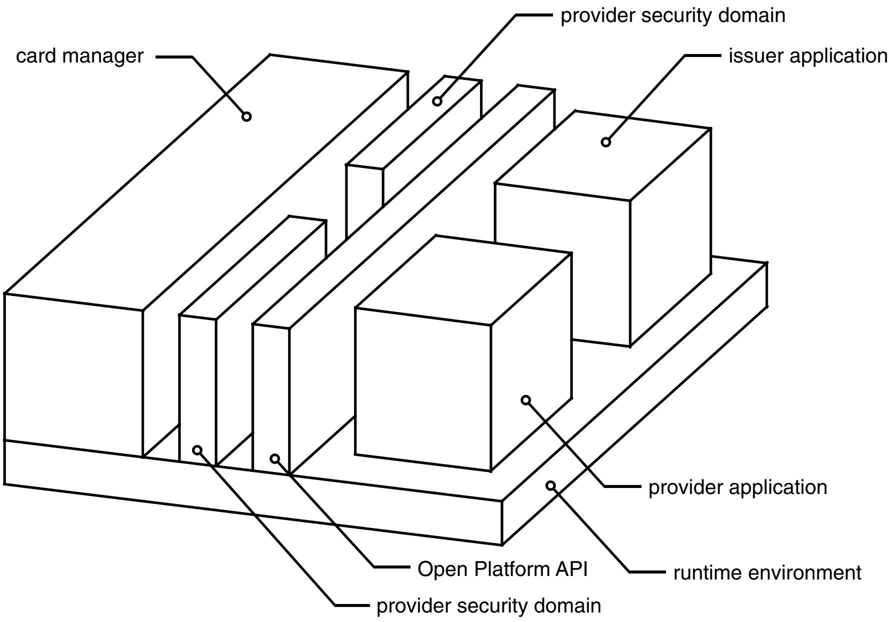


Figure 5.38 Basic architecture and components of Open Platform

The runtime environment forms the foundation for all applications. It provides an application repair facility, a hardware-independent interface (API) and storage space for the data and programs of the various applications. The card manager is built on top of this foundation. It is the core component of Open Platform, and it can be selected via a freely chosen AID. The card manager can thus be regarded as the IT representative of the card issuer in the multiapplication smart card. It manages the runtime environment with respect to the applications and provides them with interfaces for oncard services and interfaces to the outside world. This also includes offcard selection of applications in the smart card and dispatching APDUs coming from the outside world to their corresponding applications. The card manager also ensures, among other things, that the maximum memory sizes specified by the card issuer for application providers cannot be exceeded when their applications are loaded.

In order to manage the information in the smart card related to the various security domains, applications, life-cycle stages of the components and the like, the card manager has a data area called the card registry. This is the central location in the smart card for managing all data related to Open Platform.

In the same way, a security domain is the representative of an application provider for matters related to informatics and security. The function of security domains is to provide keys and cryptographic services for applications that are independent of those provided by the card issuer. Some OP implementations support the downloading of security domains. The Open Platform API (OP API) allows applications to access the management services of the card manager.

Applications in a multiapplication smart card can be divided into two classes with regard to Open Platform. The first class is *immutable applications*, which are loaded into the memory of the smart card during card completion and remain there in a fixed form. The second class is *mutable applications*, which can be loaded, installed and removed either when the smart card is completed or after it has been issued.

The OP specification also describes a generic data format for loading applications into smart cards. The data to be transferred are TLV-coded and consist of the actual data to be loaded for the application (the load file) preceded by an optional data authentication pattern (DAP) block to cryptographically secure the data and the loading process.

Once an application has been loaded into the smart card, the first stage of its life cycle begins. This life cycle is also defined by the OP specification. The first stage is called 'installed', and it means that the application has been stored in the memory allocated to it and properly linked to the operating system, so that it can be run. However, the application cannot yet be selected from outside the smart card when it is in this state. This is possible in the next state, which is called 'selectable'. The following state, which is called 'personalized', is entered after the newly loaded application receives its individual or person-specific data. After this comes an interval during which the application is used by the card user. A possible further state is 'blocked', in which the application cannot be used, although this condition can be reversed. This state can be assumed by the application itself, for instance if it diagnoses a security problem. The 'locked' state is similar to the blocked state, except that it is irreversible. The 'logically/physically deleted' state indicates that the particular application has been logically or physically deleted from the smart card. The actual deletion state depends on the smart card operating system that is used.

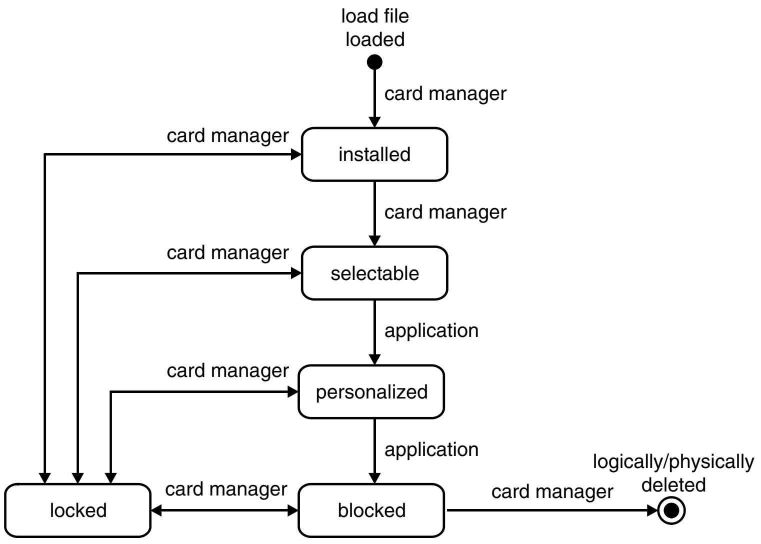


Figure 5.39 The possible states of an application in a multiapplication smart card that complies with the Open Platform specification. The entities responsible for each of the state transitions are indicated

One of the important functions related to Open Platform is 'delegated management'. This refers to functions that allow an application provider to load applications into a smart card (designated loading), install applications (designated installation) and delete applications (designated deletion), all independent of the card issuer. These functions are available to the application provider if his security domain is given these privileges when it is installed by the card issuer.

Several special commands necessary for the functions of Open Platform are defined in the OP specification. Some of these commands are strongly based on the ISO/IEC 7816-4 standard, but they do not fully correspond to the commands in this standard. Table 5.21 provides a summary of the OP commands for smart cards, with brief descriptions.

Table 5.21 Smart card commands defined in the Open Platform specification

Command	Brief description
DELETE	Delete a uniquely identifiable object (e.g., a load file, application or key)
GET DATA	Read a data object; based on ISO/IEC 7816-4
GET STATUS	Read life-cycle state information for the card manager, application and load file. This command is the complement to SET STATUS
INSTALL	Install an application by invoking various oncard functions of the card manager and/or security domain
LOAD	Load an application by transferring the load file
PIN CHANGE/UNBLOCK	Change or unblock a PIN; based on ISO/IEC 7816-4
PUT DATA	Write one or more data objects; based on ISO/IEC 7816-4
PUT KEY	Write one or more new keys or replace an existing key or keys
SELECT	Select an application or file; specified in ISO/IEC 7816-4
SET STATUS	Set life-cycle state information for the card manager, application and load file. This command is the complement to GET STATUS

5.12 DOWNLOADABLE PROGRAM CODE

In the first (German) edition of this book, which was published in 1995, the section entitled ‘Downloadable Program Code’ occupied approximately one and a half pages. In this edition, the amount of text devoted to this subject has increased by a factor of 20. This alone indicates how important this subject has become.

There is probably no risk of exaggeration in saying that a full paradigm change with regard to downloadable program code in smart cards occurred within one year (1997–98). Downloadable program code in smart cards is now considered to be the rule rather than the exception. The reasons for the sudden increase in the importance of downloading executable program code cannot be completely and unambiguously ascertained, even in retrospect. One trigger may have been the floating-point error (in the FDIV instruction) in the then widely used Pentium processor, which became common knowledge in 1994. This could not be corrected by downloading new software, since the error was in the hardware. However, there were software patches (work-arounds) available for many applications.

It is likely that this error is the reason why some large system operators, shortly after it became well known, suddenly made plans to allow executable code to be downloaded to

Terminal		Smart card
SELECT the card manager using the AID	→	return code := selection result
IF (return code = OK) THEN card manager successfully selected ELSE abort	←	*Response* [return code]
Authentication of the outside world with respect to the Open Platform card manager (e.g. using EXTERNAL AUTHENTICATE)	→	
IF (authentication = OK) THEN continue process ELSE abort	←	
INSTALL with parameter 'load an application'	→	
REPEAT {		
LOAD using the data in the load file	→	
IF (return code = OK) THEN loading successful ELSE abort }	←	*Response* [return code]
UNTIL (load file fully transferred)		
INSTALL with parameter 'install an application'	→	
INSTALL with parameter 'make application selectable'	→	

Figure 5.40 Summary of the basic command sequence for storing a new application in a smart card with Open Platform

smart cards. One of the largest applications that can accept executable program code is the German Eurocheque card. However, this capability is not presently used, so it in fact represents only a 'sheet anchor' to be used in case serious programming errors are discovered. In the GSM system, there are also operating systems for SIMs that allow program code for special applications to be downloaded via the radio interface.

However, in contrast to all other computer operating systems, with smart cards it is not common practice to load programs into the cards after they have been issued and then run these programs as desired, despite the fact that this (along with storing data) is actually a primary function of every operating system. There are naturally good reasons why this particular functionality has been largely absent in smart cards up to now.

From a technical and functional perspective, executable program code (stored in EFs, for example) does not present any problems at all. Modern operating systems can manage files containing executable code, and they also allow executable code to be downloaded after the card has been personalized. This makes it possible, for example, for an application provider to have executable code in the smart card that is not known to the producer of the operating system. An application provider could thus load a private encryption algorithm into the card and have it run there. This would allow the knowledge of the security features of the system to be distributed among several parties, which is one of the basic requirements for secure systems.

Another significant reason for allowing program code to be downloaded is that it makes it possible to correct programming errors in fully personalized cards (bug fixing). Known errors in the operating system can thereby be corrected or at least rendered less critical by using downloaded code.

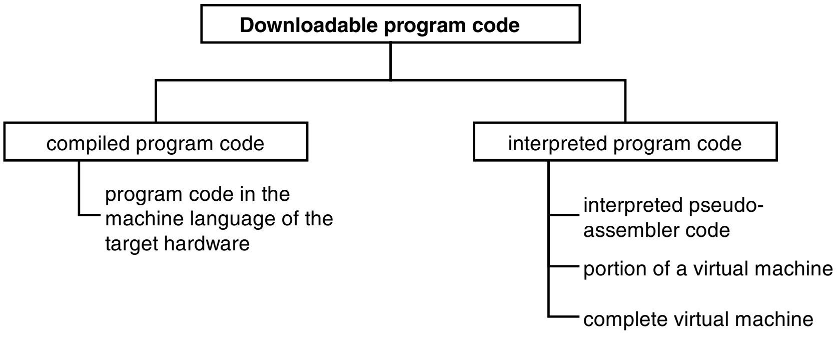

Downloadable program code
compiled program code
interpreted program code
program code in the machine language of the target hardware
interpreted pseudo-assembler code
portion of a virtual machine
complete virtual machine

Figure 5.41 Classification of the methods for downloading and running executable program code with a smart card operating system

There are two basic ways to run downloaded code in a smart card. The first and technically simplest way is to load native code (code that has been compiled into the machine language of the target processor) into files in the smart card. This program code must of course be relocatable, since the memory addresses are not known outside the card. In addition to its technical simplicity, this solution has the advantage that the program can be run at the full execution speed of the processor, which makes it particularly attractive for downloaded algorithms. In addition, there is no need for extra program code for an interpreter. The main problem with this approach is that the downloaded program can also access the memory regions of other applications if the microcontroller does not have a memory management unit (MMU).

The second way to execute downloadable code in a smart card is to interpret the code. In this case, the interpreter can check the memory regions that are being addressed while the program is running. However, the interpretation must run quickly, since there is no benefit to having code that runs slowly. The implementation of the interpreter should also occupy the least possible amount of memory, since only a very limited amount of memory is available. Presently, the best known versions of this approach are the Java Card specification [Javasoft, JFC] and the C interpreter MEL (Multos Executable Language) from Multos [Maosco] (as well as Windows for Smart Cards, which has now again been cancelled). There is even a Basic interpreter for smart cards, which has been available for several years [Zeitcontrol]. Incidentally, interpreters are not suitable for correcting errors in smart card operating systems, since they make a specific region of protected memory available to an application program and thus do not have access to the areas where the operating system routines and data are stored.

The age-old problem with interpreters is their slowness, which is an inherent property of interpreted code. There are several different approaches that can be used to compensate for this drawback and keep the size of the program code of the actual interpreter as small as possible. This simplest approach is to interpret a pseudocode, which ideally is as similar as possible to the machine instructions of the target hardware. The processing speed of the interpreter is thus relatively high because the pseudocode is close to the machine language, while machine-independent program code can still be used. Memory accesses during interpretation can be monitored, but this is not mandatory. A slower solution, which is also somewhat more complicated in terms of programming logistics, is to split the interpreter into an offcard part (the ‘offcard virtual machine’) and an oncard part (the ‘oncard virtual machine’). This approach is taken by many current Java Card implementations. Its main advantages are reliable memory protection and complete hardware independence. However, dividing of the interpreter into oncard and offcard parts has drawbacks. It makes cryptographic protection mandatory for transferring programs between the two parts of the interpreter, since otherwise the oncard part of the interpreter could be deliberately caused to misbehave by using manipulated program code.

The optimum technical solution is to have a complete interpreter in the smart card. This makes it possible to load any desired program into the card and run it without any risk to other programs located in the card. However, the code size of a full interpreter is so large that it will certainly take several years and several generations of smart card microcontrollers before this solution becomes widely established in the smart card world.

5.13 EXECUTABLE NATIVE CODE

Presently, most microcontrollers for smart card still have processors that do not have any sort of memory protection mechanisms or any supervisory or monitoring capability. As soon as the program counter finds itself addressing ‘foreign’ machine code, control of all of the memory and all of the processor’s functions rests entirely with this executable code. At this point, it is no longer possible to restrict the functions of the executable program. Every addressable memory location can be read by bypassing any memory manager or handler that may be present, and memory locations in EEPROM and RAM can also be written. The entire content of the memory could thus easily be sent to a terminal via the card interface.

This is precisely the weak point of downloadable and executable programs. If everyone was allowed to download programs, or if programs could be downloaded by circumventing protective mechanisms, the security of any secret keys or other confidential information within the entire memory region could be no longer assured. This would be the ideal form of attack on a smart card. The card would still behave in the same way as a non-manipulated card with respect to the outside world, but special commands could be used to read out its entire memory or write data to portions of the memory

There is yet another compelling argument against third-party downloadable programs. The producer of the files to be downloaded must know all of the entry points (jump addresses) and calling parameters of the operating system routines in order to use important operating system functions. However, some operating system producers prefer to reveal as little information as possible about the internal processes and addresses of their program code, since they consider such information to be critical with regard to the security of their products. In addition, it would be necessary to verify that the downloaded code does exactly what it is supposed to do without any errors, and without harboring a Trojan horse. This can only be done by an independent party.

The most elegant solution to this problem, and probably the one holding the most promise, is to use a hardware-based memory management unit (MMU) in the smart card, in addition to the actual processor. This approach uses hardware circuitry to monitor the program code while it is executing in order to verify that it stays within its assigned boundaries. Only with such a solution is it possible to allow application operators to download programs that have not been certified by the card issuer, while still preserving the security of the card. Each such application would be assigned a physically contiguous region of memory, representing a DF. The MMU would then monitor the assigned memory boundaries when a downloaded program in a DF is called. If these boundaries were exceeded, the program could be immediately halted via an interrupt, and the application could then be blocked pending further action.[8]

There are two different ways to implement downloadable code capability. The first is to put the program code in an EF whose structure is 'executable'. The content of this file can be executed by means of the EXECUTE command after it has first been selected. Depending on the application, prior authentication may be required. The parameters for running the program are passed to the smart card in the EXECUTE command, and the response generated by the program in the EF is sent back to the terminal.

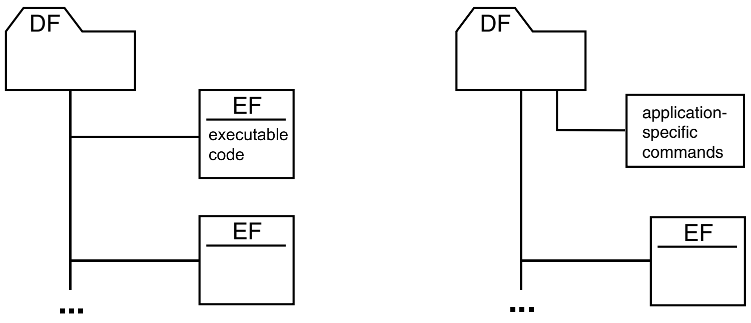


Figure 5.42 The two ways in which executable program code can be entered into a standard smart card operating system: as an executable file (left) or as application-specific commands (ASC) (right)

The second approach takes a somewhat different form, since it is based on the principles of object-oriented design. This option is described in the EN 726-3 standard (among others) as 'application-specific commands' (ASC). As specified in this standard, the complete application, including all of its files and application-specific commands, is contained in a single DF. The program code can be downloaded into a memory region within this DF that is managed by the operating system. This is done using a special command that transfers all the necessary information to the smart card. If the DF in question is selected and a command is then sent to the card, the operating system checks whether the command belongs to the downloaded commands. If it does, the operating system immediately calls the program code located in the DF. If the command instead selects a different DF, the downloaded commands effectively do not exist in the context of the command.

[8] See also Section 3.4.3, 'Supplementary hardware'

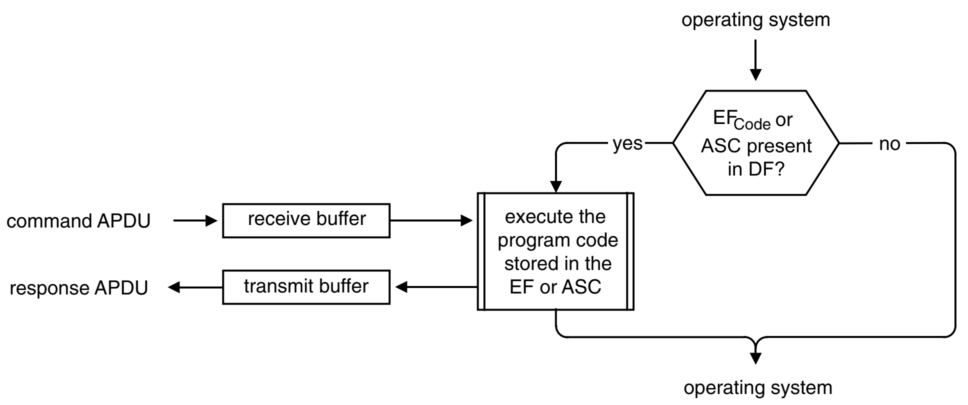


Figure 5.43 Basic calling procedure for executable program code stored in an EF, or programs that work within the framework of ASCs, in a standard smart card multiapplication operating system

Example of native program code downloaded to an EF

There are several large smart card applications whose operating systems allow executable code to be downloaded after the card has been personalized. However, the specifications for this capability are almost always confidential, and in some cases, even the fact that this capability exists is confidential. Consequently, we can only describe the general principles of this capability here, independent of any actual operating system. A possible implementation is described in detail following the description of the basic principles.

In the first place, the program code to be downloaded must satisfy certain basic prerequisites before it can be run in a smart card. It may sound obvious, but the most important prerequisite is that the processor type must be known (for example, 8051 or 6805). Particularly in a heterogeneous environment with many different types of smart card microcontrollers, satisfying this requirement may well involve a certain amount of effort. Along with this comes the requirement that the smart card operating system and its application programming interface (API) must be known, including all entry points and the parameters passed to and returned from its routines.

The program code to be downloaded, which is always native code (machine code of the target processor), should be programmed so that it is relocatable. If it is not, it must be relocated on the fly by the smart card when it is downloaded. The requirement for relocatability, which means that the program can be shifted within the memory, comes from the fact that the memory addresses where the code will be stored are known only to the smart card operating system and not to the outside world. A program is usually made relocatable during the programming stage. In concrete terms, relocatability means (for instance) that jumps to absolute physical addresses are not allowed, but only jumps relative to the address of the jump instruction.

If the program code satisfies all of these requirements, it can in principle be loaded into the memory of a smart card and run there. The program code can of course be structured as desired. Figure 5.44 shows a possible structure, but the actual structure can be completely different, depending on the operating system. The first data element in this example is a unique label that tells the smart card operating system that this is program code. Such a label is commonly called a ‘magic number’. For example, with Java Class files, it is a sequence of four bytes forming the word ‘CAFEBABE’.

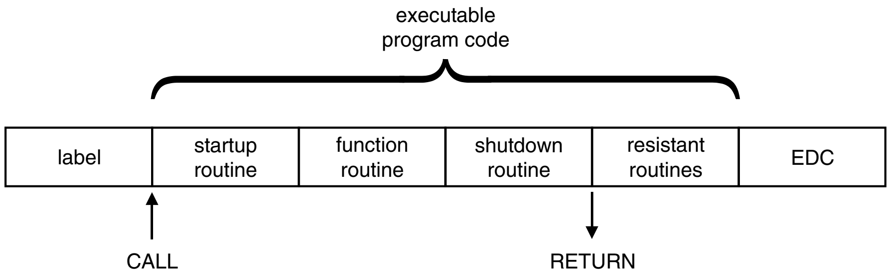


Figure 5.44 A possible structure for native program code that can be downloaded to an EF and then run from the EF

The program code starts just after the label. In this example, it is divided into four parts. The first part contains all of the necessary initializations, data saving and the like. Following this startup routine comes the actual function routine, which contains the program code for the desired task. This is followed by the shutdown routine, which is the counterpart to the startup routine. The shutdown routine ensures that the program is correctly terminated, and if necessary it restores any saved data and adjusts the stack.

The fourth part of the program, which follows the first three parts, is optional. It can include program code to be resistantly incorporated into the smart card software. Bug fixes for the operating system would typically be located here. The three prior routines would then modify pointers or handles so that the routines in this section would be permanently linked into the software of the operating system. The entire process is very similar to the well-known TSR (terminate and stay resident) routines of the DOS era. These routines only had to be called once in order for them to anchor themselves in the operating system until the next reset. In the case of a smart card, these resistant routines would be installed permanently after being called once, rather than only for the duration of a single session.

Here we assume that the downloaded program is called using a Call instruction and that it returns control to the calling program with a Return instruction. In principle, a direct jump to the first machine code instruction (using a Jump instruction) would also be possible, but this would have the disadvantage that the called program would not know which program called it.

For insurance against accidental changes, the entire data block should be protected by an error detection code (EDC). Alternatively, a digital signature could naturally be used to provide additional protection. The smart card would then have the public key, and the producer of the program code would hold the associated private key. This would provide binding assurance that authentic program code can be run in the smart card.

The downloaded program code can be stored either in an EF or in a program memory region within a DF that is not visible to the outside world. The first option is described in some detail below, since it is encountered significantly more often in actual practice.

EFs with transparent structures are ideal for storing program code, since they can be effectively written in several sections using UPDATE BINARY commands with offsets. Also, their maximum length of more than 65 kB is more than adequate, even for extensive programs. These EFs can have an attribute of ‘executable’, which means that the program code stored in them can be directly invoked using an EXECUTE command.

Instead of this, some operating systems have a file structure called ‘execute’ that is based on the transparent structure. This is not particularly important with regard to the outside world, especially since both types can normally be accessed using the UPDATE BINARY and EXECUTE commands. The EF can be selected via its FID or an SFI. The access condition for reading is always set to ‘never’. Writing data is normally allowed after prior authentication and using secure messaging.

Figure 5.45 shows the essential elements of a procedure for loading program code into an EF in a smart card in a secure manner. If a suitable EF is not already available, it must first be created. Figure 5.46 shows in simplified form that the EF must first be selected, after which the program code must be run using an EXECUTE command. Data can optionally be transferred in the body of the command. If necessary, data can also be returned to the terminal in the response in a similar manner. Naturally, the called program must retrieve any data that is sent to it by reading it from the receive buffer, generate its response and write its response to the transmit buffer.

Background system		**Smart card**
Select an EF with an ‘execute’ structure	➔	
Mutual authentication of the smart card and the terminal	🡐 ➔	
Enable secure messaging		
Send *n* UPDATE BINARY commands containing the executable program code in the data segment, protected by secure messaging	➔	

Figure 5.45 A possible procedure for loading executable program code in an existing EF with an ‘execute’ structure. The access conditions for UPDATE BINARY prescribe mutual authentication of the smart card and the terminal and the use of secure messaging for data transmission

Background system		**Smart card**
Select an EF containing executable program code	➔	
Send an EXECUTE command	➔	Check the label (‘magic number’) of the program Check the EDC of the program Run the first machine instruction using CALL

Figure 5.46 A possible procedure for running executable program code, which in this example is stored in an ‘execute’ EF

Due to the severe requirements for unambiguous identification of the microcontroller, operating system and internal software interfaces, as well as those related to system management, it is usually only possible to download programs using an online connection to a background system. The databases located in the background system either hold all the necessary data keyed to the unique chip number, or they receive this information online via a direct end-to-end link with the smart card. Using this information, a program with the desired functionality is selected from those available in the system and transferred to the smart card using the prescribed security mechanisms. The secret keys for this are normally managed and used in the background system exclusively within a security module. The usual procedure for the entire process is shown in Figure 5.47.

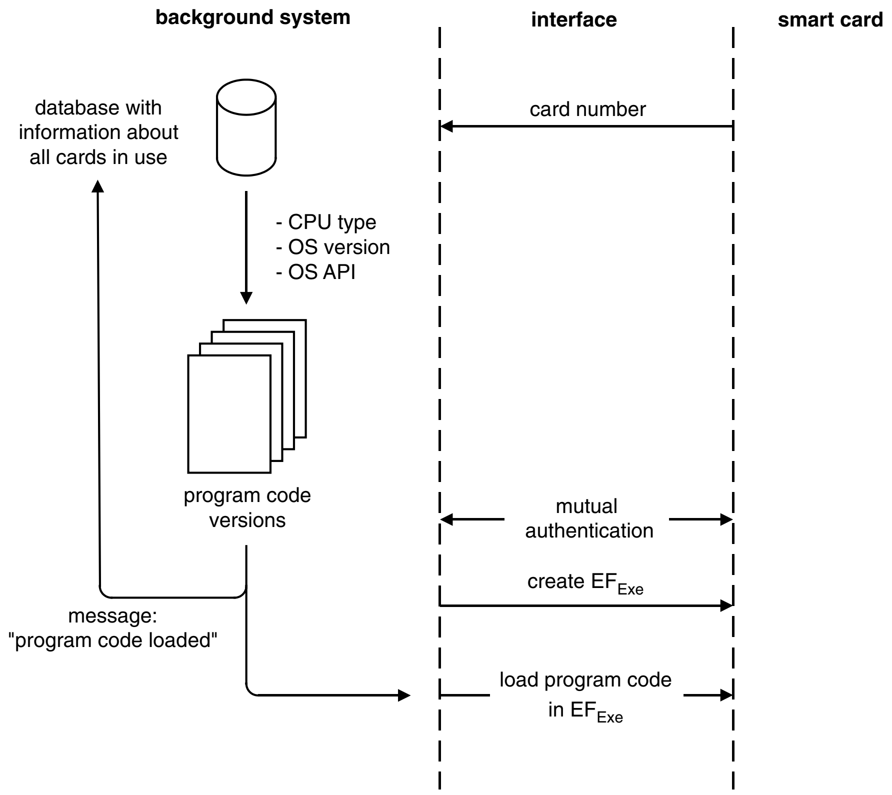


Figure 5.47 The procedure used for downloading native program code online from a background system into a smart card EF. This procedure can be used to transfer various types of program code to a smart card according to the smart card operating system and microcontroller hardware present in the card. This procedure could easily be implemented in a GSM application, for example

The method for loading native code into a smart card that has just been described has some attractive practical advantages. The procedure is simple and robust, and it can be implemented in a smart card operating system using a small amount of program code. The program code to be run does not have to be interpreted, since it can be directly executed by the processor. This yields a high processing speed, so this method can be used to downloading complex algorithms (such as DES, IDEA and the like).

Due to their low processing speeds, interpreter-based systems cannot provide this functionality in the foreseeable future. If there is no hardware-based memory management (MMU) to restrict free access to the memory, this method provides an excellent way to correct errors in the smart card software after the card has been issued. If an error is discovered, this method provides a unique 'back door' that can only be opened using this sort of software downloading. Other technologies, such as Java for smart cards, implement strict and unconditional memory partitioning and thus cannot modify the code of the operating system. If an MMU is present, it still might be possible to invoke an administrator mode to temporarily deactivate memory supervision.

This brings us to the drawbacks. Downloading executable native code presupposes a high level of knowledge of the hardware and/or the operating system of the smart card. It may be necessary to have a separate program on hand for each type of smart card used in the system, even though all of these programs would have the same functionality. The second major drawback of this method is that, for reasons of security, the program must be developed by the card issuer (or under the authority of the card issuer). Loading unknown or third-party programs into the smart card must be strictly prohibited, since the downloaded program assumes control of the microcontroller once it is started and cannot be governed in any manner. Such a program could for example read out the secret keys of other applications present in the card and send them to the terminal via the I/O interface.

Evaluation of the program code by the card issuer provides only weak protection against attacks of this sort. In this case, better protection can be obtained using hardware-based memory management, which makes only certain regions in EEPROM and RAM available to the downloaded program and immediately terminates the program if an attempt is made to exceed the boundaries of these regions.[9] This makes it possible to fully isolate the applications in the smart card. Presently, due to the absence of suitable MMUs, the only available expedient is to carefully review the program to be downloaded.

5.14 OPEN PLATFORMS

With increasing use of Java Card, Multos and Windows for Smart Cards, the term 'open platform' has also come into more general use. This term refers to smart card operating systems that allow third parties to load applications and programs into smart cards without the involvement of the producer of the operating system. Generally speaking, most open platform specifications are public and are generated by a consortium of companies (such as Java Card Forum). Most open smart card operating systems are available from several producers, who provide systems having similar or mutually compatible functions.

The opposite of an open platform is a 'proprietary' platform. This term is often used in a deprecatory sense to refer to a company-specific solution. In many cases, the specifications for such platforms are not fully published or are the property of a single company.

[9] See also Section 3.4.3, 'Supplementary hardware'

However, both terms – 'open' and 'proprietary' – are used in a manner that is by no means unambiguous or non-partisan. In many cases, it is instead marketing-driven. Objectively, many so-called 'open' smart card operating systems are rather proprietary and dependent on a particular company. Truly open platforms in the sense of Linux, with free access to the source code, no licensing restrictions and independence from specific companies or organizations, presently do not exist in the area of smart card operating systems.

5.14.1 Java Card

In 1996, Europay presented a paper on 'open terminal architecture' (OTA) that described and largely specified a Forth interpreter for terminals. The objective was to generate a uniform software architecture for terminals in order to create a basis for hardware-independent terminal programming. Given this, a specific application (such as paying with a credit card) would only have to be programmed once, and the resulting software would run without any modifications on all terminals made by various manufacturers. Although the proposed design has never been fully implemented, it certainly gave rise to extensive discussions in the smart card world.

Consequently, when it became known in the fall of 1996 that Schlumberger was developing a smart card that could run platform-independent programs written in the Java language, no one was particularly surprised. The idea of combining an interpreter with a memory-poor microcontroller was already well known from the OTA proposal. The published specification (Java Card 1.0) provided an application programming interface (API) for integrating Java into an ISO/IEC 7816-4 operating system, in order to allow Java to access the standard smart card file system with its MF, DFs and EFs.

Many producers of smart card operating systems were initially astonished at the idea that a language such as Java, which normally requires well over a megabyte of memory, should be used with smart cards. However, nearly all major smart card manufacturers were represented at the first meeting with Sun, the company that developed and promoted Java, in the spring of 1997.

This was the first conference of what has since become known as Java Card Forum (JCF), which functions as the international standardization panel for Java in smart cards. The tasks of the technical group of Java Card Forum are to define a subset of Java for use in smart cards, to specify the outlines of the Java interpreter (known as the Java virtual machine or JVM) and to define both a general-purpose API and application-specific APIs (for telecommunications and financial transactions, for example). These APIs form the interface between the smart card operating system and Java. The task of the marketing group of JCF is to promote Java technology for smart cards.

The names of the current specifications at the time of writing[10] are *Java Card Virtual Machine Specification, Java Card Runtime Environment (JCRE) Specification* and *Java Card Application Programming Interface*. Up-to-date versions are available at no charge from the WWW server of Java Card Forum [JCF].

[10] Summer 2002

The Java programming language

In 1990, a research group at Sun managed by James Gosling started developing a new programming language. The objective was to create a hardware-independent, secure and modern language that could be used for microcontrollers in consumer products (such as toasters and espresso machines). A large variety of microcontroller types with different architectures are used in such products. This non-uniformity, combined with frequent hardware modifications, makes it difficult for software developers to write portable program code. Remarkably enough, smart cards exactly match the characteristics of the original target application area.

The programming language was first called 'Oak', after the oak furniture in James Gosling's office, but in 1995 it was renamed 'Java'[11] and the objectives were redefined. In the summer of 1995, Sun began to intensively promote Java as the hardware-independent language for the heterogeneous Internet. The slogan coined by Sun, *Write Once – Run Anywhere*, which was frequently quoted at that time, probably provides the clearest indication of the intended level of hardware independence.

The beginning of the widespread use of Java coincides with the beginning of the enormous growth of the World Wide Web (WWW).[12] For a variety of reasons that were not just technical, but which entirely arose from the realms of business politics and worldviews, the new language excited researchers, universities and software companies throughout the world [Franz 98]. As a result, Java became the *de facto* standard for Internet applications within an extremely short time. Naturally, the characteristics of this new language favored this development.

The Java programming language is a fully object-oriented and strongly typed language. It is easily learned by programmers, since it has much in common with C and C++. Java is also a robust language, which means that it does not permit the tricks and popular but dubious techniques possible with C and C++ (for example). For example, no pointers are used in Java, field boundaries are monitored at run time and there is strict type checking. In addition, memory management is handled by Java and an associated 'garbage collector', so 'memory leaks' (a much-feared phenomenon in C and C++) are impossible by design. Java is also a secure programming language, which means that when a program is run the functions that it wants to perform are monitored while it is running, so the runtime environment can stop the program if necessary. This is one of several possible reasons for calling an exception handler. If an exception occurs, a call is made to a specific routine in which the response to this particular case can be defined.[13]

The majority of these features are only possible because Java is an interpreted programming language that is not executed directly by the processor. Java also has other features, such as multithreading capability and support for distributed processing, but these are presently not supported in the smart card environment.

The intention of Sun was to have Java standardized in the form of an ISO standard. However, for a number of reasons this did not happen. One of the reasons was doubtless that the five-year review and revision cycle of ISO is too long for a new programming language such as Java, since it would have made it difficult to make the necessary modifications arising from practical

[11] In this context, 'Java' is American slang for 'a cup of coffee', and does not refer to either the South Pacific island or the French biscuits with the same name

[12] In 1993, there were only three WWW servers in the whole world!

[13] See also [Arnold 00]

experience, as well as delaying the introduction of such modifications. Another conceivable reason may be that before Java technology can be used in products, such as smart cards, a rather costly licensing agreement must be concluded with Sun. This is contrary to the usual convention regarding ISO standards.

The characteristics of Java

Programs written in Java are translated into 'Java bytecode' by a compiler. Java bytecode is simply processor-independent object code. In a manner of speaking, bytecode is a program consisting of machine instructions for a virtual Java processor. This processor does not actually exist; instead, it is simulated by the target processor. This simulation takes place in the Java virtual machine (JVM or VM), which is the actual interpreter. Seen from a different perspective, the JVM is a simulation of the Java processor on an arbitrary target system. The target processor in turn naturally uses native code. The main advantage of this arrangement is that only the JVM, which is programmed in native code, has to be ported to a particular target processor. Once this has been done, the Java bytecode will run on the new system.

Since the runtime task of the VM involves more than just mindlessly interpreting the bytecode and includes type checking and monitoring accesses to objects, the VM is also called the 'sandbox'. This name graphically suggests that a Java program is only allowed to work within its own environment (sandbox) and is not allowed to leave this environment, since the VM will otherwise put a stop to its activities.

A compiled Java program, which means one that has been translated into bytecode and provided with certain supplementary information, is stored in a 'class file'. Class files are executed by the Java virtual machine after they have been loaded. One or more class files constitute an 'applet', which contains a complete smart card application and has its own application identifier (AID). In the context of Java for smart cards, applets are sometimes called 'cardlets'.

The hardware independence of Java naturally has its price, which primarily consists of its very slow execution speed compared with other standard programming languages. This problem has still not been solved in a satisfactory manner, although further developments and improvements of Java are focused on this issue. One step that is already being taken is to use a 'just-in-time' (JIT) compiler, which translates the Java bytecode into the machine language of the processor the first time the program is run. Although this causes the program to run rather slowly the first time it is used, it runs significantly faster afterwards. However, the software of a JIT compiler is rather complex, and the limited memory space of smart cards means that it will be several years before such a compiler can be implemented in a smart card.

Direct compilation of Java programs into the machine language of the target processor would not be worthwhile in the heterogeneous world of smart cards, in contrast to the PC world. The last remaining alternative is to use special hardware. One possibility is to integrate a special Java processor on the semiconductor die, in addition to the 8051 and 6805 kernels presently used. This would provide the advantage of allowing time-critical routines (such as those used for data transmission or cryptographic algorithms) to still be programmed in assembler, while allowing a high-level language, such as Java, to be used for all higher software layers.

Technically, this would not present any problems, as can be seen from the following facts: Sun's Java chip (microJava 701) has an edge length of 7 mm with a 0.25-μm process, works

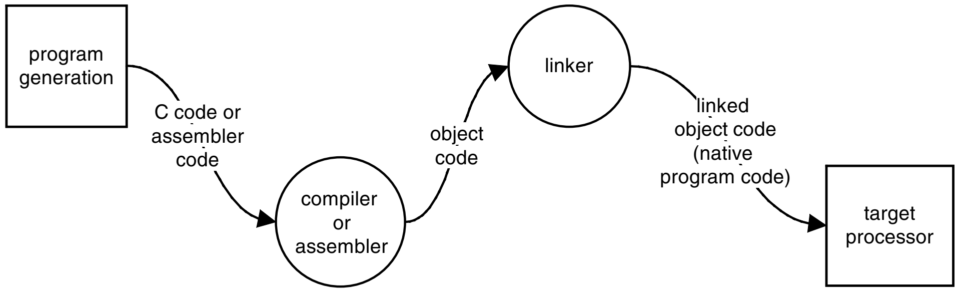


Figure 5.48 Top-level data flow diagram of the usual process for converting source code in C or assembler into executable machine code for the target processor

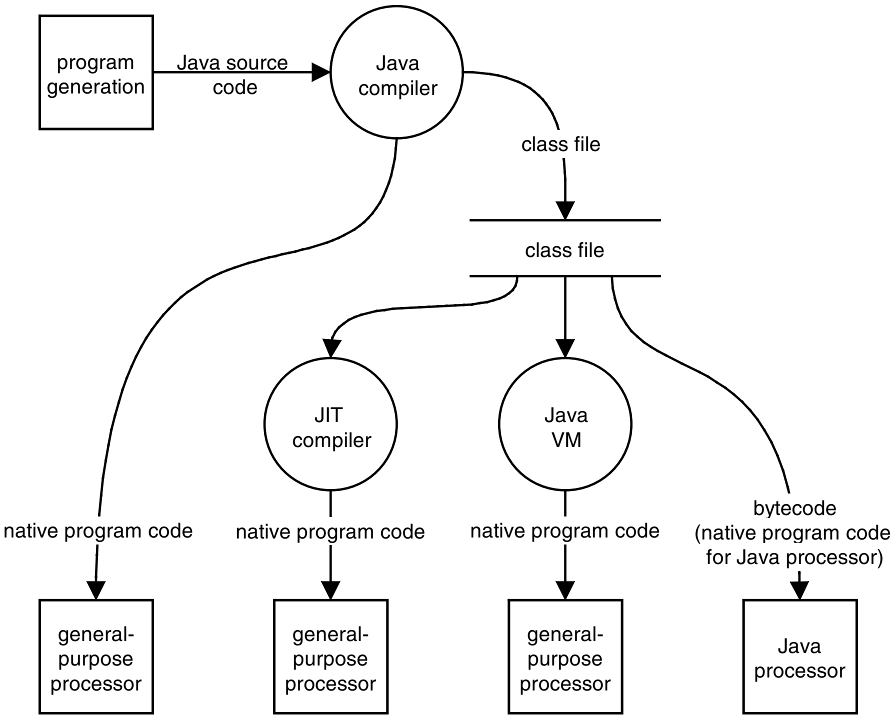


Figure 5.49 Top-level data flow diagram of possible procedures for converting Java source code into executable machine code for a target processor. Although the route shown at the left is provided by some compiler producers, it does not correspond to the original philosophy of Java, since it does not maintain hardware independence

at 200 MHz with a 2.5-V supply voltage, contains 2.8 million transistors and has a power consumption of only 4 W. Since this chip includes many functions that are not needed in smart cards (such as floating-point arithmetic and controlling external memory), it would certainly be technically possible to integrate a stripped-down Java processor meeting the needs of smart cards into a smart card microcontroller.

Another possible solution to the speed problem is hardware extension of the instruction sets of smart card processors. Using this approach, approximately 80 % of the machine instructions of the Java VM could be handled by the processor. From a technical perspective, this would not be particularly difficult, and it would offer a pronounced increase in speed. This approach is presently preferred by many manufacturers of smart card microcontrollers as a solution to the speed problem.

Table 5.22 Program execution time versus programming language, taking C/C++ as the reference. The stated values for programs running on a PC are estimates and are only intended to serve as guidelines. The reference PC processor is a 300-MHz Pentium II, while the reference Java processor is a 200-MHz picoJava 701

Programming language	Execution time
C, C++	1
Java executed by an interpreter	20–40
Java executed by an interpreter using a JIT compiler	≈5
Java program code compiled as native code	1–2
Java executed by a Java processor	≈1.2

Another option for increasing the execution speed of Java bytecode is to use a 32bit smart card microcontroller. However, even with a 32-bit processor the execution speed of interpreted Java code is still 10 to 20 times slower than program code written in C. The primary advantage of these new processors is that they can be used for a wide variety of applications.

The disadvantage of interpreted program code relative to compiled program code, namely inadequate processing speed, can be strongly reduced by using suitable programming interfaces for the interpreter. These applications programming interfaces (APIs) allow interpreted program code to call machine-language routines. The native code routines called in this manner run at the full processing speed of the host processor. Although this may initially appear to be an ideal way to achieve increased processing speed, it has its own problems. The first is that the API for native code must be carefully conceived if it is to be generally useful instead of only useful in a few special cases. Still, this can be achieved with a bit of careful planning. The second problem is more serious. The compatibility and hardware independence of a programming language such as Java is only possible if all APIs are the same. If there are several types of APIs that have different interfaces or provide different functions, using a standard programming language does not provide any significant advantage. Under such conditions, it would be necessary to make specific modifications to the source code for each platform having its own API. This is why Java Card Forum continues to invest considerable effort in standardizing APIs, since they are fundamentally important for achieving platform independence.

The Java virtual machine (JVM)

The Java virtual machine is the most essential element of the Java technology. It simulates a Java processor and can be implemented in software on any sufficiently powerful processor. If it is desired to run Java bytecode on a new type of processor, the Java virtual machine must be ported to this processor. It is usually written in the C programming language, so the actual porting may not require anything more than a few modifications and a recompilation of the source code. The size of a Java virtual machine on a PC ranges from 100 to 200 kB.

The Java virtual machine has all the elements of a real processor. It has its own instruction set in the form of the bytecode, and it has registers such as the program counter and accumulator. The data to be processed are passed to the virtual machine in the form of a class file, which contains fixed constants, the bytecode to be executed in the form of methods, and various additional information.

Table 5.23 Overall structure of a class file

Data elements of a class file
Label ('magic number')
Version number
Constants pool
Methods
Attributes of classes, fields and methods

Java bytecode takes up very little space and is nearly as compact as machine code. The overall memory space balance relative to native machine code is made worse by the obligatory presence of the virtual machine. The difference naturally becomes greater as the size of the program code becomes smaller relative to the size of the virtual machine.

Bytecode is fundamentally very similar to the machine instructions of a real processor. For example, there are instructions for stack manipulation, logical and arithmetic instructions, instructions that access the registers of the virtual Java processor, and even access methods for arrays. The Java virtual machine and the bytecode are extensively described by Tim Lindholm and Frank Yellin [Lindholm 97].

Due to the significantly restricted system resources of smart card microcontrollers, certain limitations must be imposed on the Java Card VM compared with the original Java VM for PCs. The Java Card VM does not have garbage collection to automatically return memory that is no longer needed to the free memory pool. (Version 2.2 of the Java Card specification, which was not yet officially released at the time of writing of this book, does provide for an optional garbage collector.) Support for class files is also drastically reduced. There are fewer data types available in smart cards, and the bytecode itself is reduced from 149 instructions to 76.

The size of the program code for the oncard Java VM for smart cards is on the order of 40 kB of 8051 machine code when written in C. Approximately 400 bytes of RAM are also needed. The API with the classes from *javacard.framework* and *javacardx.framework* needs 3 to 4 kB of memory, the majority of which is Java bytecode. In addition, at least a rudimentary operating system is needed, with data transmission protocols, cryptographic algorithms and many functions that are closely linked to the hardware. The code size for this is 6 to 8 kB if it is programmed in assembler [Baentsch 99]. In connection with these memory size values, it must be noted that they are highly dependent on the available functions and other complex considerations. Consequently, the values stated above can vary considerably among individual implementations.

Table 5.24 Limitations to the functional scope of Java Card relative to full Java

Function	Java Card	Java
Cloning of classes	no	yes
Cloning of objects	no	yes
Data type: int	optional	yes
Data types: Boolean, byte, short	yes	yes
Data types: long, float, double, character	no	yes
Dynamic downloading of classes	no	yes
Dynamic memory management (garbage collection)	no	yes
Dynamic object creation	yes	yes
Exception handling	yes	yes
Fields for objects	no	yes
Fields for supported data types	only one-dimensional	also multidimensional
Interfaces	yes	yes
Operators	all	all
Packages	yes	yes
Sequence control functions	yes	yes
Threads	no	yes
Virtual methods	yes	yes

Table 5.25 Data types in Java for smart cards, with their memory requirements and ranges of values. The data type 'int' is optional

Data type	Size	Range of values
Boolean	1 byte	true, false
byte	1 byte	–128 to 127
short	2 bytes	–32,768 to 32,767
int	4 bytes	–2,147,483,648 to 2,147,483,647

The correctness of this relatively small program is extraordinarily important, since an error or security gap in the Java VM could undermine the whole idea of providing a secure environment by incorporating Java in the smart card. The design and implementation must therefore be largely error-free. An ITSEC or Common Criteria evaluation is normally employed to verify this. Naturally, the small size of the program code for the Java VM considerably simplifies this process, especially since it allows the complete functionality of the VM to be formally described. ITSEC E4 certification is typical.

Due to the small amount of memory present in smart cards, it was necessary to divide the Java virtual machine into oncard and offcard parts. Static tests can easily be performed outside of the smart card in the Offcard VM without reducing performance or security. The link between the two parts of the VM is formed by data in CAP format. For complete security, such data must be cryptographically protected, ideally using digital signatures, so that they cannot be manipulated during transmission. They would otherwise present an attacker with a promising starting point, since the security mechanisms of the offcard VM could be circumvented using manipulated data.

Table 5.26 Summary of the limitations of Java Card. These broadly defined limits currently do not impose any restrictions on the development of software for smart cards

Classes	An instance of a class may contain at most 255 fields. A class may have at most 256 static methods and at most 256 static fields. A class may have at most 15 interfaces.
Package	A package may contain at most 255 public classes and interfaces.
Methods	A method may have at most 255 local variables and contain at most 32,767 bytecodes.
Arrays	Arrays may contain at most 32,767 fields.
Switch instruction	A switch may support at most 65,536 branches. If the 'int' data type is supported, the maximum number of branches in the switch instruction depends on the value range of the selected data type (char, byte, short or int), as with Java for PCs.

Since Java is stack-oriented, it naturally needs a stack as well as a heap. These are created and managed separately for each applet in the smart card. The stack is primarily used for passing data when calling methods, while the heap serves as a storage area for objects. Common sizes are approximately 700 bytes of RAM for the stack and approximately 5 to 8 kB of EEPROM for the heap. However, relatively small applets can certainly manage with 50 to 60 bytes for the stack and a few hundred bytes for the heap.

The Java VM consists of four functional parts, as shown in Figure 5.49: the bytecode verifier, the loader, the bytecode interpreter and the security manager.

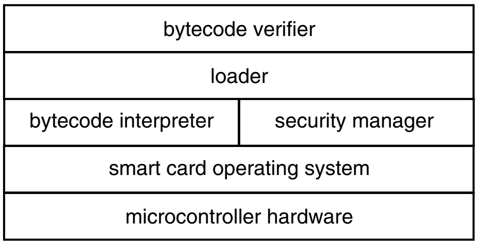


Figure 5.50 The individual components of the Java Card virtual machine

The function of the bytecode verifier is to perform a variety of static tests on the class file passed to the Java VM. It first checks the file format. Following this, it checks the constants pool, checks the bytecodes for syntactical correctness, and checks the arguments of the methods and the object inheritance hierarchies. A few other tests may also be performed. They are described in detail by Frank Yellin [Yellin 96].

After the bytecode verifier is finished, the loader takes the checked data and sends them in CAP format to the actual interpreter in the smart card. For security reasons, the data should be provided with a digital signature so they cannot be manipulated on their way from the loader to the oncard part of the Java VM. The actual loading process is independent of the

Java Card specification. Here the OP specification[14] has come to be the prevailing industry standard.

After being loaded, the executable bytecode is located in the memory of the smart card together with various of supplementary data, where it can be executed by the oncard part of the Java VM. The actual interpreter reads the bytecodes one at time, together with their associated arguments, and converts them into native machine instructions for the target processor. The security manager works in parallel with the bytecode interpreter. Among other things, it constantly checks compliance with the field, stack and heap boundaries. If it detects a violation of the defined security rules, it is authorized to immediately initiate an exception and stop the processing of the bytecode that caused the problem.

Listing 5.1 The basic functions of the main program loop of the Java bytecode interpreter

DO (	Interpreter main loop
fetch and save program counter	Fetch and save the value of the program counter for later comparison.
fetch opcode	
fetch operands	
execute machine instruction	Execute the virtual Java processor machine instruction, which consists of the opcode and the operands.
IF (machine instruction did not alter _ the program counter) THEN (increment program counter)	If the Java machine instruction that has just been executed has not altered the program counter (which means that it was not a GOTO bytecode, for instance), then set the program counter to the next opcode.
) WHILE (opcodes available)	Repeat the loop until all opcodes have been processed.

Java for smart cards

The truly significant benefits of a modern programming language such as Java with regard to its use with smart cards extend beyond the fact that it allows everyone to write programs for smart cards. This would also be conceivable for most smart card operating systems using assembler or C, publicly revealed interfaces and a few modifications. A concept such as Java is also attractive for operators of large systems. They have the problem of being forced to purchase a variety of smart card operating systems running on different microcontrollers from various card manufacturers. Although this multiple sourcing is certainly worthwhile for tactical reasons (due to reduced dependence on a single supplier and the resulting price pressure on the suppliers), it constantly creates problems with regard to compatibility and testing. Within the foreseeable future, it is not possible to have different operating systems from two different producers, with all their various versions, behave the same way at the interface level. This represents a serious problem for system operators.

[14] See also Section 5.11, ‘Open Platform’

From the system operator's point of view, the ideal solution to this problem would be hardware-independent program code that could be executed by an evaluated interpreter in smart cards in a standardized manner. An application program could then be written, tested and evaluated only once, after which it could be run using all different types of smart card operating systems. No differences would be visible 'from the outside', which means at the interface. This would preserve the advantages of multisourcing while eliminating its disadvantages.

You should bear this thought in mind when examining how Java is integrated into smart cards. The first versions only allowed Java bytecode to be stored in an EF located under the MF or under a DF. An EXECUTE command then started up the virtual machine to run the program stored in the EF. An application programming interface (API) associated with the file system allowed data to be read from and written to files.

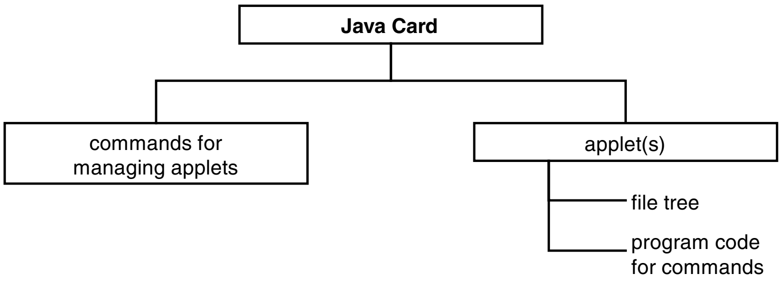


Figure 5.51 The two basic IT components of a smart card with Java

However, this approach has not prevailed. According to the Java Card specification, a smart card with Java has a Java virtual machine that is activated during card manufacturing and deactivated at the end of the card's life cycle. The lifetime of the Java virtual machine thus corresponds to the lifetime of the smart card. When a smart card is electrically deactivated, the Java virtual machine only suspends its activities, while still remaining active. This is a fundamental difference compared with a Java virtual machine on a PC, where the Java virtual machine ceases to exist when the computer is switched off. A file system in accordance with ISO/IEC 7816-4 is no longer provided, since a file system can be constructed using objects within Java applets. There are also several classes that make it relatively easy to construct a file tree that complies with the ISO/IEC 7816-4 specification. The program code and its associated file tree are both part of the applet that is loaded into the smart card. The applet in the card can be selected by its unique AID using the SELECT command. After the applet has been selected, it automatically receives all further APDUs for processing. The program code of the applet can then evaluate and process the commands and their associated data, perform appropriate accesses to the file system and finally generate a response.

This approach provides maximum flexibility and compatibility, since each application is contained in an applet along with its file tree. The actual card commands, such as READ BINARY and MUTUAL AUTHENTICATE, are located in the applet as program code. This allows a single card to have different codings and different procedures in two separate applets that support the same command in mutually independent manners.

Of course, this advantage comes at the price of a significant amount of memory space for the applets, since they must necessarily contain redundant data and routines. This conspicuous

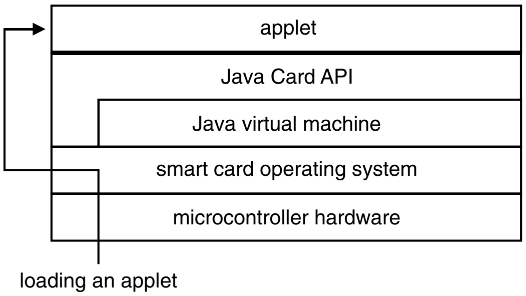


Figure 5.52 The basic process for loading an applet in a smart card operating system with Java

consumption of memory can be somewhat reduced in some cases by allowing the objects of one applet to be shared with other applets (‘object sharing’). For reasons of security, this can only be provided by the applet that creates the object in question. The process of sharing an object cannot be reversed. This means that if an object is made available for access by other applets, it remains available for the lifetime of the card.

The only applet-independent commands that remain are those used for securely loading applets into smart cards. The program code for commands is stored in EEPROM and run by the Java virtual machine. The only common elements for the entire smart card are the transmission protocols.

Java Card Framework

In order to make the programming of smart cards in Java as easy as possible, there are four packages that provide standardized application programming interfaces (APIs) with functions that are useful for smart cards. Three of these packages are mandatory for all Java Cards that do not necessarily require certain key lengths or cryptographic algorithms to be present. The fourth package, which is optional and is distinguished by an ‘x’ (for ‘extension’), can be included if necessary. There are also many other application-specific packages, such as those for GMS functions (GSM 03.48).

Since the program code ‘underneath’ the API can be generated in the machine language of the target processor, this approach not only provides a standard interface but also yields an enormous increase in processing speed.

The obligatory package *java.lang* forms the basis for Java in smart cards. It defines the elementary classes for exceptions. It is complemented by the package *javacard.framework*, which defines the core functions for Java Card applets, such as elementary classes for applet management, data exchange with the terminal and various constants in accordance with ISO/IEC 7816-4. The package *javacard.security*, which contains cryptographic functions, provides interfaces to a variety of cryptographic algorithms. For reasons related to export legislation, this package is constructed such that it does not allow the smart card to be used as a general-purpose encryption and decryption tool. The optional package *javacardx.crypto*, which contains the interfaces to the associated decryption methods, is needed to provide this capability.

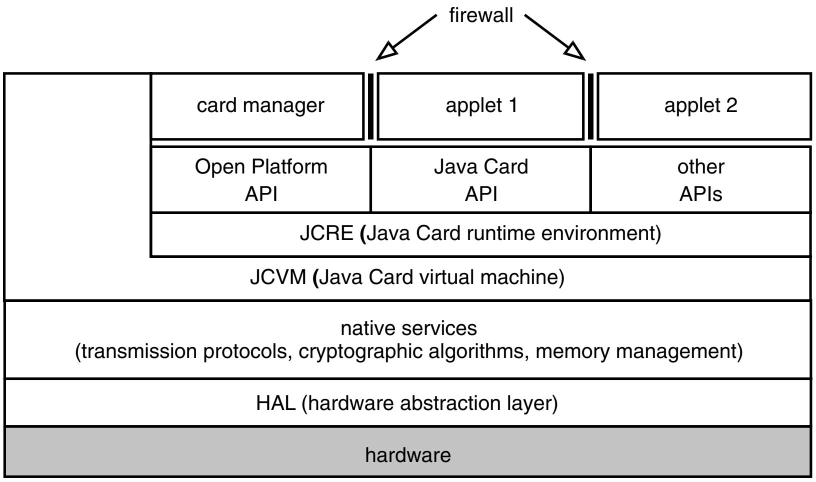

firewall
card manager
applet 1
applet 2
Open Platform API
Java Card API
other APIs
JCRE (Java Card runtime environment)
JCVM (Java Card virtual machine)
native services
(transmission protocols, cryptographic algorithms, memory management)
HAL (hardware abstraction layer)
hardware

Figure 5.53 Schematic representation of the basic architecture of a Java Card system. This example contains several typical APIs as well as application management for the Open Platform mechanisms. The location of the applications is shown in the form of two applets

Table 5.27 The most important classes of the *java.lang* package of the Java Card application programming interface

Class	Description
java.lang.Exception	Class for exception handling for Java Card.
java.lang.Object	The class for all Java Card classes.
java.lang.Throwable	The class of all exceptions and errors for the Java Card.

Software development for Java in smart cards

How does one go about developing a Java program for a smart card and then running it? The first thing the programmer does is to generate the actual Java source code using a text editor. He or she then compiles the source code using any desired Java compiler, which yields the machine-independent bytecode. Up to this point, the process is identical to Java programming for PCs.

The bytecode is then transferred as a class file to the Java Card Converter (which actually means the offcard portion of the Java virtual machine). The Java Card Converter tests the format, syntax, field references and similar items in the program. If all these tests are passed successfully, the Java Card Converter creates what is known as a card application file (CAP file). If necessary, a digital signature is generated for the CAP file, depending on the application. A digital signature provides assurance that the CAP file has been checked by the Offcard VM and is authentic. In the absence of a verifiable signature, the security of the Oncard VM could be bypassed using a manipulated applet, since the Oncard VM cannot perform a full set of tests due to memory space limitations. After this, the applet is loaded into the smart card in the form

Table 5.28 The most important classes of the *javacard.framework* package of the Java Card application programming interface

Class	Description
javacard.framework.AID	Encapsulates the 5–16 byte application identifier (AID) as specified in ISO/IEC 7816-5.
javacard.framework.APDU	Provides methods for exchanging data between the smart card and the terminal at the APDU level.
javacard.framework.Applet	Defines an applet for a smart card.
javacard.framework.ISO7816	This interface encapsulates various constants from ISO/IEC 7816-3 and ISO/IEC 7816-4. Some typical examples are offsets to various data elements within an APDU and various return codes.
javacard.framework.JCSystem	This is one of the most important classes. It includes methods for making objects persistent or transparent and allowing several applets to access an object.
javacard.framework.OwnerPIN	Provides functions for the PIN of the cardholder.
javacard.framework.PIN	This interface represents a PIN with its associated retry counter and a validity status.
javacard.framework.Util	Contains methods for handling fields and data objects and for type conversion.
javacardx.framework.Shareable	This interface is used to identify shareable objects.

of a CAP file. This is usually done using the OP mechanisms.[15] The smart card first verifies the digital signature, which is usually present, and then passes the applet to the Oncard VM once it has been checked. What happens after this is largely the same as for program execution using a virtual machine in a PC. The Oncard VM tests and interprets the bytecode line by line and generates machine instructions for the smart card processor from the bytecode.

In actual practice, the process is naturally somewhat more complicated than what has just been described. It is to be hoped that the developer will not immediately start writing Java code immediately after being given his or her assignment, but will instead use analysis and design methods to determine the actual requirements before starting to program.

In order to quickly locate errors during and after the coding, the developer uses a Java smart card simulator. This allows the developer to follow the execution of the code step by step, examine variables and make any necessary corrections quickly and easily.

Besides this, a suite of tests is run for relatively large projects and those that are critical for security. These tests check all good cases and the most important bad cases for commands and responses. Source code inspection by an independent party may also be included.

As you can see from this example, Java for smart cards significantly reduces development time, and it reduces possible error sources as a secondary benefit. However, coding by itself is only one of many aspects of developing a smart card application. The primary advantage of Java for smart cards is that it allows a large number of developers to generate executable programs for smart cards, rather than just a few software developers employed by card manufacturers.

[15] See also Section 5.11, ‘Open Platform’

Table 5.29 The most important classes of the *javacard.security* package of the Java Card application programming interface

Class	Description
javacard.security.DESKey	This interface provides access to DES and triple DES with two or three keys.
javacard.security.DSAKey	This interface provides access to DSA.
javacard.security.DSAPrivateKey	This interface provides access to DSA for generating signatures.
javacard.security.DSAPublicKey	This interface provides access to DSA for verifying signatures.
javacard.security.Key	The basic interface for all keys.
javacard.security.KeyBuilder	The class for generating keys.
javacard.security.KeyPair	A container class for a key pair (public and private keys).
javacard.security.MessageDigest	The base class for all hash algorithms.
javacard.security.PrivateKey	This is the interface for private keys for asymmetric cryptographic algorithms.
javacard.security.PublicKey	This is the interface for public keys for asymmetric cryptographic algorithms.
javacard.security.RandomData	The basic class for generating random numbers.
javacard.security.RSAPrivateCrtKey	This is the interface for private keys for asymmetric cryptographic algorithms in combination with the Chinese remainder theorem (CRT).
javacard.security.RSAPrivateKey	This interface provides access to RSA for generating signatures.
javacard.security.RSAPublicKey	This interface provides access to RSA for verifying signatures.
javacard.security.SecretKey	This is the interface for all symmetric keys.
javacard.security.Signature	The base class for all signature algorithms.

Table 5.30 The most important classes of the *javacardx.crypto* package of the Java Card application programming interface

Class	Description
javacardx.crypto.KeyEncryption	This interface provides access to the keys for decryption.
javacardx.crypto.Cipher	This is the base class for all encryption algorithms.

If a particular applet must be loaded into a very large number of smart cards in identical form, some implementations of Java Card also allow part of the applet to be defined as a portion of the ROM mask. However, the parts of the applet that contain modifiable code must remain in EEPROM. Such an applet is frequently called a 'ROMable applet'.

In generating Java applets for smart cards, several properties of the current Java Card specification should be taken into account in addition to the particular features of the operating system being used.[16] These are listed and briefly described below.

[16] A good summary of the development of software for smart cards is provided by Zhiqun Chen [Chen 00]

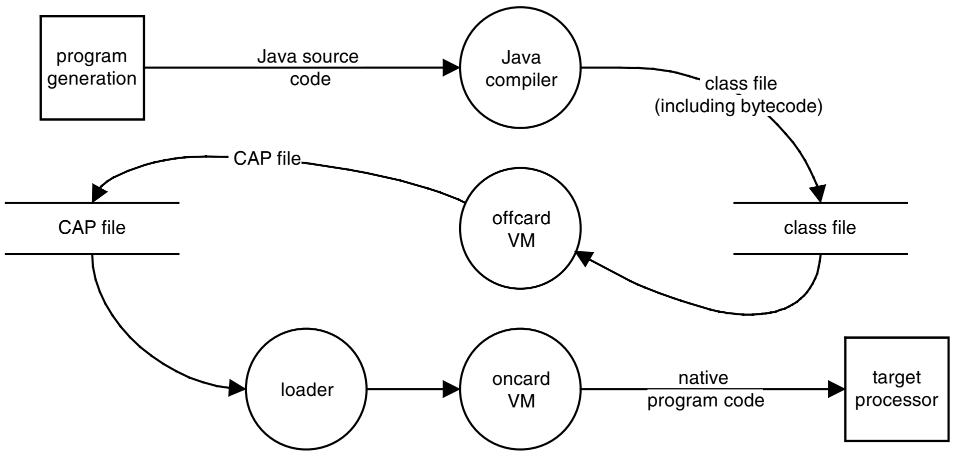


Figure 5.54 The usual program development process leading to execution of a program by the Java virtual machine in the smart card microcontroller

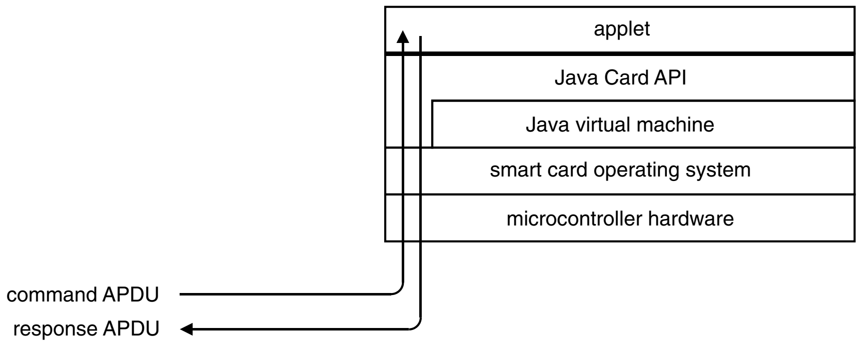


Figure 5.55 Data flow from the command APDU to the applet and the corresponding response APDU, with reference to the layer model of a Java smart card

Execution speed

Aside from its memory demands, the major point of criticism with regard to Java for smart cards is its low execution speed. However, it is relatively difficult to make fair comparisons between assembler programs and Java. This is primarily because it is not necessary to create exactly the same processes in Java as in assembler, as long as the program behaves the same way at the interface to the terminal. For example, a file system is not always necessary with a Java program, and probably nobody would ever program a cryptographic algorithm in Java.

Another general consideration is that the methods of the Java Card API should be used as much as possible, since they are in part coded in the native instructions of the target processor. This can yield a considerable increase in processing speed. When typical smart card commands are implemented in Java with intensive utilization of the interfaces to native library routines, it can be assumed that the execution time (excluding the data transmission time) will be approximately 50 % longer than for a comparable implementation in C or assembler. With unfavorable programming, a Java program can easily be a factor of 2 to 3 slower than a corresponding native program.

Application selection

Selecting a particular application in a Java Card amounts to selecting the corresponding applet by means of its unique AID. This selection starts the applet, so that it can perform any necessary initializations. After this, the applet automatically receives all command APDUs that are sent from the terminal to the smart card. If the applet is not selected, it remains inactive and is not involved in any data transfers.

Firewalls – keeping applications separate

From an IT perspective, individual applets in a smart card are completely isolated from each other. Any possible mutual interference is prevented by the security manager of the Java virtual machine. With Java Card, all applets of a package are automatically located inside the same security environment. Applets within an environment can mutually access objects. If it is necessary for applets to access common data (such as a PIN) across environment boundaries, the JCRE mechanisms provide secure access to objects belonging to a different environment. Naturally, the object that is to be accessed from within a different environment must explicitly allow such access.

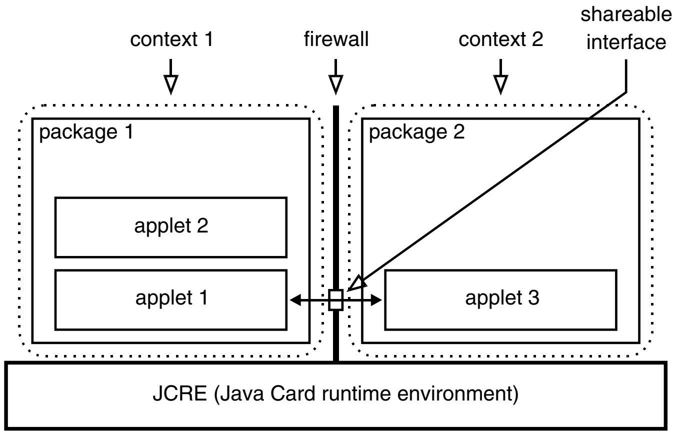


Figure 5.56 Schematic representation of the relationship between two packages with one or two applets and the firewalls of Java Card and their associated security context. By using the shareable interface, applets 1 and 3 can exchange data through the firewall

Transaction integrity – atomic operations

A sudden loss of power during a session must not be allowed to cause the data of an applet to assume undefined states. This is implicitly guaranteed by the virtual machine or operating system when an object is modified. However, if it is necessary to guarantee unconditional integrity across several objects or procedures, there are special mechanisms that can be used by the applet developer. These mechanisms can be used to ensure that the objects in question either retain their original states or properly assume new states.

File system

It is not mandatory for an applet to have its own file system. With some applications, it can be fully adequate to create file-independent data objects that can be accessed using either standard commands or commands defined by the programmer ('private-use' commands). The advantage of using applets without file systems is once again related to the ever-present underlying need to conserve memory usage in smart cards. In addition, the object-oriented nature of Java allows data objects to be accessed by calling objects according to their associated calling conditions. This makes it possible to implement accesses that meet very specific requirements of certain applications. For example, an application or user could be given the opportunity to allow another party to inherit their access privileges.

Nonetheless, common present-day applications for both smart cards and PCs show that data are very often stored and managed using file-oriented structures. Although this possibility is not excluded by the Java Card specification, it is unfortunately not directly supported. File management, for example using an ISO/IEC 7816-4 file structure, must be specifically implemented in Java using several classes. This is frequently necessary to support issuing Java smart cards that are compatible with previously developed applications having standard file trees.

Deleting objects – persistent and transient objects

All objects are inherently created as persistent objects in EEPROM when they are generated using the **new( )** method. Persistence refers to the ability of an object to exist past the end of a session, and is the opposite of transience. Persistent objects thus survive both the end of a session and a sudden loss of power, without losing data or consistency. Any object exists only as long as there is a reference that points to it. If this reference is deleted, the object is effectively no longer present, even though it still occupies memory. The only remedy for this would be a file manager with garbage collection, but this capability is not yet available in the current Java Card specification. (The next version of the Java Card specification, Version 2.2, will have provision for a garbage collector.)

Persistent objects can be converted into transient objects so that they can be stored in RAM, although this conversion cannot be reversed. Data contained in transient objects are lost at the end of the current session, and such objects are re-initialized to their default values the next time they are called.

Deleting applets

The Java Card specification provides a mechanism for deleting an applet in a smart card. The memory space occupied by the deleted applet can be made available for use by other applets by using a suitable technique, such as defragmentation.

Cryptographic algorithms

Many of the currently used cryptographic algorithms either involve modifying or swapping data at the bit level (such as DES) or utilize the arithmetic of long numbers (such as RSA).

Smart cards with Java are presently not suitable for programming such algorithms in Java, due to their low execution speeds and limited memory capacities (EEPROM and RAM).

Consequently, such cards usually contain the *javacardx.crypto* class, which provides an interface (API) to cryptographic algorithms implemented in native machine code. This allows cryptographic algorithms to be used from Java programs without having to accept significant speed penalties.

Cryptography and export restrictions

In many countries, export permits are required for smart cards with general-purpose operating systems and freely usable data encryption and decryption functions accessible via internal interfaces. This means that such cards cannot be exported at all to certain countries, and an exporter may have to wait several months for a suitable permit to be issued by the responsible agency.

Consequently, the classes for cryptographic functions are structured in Java Card such that they can be used directly for general data decryption and MAC computations but not for encryption. This is completely adequate for many applications, and in many countries it allows a 'simplified' export permit procedure to be used.

If a particular application requires data to be encrypted, the card manufacturer can incorporate the classes *....cryptoEnc.DES3_EncKey* and *....cryptoEnc.DES_EncKey*, which make encryption possible. From a purely cryptographic perspective, however, it is certainly possible to devise easily implemented procedures that can be freely used for encryption and decryption, without using these 'encryption' classes.

Memory space minimalization

In the near term, Java programs for smart cards will continue to be strongly influenced by the amount of available memory. This leads to certain compromises in programming that are not necessary for PCs. Table 5.31 lists a number of practical recommendations to help obtain the best possible match between Java programs and the special demands of the smart card environment.

Summary and future prospects

In spite of the unrelenting hype around Java, it should not be forgotten that it surely will not prove to be the solution to every informatics problem of the past and future years, and that it may not always be able to meet all of the expectations that have been invested in it. You only have to consider the fate of all of the previous programming languages that are no longer in fashion, such as Pascal ('modular'), Lisp ('AI for everyone'), C ('portable') and C++ ('reusable program code'). Although these languages have improved the state of software technology by orders of magnitude, many of their predicted benefits failed to materialize. Nevertheless, it is true that a new era in the history of smart cards began with the introduction of Java, since it is the first language that allows third parties to run executable program code in smart cards in a straightforward manner. Java has now become the standard programming language for smart card applications.

Table 5.31 Several useful programming guidelines for generating Java applets for smart cards, based on suggestions in [Schlumberger 97]. The objective of these guidelines is to make the best possible use of the severely limited memory capacities of smart cards

Guidelines for minimizing memory usage	Remarks
Use constants instead of variables	Constants need less memory space than variables and are thus to be preferred.
Use simple data types	Using the simplest possible data types saves memory.
Reuse variables	Reuse variables as much as possible to make the most economical use of the limited amount of memory in the smart card.
Avoid local variables	Local variables should be used as sparingly as possible.
Use a simple class hierarchy	A simple class hierarchy that is as flat as possible saves memory in the smart card.
Use few arguments	Methods should use no more arguments than are absolutely necessary.
Use a simple calling hierarchy	A simple calling hierarchy for the methods saves memory in the smart card.
Eliminate unused variables	The program should always be checked for unused constants, variables, methods and classes after it has been generated. The compiler may not always be able to remove such items during optimization.

A few trends in the future development of Java are clearly visible. Version 2.2 of the Java Card specification will augment the current specification with several useful functions, such as logical channels, garbage collection and remote method invocation (RMI). RMI can be used to directly invoke methods in a smart card from a terminal. The next version of the specification, Version 3, will be optimized for the processing capacity and memory sizes of new 32-bit smart card microcontrollers and will offer significantly more features than the current versions. It can safely be assumed that Version 3 will be closer to regular Java on PCs and the functionality of such Java systems.

By contrast, the large-scale loading and management of applets using applet management systems[17] must be regarded as a longer term possibility, due to its enormous complexity. However, it is probable that within the foreseeable future, card manufacturers will load applets and their associated data into Java-based smart cards as part of the normal completion process, with such applets remaining unchanged in the card until the end of its useful life.

Java Card is presently the only internationally used technology for realizing program-based applications in smart cards, and it will remain so within the foreseeable future. Within the various application areas, there are now a large number of specifications, APIs and extensions to Java Card, which provide a very broad basis for its utilization. In addition, the current degree of compatibility of Java cards produced by different manufacturers represents an attractive economic argument for card issuers, since it makes card manufacturers interchangeable. Java

[17] See also Section 10.5, 'Phase 4 of the Life Cycle in Detail'

Card has also made it possible for short development cycles, which are particularly demanded in the IT sector, to be achieved for smart cards, while allowing all well-known, proven life-cycle models[18] to be used. These reduced development and delivery times can open up new markets and applications for smart cards.

5.14.2 Multos

Multos is a multiapplication smart card operating system originating from the development of the Mondex system for electronic purses. Starting with this system, an operating system that is primarily optimized to meet the requirements of electronic payment systems was developed in several steps. The publisher of the specifications, license issuer and operator of the certification services for Multos is the Maosco Consortium [Maosco]. The majority of the Multos specifications are confidential, so here we can only present a summary of the features of this operating system. One interesting detail is that certain core operating system components of Multos are certified in accordance with ITSEC E6, which is the highest possible evaluation level.

Multos corresponds to a typical ISO/IEC 7816-4 compliant operating system and can interpret downloadable program code. Program code is typically developed in C and translated into the Multos executable language (MEL) using a special compiler. MEL is a hardware-independent program code that is executed by a stack-oriented virtual machine called the 'application abstraction machine' (AAM). From within MEL, an application can access the operating system services of the Multiapplication Operating System (MAOS) via various interfaces.

Before an application can be loaded into a Multos smart card, it must be digitally signed by a licensed Multos certification service using relatively elaborate mechanisms. As is usual with payment cards, large portions of the completion process are also specified in detail.

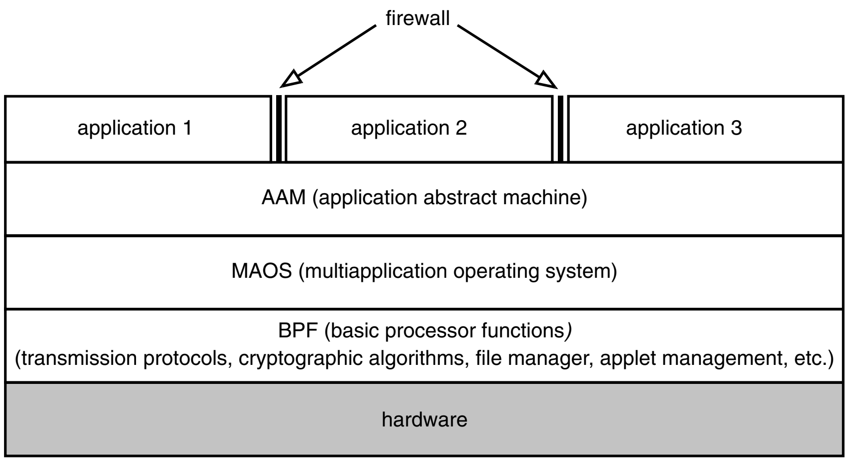

firewall
application 1
application 2
application 3
AAM (application abstract machine)
MAOS (multiapplication operating system)
BPF (basic processor functions)
(transmission protocols, cryptographic algorithms, file manager, applet management, etc.)
hardware

Figure 5.57 Schematic representation of the basic architecture of the Multos smart card operating system. This example includes three applications based on Multos that have been generated in MEL

[18] See also Section 15.7, 'Life-Cycle Models'

5.14.3 Basic Card

Since 1996, a smart card operating system with an interpreter for the Basic programming language has been available from the German company Zeitcontrol [Zeitcontrol]. This operating system is called Basic Card, and it is available in various versions with different features and for hardware platforms with various memory sizes. Besides Java Card and Multos, it is one of the few multiapplication operating systems to allow executable program code to be downloaded by third parties.

The procedure for generating downloadable programs for Basic Card is based on traditional Basic interpreters. A compiler translates the source code into P-code, which is transferred to a memory region of the smart card microcontroller reserved for this purpose using a special loader program. After this, the program code stored in this region can be processed by the interpreter as necessary.

With regard to data types, control structures and functions, the version of Basic supported by Basic Card corresponds to the presently common simpler dialects of this programming language, which has existed for several decades. It has also been extended to include some functions specific to smart cards, such as an interface to the smart card file system. In addition, there is support for the $T = 0$ and $T = 1$ data transmission protocols for contact-type cards, and for contactless data transmission in accordance with the ISO/IEC 14 443 B standard. For typical security applications, a variety of cryptographic algorithms can be called via an interface (including DES, triple DES, AES, RSA with a key length of 1024 bits, elliptic curves with a key length of 160 bits and the SHA-1 hash algorithm).

Compared with other smart card operating systems with interpreters, the program code is very compact and the execution speed is relatively high. These two aspects are primarily due to the facts that in procedural terms, Basic can be easily and quickly interpreted, and that no sophisticated security mechanisms are used for preparing applications. Nevertheless, for certain applications that require programs in smart cards to be developed quickly and easily, Basic Card can certainly represent an alternative to other smart card operating systems.

The main unconventional aspect of Basic Card is the fact that it is the product of a relatively small company, which has continued to further develop it over the course of many years, rather than one of the giants of the IT industry, as some of the other smart card operating systems with interpreters.

5.14.4 Windows for Smart Cards

In 1998, Microsoft announced Windows for Smart Cards (WSC) as a version of Windows for smart cards. This smart card operating system was intended to increase the bandwidth of the Microsoft operating system and was doubtless intended to be an alternative to Java Card, which was not yet a widely used operating system. After several years of development and several alpha, beta and final versions of the operating system, accompanied by a rather large promotional effort in terms of the modest smart card market, Windows for Smart Cards was cancelled in mid-May and the source code was offered for sale to several companies. Due to a lack of acceptance by the smart card industry, which is very demanding with regard to reliability and security, WSC never achieved any significant market success during this period.

Windows for Smart Cards was a smart card operating system designed to be used with multiple applications, and it also supported downloadable program code. Many parts of the bytecode supported by the virtual machine (VM) were very similar to the machine code of 8051-type processors, which had a positive effect on execution speed. Loading programs into the memory of the smart card was protected using the usual cryptographic mechanisms.

Downloaded programs could access various operating system functions, such as cryptographic algorithms, data transmission and file management, via several APIs. There was also a GSM 11.14 API in order to allow compliant value-added services for SIMs to be developed using the SIM Application Toolkit.[19] Such services could be developed using Microsoft's standard, powerful development environments for Basic and C. All that was necessary was to link in a few library components and place suitable compiler directives in the code.

Windows for Smart Cards was a complete operating system, which means that it also included a complete file management system with rule-based access mechanisms. File management was based on ISO/IEC 7816-4 and ISO/IEC 7816-9, thus providing a large measure of compatibility with current standards. The file management system had one unusual feature, which is that it was the first file management system for smart card operating system to use a file allocation table (FAT).[20]

Figure 5.58 shows the basic functionality of Windows for Smart Cards in the form of a data flow diagram. The programs generated using a development environment and loaded into the smart card were called 'runtime environment' (RTE) applications. The operating system was informed of the presence of an RTE application via a dispatch table to allow the corresponding program code to be executed by the VM as necessary. Another option was to permanently integrate certain applications or commands in the ROM code. Such items were called 'non-RTE applications'.

A 'file system builder' (FS builder) could be used to create the files needed by a particular application, along with their access conditions. These files could then be loaded into the smart card as well, where they could be accessed by an RTE application.

5.14.5 Linux

Since the end of the 1990s, the open-source operating system Linux has altered large portions of the software industry. Up to now, the focus of Linux has been computers in the PC area, which have much higher performance than smart cards. However, for some time there have been efforts to establish Linux in the area of typical microcontroller applications. Up to now, this has presupposed a level of performance that typically can only be provided by 32-bit processors, along with memory demands on the order of several kilobytes of ROM and several tens of kilobytes of RAM. Current smart card microcontrollers cannot yet meet these demands. However, it is certainly conceivable that the versions of Linux available up to now could have their hardware requirements further reduced. At the same time, the performance of smart card microcontrollers increases with every new generation, so it would be possible for Linux to be available for smart cards in the not too distant future.

[19] See also Section 13.2.4, 'The SIM'
[20] See also Section 5.7, 'File Management'

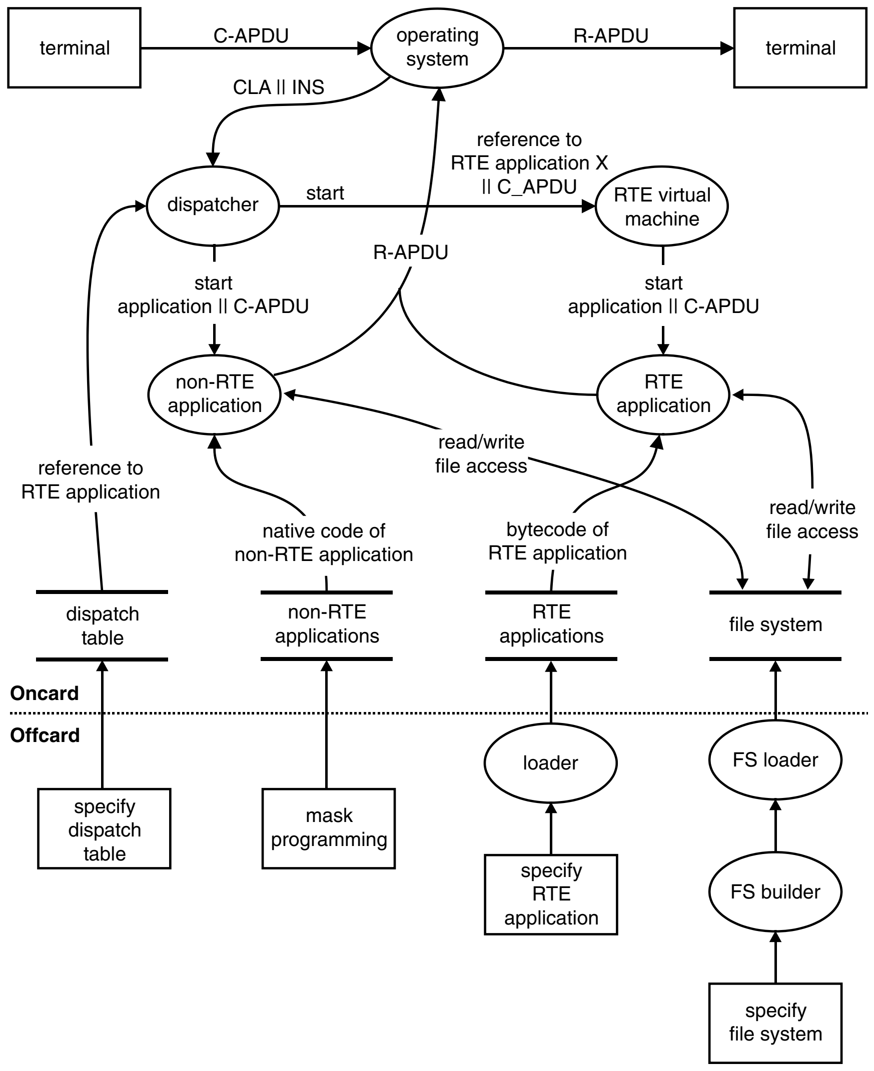


Figure 5.58 Data flow diagram for the essential components of Windows for Smart Cards. This diagram shows the fundamental IT relationships between these components. It has been produced based on published descriptions and makes no claim to being complete

Besides Linux, it is naturally possible for another open-source operating system for smart cards to appear. The only important consideration is that it can be used without licensing, since the high licensing fees resulting from the large quantities in which smart cards are produced and used represent one of the most significant barriers to the use of standard operating systems in smart cards.

5.15 THE SMALL-OS SMART CARD OPERATING SYSTEM

The features and basic working principles of smart card operating systems are summarized in the previous sections of this chapter. For those readers who wish to immerse themselves more deeply into this subject, this section provides a detailed description of the internal interrelationships of a complete operating system. This is a classical smart card operating system according to the ISO/IEC 7816-4 or GSM 11.11 standard.

The name of the operating system described in this section is ‘Small-OS’, which reflects its very small memory demands and the fact that it can be run on hardware platforms that are not particularly powerful. It is written in a pseudocode resembling Basic. The pseudocode shown here cannot be compiled as is, since some assignments have not been fully decoded to the last bit and are explained only verbally. We do not wish to present page after page of true program code in a language such as C or C++, which can be incomprehensible and boring to read. Instead, our objective is to present a graphic example that illustrates the subject.

The ready comprehensibility of pseudocode greatly outweighs the advantages of using a real programming language, which are that it can be directly compiled and executed. With pseudocode, you do not get lost in the details of the implementation, and you can instead completely concentrate on the fundamental processes. The system presented here is platform-independent and is not tailored to any particular hardware. There is an actual implementation of Small-OS, but it is not programmed in assembler to run on a smart card microcontroller. Instead, it runs as a simulation in the program *The Smart Card Simulator*. This program is available at no charge under a GPL via the Internet [Rankl].[21]

Programming in pseudocode

First we would like to make a few remarks about programming style and the programming of Small-OS. The pseudocode, which is derived from Basic, in principle represents a semi-formal description of the Small-OS smart card operating system. Similar operating system characterizations are often used for software evaluations according to the ITSEC. They are used as the basis for the evaluation and for checking the source code. The pseudocode that is presented in this section thus represents a good example of how formalized processes within a smart card operating system are portrayed. The pseudocode is listed here in tables with extensive comments. Similar forms of presentation can be found in the EN 1546 series of standards, for example, in which the internal processes of smart cards for electronic purses are semi-formally described.

The individual terms used in the pseudocode are described at the front of the book.[22] The program code is based on the standard dialects of Basic, with object-oriented extensions. Only generally understandable constructs are used. All labels, constants and references are in English. Numerical values are usually given in hexadecimal form using ISO notation (such as '42'). However, decimal or binary forms are used where necessary to aid understanding, using

[21] Due to time constraints and a lack of demand, it has unfortunately not been possible to update the Smart Card Simulator. Consequently, its current state of development is incomplete, and many details no longer correspond to the current versions of the standards

[22] See ‘Program Code Conventions’ at the front of the book

Table 5.32 Summary of the features of Small-OS

Name: Typical application areas: Hardware prerequisites:	Small-OS Multiapplication with no user-generated program code ROM: ≈8 kB, EEPROM: ≈1 kB, RAM: ≈128 bytes
Instruction set:	Profile N per ISO/IEC 7816-4, with extensions Commands: SELECT FILE, READ BINARY, UPDATE BINARY, READ RECORD, UPDATE RECORD, VERIFY, INTERNAL AUTHENTICATE
Data transmission:	Transmission protocol: T = 1 • divider (CRCF) set to a fixed value of 372 (standard ISO/IEC 7816-3 setting) • PPS not supported • secure messaging not supported
File system:	One DF level • structures for working EFs: transparent, linear fixed • structure for internal EFs: linear variable (for PINs and keys) • one EF key allowed per directory (i.e. per MF or DF) • no dynamic file system (no file deletion or creation, no free memory management) • maximum size of a transparent EF: 255 bytes
State machine:	Independent secure states for the MF and DFs • secure states: 256 (0–255) • initial state = 0 • only one allowed input state for using a PIN or a key • only one allowed input state for file access (i.e. '<' and '≥' comparisons are not possible) • no cross-level key access (EF key is always selected via the currently selected directory (MF or DF)) • PIN addressing: 2 PINs maximum (ref. no. 1 and 2) • key addressing: 31 keys maximum (ref. no. 1–31) • retry counters for PINs and keys allow a maximum of 15 unsuccessful attempts
Cryptographic algorithm:	DES
Program code:	Generation and loading by external parties is not possible

a notation that is derived from the ISO notation. For example, all countable values, such as length specifications, are shown in decimal form.

Nobody would program a smart card operating system in this form in assembler or C, since it would be far too complicated. One of the design objectives with Small-OS was to create a simple yet powerful smart card operating system in a manner that is most easily understood and well commented. Intentionally, no attempt was made to minimize program execution time, program code, RAM usage or stack depth, since doing so would seriously impair the readability of the code. In real smart card operating system programming, for example, it is sometimes common practice to use a JUMP instruction instead of a CALL instruction for calls to rarely used subroutines, since this saves two bytes of expensive stack space. A flag that is set before the subroutine is called with the JUMP instruction and is used to determine the return address when the subroutine process is completed. This sort of optimization is not found in Small-OS, for the reason just given. The pseudocode has been optimized only with regard to readability and comprehensibility. The resulting deviations from real smart card operating systems are identified wherever they occur, either in the text or in the commented pseudocode.

The majority of a smart card operating system is located in ROM, due to the notorious shortage of memory in smart card microcontrollers. It thus cannot be modified after the chip has been manufactured. Software cannot be produced with absolutely no errors (except for trivial miniprograms), but only with as few errors as possible. An error in ROM program code would have serious consequences. To make it possible to use bug fixes to patch such errors, jumps to EEPROM are provided at critical locations in the ROM. This technique is very old in software engineering and is not specific to smart cards. Nearly identical mechanisms are used in the MacOS and OS/9, for example. Locations in EEPROM that are called by the program code, called ‘handles’, are used for this. A handle normally contains only a RETURN instruction, which causes an immediate return to the calling code. If a bug fix for the ROM code is necessary, corrective code is inserted in the EEPROM at a handle location. In this case, the call to the handle does not produce an immediate return. This mechanism is not included in Small-OS, for reasons of simplicity and understandability.

The relatively detailed description of the smart card operating system provides an interesting opportunity to follow several typical types of attacks directly in pseudocode. At suitable locations, possible attacks and defensive measures at the operating system level are described in detail. For example, it is possible to examine an attack on the PIN by comparing processing times, which has now become a ‘classic’ form of attack, in all of its details in the pseudocode. A comprehensive listing of typical attacks and their defenses is given in Chapter 8, ‘Security Techniques’.

Design criteria

The above considerations led to the following design criteria that guided the conception and programming of Small-OS. Small-OS should be a simple smart card operating system that does not need a lot of program code and has a low level of complexity, just like the real models that inspired it. This makes its structure comprehensible and easy to grasp. It is built up in a strictly modular fashion, which means that it can be directly extended with additional commands at a reasonable effort. The file system and the supported commands are without exception compliant with the international ISO/IEC 7816-4 standard. The ‘N’ profile of ISO/IEC 7816-4 was chosen as an option, with some extensions that are commonly present in the smart card world.

Small-OS is thus intended to be used in situations in which it is not necessary to download applications after the cards have been issued. However, depending on the amount of available

Table 5.33 Small-OS design criteria in order of priority

Priority	Criterion	Reason
1	Compatibility with ISO/IEC 7816-4	• the international standard for smart card operating systems
2	Robustness	• high level of reliability • high level of error tolerance
3	Low level of complexity	• high level of reliability • easily understood operation
4	Modular construction	• high level of reliability • high level of error tolerance • ease of extension
5	Small memory demand	• an absolute must for smart card operating systems
6	Multiapplication operating system	• currently standard capability
7	No program code downloading	• high level of reliability • low level of complexity
8	Similar to real smart card operating systems	• with regard to the real world, Small-OS is an instructive example

memory, several different applications could be run in the smart card independently of each other. This means that Small-OS is a multiapplication operating system. However, it is not possible to download program code to the card and then execute it in the card. In summary, Small-OS is comparable to the first general smart card operating systems, such as are still used in various forms for GSM applications.

The ISO/IEC 7816-4 standard describes a basic file system and several fundamental commands for smart cards, among other things. In this way, it primarily characterizes the interface of the smart card, rather than the internal architecture of the operating system. In addition, there are numerous options and a few passages that unfortunately are subject to interpretation. The number of possible variations that are thus made possible must be sharply reduced in practice by specifications, such as the GSM 11.11 specification, to ensure compatibility between different implementations.

With actual smart card operating systems, therefore, the designation 'ISO/IEC 7816-4 compatible' by no means signifies that they behave exactly the same way in all respects. This would need a detailed specification, which would again in part be an individual interpretation of the ISO/IEC 7816-4 standard. Consequently, in practice the behavior is usually the same only if the command is successfully executed, while there are various differences in case of an error. With Small-OS, interpretations of the ISO/IEC 7816-4 standard are usually identified as such in the pseudocode. Within the limits of the interpretation of the standard, Small-OS is truly compliant with the ISO/IEC 7816-4 standard and corresponds to the normal interpretations of the standard in the smart card industry as much as possible.

Major differences between operating systems are frequently to be found in regard to return codes. Since the usage and priority of the individual return codes are not described in detail in ISO/IEC 7816-4, assumptions must be made. Small-OS, in contrast to all other operating systems, at least has the advantage that the code is public. This means that it is always possible to determine exactly which return code is generated.

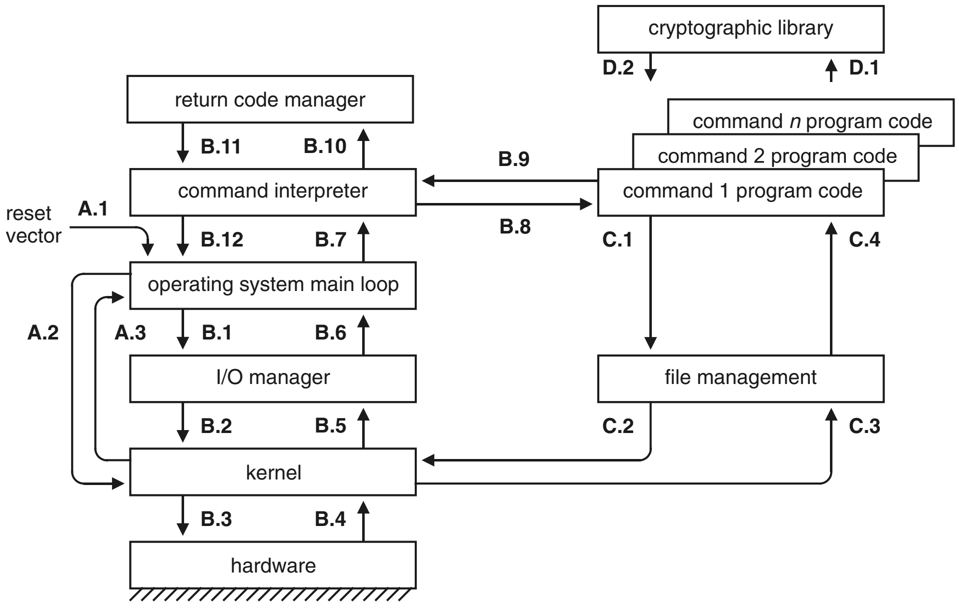


Figure 5.59 The layer structure of Small-OS and the resulting calling scheme. Calls labeled 'A' are used when the operating system starts up, those labeled 'B' are used to invoke commands and those labeled 'C' are used as necessary within commands to access the file system. Calls labeled 'D' are used with cryptographic algorithms. The numbers indicate the sequence of the program calls

If one were to implement Small-OS for a smart card microcontroller, it would need around 5–6 kB of ROM, 128 bytes of RAM and at least 1 kB of EEPROM, depending on the number of applications present (using an 8051 processor). This assumes as general conditions that only the T = 1 data transmission protocol is used and that DES is used as the cryptographic algorithm. If certain applications were to need more memory, a microcontroller with more EEPROM could be used without any problems. This would not affect the operating system or require any modifications.

File access

The file system described in the ISO/IEC 7816-4 standard allows an enormous number of options with regard to access conditions and key management. For Small-OS, therefore, a solution that is often used in practice for multiapplication operating systems has been chosen. The file headers of working EFs contain prescribed states for each of the various access commands (for example, the condition … *.AccessCondition.Read* for the READ command). Each state is described by a positive integer. This means that each EF has independent state values for read and write accesses in its header. These values must be achieved within the current directory before the command can be executed. State 0 represents the base state (idle), so that a state condition of 0 means that all accesses are allowed.

State variables are always assigned to the MF and the currently selected DF (*Security State.MF* and *SecurityState.DF*). These can be altered by commands relating to security (VERIFY and INTERNAL AUTHENTICATE). In case of an access to a specific EF, the current state in the directory is compared with the required state in the EF file header. If the actual and required states are the same, then the file may be accessed for writing or reading.

Real smart card operating systems often allow a large variety of less-than, greater-than, greater-than-or-equal and not-equal comparisons to be made here. It is also frequently possible to define several possible access states independently in the file header. This further complicates a process that is already not exactly simple, and for this reason it is not used in Small-OS. In principle, however, Small-OS could be extended to allow such comparisons without any structural modifications.

Access to internal secrets (PINs and keys)

All PINs and keys are held in special internal EFs. Such EFs are called EF Key here. This type of file can be read or written only by the operating system itself. External selection or access is not possible. There are no mechanisms at all in the design that would allow external access to such EFs. This is a part of the security philosophy of the Small-OS smart card operating system.

Only one EF Key file can be created for each directory. It automatically has a linear variable structure, so that it can store PINs and keys of various lengths in a minimum amount of memory. Each record in an EF Key file contains either a PIN or a key, with an address number that is unique within the file (*....KeyNo*). For each secret object (PIN or key), a state value is stored. This defines the state (*....EntryState*) that is required for a command to be used (VERIFY or INTERNAL AUTHENTICATE). Following the execution of a command, the result state (*....ResultState.OK* or *....ResultState.NOK*) is set in the directory to which the EF belongs according to the result achieved (such as PIN comparison successful or not successful). In addition, a retry counter (*....RCntr*) is assigned to each secret object. This counter is incremented for each unsuccessful result until it reaches its maximum value. If the retry counter has reached its maximum value (*....RCntrMaxValue*), the associated secret object can no longer be used.

Some smart card operating systems allow 'cross-level' access to keys. This allows keys stored in the next higher directory to be accessed from within a particular DF. The mechanisms

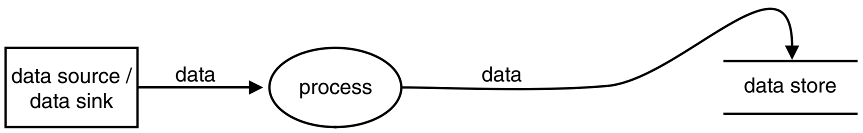


Figure 5.60 The symbols used for the data flow diagrams for Small-OS. Data sources and data sinks, which are also called *terminators*, represent objects outside the system under consideration that exchange data with the system. A *process* is located inside the system under consideration. It processes input data streams and generates output data streams. A *data store* is a storage place for data that can be read and written. A detailed introduction to this type of system analysis is contained in Robertson [Robertson 96]

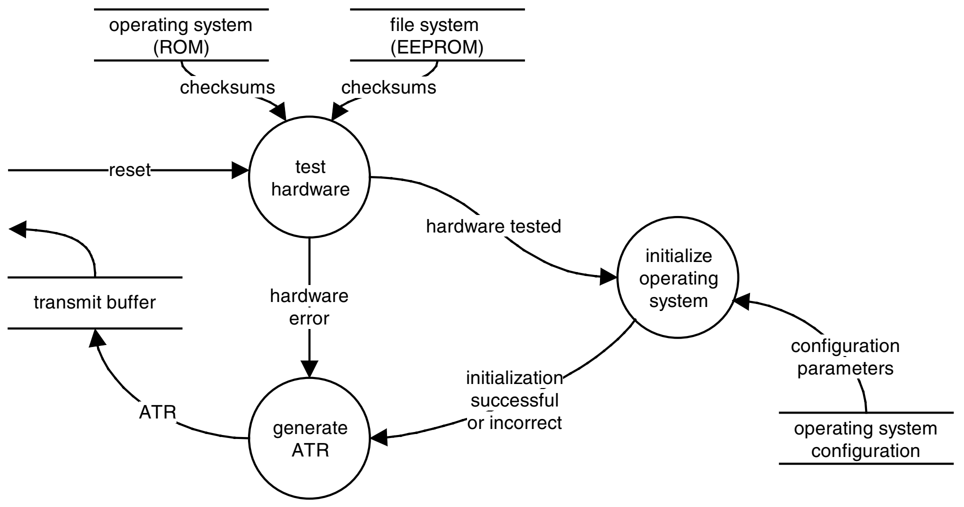


Figure 5.61 Small-OS: data flow diagram of the startup and test processes of the operating system

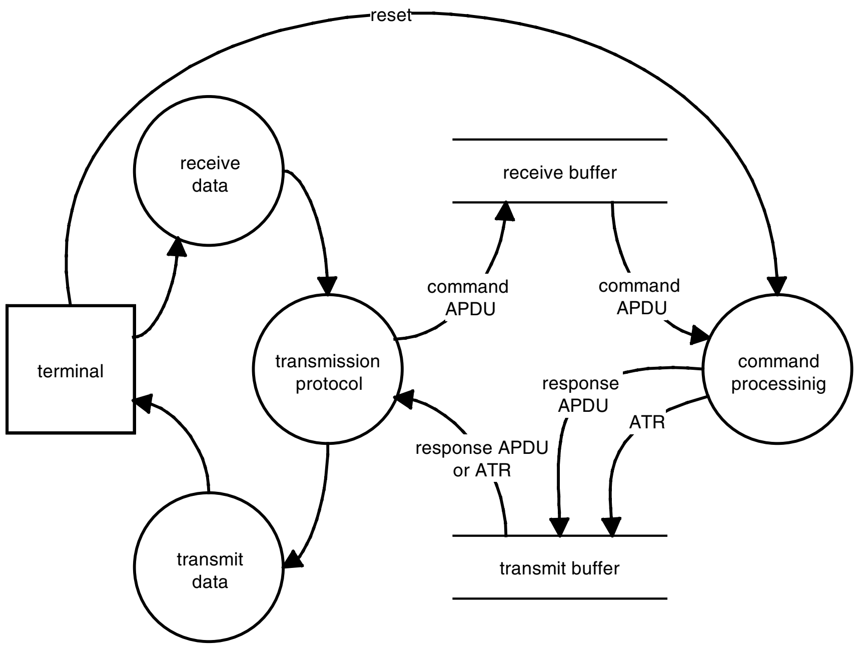


Figure 5.62 Small-OS: data flow diagram for reset and data transmission

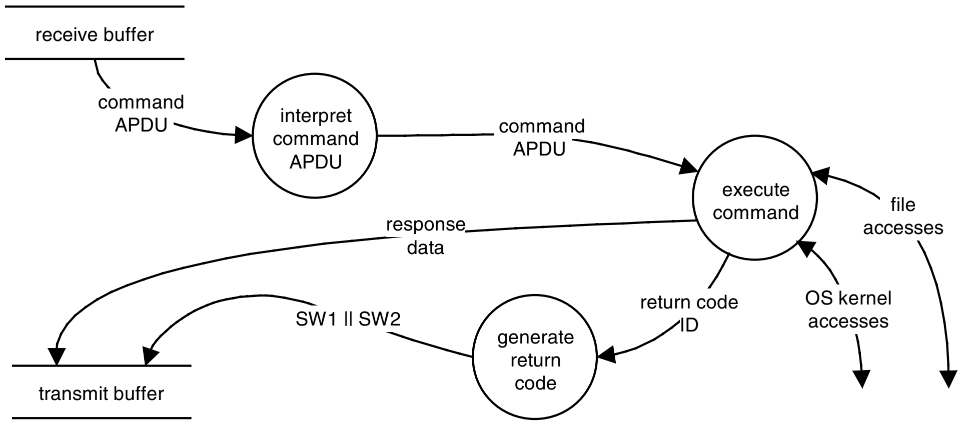


Figure 5.63 Small-OS: internal data flow diagram of the process for command processing

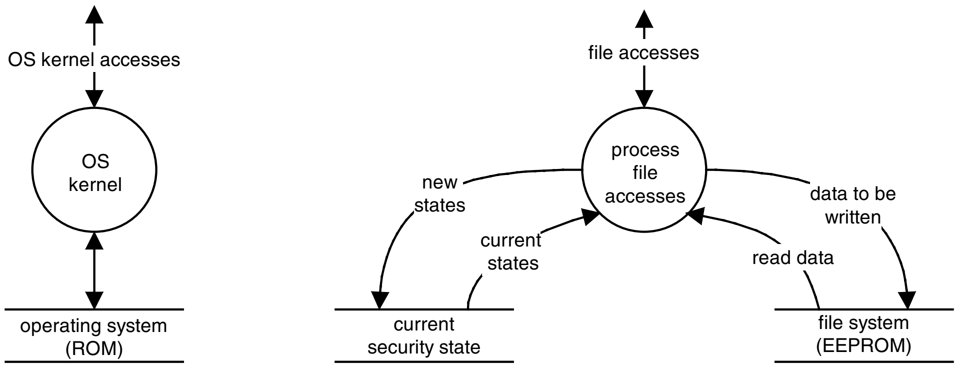


Figure 5.64 Small-OS: internal data flow diagram of the process for file and OS kernel accesses

needed for this are not included in Small-OS, since the functionality would not justify the amount of code needed to implement them. Key accesses via aliases are also not implemented, for the same reason.

Small-OS constants

Basically, constants are used in the pseudocode as much as is reasonable for numerical and non-numerical values. This considerably increases the readability of the code, and it is purely and simply good programming style. All constants are identified by the prefix C_. The values of these constants normally depend on the target hardware or the implementation, so they are not further defined here. The constants for the 2-byte return codes all have the prefix C_RC_. Table 5.21 lists the associated return codes. In practice, the constants of an operating system are usually stored in unalterable form in the ROM.

Table 5.34 Constants used in Small-OS (excluding return code constants)

Constant	Meaning
C_Error	Constant for general errors (e.g. when calling a subroutine)
C_InvalidPointer	Constant for an invalid pointer value
C_Equal	Constant for comparisons OR compared objects are equal
C_NotEqual	Constant for comparisons OR compared objects are not equal
C_Found	Constant for search functions OR sought object found
C_NotFound	Constant for search functions OR sought object not found
C_AccessDenied	Constant for access conditions, access denied
C_AccessAllowed	Constant for access conditions, access allowed
C_WriteError	Constant for a data write error (e.g. writing to EEPROM)
C_EFTypeWorking	Constant for a working EF
C_EFTypeInternal	Constant for an internal EF
C_EFStrucLinFix	Constant for a linear fixed EF
C_EFStrucTransp	Constant for a transparent EF
C_CmdVERIFY	Constant for the VERIFY command
C_CmdINTAUTH	Constant for the INTERNAL AUTHENTICATE command

Small-OS variables

The variables in Small-OS can basically be divided into RAM and EEPROM variables. The RAM variables are re-initialized each time the smart card is reset and retain their values only for the duration of one session. However, data can be stored in RAM variables without any loss of time, and the number of write cycles is unlimited. EEPROM variables, by contrast, are typically used primarily for the implementation of the file manager, for which data contents and access conditions must exist between sessions as well.

Small-OS RAM variables The regions for the transmit and receive buffers take up the majority of the RAM variables. All data elements of an APDU can be addressed via their own specific variables. In order to manage with the small amount of available RAM, the transmit and receive buffers are in practice sometimes made partially overlapping. Some commands must then learn from the operating system exception handler that data can be written to the transmit buffer only after all the data in the receive buffer has been processed. In order to promote understandability, no such memory minimization is used here, and the transmit and receive buffers are thus completely separate from each other.

The second large group of RAM variables relates to the management of the file tree. Several pointers are provided, which point to the current directory (*....Ptr.CurrentDF*), the current file (*....Ptr.CurrentWEF*) and the current valid key file (*....Ptr.CurrentIEF.Key*). With record-oriented EFs (linear fixed EFs), the currently selected record is also assigned a pointer (*....Ptr.CurrentRecord*). All pointers are identified by the prefix *Ptr* and are explicitly set to the value C_InvalidPointer whenever they are not allowed to be used.

Table 5.35 Return code constants used in Small-OS

Constant	Content	Meaning of the return code
Process completed – warning processing		
C_RC_CounterX	'63Cx'	A counter (usually the PIN retry counter) has been incremented and now has the value 'x'
Process aborted – execution error		
C_RC_MemoryFailure	'6581'	EEPROM write error
Process aborted – checking error		
C_RC_WrongLength	'6700'	Incorrect length
C_RC_CmdIncompFStruc	'6981'	Command incompatible with file structure
C_RC_SecStateNotSatisfied	'6982'	Security state not satisfied
C_RC_AuthMethodBlocked	'6983'	Authentication method blocked
C_RC_CondOfUseNotSatified	'6985'	The conditions for using the data element (PIN or key) are not satisfied
C_RC_CmdNotAllowed	'6986'	Command not allowed
C_RC_FctNotSupported	'6A81'	Function not supported
C_RC_FileNotFound	'6A82'	File not found
C_RC_RecordNotFound	'6A83'	Record not found
C_RC_LcInconsistentP1P2	'6A87'	Lc inconsistent with P1 or P2
C_RC_RefDataNotFound	'6A88'	Data referenced in command not found
C_RC_WrongP1P2	'6B00'	P1 or P2 incorrect
C_RC_INSNotSupported	'6D00'	Command not supported
C_RC_CLANotSupported	'6E00'	Class not supported
C_RC_FatalError	'6F00'	Internal error with no further description
Process completed – normal processing		
C_RC_OK	'9000'	Command successfully executed

The two variables *SecurityState.MF* and *SecurityState.DF* identify the security state in the MF and the currently selected DF, respectively. The security states that files have achieved are stored in these variables as positive integers. Here the value '0' indicates that no security state has been achieved. This value is automatically set when Small-OS is initialized after a reset.

Additional RAM memory is needed for the program stack, the DES cryptographic algorithm and working registers. Here it is assumed that these are implicit, so no special variables are assigned for them.

Small-OS EEPROM variables For the sake of simplicity, the file tree in the smart card is implemented as a multidimensional array. This approach would be much too memory-intensive for use in a real smart card operating system. The basic structures that are usually used in real systems for file management are one-way linked lists. The lengths of the list elements can be kept variable with the help of TLV encoding. This minimizes the amount of memory needed for file management, since only the necessary data elements have to be stored in memory. The size of the file headers for DFs and EFs, with all the information necessary for these files, is normally between 20 and 40 bytes. However, the multi-dimensional array file structure used in Small-OS can without question provide a simpler and more efficient solution when the operating system is implemented in a high-level language and runs in a hardware environment with relatively few memory limitations. A nearly identical construction is used in the 'Smart Card Simulator' program, for example.

Table 5.36 Variables used in Small-OS for data transmissions to and from the smart card. The listed variables are typically held in RAM

Variable	Description
.1 8	Bitwise reading or writing of a variable (for example, *APDU.Cmd.CLA.1* corresponds to bit 1 of the class byte in the command APDU)
APDU.Cmd	Command APDU received from the smart card
APDU.Cmd.CLA	Class byte of the command APDU
APDU.Cmd.INS	Command byte of the command APDU
APDU.Cmd.P1	Parameter 1 of the command APDU
APDU.Cmd.P2	Parameter 2 of the command APDU
APDU.Cmd.Lc	Long command byte of the command APDU (optional)
APDU.Cmd.Data[. . .]	Data portion of the command APDU with length 1 . . . n (optional)
APDU.Cmd.Le	Expected length of the response APDU (optional)
APDU.Rsp	Response APDU to be sent by the smart card
APDU.Rsp.Data[. . .]	Data portion of the response APDU with length 1 . . . *n* (optional)
APDU.Rsp.SW1	Status word 1 of the response APDU (byte 1 of the return code)
APDU.Rsp.SW2	Status word 2 of the response APDU (byte 2 of the return code)
Returncode	*Return code* := *APDU.Rsp.SW1* || *APDU.Rsp.SW2*

Table 5.37 Variables used in Small-OS for file management and file access. The listed variables are typically held in RAM

Variable	Description
Ptr.MF	Pointer in the smart card operating system; always points to the MF
Ptr.CurrentDF	Pointer in the smart card operating system; points to the currently selected DF or to the MF
Ptr.CurrentIEF.Key	Pointer in the smart card operating system; points to the current internal EF that holds the key for the currently selected DF (EF Key)
Ptr.CurrentWEF	Pointer in the smart card operating system; points to the currently selected working EF in the currently selected DF or the MF
Ptr.CurrentRecord	Pointer in the smart card operating system; points to the current record in a record-oriented EF. This pointer is not valid if a transparent file is selected
SecurityState.MF	Achieved security state of the MF
SecurityState.DF	Achieved security state of the currently selected DF

Data structures

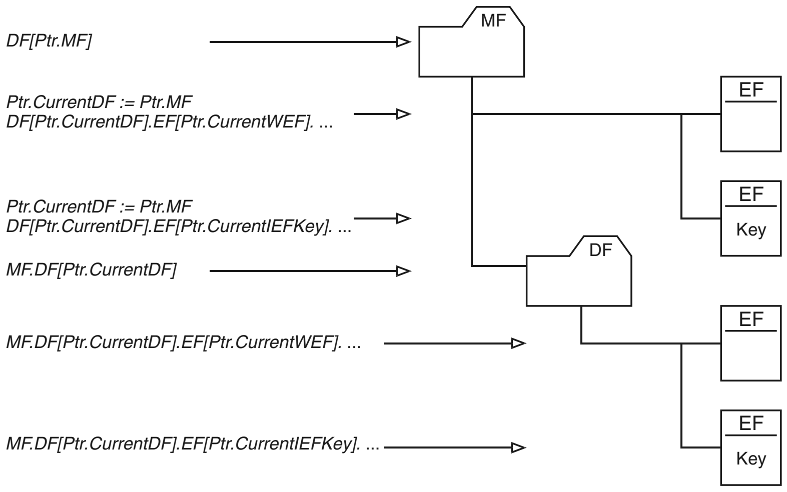

DF[Ptr.MF]
Ptr.CurrentDF := Ptr.MF
DF[Ptr.CurrentDF].EF[Ptr.CurrentWEF]. ...
Ptr.CurrentDF := Ptr.MF
DF[Ptr.CurrentDF].EF[Ptr.CurrentIEFKey]. ...
MF.DF[Ptr.CurrentDF]
MF.DF[Ptr.CurrentDF].EF[Ptr.CurrentWEF]. ...
MF.DF[Ptr.CurrentDF].EF[Ptr.CurrentIEFKey]. ...
MF
DF
EF
EF
Key
EF
EF
Key

Figure 5.65 Example of the addressing of directories (MF and DF) and files (EFs) within the data structures available in Small-OS

Table 5.38 General data structures used in Small-OS for file management. The listed variables are typically held in EEPROM

Variable	Description, contents and size
Data structures for the MF and the DFs	
DF[. . .].FID	File identifier length: 2 bytes; content: 0 . . . 255 each element
DF[. . .].DFName	DF name (includes the application identifier) length: 1 . . . 16 bytes; content: 0 . . . 255 each element
DF[. . .].LenDFName	Length of the DF name length: 1 byte; content: integer $\in \{1 \ldots 16\}$
Data structures for EFs	
DF[. . .].EF[. . .]. . . .	File tree in the form of a 2-dimensional matrix
DF[. . .].EF[. . .].FID	File identifier length: 2 bytes; content: 0 . . . 255 each element
DF[. . .].EF[. . .].Structure	EF structure content: property (transparent / linear fixed)
DF[. . .].EF[. . .].Type	EF type content: property (working / internal)
DF[. . .].EF[. . .].AccessCondition.Read	Access condition for which the READ command is allowed for the EF content: 0 . . . 255
DF[. . .].EF[. . .].AccessCondition.Update	Access condition for which the UPDATE command is allowed for the EF content: 0 . . . 255

Table 5.39 Data structures used in Small-OS for managing working EFs with transparent and linear fixed structures. The listed variables are typically held in EEPROM

Variable	Description, contents and size
Data structures for EFs with transparent structures	
DF[...].EF[...].TransparentData[1 ... n]	Data content of a transparent file length: n (= ... .*TransparentDataSize*) content: 0 ... 255 each element
DF[...].EF[...].TransparentDataSize	Data size of a transparent file content: 0 ... 255
Data structures for EFs with linear fixed structures	
DF[...].EF[...].Record[...].Data[1 ... n]	Data content of a single record of a linear fixed file length: 1 ... n (n =. ... *Size*) content: 0 ... 255 each element
DF[...].EF[...].Record[...].Size	Length of a single record of a linear fixed file (same for all records in a linear fixed file) content: 1 ... 255
DF[...].EF[...].NoOfRecords	Number of records in a linear fixed file content: 1 ... 255

Table 5.40 Data structures used in Small-OS for managing internal EFs with linear variable structure, which are used to store PINs and keys. The listed variables are typically held in EEPROM

Variable	Description, contents and size
Data structures for EFs for storing keys	
DF[...].EF[...].Record[x].KeyData	PIN or key in a linear variable file length: n (= ... .*KeySize*) content: 0 ... 255 each element
DF[...].EF[...].Record[x].KeySize	Length of the PIN or key in bytes
DF[...].EF[...].Record[x].KeyNo	Number of the PIN or key content for a PIN: 1; 2 content for a key: 1 ... 31 each element
DF[...].EF[...].Record[x].RCntr	Retry counter for a PIN or key content: 0 ... 15
DF[...].EF[...].Record[x].RCntrMaxValue	Maximum retry counter value for a PIN or key content: 1 ... 15
DF[...].EF[...].Record[x].KeyPurpose	Utilization of the EF Key records (VERIFY or INTERNAL AUTHENTICATE)
DF[...].EF[...].Record[x].EntryState	Necessary state for using a PIN or key content: 0 ... 255
DF[...].EF[...].Record[x].ResultState.OK	State following successful use of a PIN or key content: 0 ... 255
DF[...].EF[...].Record[x].ResultState.NOK	State following unsuccessful use of a PIN or key content: 0 ... 255

The main loop and the initialization of the hardware and operating system

After the smart card is reset, the program counter is loaded with the address of the reset vector, and the processor then starts to execute the first program code. The hardware is tested right away

Listing 5.2 Small-OS program code: operating system initialization and the main loop

```
Resetvector:                                  Entry point following a CPU reset.
  CALL Initialize_Hardware
  CALL Initialize_Operating_System
  CALL IO_Manager_Send_ATR

Main_Loop:                                    Main loop of the smart card operating system.
  CALL IO_Manager_Receive_APDU
  CALL Command_Interpreter
  CALL IO_Manager_Send_APDU
  GOTO Main_Loop
```

Listing 5.3 Small-OS program code: initializing the microcontroller hardware and the smart card operating system

```
Initialize_Hardware:                                     Subroutine for initializing the
                                                         smart card hardware.
  CALL Kernel_CheckRAM                                   Test the RAM to see if it is in
  IF STATUS (Kernel_CheckRAM = C_Error) THEN (           good working order.
    GOTO IO_Manager_Send_Error_ATR)
  CALL Kernel_DeleteRAM                                  Set the entire RAM to the value
  IF STATUS (Kernel_DeleteRAM = C_Error) THEN (          '00' in order to have a defined
    GOTO IO_Manager_Send_Error_ATR)                      initial state after each reset. An
                                                         intentional side effect is that all
                                                         variables in RAM are initialized.
  CALL Kernel_Check_EDC_ROM                              Check the error detection codes
  IF STATUS (Kernel_Check_EDC_ROM = C_Error)             (EDCs) at various places in the
  THEN (                                                 ROM.
    GOTO IO_Manager_Send_Error_ATR)
  CALL Kernel_Check_EDC_EEPROM                           Check the error detection codes
  IF STATUS (Kernel_Check_EDC_EEPROM = C_Error)          (EDCs) at various places in the
  THEN (                                                 EEPROM.
    GOTO IO_Manager_Send_Error_ATR)
RETURN

Initialize_Operating_System:                             Subroutine for initializing the
  CALL IO_Manager_Set_Transmission_Parameter             smart card operating system.
  CALL SELECT_FILE_MF
  IF STATUS (SELECT_FILE_MF = C_Error) THEN (
    GOTO IO_Manager_Send_Error_ATR)
  SecurityState.MF := 0
  SecurityState.DF := 0
RETURN
```

to see if it is in good working order, just as with PCs. This consists of a RAM test to start with, followed by the calculation of several checksums for the contents of the ROM and EEPROM. If an error occurs here, the smart card attempts to send a special error ATR, following which it does nothing except to wait in an endless loop for the next reset. The success of the attempt to send the error ATR depends primarily on the nature of the error that has occurred. If the error

Listing 5.4 Small-OS program code: routines for data transmissions to and from the smart card

IO_Manager:	Hardware-related subroutines for data exchange with the terminal via the serial I/O line.

IO_Manager_Send_Error_ATR:
  Transmit an ATR that indicates a
  serious _ operating system error
  Error_ATR:
  GOTO Error_ATR
	Subroutine for sending a special ATR that indicates a serious error in the smart card hardware or the operating system. Following the ATR, the operating system is suspended in an endless loop.

IO_Manager_Send_ATR:
  . . .
RETURN
	Subroutine that codes all the parameters of the transmission protocol into an ATR string and transmits the string. Uses the operating system core routine Kernel_IO_SendByte.

IO_Manager_Send_APDU:
  . . .
RETURN
	Subroutine that converts an APDU already present in the I/O buffer into a TPDU, according to the chosen transfer parameters, and then sends the TPDU to the terminal via the serial I/O line. The mechanism for correcting data transmission errors may be used as necessary for the transfer. Uses the operating system core routine Kernel_IO_SendByte.

IO_Manager_Receive_APDU:
  . . .
  APDU.Cmd := TPDU received from
  the _ terminal and converted
  . . .
RETURN
	Subroutine that receives a TPDU from the terminal via the serial I/O line according to the chosen transfer parameters, converts it into an APDU and then stores it in the I/O buffer. The mechanism for correcting data transmission errors may be used as necessary for the transfer. Uses the operating system core routine Kernel_IO_ReceiveByte.

IO_Manager_Set_Transmission_ Parameters:
  . . .
  set clock rate conversion factor to 372
  set convention
  set the length of the transmit buffer
  set the length of the receive buffer
  . . .
RETURN
	Subroutine that sets the parameters for serial data transmission.

is serious and affects the RAM or the program code of the transmission routine, only a garbled ATR or no ATR at all can be sent.

After the chip hardware has been tested and initialized, the initialization of the operating system starts. The important aspects of this are setting the data transmission parameters for the T = 1 protocol, automatically selecting the MF and establishing the security conditions for file accesses. If the MF cannot be found, this is a serious error, and Small-OS then terminates all further actions after sending an error ATR.

The I/O manager

The entire process of transferring data from and to the terminal is handled by the I/O manager. Two hardware-independent program routines in the core of the operating system called Kernel_IO_SendByte and Kernel_IO_ReceiveByte form the basis for transmitting and receiving messages. All other parts of the I/O program code are independent of the target hardware for Small-OS. The ATR and the T = 1 transmission protocol are completely executed in the I/O manager for both successful and unsuccessful instruction processing. A dedicated state machine had to be developed for this, primarily to handle the relatively complicated procedures of the T = 1 protocol. The ISO/IEC 7816-3 Amd. 1 standard describes the reactions of the I/O manager in case of both success and failure by means of many examples. With manually optimized assembler code, a good I/O manager requires at least one kilobyte of memory, which is usually located in ROM.

Real I/O managers preferably use only the RAM for the I/O buffer, since write accesses to the RAM take place at the full speed of the CPU. However, there are now many smart card operating systems whose transmit and receive buffers are larger than the available amount of RAM. In such cases, a certain part of the EEPROM is used as an extension to the data transmission buffer when the amount of data exceeds a certain threshold. This avoids any restrictions on the size of the I/O buffer, but the price for this is a considerably reduced data transmission rate, due to the time required to write to the EEPROM. In addition, with this sort of extended I/O buffer, there is a danger that the buffer could reach the end of its useful life in a relatively short time, due to frequent write accesses to the EEPROM. In spite of these limitations, this is the only technically feasible means to make the data transmission buffer larger than the amount of available RAM.

Operating system kernel

All of the routines that belong to the core of the operating system are collected together in the OS kernel. Most of these routines are either hardware-dependent or rather time-critical, and must therefore be adapted when the program is ported to a different hardware platform. The functions provided by these routines are in each case described by their names. Some of these subroutines can influence aspects of the system that are of concern to its security. For example, the execution time of a subroutine such as Kernel_CompareByteString should not vary depending on the result of the comparison, since any such variations could be used as the basis for determining the internal results of PIN computations by measuring elapsed times. Nowadays, this type of attack has been eliminated by incrementing the retry counter before each PIN comparison as a precautionary measure, but not long ago this was a very promising way to attack a PIN.

Listing 5.5 Small-OS program code: hardware-dependent core routines

OS Kernel: **Kernel_Check_EDC_x:** ... RETURN	Core hardware-related subroutines of the operating system. Subroutine for computing checksums (EDCs) to test the internal consistency of memory regions in the ROM or EEPROM. x ∈ {ROM, EEPROM}
Kernel_x: ... RETURN	Subroutine for elementary functions. x ∈ {CopyByteString, CompareByteString, DeleteRAM, CheckRAM}
Kernel_DES_x: ... RETURN	Subroutine for DES encryption and decryption. x ∈ {Encrypt, Decrypt}
Kernel_IO_x: ... RETURN	Subroutine for sending and receiving a single byte via the serial interface of the microcontroller. x ∈ {SendByte, ReceiveByte}

Command interpreter

The Small-OS command interpreter is constructed in a relatively simple manner. It works on the principle of a dispatcher that, based on the class and instruction bytes, calls a routine that processes the recognized command. This implementation uses little memory and has the additional important advantage that it is relatively easy to integrate new commands into the operating system. This can be done by just adding a few lines of code to the command interpreter. These identify the new command and the associated subroutine call. With suitable code in place, the new command will be recognized and executed as necessary.

The structures of command interpreters that are commonly used in practice, however, are far more complicated. This is partly because they must run in unalterable ROM, but also because it must be possible to download program code into the EEPROM when the operating system is completed. This downloaded code must then be recognized and called when the program runs. The principle that is used for this is a jump table located in the EEPROM, which can be extended as necessary when the card is completed.

Fixed polling of a particular class byte, as used in the Small-OS command interpreter, only makes sense in operating systems that do not support either secure messaging or logical channels, and which also support only a single command class conforming to the ISO/IEC 7816-4 standard. In all other cases, the class byte is used to identify previously defined options, so it need not be the same for all commands.

Listing 5.6 Small-OS program code: the command interpreter

```
Command_Interpreter:                       Command interpreter
  IF APDU.Cmd.CLA <> '00' THEN (           If the class byte is not the same as the class for
    Returncode := C_RC_CLANotSupported     ISO/IEC 7816-4 commands, abort command
    GOTO Command_Interpreter_Exit)         processing and set the corresponding return code.
  IF APDU.Cmd.INS = 'A4' THEN (            If the command SELECT FILE has been sent to the
    CALL SELECT_FILE                       smart card, call the corresponding subroutine.
    GOTO Command_Interpreter_Exit)
  IF APDU.Cmd.INS = 'B0' THEN (            If the command READ BINARY has been sent to
    CALL READ_BINARY                       the smart card, call the corresponding subroutine.
    GOTO Command_Interpreter_Exit)
  IF APDU.Cmd.INS = 'D6' THEN (            If the command UPDATE BINARY has been sent to
    CALL UPDATE_BINARY                     the smart card, call the corresponding subroutine.
    GOTO Command_Interpreter_Exit)
  IF APDU.Cmd.INS = 'B2' THEN (            If the command READ RECORD has been sent to
    CALL READ_RECORD                       the smart card, call the corresponding subroutine.
    GOTO Command_Interpreter_Exit)
  IF APDU.Cmd.INS = 'DC' THEN (            If the command UPDATE RECORD has been sent to
    CALL UPDATE_RECORD                     the smart card, call the corresponding subroutine.
    GOTO Command_Interpreter_Exit)
  IF APDU.Cmd.INS = '20' THEN (            If the command VERIFY has been sent to the smart
    CALL VERIFY                            card, call the corresponding subroutine.
    GOTO Command_Interpreter_Exit)
IF APDU.Cmd.INS = '88' THEN (              If the command INTERNAL AUTHENTICATE has
    CALL INTERNAL_AUTHENTICATE             been sent to the smart card, call the corresponding
    GOTO Command_Interpreter_Exit)          subroutine.
  Returncode := C_RC_INSNotSupported       The command sent to the smart card is not supported.
  Command_Interpreter_Exit:                Set the return code in the I/O buffer as specified by
  CALL Returncode_Manager                  the program code for the command.
RETURN
```

The return code manager and the file manager

The return code manager uses a passed-in value to look up the return code in a table, and then adds it to the end of any data block in the transmit buffer. A dedicated manager is used for this, for two main reasons. First, it is better programming style to have all return codes in one central location. For example, this makes it possible to substitute another return code for a particular return code, easily and in one central place, in case of a software error. This can sometimes be essential. Second, using a return code manager saves precious memory. It is easy to calculate that storing each return code only once in a common table takes up less memory than repeating it everywhere it is needed.

In real smart card operating systems, all file accesses take place via a central file manager, for reasons of security. Frequently, the file manager also computes the checksum of the file header when the file is accessed. In a concession to simplicity, Small-OS accesses the multidimensional variables of the file tree directly when a command is processed, without separating levels or first computing a checksum. The only check that it makes for all commands is to test whether access to the file is allowed. With a complete file manager, read and write accesses to files would take place in a similar manner.

Listing 5.7 Small-OS program code: the return code manager and file manager

```
Returncode_Manager:
  ...
RETURN
```

Manager for setting the return code. Sets the return code based on the value received and a return code table.

```
File_Manager_CheckACRead:

    Status := C_AccessDenied
    IF Ptr.CurrentDF = Ptr.MF THEN (
        // MF is selected
        IF.EF[ Ptr.CurrentWEF ].AccessCondition.
        Read = _
        SecurityState.MF THEN (
            Status := C_AccessAllowed))
    ELSE (
        // DF is selected
        IF.EF[ Ptr.CurrentWEF ].AccessCondition.
        Read = _
            SecurityState.DF THEN (
            Status := C_AccessAllowed))
RETURN
```

File manager for checking the ‘file access condition for ‘read’
Set the variable for the access status to the initial value. Test whether the security state required for reading the selected EF with READ BINARY or READ RECORD has been achieved. The security state that must be achieved depends on the currently selected directory (MF or DF).

```
File_Manager_CheckACUpdate:

    Status := C_AccessDenied
    IF Ptr.CurrentDF = Ptr.MF THEN (
        // MF is selected
        IF.EF[ Ptr.CurrentWEF ].AccessCondition.
        Update = _
        SecurityState.DF THEN (
            Status := C_AccessAllowed))
ELSE (
        // DF is selected
        IF.EF[ Ptr.CurrentWEF ].AccessCondition.
        Update = _
              SecurityState.DF THEN (
              Status := C_AccessAllowed))
RETURN
```

File manager for checking the file access condition for ‘update’.
Set the variable for the access status to the initial value. Test whether the security state required for reading the selected EF with UPDATE BINARY or UPDATE RECORD has been achieved. The security state that must be achieved depends on the currently selected directory (MF or DF).

The basic structure of the program code for smart card commands

The basic functions of smart card commands are described in Chapter 7 (‘Smart Card Commands’). The command processing program code listings shown here are always divided into three sections, separated by thin lines. Any associated subroutines are separated from the main body of the code (and from each other) by thick lines. The first functional block of the main body of the code investigates the command header (which means the CLS, INS, P1 and P2) as much as is possible at the time. This respects the well-known software design principle of testing the consistency and range of the data as early as possible.

Following this, the basic prerequisites for the execution of the command are checked. For example, monitoring the file structure or the access conditions occurs in this section. If all these checks are completed without any rejections, the actual command execution then takes place. This usually involves only a very small amount of code. Finally, the return code for successful execution of the command is set, and a jump to the I/O manager is made (via the command interpreter and the return code manager). The I/O manager sends the result of the command processing and then waits for a new command.

If an error is detected in any of the tests in the pseudocode, the relevant return code is immediately set and program execution branches via the central exit of the subroutine in question.

The command set

All seven of the commands listed below correspond to the ISO/IEC 7816-4 standard in their structure and coding. However, given the freedom of implementation that a standard naturally allows, it is necessary to specify the details of each command in addition to referring to the standard. The specifications of all the commands of Small-OS are thus described in the following subsections. The specifications have been kept relatively short and formal, so they represent only the essential elements.

SELECT FILE

The SELECT FILE command is used to select a file (MF, DF or EF). This normally requires a 2-byte FID (file identifier). Profile N of the ISO/IEC 7816-4 standard also allows DFs to be selected using a DF name, which can contain an AID (application identifier). In Small-OS, the AID can be passed only in its complete form. In the special case of the MF, no FID is required to make the selection, since a suitable command option can be used instead. The MF is also automatically selected after the smart card is reset, and it can be selected from every other directory during a session.

SELECT FILE falls under Case 1 when the MF is directly selected, which means that neither the command APDU nor the response APDU contains a data portion. When the selection is made using a FID or a DF name, SELECT FILE falls under Case 3. This means that a data portion is present in the command APDU but not in the response APDU.

After a reset, the security state of the MF is reset to the ground state (0). Selecting a DF does not affect the security state of the MF, but the security state of the DF is automatically set to the ground state (0) when it is selected. When a linear fixed EF is selected, the record pointer is set to 'invalid'.

The search order, which means whether the operating system should first search for a DF or an EF when selection is made using a FID, is not specified in the standard. However, this is of considerable significance in certain cases. For example, if the MF is currently selected and there is a DF as well as an EF with the same FID, it may not be possible to select the EF, depending on the search routine. Consequently, in implementing the search routine of

Small-OS, we have chosen to always search the list of EFs first, followed by the list of DFs. In case of a conflict, the DF can always be selected using its DF name, so using the FID is not absolutely necessary in such cases.

Table 5.41 Small-OS coding of the Case 1 command SELECT FILE with the option 'direct MF selection'

Data element	Coding	Remarks
CLA	'00'	—
INS	'A4'	—
P1	'00'	—
P2	'00'	—

Table 5.42 Small-OS: coding of the Case 3 command SELECT FILE with the option 'file selection using an FID'

Data element	Coding	Remarks
CLA	'00'	—
INS	'A4'	—
P1	'00'	—
P2	'00'	—
Lc	2	—
DATA	FID	The 2-byte FID of an MF, DF or EF

Table 5.43 Small-OS: coding of the Case 3 command SELECT FILE with the option 'DF selection using a DF name'

Data element	Coding	Remarks
CLA	'00'	—
INS	'A4'	—
P1	'04'	—
P2	'00'	—
Lc	1 . . . 16	—
DATA	DF name	A DF name (1 to 16 bytes) is given. The DF name may contain the AID of the file to be selected.

Table 5.44 Small-OS: coding of the response to the SELECT FILE command

Data element	Coding	Remarks
SW1 || SW2	'9000'	Return code for correct execution of the command

Listing 5.8 Small-OS: program code for the SELECT FILE command, in compliance with ISO 7816-4, Profile N

```
SELECT_FILE:
```

Command according to ISO/IEC 7816-4, Profile N, with extensions.

```
  IF APDU.Cmd.P1 = '00' THEN (
```

If P1 = '00', either the command option for the direct MF selection is set or a file is being selected using its 2-byte FID.

```
  IF ((APDU.Cmd.P2 = '00') AND _
  (LENGTH (APDU.Cmd) = 4)) THEN (
    IF LENGTH (APDU.Cmd) < 4 THEN (
      Returncode := C_RC_WrongLength
      RETURN)
    CALL SELECT_FILE_MF
    RETURN)
```

If P2 = '00' and only the 4-byte command header was passed in, the MF is being directly selected. Test whether the command was sent as a Case 1 command. If not, set the appropriate return code and abort command processing.

```
  IF APDU.Cmd.Lc = '02' THEN (
    IF LENGTH (APDU.Cmd) < 6 THEN (
      Returncode := C_RC_WrongLength
      RETURN)
    IF APDU.Cmd.Data [ 1 . . . 2 ] = '3F00' THEN (
      CALL SELECT_FILE_MF)
      RETURN)
    ELSE (
      CALL SELECT_FILE_FID)
      RETURN)
```

If two data bytes have been passed with the command, a file should be selected using a 2-byte FID. Test whether the command was sent as a Case 3 command. If not, set the appropriate return code and abort command processing. If the MF is selected (with FID = '3F00'), execute the selection immediately, With any other file (DF or EF), execute the selection via a search routine using the given FID as the search string.

```
  IF APDU.Cmd.P1 = °0000 0100° THEN (
    IF LENGTH (APDU.Cmd) < 5 THEN (
      Returncode := C_RC_WrongLength
      RETURN)
    IF ((APDU.Cmd.Lc < 1) _
    OR (APDU.Cmd.Lc > 16)) THEN (
      Returncode := C_RC_LcInconsistentP1P2
      RETURN)
    CALL SELECT_FILE_DFName)
RETURN
```

Test whether a DF can be selected with the DF name. Test whether the command was sent as a Case 3 command. If not, set the appropriate return code and abort command processing. Test whether the accompanying data portion is between 1 and 16 bytes long.

```
SELECT_FILE_MF:
```

Subroutine to select the MF.

```
  SEARCH (MF in file tree)
  IF STATUS (SEARCH = C_NotFound) THEN (
    // MF not found in file tree
    Returncode := C_RC_FatalError
    RETURN)
```

If the MF cannot be found in the file tree, abort with an internal operating system error.

```
  // MF found in file tree
  Ptr.CurrentDF := identified MF address
```

Set the current DF pointer to the address of the located MF.

```
    SEARCH (EF Key in MF)
    IF STATUS (SEARCH) = C_Found THEN (
        Ptr.CurrentIEF.Key := identified EF
        Key address)
```

Search for an EF Key and set the appropriate pointer if it is present.

```
    ELSE (Ptr.CurrentIEF.Key := C_InvalidPointer)
    Ptr.CurrentWEF := C_InvalidPointer
    Ptr.CurrentRecord := C_InvalidPointer
    SecurityState.MF := 0
    SecurityState.DF := 0
    Returncode := C_RC_OK
RETURN
```

Mark the current EF pointer and the current record pointer as invalid. Set the security states of the MF and DF to the initial value.

```
SELECT_FILE_FID:
```

Subroutine to select a DF or EF using a FID.

```
    SEARCH (EF with FID in the currently
    selected DF)
    IF STATUS (SEARCH) = C_Found THEN (
        Ptr.CurrentWEF := identified EF address
        Ptr.CurrentRecord := C_InvalidPointer
        Returncode := C_RC_OK
        RETURN)
```

Search in the current DF for an EF with the given FID. If an EF can be found, set the current EF pointer.

```
    //EF with the given FID not found
    SEARCH (DF mit FID)
    IF STATUS (SEARCH) = C_Found THEN (
        Ptr.CurrentDF := identified DF address
        SEARCH (EF Key in the current DF)
        IF STATUS (SEARCH) = C_Found THEN (
            Ptr.CurrentIEF.Key := identified EF
            Key address)
        ELSE (Ptr.CurrentIEF.Key := C_InvalidPointer)
```

If an EF cannot be found, search for a DF with the given FID. If a DF can be found, set the current DF pointer to point to it. Then search for an EF Key, and set the appropriate pointer if one can be found.

```
        Ptr.CurrentWEF := C_InvalidPointer
        Ptr.CurrentRecord := C_InvalidPointer
        SecurityState.DF := 0
        Returncode := C_RC_OK
        RETURN)
```

Mark the current EF pointer and the current record pointer as invalid. Set the security state of the DF to the initial value.

```
    // neither an EF nor a DF with the given
    FID found
    Returncode := C_RC_FileNotFound
RETURN
```

If no matching file could be found, set the appropriate return code.

```
SELECT_FILE_DFName:
```

Subroutine to select a DF using a DF name.

```
    SEARCH (DF with DF name)
    IF STATUS (SEARCH) = C_NotFound THEN (
        Returncode := C_RC_FileNotFound
        RETURN)
    ELSE (Ptr.CurrentDF := identified DF address)
```

Set the current DF pointer to the address of the DF found in the file tree. Then search for the relevant EF Key in the current DF, and set the appropriate pointer to its address if one can be found.

```
    SEARCH (EF Key in current DF)
    IF STATUS (SEARCH) = C_Found THEN (
      Ptr.CurrentIEF.Key := identified EF
      Key address)
    ELSE (Ptr.CurrentIEF.Key := C_InvalidPointer)
    Ptr.CurrentWEF := C_InvalidPointer         Mark the current EF pointer and the
    Ptr.CurrentRecord := C_InvalidPointer      current record pointer as invalid.
    SecurityState.DF := 0                      Set the security state of the DF to the
    Returncode := C_RC_OK                      initial value.
  RETURN
```

READ BINARY

The READ BINARY command can be used to read data from a transparent EF, starting from a location specified by a 15-bit offset parameter passed by the command. READ BINARY falls under Case 2, which means that there is no data part in the command APDU, but there is a data part in the response APDU.

All length specifications must be an integral number of bytes. The maximum length of the data to be read is limited to the maximum data volume of a transparent EF, which is 255 bytes for Small-OS. If a value of zero is given for the length, all data from the given offset location to the end of the file are read. Profile N of the ISO/IEC 7816-4 standard does not provide for the implicit selection of EFs by means of short file identifiers, so this option is not implemented here.

Before data can be read from an EF using this command, the associated access conditions must be satisfied. Otherwise, the command will be rejected with an appropriate error report.

Table 5.45 Small-OS: coding of the Case 2 command READ BINARY

Data element	Coding	Remarks
CLA	'00'	—
INS	'B0'	—
P1	°0XXX XXXX°	The 7 more significant bits of the offset to the data to be read (offset := XXX XXXX || Y)
P2	Y	The 8 less significant bits of the offset to the data to be read
Le	Z	Z = 0: read all bytes up to the end of the file Z > 0: Z is the number of bytes to be read

Table 5.46 Small-OS: coding of the response to the READ BINARY command

Data element	Coding	Remarks
DATA	...	If the command was correctly executed, the data requested by the command are located in this data element
SW1 || SW2	'9000'	Return code for correct execution of the command

Listing 5.9 Small-OS: program code for the READ BINARY command, in compliance with ISO 7816-4, Profile N

```
READ_BINARY:                                        Command according to ISO/IEC
                                                    7816-4, Profile N.

  IF LENGTH (APDU.Cmd) < 5 THEN (                   Test whether the command was
    Returncode := C_RC_WrongLength                  sent as a Case 2 command. If not,
    RETURN)                                         set the appropriate return code
                                                    and abort command processing.

  IF APDU.Cmd.P1.b8 = °1° THEN (                    Test whether an EF should be
    Returncode := C_RC_FctNotSupported              selected using an SFI.
    GOTO READ_BINARY_Exit)

  IF Ptr.CurrentWEF = C_InvalidPointer THEN (       Test whether an EF is already
    Returncode := C_RC_CmdNotAllowed                selected.
    GOTO READ_BINARY_Exit)

  WITH DF[ Ptr.CurrentDF ].                         Set a part of the file tree as a
                                                    reference for this command.

  IF. EF[ Ptr.CurrentWEF ].Structure =              Test whether the selected EF has a
   C_EFStrucTransparent                             transparent structure.
    THEN (
    Returncode := C_RC_CmdIncompFStruc
    GOTO READ_BINARY_Exit)

  FileOffset := (APDU.Cmd.P1 * 256) + APDU.Cmd.P2   Calculate the offset to the desired
  DataLenToRead := 0                                data in the file, and initialize the
                                                    variable for the amount of data to
                                                    be read.
-----------------------------------------------------------------------------------------
  IF STATUS (File_Manager_CheckACRead) = _          Test whether the security state has
  C_AccessDenied THEN (                             been achieved that is required for
    Returncode := C_RC_SecStateNotSatisfied         reading data from the selected EF
    GOTO READ_BINARY_Exit)                          with READ BINARY.
-----------------------------------------------------------------------------------------
  IF APDU.Cmd.Le = '00' THEN (                      Test whether all available data
    DataLenToRead := .EF [ Ptr.CurrentWEF]. _       should be read (i.e., Le = '00'), or
    TransparentDataSize–FileOffset)                 only a certain amount of data.
  ELSE (
    DataLenToRead := APDU.Cmd.Le)

  IF .EF[ Ptr.CurrentWEF ].TransparentDataSize >=   Test whether the requested offset
  FileOffset _                                      fits with the size of the file.
  THEN (
    Returncode := C_RC_WrongP1P2
    GOTO READ_BINARY_Exit)
  IF .EF[ Ptr.CurrentWEF ].TransparentDataSize < _  Test whether the selected offset
     (FileOffset +DataLenToRead) THEN (             and the requested data length (Le)
    Returncode := C_RC_WrongP1P2                    fit with the size of the file.
    GOTO READ_BINARY_Exit)
    CALL Kernel_CopyByteString                                  Copy the requested data from the
    // from: .EF[ Ptr.CurrentWEF ].TransparentData[ x . . . y ]    file to the I/O transmit buffer.
    // x = FileOffset ; y = (FileOffset + DataLenToRead)
    // to: APDU.Rsp.Data[ 1 . . . DataLenToRead ]
    Returncode := C_RC_OK                                       The command has been processed
                                                                with no error, since in all other
                                                                cases an error exit is used.
    READ_BINARY_Exit:
    END WITH
  RETURN
```

UPDATE BINARY

The UPDATE BINARY command can be used to read data from a transparent EF, starting from a location specified by a 15-bit offset parameter passed by the command. UPDATE BINARY falls under Case 3, which means that there is a data part in the command APDU, but there is no data part in the response APDU.

All length specifications must be an integral number of bytes. The maximum length of the data to be read is limited to the maximum data volume of a transparent EF, which is 255 bytes for Small-OS. If a value of zero is given for the length, all data from the given offset location to the end of the file are read. Profile N of the ISO/IEC 7816-4 standard does not provide for the implicit selection of EFs by means of short file identifiers, so this option is not implemented here. Before data can be written to an EF using this command, the associated access conditions must be satisfied. Otherwise, the command will be rejected with a suitable error report.

Table 5.47 Small-OS: coding of the Case 2 command UPDATE BINARY

Data element	Coding	Remarks
CLA	'00'	—
INS	'D6'	—
P1	°0XXX XXXX°	The 7 more significant bits of the offset to the data to be read (offset := XXX XXXX || Y)
P2	Y	The 8 less significant bits of the offset to the data to be read
Lc	...	Number of bytes to be written
DATA	...	The data bytes to be written

Table 5.48 Small-OS: coding of the response to the UPDATE BINARY command

Data element	Coding	Remarks
SW1 || SW2	'9000'	Return code for correct execution of the command

Listing 5.10 Small-OS: program code for the UPDATE BINARY command, in compliance with ISO 7816-4, Profile N

```
UPDATE_BINARY:                                         Command according to ISO/IEC
                                                       7816-4, Profile N.

  IF LENGTH (APDU.Cmd ) < 6 THEN (                     Test whether the command was
    Returncode:= C_RC_WrongLength                      sent as a Case 3 command. If not,
    RETURN)                                            set the appropriate return code
                                                       and abort command processing.
  IF APDU.Cmd.P1.b8 = °1° THEN (                       Test whether an EF should be
    Returncode:= C_RC_FctNotSupported                  selected using an SFI (i.e.,
    GOTO UPDATE_BINARY_Exit)                           the msb of P1 is set).
  IF Ptr.CurrentWEF = C_InvalidPointer THEN (          Test whether an EF is already
    Returncode := C_RC_CmdNotAllowed                   selected. If not, abort the
    GOTO UPDATE_BINARY_Exit)                           command.
  WITH DF[ Ptr.CurrentDF ].                            Set a part of the file tree as a
                                                       reference for this command.
  IF .EF[ Ptr.CurrentWEF ].Structure =                 Test whether the selected EF has
  C_EFStrucTransparent _                               a transparent file structure.
  THEN (
    Returncode:= C_RC_CmdIncompFStruc
    GOTO UPDATE_BINARY_Exit)
  FileOffset := (APDU.Cmd.P1 * 256) + APDU.Cmd.P2      Calculate the offset to the data in
                                                       the file.
-------------------------------------------------------------------------------------------
  IF STATUS (File_Manager_CheckACUpdate) = _           Test whether the security state
  C_AccessDenied THEN (                                has been achieved that is required
    Returncode:= C_RC_SecStateNotSatisfied             for writing data to the selected EF
    GOTO UPDATE_BINARY_Exit)                           with UPDATE BINARY.
-------------------------------------------------------------------------------------------
  IF .EF[ Ptr.CurrentWEF ].TransparentDataSize < _     Test whether the selected offset
  (FileOffset + APDU.Cmd.Lc) THEN (                    and requested data length (Lc) fit
    Returncode:= C_RC_WrongP1P2                        with the size of the file.
    GOTO UPDATE_BINARY_Exit)
  CALL Kernel_CopyByteString                           Copy the passed-in data from the
  // from: APDU.Cmd.Data[ 1 . . . APDU.Cmd.Lc ]        I/O receive buffer to the file. If an
  // to: .EF[                                          error occurs, abort and report the
Ptr.CurrentWEF ].TransparentData[ x . . . y ]          error to the terminal.
// x = FileOffset ; y =
(FileOffset + APDU.Cmd.Lc)
IF STATUS (Kernel_CopyByteString) = C_WriteError THEN (
  Returncode := C_RC_MemoryFailure      GOTO
UPDATE_BINARY_Exit)
  Returncode := C_RC_OK                                The command has been processed
                                                       with no error, since in all other
                                                       cases an error exit is used.
  UPDATE_BINARY_Exit:
  END WITH
RETURN
```

READ RECORD

The READ RECORD command can be used to read a record from a linear fixed EF. The maximum amount of data to be read is limited to the maximum record length of 255 bytes. The length specification must either match the length of the addressed record or be set to zero. With a length of zero, the entire record is automatically read. All length specifications must be an integer number of bytes. Profile N of the ISO/IEC 7816-4 standard does not provide for the implicit selection of EFs by means of short file identifiers, so this option is not implemented here. READ RECORD falls under Case 2, which means that there is no data part in the command APDU, but there is a data part in the response APDU.

A record in a linear fixed EF can be addressed in three different ways. The number of the desired record can be passed directly with the READ RECORD command. If this record is present in the file, its contents are returned in the response; otherwise the answer contains a suitable error report. This type of access does not affect the record pointer, which can only be modified with the command options ‘first’, ‘last’, ‘next’ and ‘previous’. The record pointer is set to ‘invalid’ immediately after an EF is newly selected. If the option ‘next’ or ‘previous’ is selected when the record pointer is invalid, the record pointer is automatically set to the first or last record of the file, respectively. This makes it possible to (for example) read the records in a file by first selecting the EF and then sending a series of READ RECORD commands with the ‘next’ option, without having to use any other commands. The third type of access is to use the ‘current’ option. In this case the record that is currently indicated by the current record pointer is read. If the record pointer is invalid, the command is aborted with an appropriate error report.

Before data can be read from an EF using this command, the associated access conditions must be satisfied. Otherwise, the command will be rejected with a suitable error report. The record returned in the response when the command is successfully executed is not TLV coded, although this is optionally allowed by the ISO/IEC 7816-4 standard.

Table 5.49 Small-OS: coding of the Case 2 command READ RECORD

Data element	Coding	Remarks	
CLA	'00'	—	
INS	'B2'	—	
P1	X	Y = °0000 0100°, X = 0	read the current record (*Ptr.CurrentRecord*)
		Y = °0000 0100°, X <> 0	read record number X
P2	Y	Y = °0000 0100°	read the record using the method indicated in P1
		X = 0, Y = °0000 0000°	read the first record in the file
		X = 0, Y = °0000 0001°	read the last record in the file
		X = 0, Y = °0000 0010°	read the next record in the file
		X = 0, Y = °0000 0011°	read the previous record in the file
Le	Z	Z = 0: read all bytes until the end of the record	
		Z > 0: Z is the record length	

Table 5.50 Small-OS: coding of the response to the READ RECORD command

Data element	Coding	Remarks
DATA	...	If the command was correctly executed, the record requested by the command is located in this data element
SW1 || SW2	'9000'	Return code for correct execution of the command

Listing 5.11 Small-OS: program code for the READ RECORD command, in compliance with ISO 7816-4, Profile N

READ_RECORD:	Command according to ISO/IEC 7816-4, Profile N

IF LENGTH (*APDU.Cmd*) < 5 THEN (
  Returncode := C_RC_WrongLength
  RETURN)
	Test whether the command was sent as a Case 2 command. If not, set the appropriate return code and abort command processing.

IF *APDU.Cmd.P2.b8* ... b4 <> °00000° THEN (
  Returncode:= C_RC_FctNotSupported
  GOTO READ_RECORD_Exit)
	Test whether an EF should be selected using an SFI.

IF *Ptr.CurrentWEF* = C_InvalidPointer THEN (
  Returncode:= C_RC_CmdNotAllowed
  GOTO READ_RECORD_Exit)
	Test whether an EF is already selected.

WITH*DF[Ptr.CurrentDF].*	Set a part of the file tree as a reference for this command.

IF *.EF[Ptr.CurrentWEF].Structure* = C_EFStrucLinFix THEN (
  Returncode:= C_RC_CmdIncompFStruc
  GOTO READ_RECORD_Exit)
	Test whether the selected EF has a linear fixed structure.

IF STATUS (File_Manager_CheckACRead) = C_AccessDenied _ THEN (
  Returncode:= C_RC_SecStateNotSatisfied
  GOTO READ_RECORD_Exit)
	Test whether the security state has been achieved that is required for reading the selected EF with READ RECORD.

RecordNoToRead := 0 *RecordLenToRead* := 0	Initialize the variable for the number of the record to be read and its length.

IF *APDU.Cmd.P2.b3 ... b1* <> °000° THEN (
  Ptr.CurrentRecord := 1
  RecordNoToRead := *Ptr.CurrentRecord*)
	If the option 'read first record' was selected, set the current record pointer to the first record in the file.

IF *APDU.Cmd.P2.b3 ... b1* <> °001° THEN (
  Ptr.CurrentRecord := *.EF[Ptr.CurrentWEF].*
  NoOfRecords
  RecordNoToRead := *Ptr.CurrentRecord*)
	If the option 'read last record' was selected, set the current record pointer to the last record in the file.

```
IF APDU.Cmd.P2.b3 . . . b1 <> °010° THEN (
    IF Ptr.CurrentRecord = C_InvalidPointer THEN (
        Ptr.CurrentRecord = 1
        RecordNoToRead := Ptr.CurrentRecord)
    ELSE (
        IFPtr.CurrentRecord < .EF[ Ptr.CurrentWEF ]. _
            NoOfRecords THEN (
            Ptr.CurrentRecord := Ptr.CurrentRecord + 1
            RecordNoToRead := Ptr.CurrentRecord)
        ELSE (
            Returncode = C_RC_RecordNotFound
            GOTO READ_RECORD_Exit)))
```

If the option 'read next record' was selected, set the current record pointer to the next record in the file if it does not already point to the last record in the file.

```
IF APDU.Cmd.P2.b3 . . . b1 <> °011° THEN (
    IF Ptr.CurrentRecord = C_InvalidPointer THEN (
        Ptr.CurrentRecord := _
        .EF[ Ptr.CurrentWEF ].NoOfRecords
        RecordNoToRead := Ptr.CurrentRecord)
    ELSE (
        IF Ptr.CurrentRecord > 1 THEN (
            Ptr.CurrentRecord := Ptr.CurrentRecord –1
            RecordNoToRead := Ptr.CurrentRecord)
        ELSE (
            Returncode = C_RC_RecordNotFound
            GOTO READ_RECORD_Exit)))
```

If the option 'read previous record' was selected, set the current record pointer to the previous record in the file, if it does not already point to the first record in the file.

```
IF ((APDU.Cmd.P2.b3 . . . b1 <> °100°) AND _
(APDU.Cmd.P1 = 0)) THEN (
    IF Ptr.CurrentRecord <> C_InvalidPointer THEN (
        Returncode = C_RC_WrongP1P2
        GOTO READ_RECORD_Exit)
    ELSE (
        RecordNoToRead := Ptr.CurrentRecord))
```

If the option 'read current record' was selected, test whether the relevant pointer is valid. If it is, set the internal variable or the command to the current record to be read.

```
IF APDU.Cmd.P2.b3 . . . b1 <> °100° THEN (
    IF.EF[ Ptr.CurrentWEF ].NoOfRecords <
    APDU.Cmd.P1 _
    THEN (
        Returncode = C_RC_WrongP1P2
        GOTO READ_RECORD_Exit)
    ELSE (
        RecordNoToRead := APDU.Cmd.P1))
```

If the option 'address record directly with P1' was selected, test whether the specified record is present in the file.

```
IF APDU.Cmd.Le = '00' THEN (
    RecordLenToRead := .EF[ Ptr.CurrentWEF ]. _
    Record[ RecordNoToRead ].Size)
ELSE (
    RecordLenToRead := APDU.Cmd.Le)
```

Test whether the entire record is to be read with an explicit length specification or with no explicit length specification (Le = '00')

```
IF RecordLenToRead <> .EF[ Ptr.CurrentWEF ]. _
Record[ RecordNoToRead ].Size) THEN (
```

Test whether the requested data length (Le) matches the record

```
        Returncode = C_RC_LcInconsistentP1P2            length.
        GOTO READ_RECORD_Exit)
    CALL Kernel_CopyByteString                          Copy the requested data from the
    // from: .EF[ Ptr.CurrentWEF ].Record                file to the I/O transmit buffer.
    [ RecordNoToRead ]. _
    // Data[ 1 . . . RecordLenToRead ]
    // to:APDU.Rsp.Data[ 1 . . . RecordLenToRead ]
    Returncode = C_RC_OK                                The command has been processed
                                                        with no error, since in all other
                                                        cases an error exit is used.
    READ_RECORD_Exit:
    END WITH
  RETURN
```

UPDATE RECORD

The UPDATE RECORD command can be used to write a record to a linear fixed EF. The data passed by the command are not allowed to be TLV coded, although this is allowed as an option by the ISO/IEC 7816-4 standard. The maximum amount of data to be written is limited to the maximum record length of 255 bytes. The length specification must exactly match the length of the addressed record, and all length specifications must be an integer number of bytes. Profile N of the ISO/IEC 7816-4 standard does not provide for the implicit selection of EFs by means of short file identifiers, so this option is not implemented here. READ RECORD falls under Case 3, which means that there is a data part in the command APDU, but there is no data part in the response APDU.

A record in a linear fixed EF can be addressed in three different ways. The number of the desired record can be passed directly with the UPDATE RECORD command. If this record

Table 5.51 Small-OS: coding of the Case 3 command UPDATE RECORD

Data element	Coding	Remarks	
CLA	'00'	—	
INS	'DC'	—	
P1	X	Y = °0000 0100°, X = 0	write the current record (*Ptr.CurrentRecord*)
		Y = °0000 0100°, X <> 0	write record number X
P2	Y	Y = °0000 0100°	write the record using the method specified by P1
		X = 0, Y = °0000 0000°	write the first record in the file
		X = 0, Y = °0000 0001°	write the last record in the file
		X = 0, Y = °0000 0010°	write the next record in the file
		X = 0, Y = °0000 0011°	write the previous record in the file
Lc	. . .	Number of bytes to be written	
DATA	. . .	The record to be written	

is present in the file, its contents are returned in the response; otherwise, the answer contains a suitable error report. This type of access does not affect the record pointer, which can only be modified with the command options ‘first’, ‘last’, ‘next’ and ‘previous’. The record pointer is set to ‘invalid’ immediately after an EF is newly selected. If the option ‘next’ or ‘previous’ is selected when the record pointer is invalid, the record pointer is automatically set to the first or last record of the file, respectively. This makes it possible to (for example) write all the records in a file by first selecting the EF and then sending a series of UPDATE RECORD commands with the ‘next’ option, without having to use any other commands. The third type of access is to use the ‘current’ option. In this case, the record that is currently indicated by the current record pointer is written. If the record pointer is invalid, the command is aborted with an appropriate error report. Before data can be written to an EF using this command, the associated access conditions must be satisfied. Otherwise, the command will be rejected with a suitable error report.

Table 5.52 Small-OS: coding of the response to the UPDATE RECORD command

Data element	Coding	Remarks
SW1 || SW2	'9000'	Return code for correct execution of the command

Listing 5.12 Small-OS: program code for the UPDATE RECORD command, in compliance with ISO 7816-4, Profile N

```
UPDATE_RECORD:                                      Command according to ISO/IEC
                                                    7816-4, Profile N.

  IF LENGTH (APDU.Cmd) < 6 THEN (                   Test whether the command was
    Returncode := C_RC_WrongLength                  sent as a Case 3 command. If not,
    RETURN)                                         set the appropriate return code and
                                                    abort command processing.

  IF APDU.Cmd.P2.b8 . . . b4 <> °00000° THEN (      Test whether an EF should be
    Returncode := C_RC_FctNotSupported              selected using an SFI.
    GOTO UPDATE_RECORD_Exit)

  IFPtr.CurrentWEF = C_InvalidPointer THEN (        Test whether an EF is already
    Returncode := C_RC_CmdNotAllowed                selected.
    GOTO UPDATE_RECORD_Exit)

  WITH DF[ Ptr.CurrentDF ].                         Set a part of the file tree as a
                                                    reference for this command.

  IF .EF[ Ptr.CurrentWEF ].Structure = C_EFStrucLinFix   Test whether the selected EF
  THEN (                                            has a linear fixed structure.
    Returncode := C_RC_CmdIncompFStruc
    GOTO UPDATE_RECORD_Exit)
---------------------------------------------------------------------------------------
  IF STATUS (File_Manager_CheckACUpdate) = _        Test whether the security state has
  C_AccessDenied) THEN (                            been achieved that is required for
    Returncode := C_RC_SecStateNotSatisfied         reading the selected EF with
    GOTO UPDATE_RECORD_Exit)                        UPDATE RECORD.
```

```
RecordNoToUpdate := 0
```

Initialize the variable for the number of the record to be written.

```
IF APDU.Cmd.P2.b3 . . . b1 <> °000° THEN (
   Ptr.CurrentRecord := 1
   RecordNoToUpdate := Ptr.CurrentRecord)
```

If the option ‘write first record’ was selected, set the current record pointer to the first record in the file.

```
IF APDU.Cmd.P2.b3 . . . b1 <> °001° THEN (
   Ptr.CurrentRecord := _
   .EF[ Ptr.CurrentWEF ].NoOfRecords
   RecordNoToUpdate := Ptr.CurrentRecord)
```

If the option ‘write last record’ was selected, set the current record pointer to the last record in the file.

```
IF APDU.Cmd.P2.b3 . . . b1 <> °010° THEN (
   IF Ptr.CurrentRecord = C_InvalidPointer THEN (
      Ptr.CurrentRecord := 1
      RecordNoToUpdate := Ptr.CurrentRecord)
   ELSE (
      IF Ptr.CurrentRecord < .EF[ Ptr.CurrentWEF ]. _
      NoOfRecords THEN (
         Ptr.CurrentRecord := Ptr.CurrentRecord + 1
         RecordNoToUpdate := Ptr.CurrentRecord)
      ELSE (
         Returncode = C_RC_RecordNotFound
         GOTO UPDATE_RECORD_Exit)))
```

If the option ‘write next record’ was selected, set the current record pointer to the next record in the file if it does not already point to the last record in the file.

```
IF APDU.Cmd.P2.b3 . . . b1 <> °011° THEN (
   IF Ptr.CurrentRecord = C_InvalidPointer THEN (
      Ptr.CurrentRecord =.EF[ Ptr.CurrentWEF ]. _
      NoOfRecords
      RecordNoToUpdate := Ptr.CurrentRecord )
   ELSE (
      IF Ptr.CurrentRecord > 1 THEN (
         Ptr.CurrentRecord := Ptr.CurrentRecord - 1
         RecordNoToUpdate := Ptr.CurrentRecord)
      ELSE (
         Returncode = C_RC_RecordNotFound
         GOTO UPDATE_RECORD_Exit)))
```

If the option ‘write previous record’ was selected, set the current record pointer to the previous record in the file if it does not already point to the first record in the file.

```
IF ((APDU.Cmd.P2.b3 . . . b1 <> °100°) AND _
(APDU.Cmd.P1 = 0)) THEN (
   IF Ptr.CurrentRecord <> C_InvalidPointer THEN (
      Returncode = C_RC_WrongP1P2
      GOTO UPDATE_RECORD_Exit)
   ELSE (
      RecordNoToUpdate:= Ptr.CurrentRecord))
```

If the option ‘write current record’ was selected, test whether the relevant pointer is valid. If it is, set the internal variable or the command to the current record to be read.

```
IF APDU.Cmd.P2.b3 . . . b1 <> °100° THEN (
IF .EF[ Ptr.CurrentWEF ].NoOfRecords <
APDU.Cmd.P1 _
THEN (
   Returncode = C_RC_WrongP1P2
   GOTO UPDATE_RECORD_Exit))
```

If the option ‘address record directly with P1’ was selected, test whether the specified record is present in the file.

```
    IF .EF[ Ptr.CurrentWEF ].                          Test whether the length of the
    Record[ Ptr.CurrentRecord ].Size _                 passed-in record (Lc) matches the
    <> APDU.Cmd.Lc THEN (                              record length of the file.
       Returncode := C_RC_LcInconsistentP1P2
       GOTO UPDATE_RECORD_Exit)
    ------------------------------------------------------------------------------
    CALL Kernel_CopyByteString                         Copy the transferred data from the
    // from: APDU.Cmd.Data[ 1 . . . APDU.Cmd.Lc ]      I/O receive buffer to the file. If an
    // to: .EF[ Ptr.CurrentWEF ].Record                error occurs, abort and report the
    [ RecordNoToUpdate ]. _                            error to the terminal.
    // Data[ 1 . . . APDU.Cmd.Lc ]
    IF STATUS (Kernel_CopyByteString) = C_WriteError
    THEN (
       Returncode := C_RC_MemoryFailure
       GOTO UPDATE_RECORD_Exit)
    Returncode := C_RC_OK                              The command has been processed
                                                       with no error, since in all other
                                                       cases an error exit is used.

    UPDATE_RECORD_Exit:
    END WITH
  RETURN
```

VERIFY

The VERIFY command is used to compare a secret item that has been passed over to the smart card, such as a PIN, to a stored reference value. The length of the PIN must be between one and eight bytes. The operating system does perform any sort of testing of the coding of the data string that is passed in. This means that, for example, a 4-digit PIN (e.g. "1234") could be coded as two BCD bytes ('12' || '34') or as four ASCII bytes [("1" || "2" || "3" || "4") = ('31' || '32' || '33' || '34')]. The VERIFY command falls under Class 3, which means that there is a data part in the command APDU but not in the response APDU.

At most two PINs (PIN number 1 and PIN number 2) can be addressed. They can be located in either the EF Key of the MF or the EF Key of the currently selected DF. A PIN that is stored in the EF Key of the MF is used as a common PIN for all applications in the smart card. If a PIN is stored in the EF Key of a DF, it can be used only for the application associated with that DF. Such a PIN is thus an application-specific PIN.

Every PIN has a retry counter that is reset to zero when a positive comparison result is obtained and incremented by one when a negative result is obtained. If the state of the retry counter is not zero, the number of PIN attempts still allowed is returned encoded in SW2. If the retry counter reaches its maximum value, this is indicated by a separate return code.

Since Small-OS does not have any command to reset the retry counter, whenever a retry counter is standing at its maximum value, there is absolutely no possibility of ever making any further PIN comparisons. Depending on the application, this might mean that the smart card could no longer be used. The PIN located in the EF Key cannot be altered by the user, although this possibility is commonly found in many smart card applications. A specific command would be needed for this (CHANGE REFERENCE DATA as per ISO/IEC 7816-8), but this is not provided in the ISO/IEC 7816-4 standard.

The implementation presented here has a small peculiarity due to simplification. The retry counter is located in EEPROM, as you know. However, EEPROM write accesses need not always be successful, due to the possibility of a write error. Consequently, it is necessary to test the data after each write operation to verify that they have been correctly written. If the result of the test is negative, a suitable return code is set. The retry counter is altered so often in the VERIFY command that no such test has been included in the pseudocode, since the basic relationships of the code would no longer be clear if it were present. You should bear this in mind when examining the code.

The VERIFY command is naturally predestined to be used for attacks on the PIN. The implementation has been designed to make it impossible to base an attack on an analysis of timing behavior or current consumption. The retry counter is always incremented before the received PIN is compared to the reference PIN stored in the EF Key. This ensures that cutting off the power supply to the card immediately following the PIN comparison does not cause the retry counter to be incremented, which would allow an attacker an unlimited number of PIN comparisons.

The actual EEPROM writing process for incrementing the retry counter is by no means as trivial as one might initially imagine. The coding of the retry counter must be constructed such that breaking off the process during the write operation, or during the erase operation that may be necessary before the write operation, cannot result in the retry counter being reset to its initial zero value. The code internal to the operating system must therefore be designed with reference to the minimum-energy state of the EEPROM, which is also known as its secure state. This means that with an EEPROM whose secure state is zero, for example, the initial value of the retry counter may not be coded as zero. If it were, the retry counter could be reset to zero by skillfully switching off the supply voltage during the EEPROM write operation. It would then be possible to determine the PIN within a relatively short time by trial and error, since the retry counter would not be able to fulfill its role as a counter for unsuccessful PIN comparisons. Ideally, the technique of using atomic operations[23] can also be used for writing the retry counter.

Table 5.53 Small-OS: coding of the Case 3 command VERIFY

Data element	Coding	Remarks	
CLA	'00'	—	
INS	'20'	—	
P1	'00'	—	
P2	Y	Y = °100Z ZZZZ°	Use a reference PIN that is stored in the EF Key of the currently selected directory (MF or DF) (*specific reference data*)
		Z = °0 0001° ∧ Z = °0 0010°	Number of the referenced PIN (1 or 2)
Lc	...	Length of the passed-in PIN	
DATA	...	The passed-in PIN	

[23] See also Section 5.10, 'Atomic Operations'

Table 5.54 Small-OS: coding of the response to the VERIFY command

Data element	Coding	Remarks
SW1 || SW2	'9000'	Return code for correct execution of the command (successful PIN comparison)

Listing 5.13 Small-OS: program code for the VERIFY command, in compliance with ISO 7816-4, Profile N

```
VERIFY:                                                  Command VERIFY
                                                         according to ISO/IEC
                                                         7816-4, Profile N.
  IF LENGTH (APDU.Cmd ) < 6 THEN (                       Test whether the command
    Returncode := C_RC_WrongLength                       was sent as a Case 3
    RETURN)                                              command. If not, set the
                                                         appropriate return code and
                                                         abort command processing.
  IF APDU.Cmd.P1 <> '00' THEN (                          Test whether P1 has the
    Returncode := C_RC_WrongP1P2                         allowed value
    GOTO VERIFY_Exit)                                    (P1 must be '00').
  IF ((APDU.Cmd.P2 < '01') OR (APDU.Cmd.P2 > '02'))      Test whether P2 has one of
  THEN (                                                 the two allowed values
    Returncode := C_RC_WrongP1P2                         (P2 must be either 1 or 2).
    GOTO VERIFY_Exit)
  IF ((APDU.Cmd.Lc < = 1) OR (APDU.Cmd.Lc > = 8))        Test whether the length of
  THEN (                                                 the passed data (i.e. the PIN)
    Returncode := C_RC_LcInconsistentP1P2                lies within the allowed range
    GOTO VERIFY_Exit)                                    (1 ≤ Lc ≤ 8).
-----------------------------------------------------------------------------------------
  IFPtr.CurrentIEF.Key = C_InvalidPointer THEN (         Test whether an EF is present
    Returncode := C_RC_RefDataNotFound                   in the current directory.
    GOTO VERIFY_Exit)
  SEARCH (for the PIN with the requested reference       Search for the requested
  number _                                               reference number in EF Key.
  in DF[ Ptr.CurrentDF ].EF[ Ptr.CurrentIEF.Key ] )
  IF STATUS (SEARCH) = C_Found) THEN (                   If a PIN with the specified
    set KeyRecord to the record containing the found PIN reference number is found,
    WITH DF[ Ptr.CurrentDF ].EF[ Ptr.CurrentIEF.Key ].)  set the current key pointer to
  ELSE (                                                 reference it.
    Returncode := C_RC_RefDataNotFound
    GOTO VERIFY_Exit)
-----------------------------------------------------------------------------------------
  IF .Record[ KeyRecord ].KeyPurpose <> C_CmdVERIFY      Test whether the selected
  THEN (                                                 data are allowed to be used
    Returncode := C_RC_CondOfUseNotSatified              with the VERIFY command.
    GOTO VERIFY_Exit)
```

```
IF Ptr.CurrentDF = Ptr.MF THEN (
  // MF is selected
  IF .Record[ KeyRecord ].EntryState <>
  SecurityState.MF _
  THEN (
    Returncode := C_RC_SecStateNotSatisfied
    GOTO VERIFY_Exit))
```

If the MF is selected, test whether its current security state allows the VERIFY command to be used. The security state called for in the key record must have been achieved in the currently selected directory.

```
ELSE (
  // DF is selected
  IF .Record[ KeyRecord ].EntryState <>
  SecurityState.DF _
  THEN (
    Returncode := C_RC_SecStateNotSatisfied
    GOTO VERIFY_Exit))
```

If a DF is selected, test whether its current security state allows the VERIFY command to be used. The security state called for in the key record must have been achieved in the currently selected directory.

```
IF .Record[ KeyRecord ].RCntr > = _
.Record[ KeyRecord ].RCntrMax THEN (
  Returncode := C_RC_AuthMethodBlocked
  GOTO VERIFY_Exit)
```

Test whether the retry counter has reached its maximum value.

```
IF APDU.Cmd.Lc <> .Record[ KeyRecord ].KeySize
THEN (
  Returncode := C_RC_LcInconsistentP1P2
  GOTO VERIFY_Exit)
```

Test whether the passed-in PIN has the same length as the reference PIN.

```
.Record[ KeyRecord ].RCntr := .Record[ KeyRecord ].
RCntr + 1
```

As a precautionary measure, increment the retry counter before making the actual PIN comparison. This defends against a possible attack by analyzing the processing time or current consumption.

```
CALL Kernel_CompareByteString
// Data 1: APDU.Cmd.Data[ 1 . . . APDU.Cmd.Lc ]
// Data 2: .Record[ KeyRecord ].
KeyData[ 1 . . . APDU.Cmd.Lc ]
```

Compare the passed-in PIN with the stored reference PIN.

```
IF STATUS (Kernel_CompareByteString) = C_Equal
THEN (
  .Record[ KeyRecord ].RCntr := 0
  IF Ptr.CurrentDF = Ptr.MF THEN (
    // MF is selected
    SecurityState.MF := .Record[ KeyRecord ].
    ResultState.OK)
  ELSE (
    // DF is selected
    SecurityState.DF := .Record[ KeyRecord ].
    ResultState.OK))
```

If the passed-in PIN matches the stored reference PIN, set the retry counter to zero false attempts and set the security state for successful PIN testing.

```
  IF STATUS (Kernel_CompareByteString) = C_NotEqual _
  THEN (
    IF Ptr.CurrentDF = Ptr.MF THEN (
      // MF is selected
      SecurityState.MF := _
      .Record[ KeyRecord ].ResultState.NOK )
    ELSE (
    // DF is selected
      SecurityState.DF := _
      .Record[ KeyRecord ].ResultState.NOK )
    IF .Record[ KeyRecord ].RCntr = _
    .Record[ KeyRecord ].RCntrMaxValue THEN (
      Returncode := C_RC_AuthMethodBlocked)
    ELSE (
      Returncode := C_RC_CounterX || (_
      .Record[ KeyRecord ].RCntrMax –_
      .Record[ KeyRecord ].RCntr))
    GOTO VERIFY_Exit)
  Returncode := C_RC_OK

  VERIFY_Exit:
  END WITH
RETURN
```

If the passed-in PIN does not match the stored reference PIN, the retry counter will have already been incremented before the PIN comparison.

Set the security state that results from an unsuccessful PIN comparison. If the retry counter has reached its maximum value, set SW2 to the number of unsuccessful attempts that are still allowed. Otherwise, indicate in the return code that no further PIN verifications are possible.

The command has been processed with no error, since in all other cases an error exit is used.

INTERNAL AUTHENTICATE

The INTERNAL AUTHENTICATE command is used to authenticate the smart card via a challenge–response procedure. An 8-byte random number is sent to the smart card, which encrypts it using the DES algorithm. The number of the key to be used must be given in parameter P2, which must indicate whether the key to be used is located in the EF Key file of the MF or the currently selected DF. INTERNAL AUTHENTICATE falls under Case 4, which means that a data part is present in both the command APDU and the response APDU.

The ISO/IEC 7816-4 standard only specifies a few parameters for the authentication commands. The cryptographic algorithm, for example, is not specified. In Small-OS, the DES algorithm has been chosen as the cryptographic algorithm. As an extension to Profile N of the ISO/IEC 7816-4 standard, a 5-byte key number is passed with the command.

In Small-OS, INTERNAL AUTHENTICATE can in principle be used to encrypt eight bytes of plaintext into eight bytes of ciphertext using a selectable key. A smart card with Small-OS would thus fall under strict export control in almost all countries, so that it would take several weeks or even months to obtain an export permit for the card. Consequently, INTERNAL AUTHENTICATE is implemented in many real smart cards such that it is not possible to directly encrypt data. This avoids the export restrictions.

The ability to directly encrypt a plaintext block into a ciphertext block is equally risky from a cryptographic perspective, since it could be used to generate plaintext–ciphertext pairs for brute-force attacks. In addition, this implementation would be very susceptible to differential fault analysis.[24] The simplest form of attack to implement would be a timing attack[25] on the computation of the DES, which thus must be noise-free. Otherwise, the key could be determined by measuring the processing time for the computation.

For all of these reasons, the starting value is most commonly extended in practice by appending a random number generated inside the smart card and the card's own unique number. The resulting number is then encrypted, and the ciphertext is sent back to the terminal along with the data used to extend the original number. This means that this command can no longer be used to encrypt data (which solves the export problem). In addition, the fact that a different value is encrypted each time provides the basis for protection against differential fault analysis (DFA) and differential performance analysis (DPA).[26] All of these measures show relatively dramatically that both the specification and the implementation of even an ostensibly simple command, such as INTERNAL AUTHENTICATE, requires considerable knowledge and experience to protect the keys of a smart card application against attack.

Table 5.55 Small-OS: coding of the Case 4 command INTERNAL AUTHENTICATE

Data element	Coding	Remarks	
CLA	'00'	—	
INS	'88'	—	
P1	'00'	—	
P2	Y	Y = °100Z ZZZZ°	use a key from the EF Key file in the currently selected directory (MF or DF) (specific reference data)
		Z ZZZZ°	number of the referenced key (1 . . . 31)
Lc	8	Length of the passed-in random number	
DATA	. . .	The passed-in random number	
Le	8	Length of the returned random number	

Table 5.56 Small-OS: coding of the response to the INTERNAL AUTHENTICATE command

Data element	Coding	Remarks
DATA	. . .	If the command was correctly executed, this data element contains the encrypted random number that has been encrypted using the key referenced by the command
SW1 || SW2	'9000'	Return code for correct execution of the command

[24] See also Section 8.2.4, 'Attacks and defense measures while the card is in use'
[25] See also Section 8.2.4.2, 'Attacks on the logical level'
[26] See also Section 8.2.4.2, 'Attacks on the logical level'

Listing 5.14 Small-OS: program code for the INTERNAL AUTHENTICATE command, in compliance with ISO 7816-4, Profile N, extended with global and specific reference data (selectable reference to the MF or DF)

```
INTERNAL_AUTHENTICATE:                                       Command INTERNAL
                                                             AUTHENTICATE according
                                                             to ISO/IEC 7816-4, Profile N.

  IF APDU.Cmd.P1 <> '00' THEN (                              Test whether P1 has the allowed
    Returncode := C_RC_WrongP1P2                             value (P1 must be '00').
    GOTO INTERNAL_AUTHENTICATE_Exit)

  IF APDU.Cmd.Lc <> 8 THEN (                                 Test whether the passed data
    Returncode := C_RC_WrongLength                           (i.e. the random number) has
    GOTO INTERNAL_AUTHENTICATE_Exit)                         the allowed length of 8 bytes.

  IF Ptr.CurrentIEF.Key = C_InvalidPointer THEN (    Returncode    Test whether an EF is present
:= C_RC_RefDataNotFound                                      in the current directory.
    GOTO INTERNAL_AUTHENTICATE_Exit)

  IF APDU.Cmd.P2.b5 . . . b1 = ◦00000◦ THEN (                Determine the number of the
    Returncode := C_RC_WrongP1P2                             key to be used from P2.
    GOTO INTERNAL_AUTHENTICATE_Exit)
  ELSE (
    KeyNumber := APDU.Cmd.P1.b5 . . . b1)

  SEARCH (for the key with theKeyNumber in _                 Search for the reference number
  DF[ Ptr.CurrentDF ].EF[ Ptr.CurrentIEF.Key ])              requested via P2 in EF Key. If a
  IF STATUS (SEARCH) = C_Found THEN (                        key with the specified reference
    set KeyRecord to the record with the found key           number is found, set the current
    WITH DF[ Ptr.CurrentDF ].EF[ Ptr.CurrentIEF.Key ].)      key pointer to reference it.
  ELSE (
    Returncode := C_RC_RefDataNotFound
    GOTO INTERNAL_AUTHENTICATE_Exit)

  IF .Record[ KeyRecord ].KeyPurpose <> _                    Test whether the selected key is
  C_CmdINTAUTH THEN (                                        allowed to be used with the
    Returncode := C_RC_CondOfUseNotSatified       GOTO       INTERNAL AUTHENTICATE
INTERNAL_AUTHENTICATE_Exit)                                  command.

  IF Ptr.CurrentDF = Ptr.MF THEN (                           The security state called for in
    // MF is selected                                        the key record must have been
    IF.Record[ KeyRecord ].EntryState <>                     achieved in the currently
    SecurityState.MF _                                       selected directory (MF or DF).
  THEN (
  IF Ptr.CurrentDF = Ptr.MF THEN (                           The security state called for in
    // MF is selected                                        the key record must have been
    IF.Record[ KeyRecord ].EntryState <>                     achieved in the currently
    SecurityState.MF _                                       selected directory (MF or DF).
  THEN (
      Returncode := C_RC_SecStateNotSatisfied
      GOTO INTERNAL_AUTHENTICATE_Exit))
    ELSE (
      // DF is selected
      IF .Record[ KeyRecord ].EntryState <>
      SecurityState.DF _
      THEN (
        Returncode := C_RC_SecStateNotSatisfied
        GOTO INTERNAL_AUTHENTICATE_Exit))
```

CALL Kernel_DES_Encrypt
// plaintext: *APDU.Cmd.Data[1 . . . 8]*
// key: *.Record[KeyRecord].KeyData[1 . . . 8]*
// ciphertext: stored in *APDU.Rsp.Data[1 . . . 8]*
	Encrypt the passed-in data (the random number) using the referenced key, and place the result in the transmit buffer.

Returncode := C_RC_OK	The command has been processed with no error, since in all other cases an error exit is used.
INTERNAL_AUTHENTICATE_Exit:	
END WITH	
RETURN	

A simple application example

The following simple smart card application illustrates the construction and contents of the variables in the EEPROM with the Small-OS operating system. The function of this application can be described in a few words. It allows the creation of a file that is 50 bytes long, whose contents can always be read, and which can be overwritten after the PIN value ‘1234’ has been successfully tested. The reference number of the PIN is 1, and a maximum of three unsuccessful attempts is allowed for the PIN input. The EF containing the file is located underneath its own DF. All file names (DF names and FIDs) can be freely chosen.

Table 5.57 shows how the required capabilities can be implemented by putting appropriate values in the structures used for the file tree. In order to realize the required access conditions, the state machine shown in Figure 5.66 is implemented. After a reset, Small-OS automatically sets the security state of the DF to zero. In this state, the EF containing the data can be read but not written. State 1 is necessary for writing to the file. If a VERIFY command is successfully executed with the correct PIN, the DF is set to security state 1 (. . . *ResultState.OK*), and the file can be written. If the PIN test is not successful, the DF is set to security state 0 (. . . .*ResultState.NOK*).

The commands READ BINARY and UPDATE BINARY are used to read and write data from and to the file, respectively. The entire file can be read or written, or only part of it. The only decisive factor here is that the security state achieved in the DF must correspond the desired type of access.

This example quite clearly shows two limitations of Small-OS, which arise only from the desire to keep the extent of the pseudocode within reasonable limits. Once security state 1 has been reached, the file can no longer be read, since reading is only allowed in security state 0. This could be remedied by integrating the option of a ‘greater than or equal’ comparison into

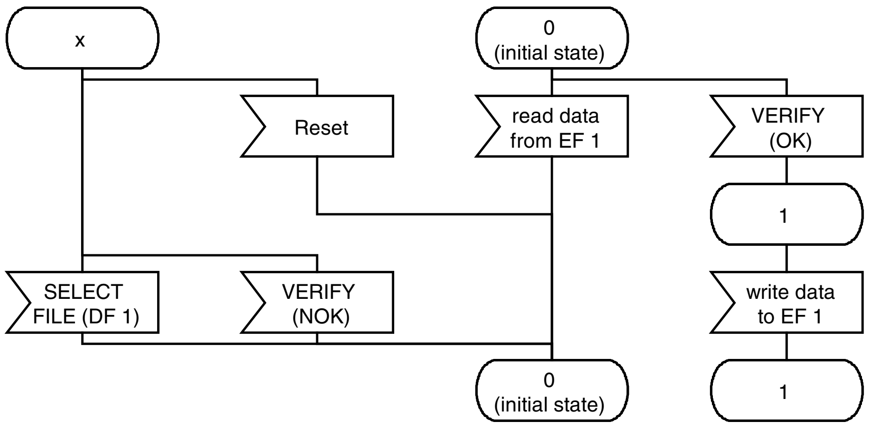


Figure 5.66 State diagram of the file with the FID '0001' (EF 1). Reading data from the EF and writing data to the EF relate to successful READ BINARY and UPDATE BINARY commands to the smart card, respectively. State 'x' is any arbitrary state

Small-OS, in addition to the 'equal' comparison. Another possible solution would be to not limit the file access conditions to only one comparison test per operation (reading or writing), but instead to allow several tests for each operation. In this case, reading the file could be allowed in state 0 as well as in state 1, even if only 'equal' comparisons are possible.

Multiple access conditions for an access operation can be integrated into Small-OS just as easily as testing for greater than or equal. However, the pseudocode would be a bit more extensive, as would be the amount of program code for an actual implementation. With real smart card operating systems, such extensions can easily cause the amount of code to exceed the amount of memory available in the microcontroller (ROM or EEPROM). In practice, these strict memory limitations often mean that certain useful functions cannot be implemented.

In the application just described, state 1 cannot be exited once it has been achieved by means of a successful PIN verification, which means that the file can no longer be read, due to the access conditions. State 0 can be exited either via an unsuccessful PIN verification or by resetting the smart card. In terms of a 'clean' application design, this is a rather unfortunate solution. However, the problem can be remedied using a simple trick to return to the initial state. As you know, selecting a DF causes the security state of the DF in question to be reset. Consequently, the DF can be selected one more time if necessary, which automatically changes the security state from 1 back to 0.

Until recently, applications such as this were coded manually in assembler, in the form shown here. Nowadays, there are application generators for almost all commercially available smart card operating systems. These programs run on PCs and have graphical user interfaces for creating files and access conditions. A similar process also takes place in the smart card simulator when a new application or file is created. Once the necessary files for the application have been created, they can be loaded into a smart card using the application generator. After this, the first trials with the new application can be carried out.

This sample application can also be profitably used to illustrate some interesting types of attacks. Although these are theoretical in nature, since they require sophisticated technical

Table 5.57 Sample values of the file management variables for a simple sample application. There is one DF directly under the MF, and it contains one transparent EF that is 50 bytes long. The content of the EF can be read at any time, but it can be modified only after a PIN has been entered.

Variable name	Description, contents and size
Data structures for the MF	
DF[1].FID := '3F00'	The standard FID of the MF.
Data structures for the DF	
DF[2].FID := 'DF01'	The DF FID is 'DF01' (freely chosen).
DF[2].DFName := 'D276' || '000060'	The registered AID of Wolfgang Rankl is used here for the DF name.
DF[2].LenDFName := 5	The length of the DF name is 5 bytes.
Data structures for the EF	
DF[2].EF[1].FID := '0001'	The EF FID is '0001' (freely chosen).
DF[2].EF[1].Structure := C_EFStrucTransp	The structure of the EF is transparent.
DF[2].EF[1].Type := C_EFTypeWorking	The type of the EF is working.
DF[2].EF[1].AccessCondition.Read := 0	The state that is necessary for the READ command to be allowed for the EF. The access condition is set to 0, which means that reading the file is always allowed.
DF[2].EF[1].AccessCondition.Update := 1	The state that is necessary for the UPDATE command to be allowed for the EF. The access condition is set to 1, which means that altering the file is allowed only after successful PIN entry.
DF[2].EF[1].TransparentDataSize := 50	The transparent file is 50 bytes long.
Data structures for the EF Key	
DF[2].EF[2].Record[x].KeyData := '1234'	The (hexadecimal) PIN value is '1234'.
DF[2].EF[2].Record[x].KeySize := 2	The length for the PIN is 2 bytes.
DF[2].EF[2].Record[x].KeyNo := 1	The reference number of the PIN is 1.
DF[2].EF[2].Record[x].RCntr := 0	The initial value of the retry counter is 0.
DF[2].EF[2].Record[x].RCntrMaxValue := 3	The maximum value of the retry counter is 3, which means that the user is allowed a maximum of three incorrect PIN entries.
DF[2].EF[2].Record[x].KeyPurpose := C_CmdVERIFY	The data content of the EF Key record may only be used for PIN testing with the VERIFY command.
DF[2].EF[2].Record[x].EntryState := 0	PIN testing is possible only in state 0.
DF[2].EF[2].Record[x].ResultState.OK := 1	If the PIN comparison is successful, state 1 is set in the current DF.
DF[2].EF[2].Record[x].ResultState.NOK := 0	If the PIN comparison is not successful, state 0 is set in the current DF.

equipment, they still illustrate some noteworthy principles. The prerequisite for these attacks is the ability to alter specific contents of the EEPROM, which in technical terms amounts to the ability to manipulate the stored charges of individual EEPROM cells. The necessary techniques are discussed in more detail in Chapter 8. Here we only want to look at the consequences of this manipulation.

If it were possible to deliberately alter the content of the file tree pointer that normally indicates the data of the EF, it could be changed to indicate the data content of the EF Key. The EF is always readable in state 0, and in this case, READ BINARY would read the PIN instead of the actual 50 bytes in the EF. Of course, the exact address of the data part of the EF Key must be known for this attack to be used, and a lot of insider knowledge is required to obtain this information. It would be simpler if the variable *DF[2].EF[1].TransparentDataSize* could be altered. If the value of this variable is increased to a large value, for example, then READ BINARY can be used to read an amount of data extending past the end of the file, according to the new value of the variable. If the EF Key is located in memory following EF 1, both the EF Key header and the actual file content could be read straightaway.

Manipulation of the EEPROM could also be used to repeatedly reset the PIN retry counter. The PIN could then be determined within an acceptable amount of time by trial and error. An even simpler attack would be to set the PIN itself to a known value.

These examples very clearly show that the security of a smart card would completely collapse if it were possible to manipulate the EEPROM. It would make no difference if the contents of the EEPROM could not be read, but could only be overwritten. In any case, the PIN and the keys of the card could be determined. The only thing that would have to be known is the exact memory addresses where the manipulations should be carried out. Checksums for the header contents, if present, would only make the necessary modifications to the EEPROM more complicated, but they ultimately could not prevent them. Of course, it is presently not technically possible to change individual bits at any desired locations in the EEPROM. The approaches just described thus represent theoretically interesting forms of attack rather than actual dangers. However, if this sort of manipulation of the EEPROM were to become possible in the future, these examples clearly and unambiguously show the potential dangers that would arise.

6
Smart Card Data Transmission

The possibility of two-way communications is a prerequisite for all interactions between a smart card and a terminal. However, only a single line is available. Digital data are exchanged between the card and the terminal via this electrical connection. Since only one line exists, the card and the terminal must take turns transmitting data, with the other party acting as the recipient. This alternate transmitting and receiving of data is called a half-duplex procedure.

A full-duplex procedure, in which both parties can transmit and receive simultaneously, is presently not implemented for smart cards. However, since most smart card processors have two I/O ports, and two of the eight contacts are reserved for future applications (such as a second I/O connection or USB interface), full-duplex operation would certainly be technically possible. This will doubtless be implemented in hardware and operating systems in the medium term.

Communication with the card is always initiated by the terminal. The card always responds to commands from the terminal, which means that the card never sends data without an external stimulus. This yields a pure master–slave relationship, with the terminal as master and the card as slave. The proactive command procedure,[1] which is used by telecommunications smart cards and allows the smart card to send commands to the terminal, is also based on the standard master–slave arrangement.

After a card has been inserted in a terminal, its contacts are first mechanically connected to those of the terminal. The five active contacts are then electrically enabled in the correct sequence.[2] Following this, the card automatically executes a power-on reset and then sends an Answer to Reset (ATR) to the terminal. The terminal evaluates the ATR, which contains various parameters relating to the card and data transmissions, and then sends the first command. The card processes the command and generates a response, which it sends back to the terminal. This back-and-forth interplay of commands and responses continues until the card is deactivated.

Between the ATR and the first command sent to the card, the terminal can also send a Protocol Parameter Selection (PPS) command. The terminal can use this command, which like the ATR is independent of the transmission protocol, to set various transmission parameters for the card's transmission protocol.

[1] See also Section 13.2.4, 'The SIM'
[2] See also Section 3.3.6, 'Activation and deactivation sequences'

Smart Card Handbook, Third Edition. W. Rankl and W. Effing
© 2004 John Wiley & Sons, Ltd ISBN: 0-470-85668-8

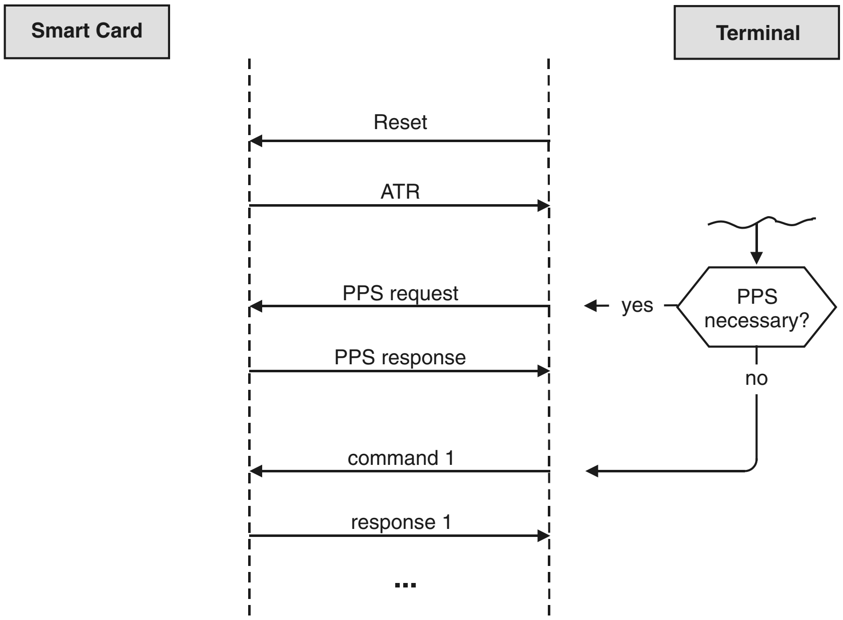


Figure 6.1 The initial data transfers between a terminal and a smart card, showing the answer to reset (ATR), protocol parameter selection (PPS) and the first command–response pair

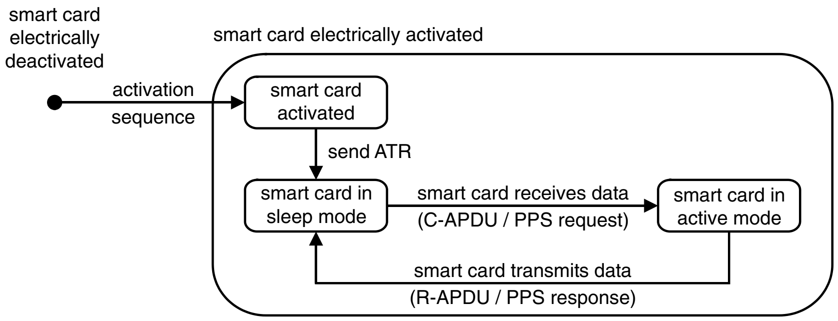

smart card electrically deactivated
smart card electrically activated
activation sequence
smart card activated
send ATR
smart card in sleep mode
smart card receives data (C-APDU / PPS request)
smart card in active mode
smart card transmits data (R-APDU / PPS response)

Figure 6.2 The general states of a smart card for activation and communication with the terminal

The entire procedure for data transmission to and from the smart card can be represented using the OSI layer model. This differentiates electrical events on the I/O line, logical processes in the actual transmission protocol and the behavior of applications that use these processes. The behavior and interactions within and between these layers are specified in several international standards. These relationships are illustrated in Figure 6.3.

In this chapter, the asynchronous transmission protocols are described with respect to relevant standards in terms of their functions. All allowed parameters and settings within the context of the protocol are described. In practice, it often happens that smart cards do not support all options of the transmission protocol, due to the limitations of available memory.

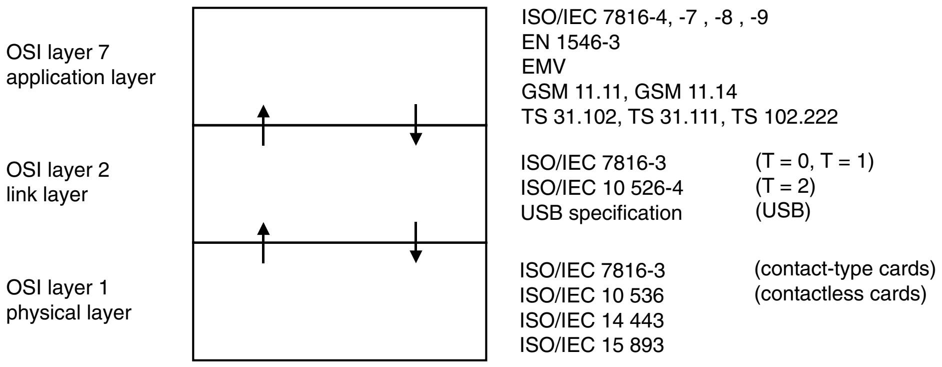


Figure 6.3 The OSI model for communication between terminals and smart cards

From a functional perspective, the various options can be regarded as simply a range of possibilities from which an optimum set can be selected for a particular application or smart card. The important consideration is that the selected parameters should not be too exotic, in order to allow the card to communicate with as many different types of terminals as possible.

In terminals, the situation with regard to data transmission protocols is somewhat different. There the full functionality of the relevant standard is normally implemented, since sufficient memory is available.

6.1 THE PHYSICAL TRANSMISSION LAYER

The general parameters of the physical transmission layer are specified in the international smart card standard ISO/IEC 7816-3. This is the fundamental standard for all aspects of communications at the physical level.

The entire data exchange with the smart card takes place digitally, which means that it employs only the logic values 0 and 1. The voltage levels used are the conventional values for digital technology, namely +5 V, +3 V and +1.8 V, with 0 V as a reference. The choice of whether a physical high or low level represents a logic 1 is freely definable, with the actual selection being indicated by the card in the first byte of the ATR. Here the 'direct convention' means that a logic 1 is represented by the +1.8-V, +3-V or +5-V level, while the 'inverse convention' means that the +1.8-V, +3-V or +5-V level represents a logic 0. In either case, the level of the I/O line is always high in the quiescent state, which is when no data are being transmitted.

Communications between a smart card and the outside world take place serially. Data handled by the processor in the form of bytes must therefore be converted into a serial bit stream. To this end, each byte is separated into its eight individual bits, which are then sent over the line one after the other. The bit order depends on the convention used. With the direct convention, the first data bit after the start bit is the least significant bit in the byte. With the inverse convention, the most significant bit is sent directly following the start bit.

Data transmission between the card and the terminal is asynchronous, which means that each byte sent must be provided with supplementary synchronization bits. A start bit is added to the beginning of each transmitted byte to mark the start of the transmission sequence for the recipient. At the end of each byte, the sender also adds a parity bit for error detection and

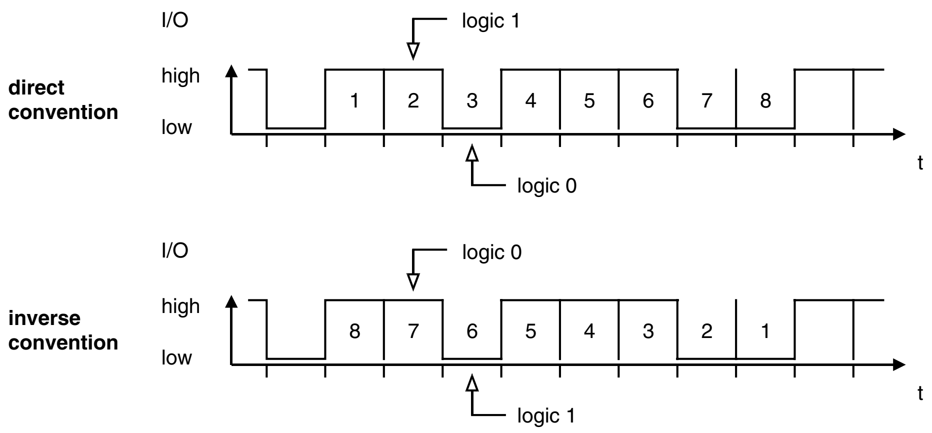


Figure 6.4 Data transmission conventions: (a) data transmission using the direct convention; (b) data transmission using the inverse convention

one or two stop bits. The time allocated to the stop bits is designated as the 'guard time' in the T = 0 protocol. In principle, the guard time is a sort of stop bit. The receiver and the transmitter can both use this time to prepare for the next byte transmission. The parity of each byte must always be even. The parity bit thus has the logic value 1 if the number of ones in the byte is odd, or 0 if the number of ones in the byte is even.

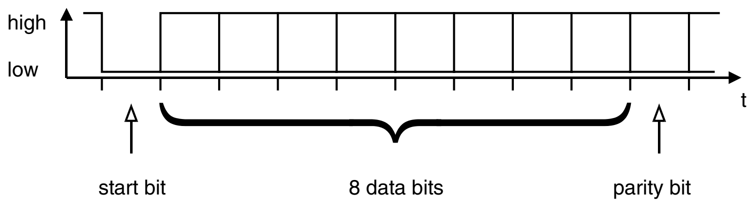


Figure 6.5 Structure of a character for data transmission

Since smart card microcontrollers do not have timers that are independent of the applied clock signal, it is not possible to specify an absolute time interval for an individual data bit. The bit interval is therefore specified in terms of the applied clock. For this purpose, a 'divider' is defined to indicate the number of clock pulses per bit interval. The duration of one bit is called an 'elementary time unit' (etu).

It is thus meaningless to specify the data transmission rate of a smart card as a fixed value (such as 9600 bit/s), since the rate is proportional to the rate of the applied clock. However, there are essentially only two divider values in use worldwide: 372 and 512. For some time now, even smaller divider values are being used increasingly often to increase the transmission rate. Reducing the divider value makes it increasingly difficult for the card's operating system to receive and transmit data, since the processor has progressively less time to perform these tasks. For instance, if data are received using a divider value of 64, the processor has only 64 clock intervals to recognize each bit and transfer it to the I/O buffer.

To calculate the transmission rates that can be achieved with the standard divider values, we only need to consider the clock rate and the divider value, as shown in the following examples:

$$3.5712\ \text{MHz} \div 372 = 9600\text{bit/s}$$

$$4.9152\ \text{MHz} \div 512 = 9600\text{bit/s}$$

A data transmission rate of exactly 9600 bit/s can thus be obtained with both commonly used clock frequencies (3.5712 MHz and 4.9152 MHz).[3] The desire for a transmission rate of 9600 bit/s is the reason for the awkward divider values. In the early days of smart card technology, inexpensive quartz crystals were available for only very few frequencies. Standard crystals for use in television sets were used, and the divider values for the cards were set to obtain a data transmission rate of 9600 bit/s, which was a common value at the time. A clock rate of 4.77 MHz was used in early PCs for the same reason, since this was compatible with US television sets, and in principle a PC could thus be connected to a television set.

If we assume that 5 MHz is the highest practical value for the clock rate and 32 is the minimum divider value, we obtain the current upper limit for the data transmission rate, at least as long as transmission is performed using software executed by the processor:

$$5\ \text{MHz} \div 32 = 156{,}250\ \text{bit/s}$$

Of course, it is possible to reduce the divider value even further in order to increase the transmission rate. However, this significantly increases the amount of program code in the card, and so it is not normally done, due to the limited amount of available memory. Many new smart card microcontrollers have a built-in hardware unit (a universal asynchronous receiver/transmitter, or UART) that handles data transmission via the serial interface. This sharply reduces the amount of software overhead in the card for handling data transmission, making it possible to use much higher data transmission rates. Such an interface unit can easily achieve the standard transmission rate of 111.6 kbit/s.[4]

The bit interval can be calculated from the clock rate and the divider value. With a 3.5712 MHz clock frequency and a divider value of 372, we obtain a bit interval of 104 µs, which by definition corresponds to one etu (elementary time unit) for this divider value. We can construct the diagram shown in Figure 6.6 for various transmission rates.

The timing of serial data transmissions does not have to be strictly controlled. For technical reasons, a certain amount of tolerance is allowed. Since many smart card microcontrollers do not have interface hardware, it is sometimes necessary to exploit the allowed tolerance to accommodate software implementations of the interface function. The timing variation between the falling edge of the start bit and the final transition of the *n*th bit may not exceed ± 0.2 etu. As far as the transmitter is concerned, this means that while the variation in the timing of individual bits may be up to ± 0.2 etu, the variation over several bits is also not allowed to exceed this value. The sum of the timing variations over a group of bits must therefore not exceed the allowed tolerance.

[3] Here and in the following text, the unit ‘bit/s’ is always used in this book for the data transmission rate. In the literature and in many standards, the unit ‘baud’ is sometimes erroneously used as equivalent to ‘bit/s’. The term ‘baud’ refers to the number of state changes per second during a data transmission. Depending on the transmission method used, one or more information bits can be transmitted for each state change. For this reason, the baud rate can only be taken to be equal to the data transmission rate in the particular case that only one bit is transmitted for each state change

[4] See also Section 16.11.3, ‘Determining the data transmission rate’

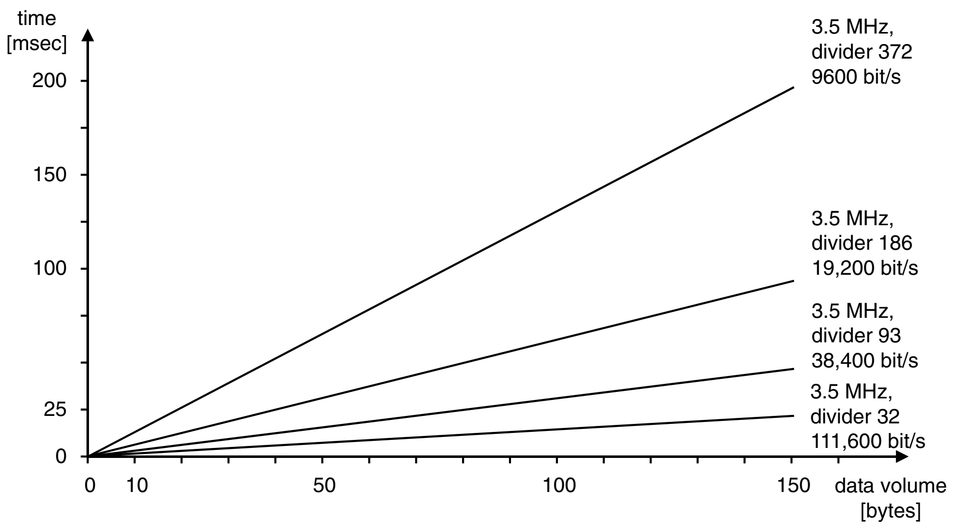


Figure 6.6 Data transmission times for typical transmission rates, assuming 1 start bit, 8 data bits, 1 parity bit and 2 stop bits per byte

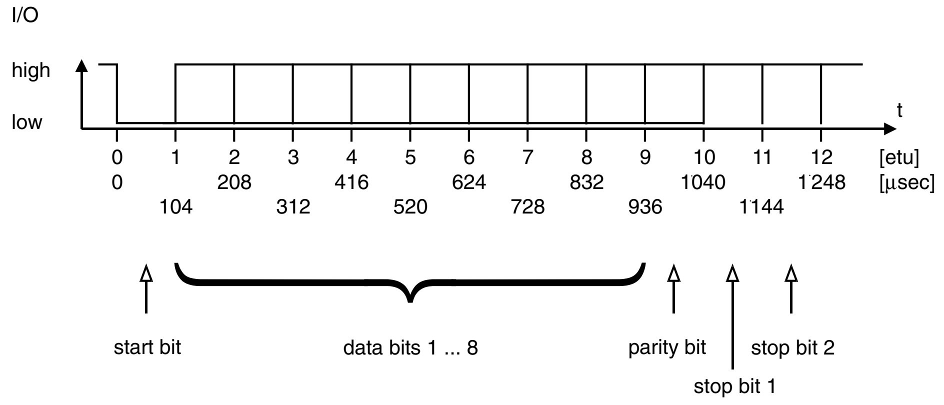

I/O
high
low
t
0 1 2 3 4 5 6 7 8 9 10 11 12 [etu]
0 104 208 312 416 520 624 728 832 936 1040 1144 1248 [µsec]
start bit
data bits 1 ... 8
parity bit
stop bit 1
stop bit 2

Figure 6.7 Timing diagram for one character at 9600 bit/s (corresponding to a 3.5712-MHz clock and a divider value of 372)

Particularly when data are transmitted via a physical conductor, it is relatively common for signal dropouts and overshoots to occur. Consequently, the incoming signal is sampled multiple times rather than just once. Triple sampling followed by a 2-of-3 majority vote is a commonly used method. Small distortions in signal levels can thus be compensated at relatively little effort. Increasing the number of samples to five or seven would make little sense, given the generally good quality of the smart card data transmissions and the amount of extra effort this would entail.

The three samples should be distributed as evenly as possible over the received bit interval, in order to best compensate for brief dropouts. This is done by sampling at the middle of the bit

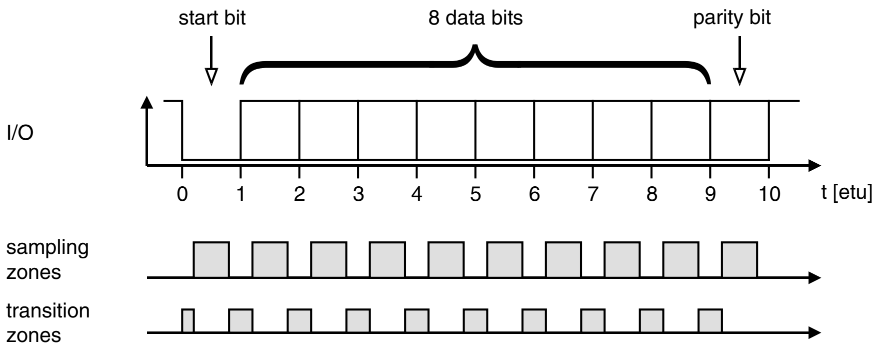


Figure 6.8 Test zones (sampling intervals) and transition zones for the reception of a data byte

interval and at both ends of the 'test zone', as defined by the applicable timing tolerances for byte transmission. The optimum sampling points can be defined by determining the boundaries of the test zone and the midpoint of the bit interval. However, these are not specified in any standard.[5] Sampling within the 'transition zone' is not allowed, since the signal level is invalid within this zone.

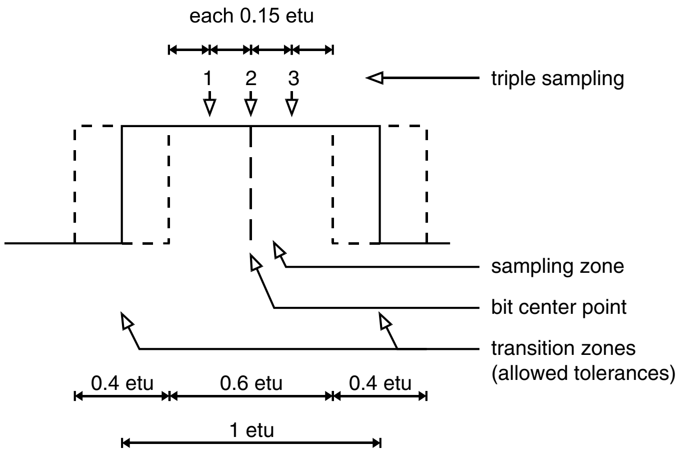


Figure 6.9 Example of threefold sampling of a received bit

6.2 ANSWER TO RESET (ATR)

After the supply voltage, clock signal and reset signal have been applied, the smart card sends an Answer to Reset (ATR) via the I/O lead. This data string, which contains at most 33 bytes,

[5] See also Section 16.11.4, 'Sampling times'

is always sent with a divider value (clock rate conversion factor) of 372 in compliance with the ISO/IEC 7816-3 standard. It contains various parameters related to the transmission protocol and the card. This divider value should be used even if the transmission protocol used after the ATR employs a different divider value (e.g. 64). This ensures that an ATR from any card can always be received, regardless of the parameters of the transmission protocol ultimately used.

It is very rare for an ATR to have the maximum allowable length. It most often consists of only a few bytes. Particularly in applications where the card should be usable very quickly after the activation sequence, the ATR should be short. A typical example is paying a road toll using a smart card electronic purse. Even if the vehicle passes through the toll gate quickly, it must be possible to reliably debit the card in the short time available.

The start of the ATR transmission must occur between 400 and 40,000 clock cycles after the terminal issues the reset signal. With a clock rate of 3.5712 MHz, this corresponds to an interval of 112 μs to 11.20 ms, while at 4.9152 MHz the interval is 81.38 μs to 8.14 ms.[6] If the terminal does not receive the start of the ATR within this interval, it repeats the activation sequence several times (usually up to three times) to try to detect an ATR. If all of these attempts fail, the terminal assumes that the card is faulty and reacts accordingly.

During the ATR, the time between the leading edges of two successive bytes may be up to 9600 etu according to ISO/IEC 7816-3. This period is designated the 'initial waiting time', and it is exactly one second at a clock rate of 3.5712 MHz. This means that the standard permits a one-second delay between the individual bytes of the ATR when it is sent to the terminal. In some smart card operating systems, this time is utilized for internal computations and EEPROM write accesses. The internal write buffer for atomic operations is often flushed at the same time.[7]

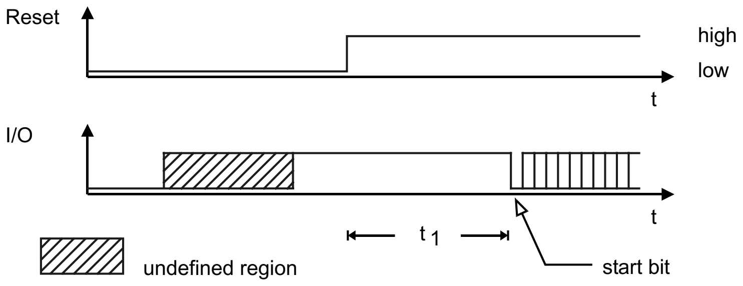


Figure 6.10 Timing diagram of the reset signal and the start of the ATR, in accordance with ISO/IEC 7816-3 (400 clock cycles $\leq t_1 \leq$ 40,000 clock cycles)

The data string and data elements of the ATR are defined and described in detail in the ISO/IEC 7816-3 standard. The basic ATR format is described in Figure 6.11 and Table 6.1. The first two bytes, designated TS and T0, define several fundamental transmission parameters and indicate the presence of subsequent bytes. The interface characters specify special transmission parameters for the protocol, which are important for the following data transmissions. The historical characters describe the extent of the smart card's basic functions. The check character,

[6] See also Section 16.11.2, 'ATR data element conversion tables'
[7] See also Section 5.10, 'Atomic Operations'

which is a checksum of the previous bytes, may optionally be sent as the last byte of the ATR, depending on the transmission protocol used.

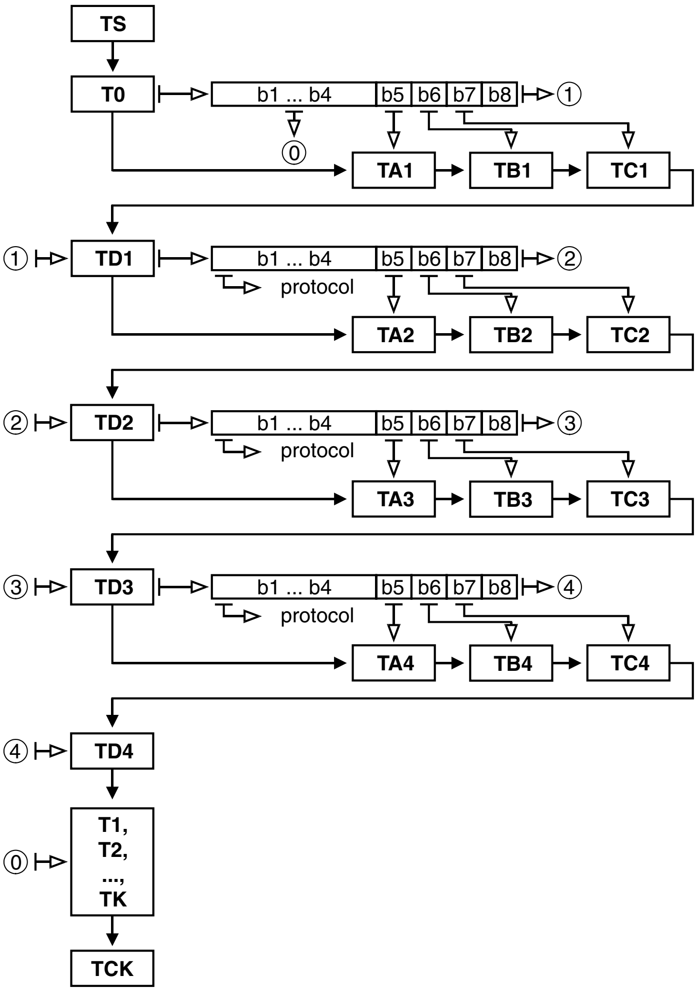


Figure 6.11 Basic structure and data elements of the ATR

6.2.1 ATR characters

The initial character

This byte, designated 'TS', specifies the convention used for all the data in the ATR and subsequent communications processes. In addition, the TS byte contains a characteristic bit pattern that can be used by the terminal to determine the value of the divider. For this purpose, the terminal can measure the time between the first two falling edges in TS and divide it by three. The result is the duration of one etu. However, since the divider is fixed at 372 for the

Table 6.1 The data elements of the ATR and their designations according to ISO/IEC 7816-3

Data element	Designation
TS	Initial character
T0	Format character
TA1, TB1, TC1, TD1, ...	Interface characters
T1, T2, ..., Tk	Historical characters
TCK	Check character

ATR, the terminal does not normally evaluate this synchronization pattern. The first byte is a mandatory part of the ATR and must always be sent. Only two codes are allowed for this byte: '3B' for the direct convention and '3F' for the inverse convention.

The direct convention is normally used in Germany, but the inverse convention is normally used in France. The convention does not affect the security of the transmission. Of course, every operating system producer prefers one or the other for historical reasons, but all terminals and many smart cards support both conventions.

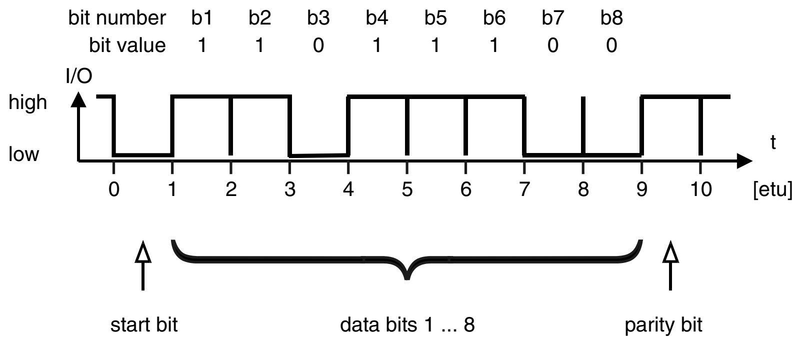

bit number b1 b2 b3 b4 b5 b6 b7 b8
bit value 1 1 0 1 1 1 0 0
I/O
high
low
t
0 1 2 3 4 5 6 7 8 9 10 [etu]
start bit
data bits 1 ... 8
parity bit

Figure 6.12 Timing diagram of the initial character TS using the direct convention, which is indicated by the value '3B' = °0011 1011°

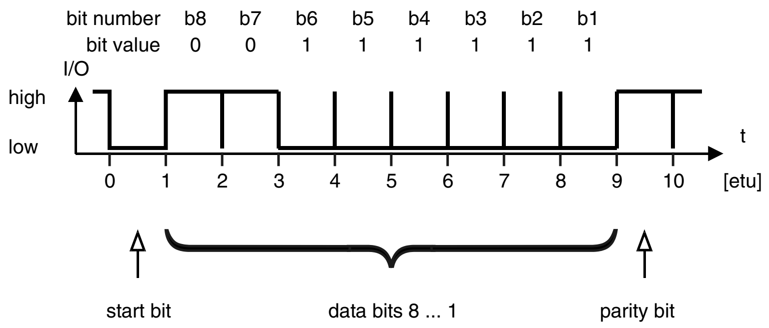

bit number b8 b7 b6 b5 b4 b3 b2 b1
bit value 0 0 1 1 1 1 1 1
I/O
high
low
t
0 1 2 3 4 5 6 7 8 9 10 [etu]
start bit
data bits 8 ... 1
parity bit

Figure 6.13 Timing diagram of initial character TS using the inverse convention, which is indicated by the value '3F' = °0011 1111°

Table 6.2 Coding of the initial character TS

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
'3B'								Direct convention
'3F'								Inverse convention

The format character

The second byte, T0, contains a bit field that indicates which interface characters follow it in the ATR. It also indicates the number of historical characters following the interface characters. Like TS, this byte must be present in every ATR.

Table 6.3 Coding of the format character T0

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
...	...	...	...	X				Number of historical characters (0 to 15)
...	...	...	1	...	...	...	...	TA1 sent
...	...	1	...	...	...	...	...	TB1 sent
...	1	...	...	...	...	...	...	TC1 sent
1	...	...	...	...	...	...	...	TD1 sent

The interface characters

The interface characters specify all of the transmission parameters for the current protocol. They consist of the TAi, TBi, TCi and TDi bytes. However, these bytes are optional in the ATR, and they may be omitted as appropriate. Since default values are defined for all of the parameters of the transmission protocol, interface characters are often not required in the ATR for a normal communications process.

The interface characters can be classified into global interface characters and specific interface characters. The global interface characters specify basic parameters for the transmission protocol, such as the divider, that apply to all subsequent protocols. The specific interface characters specify parameters for a particular transmission protocol. The 'work waiting time' for T = 0 is a typical example of such a parameter.

In principle, the global interface characters apply to all protocols, but for historical reasons (since originally only the T = 0 protocol was defined in the ISO standards), several of these characters are only relevant to the T = 0 protocol. If T = 0 is not used, they can be omitted, in which case the preset values apply.

Each TDi byte is only used to provide links to subsequent interface characters. For this purpose, the upper nibble of each TDi byte contains a bit pattern indicating which of the TA(i+1), TB(i+1), TC(i+1) and TD(i+1) interface characters follow it, using the same coding as for the T0 format character. The lower nibble of each TDi byte identifies the available transmission protocol in each case. A TDi byte must always be sent if any subsequent interface characters are to be sent.

Table 6.4 Coding of the TDi bytes

b8	b7	b6	b5	b4 b3 b2 b1	Meaning
...	...	...	...	X	Transmission protocol number (0−15)
...	...	...	1	...	TA(i+1) sent
...	...	1	...	...	TB(i+1) sent
...	1	...	...	...	TC(i+1) sent
1	...	...	...	...	TD(i+1) sent

The other interface characters (TAi, TBi and TCi), which are not used for linking, define the available transmission protocol(s). Their meanings according to the ISO/IEC 7816-3 standard are described below.

The global interface character TA1

The parameter FI in the upper nibble encodes the divider (clock rate conversion factor) F. The parameter DI in the lower nibble encodes the bit rate adjustment factor D.

Table 6.5 Coding of TA1

b8 b7 b6 b5	b4 b3 b2 b1	IFSC
X	...	FI
...	X	DI

Table 6.6 Coding of FI

F	372	372	558	744	1116	1488	1860	RFU
FI	0000	0001	0010	0011	0100	0101	0110	0111
f_{max}	4 MHz	5 MHz	6 MHz	8 MHz	12 MHz	16 MHz	20 MHz	...
F	RFU	512	768	1024	1536	2048	RFU	RFU
FI	1000	1001	1010	1011	1100	1101	1110	1111
f_{max}	...	5 MHz	7.5 MHz	10 MHz	15 MHz	20 MHz	...	...

Table 6.7 Coding of DI

D	RFU	1	2	4	8	16	32	RFU
DI	0000	0001	0010	0011	0100	0101	0110	0111
D	12	20	RFU	RFU	RFU	RFU	RFU	RFU
DI	1000	1001	1010	1011	1100	1101	1110	1111

The parameter codes for the divider F and bit rate adjustment factor D allow typical transmission rates to be specified in accordance with the standard. This is summarized in

Section 16.10.2, 'ATR parameter conversion tables'. The following relationships apply:

- The bit interval for the ATR and PPS, which is called the *initial etu*, is specified as

$$\textit{initial etu} = \frac{372}{f_\mathrm{i}}\ \mathrm{s}$$

- The bit interval for the transmission protocol used after the ATR and PPS can be defined independently of the ATR. This interval, which is called the *work etu*, is defined as

$$\textit{work etu} = \frac{1}{D}\frac{F}{f_\mathrm{s}}\ \mathrm{s}$$

The bit rate adjustment factor (D) and the clock rate conversion factor (F) allow the transmission rate to be modified and adapted to individual circumstances. The frequency of the applied clock (in units of Hertz) is shown in the above formulas as f. The value of the maximum allowable clock frequency is given by f_max. The standard value for f_max is 5 MHz.

The global interface character TA(i)

The value of TA(i) is always interpreted as XI || UI if i > 2 and T = 15 = 'F' in TD(i–1). In this case, TA(i) contains the clock stop indicator XI, which indicates the logical state the clock line must assume when the clock is stopped, and the class indicator UI, which specifies the supply voltage class.

Table 6.8 Coding of XI

X	Not supported	Low state	High state	No preferred state
XI	00	01	10	11

Table 6.9 Coding of UI

U	Voltage class A 4.5–5.5 V	Voltage class B 2.7–3.3 V	Voltage classes A & B	RFU
UI	00 0001	00 0010	00 0011	all other values

The global interface character TB1

Bits b7 and b6 of TB1 encode a programming voltage factor called 'II'. Bits b5–b1 define the parameter 'PI1'. The most significant bit, b8, is always set to 0, which effectively means that it is not used.

Table 6.10 Coding of TB1

b8	b7	b6	b5	b4	b3	b2	b1	IFSC
0	X		...	...	...	...	...	II
0	...		X	X	X	X	X	PI1

These parameters were needed for the first generation of smart cards, since they used EPROM for data storage instead of EEPROM, which is currently standard. The necessary high voltages and currents for EPROM programming had to be provided by the terminal via the Vpp contact. However, since smart cards without internal charge pumps no longer exist, the specific coding of this byte can be ignored. Parameters PI1 and II thus always have the value 0, which indicates that no external programming voltage is needed. If the TB1 parameter is omitted in the ATR, the default Vpp value of 5 V at 50 mA applies, as specified in the standard.

The global interface character TC1

TC1 encodes an extra guard time, designated N, as an unsigned hexadecimal integer. This extra guard time is defined as an extension to the duration of the stop bit. The value N indicates how many additional etu's are to be added to the guard time. TC1 is interpreted linearly except for N = 'FF', which has a special meaning. With the T = 1 protocol, the normal guard time of 2 etu is changed to 1 etu if N = 'FF'. With the T = 0 protocol, the standard guard time of 2 etu is retained in this case, to allow an error to be indicated by a low level within the guard time interval. In practice, reducing the guard time to 11 etu with the T = 1 protocol increases the effective data transmission rate by nearly 10 percent, since only 11 bits have to be sent for each character instead of 12.

Table 6.11 Coding of TC1

b8	b7	b6	b5	b4	b3	b2	b1	IFSC
				X				Extra guard time N, with a range of 0–254 etu
1	1	1	1	1	1	1	1	X = 255 and T = 0: guard time = 2 etu X = 255 and T = 1: guard time = 1 etu

The global interface character TB2

TB2 contains the value of PI2. This parameter specifies the external programming voltage in tenths of a volt. It is normally no longer used in the ATR, for the same reason as TB1.

Table 6.12 Coding of TB2

b8	b7	b6	b5	b4	b3	b2	b1	IFSC
				X				PI2

Specific interface character for the T = 0 transmission protocol

The specific interface character TC2

TC2 is the final parameter for the T = 0 protocol. It contains the parameter WI, which encodes the 'work waiting time'. This is the maximum interval between the leading edges of two consecutive bytes:

$$\textit{work waiting time} = (960 \cdot D \cdot \mathrm{WI})\ \text{work etu}$$

If the TC2 parameter is not present in the ATR, the default value of the work waiting time is used (WI = 10).

Table 6.13 Coding of TC2

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
				X				WI

Specific interface characters for the T = 1 transmission protocol

The following additional bytes are defined for the T = 1 transmission protocol in accordance with ISO/IEC 7816-3. In this case, the interface characters prescribed for T = 0 are used only as necessary. For this protocol, the parameter index i must always be greater than 2. The specific interface characters TAi, TBi and TCi ($i > 2$) always apply to the transmission protocol specified in TD(i – 1).

Specific interface character TAi (i > 2)

The TAi byte contains the maximum length of the information field that can be received by the card (IFSC). This value must be in the range of 1 through 254. The default value of IFSC is 32 bytes.

Table 6.14 Coding of TAi for $i > 2$

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
				X				IFSC

Specific interface character TBi (i > 2)

The lower nibble (bits b4 b1) contains the code CWI for the character waiting time CWT,[8] which is calculated as

$$CWT = (2^{CWI} + 11)\ \text{work etu}$$

[8] See also Section 6.4.3, 'The T = 1 transmission protocol'

The upper nibble holds the value BWI, with which the block waiting time BWT[9] can be calculated as follows:

$$BWT = 2^{BWI} \cdot 960 \cdot \frac{372}{f} \text{s} + 11 \text{ work etu}$$

Table 6.15 Coding of TBi for $i > 2$

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
		...				X		CWI
		X				...		BWI

Specific interface character TCi (i > 2)

Bit b1 encodes the method used to compute the error detection code.

Table 6.16 Coding of TCi for $i > 2$

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
...	...	...	...	...	...	...	0	LRC is used
...	...	...	...	...	...	...	1	CRC is used
0	0	0	0	0	0	0	...	Reserved for future use

Since the standard that defines the ATR parameters does not specify all possible transmission protocol parameters in terms of interface characters, specific implementations may use a variety of supplementary interface characters. A typical example is provided by the German national T = 14 protocol. Several additional ATR bytes are defined for this protocol to meet its specific needs. These can be decoded only by users of this protocol, since only they know the applicable specification. This is not standardized, nor is it made known outside of the applications that use it.

Global interface character TA2

This byte indicates the allowed modes for the PPS. This is explained in more detail in Section 6.4, which describes the PPS.

The historical characters

For a long time, the historical characters were not defined by any standard. As a result, they contain a wide variety of information, depending on the producer of the operating system.

[9] See also Section 6.4.3, 'The T = 1 transmission protocol'

Table 6.17 Coding of TA2

b8	b7	b6	b5	b4 b3 b2 b1	Meaning
0	...	...	...	...	Switching between negotiable mode and specific mode is possible.
1	...	...	...	...	Switching between negotiable mode and specific mode is not possible.
...	0	0	...	...	Reserved for future use.
...	...	...	0	...	Transmission parameters explicitly defined in the interface characters.
...	...	...	1	...	Transmission parameters implicitly defined in the interface characters.
...	...	...	...	X	Protocol T = X is to be used.

Many companies use the available bytes to identify the operating system and the associated version number of the ROM mask. This is usually encoded in ASCII, so it is easy to interpret. Historical characters are not required to be present in the ATR, so they may be entirely omitted. In some situations, this can be beneficial, since it makes the ATR shorter and thus quicker to send.

The ISO/IEC 7816-4 standard provides for an ATR file in addition to the historical characters. This file, with the reserved FID '2F01', contains additional data for the ATR. It is intended to be an extension to the historical characters, which are limited to 15 bytes. The content of this file, whose structure is not defined by the standard, is ASN.1-coded.

The parameters in the ATR file or the historical characters may contain complex information relating to the smart card and the operating system used in the card. For example, they can indicate which file selection and implicit selection functions are supported by the smart card and provide information about the logical channel mechanism. They can also hold additional information about the card issuer, the card and chip serial numbers, the ROM mask version, the chip and the operating system. The coding of the relevant data objects is defined in the ISO/IEC 7816-4 and 7816-5 standards.

According to ISO/IEC 7816-4, the historical characters can contain the following three data fields: an obligatory category indicator, one or more optional data blocks in compact TLV format and an optional status indicator. The compact TLV format has a tag in the first nibble and the length of the following data in the second nibble.

The category indicator is transferred in T1. It contains information about the structure of the data in the ATR. The data following the category indicator include information about the services supported by the smart card operating system and the operating system functions. The most important of these items are described in Tables 6.19 through 6.22. The status indicator indicates the life-cycle stage of the smart card.

Table 6.18 Coding of the category indicator in T1

Value	Meaning
'00'	Status information is located at the end of the historical characters
'10'	Reference to a DIR file (directory file)
'80'	Status information, if present, is contained in a compact TLV-coded data object
'81'...'8F'	RFU
all other values	Application-specific (proprietary)

Table 6.19 Coding of the application-independent card services using the compact TLV-coded data object '31' (= tag \|\| length)

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	...	...	...	...	...	...	...	Application selection using the full DF name
...	1	...	...	...	...	...	...	Application selection using a partial DF name
...	...	1	...	...	...	...	...	Data objects are available in the DIR file
...	...	...	1	...	...	...	...	Data objects are available in the ATR file
...	...	...	...	1	...	...	...	Data objects can be read from the DIR or ATR file using the READ BINARY command
...	...	...	...	0	...	...	...	Data objects can be read from the DIR or ATR file using the READ RECORD command
...	...	...	...	...	0	0	0	Not used

Table 6.20 Coding of the first software function table in compact TLV-coded data objects. The combined tag/length byte can have a value of '71', '72' or '73', depending on the number of data bytes

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	...	...	...	...	...	...	...	DF selection using the full DF name
...	1	...	...	...	...	...	...	DF selection using a partial DF name
...	...	1	...	...	...	...	...	DF selection by specifying a path
...	...	...	1	...	...	...	...	DF selection using an FID
...	...	...	...	1	...	...	...	Implicit DF selection (with the desired application)
...	...	...	...	...	1	...	...	Short FIDs are supported
...	...	...	...	...	...	1	...	Record numbers are supported
...	...	...	...	...	...	...	1	Record identifiers are supported

The check character

This last byte in the ATR is the check character (TCK), which contains the XOR checksum of the bytes from T0 through the last byte before the check character. This checksum can be used in addition to parity testing to verify the correctness of the ATR transmission. However,

Table 6.21 Coding of the second software function table in compact TLV-coded data objects. The combined tag/length byte can have a value of '71', '72' or '73', depending on the number of data bytes

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
..	0	0	...	...	...	...	...	The WRITE command acts as a one-time write
...	0	1	...	...	...	...	...	WRITE command behavior is application-specific
...	1	0	...	...	...	...	...	The WRITE command acts as a logical OR
...	1	1	...	...	...	...	...	The WRITE command acts as a logical AND
...	...	...	...	1	...	...	...	Implicit DF selection (with the desired application)
...	...	...	...	...	x	x	x	Size of the data units in nibbles modulo 2 (°001° = 2^2 = 2 nibbles = 1 byte)
0	...	...	0	0	...	...	...	Not used

Table 6.22 Coding of the third software function table in compact TLV-coded data objects. The combined tag/length byte can have a value of '71', '72' or '73', depending on the number of data bytes

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
...	1	...	...	...	...	...	...	Extended Lc and Le fields
...	...	...	1	0	...	...	...	Logical channels are assigned by the card when the MANAGE CHANNEL command is used
...	...	...	0	1	...	...	...	Logical channels are assigned by the terminal when the MANAGE CHANNEL command is used
...	...	...	0	0	...	...	...	Logical channels are not supported
...	...	...	...	...	...	X	Y	Maximum number of logical channels [2 × (X + Y + 1)]
0	...	0	...	...	0	...	...	Not used

despite the apparent simplicity of the structure and computation of this checksum, there are several significant differences among various transmission protocols.

If only the T = 0 transmission protocol is indicated in the ATR, no checksum is allowed to not be present at the end of the ATR. In this case, it is not sent at all, since bytewise error detection using parity checking and retransmission of erroneous bytes is mandatory in the T = 0 protocol. By contrast, a TCK byte must be present in the T = 1 protocol. The checksum is then computed starting with byte T0 and ending with the final interface character, or with the final historical character if historical characters are present.

6.2.2 Practical examples of ATRs

Tables 6.23 through 6.28 show practical examples of various types of smart card ATRs. They are very useful as an aid to interpreting ATRs or defining your own ATRs.

Table 6.23 Sample smart card ATR with T = 1

Designation	Value	Meaning		Remark
TS	'3B'	direct convention		
T0	'B5'	Y1	= 'B' = °1011°	TA1, TB1 and TD1 follow
		K	= '5'	5 historical characters
TA1	'11'	FI	= °0001° = '1'	F = 372
		DI	= °0001° = '1'	D = 1
TB1	'00'	II	= 0	I = 0
		PI1	= °0000° = '0'	Vpp contact not used
TD1	'81'	Y2	= '8' = °1000°	TD2 follows
		T	= 1	
TD2	'31'	Y2	= '3' = °0011°	TA3 and TB3 follow
		T	= 1	
TA3	'46'	I/O buffer size = 70 bytes		ICC I/O buffer size (layer 7)
TB3	'15'	BWI	= '1'	BWT = 2011 etu
		CWI	= '5'	CWT = 43 work etu
T1	'56'	"V"		"V 1.0"
T2	'20'	" "		
T3	'31'	"1"		
T4	'2E'	"."		
T5	'30'	"0"		
TCK	'1E'	check character		XOR checksum of T0 through T5

Table 6.24 Sample ATR for a STARCOS smart card with T = 1 and the direct convention, with the operating system not yet completed

Designation	Value	Meaning		Remark
TS	'3B'	direct convention		
T0	'9C'	Y1	= '9' = °1001°	TA1 and TD1 follow
		K	= 'C' = 12	12 historical characters
TA1	'11'	FI	= °0001° = '1'	F = 372
		DI	= °0001° = '1'	D = 1
TD1	'81'	Y2	= '8' = °1000°	TD2 follows
		T	= 1	T = 1 is used
TD2	'21'	Y2	= '2' = °0010°	TB3 follows
		T	= 1	T = 1 is used
TB3	'34'	CWI	= '4'	Character waiting time
		BWI	= '3'	Block waiting time
T1 ... T12	'53' || '43' || '20' || '53' || '56' || '20' || '31' || '2E' || '31' || '20'|| '4E' || '43'			"SC SV 1.1 NC"
TCK	'0F'	check character		XOR checksum of T0 through T12

Table 6.25 Sample ATR for a STARCOS smart card with T = 1 and the direct convention, with the operating system completed

Designation	Value	Meaning		Remark
TS	'3B'	direct convention		
T0	'9B'	Y1	= '9' = °1001°	TA1 and TD1 follow
		K	= 'B' = 11	11 historical characters
TA1	'11'	FI	= °0001° = '1'	F = 372
		DI	= °0001° = '1'	D = 1
TD1	'81'	Y2	= '8' = °1000°	TD2 follows
		T	= 1	T = 1 is used
TD2	'21'	Y2	= '2' = °0010°	TB3 follows
		T	= 1	T = 1 is used
TB3	'34'	CWI	= '4'	Character waiting time
		BWI	= '3'	Block waiting time
T1 ... T12	'53' || '54' || '41' || '52' || '43' || '4F' || '53' || '20' || '32' || '31' || '43'			"STARCOS 21C"
TCK	'43'	check character		XOR checksum of T0 through T12

Table 6.26 Sample ATR for a GSM smart card with T = 0 and the direct convention

Designation	Value	Meaning		Remark
TS	'3B'	direct convention		
T0	'89'	Y1	= '8' = °1000°	TD1 follows
		K	= 9	9 historical characters
TD1	'40'	Y2	= '4' = °0100°	TC2 follows
		T	= 0	T = 0 is used
TC2	'14'	WI	= '14'	The work waiting time is '14'
T1 ... T9	'47' || '47' || '32' || '34' || '4D' || '35' || '32' || '38' || '30'			"GG24M5280"

Table 6.27 Sample ATR for a GSM smart card with T = 0 and the inverse convention

Designation	Value	Meaning		Remark
TS	'3F'	inverse convention		
T0	'2F'	Y1	= '2' = °0010°	TB1 follows
		K	= 'F' = 15	15 historical characters
TB1	'00'	PI1	= °00000°	Vpp is not used; the programming voltage for the EEPROM is generated in the chip
		II	= °00°	
T1 ... T15	'80' || '69' || 'AE' || '02' || '02' ||'01'|| '36' || '00' || '00' || '0A' || '0E' || '83' || '3E' || '9F' || '16'			Individual data of the smart card manufacturer

Table 6.28 Sample ATR for a Visa Cash smart card with T = 1 and the direct convention

Designation	Value	Meaning		Remark
TS	'3B'	direct convention		
T0	'E3'	Y1	= 'E' = °1110°	TB1, TC1 and TD1 follow
		K	= 3	3 historical characters
TB1	'00'	PI1	= °00000°	Vpp not used; EEPROM programming
		II	= °00°	voltage generated internally
TC1	'00'	N	= 0	No extra guard time
TD1	'81'	Y2	= '8' = °1000°	TD2 follows
		T	= 1	T = 1 is used
TD2	'31'	Y2	= '3' = °0011°	TA3 and TB3 follow
		T	= 1	T = 1 is used
TA3	'6F'	IFSC	= '6F' = 111	The information field size of the smart card is 111 bytes
TB3	'45'	CWI	= '5'	Character waiting time
		BWI	= '4'	Block waiting time
T1	'80'	category indicator		Data object in the compact TLV format follows (compliant with ISO/IEC 7816-4)
T2	'31'	card service data		Label ('3') and length ('1') of the data object for the card service data
T3	'C0'	'C0' = °1100 0000°		Application selection by the complete or partial DF name
TCK	'08'	check character		XOR checksum of T0 through T3

6.3 PROTOCOL PARAMETER SELECTION (PPS)

The smart card specifies various data transmission parameters in the interface characters of the ATR, such as the transmission protocol and the character waiting time. If a terminal wants to modify one or more of these parameters, it must perform a protocol parameter selection (PPS) procedure in accordance with ISO/IEC 7816-3 before the transmission protocol is actually used. This allows the terminal to modify certain protocol parameters if this is permitted by the card. Prior to 1997, protocol parameter selection was called protocol type selection (PTS).

PPS can be performed in two different modes. In the negotiable mode, the standard values of the divider *F* and the bit rate adjustment factor *D* remain unchanged until a PPS is successfully executed. If the card uses the specific mode, the values of *F* and *D* specified by the ATR must be used for transmitting the PPS. The card indicates which of these two modes it supports in the TA2 byte.

The PPS request must be made immediately after the ATR has been received by the terminal. If the card allows the requested changes to the protocol parameters, it sends the received PPS bytes back to the terminal. In principle, this is an echo of the received data. Otherwise, the card sends nothing, and the terminal must perform a new reset sequence to cause the card to exit this state. PPS may be performed only once, immediately after the ATR. Repeated transmission of the PPS is prohibited by ISO/IEC 7816-3. In practice, it is very rarely necessary to perform a

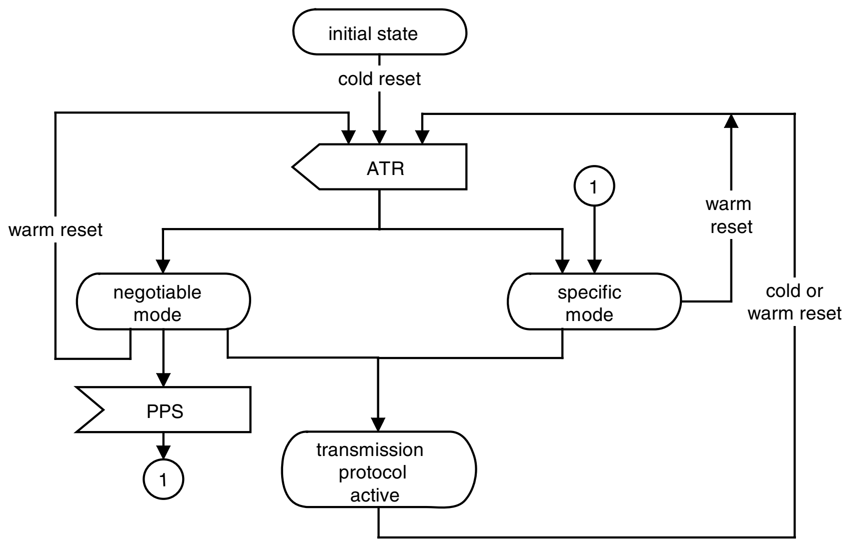


Figure 6.14 State diagram of the two PPS modes (negotiable mode and specific mode) according to ISO/IEC 7816-3

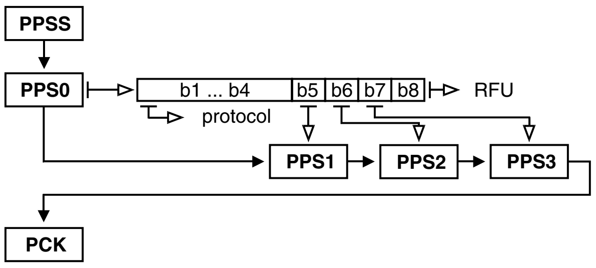


Figure 6.15 The basic structure and data elements of the PPS

PPS, since the transmission parameters of the smart cards being used are exactly matched to the requirements of the terminal.

Table 6.29 PPS data elements and their designations according to ISO/IEC 7816-3

Data element	Designation
PPSS	Initial character
PPS0	Format character
PPS1, PPS2, PPS3	Parameter characters
PCK	Check character

The first byte is the initial character (PPSS), which unambiguously informs the card that the terminal is initiating a PPS request immediately after the ATR. It always has the value 'FF' and is a required component of each PPS.

The data element following the PPSS is the format character (PPS0). It is also a required component of each PPS. It may optionally be followed by up to three bytes, which are called the parameter characters and are designated PPS1, PPS2 and PPS3. They encode various parameters for the transmission protocol to be used following the PPS. Data element PPS3 is reserved for future use, so it cannot yet be described here. The final byte of the PPS is called the control character (PCK). It contains the XOR checksum of all previous bytes, starting with PPSS. Unlike the other data elements, which are optional, PCK is a mandatory component of the PPS, as are PPSS and PPS0.

Table 6.30 Coding of PPS0

b8	b7	b6	b5	b4 b3 b2 b1	Meaning
...	...	...	...	X	Transmission protocol to be used
...	...	...	1	...	PPS1 is present
...	...	1	...	...	PPS2 is present
...	1	...	...	...	PPS3 is present
0	...	...	...	...	Reserved for future use

Table 6.31 Coding of PPS1

b8	b7 b6 b5 b4 b3 b2 b1	Meaning
X	...	FI
...	X	DI

If the card can interpret the PPS and modify the transmission protocol accordingly, it acknowledges this by sending the received PPS back to the terminal. If the PPS request contains items that the card cannot process, it simply waits until the terminal executes a reset. The main disadvantage of this procedure is that a large amount of time can be lost before the actual transmission protocol is used.

The PPS just described cannot be used for protocol switching with terminals that cannot execute a PPS but that nonetheless have their own special transmission protocols. This is precisely the situation with German card phones, for example. A special procedure has been devised to permit protocol switching despite this limitation. Since all terminals perform multiple reset sequences if they do not recognize the ATR, it was decided that the smart card should switch transmission protocols after every reset. This can best be illustrated by an example. After the first reset, the card sends the ATR for T = 14 and is then ready to communicate using the T = 14 protocol. After the second reset, it sends an ATR for T = 1 and is then ready to communicate using the T = 1 protocol. After the third reset, it is again prepared to use the T = 14 protocol. This is not an ideal technical solution, since a device should always behave the same after every reset, but it is a fully practical solution for a heterogeneous terminal world.

It is possible to mitigate this objection by having the smart card always respond with the same ATR after a power-on reset (cold reset). In this case, the card always executes a cold reset directly after it has been inserted in the card reader and the activation sequence has been completed. A reset that it triggered via the card's reset line (warm reset), on the other hand, switches the transmission protocol. The card thus behaves the same after every 'real' reset, while any supplementary triggering of a reset causes it to switch to a different transmission protocol.

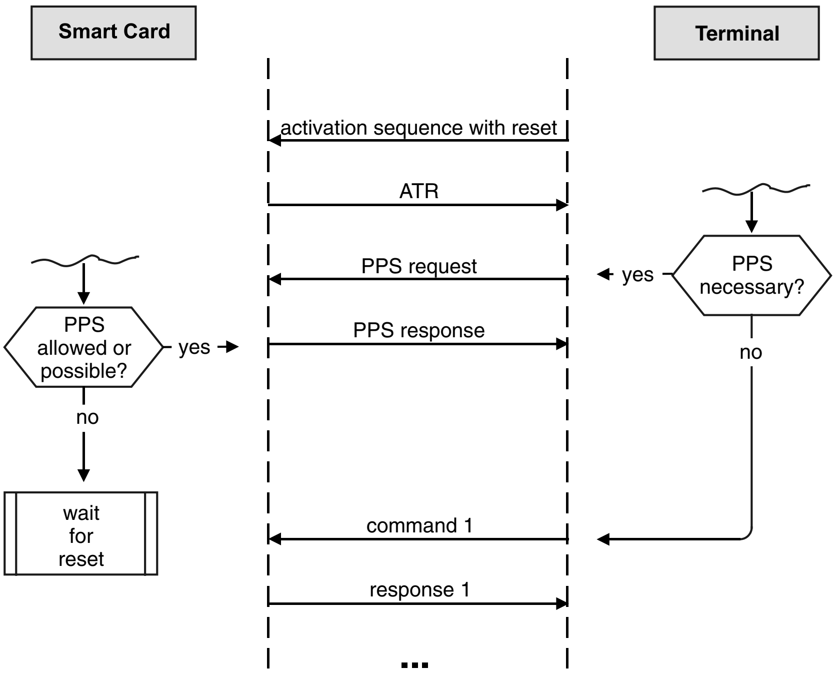


Figure 6.16 A typical PPS procedure with a GSM card

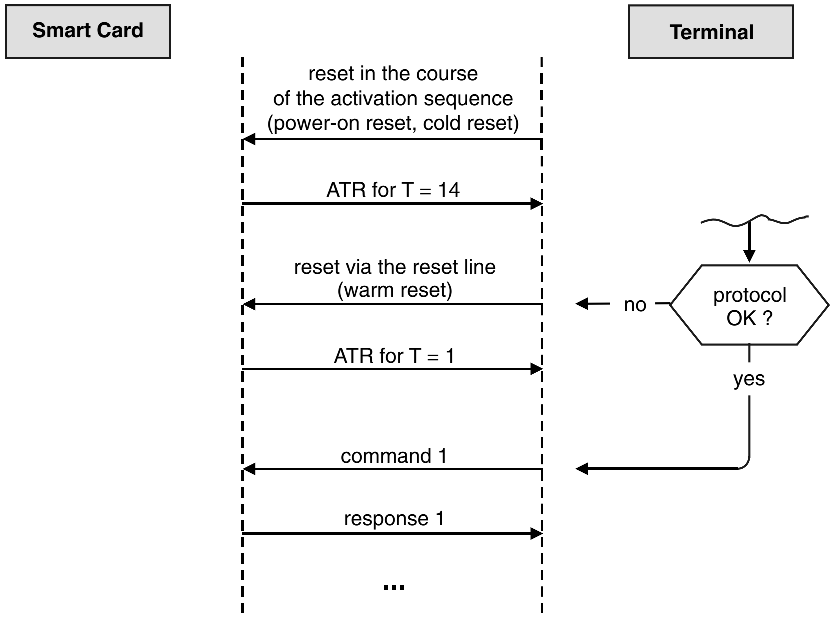

Smart Card
Terminal
reset in the course of the activation sequence (power-on reset, cold reset)
ATR for T = 14
reset via the reset line (warm reset)
protocol OK ?
no
yes
ATR for T = 1
command 1
response 1
...

Figure 6.17 A typical PPS procedure triggered by a reset

6.4 DATA TRANSMISSION PROTOCOLS

After the smart card has sent an ATR, possibly followed by a PPS, it waits for the first command from the terminal. The subsequent process always corresponds to the master–slave principle, with the terminal as master and the card as slave. In concrete terms, the terminal sends a command to the card, and the latter executes it and subsequently returns a response. This back-and-forth interplay of commands and responses never changes.

There are various ways in which communication with a smart card can be established. There are also a number of different methods for resynchronizing communications if a disturbance occurs. The exact implementation of the commands, the corresponding responses and the procedures used in the event of transmission errors are defined in the form of transmission protocols.

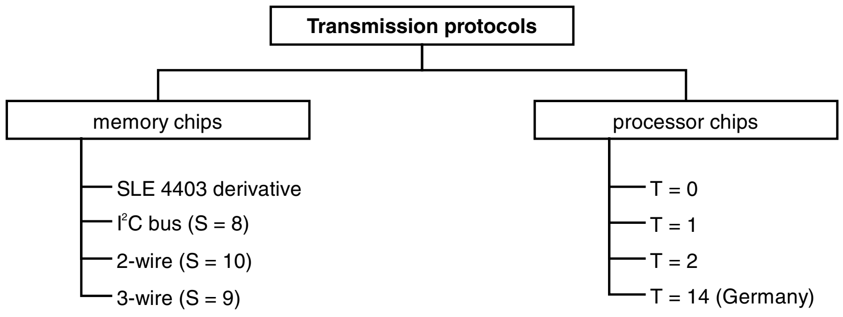


Figure 6.18 Classification of transmission protocols used with contact-type smart cards

A total of 15 transmission protocols have been identified and defined in terms of their basic functions. These protocols, which are designated as 'T =' (for 'transmission protocol') plus a sequential number, are summarized in Table 6.32.

Table 6.32 Summary of transmission protocols according to ISO/IEC 7816-3

Protocol	Meaning
T = 0	Asynchronous, half-duplex, byte oriented, specified in ISO/IEC 7816-3
T = 1	Asynchronous, half-duplex, block oriented, specified in ISO/IEC 7816-3 Amd. 1
T = 2	Asynchronous, full duplex, block oriented, specified in ISO/IEC 10536-4
T = 3	Full duplex; not yet specified.
T = 4	Asynchronous, half-duplex, byte oriented, extension of T = 0, not yet specified
T = 5 ... T = 13	Reserved for future use, not yet specified
T = 14	For national use, not standardized by ISO
T = 15	Reserved for future use and not yet specified

Two of these protocols predominate in international use. The first is the T = 0 protocol, which became an international standard in 1989 (ISO/IEC 7816-3). The other is T = 1, which was introduced in 1992 in an amendment to an international standard (at the time ISO/IEC 7816-3 Amd. 1, now ISO/IEC 7816-3). The full-duplex transmission protocol T = 2, which is strongly based on T = 1, is currently in the definition stage and will be available as an international standard in a few years.

In Germany, the widely distributed card-phone system uses yet a third protocol, designated T = 14. It is defined in an internal specification of Deutsche Telekom.

The data elements transported by transmission protocols are called transmission protocol data units (TPDUs). They can be regarded as protocol-dependent containers that transport data to and from the card. The actual application data are embedded in these containers.

In addition to the technically complex smart card transmission protocols, there is an additional set of very simple synchronous protocols for memory cards. They are typically used with telephone cards, health insurance cards and the like. However, they do not have error-correction mechanisms, and they are based on hard-wired logic in the chip.

6.4.1 Synchronous data transmission

Synchronous data transmission is not used with microcontroller-based smart cards, since they only communicate with the terminal asynchronously. However, it is the standard method for memory cards, which are used in very large numbers in applications such as prepaid electronic purses for card phones. This widespread use justifies a description of the operation of synchronous data transmission.

In memory cards, synchronous data transmission is very closely linked to the chip's hardware and is designed to be as simple as possible. There is no separation of layers in the transmission protocol, nor is logical addressing present, so the application in the terminal must directly access memory locations in the chip. The protocol allows the data stored in the chip to be physically addressed and then read or written. This means that the actual data transmission process is also tied to the functions of memory addressing and management.

There is also no procedure for detecting or correcting errors during data transmission, although it must be said that such errors between the card and the terminal occur very rarely. If the terminal application nonetheless detects a transmission error, it must re-read the relevant area in the card's memory. All these restrictions serve to allow data to be transmitted between the card and the terminal at a high rate using only a small amount of logic circuitry.

Since synchronous data transmission is only used to keep data transmission as simple as possible (which means using a minimum amount of logic circuitry), an almost inevitable consequence is strong hardware dependence. As a result, synchronous transmission protocols are not uniform and sometimes vary greatly from chip to chip. Only the ATR is standardized. A terminal that has to communicate with various types of memory cards must incorporate several different types of synchronous data transmission protocols.

The exact designation of the type of data transmission used for memory cards is 'clock-synchronous serial data transmission'. This clearly indicates the basic conditions that apply to this type of communication. As with asynchronous communications, data are transmitted between the card and the terminal serially, or bit by bit. However, the bits are sent synchronous to a supplementary transmit clock signal. This makes the transmission of start and stop information unnecessary.

In the case of a simple memory card, there is also no error detection information, which means that neither a parity bit nor a supplementary checksum is sent. The low probability of transmission errors is due to the very low clock rate, which ranges from 10 kHz to 100 kHz. Since one bit is sent for each clock cycle, a clock rate of 20 kHz yields a transmission rate of 20 kbit/s. However, the effective data rate is lower, since additional address information must also be sent in the case of memory cards.

In order to describe synchronous data transmissions for memory cards in an understandable manner, we first need to describe some of the basic features of memory cards. In their simplest form, these cards have memories that are divided into two parts, consisting of an unchangeable ROM region and an EEPROM region that can be written and erased. Both regions are bit-addressable and can be freely read, and the EEPROM can also be written and erased.

The master–slave relationship is even more pronounced in memory cards than in micro-controller smart cards. For example, the terminal completely takes over physical addressing of memory. The card itself can only globally block certain areas against erasing. This is controlled by hard-wired logic located ahead of the memory. This logic also manages the very simple data transmission process.

6.4.1.1 The telephone chip protocol

Data transmission is illustrated here using a phone card containing an Infineon SLE4403 chip as an example. The memory in this IC is bit-oriented, which means that all operations are carried out on individual bits. Other types of chips may have protocols that differ from the protocol described here. However, the basic data transmission principles are the same for all synchronous cards.

Data are transmitted using three leads. The bidirectional data lead is used by both the card and the terminal to exchange single-bit data. The clock lead transmits the clock generated by the terminal to the card. This clock provides the reference for the synchronous data transmission. The third connection needed for the data transmission is the control lead. It determines what the chip actually does, based on the states of the other two leads.

In principle, complete control of a memory card requires the chip's logic circuitry to decode four different functions. These are *read*, *write*, *clear memory* and *increment the address pointer*. A memory card has a global memory pointer that can be used to address all memory regions bit by bit. If the pointer reaches the upper memory boundary, it rolls over to zero. With a bit-oriented chip design, it then points to the first bit in memory. One of the functions of synchronous data transmission is to reset this pointer to an initial value, which is normally zero. The next function is to read data from the memory. The other two functions are writing and erasing EEPROM bits. Erasing EEPROM bits, which would allow them to be rewritten, is of course blocked in phone cards, since otherwise the cards could be reloaded.

Resetting the address pointer

The address pointer is reset to its initial value of zero by the power-up logic of the card if the clock lead and the control lead are simultaneously at a high level. However, the control pulse must be applied for a somewhat longer interval than the clock pulse, in order to prevent the address from being immediately incremented. The address pointer should be reset to its original value after each activation sequence, since it would otherwise be pointing to an undefined address.

Incrementing the address pointer and reading data

If there is a rising edge on the clock line while the control lead is at a low level, the internal logic of the card increments the address pointer by one. The falling edge of the clock causes

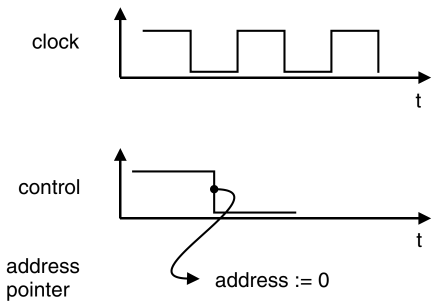


Figure 6.19 Resetting the address pointer to zero

the content of the address indicated by the pointer to be placed on the data lead. If the pointer reaches its maximum value, which depends on the size of the memory, it rolls over to zero and thus starts over from the beginning.

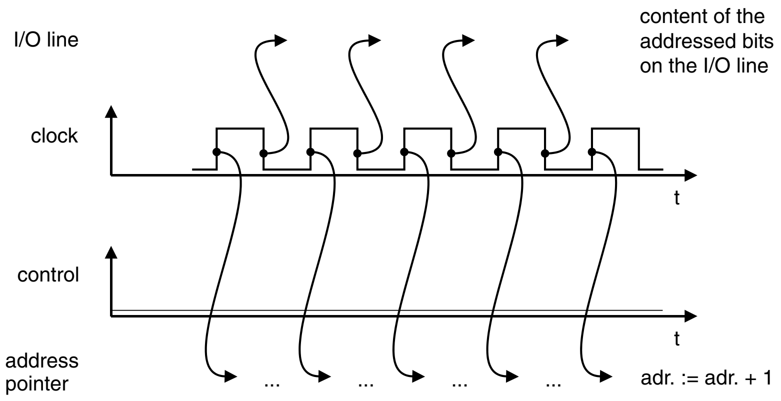


Figure 6.20 Incrementing the address pointer and reading data from an address

Writing to an address

If the address pointer is within a writeable EEPROM region, the value on the data lead can be written to EEPROM by applying a high level to the control lead and then pulling the clock lead low. The length of the write cycle is determined by the duration of the immediately following clock pulse. If the bit was written correctly, the content of the written memory cell appears at the data output.

Erasing bytes

Part of the EEPROM memory in a typical phone card is always organized as a multi-place octal counter. If a byte has to be erased in this counter due to a carry to the next place, this is performed by the logic circuitry. Erasing a byte in memory is thus somewhat more complicated. The procedure is as follows: if a bit within a byte is written twice in a row, the chip's hardware logic

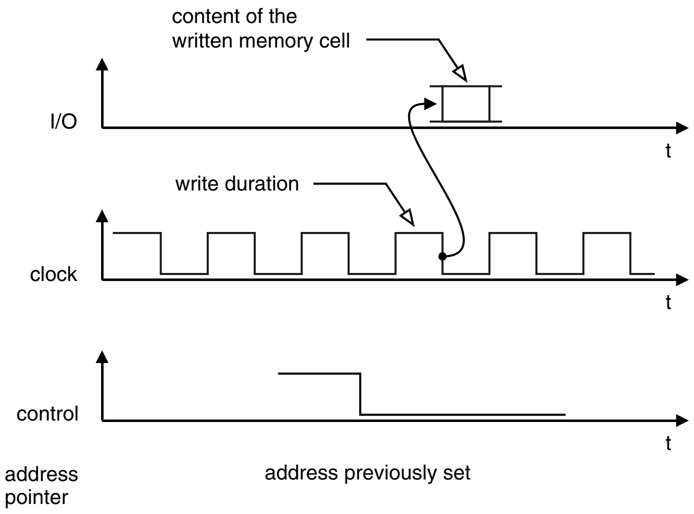


Figure 6.21 Writing a bit to an address in EEPROM memory

automatically erases the associated less significant byte. This ensures that a carry to the next higher place occurs, while the lower place is erased without allowing any opportunity for fraud.

The four types of access that have been described may vary from chip to chip and also among manufacturers. Another type of data transmission, which by contrast is standardized, is represented by the I^{2}C bus. Many of the newer memory cards use this bus for communicating with the terminal. This naturally has the advantage that different chips, made by different manufacturers, can be used together within a single system. Problems due to several different transmission protocols are thus eliminated, since all chips are mutually compatible at the transmission interface.

6.4.1.2 The I^{2}C bus

Since serial, clock-synchronized data transmission protocols are uncomplicated and versatile, they are used relatively frequently. Components for use with the I^{2}C (inter-integrated circuit) bus, which was developed by Philips, have been available since 1990. This bus is based on a bidirectional serial data lead and a serial clock lead. The definition of the I^{2}C bus encompasses both the hardware (the two leads) and the software, in the sense of the data transmission format. Each IC on the bus can take control of the bus and send requests to other ICs connected to the bus.

Since memory cards are also controlled by a synchronous clock, the I^{2}C bus has very quickly established itself in the smart card industry. A wide range of memory ICs has become available for use in cards. The following example is based on the SGS-Thomson ST24C04 memory chip. It has a 512-byte EEPROM that can be freely written and read. Timing computations for EEPROM programming are handled internally by the chip, so this does not have to be controlled externally.

The hardware for the I^{2}C bus consists of two lines between the terminal and the card. The serial clock (SCL) line carries the clock, which can range up to 100 kHz. This yields a data transmission rate of up to 100 kbit/s, which is relatively high for smart cards. The other line, serial data (SDA), is used bidirectionally to exchange data between the card and the terminal. The SDA line is connected to the supply voltage (Vcc) in the terminal via a pull-up resistor. Both communicating parties can only pull this line to ground. Sending a high level is therefore passive, and involves the sender switching its output to a high-impedance state (tri-state), thus allowing the pull-up resistor to pull the SDA line up to the supply voltage level.

In the smart card context, the terminal is always the master of the I^{2}C bus and the card is always the slave. Data transmission always uses single-byte packets. The most significant bit (bit 8) of the byte is sent first. Each transmission over the SDA line is initiated by a start signal and terminated by a stop signal. The start signal consists of a falling edge on the SDA line while the level on the SCL line is high. Conversely, a rising edge on the SDA line while the level on the SCL line is high indicates a stop signal. The recipient must acknowledge receipt of each byte by pulling the SDA line low for one clock cycle.

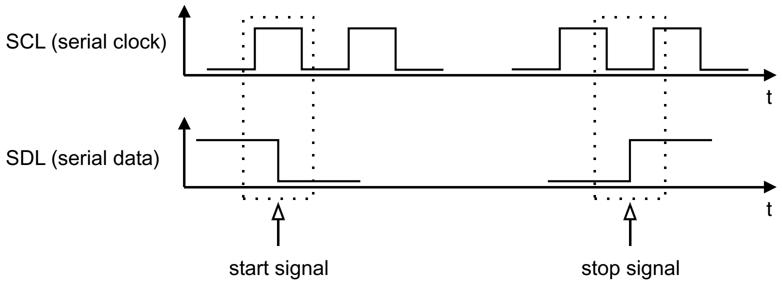


Figure 6.22 Start and stop signals on the I^{2}C bus

The first seven bits of the first byte after the start of communications are the address of the recipient. In our example, we assume for simplicity that the address has the binary value °1010000x°. Of course, this may vary depending on the chip type, and it can be chosen within certain limits for some memory ICs. The last bit in the address (x) tells the recipient whether data are to be read or written. A 1 indicates reading, while 0 is for writing.

The following examples illustrate the general functions of the I^{2}C bus as used with smart cards.

Reading from an address

There are several types of access for reading the EEPROM of a smart card. In the type described here, one byte is read at a time. However, it is also possible to read several bytes in succession.

The read sequence is initiated by the start signal. The subsequent bits contain the address of the card, with the control bit set to 'write'. This indicates to the card that it must temporarily store the following data in an internal buffer. This buffer is nothing more than a byte-oriented address pointer for the EEPROM. After the card receives the first byte, it sends an acknowledgement by grounding the SDA line for one clock cycle. After this, the terminal sends the EEPROM address to the card. Once again, the card acknowledges receipt of the data. The terminal then sends a start signal and the card address with the read bit set. On receiving this, the card sends the data from the location addressed by the pointer to the terminal. The terminal does not have to acknowledge the receipt of the data; it only sends a stop signal to the card. This completes the read sequence for one byte.

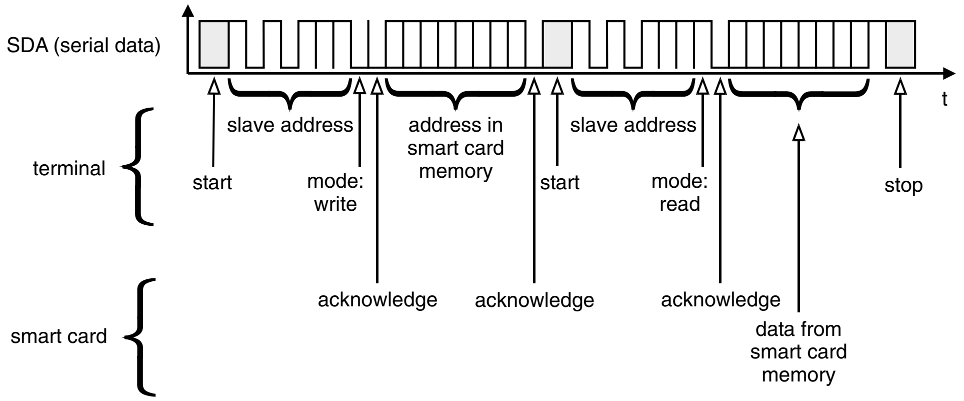


Figure 6.23 Unconstrained reading of a byte from memory using the I^{2}C bus

Writing to an address

As with reading data from the card's EEPROM, there are also various modes for writing data. The simplest mode, which can be used to write a single byte anywhere in memory, is described here.

Again, the sequence begins with a start signal from the terminal. This is followed by the card's address, with the write bit set. The card acknowledges receipt and then receives from the terminal the address in the EEPROM where the data are to be written. The card acknowledges this as well, and then receives the data. After the terminal receives the third acknowledgement, which indicates that the card has received the data, it sends a stop signal. Following this, the card starts to write the received data to the EEPROM, which does not require external timing signals. This completes the writing sequence, and the byte is now stored in the EEPROM.

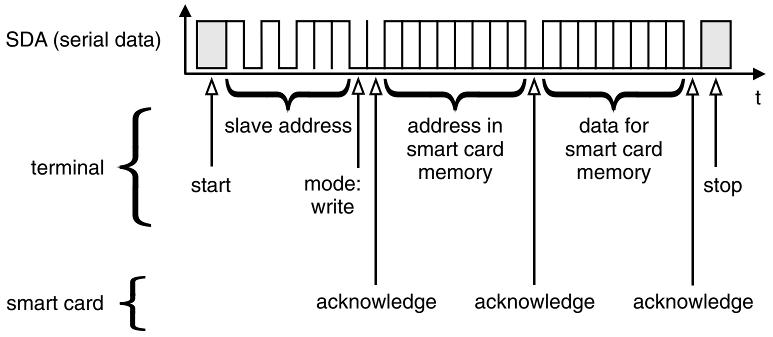


Figure 6.24 Unconstrained writing of a byte in memory using the I^{2}C bus

6.4.2 The T = 0 transmission protocol

The T = 0 transmission protocol was first used in France during the initial development of smart cards, and it was also the first internationally standardized smart card protocol. It was generated in the early years of smart card technology, and it is thus designed for minimum memory usage and maximum simplicity. This protocol is used worldwide in GSM cards, which makes it the most widely used of all current smart card protocols. The T = 0 protocol is standardized in ISO/IEC 7816-3. Additional compatible specifications are contained in the GSM 11.11, TS 102.221 and EMV specifications.

The T = 0 protocol is byte-oriented, which means that the smallest unit processed by the protocol is a single byte. The transmission data unit consists of a header containing a class byte, a command byte and three parameter bytes, optionally followed by a data section. In contrast to the application protocol data unit (APDU) specified by ISO/IEC 7816-4, length information is provided only by parameter P3. This indicates the length of the command data or response data. It is also specified by the ISO/IEC 7816-3 standard.

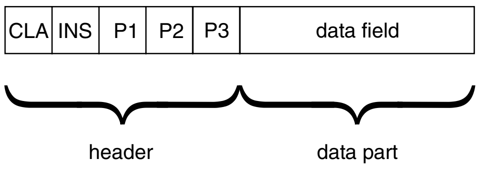


Figure 6.25 Structure of a command with the T = 0 protocol

Due to the byte orientation of the T = 0 protocol, if a transmission error is detected, retransmission of the incorrect byte must be requested immediately. With block protocols, by contrast, an entire block (a sequence of bytes) must be retransmitted if an error occurs. Error detection with T = 0 is based exclusively on a parity bit appended to each sent byte.

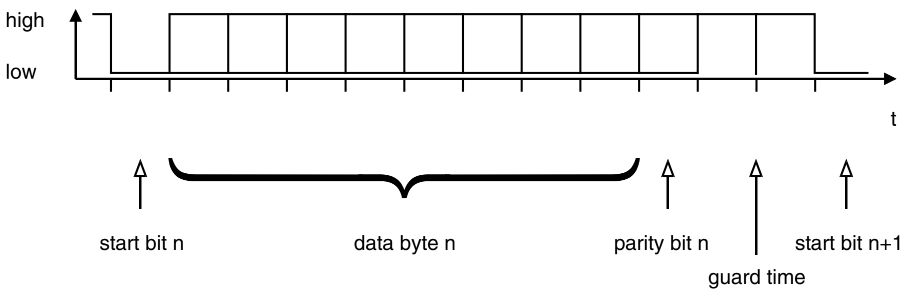


Figure 6.26 A byte transmitted via the I/O interface with no error using the T = 0 protocol

If the recipient detects a transmission error, it must set the I/O line to a low level for the duration of one etu starting halfway through the first bit interval of the guard time of the faulty byte. This indicates to the other party that the most recent byte must be retransmitted. This byte repetition mechanism is very simple, and it has the advantage that it is selective, since only incorrect bytes have to be repeated. Unfortunately, this mechanism suffers from a severe disadvantage. Most interface ICs treat the etu interval as the smallest detectable unit, so they cannot recognize a low level on the I/O line that is set halfway through a stop bit. Standard interface ICs are thus not suitable for the T = 0 protocol. However, if each bit is received separately by software, this is not a problem.

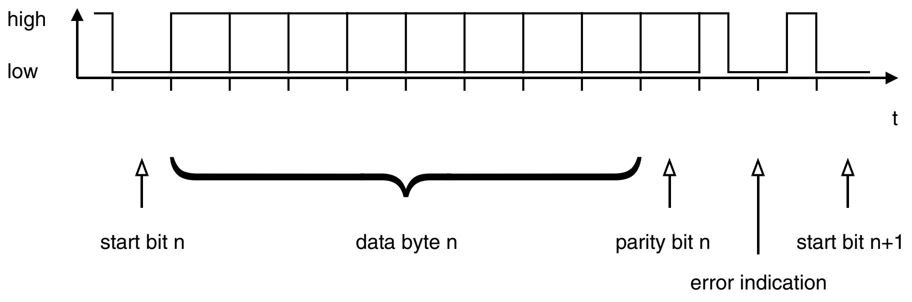


Figure 6.27 A data transmission error is indicated in the T = 0 protocol by a low level at the I/O interface during the guard time

The T = 0 protocol also allows an external programming voltage for the EEPROM or EPROM to be switched on or off. This is done by adding 1 to the received command byte and sending it back to the terminal as an acknowledge byte. This is why only even-valued command bytes are permitted, since otherwise this mechanism would not work. However, switching an external programming voltage is technically obsolete, since all smart card microcontrollers now generate the programming voltage in the chip itself. We thus need not discuss this topic any further.[10]

To illustrate the T = 0 command–response sequence, let us assume that the terminal sends the card a command with a data section, and the card responds with data and a return code (see Figure 6.28). The terminal first sends the card a 5-byte command header, consisting of a class byte, a command byte and the P1, P2 and P3 bytes. If this is received correctly, the card returns an acknowledgement (ACK) in the form of a procedure byte (PB). This acknowledgement is coded the same as the received command byte. On receipt of the procedure byte, the terminal sends exactly the number of data bytes indicated by the P3 byte. Now the card has received the complete command, and it can process it and generate a response.

If the response contains data in addition to the 2-byte return code, the card informs the terminal of this via a special return code, with the amount of data indicated by SW2. After receiving this response, the terminal sends the card a GET RESPONSE command, which consists only of a command header and an indication of the amount of data to be sent. The card now sends the terminal the requested amount of data generated in response to the first command, with the appropriate return code. This completes one command sequence.

If a command is sent to the card and the card only generates a return code with no data section, the GET RESPONSE portion of Figure 6.28 does not occur. Since an additional command from the application layer is needed to perform this action (fetching data related to a previous command), there is naturally no longer strict separation between the protocol layers. An application-layer command (GET RESPONSE) must be used here to support the

[10] There is currently a proposal for revising the ISO/IEC 7816-3 standard to eliminate the provision for controlling an external programming voltage via the instruction byte. This would double the number of possible commands

Smart card		**Terminal**
	←	5-byte command header [CLA, INS, P1, P2, P3]
Header received without error, request transmission of data section: [ACK] (acknowledge ACK)	→	
	←	[data section] with number of data bytes = P3
command processing Command executed and data available; data length in SW2: [SW1 || SW2]	→	
	←	GET RESPONSE P3 =quantity of data to be sent [CLA, INS, P1, P2, P3]
[data || SW1 || SW2]	→	Command–response sequence completed.

Figure 6.28 A typical T = 0 communications sequence with data in both the command and the response (a Case 4 command, such as MUTUAL AUTHENTICATE)

data link layer, which has certain effects on the application in question. All of this may appear complicated at first sight, so it is shown again graphically in Figure 6.30.

The maximum interval between the leading edges of two successive bytes is designated the work waiting time. This is coded in data element TC2 of the ATR.

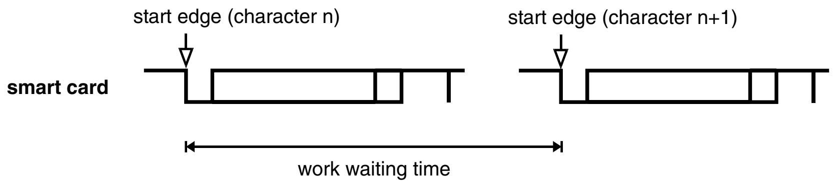


Figure 6.29 Definition of the work waiting time

The primary function of the guard time is to separate individual bytes during the transmission. This gives the sender and the recipient more time to perform the functions of the transmission protocol.

If the smart card returns a procedure byte containing the null value ('60') to the terminal, this does not have any effect on the actual sequence of the protocol, but it does inform the terminal that the smart card is still processing the last command that it received. Sending a null value can thus be used as a sort of waiting time extension (WTX), although it is not standardized in this form.

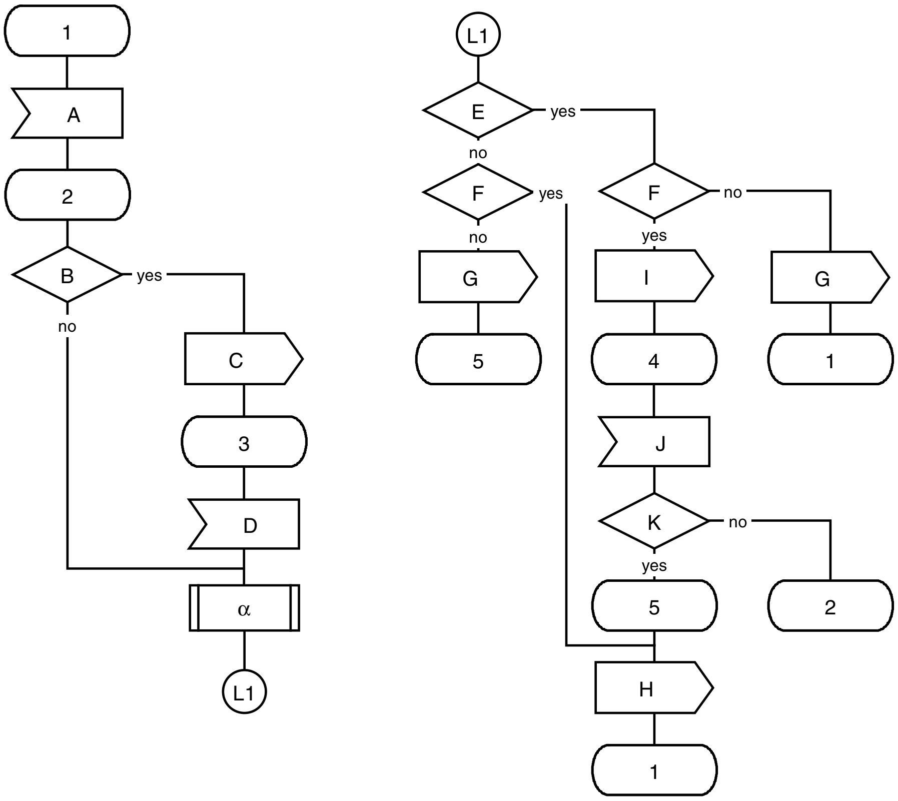


Figure 6.30 The smart card state machine for the communications process using the T = 0 communications protocol, without error handling

α	Command processing
1	Quiescent state
2	Header received with CLA, INS, P1, P2 & P3
3	Wait for the data section (P3 = number of bytes)
4	Wait for a command (header with CLA, INS, P1, P2 & P3) (P3 = amount of response data)
5	SW1, SW2 sent and GET RESPONSE received
A	Receive the header (5 bytes)
B	Data available (P3 != 0)?
C	Data section available; send procedure byte to terminal
D	Receive the data section (P3 = number of bytes)
E	Did the command contain a data section (i.e., C and D executed)?
F	Are response data available (no error occurred)?
G	Send SW1 and SW2
H	Send the available response data and SW1 + SW2
I	Send SW1 and SW2 (SW2 = amount of response data)
J	Receive a command (header = 5 bytes)
K	Is the received command GET RESPONSE?

The T = 0 protocol allows the card to receive the bytes in the data section individually after it has received the header. To do so, it only has to send the inverted command byte to the terminal as a procedure byte, whereupon the terminal will send a single data byte. The next data byte follows the next procedure byte from the card. This bytewise transmission can continue until the card has received all the bytes in the data section, or until it sends the non-inverted command byte to the terminal as a procedure byte. Upon receiving the non-inverted command byte, the terminal sends all the remaining data bytes to the card, which will have then received the complete command.

There are two incompatibilities here between GSM 11.11 and ISO/IEC 7816-3. The first is that according to the GSM standard, a GET RESPONSE command is requested using SW1 = '9F', while according to the ISO/IEC standard the usual value is '61'.[11] In each case, SW2 contains the amount of data to be fetched. The second incompatibility between the two standards relates to the manner in which data is fetched using GET RESPONSE. The manner described above corresponds to the GSM standard and is representative for the majority of smart card applications throughout the world. According to the ISO/IEC standard, a certain amount of data can be fetched using GET RESPONSE, but there is no marker to allow subsequent data packets to be requested one after the other. With the ISO/IEC standard, GET RESPONSE always starts with the first byte.

These two incompatibilities can be easily handled in the terminal by suitable software. The important thing is to be aware that they exist.

Smart card		**Terminal**
	←	5-byte command header [CLA, INS, P1, P2, P3]
'Send one data byte'	→	
	←	[Data byte 1]
'Send one data byte'	→	
	←	[Data byte 2]
further reception of individual bytes		*further transmission of individual bytes*
'Send all remaining data bytes'	→	
	←	[Data bytes *n*–P3] (P3 = number of data bytes)
command processing [SW1 || SW2]	→	Command–response sequence completed

Figure 6.31 Single-byte reception with T = 0 (a Case 3 command, such as UPDATE BINARY)

[11] There is a proposal for revising the ISO/IEC 7816-3 standard to have it allow both values

With a transmission protocol, the primary concerns of the user are ultimately only the data transmission rate and the error detection and correction mechanisms. Transmitting an 8-bit byte requires sending 12 bits, including one start bit, one parity bit and two etu for the guard time. Transmitting one byte thus takes 12 etu, which is equivalent to 1.25 ms with a 3.5712-MHz clock frequency and a divider value of 372. Table 6.33 lists data transmission times for some typical commands.

Table 6.33 Data transmission times for some typical commands using the T = 0 protocol and a clock rate of 3.5712 MHz, a divider value of 372, 2 stop bits and 8 data bytes per command (C = command, R = response)

Command	User data	Protocol data	Data transmission time
READ BINARY	C: 5 bytes R: 2 + 8 bytes	—	18.75 ms
UPDATE BINARY	C: 5 + 8 bytes R: 2 bytes	—	18.75 ms
ENCRYPT	C: 5 + 8 bytes R: 2 + 8 bytes	C: 5 bytes R: 2 bytes	37.50 ms

The data transmission rate naturally drops if transmission errors occur. However, the single-byte repetition mechanism is very advantageous here, since only incorrectly received bytes have to be retransmitted.

The error detection mechanism of the T = 0 protocol consists only of a parity check at the end of each byte. This allows reliable recognition of single-bit errors, but two-bit errors cannot be detected. Furthermore, if a byte is lost during the transmission from the terminal to the card, this results in an endless loop (deadlock) in the card, since it is expecting a specific number of bytes and has no possibility of timing out. The only practical way for the terminal to escape from this situation is to reset the card and establish communications again from the beginning.

There is a very similar situation when the terminal is expecting more data than the smart card sends. This also unavoidably leads to a deadlock. For this reason, some implementations of the T = 0 protocol in terminals have a timer that triggers termination of communications after a configurable maximum interval. The mechanism used for this is similar to that for the block waiting time (BWT) with the T = 1 protocol. However, it is not standardized and is thus implementation-dependent.

In normal communications, the insufficient separation of the link and transport layers does not cause any major problems. The smooth operation of the GSM application is the best proof of this. However, problems quickly arise if secure messaging is used. With a partly encrypted header and a fully encrypted data section, it is no longer possible to support the T = 0 protocol using the previously described procedure without incurring large overheads. This is because the unencrypted command byte must be used for the procedure byte in the T = 0 protocol. However, this fact has been recognized by the standards organizations and is taken into account in the standards related to secure messaging, so all varieties of secure messaging are possible using the T = 0 protocol.

Due to the absence of layer separation and the obvious problems in the event of a bad connection, the T = 0 protocol is often considered to be outdated. However, transmission errors almost never occur in communications between the terminal and the card. The main advantages of the T = 0 protocol are its good average transmission rate, minimal implementation overhead and widespread use.

6.4.3 The T = 1 transmission protocol

The T = 1 transmission protocol is an asynchronous half-duplex protocol for smart cards. It is based on the international ISO/IEC 7816-3 standard. The TS 102.221 and EMV specifications are also relevant for this protocol. The T = 1 protocol is a block-oriented protocol, which means that one block is the smallest data unit that can be transmitted between the card and the terminal.

This protocol features strict layer separation, and it can be assigned to the data link layer (transport layer) in the OSI reference model. In this context, layer separation means that data destined for higher layers, such as the application layer, can be processed completely transparently by the data link layer. It is not necessary for layers other than the ones directly involved to interpret or modify the contents of the transmitted data.

Secure messaging (SM), in particular, requires adherence to layer separation. Only then can encrypted user data be passed across the interface without resorting to complicated procedures or tricks. The T = 1 protocol is currently the only international smart card protocol that permits all varieties of secure data transmission without any compromises.

The transmission protocol sequence starts after the card has sent the ATR or after a successful PPS has been executed. The first block is sent by the terminal, and the next block is sent by the card. Communication then continues in this manner, with the transmit privilege alternating between the terminal and the card.

Incidentally, the T = 1 protocol is not limited to being used for communication between smart cards and terminals. It is also used by many terminals to exchange application and control data with the computers to which they are connected.

The data transmission rate is naturally of particular interest for any data transmission protocol. Table 6.34 lists the transmission times for some typical commands using the T = 1 protocol.

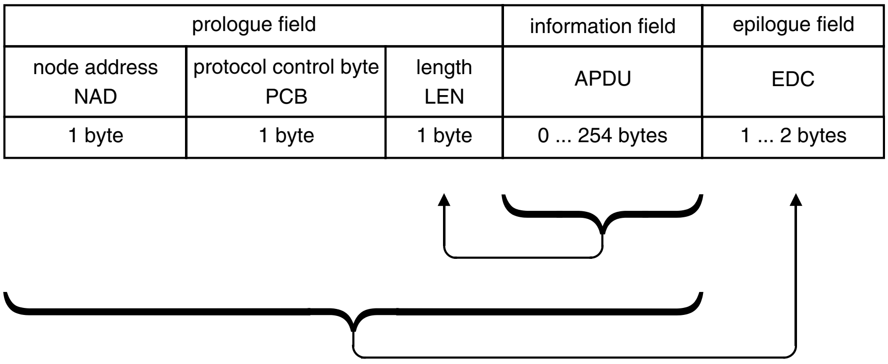
Table 6.34 Data transmission times for some typical commands with the T = 1 protocol and a clock rate of 3.5712 MHz, a divider value of 372, an XOR error detection code, 2 stop bits and 8 data bytes per command (C = command, R = response)

Command	User data	Protocol data	Data transmission time
READ BINARY	C: 5 bytes	C: 4 bytes	28.75 ms
	R: 2 + 8 bytes	R: 4 bytes	
UPDATE BINARY	C: 5 + 8 bytes	C: 4 bytes	23.00 ms
	R: 2 bytes	R: 4 bytes	
ENCRYPT	C: 5 + 8 bytes	C: 4 bytes	38.75 ms
	R: 2 + 8 bytes	R: 4 bytes	

6.4.3.1 Block structure

The transmitted blocks are basically used for two different purposes. One of these is the transparent transmission of application-specific data, while the other is sending protocol control data or handling transmission errors.

A transmission block consists of an initial prologue field, an information field and a final epilogue field. The prologue and epilogue fields are mandatory and must always be sent. The information field is optional and contains data for the application layer, which is either a command APDU sent to the card or a response APDU from the card.



prologue field			information field	epilogue field
node address NAD	protocol control byte PCB	length LEN	APDU	EDC
1 byte	1 byte	1 byte	0 ... 254 bytes	1 ... 2 bytes

Figure 6.32 The structure of a T = 1 transmission block

There are three fundamentally different types of blocks in T = 1: information blocks, receipt acknowledgement blocks and system blocks. Information blocks (I blocks) are used to transparently exchange application-layer data. Receipt acknowledgement blocks (R blocks), which do not contain any data fields, are used for positive or negative reception confirmation. System blocks (S blocks) are used for control information related to the protocol itself. Depending on the specific control data, they may have an information field.

The prologue field

The prologue field consists of three subfields: node address (NAD), protocol control byte (PCB) and length (LEN). It is three bytes long and contains basic control and pointer data for the actual transmission block.

Node address (NAD)

The first byte in the prologue field is called the node address (NAD) byte. It contains the destination and source addresses for the block. Each of these is coded using three bits. If an address is not used, its bits are set to 0. Furthermore, for compatibility with older microcontrollers, control is provided for the EEPROM or EPROM programming voltage. However, there is no practical use for this, since all smart card microcontrollers now have on-board charge pumps.

Protocol control byte (PCB)

The subfield following the node address is the protocol control byte (PCB). As the name suggests, it serves to control and supervise the transmission protocol. This increases the amount

Table 6.35 Node address (NAD field)

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
X	...	...	...	X	...	...	...	Vpp control
...	X	X	X	...	...	...	...	DAD (destination address)
...	...	...	...	...	X	X	X	SAD (source address)

of coding required. The PCB field primarily encodes the block type, as well as associated supplementary information.

Table 6.36 PCB field for an I block

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	...	...	...	...	...	...	...	I block identifier
...	N(S)	...	...	...	...	...	...	Send sequence number
...	...	X	...	...	...	...	...	Sequence data bit M
...	...	...	X	X	X	X	X	Reserved

Table 6.37 PCB field for an R block

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	0	...	...	...	...	...	...	R block identifier
...	...	0	N(R)	0	0	0	0	No error
...	...	0	N(R)	0	0	0	1	EDC or parity error
...	...	0	N(R)	0	0	1	0	Other error

Length field (LEN)

The one-byte length (LEN) field indicates the length of the information field in hexadecimal form. Its value can be '00' to 'FE'. The code 'FF' is reserved for future extensions and currently should not be used.

The information field

In an I block, the information field serves as a container for application layer data (OSI layer 7). The content of this field is transmitted completely transparently. This means that the content is directly passed on by the transmission protocol without any analysis or evaluation.

In an S block, the information field transfers data for the transmission protocol. This is the only case in which this content of this field is used by the transport layer.

According to the ISO standard, the size of the information field can range from '00' to 'FE' (254) bytes. The value 'FF' (255) is reserved by ISO for future use. The terminal and card may have I fields with different sizes. The default size of the terminal I field is 32 bytes (IFSD = information field size for the interface device); this can be modified via a special S field. This

Table 6.38 PCB field for an S block

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	1	...	...	...	...	...	...	S block identifier
...	...	0	0	0	0	0	0	Resync request (only from terminal)
...	...	1	0	0	0	0	0	Resync response (only from smart card)
...	...	0	0	0	0	0	1	Request change to information field size
...	...	1	0	0	0	0	1	Response to request change to information field size
...	...	0	0	0	0	1	0	Request abort
...	...	1	0	0	0	1	0	Response to abort request
...	...	0	0	0	0	1	1	Request waiting time extension (only from smart card)
...	...	1	0	0	0	1	1	Response to waiting time extension (only from terminal)
...	...	1	0	0	1	0	0	Vpp error response (only from smart card)

default value of 32 bytes also applies to the card (IFSC = information field size for the card), but this can be modified by a parameter in the ATR.[12] In practice, the size of the I field for both the terminal and the card is between 50 and 254 bytes.

Epilogue field

The epilogue field, which is transmitted at the end of the block, contains an error detection code computed from all previous bytes in the block. The computation employs either an LRC (longitudinal redundancy check) or a CRC (cyclic redundancy check). The method used must be specified in the interface characters of the ATR. If it is not specified, by convention the LRC method is implicitly used. Otherwise, the CRC computation is carried out according to ISO 3309. The divider polynomial used, $G(x) = x^{16}+x^{12} + x^5 + 1$, is the same as for CCITT Recommendation V.41.[13] Both of these error detection codes can only be used for error detection; they cannot correct a block error.

The single-byte longitudinal redundancy checksum is computed using XOR concatenation of all previous bytes in the block. This computation can be executed very quickly, and its implementation is not code-intensive. It is usually performed on the fly during data transmission or reception. It is a standard part of practically all T = 1 implementations.

Using the CRC procedure to generate an error detection code yields a much greater probability of error detection than the relatively primitive XOR checksum. However, this procedure

[12] The TAi (i > 2) data element in the ATR
[13] See also Section 4.5.2, 'CRC checksums'

is presently not used in practice, since the XOR checksum has become the established standard throughout the world. With the CRC procedure, the epilogue field must be extended to two bytes, which further reduces the data transmission rate.

6.4.3.2 Send/receive sequence counter

Each information block in the T = 1 protocol has a send sequence number consisting of only one bit located in the PCB byte. This number is incremented modulo 2, which means that it alternates between 0 and 1. The send sequence counter is also designated N(S). Its starting value at protocol initiation is 0. The counters in the terminal and the smart card are incremented independently of each other.

The primary purpose of the send sequence counter is to support requests for resending blocks received with errors, since individual data blocks can be unambiguously addressed via N(S).

6.4.3.3 Waiting times

Several waiting times are defined to provide transmitters and receivers with precisely specified minimum and maximum intervals for various actions during data transmission. They also provide defined ways to terminate communications in order to prevent deadlocks in case of errors. Default values are defined for all of these waiting times in the standard, but these may be modified to maximize the transmission rate. The modified values are indicated in the specific interface characters of the ATR.

Character waiting time (CWT)

The character waiting time is defined as the maximum interval between the leading edges of two consecutive characters within a block. The recipient starts a countdown timer on each leading edge, using the character waiting time as the initial value. If the timer expires and no leading edge for a new bit has been detected, the recipient assumes that the transmission block has been received in full. The 'CWT reception criterion' can thus be generally used for end-of-block detection. However, this considerably reduces the data transmission rate, since the time for each block is increased by the duration of the CWT. It is thus better to detect the end of the block by counting received bytes.

The CWT is calculated using the data element CWI contained in the ATR, according to the following formula:

$$CWT = (2^{CWI} + 11) \text{ work etu}$$

The default value for the CWI is 13, which yields the following value for CWT:

$$CWT = (2^{13} + 11) \text{ work etu} = 8203 \text{ work etu}$$

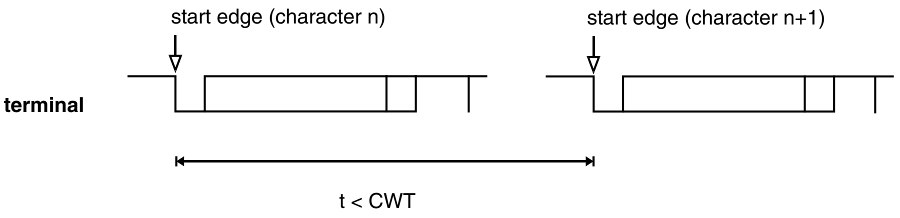


Figure 6.33 Definition of the character waiting time (CWT)

With a clock frequency of 3.5712 MHz and a divider value of 372, this yields an interval of 0.85 seconds.[14]

This interval, which is specified in the standard as the default setting, is too long for fast data communications. In practice, the usual value of CWI is between 3 and 5. This means that for a normal transmission sequence, in which the characters follow each other without any time delay, the receiver waits for an interval of one to two bytes before detecting the end of the block or interruption of communications.

Normally, the reception routine detects the end of a block from the block length information in the LEN field. However, if the content of this field is erroneous, the character waiting time can be used as an additional means to terminating reception. This problem only manifests itself when the length information is too long, since in this case the receiver would wait for additional characters that never arrive. This would block the transmission protocol, and this situation could only be cleared by a card reset. The character waiting time mechanism gets around this problem.

Block waiting time (BWT)

The purpose of the block waiting time is to allow communications to be terminated in a defined manner if the smart card does not respond. The block waiting time is the maximum allowed interval between the leading edge of the last byte of a block sent to the card and the leading edge of the first byte returned by the card.

In terms of a conventional T = 1 block, this is the allowed maximum interval between the leading edge of the XOR byte in the epilogue field of the command block and the leading edge of the NAD byte in the response from the card. If this waiting period expires without a response being received from the card, the terminal may assume that the card is faulty and initiate an appropriate response. This could for example be a card reset, followed by a new attempt to establish communication. The BWT is specified in abbreviated form in the interface characters of the ATR by the BWI parameter. The value of the BWT is given by the formula

$$BWT = 2^{BWI} \times 960 \times \frac{372}{f}\,\text{s} + \text{work etu}$$

[14] See also Section 16.11.2, 'ATR data element conversion tables'

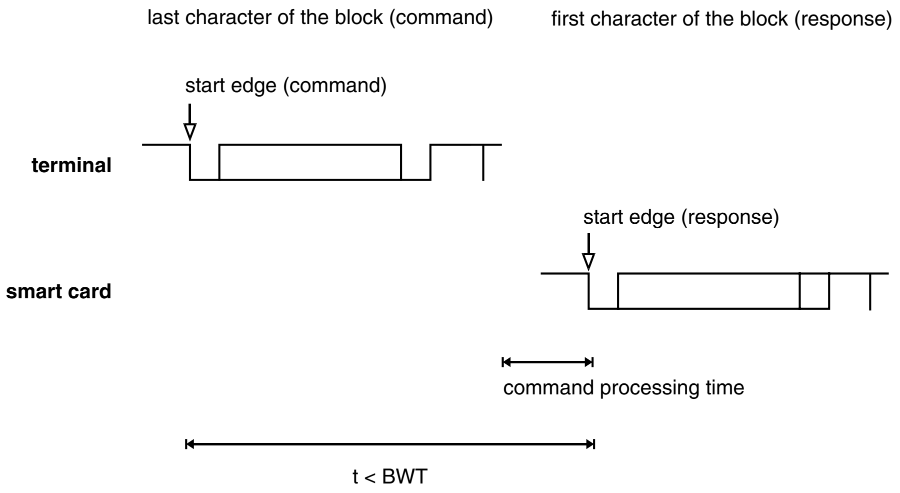


Figure 6.34 Definition of the block waiting time (BWT)

If no BWI value is given in the ATR, the default value of 4 is used. With 3.5712 MHz and a divider value of 372, this gives 1.6 s as the value for the block waiting time:

$$BWT = 2^4 \times 960 \times \frac{372}{3{,}571{,}200 \text{ Hz}} \text{ s} + 11 \text{ work etu} = 2^4 \times 0.1 \text{ s} + 11 \text{ work etu} \approx 1.6 \text{ s}$$

As can be seen, this value is quite generous. In practice, a value of 3 is often used for BWI, which yields a block waiting time of 0.8 s.[15] Typical command processing times in the card are usually around 0.2 s.[16] A BWT of the above duration thus represents a compromise between normal command processing times and quick detection of a smart card that is no longer responding to commands.

Block guard time (BGT)

The block guard time is defined as the minimum interval between the leading edge of the final byte and the leading edge of the first byte in the opposite direction. It is the opposite of the BWT, which is defined as the maximum time between the two specified leading edges. Another difference is that the block guard time is obligatory for both parties and must be observed, while the block waiting time is only significant for the smart card. The purpose of the block guard time is to provide the sender with a minimum time interval in which to switch over from transmitting to receiving.

The block guard time has a standard fixed value of 22 etu. In a smart card running at 3.5712 MHz with a divider value of 372, this yields an interval of approximately 2.3 ms.

[15] See also Section 16.11.2, 'ATR data element conversion tables'
[16] See also Section 15.2, 'Formulas for Estimating Processing Times'

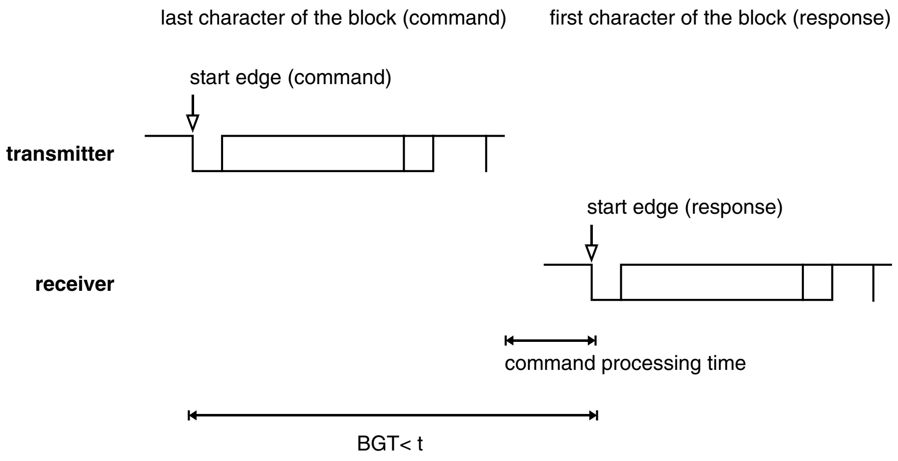


Figure 6.35 Definition of the block guard time (BGT)

6.4.3.4 Transmission protocol mechanisms

Waiting time extension

If the smart card needs more time to generate a response than the maximum time allowed by the block waiting time (BWT), it can request a waiting time extension from the terminal. It does so by sending a special S block requesting an extension, and it receives a corresponding S block from the terminal in acknowledgement. The terminal is not allowed to refuse this request.

A byte in the information field informs the terminal of the length of the extension. This byte, multiplied by the block waiting time, gives a new block waiting time. This extension is only valid for the most recently sent I block.

Smart card		Terminal
	←	I block
S block (request waiting time extension)	→	
	←	S block (acknowledge waiting time extension)
I block	→	

Figure 6.36 Procedure for extending the waiting time

Block chaining

One of the essential performance features of the T = 1 protocol is the block chaining function. This allows either party to send data blocks that are larger than the size of its transmit or receive buffer. This is particularly useful in light of the limited memory capacities of smart cards. Chaining is permitted only for information blocks, since only such blocks can contain large amounts of data. In the chaining process, the application data are partitioned into individual blocks that are sent to the recipient one after the other.

The application layer data must be partitioned such that none of the resulting segments is larger than the maximum block size of the recipient. The first segment is then placed in an information field in accordance with the T = 1 protocol, supplied with prologue and epilogue fields and sent to the recipient. The M bit ('more data' bit) is set in the block's PCB field to indicate to the recipient that the block chaining function is being used and that chained data are located in the following blocks.

As soon as the recipient has successfully received this information block with the first segment of the user data, it indicates that it is ready to receive the next chained I block by returning an R block whose sequence number N(R) is the same as the send sequence count N(S) of the next I block. The next block is then sent to the recipient.

This reciprocal exchange of I and R blocks continues until the sender issues an I block with an M bit in the PCB field indicating that it is the last block in the chain (M bit = 0). After this block has been received, the recipient has all the application layer data and can process the full data block.

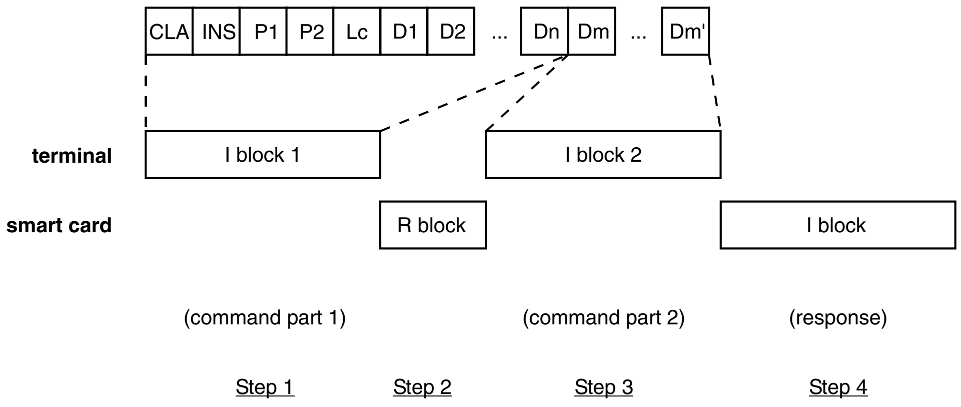


Figure 6.37 Example of block chaining for transmitting data from the terminal to the smart card

There is a restriction with regard to the block chaining mechanism. Within a single command–response cycle, chaining may proceed in one direction only. For example, if the terminal is sending chained blocks, the card may not send chained blocks in response.

There is another restriction that has nothing to do with the mechanism itself, but rather with the very limited amount of memory in smart cards. Implementing the block chaining mechanism involves extra overhead, and its usefulness is very limited, since commands and responses are not especially long and thus do not normally require chaining.

If the card's receive buffer in RAM is not large enough to store all of the data passed using block chaining, it must be moved to the EEPROM. This leads to a sharp reduction in the transmission rate, since the EEPROM (in contrast to the RAM) cannot be written at the full speed of the processor. Consequently, many T = 1 implementations have no block chaining function, as the cost/benefit ratio often does not justify it. This is a typical example of the fact that standards are often interpreted very liberally in actual practice. In this case, the interpretation amounts to considering block chaining to be a supplementary option in T = 1 that is not absolutely necessary. This can lead to significant compatibility problems in practice.

Error handling

The T = 1 protocol has elaborate error detection and handling mechanisms. If an invalid block is received, the protocol attempts to restore error-free communications by means of precisely defined procedures.

Seen from the terminal's perspective, there are three synchronization stages. In the first stage, the sender of a faulty block receives an R block indicating an EDC/parity bit error or a general error. The recipient of this R block (the original sender) must then retransmit the last block that it sent.

If it proves impossible to restore an error-free connection using this mechanism, the next stage is invoked. This means that the smart card receives a resynchronization request from the terminal in an S block. The terminal expects a resync response in reply. The terminal and card then both reset their send and receive counters to zero, which corresponds to the protocol state immediately following the ATR. Starting from this situation, the terminal attempts to restore communications.

The first two stages affect only the protocol layer. They have no effect on the actual application. However, the third synchronization stage does affect all layers in the smart card. If the terminal cannot establish an error-free connection using the first two synchronization stages, it triggers a smart card reset via the reset lead. This unfortunately means that all the data and states of the current session are lost. Following the reset, communications must be completely re-established from the bottom up. If even this procedure fails to produce a working connection after three attempts, the terminal deactivates the card. The user then usually receives an error message to the effect that the card is defective.

Table 6.39 T = 1 error-handling stages

Synchronization stage	Mechanism
Stage 1	Repeat the erroneous block
Stage 2	Resynchronize and then repeat the erroneous block
Stage 3	Reset the smart card and establish the connection anew

6.4.3.5 Example of data transmission with the T = 1 protocol

Figure 6.38 shows an example of a data transmission for a SELECT FILE command using the T = 1 protocol.

6.4.3.6 Differences between ISO/IEC T = 1 and EMV T = 1

The original definition of the T = 1 protocol according to the ISO/IEC 7816-3 standard has provisions for many options and mechanisms, some of which are code-intensive and rarely

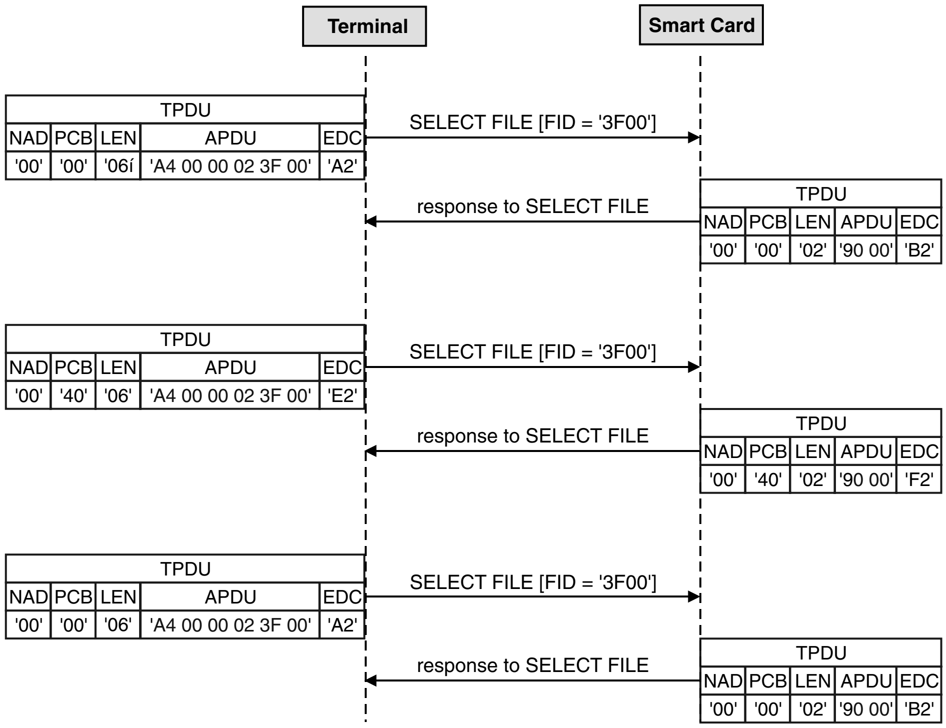

Terminal
Smart Card
TPDU
NAD PCB LEN APDU EDC
'00' '00' '06í' 'A4 00 00 02 3F 00' 'A2'
SELECT FILE [FID = '3F00']
response to SELECT FILE
TPDU
NAD PCB LEN APDU EDC
'00' '00' '02' '90 00' 'B2'
TPDU
NAD PCB LEN APDU EDC
'00' '40' '06' 'A4 00 00 02 3F 00' 'E2'
SELECT FILE [FID = '3F00']
response to SELECT FILE
TPDU
NAD PCB LEN APDU EDC
'00' '40' '02' '90 00' 'F2'
TPDU
NAD PCB LEN APDU EDC
'00' '00' '06' 'A4 00 00 02 3F 00' 'A2'
SELECT FILE [FID = '3F00']
response to SELECT FILE
TPDU
NAD PCB LEN APDU EDC
'00' '00' '02' '90 00' 'B2'

Figure 6.38 Successful transmission of a TPDU using the T = 1 transmission protocol. The XOR option is used for the error detection code (EDC). A SELECT FILE command with a FID of '3F00', which selects the MF, is transmitted in the APDU. The incrementing of the send sequence count in the PCB byte for each transaction, and the corresponding changes in the EDC, can be readily seen

used. The elaborate error correction mechanisms are a typical example. Although they may be theoretically interesting, in many cases they are ineffective in dealing with actual transmission errors. In practice, it is usually better to simply reinsert the card in the terminal and start a new session, rather than using innumerable resynchronization requests to attempt to restabilize communications. Consequently, the EMV specification imposes several restrictions relative to the original ISO/IEC standard, as summarized in Table 6.40.

6.4.4 The T = 14 transmission protocol (Germany)

The ISO/IEC 7816-3 standard includes a tag in the ATR for designating a national transmission protocol. Such a transmission protocol is designated T = 14. With the introduction of the C-Netz for mobile telephony and card phones in Germany, a protocol was needed for communicating with the smart cards used in these systems. The character-oriented T = 0 protocol was considered undesirable, and at that time there was not yet a standardized block-oriented protocol. Consequently, in 1987 Telekom decided to use a protocol developed by a DIN working group. This protocol received the designation T = 14, which simply means that it is a country-specific implementation. It had no significance outside of Germany, but it had

Table 6.40 Summary of the differences in the implementation of the T = 1 transmission protocol between the ISO/IEC 7816-3 standard and the EMV specification

Mechanism or option	ISO/IEC 7816-3	EMV
BWT expired	Reset the card (for example)	Deactivate the card
Smart card sends a request to change the IFS	Allowed	A maximum of three successive requests is allowed
Smart card sends an S block with an abort request	Allowed	Deactivate the card
Zero-length I block	Allowed	Prohibited
Terminal sends three successive blocks without receiving a valid response	Behavior according to specified error handling; usually a resync request	Deactivate the card

enormous influence on the development of the internationally standardized T = 1 protocol, since the T = 14 protocol formed the principal foundation for this protocol.

The T = 14 protocol was used extremely widely in Germany, since it was employed in the C-Netz mobile telecommunications network and public card phones. With the closing down of the C-Netz at the end of 2000 and the change to the T = 1 protocol for public card phones, the T = 14 protocol is no longer an important protocol in Germany, which is why it is described here only briefly.

The T = 14 protocol has a block-oriented structure and works asynchronously using the applied clock signal. The divider (clock rate conversion factor) has a value of 512, which yields a data transmission rate of 9600 bit/s at a clock frequency of 4.9512 MHz. Data transmission at layer 2 (the data link layer) always takes place using the direct convention. The size of the buffers for transmitted and received blocks must be at least 50 bytes, with a maximum value of 255 bytes. There is no block chaining mechanism.

6.4.5 The USB transmission protocol

A future amendment to the ISO/IEC 7816-3 standard will contain a specification for a new transmission protocol for smart cards. This is the Universal Serial Bus (USB) protocol, which has come to prevail over competing protocols, such as Firewire and the like, for use with smart card applications.

The USB protocol requires a special hardware component in the smart card microcontroller, but this component is already present in many chips, at least as an option. The major advantage of USB with respect to currently used transmission protocols is that it is an established industrial standard coming from the PC world. USB also offers a higher transmission rate than T = 0 or T = 1.

It presently appears that Version 1.1 of the USB specification will be used for smart cards, with both the low-speed option (1.5 Mbit/s) and the full-speed option (12 Mbit/s) being supported, depending on the type of microcontroller. It should be noted that the effective transmission rate is significantly lower than the stated values once the required protocol data have been subtracted. USB Version 2.0, which has a data transmission rate of up to 480 Mbit/s (in the high-speed mode), will not be used in the foreseeable future.

Contacts C4 (AUX1) and C8 (AUX2), which up to now have been specified but not used, will be used to implement the USB interface in smart cards. Further details will be specified in the above-mentioned standard, but it can be expected that it will take several years until all of the related standardization work is completed.

6.4.6 Comparison of asynchronous transmission protocols

With regard to comparing the various transmission protocols, a brief remark with regard to achievable data transmission rates is in order. Attempts are often made to compare the T = 0 and T = 1 protocols based on calculated effective transmission rates. However, such calculations are only valid for specific commands in specific contexts. If they are generalized, they become meaningless and invalid.

Both protocols have their strengths and weaknesses with regard to achievable transmission rates. These are strongly dependent on very many individual factors, such as the transmission error rate, the size of the I/O buffer in the card and the specific implementation of the protocol. In short, it can be assumed that on average and with most applications, the effective transmission rates of both protocols are nearly the same. If you want to increase the transmission rate, changing the protocol will have little effect. It is more effective to reduce the divider value, since this yields significantly better results.

Two international transmission protocols have been described in the previous sections. In order to provide an overview, Table 6.41 summarizes the essential features of these protocols, as well as their advantages and disadvantages.

Table 6.41 Comparison of internationally standardized asynchronous data transmission protocols

Criterion	T = 0	T = 1	T = 2 (proposed)
Data transmission	asynchronous, half-duplex, byte-oriented	asynchronous, half-duplex, block-oriented	asynchronous, full-duplex, block-oriented
Standard	ISO/IEC 7816-3, GSM 11.11, EMV	ISO/IEC 7816-3, EMV	ISO/IEC 10536-4
Divider	freely definable, usually 372	freely definable, usually 372	freely definable, usually 372
Block chaining	not possible	possible	possible
Error detection	parity bits	parity bits, EDC at end of block	parity bits, EDC at end of block
Memory required for implementation	≈300 bytes	≈1100 bytes	≈1600 bytes

6.5 MESSAGE STRUCTURE: APDUs

Applications protocol data units (APDUs) are used to exchange all data that passes between the smart card and the terminal. The APDU is an internationally standardized data unit for the application layer, which is layer 7 in the OSI model. In smart cards, this layer is located directly above the transmission protocol layer. The protocol-dependent data units of the transmission protocol layer are called ‘transmission protocol data units’ (TPDUs).

A distinction is made between command APDUs (C-APDUs), which represent commands to the card, and response APDUs (R-APDUs), which represent replies to these commands from the card. In simple terms, an APDU is a sort of container that holds a complete command to the card or a complete response from the card. APDUs are passed by the transmission protocol transparently, which means without modification or interpretation.

APDUs that comply with the ISO/IEC 7816-4 standard are intended to be independent of the transmission protocol. Consequently, the content and format of an APDU must not change when a different transmission protocol is used. This applies above all to the two standard protocols, T = 0 and T = 1. This demand for protocol independence affects the structure of the APDUs, since it must be possible to transmit them transparently using both the byte-oriented T = 0 protocol and the block-oriented T = 1 protocol.

6.5.1 Structure of the command APDU

A command APDU is composed of a header and a body. The body may have a variable length, or it may be entirely absent if the associated data field is empty.

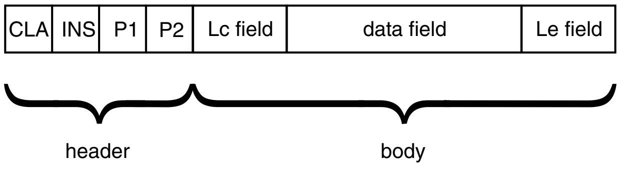


Figure 6.39 Structure of a command APDU

The header consists of four elements, which are the class byte (CLA), the instruction byte (INS) and two parameter bytes (P1 and P2). The class byte is also used to identify applications and their specific command sets. For instance, the class byte 'A0' is used for GSM, while the code '8X' is most commonly used for company-specific (private-use) commands. ISO-based commands are coded by class byte '0X'. The standard additionally specifies the class bytes to be used to indicate the use of secure messaging and logical channels. All of this is still compatible with using the class byte as an application identifier.

The next byte in the command APDU is the instruction byte, which encodes the actual command. Almost the entire address space of this byte can be exploited, with the sole restriction that only even codes can be used. This is because the T = 0 protocol allows the programming voltage to be activated by returning the instruction byte incremented by one in the procedure byte. The instruction byte thus always has had an even value.[17]

The two parameter bytes are primarily used to provide more information about the command selected by the instruction byte. They thus serve mainly as switches that select various command options. For example, they are used to choose various options for SELECT FILE or specify the offset for READ BINARY.

[17] See also Section 16.11.5, ‘The most important smart card commands’

Table 6.42 The most important class byte (CLA) codes according to ISO/IEC 7816-4

b8–b5	b4	b3	b2	b1	Meaning
...	...	...	X	X	Logical channel number
...	0	0	...	...	Secure messaging not used
...	0	1	...	...	Non-ISO secure messaging using a private method
...	1	0	...	...	ISO secure messaging, header not authentic
...	1	1	...	...	ISO secure messaging, header authentic
'0'	...	...	...	...	Structure and coding compliant with ISO/IEC 7816-4/7/8
'8', '9'	...	...	...	...	Structure compliant with ISO/IEC 7816-4, user-specific coding and meaning of commands and responses (private use)
'A'	...	...	...	...	Structure and codes compliant with ISO/IEC 7816-4, specified in supplementary documents (e.g. GSM 11.11)
'F'	1	1	1	1	Reserved for PPS

Table 6.43 Summary of the assignment of class bytes to applications

Class	Application
'0X'	Standard commands compliant with ISO/IEC 7816-4/7/8
'80'	Electronic purses compliant with EN 1546-3
'8X'	Application-specific and company-specific commands (private use)
'8X'	Credit cards with chips, compliant with EMV
'A0'	GSM mobile telecommunication systems compliant with GSM 11.11

The section following the header is the body, which can be omitted except for a length specification. The body serves a dual function. First, it specifies the length of the data section sent to the card (in the Lc field)[18] and the length of the data section to be sent back from the card (in the Le field).[19] Second, it contains the data for the commands that are sent to the card. If the value of the Le field is '00', the terminal expects the card to return the maximum amount of data available for the command. This is the only exception to the numerical description of the length.

The Le and Lc fields are usually one byte long, but they can be converted into fields that are each three bytes long. Such fields can be used to represent lengths up to 65,536, since the first byte contains the escape sequence '00'. The standard defines this three-byte length specification for future applications. Due to the limitations of currently available memory sizes, it has not yet been implemented.

The previously described parts of the command APDU can be combined to produce the four general cases illustrated in Figure 6.41.

[18] 'Lc' stands for 'length command'
[19] 'Le' stands for 'length expected'

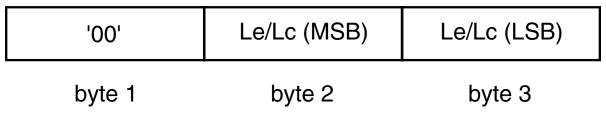


Figure 6.40 Structure of an extended Lc or Le field

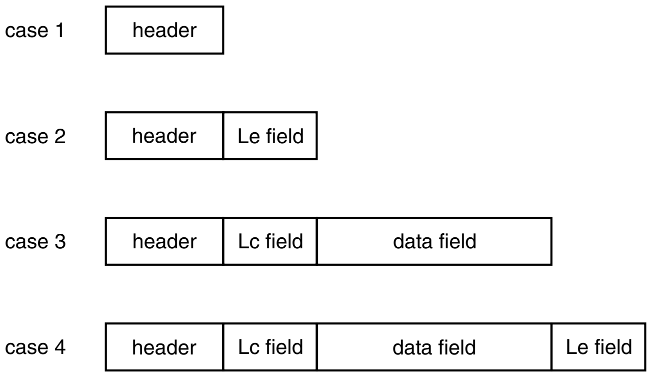


Figure 6.41 The four possible command APDU cases

6.5.2 Structure of the response APDU

The response APDU, which is sent by the card in reply to a command APDU, consists of an optional body and a mandatory trailer, as shown in Figure 6.42. The body consists of the data field, whose length is specified by the Le byte of the preceding command APDU. Regardless of the value specified in the Le byte, the length of the data field can be zero if the smart card terminates command processing due to an error or incorrect parameters. This is indicated in the two single-byte status words SW1 and SW2 in the trailer.

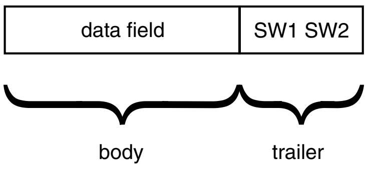


Figure 6.42 Structure of the response APDU

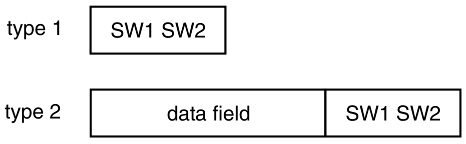


Figure 6.43 The two types of response APDUs

The card must always send a trailer in response to a command. The two bytes SW1 and SW2, which are also called the 'return code', encode the response to the command. For example, the return code '9000' means that the command was executed completely and successfully. There are more than 50 different codes. Their basic classification scheme is shown in Figure 6.44.[20]

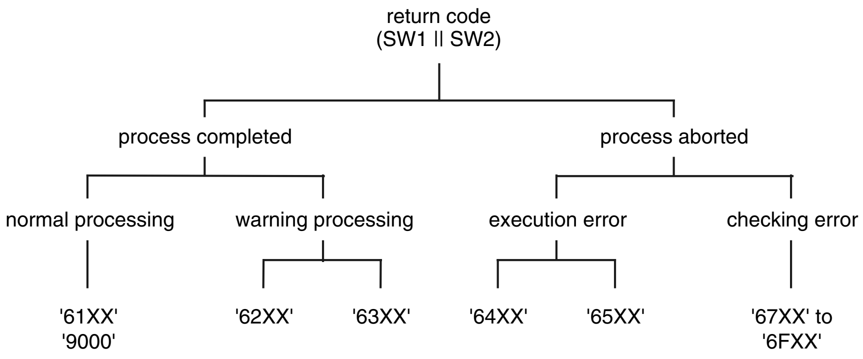


Figure 6.44 Return code classification scheme as defined by the ISO/IEC 7816-4 standard. The return codes '63XX' and '65XX' indicate that data in non-volatile memory (EEPROM) have been altered, while the remaining '6X' codes indicate that this has not occurred

If a '63XX' or '65XX' return code is received after a command has been executed, this means that the card's non-volatile memory (usually the EEPROM) has been modified. If another code starting with '6X' is returned, command execution was terminated prematurely without modifying the non-volatile memory.

It should be noted that although there is a standard for return codes, many applications use non-standard codes. The only exception is the code '9000', which almost universally indicates successful processing. With all other codes, it is always necessary to consult the relevant specification in order to be sure of their meanings.

6.6 SECURING DATA TRANSMISSIONS

The entire data exchange between a terminal and a smart card uses digital electrical pulses on the I/O line of the smart card. It is conceivable and not technically difficult to solder a wire to the I/O contact, record all the communications for a session and later analyze them. In this way, it is possible to gain knowledge of all the data transmitted in both directions.

A somewhat more difficult task is to electrically isolate the I/O contact, mount a dummy contact on top of it, and then use thin wires to connect both of these contacts to a computer. With this arrangement, it is easy to allow only certain commands to reach the card or insert 'foreign' commands into the communications sequence.

Both of these typical types of attack can succeed only if secret data passes unprotected over the I/O line. Data transmission should thus basically be designed such that even if an attacker

[20] See also Section 16.10.8, 'Smart card return codes'

is able to eavesdrop on data transmissions and insert his own message blocks, he will not be able to gain any advantage from doing so.

There are various mechanisms and methods that can be used to protect against these attacks and against even more sophisticated types of attack. They are collectively referred to as ‘secure messaging’. These mechanisms are not specific to smart cards, and they have been used for a long time in data communications systems. What is special in the smart card domain is that neither the processing capacity of the communicating parties nor the transmission rate is particularly great. Consequently, commonly used standard methods have been scaled down to match the capabilities of smart cards, without in any way reducing the security of these methods.

The objective of secure messaging is to ensure the authenticity, and if necessary the confidentiality, of part or all of the transmitted data. A variety of security mechanisms are used to meet this objective. A security mechanism is defined as a function requiring the following items: a cryptographic algorithm, a key, an argument and initial data as necessary. A general condition must also be satisfied, which is that all security mechanisms must behave completely transparently with regard to existing protocol layers, in order to ensure that existing, standardized procedures are not adversely affected by secure messaging. This applies particularly to the two transmission protocols T = 0 and T = 1, as well as to commonly used standard smart card commands.

Before using a secure messaging method, both parties must agree on the cryptographic algorithm to be used and a common secret key. According to Kerckhoff’s principle, the security of the method relies entirely on this key. If it is revealed, secure messaging is reduced to a generally known additional checksum that decreases the effective data transmission rate and at best can be used to correct transmission errors.

Several different types of secure messaging methods have been known for many years. They are all relatively rigid and tailored to specific applications. Most of them cannot be faulted as far as security is concerned. However, none of them has become internationally predominant or has proved to be sufficiently flexible to be included in current standards.

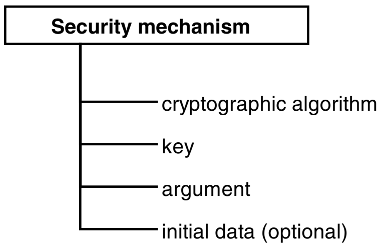


Figure 6.45 The data and functions required for a security mechanism

The requirements of transparency with respect to existing commands, use with two fundamentally different transmission protocols and maximum adaptability have led to the standardization of a very flexible (and correspondingly complex and elaborate) secure messaging method in the ISO/IEC 7816-4 standard, with additional related functions defined in ISO/IEC

7816–8.[21] This method is based on embedding all user data in TLV-coded data objects. Three different types of data objects are defined:

• data objects for plaintext:	contains data in plaintext (e.g., the data section of an APDU)
• data objects for security mechanisms:	contains the results of a security mechanism (e.g., a MAC)
• data objects for auxiliary functions:	contains control data for secure messaging (e.g., the padding method used)

The class byte indicates whether secure messaging is used for the command. The two available bytes can encode whether the method specified in ISO/IEC 7816-4 is used and whether the header is also included in the cryptographic checksum (CCS).[22] If the header is included in the computation, it is authentic, as it cannot be changed during the transmission without this being evident.

Data objects for plaintext

According to the standard, all data that are not BER-TLV coded must be encapsulated, which means they must be embedded in data objects. Several different tags are used, as shown in Table 6.44. Bit 1 of each tag indicates whether the data object is included in the computation of the cryptographic checksum. If this bit is not set (e.g., 'B0'), the data object is not included in the computation, while if it is set (e.g., 'B1'), the data object is included.

Table 6.44 Tags for plaintext data objects

Tag	Meaning
'B0', 'B1'	BER-TLV coded; contains data objects related to secure messaging
'B2', 'B3'	BER-TLV coded; contains data objects not related to secure messaging
'80', '81'	No BER-TLV coded data
'99'	State information for secure messaging

Data objects for security mechanisms

The data objects used for security mechanisms are divided into those used for authentication and those used for confidentiality. The tags defined for this purpose are listed in Tables 6.45 and 6.46.

Here ‘authenticity’ refers to all data objects related to cryptographic checksums and digital signatures. Data encryption, and marking such data as encrypted in the context of secure

[21] Secure messaging is usually abbreviated to ‘SM’, which many programmers interpret as ‘sado-masochism’ on account of the many degrees of freedom and room for interpretation provided by these two standards
[22] For the coding of the class byte, see Section 6.5.1, ‘Structure of the Command APDU’

Table 6.45 Tags for authentication data objects

Tag	Meaning
'8E'	Cryptographic checksum
'9A', 'BA'	Initial value for a digital signature
'9E'	Digital signature

Table 6.46 Tags for confidential data objects

Tag	Meaning
'82', '83'	Cryptogram; the plaintext is BER-TLV coded and includes data objects for secure messaging
'84', '85'	Cryptogram; the plaintext is BER-TLV coded and does not include data objects for secure messaging
'86', '87'	Indicates the padding method used: '01' – padding with '80 00 . . . ' '02' – no padding

messaging, fall under the heading of ‘confidentiality’. The tags listed shown in the above tables must be used for secure messaging according to the type of method used.

Data objects for auxiliary functions

The data objects for auxiliary functions are used in secure messaging to coordinate the general constraints. The two parties use these data objects to exchange information about the cryptographic algorithms and keys used, initial data and similar basic information. In principle, these items can be different for each transmitted APDU, or even between a command and its response. In practice, though, auxiliary function data objects are rarely used, since all of the general constraints for secure messaging are defined implicitly, so they do not have to be specifically defined during communications.

Based on the options for secure messaging specified in ISO/IEC 7816-4, which have been only briefly outlined above, we can describe two fundamental procedures. We have kept these descriptions as simple as possible in order to make it easier to understand the complex mechanisms involved. Due to the high degree of flexibility provided by the standard, there are many other possible combinations of security mechanisms, some of which are even more complex. The two procedures described here represent a compromise between simplicity and security.

The ‘authentic mode’ procedure uses a cryptographic checksum (CCS or MAC) to protect the application data (APDU) against manipulation during transmission. The ‘combined mode’ procedure, by contrast, is used to completely encrypt the application data, so that an attacker cannot draw any conclusions about the data content of the commands and responses that are exchanged. A send sequence counter is only used with one of these two procedures. This counter, whose initial value is a random number, is incremented for each command and each response. This allows both parties to determine whether a command or response has been omitted or inserted. When a send sequence counter is used in combination with the 'combined mode' procedure, identical APDUs appear to be different. This is called 'diversity'.

6.6.1 The authentic mode procedure

The authentic mode procedure guarantees authentic transmission of APDUs, which means that the APDUs are protected against manipulation during transmission. The recipient of an APDU, which means a command or a response, can determine whether it has been altered during transmission. This makes it impossible for an attacker to modify data within an APDU without this being noticed by the recipient.

The fact that this procedure is being used is indicated by a bit in the class byte, so that the recipient can act accordingly and check the received APDU for authenticity. The actual APDUs are sent in plaintext and are not encrypted. The transmitted data are thus still public, and with suitable manipulation of the transmission channel they could be intercepted and evaluated by an attacker. This is not necessarily a disadvantage, since with respect to privacy legislation it is better not to send confidential data via a public channel. In addition, the card user is at least theoretically allowed the possibility of seeing what data are exchanged between his or her smart card and the terminal.

In principle, any block encryption algorithm can be used to compute the cryptographic checksum. For practical reasons, we assume that DES is used with a fixed 8-byte block length. The individual data objects must therefore be 'filled out' to an integer multiple of eight bytes, which is known as padding. In this process, data objects that are already an integer multiple of eight bytes are nevertheless extended by one block. After padding, the cryptographic checksum (CCS) of the entire APDU is computed using the DES algorithm in CBC mode. This 8-byte checksum is appended directly to the APDU as a TLV-coded data object, with the four least significant bytes omitted. All padding bytes are deleted after the checksum has been computed. The modified APDU is then sent via the interface. This procedure extends the length of the APDU by eight bytes, which only marginally reduces the transmission rate if normal transmission block sizes are used.

The data objects for the control structures can also explicitly identify the algorithm and padding method that are used. Here again we assume for the sake of simplicity that the smart card and the terminal implicitly know all the parameters of the secure messaging system being used.

When the protected APDU arrives at the recipient, the latter again pads it to an integer multiple of eight bytes and then computes its own MAC for the APDU. By comparing the MAC it has generated with the MAC generated by the sender, the recipient can determine whether the APDU has been altered during the transmission.

A prerequisite for computing a cryptographic checksum is a secret DES key that is known to both parties. If this key were not secret, an attacker would be able to break the authentic mode communication procedure by intercepting an APDU, modifying it as desired and computing a new 'correct' MAC. After this, he would only have to replace the original MAC with the new one and send the newly created APDU on its way.

In order to better protect the keys used to generate the MAC against attacks based on known plaintext–ciphertext pairs, dynamic keys are normally used. These are generated by encrypting

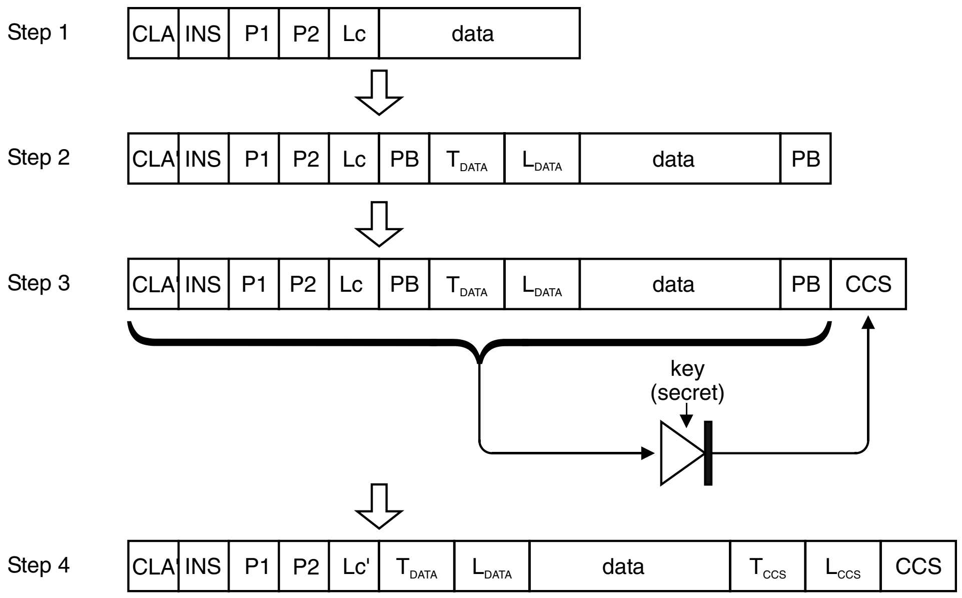

Step 1
CLA INS P1 P2 Lc data
Step 2
CLA INS P1 P2 Lc PB T_DATA L_DATA data PB
Step 3
CLA INS P1 P2 Lc PB T_DATA L_DATA data PB CCS
key (secret)
Step 4
CLA INS P1 P2 Lc' T_DATA L_DATA data T_CCS L_CCS CCS

Figure 6.46 Generating a command APDU using the authentic mode procedure. This example uses a case-3 command (e.g. UPDATE BINARY), with its header included in the cryptographic checksum (CCS). A response APDU can be generated in a similar manner. 'PB' indicates the padding bytes

Step 1:	The initial format of the APDU.
Step 2:	The data section is converted into TLV-coded data, and the data objects are padded to an integer multiple of eight bytes.
Step 3:	The CCS is computed.
Step 4:	A TLV-coded data object containing the CCS is added to the APDU.

a random number that has been previously exchanged between the terminal and the card. A secret key known to both parties is used for this encryption.

The additional steps that are needed for the transmission and reception of an APDU that is protected by the authentic mode procedure naturally reduce the effective data transmission rate. On average, a good approximation is to assume that the rate will be half of that for unprotected plaintext.

6.6.2 The combined mode procedure

Compared with the authentic mode procedure, the combined mode procedure represents the next higher level of security. The data section of the APDU is no longer transmitted as plaintext, but instead in an encrypted form. The procedure is an extension of the authentic mode procedure.

In the combined mode procedure, as in the authentic mode procedure, the data objects to be protected with a cryptographic checksum are first padded to an integer multiple of eight bytes and then encrypted using the DES in CBC mode. The header is excluded from this process, as required for compatibility with the T = 0 protocol. (If it is desired to encrypt the header as well, so that the command being sent the card is unrecognizable, the T = 0 ENVELOPE command must be used.) One bit in the class byte indicates the use of secure messaging. The data are transmitted across the interface after they have been encrypted. Since the recipient knows the secret key that was used for encryption, it can decrypt the APDU. The recipient then checks the correctness of decryption by recomputing the appended cryptographic checksum in the same level of the transmission layer.

When this procedure is used, an attacker eavesdropping on the I/O line cannot discover which data are exchanged between the card and the terminal in the command and response. It is also not possible to replace one of the encrypted blocks within the APDU, since the

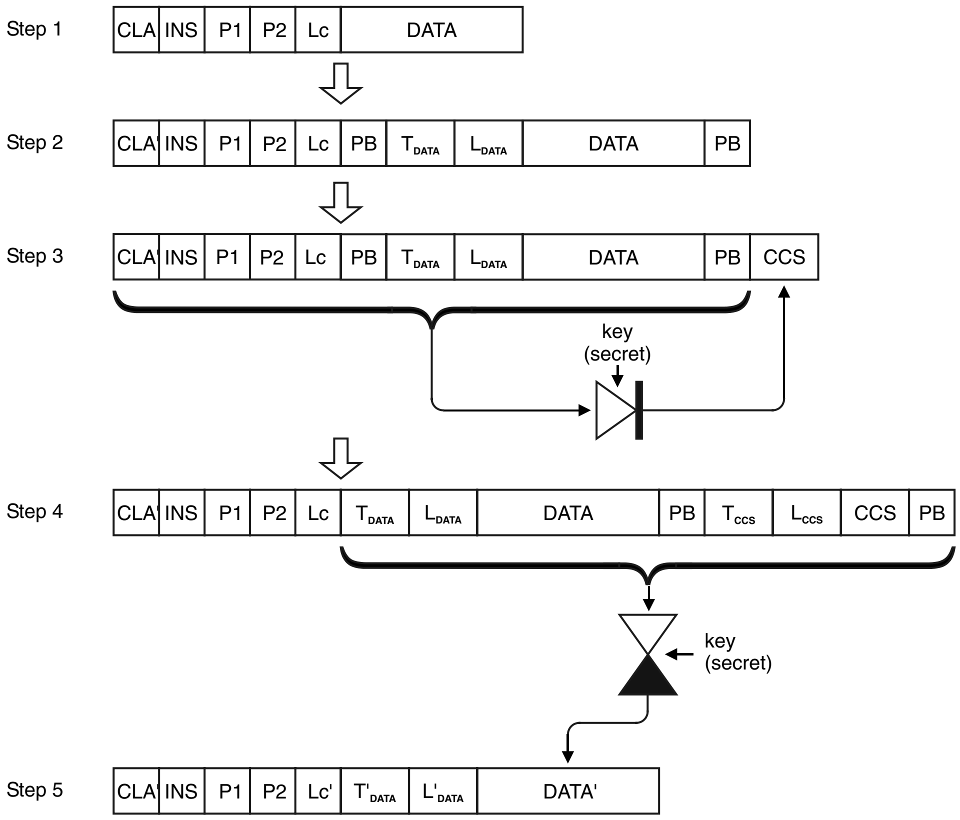


Figure 6.47 Generating a command APDU using the combined mode procedure. This example uses a Case 3 command (e.g. UPDATE BINARY), with its header included in the cryptographic checksum (CCS). A response APDU is created in a similar manner. The padding bytes are indicated as 'PB'.

Step 1: The initial format of the APDU.
Step 2: The data section is converted into TLV-coded data, and the data objects are padded to an integer multiple of eight bytes.
Step 3: The CCS is computed.
Step 4: A TLV-coded data object containing the CCS is added to the APDU.
Step 5: The APDU data section is encrypted.

blocks are linked to each other by using the DES in CBC mode. Any replacement would be immediately noticed by the recipient.

With regard to the cryptographic algorithm, the comments made in the description of the authentic mode procedure apply here as well. In principle, any block encryption algorithm can be used. The keys should be dynamic, as with the authentic mode procedure, which means that session-specific derived key should be used for every session.

With regard to the security benefits, general usage of the combined mode procedure for all APDUs can be recommended. However, increased security is accompanied by a considerable reduction of the effective transmission rate. A good approximation for the difference in the transmission rates for unprotected APDUs and those protected using the combined mode procedure is a factor of four. The speed difference between the authentic mode and combined mode procedures thus amounts to a factor of two. It is therefore necessary to carefully examine each case, in order to determine which data should be transmitted in such a secure but time-consuming fashion.

6.6.3 Send sequence counter

Using a send sequence counter mechanism for secure messaging does not by itself constitute a security mechanism. It only makes sense to use a send sequence counter in combination with the authentic mode or combined mode procedure, since otherwise any modification of the count by an attacker would remain undetected.

The working principle of a send sequence counter is that each APDU contains a sequence number that depends on when it is sent. This allows the deletion or insertion of an APDU in the course of the procedure to be immediately noticed, so that appropriate measures (terminating the communications) can be taken by the recipient.

This function is based on a counter that is initialized with a random number. This number is sent to the terminal by the card at the start of the communications process, in response to a request from the terminal. The counter is incremented each time an APDU is sent. The counter should not be too short, but it should also not be too long, in order to avoid generating excessive transmission overhead. The following description assumes the commonly used value of two bytes, but longer counters may be used in practice.

There are two basic ways to incorporate a sequence count into command and response APDUs. The counter value can be placed directly in the APDU as a numerical value in a data object, or the counter value may be XOR-ed with a matching amount of data in the APDU, following which a cryptographic checksum is computed and the modified data are restored to the APDU. The recipient of this APDU knows the expected counter value, and can use this value to modify the APDU in the same way as the sender. After this it can compute the cryptographic checksum and check the correctness of the received APDU.

The following process takes place during communications. The terminal first requests an initial counter value from the smart card. The smart card returns a two-byte random number to the terminal. The terminal then sends the first secured command to the card, accompanied by a send sequence count. Either the authentic mode or combined mode procedure can be used to protect the counter and the body. The card receives the protected APDU and first checks whether there is any sign of manipulation, based on the authentic mode or combined mode

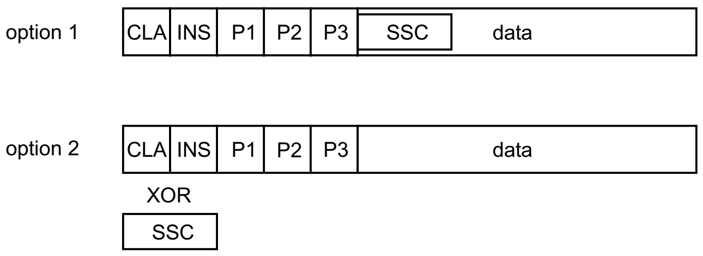


Figure 6.48 Two options for a send sequence count in a command APDU. In the first option, the send sequence counter is a TLV-coded data object in the data section. In the second option, the send sequence count is only coupled to the APDU data by an XOR operation used to compute the CCS

procedure. It then compares the counter value to the expected value. If these values match, no APDU has been inserted or deleted during the transmission.

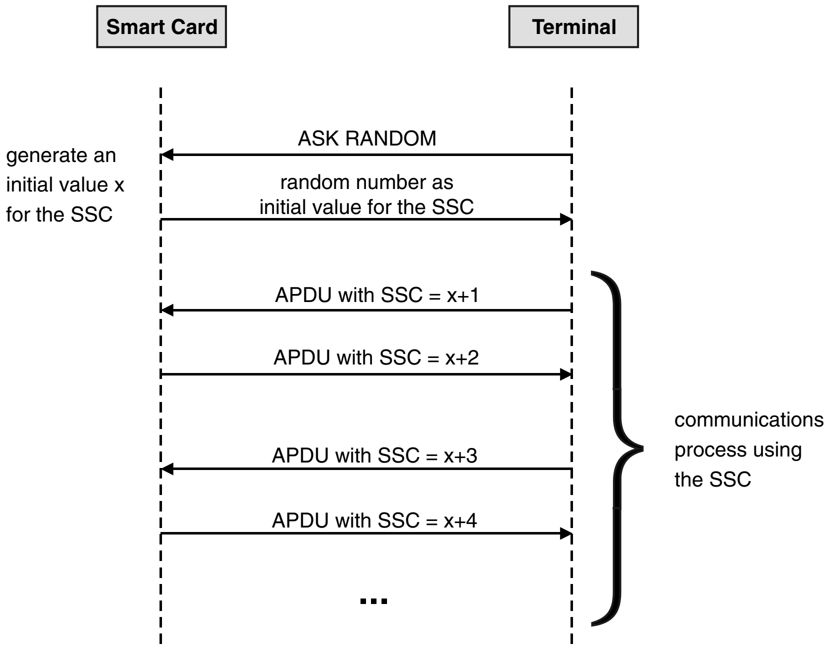

Smart Card
Terminal
ASK RANDOM
generate an initial value x for the SSC
random number as initial value for the SSC
APDU with SSC = x+1
APDU with SSC = x+2
communications process using the SSC
APDU with SSC = x+3
APDU with SSC = x+4
...

Figure 6.49 Transmitting APDUs using a send sequence counter (SSC)

It is apparent that using a send sequence counter is attractive not only when several commands have to be executed in a particular order, but also for individual commands, since each session is made unique if a counter is used. Using a counter primarily provides protection against 'replaying' previously sent APDUs and deleting APDUs.

If a send sequence counter is used together with the combined mode procedure, each encrypted block is different, which creates a condition known as diversity. This is the result of incrementing the counter for each APDU and the fact that with a good encryption algorithm, changing a single bit in the plaintext affects the appearance of the entire ciphertext block.

6.7 LOGICAL CHANNELS

In smart cards containing several independent applications, it is optionally possible to address these applications via logical channels. If logical channels are used, up to four applications in a single card can concurrently exchange data with the terminal. The existing single serial interface is still used, but the applications can be addressed individually at the logical level.

Two bits in the class byte (bit 1 and bit 2) are used to determine which command belongs to which application.[23] This permits up to four logical channels,[24] so up to four sessions with applications in the card can run in parallel. However, there is a limitation with regard to communicating with the various applications in the smart card, which is that the external processes that access the card must be mutually synchronized and are not allowed to interleave their commands, since the response APDU from the card does not contain any information about the logical channel of the originating command. This means that it is impossible to externally determine which return code has been sent back in response to which command. Due to the absence of channel identification, no new command can be sent until the response to the previous command has been received.

The primary application for this very powerful mechanism is using several applications in parallel. For example, suppose a cardholder is conducting a telephone conversation using the GSM application in a multiapplication smart card. In order to confirm an appointment with the other party, she needs to briefly consult her personal organizer, which is located in the same card. Using a second logical channel, the terminal searches for a file in the personal organizer application, in parallel with the GSM application, and then tells our highly stressed manager whether she can agree to the proposed date. This is a typical use of logical channels. Another conceivable example is securely transferring electronic funds between two electronic purses in the same card.

The potential utility of logical channels for applications is matched by the difficulties that their management entails for the smart card operating system. In principle, each logical channel represents nothing less than a separate smart card, with all of its states and conditions. This effectively means that the operating system must concurrently manage all the data for several parallel sessions within its memory. The associated costs should not be underestimated, and in particular this requires microcontrollers with large amounts of RAM. If secure messaging and all possible types of authentication are also required for each logical channel, the amount of memory needed very quickly rises to a level that can only be met by the highest-performance types of currently available smart card microcontrollers.

[23] See also Section 6.5.1, ‘Structure of the command APDU’

[24] There is a proposal to revise and upgrade the ISO/IEC 7816-3 standard to increase the number of possible logical channels to eight by using an additional RFU bit

7 Smart Card Commands

Communications procedures between a terminal and a smart card are always based on the master–slave principle. This means that the terminal, acting as the master, sends a command to the card, which as the slave immediately processes the command, generates a response and returns its response to the terminal. The card thus never sends data without first having received a corresponding command from the terminal. Even the ATR is no exception to this rule, since it is a response to the reset signal, which is also a type of command to the card.

Actual communications always employ a transmission protocol, such as T = 0 or T = 1. These relatively uncomplicated protocols meet the special requirements of smart card applications and are optimized for this purpose. Deviations from these precisely specified protocols within application procedures are not permitted. The transmission protocols allow data to be sent to the card and received from the card in a manner that is completely transparent to the transport layer. The data are embedded in a sort of container called an application protocol data unit (APDU). APDUs sent by the terminal to the card are the commands to the card. The terminal also receives the responses to its commands in APDUs embedded in the transmission protocol. There are a large number of commands based on this mechanism, and they initiate specific activities within the card. The simplest examples are read and write commands for smart card files.

In smart card applications, the card is used as a data storage medium, an authorization medium or both at the same time. This has led to the generation of command sets that are optimized for these applications and transmission protocols and that are used only in the smart card realm. Due to the severely limited memory capacity of smart cards, combined with market pressure to allow only moderate increases in this capacity for cost reasons, command sets are usually tailored to specific applications. All commands that are not needed in a given application are relentlessly removed during program optimization. Only a few operating systems exhibit extensive command sets that have not been reduced to those needed for a particular application.

A diversification effect is also seen with smart card command sets, as is typical with new technologies. Each company active in this area attempts to create its own commands that are tailored to the needs of its operating system or anticipated application. This often arises from necessity, since functionally equivalent commands may not exist in the standards. Companies may also deliberately attempt to improve their positions relative to the competition, or to deny their competitors access to a particular application area, by using commands that are

Smart Card Handbook, Third Edition. W. Rankl and W. Effing
© 2004 John Wiley & Sons, Ltd ISBN: 0-470-85668-8

highly optimized with regard to card functions and memory usage. In any case, a decision to use commands based on available standards always means choosing an open, more easily expandable and proven system, which may later allow additional functions to be incorporated into a single card. On the other hand, there are many examples of systems in which the use of smart cards was only made possible by using highly optimized special commands.

There are currently 13 international standards and more or less stable draft versions of standards that define typical smart card commands. They define considerably more than 100 commands, along with their associated procedures. To a large extent, the defined commands are mutually compatible in terms of coding and functionality.

The majority of the commands currently used with smart cards are defined in the ISO/IEC 7816-4 standard, which is a general international standard. It is not dedicated to any particular area, such as telecommunications or financial transactions, but instead attempts to address all smart card applications. The commands in ISO/IEC 7816-4 are complemented by three supplementary, specialized sections of this family of standards. ISO 7816-7 defines commands for querying and managing smart card databases with structures based on structured query language (SQL). ISO/IEC 7816-8 contains commands for executing and parameterizing cryptographic functions, and Part 9 of the ISO/IEC 7816 family adds file management commands to the basic command set.

There is no significant international standard in the area of financial transactions, but there is an industry standard. This is the EMV specification, whose name comes from the initial letters of Europay, MasterCard and Visa, the three initiators of this specification. Due to the strong market position of the companies behind it, this specification has achieved the status of a reference for all smart card operating systems, and it has the same degree of significance as the ISO/IEC 7816 family of standards.

The GSM 11.11 specification, which was developed for use in the telecommunications area, forms the normative basis for the SIM, while the TS 13.101, TS 31.111 and TS 102.222 standards specify the basic commands for the USIM. Due to the simple fact that a tremendous number of smart cards are used in telecommunications applications, these standards represent a *de facto* standard for commands for smart card operating systems.

In principle, special commands used only in a restricted area are not covered by these standards and must therefore be specified individually. One example is the command set specified for multisector electronic purse systems, which is defined in the CEN EN 1546 standard. This European standard defines all commands necessary for an electronic purse, along with the associated procedures. A standard such as this, which is limited to a single application, arises only in areas of particular interest to government agencies or specific branches of industry, due to the very high cost of generating such standards.

The commands in the standards and specifications described above can be classified according to their functionalities. However, it must be remembered that only subsets of all these commands are implemented in real-life smart card operating systems. Depending on the producer of the operating system, more or less significant deviations from the functionality and coding described in this chapter may be encountered. However, the basic functions described here are in principle present in all operating systems. Of course, the functionality may be severely restricted due to considerations of memory capacity or cost. Whenever a new application is being planned, exact specifications of the coding and functions of all of the commands must be requested from the producer(s) of the operating system(s) under consideration.

The following sections describe the most important and most widely used smart card commands. The basis for this selection is formed by the following standards and specifications:

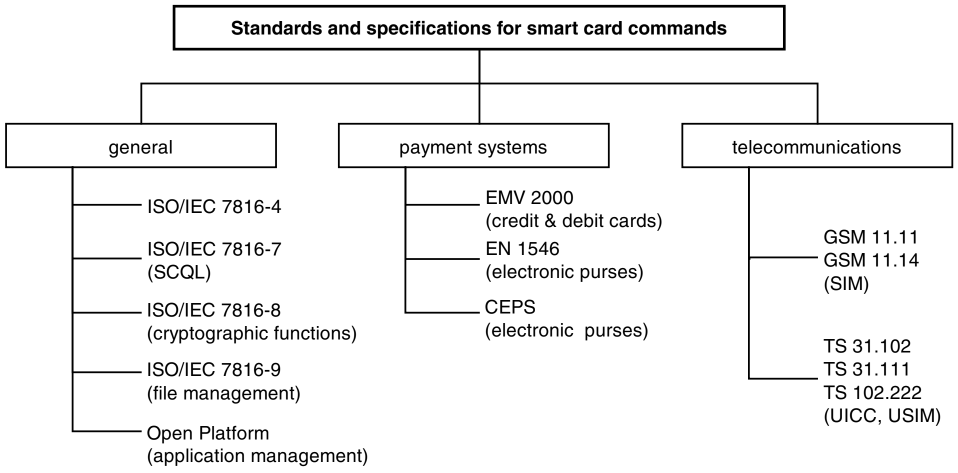

Standards and specifications for smart card commands
general
payment systems
telecommunications
ISO/IEC 7816-4
ISO/IEC 7816-7 (SCQL)
ISO/IEC 7816-8 (cryptographic functions)
ISO/IEC 7816-9 (file management)
Open Platform (application management)
EMV 2000 (credit & debit cards)
EN 1546 (electronic purses)
CEPS (electronic purses)
GSM 11.11 GSM 11.14 (SIM)
TS 31.102 TS 31.111 TS 102.222 (UICC, USIM)

Figure 7.1 The most important standards and specifications for smart card commands

ISO/IEC 7816-4/7/8/9, EMV 2000, GSM 11.11, TS 311.111, EN 726–3 and EN 1546–3. Extensive tables listing the coding of the most important smart card commands can be found in Section 16.10.7, 'Smart card command encoding'.

Naturally, it is impossible to buy a single smart card anywhere in the world that contains all the commands described here. As a conservative estimate, the memory required for their full implementation would be five to 10 times as large as the total amount of memory in the largest currently available smart card microcontrollers. However, it is not at all necessary for a smart card to be able to execute all of these commands. Depending on the intended application area and operating system, certain classes of commands may be supported more comprehensively than others.

For example, with a multiapplication card you would certainly want to make sure that additional applications can be installed in the card after it has been personalized. A card for cryptographic applications, assuming it has adequate memory capacity, will contain the full spectrum of cryptographic commands, along with the various algorithms. Each application area requires a different selection of commands from the various classes.

In each of the following descriptions, the standard or specification in which the command is defined is identified in order to maintain an overview. If no source is given, the command in question is used internally by smart card manufacturers and cannot be assigned to any of the above-mentioned standards. Some of these commands are nonetheless very useful and will probably be incorporated into a standard in the future. They thus are listed here with descriptions of their basic functionality. In the interest of readability, we have omitted descriptions of the coding of typical smart card commands in this chapter. This information can be found in Section 16.10.7, 'Smart card command encoding'.

Some commands are supported by nearly all smart card operating systems and have only a limited number of options. For such commands, the command and response APDUs are fully decoded in their descriptions. The internal processing sequences of seven typical commands are also shown in psuedocode in Chapter 5 in the description of the 'Small OS' operating system.

For a given application, smart card commands can be classified into operational commands, which are commands needed for normal use, and administrative commands, which are commands needed for managing the smart cards and the application. For reasons related to interoperability, operational commands are generally specified in complete detail in the standards. Administrative commands are often specific to particular operating systems and are not necessarily specified by standards.

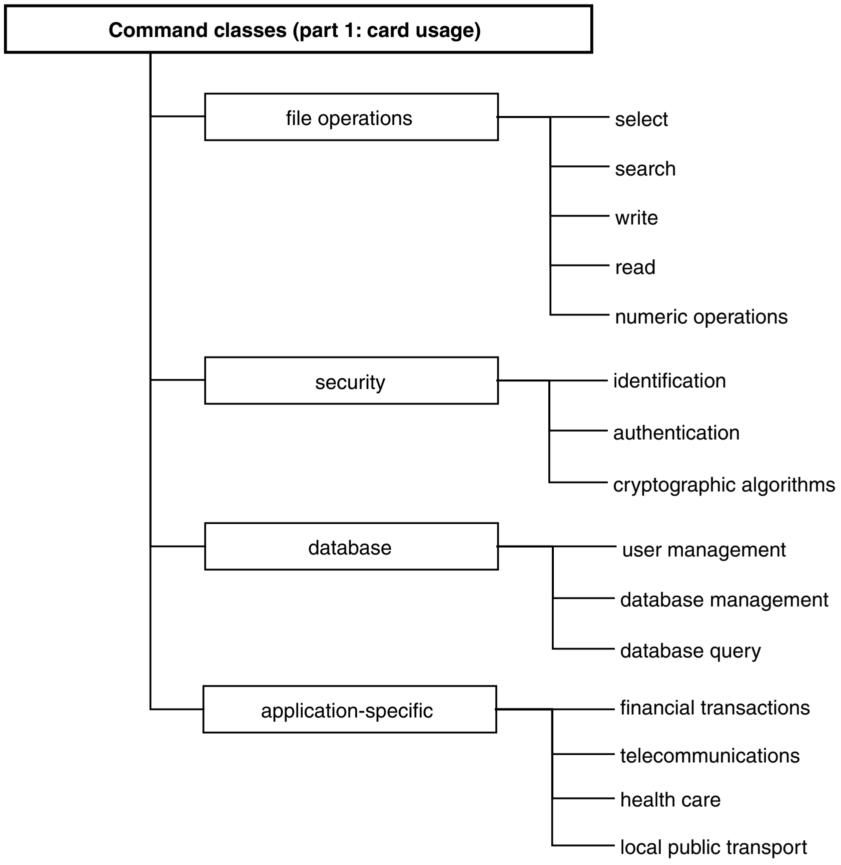


Figure 7.2 Classification of smart card commands that are primarily used for operational functions (while the card is in actual use)

For each command, the response shown in its description is the one received by the terminal in the event of successful execution. Otherwise, if an operation is forbidden or an error occurs in the card, the terminal receives only a 2-byte return code. Some of the described commands also have parameters for selecting additional functions. These options often exist only in the standard, rather than in actual operating systems, since they may be too complicated or have no practical significance. Therefore, this chapter does not list or explain every option defined in the standards, since our aim is to concentrate on practical functions. In describing the commands,

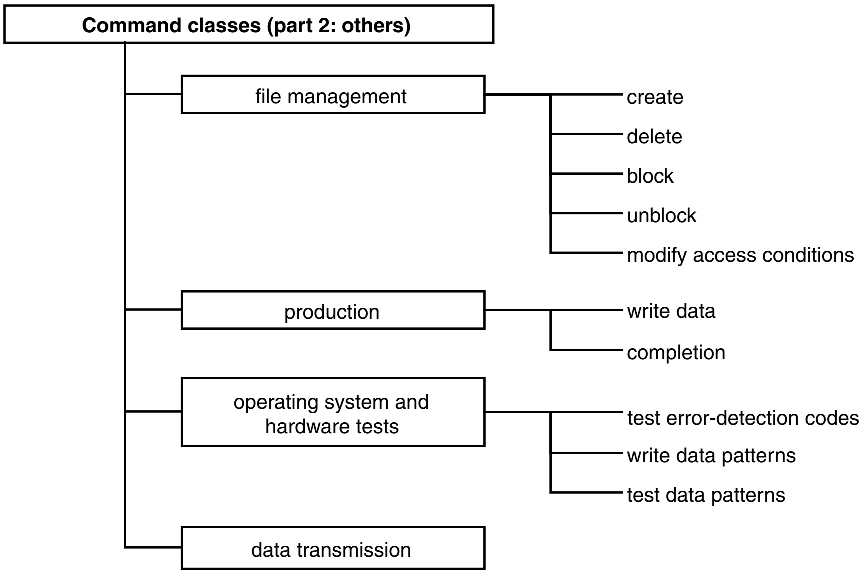


Figure 7.3 Classification of smart card commands that are primarily used for administrative functions (before and after the smart card is in actual use)

we have generally used the standard that has the largest number of functional options for the command in question.

7.1 FILE SELECTION COMMANDS

Without exception, file management in all modern smart card operating systems is object-oriented. Among other things, this means that before any action can be performed on an object (which corresponds to a file), it must first be selected. Only then does the system know which file is meant, and all subsequent file-specific commands apply to this file alone. Of course, the access conditions for the file still must be checked within the operating system, in order to determine whether the command in question is allowed or even possible.

The master file (MF) is always implicitly selected after the card has been reset, so it does not have to be specifically selected. Other files are subsequently selected by executing the SELECT FILE command. A file is addressed using its 2-byte file identifier (FID) or, in the case of a directory file (DF), a 1-byte to 16-byte DF name. A DF name can contain an internationally unique application identifier (AID) that is 5 to16 bytes long. It is possible to pass only a portion of the AID, omitting the less significant bytes (those to the right). An additional parameter causes the card to select the first, last, next or previous DF relative to the DF identified by the abbreviated AID.

In connection with an older set of command definitions, the GSM 11.11 standard allows only the 2-byte FID to be used for file selection. The ISO command set, by contrast, also supports a type of file selection using the path name of the file in question. The path name can be either relative, in which case the file is selected starting from the currently selected DF, or absolute, in which case the file is selected starting from the MF.

Only successful selection of a new file causes the previously selected file to be deselected. If the selection cannot be completed, for instance because the requested file does not exist, the previous selection remains in force. This ensures that a file is always selected, even in the event of an error.

After successful file selection, the terminal may request information about the new current file if necessary. This request, including the desired number of data items, is sent to the card using the SELECT FILE command. The exact contents of these data items are defined in the applicable standard. The data items returned by the card may include information about the structure, size and amount of free memory of the newly selected file. The amount of data may also depend on the file type.

Table 7.1 lists the explicit file selection options permitted by ISO/IEC 7816-4 for the SELECT FILE command, and Figure 7.4 depicts the sequence of events in a typical file selection process.

Table 7.1 The functionality of SELECT FILE according to ISO/IEC 7816-4

SELECT FILE	
Command	• FID (if EF, DF or MF) *or* DF name (if DF) *or* path to file from currently selected DF *or* path to file from MF *or* *switch:* select next higher-level DF *or* first, last, next, or previous DF (if a partial AID is transferred) • *switch:* return information about the selected file
Response	• information about the selected file (if selected via the switch) • return code

In addition to explicit file selection using an FID, DF name or a path specification in a SELECT FILE command, implicit file selection can be used. However, this is only possible with standard read and write commands. A file can be selected before the command is actually executed by specifying its 5-bit short FID as a supplementary parameter. However, in this case the file must be an EF and it must be located within the current DF. The advantage of this method lies in simplified command execution and increased processing speed, since it is not necessary to send an explicit SELECT FILE command to the card.[1]

[1] See also Section 5.6.2, 'File names'

Smart card		**Terminal**

	←	SELECT FILE
		Command [FID='3F 00';
		no additional file information necessary]

Search for the file with FID ='3F 00'
IF (file found)
THEN return code = OK
ELSE return code = file not found
Response [return code]
	→	IF (return code = OK)
		THEN file selection successful
		ELSE file could not be selected

Figure 7.4 Sample processing sequence for the SELECT FILE command

GSM 11.11 defines the STATUS command, which returns the same data to the terminal as successful file selection using SELECT FILE. These data provide information about the currently selected file: its type and structure, size, FID, access conditions and whether it is blocked. This command is rarely used, and its main purpose is to allow the terminal to determine which file is currently selected during a session and the currently valid access conditions.

Table 7.2 The functionality of STATUS according to GSM 11.11

STATUS	
Command	• —
Response	• information about the currently selected file • return code

EN 726–3 specifies the CLOSE APPLICATION command, which supplements SELECT FILE and STATUS and is used to close applications. The FID of the application to be closed is provided with the command, and the card responds by deleting the previously attained security state. This command is mainly useful when a terminal needs to ensure resetting of the state attained by the card. If the card's operating system does not support such a command, this result can only be achieved by a card reset. In the ISC/IEC 7816-4 definitions, selecting the MF is sufficient to cause the security state of the previously selected file to be reset to its initial state.

Table 7.3 The functionality of CLOSE APPLICATION according to EN 726-3

CLOSE APPLICATION	
Command	• FID (of the current DF)
Response	• return code

7.2 READ AND WRITE COMMANDS

Read and write commands primarily support using smart cards for secure data storage. These commands can be used to write data to appropriate EFs and subsequently read these data. If these EFs have specific access conditions, only authorized users are allowed to read them. The data are thus stored in the card with protection against unauthorized access.

Since there are various types of EF data structures, there are also various types of read and write commands for these files. Unfortunately, this does not fully correspond to an object-oriented file management system. In a purely object-oriented system, the operating system must be built such that an object can determine its own access mechanisms. This is not the case for smart card file management. This non-compliance dates back to the historical emergence of commands that were subsequently incorporated into current standards. The precursors of smart cards, which are memory cards, have only one memory region that can be read and written using offset and length parameters. Externally, this memory can be regarded as a single file with a transparent structure. The first smart cards were built according to the same principle, and the definitions of the commands for reading and writing transparent files date from this time. Later, when other types of file structures were defined, new commands specifically adapted to these structures were defined for use with such files. As a result, there are two different types of file access.

Therefore this class of commands must be divided into commands for accessing EFs with transparent structures and commands for accessing EFs with the other types of structures (cyclic, linear fixed and linear variable). However, several standards (such as EN 1546 for electronic purses) explicitly state that read commands for files with transparent structures may also be used to read files with other types of structures. In any case, such commands can be used to obtain additional data about the internal structure of the file.

An EF with a transparent logical structure is amorphous, which means that it does not have any internal structure. It corresponds to a linearly addressable memory with byte access. The READ BINARY command is used to read such a file, while the WRITE BINARY and UPDATE BINARY commands are used for writing.

Table 7.4 The functionality of READ BINARY according to ISO/IEC 7816-4

READ BINARY	
Command	• number of bytes to be read • offset to the first byte to be read • *optional:* short FID for implicit selection
Response	• data read from the file • return code

The fundamental difference between the WRITE BINARY and UPDATE BINARY commands relates to the secure state of the card's EEPROM. The secure EEPROM state is the logical state of the EEPROM bits when the memory cells have taken on their minimum-energy state. Since the memory cells are small capacitors, this means the state in which they contain

Table 7.5 The functionality of WRITE BINARY according to ISO/IEC 7816-4

WRITE BINARY	
Command	• number of bytes to be written • data bytes to be written • offset to the first byte to be written • *optional:* short FID for implicit selection
Response	• return code

Table 7.6 The functionality of UPDATE BINARY according to ISO/IEC 7816-4

UPDATE BINARY	
Command	• number of bytes to be overwritten • offset to the first byte to be overwritten • *optional:* short FID for implicit selection
Response	• return code

no charge, which is usually the logic 0 state. In order to change a bit from state 0 to state 1, it must be erased. This restores the charge on the capacitor.

A WRITE command can only be used to change bits from the non-secure state, which is usually logic 1, to the secure state, which is usually logic 0. In this case, the WRITE command produces the logical AND of the transferred data and the file content. By contrast, if the secure state of the chip corresponds to logic 1, the WRITE command must produce the logical OR of the data provided by the command and the data in the file. The result of the logical coupling of the data provided by the command and the data in the file is that the secure state of the EEPROM is always achieved using a WRITE command. In addition, a WRITE command may support write once, read multiple (WORM) access, depending on the file. WRITE commands originate from a time when using atomic operations for file access was still unknown in the smart card realm. They are presently used only very rarely.

An UPDATE command, by contrast, performs a genuine write to the file. The previous state of the data in the file does not affect the content of the file following execution of an UPDATE command. UPDATE BINARY can thus be considered to be equivalent to using ERASE to erase the file, followed by WRITE BINARY.

These commands can be utilized to construct physically secure smart card counters. The principle involves a bit field in which each bit that is set represents a monetary unit. When a payment is made, the counter is decremented bit by bit using OR operations generated by WRITE BINARY commands. After authentication, the counter value can be increased again using an UPDATE BINARY command. The main advantage of this technique is that it makes it impossible to increase the counter value by manipulating the EEPROM, such as by heating it, since the secure state of each bit represents a value of 0.

As their names suggest, READ BINARY is a read command, while WRITE BINARY and UPDATE BINARY are write commands. The file is always accessed using a length parameter and an offset to the first byte to be addressed. Some operating systems also permit implicit file selection before the actual data access occurs, using a supplementary short FID parameter. However, this option is not present in all standards and operating systems.

Figure 7.5 illustrates a typical command sequence using READ BINARY followed by WRITE BINARY and finally UPDATE BINARY. The effects on the content of the selected file are shown in Figure 7.6. Of course, this example assumes that file selection is successful and that the access conditions for the file to be written are met.

Smart card		Terminal
	←	READ BINARY *Command* [offset = 2 bytes, number of bytes to be read = 5]

requested data := '03' || 'FF' || '00'
|| 'FF' || '00'
Response[requested data || return code]
	→	IF (return code = OK)
		THEN READ BINARY successful
		ELSE abort

	←	WRITE BINARY *Command* [offset = 3 bytes, number of bytes to be written = 2, data = 'F0 F0']

Response [return code]	→	IF (return code = OK)
		THEN WRITE BINARY successful
		ELSE abort

	←	UPDATE BINARY *Command* [offset = 5 bytes, number of bytes to be written = 2, data = 'F0 F0']

Response [return code]	→	IF (return code = OK)
		THEN UPDATE BINARY successful
		ELSE abort

Figure 7.5 Accessing a file with a transparent structure

ERASE BINARY is an exception among the commands that operate on transparent EFs. It cannot be used to write data to a file, but only to erase data starting from a given offset. If no second offset parameter is stated, the command erases all data to the end of the selected file. In this case, erasing data means that the data region specified in the command is set to the logical erased state. This state must be defined separately for each operating system, since it may not be the same as the physically erased state of the memory.

Because the structures of linear fixed, linear variable and cyclic EFs are fundamentally different from the structure of a transparent EF, special commands for accessing these particular data structures are available in addition to the commands described above. All of these files have record-oriented structures. For write access, the smallest addressable unit in the data field is a single record. For read access, either an entire record or part of a record may be read,

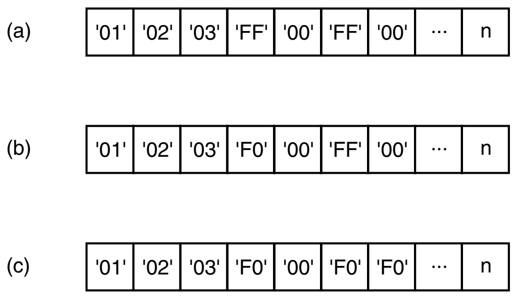

(a)
'01' '02' '03' 'FF' '00' 'FF' '00' ··· n
(b)
'01' '02' '03' 'F0' '00' 'FF' '00' ··· n
(c)
'01' '02' '03' 'F0' '00' 'F0' 'F0' ··· n

Figure 7.6 Example of write accesses to an EF with a transparent structure using the sequence of commands shown in Figure 7.5
(a) file content with READ BINARY
(b) file content after WRITE BINARY
(c) file content after UPDATE BINARY

Table 7.7 The functionality of ERASE BINARY according to ISO/IEC 7816-4

ERASE BINARY	
Command	• offset to the first byte to be erased • *optional:* offset to the last byte to be erased • *optional:* short FID for implicit file selection
Response	• return code

starting with the first byte of the record. These file structures, which transform a linear, one-dimensional memory into a memory that can be addressed in two dimensions, yield access types that are significantly more complex than those used with a transparent structure. In principle, all possible data structures can be emulated using a transparent structure, but in specific cases this may prove considerably more complicated than using a higher-level structure.

After an EF with a record-oriented structure has been selected, the card's operating system creates a record pointer whose initial value is undefined. The value of this pointer can be set using a READ, WRITE, UPDATE RECORD or SEEK command. The pointer for the current file is saved as long as this file is selected. After successful explicit or implicit selection of another file, the value of the record pointer is again undefined.

All commands for record-oriented files can use a parameter byte to specify the type of access to the file. The basic type is direct access using the absolute number of the desired record. This type of access does not alter the record pointer. The number of the desired record is sent to the card, and the response contains the content of the record in question.

If the parameter byte specifies the first record, the operating system sets the record pointer to the first record in the file, and this record is read or written according to the type of command used. The parameter value 'last' accesses the final record in a similar manner. The additional parameter values 'next' and 'previous' allow the next and previous records, respectively, to be selected and read or written. Finally, the parameter value 'current' can be used to address the record marked by the current value of the record pointer.

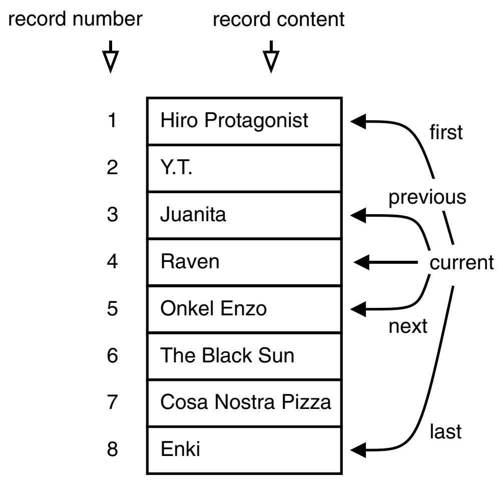


Figure 7.7 Accessing a file having a record-oriented structure

This large variety of access methods for record-oriented data structures originates from the typical structure of a telephone directory. Consider a record whose initial part contains a surname and given name, followed in the same record by the associated telephone number. Using READ RECORD and the parameters described above with a telephone directory mapped into an EF, you can 'page' forwards and backwards as desired within the directory or jump to the first or last entry. The record pointer can also be changed using the search command SEEK, which is described below.

name	telephone number
Joshua Calvert	47 84 46
Quinn Dexter	089 11 00 11
Louise Kavanagh	089 47 51 22
Alkad Mzu	089 178 098
Ione Saldana	08933 178 234
Kiera Salter	089 23 76 87 33
Ralph Hiltch	089 78 123 78 1
Fletcher Christian	089 12 111 222
Kelly Tirrel	089 92 178 234
Gerald Skibbow	089 129 167 189

Figure 7.8 Example of a telephone number list in a file with a 'linear fixed' structure

ISO/IEC 7816-4 also provides the option of reading all records from the first record up to a specified record number. Similarly, all records from a specified record number through to the last record can be read in a single command–response cycle using READ RECORD. Although these commands are very practical, the capacity of the I/O buffer can quite easily be exceeded if they are used with large files.

Figure 7.9 illustrates the execution of several read and write operations on the file shown in Figure 7.8.

Table 7.8 The functionality of READ RECORD according to ISO/IEC 7816-4

READ RECORD	
Command	• number of records to be read *or* mode (current, first, last, next or previous record) *or* read all records from *n* to the last record *or* read all records from the first record to *n* • *optional:* short FID for implicit file selection
Response	• data read from the file • return code

Table 7.9 The functionality of WRITE RECORD according to ISO/IEC 7816-4

WRITE RECORD	
Command	• record data for writing • number of the record to be written *or* mode (current, first, last, next or previous record) • *optional:* short FID for implicit file selection
Response	• return code

Table 7.10 The functionality of UPDATE RECORD according to ISO/IEC 7816-4

UPDATE RECORD	
Command	• record data for overwriting • number of the record to be overwritten *or* mode (current, first, last, next or previous record) • *optional:* short FID for implicit selection
Response	• return code

Smart card		**Terminal**
		READ RECORD
	←	*Command* [record number = "2"]
command processing		
Response ["Oliver" || return code]	→	IF (return code = OK)
		THEN READ RECORD successful
		ELSE abort
		UPDATE RECORD
	←	*Command* ["Wolfgang", firs,]
command processing		
Response [return code]	→	IF (return code = OK)
		THEN UPDATE RECORD successful
		ELSE abort
		UPDATE RECORD
	←	*Command* ["Alex", next]
command processing		
Response [return code]	→	IF (return code = OK)
		THEN UPDATE RECORD successful
		ELSE abort
		READ RECORD
	←	*Command* [record number = 2]
command processing		
Response ["Alex" || return code]	→	IF (return code = OK)
		THEN READ RECORD successful
		ELSE abort

Figure 7.9 Sample read and write operations for a record-oriented file

The APPEND RECORD command, given its functionality, could just as well be classified as a file management command. It can be used to append records to existing record-oriented files. The data for the entire new record are provided together with the command. A relatively complex memory manager, in smart card terms, is a prerequisite for the availability of this command with its full functionality. The function of the memory manager is to create a link between the new record and the ones already present in the file. It is then possible, within the limits of the available memory, to add an arbitrary number of new records. However, the number of new records that can be added is often restricted in order to simplify matters. In this case, when a record-oriented file is created, memory is reserved as necessary for adding future records. This space can later be filled using APPEND RECORD commands. Once this free space is used up, the APPEND RECORD command cannot be used again for this file.

If APPEND RECORD is used in conjunction with a linear fixed or linear variable file, the new record is always added at the end of the file. If the structure is cyclic, however, the new record is always numbered 1, which corresponds to the currently written record in files of this type.

APPEND RECORD can be used for various purposes. One possibility is a telephone directory, as already mentioned. Another possibility is a log file, in which the data to be recorded are written directly to the card by creating new records.

Table 7.11 The functionality of APPEND RECORD according to ISO/IEC 7816-4

APPEND RECORD	
Command	• record to be written • *optional*: short FID for implicit file selection
Response	• return code

There are two commands that complement the file-based read and write commands. They are designed for direct access to data objects. Depending on the selected DF, certain data can be written to or read from files or internal operating system structures, bypassing the file-oriented access mechanisms. Data objects can be written using PUT DATA and read using GET DATA. For both of these commands, the exact structure of the TLV-coded data objects must be transferred with the command. This means that it is necessary to know whether application-specific or standard coding is used for the data objects. This information is important inside the operating system, since it allows the objects to recognize the data according to how they are packaged. The appropriate access conditions must be satisfied in advance for both of these commands.

Table 7.12 The functionality of GET DATA according to ISO/IEC 7816-4

GET DATA	
Command	• number of data objects to be read • tag of the data objects to be read
Response	• read data objects • return code

Table 7.13 The functionality of PUT DATA according to ISO/IEC 7816-4

PUT DATA	
Command	• structure of the data objects to be written • data objects to be written
Response	• return code

7.3 SEARCH COMMANDS

Record-oriented structures are well suited to storing sets of related data with identical structures in a single file. A typical example is a telephone directory containing names and telephone numbers. A search command can be used to avoid having to read the entire directory, record by record, when looking for a particular name.

The SEEK command can be used to search for a specified character string in a record-oriented data structure. An offset can be supplied with the command. The length of the search string is variable. The command must tell the operating system in which direction to search. This can be either from the starting location onwards (in the direction of increasing record numbers) or from the starting location backwards. The starting location for the search must also be specified. The first record, last record or current record can be specified as the starting position. If the search string is found, the operating system sets the record pointer to the location of the string and informs the terminal that the search was successful.

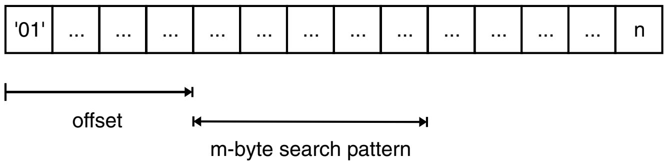


Figure 7.10 Searching within a record-oriented file

The ISO/IEC 7816-9 standard describes two commands that can be used to search for data in transparent and record-oriented files. The SEARCH RECORD command is the ISO/IEC version of the GSM 11.11 SEEK command. The major difference between these two commands is that according to ISO/IEC 7816-9, a short FID can also be transferred within the command for implicit EF selection.

The command sequence shown in Figure 7.11 illustrates some ways in which the SEEK command can be used. This example is based on the linear fixed file shown in Figure 7.7.

Table 7.14 The functionality of SEEK according to GSM 11.11

SEEK	
Command	• length of the search string • search string • offset • search mode (forward from the beginning, backward from the end, forward from the next position, backward from the previous position) • *switch:* return the record number of the located record
Response	• record number (if selected by the switch) • return code

The SEARCH BINARY command can be used to search for specific data in a selected transparent file. The file may be selected either explicitly by a previously transferred command

Smart card		**Terminal**
	←	SEEK *Command* [search string = "Hans" || search direction = "forward from the beginning" || send record number]

Response [record number = 8 || return code]	→	IF (return code = OK)
		THEN "Hans" found
		ELSE "Hans" not found

	←	SEEK *Command* [search string = "Alex" || search direction = "backward from the end" || send record number]

Response [record number = 1 || return code]	→	IF (return code = OK)
		THEN "Alex" found
		ELSE "Alex" not found

Figure 7.11 Sample command sequence using the SEEK command

Table 7.15 The functionality of SEARCH RECORD according to ISO/IEC 7816-9

SEARCH RECORD	
Command	• length of the search string • search string • offset • mode (forward from the beginning, backward from the end, forward from the next position, backward from the previous position) • *switch:* return the record number of the located record • *optional:* short FID for implicit file selection
Response	• record number (if selected by the switch) • return code

Table 7.16 The functionality of SEARCH BINARY according to ISO/IEC 7816-9

SEARCH BINARY	
Command	• length of the search string • search string • offset • *optional:* short FID for implicit file selection
Response	• offset to the located data • return code

or implicitly via a command parameter. The result is the offset from the start of the file to the first byte of the located search string.

7.4 FILE MANIPULATION COMMANDS

There are several commands that allow the content of a file to be modified by means other than simple writing. The main representatives of this class are the INCREASE and DECREASE commands. They increase or decrease the value of a cyclically structured file whose content takes the form of a counter. The amount of the increase or decrease is transferred as a command parameter.

The cyclic file structure is defined to meet the needs of logging functions. These commands are primarily used for simple electronic change purses and counters. For the sake of simplicity, an example cyclic file with only one record is shown in Figure 7.12. The starting value of the record is 10. On conclusion of the process, the record has the same value again.

Table 7.17 The functionality of DECREASE according to GSM 11.11

DECREASE	
Command	• value to be subtracted
Response	• subtracted value • new value of the record • return code

Table 7.18 The functionality of INCREASE according to GSM 11.11

INCREASE	
Command	• value to be added
Response	• added value • new value of the record • return code

The EXECUTE command can also be considered to be a file manipulation command in a certain sense. It is used to run executable EFs (whose structure is 'executable'). The program to be executed can receive data from the terminal via the command, and it can also send data that it generated back to the terminal as a response.

This command and the related file structure are controversial, since they can potentially be used to bypass the entire security system of a smart card.

Smart card		**Terminal**
	←	DECREASE *Command* [value to be subtracted = 3]

command processing
Response [subtracted value = 3 ||
new value = 7 || return code]
	→	IF (return code = OK)
		THEN DECREASE successful
		ELSE DECREASE could not be executed

	←	DECREASE *Command* [value to be subtracted = 2]

command processing
Response [subtracted value = 2 ||
new value = 5 || return code]
	→	IF (return code = OK)
		THEN DECREASE successful
		ELSE DECREASE could not be executed

	←	INCREASE *Command* [value to be added = 5]

command processing
Response [added value = 5 ||
new value = 10 || return code]
	→	IF (return code = OK)
		THEN INCREASE successful
		ELSE INCREASE could not be executed

Figure 7.12 Sample command sequence using INCREASE and DECREASE

Table 7.19 The functionality of EXECUTE according to EN726-3

EXECUTE	
Command	• data to be passed to the executable file
Response	• data returned by the executable file • return code

7.5 IDENTIFICATION COMMANDS

In addition to being used as secure data storage media, smart cards can also be used to identify individuals. The usual procedure involves exchanging secret information that is known only to the user and the card. This is usually a personal identification number (PIN).

Everyone is familiar with PIN verification from personal experience. The PIN is entered at a terminal, and shortly thereafter the display shows whether the PIN was correct or, if not, how many attempts are still allowed. In this procedure, the smart card receives the PIN from the terminal in a VERIFY command. The PIN is usually a four-digit number, which the smart card compares with a value stored in its EEPROM. If the entered PIN matches the stored PIN, the card's internal state changes, the terminal receives a response confirming a positive result, and the retry counter is reset to its original value of 0. If the entered PIN does not match the stored PIN, the retry counter is incremented. If it reaches its predefined maximum value, the card is blocked for further PIN verification.

Many smart card operating systems allow several PINs to be used. In such cases, it is mandatory to send the identification number of the relevant PIN in all associated commands, so that it can be correctly addressed. As a rule, however, card issuers attach great importance to having only one PIN per card, even when it is technically possible to have more than one. This is essential for customer acceptance and user friendliness.

The abbreviation ‘CHV’ is often used in place of ‘PIN’ in the telecommunications industry. CHV stands for ‘cardholder verification’ and means exactly the same thing as PIN. Since some of the commands described below originated in the telecommunications industry, their names use the abbreviation CHV instead of PIN.

Table 7.20 The functionality of VERIFY CHV according to GSM 11.11

VERIFY CHV	
Command	• PIN • number of the PIN
Response	• return code

The ISO/IEC 7816-4 standard describes a PIN verification command that is largely the same as the GSM 11.11 command. Its name is VERIFY, and it can be used not only for PIN comparison, but also for verifying biometric features. Compared with PIN verification according to the GSM specification, there is only one significant difference in the coding. ISO/IEC makes a distinction in the command between a global PIN and an application-specific PIN. The command can thus be used to specify whether to verify a PIN that applies to the entire smart card or one that is only applicable to the current DF.

Table 7.21 The functionality of VERIFY according to ISO/IEC 7816-4

VERIFY	
Command	• PIN or biometric feature (= secret) • number of the PIN or biometric feature • *switch:* global or application-specific secret
Response	• return code

In some applications, the cardholder is expected to choose his or her own PIN the first time the PIN is entered. The null-PIN method can be used for this purpose. With this method, the PIN is set to zero when the card is personalized. This PIN code has a special meaning for some smart card operating systems, which reject all VERIFY commands containing a null PIN and demand that the PIN first be changed using the CHANGE REFERENCE DATA command. In this way, the user is forced to change his or her PIN, since it is not possible to alter the security state of the card if a null PIN is entered. This method is not standardized, but it can sometimes be used to advantage if it is supported by the operating system.

PINs have steadily proliferated since their introduction as identification numbers for cardholders. Currently, the average card user is expected to keep track of perhaps 10 to 20 different PINs for various cards and other authorizations. The fact that this expectation is unrealistic is shown by the large number of people who jot down the PIN on the card itself. If smart cards are used, the user can be allowed to choose a PIN at will, and thus to use the same PIN for all of his or her cards. Although this may cause security problems, since anyone who illicitly acquires one PIN thereby knows all of them, it is still better than writing the PIN on the card where everyone can read it.

The CHANGE CHV command allows the PIN to be altered. The ISO/IEC 7816-8 equivalent to this command is CHANGE REFERENCE DATA, which has the same input and output parameters. If the PIN currently stored in the card is known, it can be replaced with a new one. If the current PIN is entered incorrectly, the operating system increments the retry counter to protect against possible attempts to use this mechanism to discover the PIN. As soon as the current PIN is correctly passed to the card, the card stores the new PIN it has received in the appropriate memory location and resets the retry counter.

Table 7.22 The functionality of CHANGE CHV according to GSM 11.11

CHANGE CHV	
Command	• old PIN • new PIN • PIN number
Response	• return code

If a retry counter has reached its maximum value, it can be reset using the UNBLOCK CHV command with a second PIN, which is called the personal unblocking key (PUK). The PUK is usually longer than the standard 4-digit PIN (8 digits, for example). The user need not memorize the PUK, since it is only needed if the PIN has been forgotten. It is sufficient if the user has a record of the PUK somewhere at home. However, just resetting the retry counter would not help the user very much, since he or she would still not know the correct PIN. Consequently, the command UNBLOCK CHV must also provide the card with a new PIN.

It should not be possible to use this command to alter the PIN of a hybrid card, which has a magnetic stripe as well as a chip. Otherwise the PIN recorded on the magnetic stripe would not match the PIN in the chip, which would cause severe problems. With such cards, the retry counter is simply reset and the customer is sent a letter with the original PIN.

Table 7.23 The functionality of UNBLOCK CHV according to GSM 11.11

UNBLOCK CHV	
Command	• PUK • new PIN
Response	• return code

The ISO/IEC 7816-8 standard has its own command for resetting the retry counter when it has reached its maximum value. This command is called RESET RETRY COUNTER. A certain security state must be achieved before this command can be executed. As a rule, this state is achieved by means of a successful authentication. Despite its name, this command can also be used to replace the current PIN with a new PIN.

Table 7.24 The functionality of RESET RETRY COUNTER according to ISO/IEC 7816-8

RESET RETRY COUNTER	
Command	• number of the PIN • *option:* [PUK || new PIN] • *if option selected:* number of the PUK • *switch:* global PIN or application-specific PIN
Response	• return code

GSM includes two other commands that can be used to control PIN querying. They are DISABLE CHV and ENABLE CHV, which switch PIN verification off and on. If PIN verification is disabled, all file access restrictions that require prior PIN verification are disabled. Both of these commands are very popular in the mobile telecommunications area, since they eliminate the need to re-enter the PIN every time the mobile telephone is switched on. From a security perspective, these commands are questionable, since they disable protection against unauthorized use that is provided by the PIN. Of course, the user could also use CHANGE CHV to choose a trivial PIN, such as "0000", which offers just as little protection.

The ISO/IEC 7816-8 commands for these functions are called ENABLE VERIFICATION REQUIREMENT and DISABLE VERIFICATION REQUIREMENT. Depending on the application, a certain security state must be achieved before these commands can be executed.

Table 7.25 The functionality of DISABLE VERIFICATION REQUIREMENT according to ISO/IEC 7816-8

DISABLE VERIFICATION REQUIREMENT	
Command	• reference data (e.g., PIN) • number of the reference data
Response	• return code

For obvious reasons, the PIN verification procedures described above are subject to attack, since a large financial gain could be realized using a lost or stolen card and the right PIN. All commands associated with PIN and PUK comparison must be protected against analysis of the card's electrical or timing behavior. For instance, current consumption during the execution of VERIFY PIN must be constant, regardless of whether the entered PIN is correct. It is equally important for the time required to execute PIN commands to be independent of whether the PIN is correct. Varying execution times could have fatal consequences for the card's security,

Table 7.26 The functionality of ENABLE VERIFICATION REQUIREMENT according to ISO/IEC 7816-8

ENABLE VERIFICATION REQUIREMENT	
Command	• reference data (e.g., PIN) • number of the reference data
Response	• return code

and thus ultimately the security of the entire system. Such variations could be used to determine the value of the correct PIN in a very simple manner, causing all of the system's PIN codes to be rendered worthless as a means of user identification.

7.6 AUTHENTICATION COMMANDS

In addition to commands for identifying the cardholder, there is a further set of commands for authenticating the terminal and the card. Since each of these communications partners is equipped with a complete computer, the procedures can be made much more complex, and thus more secure, than those used for PIN verification.

In PIN verification, the card receives a secret code in plaintext (the PIN) via the interface, and it only has to compare this with the PIN held in memory. Tapping the transmission line would thus have fatal consequences. Modern authentication procedures are designed to make such attacks impossible.

In principle, authentication involves verifying a secret known to both of the communicating parties without requiring it to be sent across the interface. The procedures are constructed such that tapping the data transmission would not compromise the security of the authentication.[2]

Depending on the operating system, various commands are available for authenticating the card or the terminal, or both at the same time. For the sake of clarity, here and in the rest of this chapter we refer to authentication between the card and the terminal. However, in terms of information technology what actually happens is that the 'outside world' authenticates itself with respect to an application in the card. This does not involve verifying that the card as a whole is genuine, but only that the embedded microcontroller shares a secret with the external world. This should be taken into account in certain applications.

In many operating systems, the keys used for authentication are protected by a retry counter. If a terminal unsuccessfully attempts authentication too many times, the card blocks the associated key for further authentication tests. This is a perfectly acceptable procedure with regard to system security, but it does have one drawback. Resetting the retry counter for the authentication key to its initial value often involves very complex, logistically cumbersome and costly administrative procedures. Consequently, some systems do not have retry counters for authentication keys.

For security reasons, only card-specific keys should be used for authentication. These keys can be generated from a unique feature of the card. A serial number or chip production number

[2] See also Section 4.11, 'Authentication'

is very suitable for this purpose. Such non-confidential (and thus public) numbers can be read from the card using a suitable command. There is presently no standard in this regard; here we use the GET CHIP NUMBER command. The name varies from one operating system to the next, as do the data that are exchanged. Here we are only interested in the functionality. The command GET CHIP NUMBER obtains a unique serial number from the card, which in light of the DES algorithm should preferably be eight bytes long. This number is used to uniquely identify the chip and to compute card-specific keys.

Table 7.27 The functionality of GET CHIP NUMBER

GET CHIP NUMBER	
Command	• —
Response	• chip number • return code

There is one more command that is needed for authentication. This is GET CHALLENGE, which is specified in ISO/IEC 7816-4. It is used to request a random number from the card. This number is subsequently used during authentication. When DES authentication is used, the length of the number is typically eight bytes, but it may be different for other cryptographic algorithms.

Table 7.28 The functionality of GET CHALLENGE according to ISO/IEC 7816-4

GET CHALLENGE	
Command	• —
Response	• random number • return code

In order to make the following examples relatively easy to understand and avoid unnecessary complexity, we have omitted describing the derivation of the card-specific key. However, deriving card-specific keys is absolutely essential for reasons of security.

The INTERNAL AUTHENTICATE command allows the terminal to authenticate a card, or in the case of a multiapplication card, to authenticate an application. This command can be used to verify that a card is genuine. The card receives a random number, which it encrypts using a cryptographic algorithm such as DES using a key that is known only to it and the terminal. The result of this encryption is returned to the terminal in the response. The terminal performs the same encryption as the card and compares its result with the result in the response received from the card. If the two results match, it follows that the card also knows the secret authentication key and must be genuine. The card has thus been authenticated.

This command implements the classic challenge–response procedure for authenticating a communications partner. The exact data content of the challenge is not specified in detail in the ISO/IEC 7816-4 standard. The only thing that is standardized is that the value sent to the card

Table 7.29 The functionality of INTERNAL AUTHENTICATE according to IEO/IEC 7816-4

INTERNAL AUTHENTICATE	
Command	• random number • number of the algorithm to be used • number of the key to be used
Response	• **enc** (key; random number) • return code

Smart card		**Terminal**
	←	INTERNAL AUTHENTICATE
X := **enc** (key; random number)		*Command* [random number, key number]
		X' := **enc** (key; random number)
Response [X || return code]	→	IF (return code = OK) AND (X = X')
		THEN command successful,
		smart card authenticated
		ELSE authentication failed

Figure 7.13 Sample command sequence using INTERNAL AUTHENTICATE

must be random and session-specific. Consequently, when the INTERNAL AUTHENTICATE command and the other authentication commands described below are used in an application, a detailed specification is always necessary to ensure that they can be used interoperably.

The terminal uses the EXTERNAL AUTHENTICATE command to show the card that it is connected to a genuine terminal. This command must be initiated by the terminal, since the communications process must always operate within the command–response framework. However, the card can force terminal authentication by blocking access to certain files until the terminal has been successfully authenticated.

This authentication is performed as follows. First, the terminal sends a GET CHALLENGE command to request a random number from the card, which it then encrypts using a secret key. Using the next command, EXTERNAL AUTHENTICATE, the terminal returns the encrypted random number to the card. The card performs the same encryption using the secret key, which it also knows, and then compares the result with the value received from the terminal. If they match, the terminal must also possess the secret authentication key and is thus genuine. After successful terminal authentication, the operating system changes the state of its state machine. This allows the terminal to access certain files for reading or writing. For the user, this is the visible result of an external authentication.

If the INTERNAL AUTHENTICATE and EXTERNAL AUTHENTICATE commands are executed one after the other, the communicating parties are mutually authenticated. Each one thus knows that the other one is genuine. However, this requires a total of three complete command sequences. In order to simplify this complicated and time-consuming procedure, the three commands and their data have been merged into a single command, which is called MUTUAL AUTHENTICATE.

Table 7.30 The functionality of EXTERNAL AUTHENTICATE according to ISO/IEC 7816-4

EXTERNAL AUTHENTICATE	
Command	• **enc** (key; random number) • number of the algorithm to be used • number of the key to be used
Response	• return code

Smart card		**Terminal**
		GET CHALLENGE
	←	*Command* [ ]
Response [random number || return code]	→	IF (return code = OK)
		THEN command successful
		ELSE abort
		X := **enc** (key; random number)
		EXTERNAL AUTHENTICATE
X' := **enc** (key; random number)	←	*Command* [X, key number]
IF (X = X')		
THEN terminal authenticated		
ELSE authentication failed		
Response [return code]	→	IF (return code = OK)
		THEN command successful, terminal authenticated
		ELSE authentication failed

Figure 7.14 Sample command sequence using EXTERNAL AUTHENTICATE

This command is defined in the ISO/IEC 7816-8 standard. It can be used to perform mutual authentication of two parties in accordance with the ISO/IEC 9798–2/3 standard. Using a single authentication command also increases the security of the overall procedure, since it prevents the fraudulent insertion of commands between two one-way authentications. A further improvement in the security of the procedure results from the fact that it is impossible to obtain plaintext–ciphertext pairs by tapping the communications between the terminal and the card, which would otherwise provide an ideal basis for an attack.

Mutual authentication works as follows. First, the terminal uses the GET CHALLENGE command to request the chip number of the card. It can then compute the card-specific key. The terminal then uses the ASK RANDOM command to obtain a random number from the card, and it generates its own random number.

After the terminal receives the random number from the card, it combines the two random numbers and the chip number to form a single data block, which it encrypts using the secret authentication key and a cryptographic algorithm in CBC mode. It sends the resulting ciphertext block to the card, which decrypts it and compares the chip number and the

Table 7.31 The functionality of MUTUAL AUTHENTICATE according to ISO/IEC 7816-8

MUTUAL AUTHENTICATE	
Command	• **enc** (key; terminal random number, smart card random number, chip number) • number of the algorithm to be used • number of the key to be used
Response	• **enc** (key; smart card random number, terminal random number) • return code

random number with the numbers it previously sent to the terminal. If they match, the terminal has been authenticated. Now the card swaps the two random numbers, deletes the chip number and again encrypts the resulting block using the secret key. After the terminal has received and decrypted this block, it can determine whether the card possesses the secret authentication key by comparing the known random numbers. If they match, the card is also authenticated.

Smart card		**Terminal**
		GET CHIP NUMBER
	←	*Command* [ ]
Response [chip number || return code]	→	IF (return code = OK)
		THEN command successful
		key derivation
		(compute the card-specific key)
		GET CHALLENGE
generate RND_CK	←	*Command* [ ]
Response [RND_CK || return code]	→	IF (return code = OK)
		THEN command successful
		MUTUAL AUTHENTICATE
		generate RND_T
		X1 := **enc** (key; RND_T || RND_CK ||
		chip number)
X1' := **dec** (key; X1)	←	*Command* [X1 || key number]
IF (RND_CK and chip number are the		
same as those sent)		
THEN terminal authenticated		
ELSE abort		
X2 := **enc** (key; RND_CK || RND_T)		
Response [X2]	→	X2' := **dec** (key; X2)
		IF (RND_T = same as the one sent)
		THEN smart card authenticated

Figure 7.15 Sample command sequence using MUTUAL AUTHENTICATE

7.7 COMMANDS FOR CRYPTOGRAPHIC ALGORITHMS

Commands for cryptographic algorithms are very important for many applications. For example, they make it relatively easy to use smart cards as encryption and decryption devices or to verify digital signatures. Many smart card operating systems have their own command sets for executing cryptographic algorithms. Smart card commands such as ENCRYPT, DECRYPT, SIGN DATA and VERIFY SIGNATURE have arisen because there was no standard for this sort of functionality. However, commands specially designed for processing cryptographic algorithms have now been defined in the ISO/IEC 7816-8 standard.

In the ISO/IEC 7816-8 standard, the functions related to cryptography are split between two commands. The MANAGE SECURITY ENVIRONMENT (MSE) command allows various general constraints to be set before the cryptographic algorithm is executed. This command passes a 'template' to the card, and this template contains the relevant parameters. They remain valid until they are replaced using a new MANAGE SECURITY ENVIRONMENT command. The templates themselves consist of TLV-coded data objects, which provide a high degree of flexibility (and unfortunately, complexity) for transferring parameters.

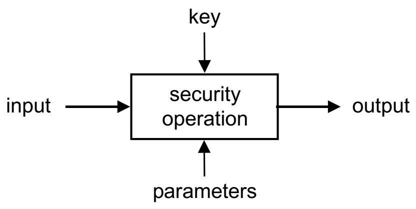


Figure 7.16 The basic principle of the ISO/IEC 7816-8 MANAGE SECURITY ENVIRONMENT and PERFORM SECURITY OPERATION commands for using cryptographic functions

Table 7.32 The functionality of MANAGE SECURITY ENVIRONMENT according to ISO/IEC 7816-9

MANAGE SECURITY ENVIRONMENT	
Command	• template with parameters for a cryptographic checksum *or* template with parameters for a digital signature *or* template with parameters for a hash algorithm *or* template with parameters for authentication *or* template with parameters for encryption and decryption
Response	• return code

After all the options for the cryptographic function have been configured using the MANAGE SECURITY ENVIRONMENT command, the PERFORM SECURITY OPERATION

(PSO) command can be invoked. A wide variety of security operations can be performed using this command, provided that they are supported by the smart card operating system. However, the number of options with this command is so large that this support is not always mandatory. Although PERFORM SECURITY OPERATION is coded using only one instruction byte, it has eight fundamentally different functions that are distinguished by the P1 parameter byte. This was done because the number of codes remaining for coding commands has become somewhat scant in the last while.

Since the PERFORM SECURITY OPERATION command can be used in so many different ways, we have divided the following description of its functions along the lines of its various options, without describing any of them in detail.

PERFORM SECURITY OPERATION with the COMPUTE CRYPTOGRAPHIC CHECKSUM option is used to compute a cryptographic checksum (CCS), which is commonly referred to as a MAC. The padding, as well as the key to be used, can be either implicitly provided by the operating system or explicitly supplied using the MANAGE SECURITY ENVIRONMENT command. The counterpart to this command option is the VERIFY CRYPTOGRAPHIC CHECKSUM option, which computes the cryptographic checksum of the transferred data and compares it with a reference value, which is also transferred with the command. The result of this operation is a match/no-match decision, which is returned to the terminal.

Table 7.33 The functionality of PERFORM SECURITY OPERATION with the COMPUTE CRYPTOGRAPHIC CHECKSUM option, according to ISO/IEC 7816-8

PERFORM SECURITY OPERATION *Option:* COMPUTE CRYPTOGRAPHIC CHECKSUM	
Command	• data to be encrypted
Response	• CCS • return code

Table 7.34 The functionality of PERFORM SECURITY OPERATION with the VERIFY CRYPTOGRAPHIC CHECKSUM option, according to ISO/IEC 7816-8

PERFORM SECURITY OPERATION *Option:* VERIFY CRYPTOGRAPHIC CHECKSUM	
Command	• data to be encrypted • CCS
Response	• return code

The ENCIPHER and DECIPHER options are provided for the pure encryption and decryption of data. The ENCIPHER option is used to encrypt the data transferred with the command. The encryption algorithm to be used can be selected, according to the options available in the operating system, by first sending the MANAGE SECURITY ENVIRONMENT command. Similarly, the mode of the encryption algorithm must also be set in advance by parameters transferred before the command is issued. With a block encryption algorithm, it is also possible to select the ECB or CBC mode. Since the length of the data block sent to the card is not always an exact multiple of the block size of the cryptographic algorithm, the padding method must be defined via an additional parameter. The address of the key stored in the smart card that is to be used by the algorithm to encrypt the data is also important.

Table 7.35 The functionality of PERFORM SECURITY OPERATION with the ENCIPHER option, according to ISO/IEC 7816-8

PERFORM SECURITY OPERATION *Option:* ENCIPHER	
Command	• data to be encrypted
Response	• encrypted data • return code

The reverse function of ENCIPHER is DECIPHER. With this function, the transferred data can be decrypted in the same mode as that used for ENCIPHER. Naturally, the smart card must know the appropriate key, algorithm mode and padding mode. This information must be passed to the card's operating system using the MANAGE SECURITY ENVIRONMENT command.

Table 7.36 The functionality of PERFORM SECURITY OPERATION with the DECIPHER option, according to ISO/IEC 7816-8

PERFORM SECURITY OPERATION *Option:* DECIPHER	
Command	• encrypted data
Response	• decrypted data • return code

With the introduction of public-key algorithms into smart card applications, there was a need for suitable commands for using the newly available functions. Smart cards are particularly suitable for digital signature applications, since the private key for the signature algorithm can be securely stored in memory, where it cannot be read. The ISO/IEC 7816-8 standard describes four command options that can be used for digital signatures.[3]

The HASH option of the PERFORM SECURITY OPERATION command can be used to compute a hash value. The command may transfer either the data to be hashed or a hash value

[3] See also Section 14.4, 'Digital Signatures'

already computed outside the smart card along with the data needed for the final step of the computation. In the latter case, the hash computation for the final block is performed in the card. The advantage of the latter method is that the hash value can be generated significantly faster outside of the card, but the final step still occurs inside the card. From a purely cryptological perspective, this provides only a small amount of extra security, but it does somewhat limit the possibilities for manipulating the hash value. For this reason, it is widely used in practice.

Since the amount of data to be hashed is usually larger than maximum allowed length of a command data field, the HASH option employs 'layer-7' chaining, which means that the data blocks are logically chained at the application level. The final data block to be hashed includes a marker that informs the command that it is the final block for the hash process.

This command option also has its own options. The computed hash value can either be transferred immediately to the terminal in the response to the command or stored in the card for use with a subsequent command. The padding and the key to be used are defined as necessary using a prior MANAGE SECURITY ENVIRONMENT command, in the same way as for previously described commands.

Table 7.37 The functionality of PERFORM SECURITY OPERATION with the HASH option, according to ISO/IEC 7816-8

PERFORM SECURITY OPERATION *Option:* HASH	
Command	• data to be hashed • hash value (if only part of the hash computation is to be performed in the card) • *switch:* perform only part of the hash computation in the card • *switch:* return the computed hash value in the response • *switch:* final command with data to be hashed
Response	• hash value (if selected by the switch) • return code

The COMPUTE DIGITAL SIGNATURE option can be used for signing data. The data string to be signed, which has usually been compressed to a hash value, must be passed to the smart card if it is not already present in the card as the result of a previous PERFORM SECURITY OPERATION command with the HASH option. The COMPUTE DIGITAL SIGNATURE option also allows the data to be signed to be transferred directly to the card. The data can then be hashed in the card before the signature is generated. For large amounts of data, 'layer-7' chaining can be used as with the HASH option.

If the length of the hash value does not correspond to the input data length of the public-key algorithm, padding must be added. The options for this are set by parameters of the MANAGE SECURITY ENVIRONMENT command, which is also used to identify the key to be used.

The verification counterpart to the COMPUTE DIGITAL SIGNATURE option is provided by the VERIFY DIGITAL SIGNATURE option. In principle, any sufficiently fast digital computer could be used to verify a digital signature, since the necessary key is public. However, in many cases the validity of the public key must first be verified using an additional digital signature. This is certainly a security consideration, so this should not be performed using

Table 7.38 The functionality of PERFORM SECURITY OPERATION with the COMPUTE DIGITAL SIGNATURE option, according to ISO/IEC 7816-8

PERFORM SECURITY OPERATION *Option:* COMPUTE DIGITAL SIGNATURE	
Command	• data to be signed *or* hash value of the data to be signed • *switch*: final command with data to be signed
Response	• digital signature • return code

an insecure computer. In order to verify a digital signature, the associated public key must either be implicitly known to the smart card or be explicitly made known to the card using the command option VERIFY CERTIFICATE, which is described immediately below. The data to be verified may be passed by VERIFY DIGITAL SIGNATURE either directly or in the form of the associated hash value. All other parameters are the same as for the COMPUTE DIGITAL SIGNATURE option.

Table 7.39 The functionality of PERFORM SECURITY OPERATION with the VERIFY DIGITAL SIGNATURE option, according to ISO/IEC 7816-8

PERFORM SECURITY OPERATION *Option:* VERIFY DIGITAL SIGNATURE	
Command	• data to be verified *or* hash value of the data to be verified • digital signature • *switch*: final command with data to be verified
Response	• return code

In open systems, public keys for verifying digital signatures are usually signed using the private key of the certification authority. The authenticity of a public key must be verified before it is used, since this is the only way to be sure that the key is not a forgery. This verification must take place in a secure environment, such as a smart card, since it would otherwise be subject to manipulation. The VERIFY CERTIFICATE command option is provided specifically for verifying signed public keys, which are also called 'certificates'. Once a public key has been identified as authentic, it can either be stored permanently in the smart card or be used with an immediately following VERIFY DIGITAL SIGNATURE command.

Figure 7.17 illustrates how the commands described above can be used to generate a digital signature and then verify it.

If a smart card operating system supports generating key pairs for asymmetric cryptographic algorithms, this process can be instigated by the ISO/IEC 7816-8 GENERATE PUBLIC KEY PAIR

Table 7.40 The functionality of PERFORM SECURITY OPERATION with the VERIFY CERTIFICATE option, according to ISO/IEC 7816-8

PERFORM SECURITY OPERATION *Option:* VERIFY CERTIFICATE	
Command	• certificate • *switch:* store the public key
Response	• return code

Smart card		**Terminal**
		COMPUTE DIGITAL SIGNATURE
S := **sign** (private key; hash value of the data)	←	*Command* [hash value of the data]

Response [S || return code]	→	IF (return code = OK)
		THEN signature successful
		ELSE abort

		VERIFY DIGITAL SIGNATURE

H' := **encrypt** (public key; S)
H := hash value of the data
IF (H = H')
THEN (return code = OK)
ELSE (return code = not OK)
	←	*Command* [hash value of the data]

Response [return code]	←	IF (return code = OK)
		THEN signature is genuine
		ELSE signature is false

Figure 7.17 Sample procedure for using PERFORM SECURITY OPERATION with the COMPUTE DIGITAL SIGNATURE and VERIFY DIGITAL SIGNATURE options. The basic parameters (key and algorithm to be used) are specified in advance, either implicitly or using a MANAGE SECURITY ENVIRONMENT command

command. All parameters needed for key generation must be set in advance using the MANAGE SECURITY ENVIRONMENT command.

Table 7.41 The functionality of GENERATE PUBLIC KEY PAIR according to ISO/IEC 7816-8

GENERATE PUBLIC KEY PAIR	
Command	• —
Response	• return code

7.8 FILE MANAGEMENT COMMANDS

Most modern smart card operating systems allow various management operations to be performed on files within limits set by specific security conditions, including extending, creating, deleting and blocking files. Nevertheless, most or even all management functions are frequently omitted in single-application cards, since these functions generally require a large amount of program code, which would increase the memory capacity and thus the cost of the card. With multiapplication cards, support for certain management functions is essential in order to avoid having to load all the applications into the card when it is personalized.

With regard to system security, management functions should only be allowed to be executed following mutual authentication, since they are ideal starting points for an attack. For example, suppose an unauthorized person were to delete a file holding confidential data and then create a new file with the same name but without any read access restrictions, so that it could be still be read with its original name, therefore confidential data could be writen to the manipulated file. This type of attack is by no means new; it has been around for many years in a somewhat different form. However, it is used successfully over and over again in file management systems.

Another possible point of attack is provided if management functions are used in publicly accessible terminals, which in principle are insecure. In such situations, data transfers must always be protected using secure messaging functions. Only then will an application provider be able to securely load files and applications into cards already in use, for instance via public card phones. This is a very attractive option for logistics reasons.

Particularly in the case of multiapplication smart cards, which can be used by several application providers, it is necessary to partition the memory and assign authorization keys for file creation before the individual applications are generated. This prevents any individual application provider from allocating the entire available memory to his own application, leaving none for other applications. One way to prevent such behavior is to use a procedure that preallocates memory space to each application and at the same time stores card- and application-specific keys for creating files in the card. This can be done using the REGISTER command, which is not standardized. New files can be created at a later date if the application-specific key is known. This method creates a strict separation between allocating memory space and loading new files into the card. The issuer of a multiapplication card can thus sell memory space to several application providers without having to worry about memory piracy.

Table 7.42 The functionality of REGISTER (not standardized)

REGISTER	
Command	• DF name of the new DF • maximum memory space for the new application (i.e., the DF) • key for creating new files (i.e., for CREATE FILE)
Response	• return code

The CREATE FILE command allows DFs or EFs to be created in smart cards after they have been completed. Before this command can be executed, a particular logic state must have been achieved, for example via successful mutual authentication. Depending on the environment in which the CREATE FILE command is executed, data transfers may have to be protected by secure messaging. The CREATE FILE command is defined in the ISO/IEC 7816-9 standard.

Table 7.43 The functionality of CREATE FILE according to ISO/IEC 7816-9

CREATE FILE	
Command	• file type of the new file • IF (file type = DF) THEN [DF name of the new file] • IF (file type = EF) THEN [• FID of the new file • access conditions • structure of the new file] • IF (file structure = transparent) THEN [file size] • IF (file structure = linear fixed) OR (file structure = cyclic) THEN [• number of records • record length] • IF (file structure = linear variable) THEN [• number of records • length of each individual record]
Response	• return code

After a file has been created with all of its access conditions, attributes and other properties, it can be selected using SELECT FILE and then accessed. The operating system must make it impossible to create an incomplete file by interrupting the file creation process, since such a file could provide a basis for an attack on the card. Furthermore, it must not be possible to read out the residual contents of memory areas that are only partially overwritten when a new file is created.

The file header contains the complete access conditions for the file. For example, it can store data relating to the states in which a READ or UPDATE command is allowed to access the content of the file. Particularly when a card is being personalized, as well as for extensive management procedures within the file tree, it is a major advantage to be able to access files directly without object protection. For this reason, a previous version of the ISO/IEC 7816-9 included a command called MANAGE ATTRIBUTES that could be used to change the access conditions of a previously selected file. This command could only be used following mutual authentication and within a secure environment. However, it was made redundant by the introduction of security environments and rule-based access conditions as defined in ISO/IEC 7816-9,[4] and it is no longer included in the standard. Nevertheless, it is supported by some smart card operating systems, due to the simplicity of its implementation.

If the REGISTER command is available, the card issuer uses it to specify the maximum amount of memory that can later be allocated to an application. In addition, a temporary DF name, which may contain an application identifier (AID), is specified, and a key for the CREATE FILE command is stored in the card. Files can subsequently be created if this information is known.

[4] See also Section 5.9, 'Resource Accesses in Accordance with ISO/IEC 7816-9'

Table 7.44 The functionality of the now obsolete MANAGE ATTRIBUTES command, according to a draft version of ISO/IEC 7816-9

MANAGE ATTRIBUTES	
Command	• new access conditions
Response	• return code

The user receives a smart card that has been prepared in this manner. If the user so desires, he or she can have an application provider load an additional application into the card, such as a card phone application. However, the application provider must know the secret download key before he can load a new application into the card, and the card issuer will naturally not provide this key unless there is a contractual relationship between the two parties.

After successful mutual authentication between the smart card and the terminal, the application provider can create his files in the DF allocated to him. This can take place either on the provider's premises or via a public card phone. Next, the provider fills the EFs with the necessary data and keys and sets the access attributes of the files. After this, the application is ready to use, and the user can enjoy his or her newly acquired functions.

The procedure for loading a new application in the file tree of a smart card is illustrated in Figure 7.18.

The DEACTIVATE FILE command (defined in ISO/IEC 7816-9) and the INVALIDATE command (defined in GSM 11.11) allow the terminal to reversibly block a previously selected file. When a file is blocked, read and write accesses to the file are prohibited, and only selection is allowed.

The EMV specification provides a similar command for reversibly blocking applications, called APPLICATION BLOCK. This command does not block the EFs within an application, but only the commands for file selection, authentication and financial transactions within the DF of the application. The functionality of the APPLICATION BLOCK command is otherwise similar to that of the DEACTIVATE FILE and INVALIDATE commands.

Table 7.45 The functionality of DEACTIVATE FILE according to ISO/IEC 7816-9

DEACTIVATE FILE	
Command	• — *or* FID *or* short FID *or* DF name
Response	• return code

Smart card		**Terminal**
	mutual authentication between the smart card and the terminal	
	←	CREATE FILE *Command* [. . .]

command processing
Response [return code]
	→	IF (return code = OK)
		THEN command successfully executed
		ELSE command failed

	several iterations:	

	←	UPDATE BINARY or
		UPDATE RECORD
		Command [. . .]

command processing
Response [return code]
	→	IF (return code = OK)
		THEN command successfully executed
		ELSE command failed

	several iterations:	
	←	MANAGE ATTRIBUTES *Command* [. . .]

command processing
Response [return code]
	→	IF (return code = OK)
		THEN command successfully executed
		ELSE command failed

Figure 7.18 Sample procedure for loading a new application

Table 7.46 The functionality of INVALIDATE according to GSM 11.11

INVALIDATE	
Command	• FID
Response	• return code

The inverse functions for INVALIDATE and DEACTIVATE FILE are provided by the REHABILITATE and REACTIVATE FILE commands, which can be used to unblock a blocked file. The file must be selected before it can be unblocked. It goes without saying that execution of all these commands can only be permitted in a certain security state, since otherwise anyone would be able to block or unblock files at will.

Table 7.47 The functionality of REHABILITATE according to GSM 11.11

REHABILITATE	
Command	• —
Response	• return code

Table 7.48 The functionality of REACTIVATE FILE according to ISO/IEC 7816-9

REACTIVATE FILE	
Command	• — *or* FID *or* short FID *or* DF name
Response	• return code

Smart card		**Terminal**
mutual authentication between the smart card and the terminal		
	←	DEACTIVATE FILE *Command* [FID]

command processing
Response [return code]
	→	IF (return code = OK)
		THEN file is blocked
		ELSE file not found or
		file blocking failed

	←	REACTIVATE FILE *Command* [FID]

command processing
Response [return code]
	→	IF (return code = OK)
		THEN command successfully executed
		ELSE file not found or
		file unblocking failed

Figure 7.19 Sample command sequence for DEACTIVATE FILE and REACTIVATE FILE

The LOCK command is the non-reversible version of INVALIDATE. A file that has been blocked with LOCK cannot be unblocked. Its state is completely irreversible. Another use for this command is to permanently block an application when it reaches its expiry date. A file that has been blocked with the LOCK command can only be selected; all other types of access will be denied by the operating system.

Table 7.49 The functionality of LOCK according to EN 726-3

LOCK	
Command	• —
Response	• return code

The main drawback of permanent, irreversible blocking is that it also blocks any further use of valuable memory space in the card. A much more elegant solution is to erase the memory regions occupied by files that are no longer needed, so they can be used by other applications or new applications. When doing so, it is important to not only delete the file from the file tree, but also to physically erase the entire memory area that was used by the file(s). Only in this way can one be sure that all of the file contents, which may certainly be confidential and worth protecting, have been overwritten and are thus no longer accessible to anyone. If the memory that comes free when a file is deleted is to be made available for use by other files, deleting a file becomes a complicated and costly process. Consequently, it is not fully implemented in all operating systems.

In principle, the DELETE FILE command can be used to block and unblock files in exactly the same way as the previously described commands. A file that is implicitly selected with this command can be completely removed from the file memory of the smart card. Whether the memory space that is released as a result can be used for other files depends on the individual operating system. As a rule, free-memory management is not available, so the memory space is lost forever after this command has been executed.

Table 7.50 The functionality of DELETE FILE according to ISO/IEC 7816-9

DELETE FILE	
Command	• FID *or* DF name
Response	• return code

The command TERMINATE DF is provided in the ISO/IEC 7816-9 standard for irreversibly blocking a DF. A blocked DF can still be selected, but the functions (such as program code) and files it contains are no longer accessible. This command can be used to 'switch off' an application while allowing the previous presence of the application to still be recognized.

Table 7.51 The functionality of TERMINATE DF according to ISO/IEC 7816-9

TERMINATE DF	
Command	• FID *or* DF name
Response	• return code

The functionality of the ISO/IEC 7816-9 TERMINATE CARD USAGE command is similar to that of the TERMINATE DF command. However, this command blocks the entire card, and thus the execution of any subsequent commands. After this, the only thing the card can do is indicate its new status in the ATR. The EMV equivalent of this command is the CARD BLOCK command, which has similar functionality.

Table 7.52 The functionality of TERMINATE CARD USAGE according to ISO/IEC 7816-9

TERMINATE CARD USAGE	
Command	• —
Response	• return code

7.9 COMMANDS FOR MANAGING APPLETS

In the case of smart card operating systems that support downloadable program code, one of the requirements is that code must be loaded into the memory of the smart card in a secure manner and in compliance with conditions imposed by the issuer of the smart card. There are a number of highly different, generally company-specific concepts for achieving this. The only internationally used industrial standard for this purpose is the Open Platform (OP) specification from Global Platform [GP].

An application is loaded into the smart card by transferring its load file to the card using the LOAD command. The INSTALL command, which continues the process, establishes the loaded application by invoking various on-card functions of the card manager and the security domain. The DELETE command is used to delete uniquely identifiable objects in a smart card, such as load files, applications and keys. Open Platform is described in Section 5.11, ‘Open Platform’.

7.10 COMMANDS FOR COMPLETING THE OPERATING SYSTEM

When smart card microprocessors are manufactured, only the ROM is programmed. The EEPROM remains empty, apart from a chip number and a card-specific key. After the card body and the microcontroller have been combined to produce a smart card, the operating system code in the ROM must be supplemented by the portions that reside in the EEPROM. This is called ‘completing’ the smart card. Only after it has been completed does a smart card contain a full operating system, with all its functionality.

There is a relatively simple loader program in the ROM for writing these parts of the operating system to the EEPROM. It can be used to write data to the EEPROM following a key verification. The EEPROM memory is addressed linearly, either byte by byte or page by page, using direct physical addresses.

Once all necessary data have been entered into the EEPROM in this way, the operating system is switched over from pure ROM operation. From this point on, processes and routines also run in the EEPROM. This switchover can be performed by a command whose execution condition has been satisfied, following a prior checksum comparison for all of the completed EEPROM data. The checksum ensures that all the data have been correctly stored in the EEPROM. Completion does not employ particularly complex functions or authorization procedures, since it is necessary to rely on the ROM portion of the operating system. Even the smallest error here could make it impossible to complete the smart card. Such an error would be time-consuming and, all told, very expensive.

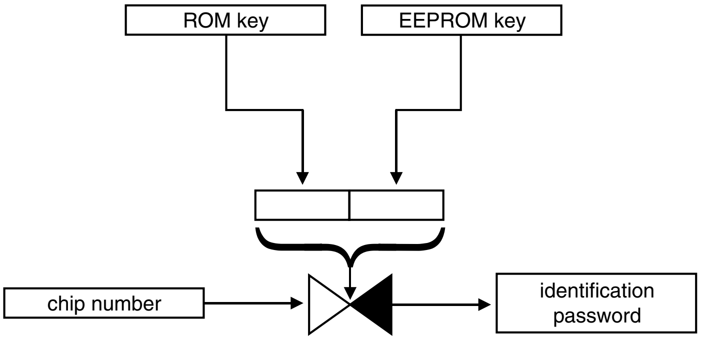


Figure 7.20 One of many possible ways to generate a chip-specific password for authorizing the completion of the operating system

In the rest of this section, three representative commands for completing the smart card operating system are described. The commands used for this purpose vary greatly depending on the operating system and chip manufacturer. Here we can only illustrate the necessary functions. However, practically all smart cards use this procedure or a similar procedure to complete the operating system.

The COMPARE KEY command tests a password sent to the card against a reference password stored in the ROM and EEPROM by the manufacturer during chip fabrication. This key (password) is card-specific and is quite long (approximately 32 bytes). If the comparison is successful, subsequent load commands are permitted. Otherwise, a retry counter is incremented. Once the retry counter reaches a predefined value (e.g., 3), it blocks any further access to the card. The card can then be sent for recycling, since the only thing it can do is to generate an ATR.

After the load password has been successfully verified using COMPARE KEY, all necessary data can be written to the EEPROM using the WRITE DATA command. At this point, the EEPROM can be addressed and written byte by byte. This means that in addition to the data

Table 7.53 The functionality of COMPARE KEY

COMPARE KEY	
Command	• key
Response	• return code

for the operating system, complete applications may also be loaded. Incidentally, this is the method normally used to load applications into smart cards having very little memory, since they do not have enough room for complex CREATE FILE commands and their associated state machines.

Table 7.54 The functionality of WRITE DATA

WRITE DATA	
Command	• data • memory address in EEPROM
Response	• return code

If the amount of ROM in the smart card is so small that there is not even enough room for EEPROM test commands, the basic functions of the test commands can be simulated using this command. This is done by repeatedly writing data to a specific memory location until the card reports a write error. If the number of write cycles is totaled, the number of possible write/erase cycles is known. This is the primary result expected from an EEPROM test command in the context of quality assurance.

Once all data have been written to the EEPROM using one or more successive WRITE DATA commands, the content of the EEPROM is tested to see that it is correct, and the completion procedure is then terminated. The command used for this is COMPLETION END. After successful execution of this command, a card reset is normally triggered to reinitialize the operating system and allow it to achieve a new state.

Table 7.55 The functionality of COMPLETION END

COMPLETION END	
Command	• checksum of the EEPROM contents
Response	• return code

For tests and experiments, it is often necessary to be able to delete an existing completion. This is usually not possible, but a command that provides this capability is available in a few isolated operating systems. This is the DELETE COMPLETION command, which can reverse the completion of the operating system. With this command, it is not necessary to use a new smart card for every new completion. This command is used exclusively for testing purposes.

Table 7.56 The functionality of DELETE COMPLETION

DELETE COMPLETION	
Command	• —
Response	• return code

The correct completion of a smart card operating system can be verified using the CHECK COMPLETION command. For this purpose, the smart card computes a checksum from the completion data and compares it with a reference value stored along with the completion data. If the two checksums match, the completion is correct.

Table 7.57 The functionality of CHECK COMPLETION

CHECK COMPLETION	
Command	• —
Response	• return code

Figure 7.21 further illustrates the sequence of commands for completing a smart card as just described. This procedure is governed by a state machine during the completion process to ensure that only the indicated commands can be executed and only exactly in the order shown. This state machine is described by Figure 7.22.

7.11 COMMANDS FOR HARDWARE TESTING

During initialization, the operating system of the smart card tests various hardware components, both implicitly and explicitly. The commands described here, however, go far beyond the self-test routines integrated directly into the operating system. As part of production quality assurance, it is necessary to separately verify certain critical parts of the microprocessor. These tests focus particularly on the EEPROM, since experience shows that most problems show up here. The operation of the processor is implicitly verified if the terminal can receive a correct ATR.

Since there are no standards for test commands, their functionality and coding depend on the producer of the operating system, and sometimes on the operating system itself.

The test commands may be permanently stored in ROM. However, it is also quite common for them to be loaded into the EEPROM via the completion commands and then executed in the EEPROM. Obviously, this can cause problems if the EEPROM is not fully functional. The advantage of using EEPROM is that it makes more space available in the ROM. In the interest of operating system security, using test commands must be restricted to the stage preceding completion. This means that all test commands are blocked in a card that has been initialized or even personalized. Possible exceptions are the RAM test and the ROM or EEPROM checksum tests, since these commands do not affect security.

Smart card		**Terminal**
		COMPARE KEY
command processing	←	*Command* [key for completion]
Response [return code]	→	IF (return code = OK)
		THEN command successfully executed
		ELSE command failed
	numerous iterations:	
		WRITE DATA
command processing	←	*Command* [data || address]
Response [return code]	→	IF (return code = OK)
		THEN command successfully executed
		ELSE command failed
	conclusion:	
		COMPLETION END
command processing	←	*Command* [EEPROM checksum]
Response [return code]	→	IF (return code = OK)
		THEN command successfully executed
		ELSE command failed

Figure 7.21 Example of a typical completion procedure

The following commands may be used for extensive hardware tests. A dysfunctional RAM would cause a complete operating system crash even before the ATR could be sent, but it is possible for a few RAM bits or bytes to be ‘dead’. This would only affect certain functions or routines.

The TEST RAM command checks the entire RAM and sends an appropriate response to the terminal. During the test, all available bytes must be written and read using a variety of test patterns. A typical test consists of writing '55' and 'AA' alternately, since these two hexadecimal values form a checkerboard pattern at the bit level. Another effective method is the wave test, in which the RAM is first written from the lowest to the highest address and then read, and then from the highest to the lowest address. The exact implementation of this test depends on the operating system. Sometimes this test is omitted.

Table 7.58 The functionality of TEST RAM

TEST RAM	
Command	• —
Response	• return code

CALCULATE EDC is a very simple test that computes an EDC checksum for the whole ROM or specified portions of the EEPROM and returns it to the terminal. This is one way to

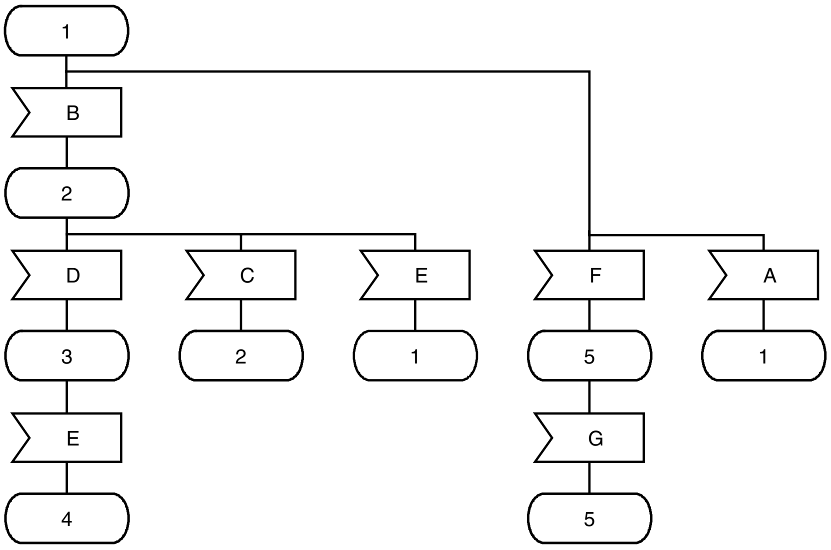


Figure 7.22 State machine for completing a smart card operating system

States:	Transitions:
1 initial state after smart card reset	A all commands except COMPARE KEY
2 smart card ready for loading data into EEPROM	B COMPARE KEY (successfully executed)
3 smart card completed	C WRITE DATA
4 initial state of smart card after completion	D COMPLETION END
5 smart card irreversibly blocked	E reset smart card
	F COMPARE KEY (3 times unsuccessful)
	G all commands and reset

determine whether the ROM mask has changed or any EEPROM cells have flipped. EEPROM testing relates only to static areas that cannot be intentionally changed during the lifetime of the card. The terminal compares the checksum received from the card with a reference value and decides whether the memory contents are still consistent.

Table 7.59 The functionality of CALCULATE EDC

CALCULATE EDC	
Command	• *switch:* ROM / EEPROM
Response	• checksum • return code

The TEST EEPROM command is used to perform an EEPROM endurance test by overwriting the contents of the EEPROM. The smart card receives two patterns, which it writes alternately to an area of memory. The size of this area and the number of write attempts can be specified in the command, within certain limits. Naturally, the operating system must check the memory content for errors after each write cycle.

Since the number of write cycles is supplied to the smart card as a command parameter, this command can be fully processed inside the smart card. This is much faster than sending a sequence of individual commands. If the card discovers a write error, it returns the number of already completed write cycles and the address of the error to the terminal. The EEPROM endurance test can be used not only for destructive testing of the EEPROM, but also for writing data to freely chosen regions of the EEPROM. In the latter case, the number of write cycles is set to one.

Table 7.60 The functionality of TEST EEPROM

TEST EEPROM	
Command	• pattern 1 • pattern 2 • number of write cycles • start address in EEPROM • end address in EEPROM
Response	• number of write cycles executed (in the event of an error) • error address (in the event of an error) • return code

The COMPARE EEPROM test is used to test whether a pattern written to the EEPROM is still present in the EEPROM. This command is primarily used in combination with the TEST EEPROM command to test EEPROM data retention at various temperatures. This is done by writing a pattern to several memory pages, placing the smart card in an environmental test chamber for a certain length of time, and then checking whether the pattern has been retained.

Table 7.61 The functionality of COMPARE EEPROM

COMPARE EEPROM	
Command	• comparison value • address
Response	• return code

In principle, the TEST EEPROM command can be used to erase the entire EEPROM if the start and end addresses are appropriately specified. However, many operating systems have a command that we can call DELETE EEPROM. It erases the entire EEPROM in one go by overwriting the entire contents of the EEPROM. This command is employed for only two purposes. The first is to restore the EEPROM to a predefined state with a minimum of effort after various tests have left their test data in the EEPROM. The second is to reduce the time required to initialize or complete the operating system. If the DELETE EEPROM command is executed prior to either of these activities, the EEPROM can subsequently be written faster, since it avoids the frequently encountered need to erase an EEPROM page before writing new data to it.

Table 7.62 The functionality of DELETE EEPROM

DELETE EEPROM	
Command	• —
Response	• return code

7.12 COMMANDS FOR DATA TRANSMISSION PROTOCOLS

In principle, data transmission protocols should be designed to be fully independent of the data and commands of the application layer. This is also the intention of the OSI layer model. Unfortunately, there is a discrepancy here between theoretical requirements and actual practice. There are two commands whose only purpose is to allow transport mechanisms to be utilized at the application level, namely GET RESPONSE and ENVELOPE. There is also another command, MANAGE CHANNEL, whose function is not used by the application layer alone.

In the T = 0 protocol, it is not possible to send a block of data to the smart card and receive a block of data from the smart cardwithin a single command–response cycle.[5] This protocol thus does not support case 4 commands, although they are frequently used. It is thus necessary to use a work-around for the T = 0 protocol. This operates in a simple manner. The case 4 command is first sent to the card, and if it is successful, a special return code is sent to the terminal to advise the terminal that the command has generated data that are waiting to be retrieved. The terminal then sends a GET RESPONSE command to the smart card and receives the data in the response. This completes the command–response cycle for the first command. As long as no command other than GET RESPONSE is sent to the card, the response data can be requested multiple times.

Table 7.63 The functionality of GET RESPONSE according to ISO/IEC 7816-4 and GSM11.11

GET RESPONSE	
Command	• amount of data to be sent
Response	• data • return code

If commands are completely encrypted as part of secure messaging, transmission problems can occur with the T = 0 protocol, which requires both an unencrypted instruction byte and an unencrypted Le byte. The ENVELOPE command gets around this restriction by embedding a complete APDU, with its header and data section, in the data section of the APDU of the ENVELOPE command. This can be encrypted without any restrictions and transmitted using

[5] See also Section 6.4.2, 'The T = 0 transmission protocol'

any protocol. The same procedure is used for the response generated by the smart card, which is also embedded in the APDU of the ENVELOPE command. The ENVELOPE command is also used to implement proactive commands for the SAT and USAT.[6] This use corresponds to the above description, with the exception that the data are not encrypted.

Table 7.64 The functionality of ENVELOPE according to ISO/IEC 7816-4

ENVELOPE	
Command	• command APDU
Response	• response APDU • return code

Logical channels can be used which allow up to four applications in a single smart card to be addressed independently of each other.[7] The commands and applications are coordinated by two bits in the class byte. Before the terminal can use a new logical channel, it must explicitly advise the smart card via the MANAGE CHANNEL command. This signals to the card that an additional channel is needed. The channel number can be specified explicitly by the terminal, or the card can supply the number of a free channel in its response. When a new logical channel is opened from the standard channel (channel 0), the behavior of the card in the new channel is the same as after a reset, which means that the MF is selected and no security state has been attained. When a new logical channel is opened from a channel other than channel 0, the current DF selection and current security state are retained. When a logical channel is closed, the associated file selection and security state are deleted.

Table 7.65 The functionality of MANAGE CHANNEL according to ISO/IEC 7816-4

MANAGE CHANNEL	
Command	• *switch:* logical channel open/closed • IF (a particular channel is desired) THEN • logical channel number
Response	• logical channel number (if a new logical channel is opened) • return code

7.13 DATABASE COMMANDS: SCQL

In the beginning, smart cards were used as identification media and thus represented typical single-user systems. However, the sophisticated security mechanisms and access mechanisms

[6] See also Section 13.2.4, 'The SIM'
[7] See also Section 6.7, 'Logical Channels'

of multiapplication smart cards can certainly be used to configure and use smart cards as multiuser systems. Of course, the cost and effort involved should not be underestimated. The cryptographic protective mechanisms needed for this can also quickly reach a critical level of complexity.

The situation here is similar to the situation with the various file structures. In principle, a record-oriented telephone directory can be constructed using a transparent file structure, but it is much easier to use a linear variable file structure.

This is the basic reason why a subset of the database query language SQL (structured query language) has been standardized for use with smart cards in ISO/IEC 7816-7 and has been incorporated into the product range of various operating system producers. The subset of SQL for smart cards defined in the ISO/IEC 9075 standard is called 'structured card query language' (SCQL). The primary application area for smart cards with SCQL capability is health care, since in this area, a variety of persons and organizations must access data using several different read and write access privileges. However, there is presently no large application that utilizes SCQL.

SCQL, like SQL, supports table-oriented database systems. Such systems consist of a table name, a fixed number of named columns and a variable number of rows. 'Logical views' can be applied to this table. If a view is static, this means that a certain number of columns are assigned to this view. A dynamic view, by contrast, is a selection of rows that satisfy a particular condition (e.g., given name = "Wolfgang"). Combinations of static and dynamic views are allowed. Each view has its own name and can be used as the basis for reading and writing data.

Table		
column 1	column 2	column 3
row 1	row 1	row 1
row 2	row 2	row 2
row 3	row 3	row 3
row 4	row 4	row 4
row 5	row 5	row 5

View 1 (static)	
column 1	column 2
row 1	row 1
row 2	row 2
row 3	row 3
row 4	row 4
row 5	row 5

View 2 (dynamic)		
column 1	column 2	column 3
row 3	row 3	row 3
row 5	row 5	row 5

Figure 7.23 An SCQL table with three columns and five rows. Static view and dynamic views have been created for this table. View 1 is static and shows columns 1 and 2 in their entirety. View 2 is dynamic and shows the rows in all three columns that satisfy a certain condition (not defined here)

SCQL in smart cards requires an additional type of file structure, known as 'database', to be included in the operating system. A file with this structure is called a database file (DBF), and it is a database object that can be located directly below the MF or under a DF. It contains the data tables and associated system tables, and it can be addressed by database commands without first being selected. A DBF is a logical structure that may be distributed across several EFs, depending on the smart card operating system.

Three system tables must be set up in the DBF to manage users and privileges. The first of these is the object description table, which stores information about the tables and their associated views. This table is complemented by the user description table, which is defined by the user of the database system. The third DBF table is the privilege description table, which defines the privileges of the individual users with regard to tables and views, as well as the operations they are allowed to perform on the tables and views.

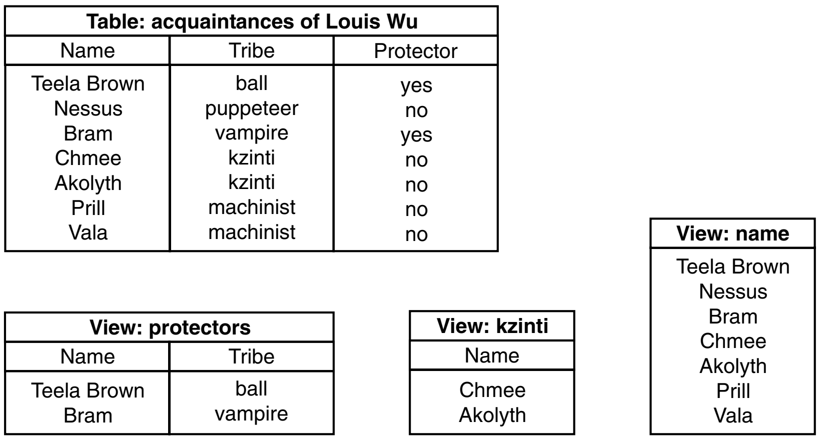


Table: acquaintances of Louis Wu		
Name	Tribe	Protector
Teela Brown	ball	yes
Nessus	puppeteer	no
Bram	vampire	yes
Chmee	kzinti	no
Akolyth	kzinti	no
Prill	machinist	no
Vala	machinist	no

View: protectors	
Name	Tribe
Teela Brown	ball
Bram	vampire

View: kzinti
Name
Chmee
Akolyth

View: name
Teela Brown
Nessus
Bram
Chmee
Akolyth
Prill
Vala

Figure 7.24 A sample SCQL table with the columns 'Name' and 'Tribe', the attribute 'Protector' and seven entries. Three views are also shown. Different access privileges for the various views can be defined in SCQL

The operations that can be performed on an SCQL table or view are read, insert, write and delete, all of which are governed by access privileges. A cursor is defined for read and write operations in a table or view, in order to mark the row to which an operation will be applied.

There are three basic SCQL commands: PERFORM SCQL OPERATION, PERFORM TRANSACTION OPERATION and PERFORM USER OPERATION. Only three instruction codes (INS) are needed for these, since the actual SCQL operations are initiated via parameter byte P2. All data in the command body and response body are TLV-coded data objects. The three commands are summarized in Tables 7.66, 7.67 and 7.68.

Although SCQL has many limitations relative to SQL, such as no sorting, no nested queries, no joins and so on, it still has definite potential for use in the multiuser area. However, the memories of currently available microcontrollers are not yet large enough for extensive SCQL databases in smart cards. It may thus take a while until there is a major application that uses SCQL.

Table 7.66 The SCQL command PERFORM SCQL OPERATION and its functions according to ISO/IEC 7816-7

PERFORM SCQL OPERATION	
CREATE TABLE	Generates a table with its columns and column names.
CREATE VIEW	Creates a new view (static or dynamic) for a table.
CREATE DICTIONARY	Creates the object description table, user description table and privilege description table.
DROP TABLE	Deletes a table.
DROP VIEW	Deletes a view.
GRANT	Grants access privileges to a single user, a group of users or all users.
REVOKE	Revokes access privileges previously granted by GRANT.
DECLARE CURSOR	Positions a cursor that marks a row in a table, view or system table.
OPEN	Activates a cursor in the first row.
NEXT	Moves the cursor to the next row.
FETCH	Reads the row marked by the cursor.
FETCH NEXT	Reads the next logical row after the row where the cursor is located. The cursor is moved to the row to be read.
INSERT	Adds a row to the end of the table without modifying the cursor.
UPDATE	Writes data to one or more fields in a row in a table. The row is selected by the cursor.
DELETE	Deletes the row marked by the cursor from the table. The cursor is positioned to the following row.

Table 7.67 The SCQL command PERFORM TRANSACTION OPERATION and its functions according to ISO/IEC 7816-7

PERFORM TRANSACTION OPERATION	
BEGIN	Reserves space for a memory image for the functions COMMIT and ROLLBACK, for example for a row of a table.
COMMIT	Tests all changes that have been made to a table since the last BEGIN command.
ROLLBACK	Restores a table to the state it had before the last BEGIN command.

Table 7.68 The SCQL command PERFORM USER OPERATION and its functions according to ISO/IEC 7816-7

PERFORM USER OPERATION	
PRESENT USER	Identifies a user by means of his or her user ID.
CREATE USER	Creates an entry for a new user.
DELETE USER	Deletes the entry for a user.

7.14 COMMANDS FOR ELECTRONIC PURSES

Part 3 of the European standard for universal electronic purses, EN 1546, defines six commands for electronic purses and 12 commands for the security module in the terminal, which itself may be a smart card. The basic structures of the four most important commands used with smart card electronic purses[8] are described here. These commands can be utilized to run an application in a smart card for making 'cashless' payments from a prepaid purse and refilling the purse. The commands for error recovery, currency conversion, parameter modification and canceling a payment are not described here, nor are those for the security module. The Common European Purse System (CEPS) specification for electronic purses defines commands that are very similar to those defined by EN 1546.

The commands described here would fit just as well under 'Application-Specific Commands' (Section 7.16), since they are defined specifically for this one application. They can never be used for any other purpose than electronic purses, since they have been optimized for this application. However, we dedicate a section to them because electronic purses are one of the main future applications for smart cards, besides telecommunications.

All electronic purse transactions are divided into three steps according to EN 1546. In the first step, the card is initialized using the command INITIALIZE IEP for Load / for Purchase. In the second step, a command is executed to perform the actual transaction, such as filling the purse or paying with the purse. In the optional third step, the transaction just performed is confirmed. All purse commands directly access files in the purse application of the smart card for both writing and reading. These files hold the purse balance, log entries and various parameters.

The individual steps of a purse transaction are executed using the commands described below. The EN 1546 standard precisely defines the internal processes of each command with regard to functionality and the sequence of the individual steps. All implementations thus have at least the same general processes.

The INITIALIZE IEP command can be used for several purposes. A parameter is used to select initialization of a purse loading transaction, a purchase transaction or another type of transaction.

Loading (crediting) the purse in the smart card is initiated by the command INITIALIZE IEP for Load. The transferred data, such as a currency code and amount to be loaded, are checked in the card to see whether they match prescribed values in the parameter files. Freely definable data (user-determined data) can also be stored in a log file. Next, a transaction counter is incremented and a signature S_1 is generated for various data (such as the current balance and expiry date), so that this information can be transferred to the terminal without risk of manipulation.

In the second step of the load transaction, the card essentially receives information about the keys to be used and a signature S_2 via the CREDIT IEP command. This information comes from the security module in the terminal, and besides protecting the data, it allows the card to authenticate the security module. The smart card has already been authenticated with respect to the security module in the terminal by the previous INITIALIZE IEP for Load

[8] Command sequences and general system structures of electronic purse systems are described in detail in Section 12.3.1, 'The CEN EN 1546 standard'

Table 7.69 The functionality of INITIALIZE IEP for Load according to EN 1546-3

INITIALIZE IEP for Load	
Command	• — • amount to be loaded (M_{LDA}) • currency code ($CURR_{LDA}$) • PPSAM descriptor (PPSAM) • random number (R) • user-determined data (DD)
Response	• purse provider identifier (PP_{IEP}) • IEP identifier • cryptographic algorithm used (ALG_{IEP}) • expiry date ($DEXP_{IEP}$) • purse balance (BAL_{IEP}) • IEP transaction number (NT_{IEP}) • key information (IK_{IEP}) • signature S_1 • return code (CC_{IEP})

command. After successfully testing S_2, the card increases the purse balance, makes a new entry in the log file and generates signature S_3 for confirmation. This signature is then used by the security module in the terminal as confirmation of correct booking of the amount to be loaded.

Table 7.70 The functionality of CREDIT IEP according to EN 1546–3

CREDIT IEP	
Command	• key information (IK_{PPSAM}) • signature S_2 • user-determined data (DD)
Response	• signature S_3 • return code (CC_{IEP})

The second process consists of making a purchase using the electronic purse. This transaction is again initiated by the INITIALIZE IEP command, this time using the ‘for Purchase’ option. The smart card does not receive any data with this command, but it does increment the transaction counter. Next, a signature S_1 is generated for data such as the expiry date, transaction counter and IEP identifier. This information, which is protected for transmission by signature S_1, is then sent to the terminal together with some additional data.

The actual payment transaction is performed by the DEBIT IEP command. It sends information to the electronic purse in the smart card regarding the amount to be debited and current key versions, as well as another signature. This signature can be used to test the authenticity

Table 7.71 The functionality of INITIALIZE IEP for Purchase according to EN 1546-3

INITIALIZE IEP for Purchase	
Command	• —
Response	• identifier of the purse provider (PP_{IEP}) • IEP identifier • cryptographic algorithm used (ALG_{IEP}) • expiry date ($DEXP_{IEP}$) • purse balance (BAL_{IEP}) • currency code ($CURR_{IEP}$) • authentication mode (AM_{IEP}) • IEP transaction number (NT_{IEP}) • key information (IK_{IEP}) • signature S_1 • return code (CC_{IEP})

of the security module in the terminal in the same manner as for loading the purse. If this test is successful, the purse balance is decreased, the purchase transaction log is updated and another signature, S_3, is generated to confirm the entire transaction. This signature is placed in the response to the DEBIT IEP command. It serves as confirmation to the security module in the terminal that the amount was properly debited in the smart card.

Table 7.72 The functionality of DEBIT IEP according to EN 1546–3

DEBIT IEP	
Command	• PSAM identifier • PSAM transaction number (NT_{PSAM}) • amount to be debited (M_{PDA}) • currency code ($CURR_{PDA}$) • key information (IK_{PSAM}) • signature S_2 • user-specific data (DD)
Response	• signature S_3 • return code (CC_{IEP})

Here we have only presented a brief summary of the four most important commands for electronic purses as specified in EN 1546. This standard is extremely comprehensive and includes many options and possibilities for system design, which naturally can affect the structures of the commands.[9]

[9] See also Section 12.3.1, 'The CEN EN 1546 standard'

7.15 COMMANDS FOR CREDIT AND DEBIT CARDS

The joint specification of Europay, MasterCard and Visa for smart cards used for financial transactions, EMV,[10] specifies two commands especially designed for financial transactions. In principle, these two extremely flexible commands could be used to implement an electronic purse in a smart card. However, their intended use lies more in the realm of credit and debit transactions, which is why their use for these two applications is described here. We devote a separate section to these commands because credit and debit cards incorporating chips are expected to be produced in very large numbers in the future. Their significance is thus correspondingly large.

The two commands GET PROCESSING OPTIONS and GENERATE APPLICATION CRYPTOGRAM are based on TLV-coded data objects in the data section of the command and response. This creates a considerable variety of possible variations and options, which can be exploited by each application as necessary.

The GET PROCESSING OPTIONS command is used to initiate a payment transaction. It transfers the processing options data object list (PDOL), which contains TLV-coded data for processing the rest of the payment transaction, from the terminal to the smart card. This data could be the transaction amount, for example. The card returns a BER-TLV coded data object containing the application interchange profile (AIP), which describes the functions supported by the smart card, and the application file locator (AFL), which specifies the location of the application data.

Table 7.73 The functionality of GET PROCESSING OPTIONS according to EMV

GET PROCESSING OPTIONS	
Command	• processing options data object list (PDOL)
Response	• application interchange profile (AIP) • application file locator (AFL) • return code

The second command for the payment transaction process in a credit card with a chip is GENERATE APPLICATION CRYPTOGRAM. The data in the command APDU and response APDU of this command are TLV-coded. It transfers all of the data necessary for a payment transaction to the card, together with the desired application cryptogram. The card then determines how the rest of the payment transaction should proceed, based on the received and stored data. As a result, it returns an application cryptogram to the terminal. In the simplest case, this may be the transaction cryptogram. This concludes the payment transaction process.

The application cryptogram returned by the smart card may contain an authorization request instead of a transaction cryptogram. In the process of determining how the payment transaction should proceed, if the smart card concludes that online authorization is necessary, the application cryptogram that it returns to the terminal contains a request to the authorization

[10] See also Section 12.4, 'The EMV Application'

Table 7.74 The functionality of GENERATE APPLICATION CRYPTOGRAM according to EMV

GENERATE APPLICATION CRYPTOGRAM	
Command	• desired application cryptogram • transaction-related data
Response	• cryptogram information data • application transaction counter (ATC) • application cryptogram (AC) • return code

center superior to the terminal. After the authorization center has processed this request, the corresponding information is sent to the smart card using a second GENERATE APPLICATION CRYPTOGRAM command. The card can then generate the transaction cryptogram for the payment and send it to the terminal.[11]

7.16 APPLICATION-SPECIFIC COMMANDS

There are a large number of commands that are tailored to specific applications. They are mainly used to minimize memory space or processing time. The majority of these commands are so specific that they are not included in any standard, or they are defined in a standard for use in a particular application area.

A list of all application-specific commands would exceed the scope of this chapter. As a representative example of such commands, we present the RUN GSM ALGORITHM command, which is the only application-specific command in the GSM 11.11 specification. It is used to simultaneously generate a dynamic, card-specific key and authenticate the card with respect to the GSM background system. This function is so specific to the GSM application that it would make no sense to include it in a general smart card standard. The command uses a cryptographic algorithm specific to GSM, and the two initial values generated from the transferred random number would be useless in any other application.

Table 7.75 The functionality of RUN GSM ALGORITHM

RUN GSM ALGORITHM	
Command	• random number
Response	• dynamic key • **enc** (key; random number) • return code

[11] See also Section 12.4, 'The EMV Application'

8 Security Techniques

One of the main advantages of smart cards in comparison with other data storage media, such as magnetic-stripe cards and diskettes, is that they can store data such that it is protected and kept secret. An essential requirement for this is chip hardware that is tailored and optimized for this purpose, along with suitable cryptographic methods for protecting confidential data. However, security depends on more than just special microcontroller hardware and algorithms implemented in operating system software. The security of the smart card application, and the design principles used by its developers, are also of fundamental importance. This chapter is a compendium of essential principles, methods and strategies for producing secure smart cards and secure smart card applications.

8.1 USER IDENTIFICATION

Since ancient times, a variety of techniques have been used for the unambiguous identification of persons. The simplest form of identification is an identity card bearing a photograph or a signature written in the presence of the examiner. The photograph on an identity card can be compared with the actual person, with the result being an assessment of the genuineness of the person's identity.

In the field of information technology, this comparison is not so easy, since it must be performed by a computer instead of another person. Despite their success in performing mindless activities, computers still have tremendous difficulty in performing intelligent tasks. Consequently, entering a password via a keypad has generally become the preferred identification method. The effort needed for the comparison is minimal, since essentially all the computer has to do is to compare the entered password with a stored reference value and make a simple yes/no decision. Password comparison effectively amounts to making a decision regarding the genuineness of the identity of the person being tested.

There are basically only three different methods that can be used to identify a person. If a password is used, what is tested is whether the person knows a particular secret. If he or she does, the conclusion is that the person is who he or she claims to be. The second option is

Smart Card Handbook, Third Edition. W. Rankl and W. Effing
© 2004 John Wiley & Sons, Ltd ISBN: 0-470-85668-8

to test whether a person possesses a particular object. The third possibility is to test specific, unique bodily features of the person.

Methods that rely on knowing a secret or possessing a particular item have a significant drawback, which is that the person to be identified must either remember something or carry something on his or her person. Depending on the situation, the fact that the secret or object can be passed to another person can be considered to be an advantage or a disadvantage. In any case, it is not possible to unambiguously ascertain that the person holding the secret or the object is truly its legitimate owner, instead of someone else who may have illegitimately acquired the secret or item that is tested.

The third identification method eliminates this transferability, since it is based on using specific features of the human body for purposes of identification. Of course, the measurements are in most cases technically difficult, since for obvious reasons biological features that can be easily measured, such as weight or height, cannot be used.

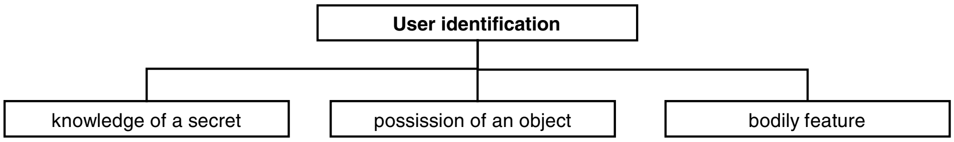


Figure 8.1 Classification of methods for identifying a person

It is easier to understand these three possible identification methods if you consider the following example. Suppose you have to meet an unknown person at the train station. As soon as you see a possible candidate, you have the problem of deciding whether he is really the person you are looking for. However, if the unknown person shows up at the right place and the right time, this actually amounts to an implicit test of a secret, since you can at least hope that the place and time of your meeting are not generally known. An explicit test of a secret would occur if the unknown person were to utter a password that is known only to you and him. Alternatively, he could identify himself by means of an item that he possesses, for example by holding a newspaper printed on a specific day under his arm. Certainly, the most secure method would be to check the person for a specific bodily feature. Perhaps he has an unusually large nose, like Pinocchio's (which grows very long when he lies . . .).

This train station scenario clearly shows that identifying an unknown person can be regarded as a classic problem that occurs in everyday life as well as in spy novels, rather than just being limited to computers and smart cards.

It has now become a common practice to enter a PIN into many types of automated equipment and computers. The resulting marked increase in the number of PINs used for various purposes makes it very difficult for ordinary people to keep track of all of their PIN codes. After all, who can remember 20 or more different PINs? The security and good name of a system are naturally not improved if every user jots down his or her PIN on the card, since the number of cases of fraud will be excessive. For this reason, a desire to use other identification methods in place of PIN codes has arisen in recent years. Biometric features that allow a particular person to be unambiguously identified by a machine are ideal for this purpose.

8.1.1 Testing a secret number

The most commonly used method of user identification is entering a secret number, which is generally referred to by the abbreviation PIN (for ‘personal identification number’), or sometimes CHV (cardholder verification).

A PIN is usually a four-digit number, usually composed of the decimal numerals 0 through 9. The reason for using a purely numeric entry is simply that card terminals generally only have numeric keypads. The PIN is entered using the terminal keypad or a computer keyboard, and then sent to the smart card. The smart card compares the value that it receives with an internally stored reference value and reports the result to the terminal.

PIN entry is particularly considered to be a security issue in financial transaction applications, so requirements relating to the nature of the keypad are frequently found in this application area. Special keypads that satisfy these requirements are often called ‘PIN pads’. In Germany, for example, there is a requirement (from the ZKA) that the PIN for a Eurocheque card can only be entered using a keypad having special mechanical and cryptographic protection. PIN pads have all the features of a security module, such as case-opening sensors and foils to protect against drilling, and they encrypt the PIN directly as it is entered. This provides reliable protection against tampering with a keypad in order to allow a PIN to be intercepted while it is being entered.

A distinction can be made between static and modifiable PINs. A static pin cannot be changed by the user, so it effectively must be memorized by the user. If it becomes known, the user should destroy the card and obtain a new one with a different static PIN. A modifiable PIN can be altered according to the wishes of the user, or changed to a number that the user finds easy to remember. There is a danger in this, since the numbers that many people find easy to remember are ones such as "1234", "4711" and "0815". The smart card does not check for the use of such trivial numbers, since there is not enough memory available to store the necessary table. However, it would be perfectly conceivable for the terminal to prohibit the PIN from being changed to such a number. In order to change a PIN, it is always necessary to enter the PIN, since otherwise an attacker could replace every existing PIN with one of his own.

The situation is different with personal unblocking keys (PUKs), which are also called ‘super PINs’. These keys usually have more digits than a normal PIN (a typical value is six), and they are used to reset the retry counter of a PIN to zero if it has reached its maximum value. A new PIN is also entered into the card when the PUK is entered, since resetting the retry counter is of little use if the user has forgotten the PIN. This is usually the case when the retry counter has reached its maximum value.

There are also applications that use transport PINs. In this case, the smart card is personalized with a random PIN and the cardholder receives the PIN value by letter. The cardholder then replaces the PIN used for card personalization with a PIN of his or her choice before actually using the card. In a similar method, called the ‘null PIN’ method, the card is preloaded with a trivial PIN, such as "0000", and the smart card again forces the PIN to be changed before it can be used. Both of these methods prevent a PIN that has been ‘spied out’ during card personalization from later being put to good use.

According to a recommendation of the ISO 9564-1 standard, the PIN should consist of four to 12 alphanumeric characters in order to minimize the probability of determining the correct PIN by pure trial and error. However, the situation in actual practice is often somewhat different. Entering non-numeric characters is technically impossible in many locations, since the keypad has only numeric characters.

The number of characters in a PIN depends not only on the desired level of security, but also to a large degree on the memory capacity of the average card user. For years, people have been accustomed to using four-digit PINs, which means that changing to PINs with six or more digits would be very difficult. In practice, the presumed improvement in security provided by using a six-digit or eight-digit PIN might turn out to be purely theoretical. Many people find it difficult to remember numbers of this length, especially if they do not use them very often, and consequently write them down on the card or on a slip of paper kept near the card. The level of security with a long PIN is then significantly lower than with a short PIN.

The perfectly well-founded insistence on periodically changing PINs meets with a similar fate. It may work with a high-security application having only a few users, but it is fatal for the acceptance of a mass-market application, which tries to use the simplest possible methods in order to accommodate people with poor memories.

In this regard, there is another very important issue. In many cases, entering and verifying a PIN does more than just identify the user and indicate legitimate possession of the card. It also represents a profession of intent by the user, who agrees to a particular transaction by entering his or her PIN. A good example is entering a PIN into a cash dispenser. This identifies the card user by means of his of her knowledge of the secret PIN, but it also represents a declaration by the user that he or she agrees to have a certain amount of cash paid out from his or her account. This is a very important consideration in connection with certain biometric features, some of which can be tested without the explicit permission of the person in question and do not necessarily represent a profession of intent.

The probability of guessing a PIN

The simplest attack on a PIN, aside from watching it being entered, is just guessing. The probability of success depends in part on the length of the PIN, the characters from which it can be composed and how many attempts are allowed. The probability of correctly guessing a four-digit PIN in three tries is 0.03 %, which is not particularly high. Two basic formulas for guessing passwords are presented here. They can be used in actual practice to estimate the risk associated with using a particular password.

$$x = m^n \tag{8.1}$$

$$P = i/m^n \tag{8.2}$$

Incidentally, there is yet a fourth factor related to guessing a PIN, which for a long time has been inexcusably neglected. This is the uniformity of the distribution of PINs within an application. It is much easier to guess a PIN if you know that certain PINs are more common than others. The actual significance of this important secondary factor became evident almost overnight in 1977 in connection with German Eurocheque cards. Although the detailed procedure for computing a PIN from the data stored in the magnetic stripe of the Eurocheque

Table 8.1 Definitions and descriptions of the variables in Formulas (8.1) and (8.2)

Variable	Example	Description
i	3	number of guesses
m	10	number of possible characters per position
n	4	number of positions
P	0.0003 (0.03 %)	probability of guessing the password
x	10,000	number of possible passwords

card is still secret, at least a few general steps of the procedure became known. From this information, it could be concluded that the PINs that are generated are not uniformly distributed, since the algorithm used produces the numerals 0 through 5 significantly more often than 6 through 9. It also became known that the PIN algorithm suppressed leading zeros when generating PINs. With such a non-uniform distribution, it is not necessary to make 3333 attempts in order to correctly guess a four-digit PIN with the permitted number of incorrect guesses (3), but only 150 [Karten 97]. With 10.5 % of the cards, the distribution is so poor that only 72 attempts will suffice if the characteristics of the PIN generation algorithm are taken into account [Schindler 97]. The end result of all this is that an improved PIN generation

Table 8.2 PINs and passwords with various lengths and codings, and the number of possible combinations

Type of PIN or password	Range of values or coding of the PIN or password	Number of possible PINs or passwords
1-digit PIN	PIN ∈ {0 ... 9}	10
1-character password	password ∈ {0 ... 9, "A" ... "Z"}	36
1-character password	password ∈ {0 ... 9, "a" ... "z", "A" ... "Z"}	62
1-character password	password ∈ {0 ... 9, "a" ... "z", "A" ... "Z", 20 arbitrary special characters }	82
4-digit PIN, no leading zero	PIN ∈ {1000 ... 9999}	9.00×10^3
4-digit PIN	PIN ∈ {0000 ... 9999}	1.00×10^4
4-character password	password ∈ {0 ... 9, "A" ... "Z"}	1.68×10^6
4-character password	password ∈ {0 ... 9, "a" ... "z", "A" ... "Z"}	1.48×10^7
4-character password	password ∈ {0 ... 9, "a" ... "z", "A" ... "Z", 20 arbitrary special characters}	4.52×10^7
5-digit PIN, no leading zero	PIN ∈ {10000 ... 99999}	8.9×10^4
5-digit PIN	PIN ∈ {00000 ... 99999}	1.00×10^5
6-digit PIN, no leading zero	PIN ∈ {100000 ... 999999}	8.99×10^5
6-digit PIN	PIN ∈ {000000 ... 999999}	1.00×10^6
6-character password	password ∈ {0 ... 9, "A" ... "Z"}	2.18×10^9
6-character password	password ∈ {0 ... 9, "a" ... "z", "A" ... "Z"}	5.68×10^{10}
6-character password	password ∈ {0 ... 9, "a" ... "z", "A" ... "Z", 20 arbitrary special characters}	3.04×10^{11}

algorithm is used with new Eurocheque cards, and the DES algorithm originally used has been replaced by a triple-DES algorithm.

Generating a PIN

In order to generate a PIN for a smart card, it is first necessary to have a random number generator and an algorithm that converts a random number into an ASCII-coded PIN of the required length. A table of known trivial combinations can then be used to detect and discard trivial PINs. Finally, the PIN must be stored in the smart card, and the VERIFY command must then be used as necessary to compare it with PIN codes transferred to the card from the terminal.

A somewhat more complicated procedure for generating PINs is required for a system that uses magnetic-stripe cards instead of smart cards. This is because it must be possible for a cash dispenser operating offline to test an entered PIN using data contained in the magnetic stripe. This requirement does not actually apply to smart cards, but all debit cards (such as Eurocheque cards) presently have magnetic stripes for reasons of compatibility, even if they also have microcontrollers. When hybrid cards with both chips and magnetic stripes are used, the PIN generation algorithm must therefore be deterministic, which means that it must always produce the same result for a given set of input values. A random number generator cannot do this.

A procedure is thus needed that can generate a PIN based on the magnetic stripe data. In order to avoid having the security of the system depend on the procedure itself, a secret key should also be involved in the computation. Figure 8.2 illustrates an algorithm similar to the one that is used for German Eurocheque cards. Its inputs consist of the bank routing code, the account number and the serial number of the card. This algorithm uses the DES algorithm with a secret key to generate a four-digit PIN. This procedure suffers from the previously mentioned disadvantage that it produces PINs that are not uniformly distributed over the total possible number space ("0000" to "9999" in the case of a four-digit PIN). This is due to the mapping rule that is used to convert the hexadecimal numerals ('A', 'B', 'C', 'D', 'E', and 'F') into decimal numerals following the encryption process. This undesirable feature could be easily avoided by using a better mapping rule. The DES algorithm is used in part because the key rather than the procedure must be kept secret, and in part due to its properties of confusion and diffusion.[1]

The PIN generation procedure that was used between 1981 and 1997[2] for German Eurocheque cards produced PINs that were not uniformly distributed over the entire number space. Consequently, some PINs were significantly more probable than others, and such PINs could be used for attacks (which were generally not successful). For this reason, it is important to ensure that procedures used to generate PINs produce PINs that are distributed uniformly over the available number space.

[1] See also Section 4.6.1, ‘Symmetric cryptographic algorithms’
[2] See also [Kuhn 97]

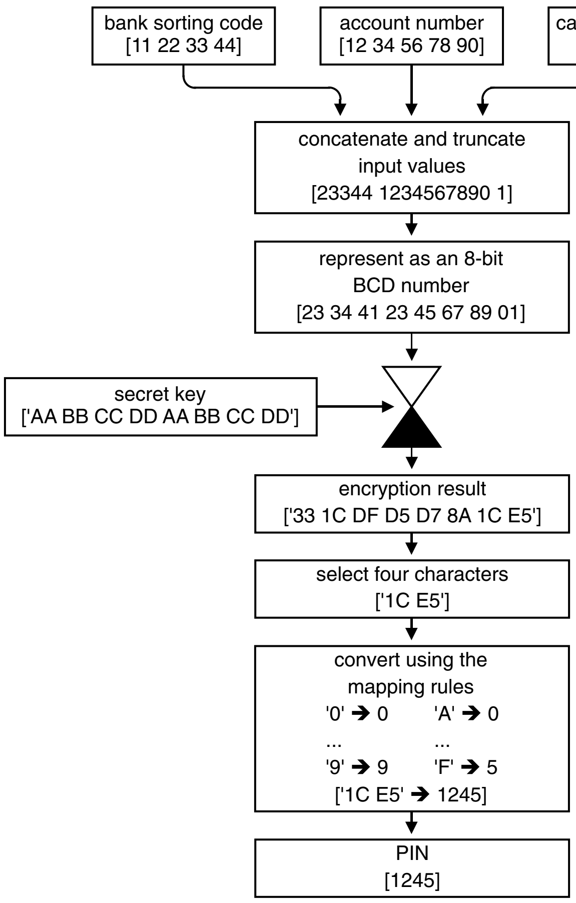


Figure 8.2 Example of a procedure for generating a four-digit PIN using the DES algorithm and three card-specific data elements (bank routing code, account number and card serial number). This procedure has the disadvantage that it produces PINs that are not uniformly distributed (i.e., some PINs occur more often than others), due to the mapping rule used. Sample values are shown in square brackets. This procedure is remotely similar to the procedure used for German Eurocheque cards and is based on two articles in *Die Datenschleuder* [Müller-Maguhn 97a, Müller-Maguhn 97b]

Testing the authenticity of a terminal

As is well known, entering a PIN is used to verify the identity of the user. However, the user might equally well want to be able to verify the genuineness of a terminal. For instance, consider the possibility of a dummy cash dispenser. Someone with fraudulent intentions could use such a machine to collect PINs entered by unsuspecting users. If the person who set up the machine then steals the users' cards, he or she could use the PINs acquired via the dummy cash dispenser to withdraw money from the cardholders' accounts. All of this is possible because there is no way for the user to test the genuineness of the terminal.

However, there is a procedure that can be used to defend against this type of attack. It involves storing a password in a file in the card. This password is known only to the card user and can be changed only by the user. It can be a name or a number chosen by the user. The smart card operating system allows read access to this file only after the terminal has been authenticated by the card.[3]

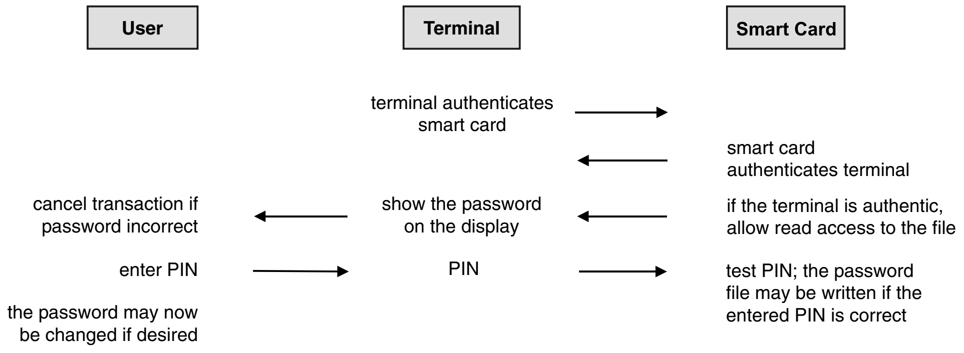

User
Terminal
Smart Card
terminal authenticates smart card
smart card authenticates terminal
cancel transaction if password incorrect
show the password on the display
if the terminal is authentic, allow read access to the file
enter PIN
PIN
test PIN; the password file may be written if the entered PIN is correct
the password may now be changed if desired

Figure 8.3 A procedure for ensuring that a PIN can only be entered using a genuine terminal. A prerequisite is that the cardholder does not enter the PIN until the correct password has been displayed by the terminal

The first thing that happens after the user has inserted a smart card into the terminal is a mutual authentication transaction between the card and the terminal. If this is successful, each party knows that the other party is genuine. The card then allows read access to the file containing the user's secret password, which is displayed on the terminal. The user sees his or her password and thus knows that the terminal is genuine, since it would have otherwise had no access to the file containing the user's secret password. He or she can now safely enter the PIN.

This procedure is also a simple way to prevent PINs from being entered into terminals that have been manipulated. Any arbitrary word or number can be used for the password. It should be possible for the cardholder to change the password as desired, in order to prevent potential attackers from being able to ferret it out. This procedure can also be extended or modified as necessary to meet other demands of a similar nature.

8.1.2 Biometric methods

The steadily increasing use of passwords and PINs is producing a steadily increasing level of user resistance to this type of identification. Few people find it particularly difficult to remember

[3] The structure of a suitable smart card application is described in detail in Section 15.9.3, 'Testing the genuineness of a terminal'

a few frequently used combinations of numbers or letters. However, if a card-specific PIN code is used only rarely, such as every two months to obtain money from a cash dispenser, most people find it difficult to remember the PIN. Matters are only made worse by the unconscious fear that the machine will confiscate the card if the PIN is entered incorrectly three times in a row.

This is certainly one of the main reasons why biometric methods are finding increasing favor in many areas. They are not necessarily faster or more secure than PIN entry, but they can make things much easier for users. If the level of security provided by biometric methods is equivalent to that provided by PIN codes, system operators will also be prepared to use them. After all, biometric features cannot be transferred to another person as easily as PINs. This means that the actual person is identified, rather than a secret shared by the user and the system operator.

8.1.2.1 Basic principles

A biometric identification method is a method that can unambiguously identify a person by means of unique, individual biological features. Here a distinction can be made between physiological and behavioral features. If the features tested by the method are directly related to the person's body and are fully independent of conscious behavior patterns, they are called physiological biometric features. Biometric methods based on behavioral features, by contrast, utilize certain features that can be consciously changed within certain limits, but that are still characteristic of a particular person.

An essential aspect of biometric feature testing is the question of user acceptance. If the method used is similar to existing, well-known methods, users will be more willing to accept and use it. A typical example is a handwritten signature, which has been used for generations in almost all cultures for identification and indicating agreement or consent. Social aspects also play an important role. In many countries, fingerprinting is primarily used by the police and security forces. This could adversely affect the acceptance of biometric methods based on fingerprints.

Another point that must be considered is the concerns that users may have regarding medical and hygienic aspects. For instance, users may be afraid of acquiring a disease from optical scanning of their retinas, or that the laser light will damage their eyes. Even though such fears may be fully subjective and lack any scientific basis, they can still strongly affect user behavior, and above all user acceptance of the method. Before any biometric identification method is employed, such aspects should be fully understood.

There is yet another difference between biometric and knowledge-based identification methods, which can be considered to be either an advantage or a disadvantage according to one's point of view. This is that biological features cannot be transferred to another person. With a system that uses biometric methods for identification, this means, for example, that it is not possible to give your card and your PIN to a trusted person who can then use the card in the intended manner. System operators naturally find such actions absolutely shocking, since revealing a PIN is prohibited in almost all systems. However, nearly everyone knows how loosely such prohibitions are observed in practice.

Biometric features are usually not modifiable, which is precisely what makes them attractive for the unambiguous identification of persons. However, this non-modifiability can certainly

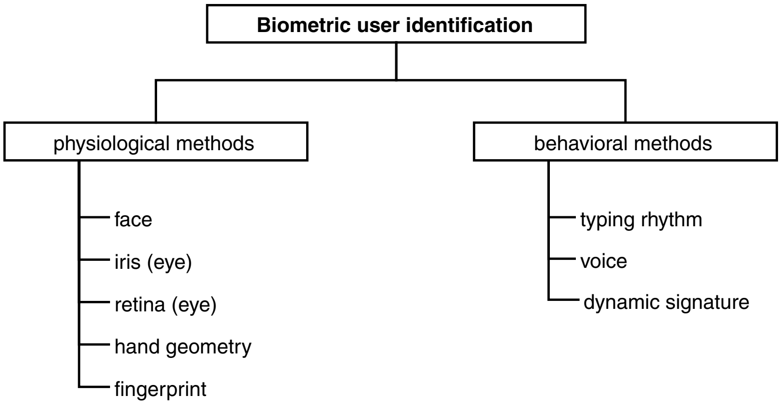


Figure 8.4 Classification of the most important biometric methods for user identification

lead to major complications if a system is compromised. In addition, this non-modifiability in combination with the fact that some biometric features can also be measured with a reasonable amount of effort and cost without the consent of the person involved can lead to serious problems. This can be clearly illustrated using our fingerprint example. Suppose the fingerprints of a person are illicitly taken by means of suitable analysis of an object that this person held in his or her hand while eating in a restaurant, and the data are then made public on the Internet so they can be copied by anyone who wants to do so and has suitable equipment. In such a situation, for the rest of this person's life it would effectively be impossible to unambiguously identify him or her using fingerprints as a biometric feature, since it would never be possible to be sure that he or she actually produced a particular fingerprint.

Biometric features can also be classified according to the ease with which they can be acquired. The classifications that are used are 'open', 'slightly concealed', 'concealed' and 'strongly concealed'. This classification relates to how easily the biometric feature can be acquired by a third party without the consent of the person in question. Open features, such as a person's facial features, can be recorded by simple observation. An example of a slightly concealed feature is a fingerprint, which can be acquired using simple equipment without the awareness of the person in question. A significant amount of equipment is required to acquire a retinal scan, and it is practically impossible to do so without the awareness of the person in question, so retinal patterns belong to the category of concealed features. Strongly concealed features are frequently behavioral features, since in most cases they must be consciously revealed.

Entering a PIN not only tests whether the user knows a secret code, it is also a legally binding equivalent of saying, 'I consent'. This relationship is very important if some other method is to be used in place of a PIN. A test based on a retinal scan performed at a distance of three meters, which happens to not be technically possible at present, could certainly not be considered to indicate the consent of the person in question to any sort of action. In almost all countries, only an intentional manual action of the user can be interpreted as an indication of consent. For instance, breaking the seal of a cardboard box containing software is an unambiguous indication that the user agrees with the printed license conditions. Biometric methods involving fully passive testing of the person in question must therefore be augmented by appropriate user instructions together with something that provides the element of consent.

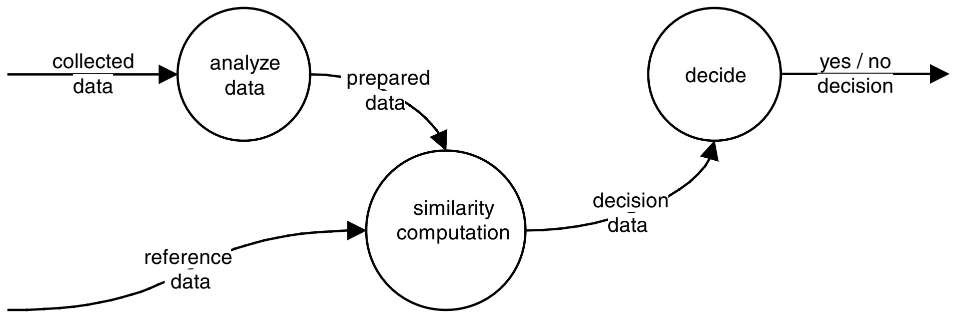


Figure 8.5 Basic data flow for the computational evaluation of a biometric feature

Naturally, not all biological features are suitable for personal identification. A feature must satisfy at least the following criteria before it can be reasonably used:

- it can be measured effectively (in terms of the measuring method, time and costs)
- it must be capable of being uniquely associated with a particular individual
- it must be widely distributed within the population
- it must not be possible to alter the feature with fraudulent intent
- the amount of reference data generated must be small (a few hundred bytes to at most several thousand bytes)
- natural changes to the feature over time must be so small that correct measurement of the feature is always possible
- the measurement method and the feature must be acceptable to users.

With any type of measurement, the result is not always the same, but instead varies from instance to instance. This occurs even with the simplest measurements. For example, if you measure the length of a sheet of paper several times, each result will be slightly different. There are many reasons for this, but it does not create difficulties in practice, since the average value of the measurements will be close to the true value.

Experience shows that the amount of variation among individual measurements depends on the difficulty of making the measurement. For instance, there is a significant technical difference between measuring the weight of a bar of chocolate and measuring the distance between the earth and the moon. Measurements performed on human beings are always difficult and are subject to a wide range of variation.

Figure 8.6 shows an example of the results of measuring a biological feature, such as the length of a finger. The range of variation in the measurement is plotted on the horizontal axis, while the vertical axis indicates the probability of correct identification based on the measured biometric feature. With an ideal biometric feature and an ideal measurement method, there would be no variation, and the curve would be reduced to a vertical line. However, a real feature together in combination with a real method results in the Gaussian ‘bell curve’ shown in the figure. If the measurement result deviates from the reference value, it is not possible to be absolutely sure that the person to be identified has been correctly recognized.

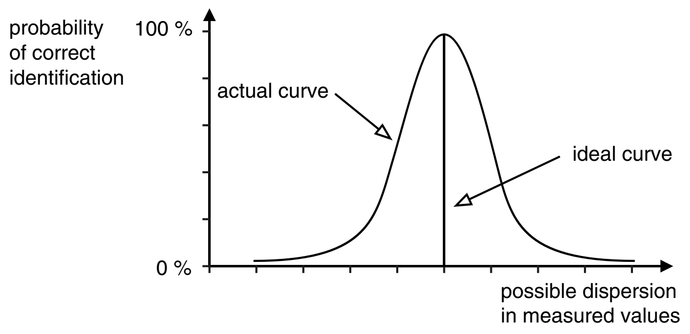


Figure 8.6 Probability distribution for repeated measurements of a biometric feature of a person

Before a biological feature can be tested, the feature of the person in question must first be acquired. This can be done by making repeated measurements and computing the average value. This yields a reference value, which is then stored in the smart card. After this, the smart card can as necessary test whether an actual measurement value sent to it matches the reference value. Depending on the biometric method used, it may be necessary to use a powerful computer to process the actual measurement data into a form that the card can use for comparison. Since identification cannot be established with absolute certainty, a threshold level is needed in order to decide whether the person in question should be recognized as genuine. This threshold level must be set separately for each method and application.

If we take our probability distribution diagram and add a curve for a second person to it, we obtain the diagram shown in Figure 8.7. The additional curve represents an arbitrary person whose measurement curve is close enough to that of the first person for it to affect the identity decision. Since both curves approach the horizontal axis asymptotically, they have an intersection point. At this point, there is an equal probability that the person being tested is genuine or not genuine. Consequently, biometric identification systems use an adjustable threshold level that marks the probability above which the identification is assumed to be correct. The threshold level shown in Figure 8.7 divides the two curves into four regions. These indicate the decision to be taken regarding the identity of the person as deduced from the biometric feature.

What this diagram essentially demonstrates is that there is no such thing as absolutely positive identification. It is only possible to assume, with a high degree of probability, that the person has been correctly identified. The level of this probability can be adjusted using the threshold value. However, in practice the threshold value cannot be set arbitrarily high, since an excessively strict criterion for correct identification produces a large number of false rejections.

The two basic parameters for judging a biometric method are its false acceptance rate (FAR) and its false rejection rate (FRR). The FAR is the probability of incorrect acceptance of the wrong person, while the FRR is the probability of incorrect rejection of the right person. Naturally, these two probabilities cannot be freely chosen, since they are primarily properties

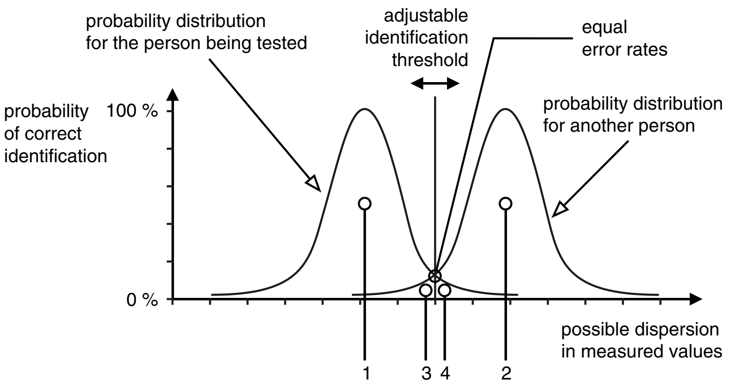


Figure 8.7 Probability distribution and decision regions for a biometric feature measurement:
1 – true positive identification (true acceptance) 3 – false positive identification (false acceptance)
2 – true negative identification (true rejection) 4 – false negative identification (false rejection)

of the biometric method being used and can be modified only within certain limits. In addition, the FAR and FRR are mutually dependent, since a low FRR produces a high FAR and vice versa. For the user, a high FRR means that he or she may be rejected in spite of presenting a legitimate feature, which naturally affects user acceptance of the system. The system operator wants to have not only a low FRR but also a low FAR, in order to prevent false positive identifications.

PIN testing does not require complicated algorithms in the smart card, since it only involves comparing received and stored PIN values. Unfortunately, things are not this easy with biometric features. The reference value is of course stored in the card, but the comparison with the measurement in question normally cannot be performed in the card. This is due to the large amount of processing capacity needed to evaluate biometric features. Since smart cards usually do not have adequate processing capacity for this, the computer-intensive preprocessing of the measurement values is performed externally. The result is then sent to the card, which evaluates the preprocessed data using special algorithms that do not require a lot of memory or processing capacity and then makes a yes/no decision based on the stored reference value.

This method is called ‘oncard matching’ or ‘matching on chip’ (MOC). The amount of time required for matching depends on many factors, such as the biometric method used and whether the data have been preprocessed. For example, it takes approximately two seconds to test fingerprint data in a smart card with an 8-bit processor using data that have been preprocessed in the terminal.

Biometric features are personal data and thus should be appropriately protected. This represents a very good application for smart cards, since the reference data needed for testing never have to leave the card, which makes attacks significantly more difficult. However, if the reference data are stored in a non-secure environment, they can be manipulated and read as desired. In such a situation, a biometric identification method does not provide any significant benefit. The steadily increasing processing capacity of microcontrollers and the possibility of integrating sensors for biometric data acquisition into cards may allow smart cards to be used in new application areas.

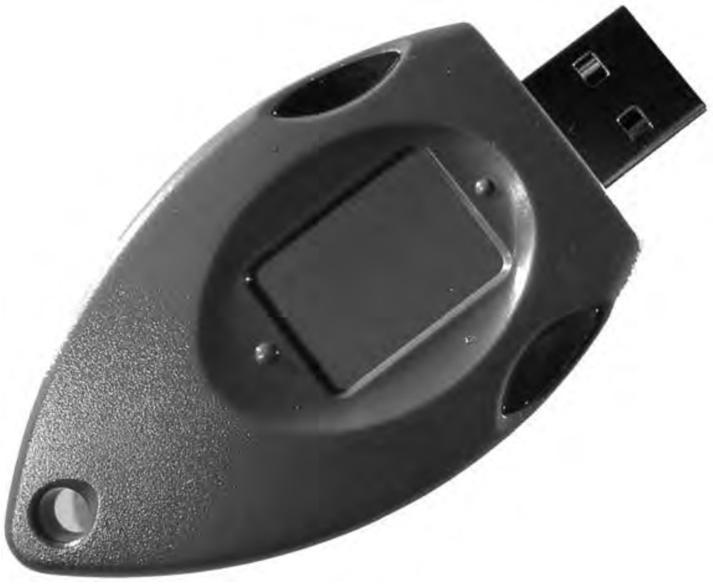


Figure 8.8 Photograph of a USB token with an integrated fingerprint sensor and smart card microcontroller (*Source:* Giesecke & Devrient)

8.1.2.2 Physiological features

Additional biometric features could easily be added to the features described below. However, we have limited our descriptions to the most important and most commonly used features in order to avoid getting bogged down in details.

Some physiological features that cannot be consciously altered also do not change much over time. For example, the characteristic patterns of fingerprints never change during a person's lifetime, nor do the patterns of the retinal blood vessels. The face is certainly an exception, since even though it basically does not change, it can be transformed to a large degree by a different haircut, growing or shaving a beard and the like. Basically, though, it is possible to say that biometric features based on adult physiology do not require ongoing adjustment of the reference pattern, since any changes are negligibly small or non-existent.

Facial features

Based on everyday experience, we can assume that the human face is suitable for use as a biometric feature. However, transforming this assumption into a technical implementation is fraught with difficulty. Faces can change greatly within a short time, and their appearance depends strongly on external factors such as eyeglasses, beards, make-up, illumination and viewing angle.

If a person's face is photographed using visible light and the information captured by the photograph is suitably processed, is will often be possible to make a decision about the identity of the person behind the face. The technical toolkit for this process includes very powerful computers, fuzzy logic and neural networks, which indicate the amount of effort that it entails. In addition, the stored image should be three-dimensional, or the person being examined should turn his or her face, to prevent the system from being deceived by holding a photograph in front of its sensor. In general, facial features may prove to be a very interesting subject for future biometric methods, but presently they cannot yield a high enough probability of accurate identification to allow them to be widely used.

Retinal features

Every human retina has its own unique pattern of blood vessels, with their branches and nodes. This pattern can be captured using a beam of infrared light directed through the pupil. The light reflected by the retina is collected by a CCD camera, which in turn sends the recorded image data to a computer for analysis.

Retinal imaging is one of the very best biometric methods, since it can be used to uniquely identify a person with a very high degree of probability. However, it is not readily accepted by users, since they must place their eyes very close to the scanner in order to be identified. This often results in a fear of infection and anxiety with regard to the infrared beam.

Iris features

The iris is a variable diaphragm that controls the amount of light reaching the retina. Like the retina, it is a biological feature that is unique to each individual. An iris scan can be performed at a greater distance than a retinal scan, since the measurement process is simpler. With this method, the iris (which is located at the front of the eye) is imaged by a CCD camera using visible light. The data evaluation is similar to that used for retinal images. Contact lenses can strongly influence the measurement results under certain conditions, and thus cause problems.

Hand geometry

Identification systems based on three-dimensional measurements of the hand, or parts or the hand, were used as early as the 1970s. These measurements can be based on features such as finger length, finger diameter and fingertip radius. Unique individual features can be determined using very few measurement points (e.g. five). The actual measurements can be made very simply using infrared LEDs and photodiodes, with the hand geometry being measured by recording which photodiodes are fully or partially blocked by the hand. Since only a few measurements are needed for identification, the procedure is sufficiently fast and uncomplicated for users. The user only has to place his or her hand in an instrument, which then performs the measurements.

Fingerprints

The best-known biometric identification method based on a physiological feature is without doubt fingerprinting. In the electronic version, it is naturally no longer necessary to coat the fingers with black ink and press them onto a piece of paper. Instead, a thumb or fingertip is placed against a transparent plate, and a camera mounted under the plate scans the skin surface without any contact. Alternatively, ultrasonic sensors or semiconductor-based capacitive sensors can be used.

The comparison with the reference pattern is usually based on the primary features of the classification scheme developed by Edward Richard Henry,[4] which are arches, loops and whorls. Information about the type, position and orientation of approximately 20 such features is stored, and this information is used to generate the reference pattern. These characteristic features are called the 'minutiae'.

Certain groups of users dislike this method, since fingerprints have been used for years as a tool for combating crime. Wounds on the fingertips can also make unambiguous personal identification difficult. It can also be difficult to identify persons who work a lot with their hands. Many systems have sensors for measuring the temperature of the finger or the pulse rate, in addition to the scanner. This is intended to prevent an amputated finger from being used for identification purposes.

Nevertheless, fingerprint systems are widely used, since they present relatively few problems in terms of technical difficulty and user acceptance. The time needed to sense the fingerprint and perform the subsequent test also lies within reasonable limits. The sensors used have a resolution of around 400 dpi.

Table 8.3 Comparison of biometric methods based on physiological features. The listed values are typical and can strongly vary among manufacturers. The volume of the reference data is also strongly dependent on preprocessing

	Test duration (seconds)	Amount of reference data (bytes)	Probability of false rejection (FRR), %	Probability of false acceptance (FAR), %
Facial image	—	—	≈2	≈1
Retinal image	1.5–7	40–80	0.005	10^{-9}
Hand geometry	1–2	10–30	0.8	0.8
Fingerprint	1.5–9	300–800	0.014	10^{-6}

8.1.2.3 Behavioral features

With many persons, biometric features based on behavior are not stable over time. One example is a signature, which can change considerably during the course of a person's life. However, it is rare for these changes to occur suddenly, and they are usually quite gradual and slow. Many systems that use biometric features therefore use adaptive methods that accept any changes in the feature that are detected during a correct identification as a new reference pattern, which is then stored in the smart card.

[4] Fingerprints were first used for identifying persons by the Bengal police in India around the end of the 19th century, under the leadership of Sir Edward Richard Henry. After returning to London, Edward Henry established the first British fingerprint collection in 1901, and he generated a classification method for fingerprint features that is still in use

Table 8.4 Comparison of biometric procedures based on behavioral features. The listed values are typical and can strongly vary among manufacturers

	Test duration (seconds)	Amount of reference data (bytes)	Probability of false rejection (FRR)	Probability of false acceptance (FAR)
Voice	5	100–1000	1 %	1 %
Dynamic signature	2–4	40–000	1 %	0.5 %

Typing rhythm

It has been determined that there are large differences in the manners in which different individuals type characters on a keyboard. These primarily relate to the pauses between individual letters. This can naturally be used as a biometric feature for identification. The procedure works by having the person to be identified type a prescribed character string (which is different for each test) on a keyboard. The computer to which the keyboard is attached evaluates the typing rhythm as the character string is typed. A text chosen by the user can also be used to evaluate the typing rhythm, but this requires more characters to be typed than with a prescribed text.

The primary advantage of this method is that it does not need any additional hardware, since in most cases a keyboard and computer are already available. Unfortunately, between 100 and 150 alphanumeric characters are needed for the test, and they must be typed using the 10-finger system. This is the main drawback of this method.

Vocal features

Like the face, a person's voice is characteristic of the person, so it can also be used for identification purposes. The person to be identified speaks one or more sentences into a microphone. These must be different for each session, since otherwise the system could be attacked very easily by playing back a previous identification session, which for example may have been recorded on magnetic tape. The waveforms of the spoken text are subjected to a Fourier analysis, which yields the characteristic frequency spectrum of the speaker. This is then compared with a reference value to determine whether the speaker's identity is genuine. The entire gamut of modern computational wizardry, such as fuzzy logic, neural networks and the like, is also employed with this method.

Of course, this method also has its shortcomings. A person's voice is very strongly influenced by his or her current bodily condition. Furthermore, all background noises must be reliably filtered out to make unambiguous spectral analysis possible in the first place. A different sentence must be spoken for each test to prevent recorded speech from being played back, which very much complicates the procedure and makes recognition more difficult. However, these technical difficulties are offset by good user acceptance, which makes this a very attractive biometric identification method.

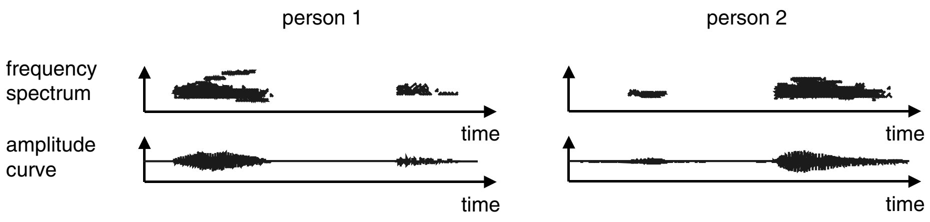


Figure 8.9 Amplitude waveform and time-dependent frequency spectrum of the name 'Wolfgang', spoken by two different people

Dynamic signature

The only identification method that is commonly used in everyday life is writing a signature. Due to its very individual character, a signature can also be used as a biometric feature. With a static method, the signature is evaluated after it has been written. With a dynamic method, by contrast, measurements are made while the signature is being written. The static method is only of theoretical interest, since it cannot distinguish a photocopied signature from a genuine one.

The parameters measured in the dynamic method may for example be the general form of the signature, the speed, acceleration and pressure of the pen on the writing surface, and the time required to write the signature. A special pen, or a special pad that can sense the parameters to be measured, can be used to make the measurements. Figure 8.10 shows an example of a possible arrangement in which an ordinary pen is used on a special pad, and Figures 8.11 through 8.14 show examples of measured signals that can be used as the basis for a biometric identification process. Pressure sensors are located at the intersections of the grid wires, and their signal amplitudes are transmitted to the computer via conditioning logic. The computer can then use various algorithms to process the measured data into a standardized format and compare the results with a stored reference pattern.

Using a dynamic signature for purposes of identification has the highest degree of acceptance of all personal identification methods, since signatures are used daily by everybody

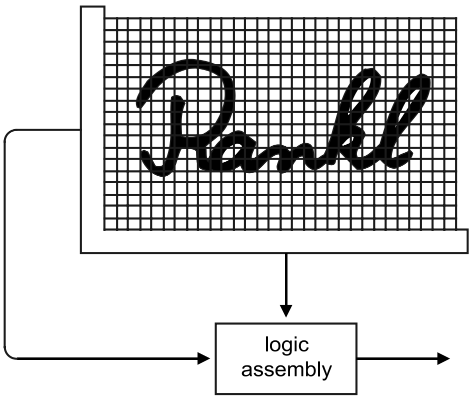


Figure 8.10 Sample measurement setup for testing a dynamically generated signature

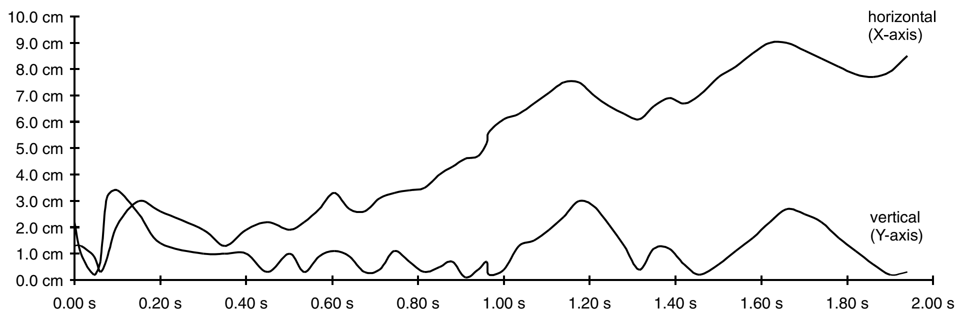


Figure 8.11 Horizontal and vertical pen position as function of time, for writing the word 'Rankl'

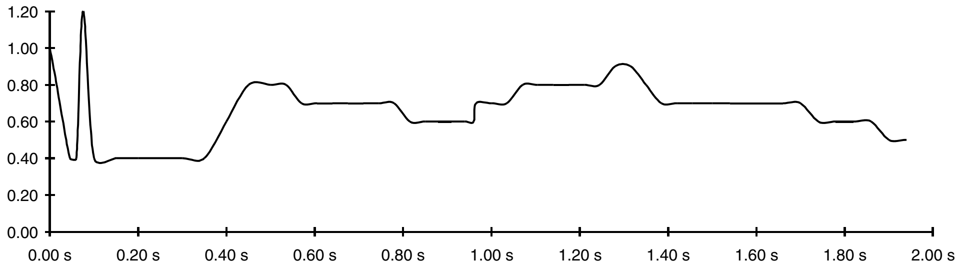


Figure 8.12 Pen pressure as function of time, for writing the word 'Rankl'

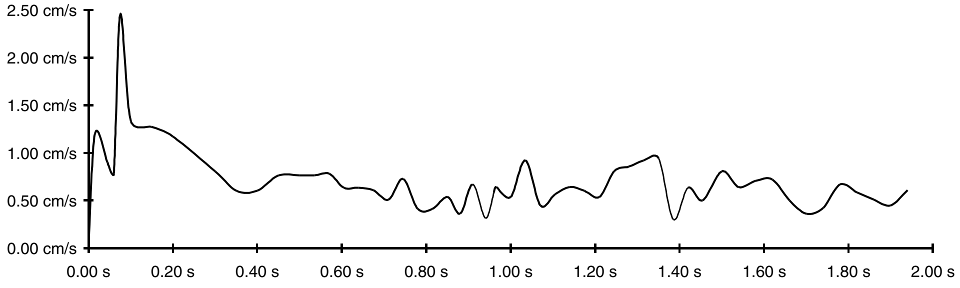


Figure 8.13 Pen speed as function of time, for writing the word 'Rankl'

in almost the same fashion. However, here the technical solutions are not simple, since signatures change over time and are never fully identical. You need only consider the difference in your signature if you write it while sitting or standing to appreciate the truth of this.

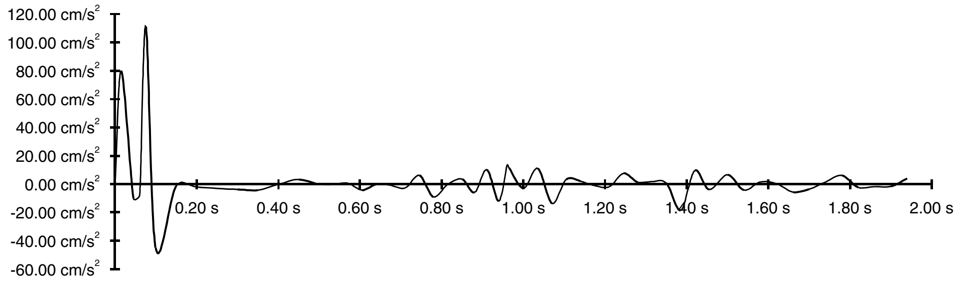


Figure 8.14 Pen acceleration as function of time, for writing the word 'Rankl'

8.2 SMART CARD SECURITY

The essential characteristic of a smart card is that it provides a secure environment for data and programs. If the amount of effort needed to read data from a smart card were not so large, it would essentially be nothing more than a diskette with a different interface.

It is naturally practically impossible to configure a complete system, or even a smart card, such that it has perfect security that is proof against everything and everybody. If the effort expended on the attack is raised to a high enough level, it is possible to gain access to any system or manipulate it. However, every potential attacker makes a conscious or unconscious cost/benefit analysis for himself and his targets. The rewards of breaking into a system must be worth the time, money and effort that the potential attacker must expend to attain his objective. Regardless of whether the reward is money or prestige within a peer group, if it is not worth the effort, no one will invest much energy in breaking a system or a smart card.

The security of a smart card is ensured by four components. The first component is the card body, in which the microcontroller is embedded. Many of the security features used for the card body are not only machine-readable, but can also be visually checked by humans. The techniques used for these features are not specific to smart cards, but are also used with other types of cards. The remaining components – the chip hardware, the operating system and the application – protect the data and programs in the smart card microcontroller.

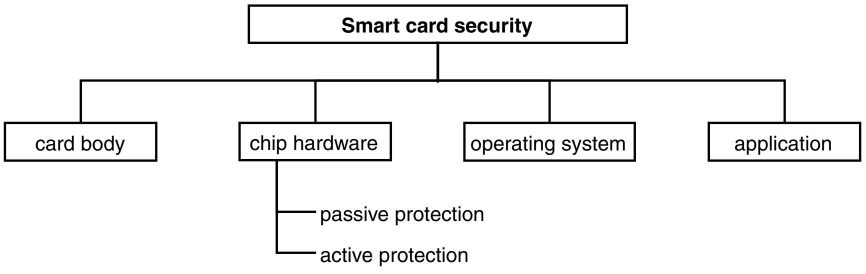


Figure 8.15 Classification of smart card security components

The security of a smart card is assured only when all of these components are present and their defense mechanisms are working properly. If the card is used exclusively within an environment where it is not subject to human verification, the card body component is not necessary. The three components that are independent of the card body, however, are indispensable for the physical and logical security of a smart card with respect to attacks. If any of these components fails, or if any one of them does not meet the applicable requirements, the smart card is no longer secure, since these components are coupled to each other in a logical AND relationship.

The useful life of a smart card is generally three years. The challenge to manufacturers of smart card microcontrollers and producers of smart card operating systems is to maintain a lead on all attackers that is at least this long. This allows the consequences of possible attacks to be minimized or avoided by employing suitable countermeasures. However, it is not always possible to maintain such a lead, which is why it is always important in the design and development of application architectures to pay strict attention to preventing a successful attack on a single smart card from compromising the entire system.

8.2.1 A classification of attacks and attackers

The primary problem faced by all information technology systems that are subject to attack is the ‘avalanche effect’, which is often seen after a successful attack. If a printer (for example) succeeds in counterfeiting good-quality bank notes in quantity, this is naturally a matter of concern for the affected national bank, but in practice it never leads to inflation of the national currency. In the first place, the counterfeiter would never be able to produce a sufficient number of false bank notes for this, and in the second place, it is very risky to bring a large number of counterfeit notes into circulation.

With electronic money, on the other hand, the situation is somewhat different. Since it consists of nothing more than immaterial information, in practice it is not possible to distinguish between an original and a copy. In addition, if a new counterfeiting method becomes known, this can lead to an avalanche effect when other people copy the original technique. This effect can be very clearly seen by considering the history of counterfeit prepaid telephone cards, which have been produced in very large numbers. Some network operators can still only defend themselves against this type of attack by restricting calling destinations for card phones.

If a design error or weakness in a major smart card system becomes known, it can be assumed that this information will be distributed over the entire world via the Internet within days or a few weeks. Very quickly, suitable software and hardware as necessary will be offered via the Internet, usually with complete documentation, making it easy for others to reproduce the original attack. This software is also usually provided in the form of source code, thus allowing it to be further refined by others, which also generally happens. This leads to a very rapid evolution of the hardware and software, which quickly becomes optimized to suit its intended purpose.

The following material represents an attempt to systematically classify possible types of attack and attackers. The emphasis naturally lies on the IT aspects of smart cards, rather than the security features of the card body that can be checked by humans. This classification allows potential attacks to be evaluated so that suitable measures can be taken against them. As is well known, it is easier to defend against a known type of attack than an unknown one.

We have based our classification of the different types of attack on the ISO 13491-1 standard, which describes the concepts, requirements and evaluation methods for cryptographically secure equipment in the banking sector.

Classification of attacks

There are several different approaches to the systematic classification of attacks on smart cards. For instance, in a security evaluation, all possible types of attack are grouped and formally described according to each phase of the card's life cycle [IC Protection 97, Isselhorst 97]. This yields multi-page lists that identify all conceivable attacks for each phase. The actual evaluation consists of examining each item on the list to see whether the system or smart card can defend against it. In analogy to fault tree analysis, it is also possible to generate an attack tree analysis [Schneier 99]. Such an analysis is of great benefit for detailed investigations and representing dependencies.

In this book, we use a different type of classification in order to present the subject in as realistic a manner as possible and illustrate the ping-pong game of attack and defense. In addition, our intention is to present a general summary of methods of attack and defense that is not specific to any particular system.

In principle, attacks on smart cards can be divided into three different types: attacks at the social level, attacks at the physical level and attacks at the logical level. Naturally, mixed types of attacks also occur in practice. For example, an attack at the physical level could prepare the way for a subsequent attack at the logical level, which is for example the case with differential fault analysis.

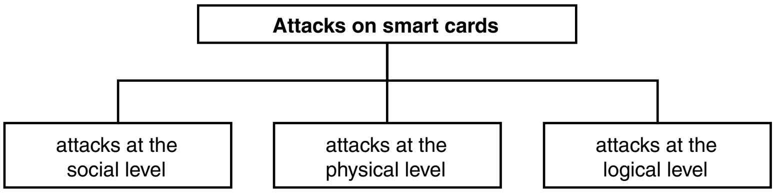

Attacks on smart cards
attacks at the social level
attacks at the physical level
attacks at the logical level

Figure 8.16 Classification scheme for attacks on smart cards

Attacks at the social level are attacks that are primarily directed against people that work with smart cards. These can be chip designers working for semiconductor manufacturers, software designers or, further on in the life cycle of the card, cardholders. These attacks can only partially be countered by technical measures. They must primarily be countered by organizational measures. Surreptitiously acquiring a PIN by watching it being keyed in can easily be prevented by providing visual screens on either side of the keypad. Attacks at the social level against smart card programmers are rendered pointless by making the procedures that are used public, as well as by having third parties evaluate the program code that they produce. In this case, security depends only on secret keys, and the knowledge possessed by software developers is of no use to an attacker.

Attacks on smart cards at the physical level usually require technical equipment, since it is necessary to obtain physical access to the smart card microcontroller hardware in one way or another. Such attacks can be either *static*, which means that no power is applied to the microcontroller, or *dynamic*, with the microcontroller operating. Static physical attacks impose no timing restrictions on the attacker, who can do his job at his own pace. With a dynamic attack, by contrast, the attacker must have access to sufficiently fast equipment for acquiring and evaluating data.

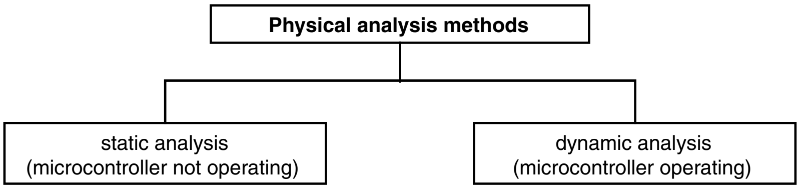

Physical analysis methods
static analysis (microcontroller not operating)
dynamic analysis (microcontroller operating)

Figure 8.17 Classification of physical methods that can be used to analyze smart card microcontrollers

Up to now, most known successful attacks on smart cards have been at the logical level. These attacks arise from pure mental reflection or computation. This category includes classical cryptanalysis, as well as attacks that exploit known faults in smart card operating systems and Trojan horses in the executable code of smart card applications.

Just as with the cryptanalysis of cryptographic protocols, these attacks can be divided into passive and active types. In a passive attack, the attacker analyzes the ciphertext or cryptographic protocol without modifying it, and may for example make measurements on the semiconductor device. In an active attack, by contrast, the attacker manipulates the data transmission process or the microcontroller.

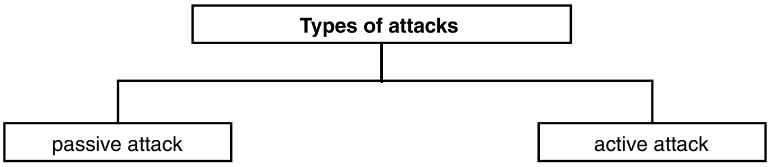


Figure 8.18 Classification of types of attacks on smart cards

The phases of the life cycle of a smart card as defined by the ISO 10202-1 standard [5] could be used with regard to the timing of possible attacks. However, this would result in verbose and long-winded descriptions, so for the sake of readability we have undertaken a simplification and classified the attacks into three intervals: (a) development, (b) production and (c) card usage.

Attacks during development relate to system design, chip development, operating system development and the generation of applications. The term ‘production’ is used in this context to refer in general to all processes used to make hardware. This covers the whole range from wafer fabrication by semiconductor manufacturers to card personalization and sending cards to users. Card usage refers to the stage in which the smart cards are in the field, which means when they are being used by cardholders.

[5] See also Chapter 10, ‘The Smart Card Life Cycle’

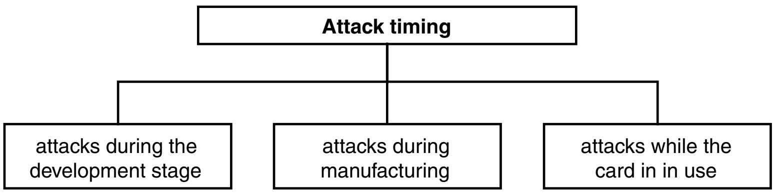

Attack timing
attacks during the development stage
attacks during manufacturing
attacks while the card in in use

Figure 8.19 Classification of the timing of possible attacks

The consequences of an attack and a classification of attackers

In order to be able to realistically estimate the strengths and weaknesses of attacks on the security of a smart card, it is important to first have at least a rough idea of the possible types of attackers. We can also use this information to help us devise defensive strategies and mechanisms.

As a rule, typical attackers have one of two basic motivations. The first motivation is simple greed, while the second motivation is the desire for fame and status within a particular 'scene'. These two motivations have different consequences for the system operator. An attacker who seeks a financial reward for his activities may take a certain risk by becoming a 'card issuer' in his own right,[6] or he may attempt to blackmail the system operator. Both approaches can be combated using the usual judicial measures. If details of the attack become public, the reputation of the smart card system will be damaged. The worst damage to the reputation of a system operator occurs when a large number of cardholders lose money as a result of an attack.

The reputation of a smart card system can be similarly damaged by an attack prompted by a compulsion to perform scientific research, rather than criminal tendencies. An attacker of this sort will consider his or her activities to be successful only if the results can be published in a suitable manner. The attacker is also under strong pressure to publish these discoveries as quickly as possible, since in this field, as is well known, being first is what counts. The end result is that the system operator, with little or no warning, is confronted with the publication of a detailed description of an attack on his system. Following this, the published attack is refined step by step by other interested parties and explained in terms that can be grasped by outsiders. The final blow comes when programs that carry out the attack in a fully automated manner are published on the Internet. In the spring of 1998, several GSM network operators found themselves confronted with a series of events similar to what has just been described. However, in this case the attack on the COMP128 cryptographic algorithm, which is used for A3/A8, did not have major negative effects on normal network operation.

There is a particularly significant aspect of this form of attack with regard to the attacker. He is regarded as the successful discoverer of a security leak, and thus as one of the 'good guys', and almost never need fear legal action as a result of his actions.

The quintessential conclusion that can be drawn from these scenarios is that it ultimately does not particularly matter to a system operator whether an attack comes from a 'good guy' or a 'bad guy'. In the case of a truly dangerous attack, the financial damage and the damage to the reputation of the system are most often rather large. In the worst case, the system must

[6] This is the usual approach for producing self-reloading telephone cards

be shut down, all cards must be blocked and new cards that are immune to the attack must be issued. With a large system having several million cards in use, such a process can take more than half a year.

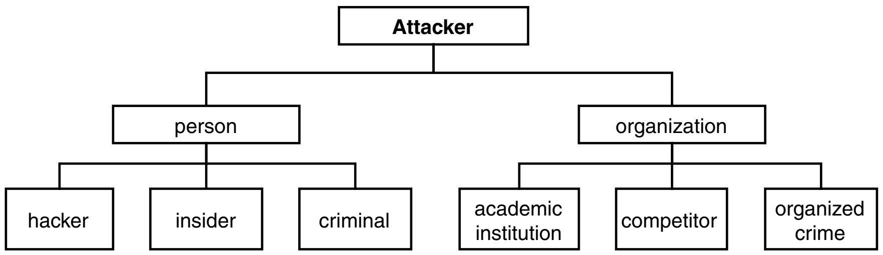


Figure 8.20 Classification of the possible types of attackers

The classification chart in Figure 8.20 shows a classification of attackers based on the previously described aspects and practical experience. All types of attackers can be equally dangerous to a smart card system, but they have different capabilities and options. A typical hacker, for instance, has a moderate amount of system knowledge, good creative ideas and usually a similar group of friends. He normally does not have an extensive amount of equipment, and his financial means are also limited. However, if he is competent and employs a suitable approach, he can certainly obtain access to a large amount of processing capacity, for example by means of an Internet campaign.

All insiders form a special class of attackers, under the assumption that they have very good knowledge of the system. They may have access to hardware and software components, and they may be aware of weaknesses in the system. As long as only single individuals are involved, they are equivalent to hackers in terms of their resources and options. However, since insiders are neither anonymous nor especially numerous, it is usually possible to identify the sources of their attacks.

The third class of persons who can be regarded as potential attackers is criminals. Although they usually do not have a high level of technical knowledge, they exhibit considerable energy when it comes to obtaining personal benefits (primarily financial) as a result of their activities.

A potential source of attack that cannot be ignored in practice consists of academic institutions, such as universities and technical institutes, including their students and professors. They do not necessarily have special knowledge of particular smart card microcontrollers or applications, but they do have a large amount of generally useful knowledge. In addition, they have access to a large pool of qualified and inexpensive labor in the form of students and graduates, as well as adequate technical equipment in their laboratories. Many of these institutions also house a plentiful amount of processing capacity and highly motivated people with an experimental bent.

A special class of attackers is formed by competitors. They normally have considerable technical knowledge, and some of them may have very sophisticated analytical equipment.

Organized criminal organizations naturally represent a completely different level of attacks on smart card systems. They have sufficient financial resources to acquire all the knowledge and tools necessary for a successful attack, either commercially or by illicit means.

Classification of the attractiveness of an attack

In order to allow effective perimeter defense measures to be put in place, the attractiveness of an attack should be evaluated for each of the potential weaknesses of the system. This can be done in an objective mathematical manner using value analysis, in order to compute a prioritized list of probable attack targets. The scheme that is presented here is simplified, but it still allows us to make a relatively good estimate of the attractiveness of the various types of attack, and thus the probable lines of attack. Naturally, an attacker would normally choose an attack that requires the least effort and expense. The six criteria listed in Table 8.5 will consciously or unconsciously influence the attacker's behavior.

Table 8.5 Criteria for determining the effort and cost required for an attack on the hardware or software of the security components, based on the prerequisites for an attack

Degree of attractiveness	Low	Medium	High
Level of knowledge and skills required	high	medium	low
Number of secrets required	many	moderate	few
Amount of time required	much	moderate	little
Acquisition of the necessary technical equipment (purchase or access)	difficult	moderate	easy
Access to the components to be attacked	difficult	moderate	easy
Value of the result (money or prestige)	low	medium	high

The lower the level of specific knowledge or skills required for an attack, the more attractive it is to an individual or an organization. Similarly, an attack that does not require the knowledge of any secrets is more attractive than one that requires many secrets to be known. This is not inconsistent with Kerckhoff's principle, which says that security should depend only on the key and not on the cryptographic algorithm itself, since Kerckhoff's principle does not mean that it is necessary to reveal everything about a system in order to make it secure. The presence of many secrets represents an enormous obstacle to mounting a successful attack.

Especially in the case of systematically searching for a key, the amount of time required plays an important role. The classic example is breaking a cryptographic algorithm using a brute-force attack that would require 10,000 years on average. No serious attack could be mounted on such a basis.

The attractiveness of an attack is equally dependent on the technical equipment required for the attack. This need not necessarily refer only to the purchase of equipment, since it may be sufficient to be able to rent the equipment or somehow acquire access to it. For example, a device that can generate and precisely position focused ion beams costs several hundred thousand euros, but such equipment can be rented by the day at research institutes, and some students can use this sort of equipment for free in their research work.

The availability of the components to be attacked also strongly influences the attractiveness of a particular type of attack. For instance, you could attack a card-based electronic purse system either at home, by analyzing your own personal card and its card-specific keys, or at the system level by trying to analyze a security module with its system-wide master keys. The problem with the latter approach is that access to the security module is protected by multiple security measures.

Incidentally, this is why smart cards for pay television are so strongly exposed to attack. An attacker can work undisturbed in his own living room, studying the communications and behavior of his smart card in order to try to duplicate them using a computer or a DIY electronic circuit, without being observed by anyone else and without any interference to his work. However, if he were to attempt to do the same thing with a smart card terminal in a supermarket, the cashier would immediately forbid any further experiments and thus interrupt his work. A good review of the subject of the security of electronic money with and without smart cards can be found in [BIS 96].

The final criterion, which is of decisive importance, is naturally the value of the result of the attacker's efforts. His efforts must be rewarded, either in a monetary form or in the form of enhanced prestige. From this, it can for instance be concluded that various field trials of electronic purses are only at risk of being attacked by hackers and academic groups. There are far too few locations where the cards can be used, and the businesses are mostly too simple (bakers, kiosks and the like), for any significant amount of money to be gained from an attack.

8.2.2 Attacks and defensive measures during development

A wide variety of security measures are employed starting with the development of microcontroller hardware and the software for smart card operating systems. Like quality, security is a factor that must be addressed from the very beginning of a development project; it cannot be designed into a product afterwards.

With regard to attacks in the development stage, it can generally be said that access to the facilities in question is very difficult and the required level of expertise is very high. The attractiveness of an attack is thus correspondingly reduced. Nevertheless, the potential danger of a successful attack at this stage is significant, since there are very extensive possibilities for manipulating the hardware and software.

8.2.2.1 Development of the smart card microcontroller

The development of the hardware for a smart card microcontroller takes many months. It involves only a few persons and takes place in controlled-access, supervised rooms within the facilities of a semiconductor manufacturer. The computer systems used to design the IC are usually part of an independent network that is isolated from the rest of the world. This makes it impossible to alter the chip design from outside, as well as preventing outsiders from obtaining information about the internal design of the chip.

A very extensive amount of insider knowledge is needed to undertake manipulations to a chip design that would weaken its security, so this type of attack is probably very unlikely. In addition, nowadays the designs and protection mechanisms of almost all smart card chips are evaluated by independent testing agencies, so an insider attack would not go undetected. However, it certainly could be advantageous to an attacker to know the exact design criteria and the arrangement of the functional elements on the chip, since he would then be aware of the protective mechanisms and sensors present in the chip and the scrambling of busses and memories. This knowledge could later be useful to an attacker with regard to physical chip analysis.

Protection: design criteria

There are a number of basic criteria that apply to the definition of the functions of a smart card microcontroller. One of these is that the mechanisms for protection against static and dynamic attacks must actually work. Sensors and other protective elements are of little use if they can be too easily circumvented or if they are not effective under certain conditions. An example is a sensor on the microcontroller chip that has such a large area that it can easily be destroyed by a needle, after which it no longer can fulfill its protective role.

One design criterion that differs from the criteria used for standard chips but is nonetheless very important is that absolutely no undocumented mechanisms or functions must be present in the chip ('that's not a bug, that's a feature'). Such undocumented features are usually not fully tested, since only a few people know about them, so they often exhibit various errors and weaknesses. Since they are not documented, they can be unintentionally overlooked during hardware evaluation and possibly be used later for attacks. The use of such undocumented features is thus strictly prohibited, even though they can often be very helpful for developers.

Protection: unique chip number

When the semiconductor hardware is being developed, all hardware security components must be first defined and then converted into hardware for the resulting microcontroller. One such component, in addition to sensors and protective coverings, is write once, read multiple (WORM) memory, which is also referred to as one-time programmable (OTP) memory. When the semiconductor chips are manufactured, a unique chip number is written to this memory. This means that each chip is different and can be uniquely traced, and smart cards can later be unambiguously identified within the system. In addition, chip numbers can also be used for the derivation of keys, and they make it possible to generate 'blacklists' that can be used to take suspect cards out of circulation.

It should not be overlooked that although these numbers cannot be altered in the original chips, they naturally provide no protection against imitation chips using freely programmable microcontrollers. This means that security measures cannot be based solely on the presence of a particular chip number in the WORM memory of a particular chip. Such a unique number can only be used as the basis for true cryptographic security mechanisms. For example, a chip number can be used for the derivation of secret keys, which are in turn used in a challenge–response authentication process.

8.2.2.2 Development of the smart card operating system

Software for smart cards is developed according to modern software development principles. Regardless of which life-cycle model is used (waterfall, spiral or whatever), certain general conditions must be observed.

The development computer always requires a separate, completely isolated network that does not allow any external access. The development tools, such as compilers and simulators, are software packages whose proper operation must be verified in dedicated tests. Sometimes two different compilers are used in order to be sure that the results are correct. Using software whose origins are not completely traceable is fundamentally prohibited, since such software would offer a possible means to manipulate development tools and consequently modify the programs to be generated.

Protection: development principles

Just as with hardware development, no undocumented features may be built into the software. For example, it would certainly be possible to convert the laborious black-box tests that are commonly used with smart cards into white-box tests by incorporating commands that could be used to read out arbitrary regions of memory. However, should one of these commands be inadvertently left in the operating system, it could be used to read out secret keys in real smart cards. In order to eliminate the possibility of such an attack, the creation of dump commands is undesirable, even through they can save valuable development time. However, deadline pressure and the steadily increasing complexity of smart card operating systems have led to a relaxation of this principle. In order to ensure that none of these development-state commands ever comes to be present in real smart cards in actual use, special tests for the absence of such commands are performed during smart card completion.

An additional principle is that programmers should never work alone on a project. This is already forbidden by considerations of software quality assurance, but the ‘four eyes’ principle must be observed for reasons of security as well. This effectively hinders attacks by insiders, since at least two developers must agree to work together on any attack. In addition, internal source code reviews are performed regularly, which assures the quality of the code and also supervises the development process.

Once the software development is finished, the entire source code and its functions are often inspected by an independent testing agency, as part of a software evaluation.[7] The main reason for performing this time-consuming and costly review is to check for software errors, but it also has the effect of making it impossible for a developer to hide a Trojan horse (for example) in the operating system. In practice, such items can be found only by reviewing the entire code, since an experienced programmer can certainly find means and techniques to hide a Trojan horse so that it cannot be found by a black-box test.

Protection: distributing knowledge

If several people work on a task, the result will be significantly more resistant to attack, due to the various opinions and experience of the people involved. The principle of distributing knowledge (shared secrets) is the opposite of the idea that ‘everybody knows everything about everything’. In the development of security components, complete knowledge of the component should fundamentally never be vested in a single individual, since that person would then be a target for an attack. As in many military realms, knowledge is divided over several individuals in the development stage, so that although it is possible for experts to discuss particular subjects, there is never any single person who knows everything.

[7] See also Section 9.3, ‘Evaluating and Testing Software’

A similar situation exists with regard to completing the smart card operating system, during which tables, program code and configuration data are loaded into the EEPROM. In addition to providing increased flexibility, this procedure also has a security aspect. This is because the chip manufacturer, who receives the final, assembled ROM code for producing the fabrication masks, does not have complete knowledge of the operating system. The parts of the operating system that are located in the EEPROM are unknown to the chip manufacturer, so he cannot discover the complete security mechanisms and functions of the operating system by analyzing the ROM code.

8.2.3 Attacks and defensive measures during production

Attacks during the production of chips or smart cards are typical insider attacks, since the production environments are closed. Access is strictly controlled, and every entry is logged. Nevertheless, security measures cannot be dispensed with in the production stage, since some technically very interesting and effective attacks can be carried out in this stage.

Protection: authentication during the finishing stage

Already at the wafer fabrication stage, smart card microcontrollers are individualized using chip numbers and protected using transport codes. With recent operating systems, the transport code is chip-specific, and an authentication is a mandatory requirement for each access in the finishing process. Although this increases costs and the amount of time required to finish the chips, and naturally requires a security module for every machine, it considerably increases security.

An obvious type of attack during finishing is to feed in dummy chips or dummy smart cards, which behave the same as genuine components but which, for example, include a ‘memory dump’ command. The earliest opportunity to replace a genuine chip with a dummy chip is of course after the wafer has been separated into individual dice. This type of attack can be illustrated using a smart card for digital signatures[8] as an example. In this case, the attacker replaces a genuine smart card with a dummy card at the initialization stage. This card is then initialized with genuine data and afterwards personalized. Since this smart card has all the functions of a real smart card, the process for generating the key for the asymmetric cryptographic algorithm will also be executed by the microcontroller. It obtains the data needed for this from the initialization and personalization data. After this, the attacker must manage to recover possession of this card, and then he can read the secret signature key from the card using his special dump command. Since the associated public key has been signed by the trust center and is thus confirmed to be genuine, the attacker now knows everything necessary to produce as many duplicate cards as he wishes, all of which will be seen as genuine.

This attack is unrealistic, since administrative measures are taken to prevent chips and smart cards from being taken into or removed from finishing stations. In addition, mandatory authentication between the smart card and the security module of the finishing machine before every finishing step makes it difficult to swap chips or cards.[9]

[8] See also Section 14.4, ‘Digital Signatures’
[9] An extensive and detailed description of the usual cryptographic process for the initialization and personalization

8.2.4 Attacks and defense measures while the card is in use

Access to the component to be attacked – the smart card – is usually much easier for the attacker after the smart card has been issued than in the previous phases of its life cycle. This is why the probability of attack is relatively high while the card is in use.

The idea of a self-destroying smart card microcontroller appears again and again in many publications, as a sort of panacea against all sorts of attacks. There are hardware security modules, such as those used for military applications, in which such mechanisms are sometimes employed, but such a defensive measure is not possible in smart cards for a number of reasons. First of all, in the absence of external power a smart card has no way to recognize a potential attack, and there is no possibility of any sort of active defense mechanism, since the smart card does not have any reserve source of energy. Besides this, for purely legal reasons it would probably not be possible to impose true self-destruction capability on cardholders. Who would be responsible for the loss or damage that might occur under unfavorable circumstances simply because a smart card has incorrectly destroyed itself? In addition, true self-destruction is not at all necessary, since in almost all cases it is sufficient to erase the secret keys stored in the card.

There is yet another aspect to this subject, which relates to erasing keys or blocking smart cards. It is very difficult for a smart card to even recognize that it is being attacked. There is simply not any sensor that can report 'Attack! Erase everything!' Too low a voltage or too high a clock rate could be a sign of an attack, but these situations also occur in normal operation due to unfavorable ambient conditions. Dirty or corroded contacts have high contact resistance and thus cause the operating voltage to be lower than normal. An excessive clock rate can be present in a smart card terminal that is intended to be used with cards that work at high clock rates. Since recognizing an actual attack is so difficult, and usually not even possible, automatic mechanisms for blocking the card or erasing the keys are usually not used.

In the following section, some types of attack that can be considered to be nearly 'classic' are described and explained. The descriptions of the attacks can be said to represent the 'state of the art'. They are intended primarily to provide people who are inexperienced in the area of smart card security with a reasonably solid basic understanding, in order to prevent mechanisms that are already known to be vulnerable from being reused out of simple ignorance. These attacks can be foiled by the defensive measures described below, which in turn can be countered by slightly modified attack scenarios. This leads to the well-known cat and mouse game of measures and countermeasures for attacks and defenses.

The scenarios presented here do not form an invitation to break the security of smart card systems, since without exception they are both known and published [Kommerling 99]. They do not represent any serious threat to the security of any contemporary smart card system, since they have long since been dealt with by suitable protective measures. However, a few years ago it would have been possible to achieve a certain amount of success using such scenarios.

The attacks are divided into those that are directed against the chip hardware and those in which an attempt is made to break the smart card system at the logical level. The physical attacks and analysis methods can also be subdivided into static and dynamic types. In a static analysis, the chip is not operating, but it may be electrically powered. In a dynamic analysis, which is much more difficult to perform, the chip operates with its full range of functions during the analysis.

of smart cards can be found in Section 10.4, 'Phase 3 of the Life Cycle in Detail'

Table 8.6 Summary of typical attacks affecting systems using smart cards, in order of the date when they first became known. The listed attacks and associated primary countermeasures are described in greater detail in the text

Known since	Attack	Brief desription
Before 1990	Tapping data communications	Using wires attached to the module, it is possible to tap data transmissions between the terminal and the card. The countermeasure was the introduction of secure messaging.
≈1990	Dissolving the passivation layer	Dissolving the passivation layer on top of the microcontroller is the prerequisite for physical access to the components on the microcontroller die. The countermeasure for this was the introduction of passivation detectors in microcontrollers.
≈1990	Manipulation of data communications	By electrically insulating the contact surfaces of the module and suitably attaching wires to the module, it is possible to manipulate data tranfers between the terminal and the card as desired. The countermeasure was the introduction of secure messaging.
≈1991	Erasing the EEPROM using UV light	By erasing the EEPROM using UV light, it is posssible to do things such as resetting counters to their initial values. The countermeasure for this was the introduction of light sensors in microcontrollers.
≈1991	Substitute circuits for memory cards	Substitute circuits for memory cards can be used to emulate the functions of the memory card and the secret authentication feature. The countermeasure to this was the introduction of challenge–response authentication for memory cards.
≈1992	Switching off the supply voltage	Writing the retry counter can be prevented by switching off the supply voltage during PIN testing. The countermeasure to this was to increment the retry counter before performing the PIN test, as a precautionary measure.
≈1993	Stopping the clock	By stopping the clock and analyzing the RAM using an electron-beam tester, conclusions about the content of the RAM can be drawn. The countermeasure to this was the introduction of underfrequency detectors in microcontrollers.
≈1993	Manipulating the microcontroller with a laser cutter	The components on the microcontroller die can be manipulated using a laser cutter. The countermeasure to this was the introduction of protective cover layers on microcontrollers.
1995	Timing attack	Due to ignorance, a dependence between the key value and the processing time was created in the implementations of many cryptoalgorithms. This can be used to help determine the secret key. The countermeasure to this was the implementation of noise-free cryptographic algorithms.

Table 8.6 (*Cont.*)

Known since	Attack	Brief desription
≈1995	Tapping the bus using microprobes	The busses on the microcontroller die can be tapped using microprobes. The countermeasure to this was scrambling the busses on the microcontroller dies.
1996	DFA	Secret keys for cryptographic algorithms can be deternined by selectively introducing scattered computation errors in the processor. The countermeasures to this were introducing glitch detectors on microcontroller dies and using suitable preventive measures in cryptographic algorithms.
≈1996	Manipulating the microcontroller using FIB	The components on the microcontroller die can be manipulated using FIB. The countermeasure to this was the introduction of protective layers on top of the microcontrollers.
1997	Exhaustive key search with DES	Using powerful computers or networks of computers, DES keys can be computed within a few hours using a brute-force attack. The countermeasure to this was the use of triple DES.
1997	Statistical distribution of PIN codes	The generation of PIN codes for the German Eurocheque-card system did not have a uniform statistical distribution, with the result that some PIN values were significantly more common than others. The countermeasure to this was using an improved PIN generation algorithm.
1998	SPA/DPA	The data being processed can be determined from the current consumption of the processor. The countermeasures to this were the introduction of randomly driven delays in the processor, using processors with constant current consumption and a large number of precautionary measures in the microcontroller software.
1998	COMP 128	Due to a design weakness in the COMP 128 authentication algorithm, which is used by several GSM network operators, it is possible to determine the secret keys using a brute-force attack. The countermeasure to this was using a different authentication algorithm and limiting the number of authentications.
1998	Disturbing the processor	By disturbing the processor (e.g. using intense flashes of light), it is possible to interfere with its operation at critical points while it is processing the machine code. The countermeasures to this were using suitable detectors in microcontrollers along with a large number of precautionary measures in the software.

8.2.4.1 Attacks at the physical level

Manipulations at the semiconductor level require a large amount of technical effort. Depending on the attack scenario, the equipment required may include a microscope, a laser cutter, micromanipulators, focused ion beams, chemical etching equipment and very fast computers for analyzing, logging and evaluating the electrical processes in the chip. This equipment and the knowledge of how to use it are available to only a few specialists and organizations, which strongly reduces the probability of an attack at the physical level. Nevertheless, a card or semiconductor manufacturer must assume that a potential attacker could employ the devices and equipment necessary for such an attack, which means that suitable protection must be built into the hardware.

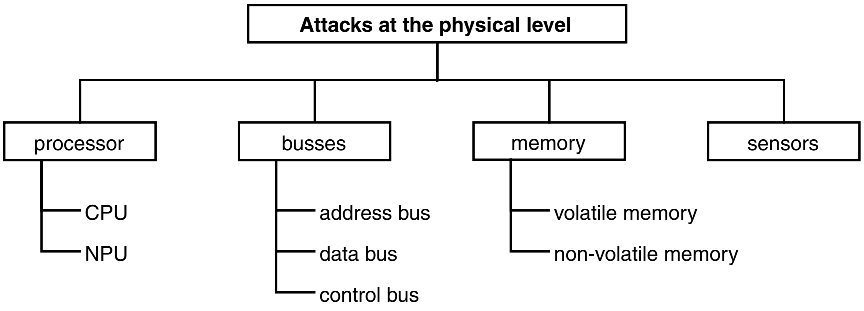


Figure 8.21 Classification of the points of attack on a smart card microcontroller at the physical level

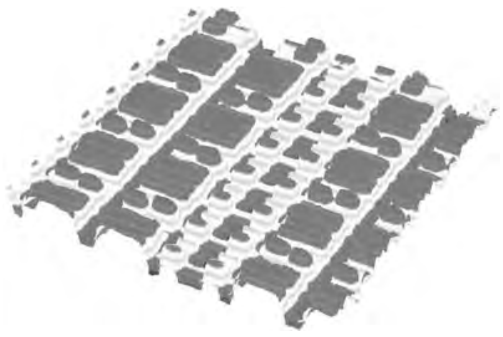


Figure 8.22 Graphic representation of the surface profile of a smart card microcontroller, as measured using an atomic force microscope. The maximum surface relief in this illustration is only 2.3 µm (*Source:* Giesecke & Devrient)

In order to conduct an attack at the physical level, a few preliminary steps are necessary. The first thing that has to be done is to remove the module from the card, which can easily be done using a sharp knife. After this, the epoxy resin must be removed from the chip. Anderson and Kuhne [Anderson 96b] used fuming nitric acid for this with an infrared lamp as a heat source, followed by an acetone rinse to clean the chip. After this, the semiconductor chip is free and still fully operational. Many people think that the chip now lies unprotected before them and only has to be ‘read out’, but this is by no means so. An attacker still has to work through a manifold of security measures before he can gain access to the secrets.

The protective measures in the hardware can be divided into passive and active components. The passive components are based directly on the techniques used in semiconductor manufacturing. They include all processes and options that can be used to protect the memory region and the other functional parts of the microcontroller against various types of analysis.

There is a full spectrum of active components available on a silicon chip to complement the passive possibilities offered by the semiconductor technology. Active protection means the integration of various types of sensors into the silicon crystal. These sensors are queried and evaluated by the smart card software as needed. This is naturally only possible when the chip is fully powered and operational. A chip without electrical power cannot measure any sensor signals, let alone evaluate them. Sometimes the boundary between useful protective components and technical gadgetry is particularly narrow where sensors are concerned. A light-sensitive sensor that is supposed to prevent optical analysis of the memory will not respond if the chip is located on the object carrier of an optical microscope without power or a clock signal. In addition, it is very easy to visually identify such a sensor on the chip surface and cover it with a drop of black ink, so its protective function can easily be neutralized even when the chip is operating. However, this can be countered by distributing a large number of light sensors over the entire chip.

Long-term functional security is also an important consideration. For example, a temperature sensor that causes the smart card software to erase the entire EEPROM in response to a brief but non-damaging overheating of the chip makes absolutely no contribution to increased functional security or security against an attack. Consequently, most smart card microcontrollers employ only a few sensors.

In the following descriptions, we explain the protective mechanisms of smart card microcontrollers that are the most important and the most often used in practice.

Static analysis of smart card microcontrollers

Protection: semiconductor technology

The dimensions of structures on the chip (track widths, transistor sizes and so on) approach the limit of what is currently technically possible. The usual structural widths lie in the range of 0.35 μm to 0.13 μm, which in itself is no longer technically remarkable. However, the transistor density on the silicon belongs to the highest level that can currently be achieved using standard lithographic fabrication processes. These very fine structures alone make it nearly impossible to extract any information from the chip using analytic procedures, for which reason semiconductor technologies with structure sizes of around of 1 μm are currently considered to be secure. This dimension is sure to be reduced in the future.

Protection: chip design

‘Standard cells’ are frequently used in designing semiconductor integrated circuits. They can contain the core elements of a processor or a particular type of memory. The advantage of

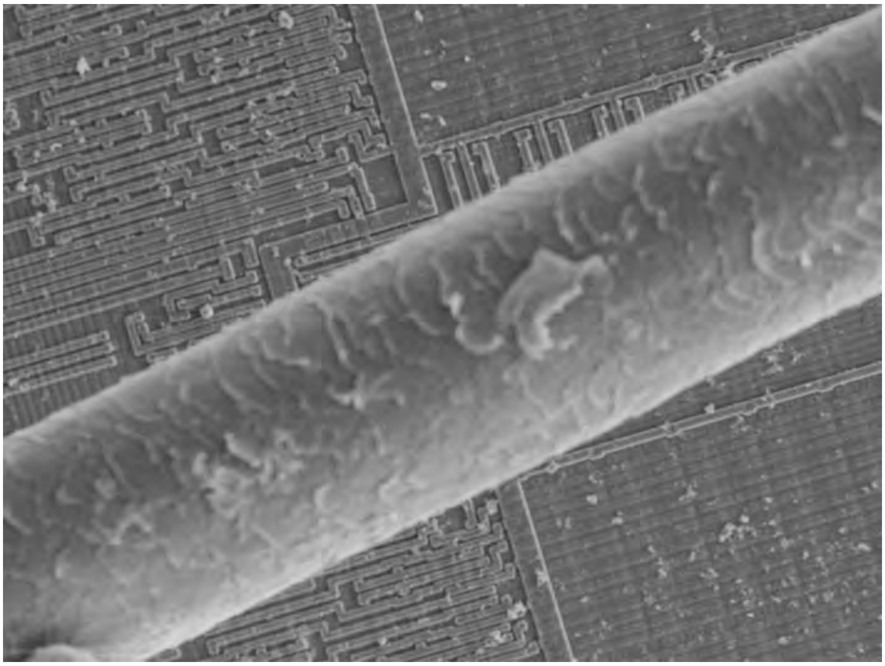


Figure 8.23 Photograph of a human hair in comparison with semiconductor structures of a smart card microcontroller, magnified 1000× (*Source:* Giesecke & Devrient)

using standard cells is that it allows a semiconductor manufacturer to quickly produce a variety of different types of chips with a high level of quality. This technique, which has been developed for mass-produced components where security is not an issue, is not allowed to be used for smart card microcontrollers. This is because the designs and functions of standard cells are known, and their use would thus provide a potential attacker with too much information and thus considerably simplify his task. The functional elements of smart card microcontrollers are developed especially for this application and are not used for any other purpose.

Protection: dummy structures

Using dummy structures on the chip is a measure that is the subject of frequently controversial discussions among experts. Dummy structures are elements of the semiconductor that do not have any actual function, but instead are intended to confuse and mislead an attacker. The associated security is based purely on keeping the existence and locations of such structures secret. Dummy structures can also be monitored, so that any changes to them can be detected and can cause the chip to switch off. The main disadvantage of dummy structures is the additional room that they occupy on the chip.

Protection: chip busses

All internal busses of the chip, which connect the processor to the three different types of memory (ROM, EEPROM and RAM), are not brought out from the chip. This means that it is not possible to directly make connections to these busses. It is thus not possible for an attacker to tap into the address, data or control bus of the microcontroller or influence the bus signals in order to read out the memory contents. The busses are usually fabricated in the lower layers of the semiconductor device in order to make it difficult to make direct contact with them from the surface. In addition, the busses on the chip are scrambled in a static, chip-specific or session-specific manner, so the functions of the individual bus lines cannot be recognized from the outside. There are even smart card microcontrollers whose bus scrambling is continuously modified during a session.

Protection: memory design

The storage medium used for most programs is the ROM. The contents of the type of ROM commonly used in the industry can be read bit by bit using an optical microscope. It would not be particularly difficult to assemble these bits into bytes and then arrange these bytes to obtain the complete ROM code. In order to prevent exactly this type of analysis, the ROM is not located in the top level of the chip, which is the most easily accessible layer. It is instead located in the lower layers of the silicon. This impedes an optical analysis.

However, if the chip were to be glued to a carrier upside down and the rear surface were then ground off, it would be possible to read the contents of the ROM. For this reason, only ion-implanted ROM is used in smart card microcontrollers, since the contents of such a ROM cannot be seen using either visible or ultraviolet light. This also largely protects against 'selective etching', which is a process that can be used to attempt to etch the semiconductor in order to make the contents of the ROM visible.

Protection: protective layers (shields)

Analyzing the electrical potentials on the surface of the chip while it is operating represents a threat. With a suitably high scanning resolution, this technique can be used to measure charge potentials (voltages) on very small regions of the crystal. With this information, it is possible to draw conclusions about the contents of the RAM while the chip is operating. This analysis can be very effectively prevented by placing current-carrying metalization layers on top of the memory region or the entire chip. If these metalization layers are removed by chemical etching, the chip will no longer operate properly, since they are needed to distribute the electrical power the chip needs in order to function. Frequently, several protective layers are arranged on top of each other and continuously monitored for integrity.

In addition, the chip can be fabricated with meandering current-carrying structures on top of the entire chip or on top of a region that needs special protection, such as an underfrequency detector. These structures can be easily monitored using resistance or capacitance measurements, or they can be incorporated into the circuitry of the chip such that it immediately stops working if they are damaged. Security can be further increased by modifying the connections or interconnections of these meandering structures during a session. This provides protection against using a focused ion beam (FIB) tool to bridge the meanders.

It is also conceivable to use opaque protective layers whose integrity is continuously monitored by phototransistors, which are easily implemented in semiconductor devices. If such a layer were removed, this would immediately be detected and the chip could refuse to operate any further.

Attack and defense: reading out the volatile memory

As is well known, a RAM loses its data contents when its power supply is cut off. However, this does not occur if the memory cells are cooled to a temperature of –60° C. Also, the content of the RAM is not necessarily lost if the stored data remain unchanged for a long time. The background of this effect is described in a paper by Peter Gutmann on the subject of securely erasing memory media [Gutmann 96]. Consequently, secret keys are not held in RAM any longer than is absolutely necessary, following which they are immediately erased or overwritten with other values. This minimizes the risk that traces of secret keys may be left in the RAM cells and weakens attacks based on fixing the RAM contents by freezing or burning.

Reading out RAM cells is very difficult, since it requires detecting the switching states of the transistors involved. However, it is certainly possible to extract stored data from RAM cells using sophisticated electron microscopes and special contrast-enhancement methods. A prerequisite for this is removing the passivation layer and the metalization layers underneath the passivation layer, which protect the RAM against exactly this type of attack. Removing the metalization layers unavoidably causes the RAM cells to be destroyed, since part of their functionality is incorporated in these layers.

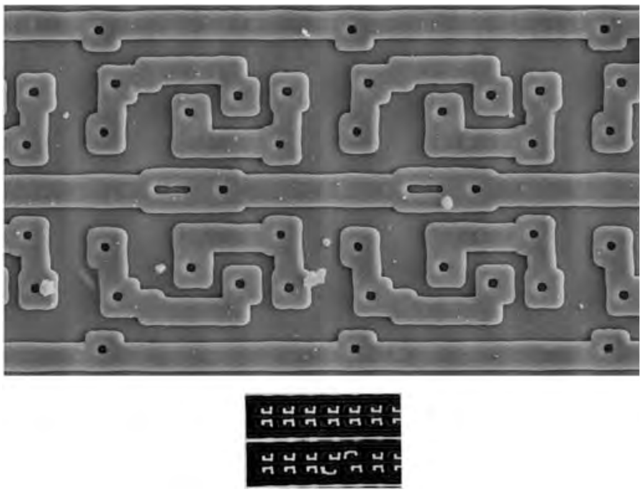


Figure 8.24 Photograph of several RAM cells magnified 3000×, without protection by means of additional metalization layers. The lower picture shows the electrical potentials of the same RAM cells as measured using an electron beam tester with the chip in operation. The distribution of zeros and ones over the RAM can be clearly recognized (*Source:* Giesecke & Devrient)

Protection: memory scrambling

Scrambling the memory on the microcontroller chip, which is similar to the well-established practice of scrambling the busses, is being used increasingly often. The security of this technique is based on the secrecy of the scrambling scheme for the memory cells. Memory scrambling is easily implemented and does not require much additional space on the chip. Without the relevant scrambling information, it is extremely difficult for an attacker to discover how the memory cells are actually addressed.

The EEPROM can also be scrambled using software. However, this requires complicated programming, and all write accesses must be protected by making them atomic operations, since otherwise the system would be very vulnerable to the sudden removal of the supply voltage. Software memory scrambling does, however, have the advantage that it can be made chip-specific and even dynamic, so that the memory contents can be redistributed within the memory in the course of a session.

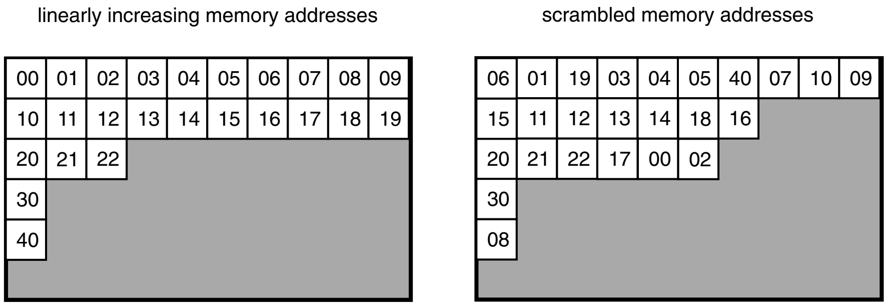


Figure 8.25 Comparison of a conventional semiconductor memory and a scrambled memory used in a smart card microcontroller

Protection: memory encryption

Besides scrambling the data in the memory, modern smart card microcontrollers also provide batch- or chip-specific encryption of the memory and some of the processor registers. This involves decrypting or encrypting the corresponding data in real time when they are read or written. Besides the key, with some types of chips the memory address can also be incorporated in the encryption and decryption process, so that identical data in different memory locations have different values after having been encrypted. Particularly for RAM regions, session-specific keys can also be used.

With such encryption, if data are read from the memory by means of a successful attack, it will still be necessary to have the secret key in order to recover the plaintext values. This considerably increases the amount of effort that must be expended by an attacker, since he must either know where the key is stored or systematically read out all of the data present in the chip.

Attack and defense: encrypted storage of secret data

This example of an attack at the physical level is actually a textbook example of a defensive measure that is not particularly effective. The basic idea of this defense is that if it is possible to read out the EEPROM of a smart card, an attacker should at least be prevented from thereby learning secret data, such as a PIN or key. Naturally, there are preventive measures that can be taken. For example, the PIN can be simply encrypted using a one-way function, with the result being stored in the EEPROM as the reference value for PIN comparison. A card-specific key could be used for the one-way function, so that the reference data for two identical PINs would be different in two different cards.

If the reference value is now read from the EEPROM, it would seem to be impossible to derive the PIN from this value. However, a clever attacker would not even take such an approach. If normal PINs consisting of four decimal digits are used, the number space of the PINs has a lower boundary of "0000" and an upper boundary of "9999". This means that the number of possible PINs is exactly 10,000. If the attacker can read the entire memory of the smart card, he can also read the one-way function and its associated card-specific key. With this information, he can start encrypting all possible PINs using the one-way function. After an average of 5000 attempts, he will have obtained a result that matches the reference value in the smart card, which means he knows the PIN. As can be seen, in this case using a one-way function to store the PIN does not provide any significant benefit. The reason this mechanism is frequently used in practice is because it requires much more effort to read a large amount of data from a memory than only a few bytes for the PIN. Consequently, the keys in a smart card are frequently encrypted using a card-specific key before being stored in the card.

Dynamic analyses of smart card microcontrollers

Protection: monitoring the passivation layer

A passivation layer is placed on top of the microcontroller in the silicon at the conclusion of the fabrication process. This layer impedes oxidation (due to atmospheric oxygen) and other chemical processes at the surface of the chip. The passivation layer must always be removed before any sort of manipulation of the chip can be performed. It should be borne in mind that although it is possible to chemically remove the passivation layer, the chip is then exposed to a major risk of oxidation, which can destroy it relatively quickly. A sensor circuit can employ resistance or capacitance measurements to determine whether the passivation layer is still present. If it is missing or damaged, this can either trigger an interrupt to the chip software or cause the complete hardware of the chip to be shut down, which reliably prevents any sort of dynamic analysis.

Protection: voltage monitoring

A voltage monitor is present in every smart card microcontroller. It provides a well-defined shutdown of the IC if the supply voltage exceeds its allowed lower or upper limits. This gives the software the assurance that it is not possible to operate the chip in marginal regions in which the chip may not function properly. Without such a voltage monitor, it would be possible for the program counter to become unstable when the chip was operated in a marginal region,

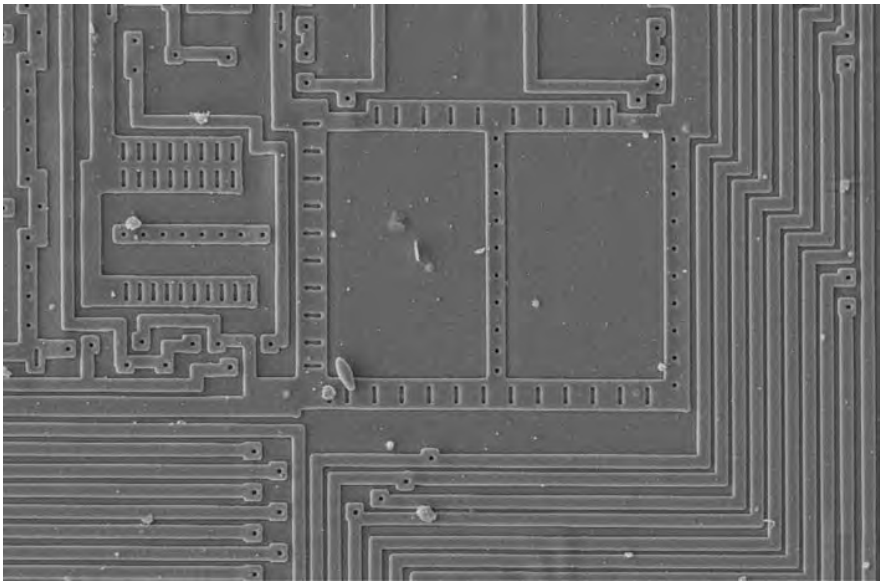


Figure 8.26 Photograph of a passivation layer monitor at 1000× magnification. The passivation layer detector consists essentially of the two rectangular semiconductor elements, and it uses capacitance measurements (*Source:* Giesecke & Devrient)

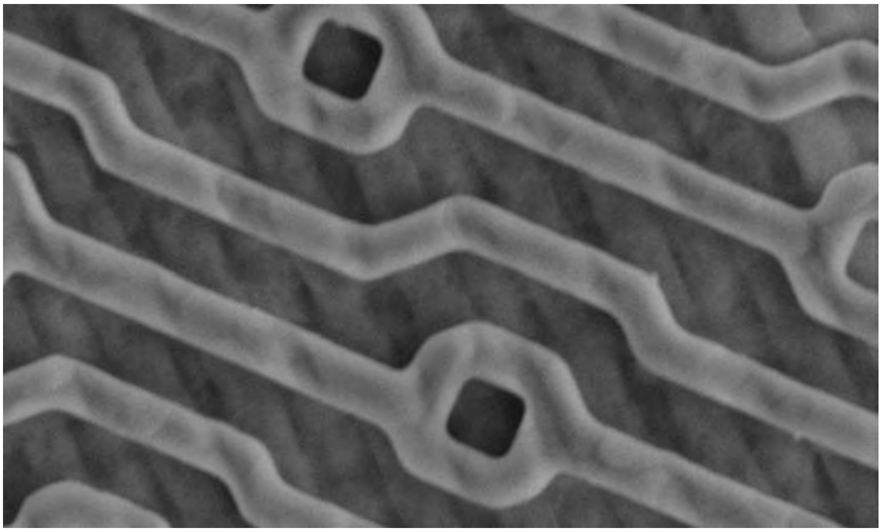


Figure 8.27 A modern passivation layer monitor at 8000× magnification. The track separation is 4 μm, and the working principle of this detector is based on resistance measurements (*Source:* Giesecke & Devrient)

leading to uncontrolled program jumps or plain computation errors in the processor. Such faulty behavior could be used to determine secret keys by using the technique of differential fault analysis (DFA), which is described elsewhere in this book.

For this reason, it is important for the voltage monitor to also be able to detect very brief voltage peaks or dropouts, in order to protect against typical attacks involving the intentional introduction of processor errors. As an example, in the case of a smart card intended to be used with a supply voltage of 3–5 V, the usual shutdown thresholds are 2.3 V and 6.3 V. These value lie slightly outside the range of 2.7–5.5 V specified by various standards, in order to allow for tolerances in sensor calibration during semiconductor fabrication.

Voltage monitoring in particular is highly important for the security of the microcontroller. A conceivable method of attack would be to first use a focused ion beam (FIB) or similar tool to disable the relevant detectors and then start the actual attack. For this reason, the components that are vital to the security of the microcontroller are often specially protected so that manipulation can be detected, causing the smart card to automatically deactivate itself.

Another type of sensor that is partly based on the voltage detector is the power-on detector. This detector, which is also present in all chips, recognizes a power-on condition independently of the external reset signal and ensures that the chip is always placed in a defined initial state when power is first applied. The reasons for doing this are similar to those for using voltage monitoring.

Protection: frequency monitoring

A smart card is always driven by an external clock, so its processing speed is completely determined outside the card. This means that, at least in theory, it is possible to operate the microcontroller in single-step mode. This would provide outstanding opportunities for analyzing the microcontroller, in particular by measuring its current consumption while it is operating (power analysis) and measuring electrical potentials on the surface of the chip. In order to prevent such attacks, a functional component for detecting underfrequency and overfrequency conditions is built into the chip. This eliminates the possibility of reducing the clock rate to unallowable levels. The minimum clock rate stated in most specifications is 1 MHz. However, for technical reasons the underfrequency detector has a wide tolerance range, so the chip usually stops working at around 500 kHz. This ensures that the chip will always work at the minimum specified clock rate of 1 MHz. The upper frequency limit is 5 MHz in most specifications, and typical overfrequency detectors disable the chip at a frequency of approximately 7 MHz. Modern microcontroller hardware is often built such that the chip cannot be used if the clock rate is too high.

In order to protect the microcontroller against the dangers of single-step operation, it is naturally necessary to secure the underfrequency detector with protective layers, so that any attempt to tamper with the detector will be recognized.

Protection: temperature monitoring

A temperature sensor is used in some types of chips, but the benefit of such a sensor is debatable. The chip will not be damaged if the temperature briefly exceeds the specified operating range, and this does not in itself represent an attack. Shutting down the chip in this marginal situation, however, could lead to an artificially increased failure rate without providing the operator of the smart card system with any additional security.

Protection: bus scrambling

In many smart card microcontrollers, the internal busses that drive the memory are scrambled. This means that the individual bus lines are not laid out next to each other in increasing or decreasing order, but are instead arranged randomly next to each other and ‘swapped’ several times, or even arranged in several layers on top of each other. This represents an additional hurdle for a potential attacker, who does not know which bus line is associated with which address bit or function.

Scrambling the bus lines was originally introduced only in a static version, with the same scrambling scheme used on every chip. With static scrambling, it would probably not be all that difficult for an attacker to discover the scrambling scheme over a moderate length of time, and thus be able to take it into account when tapping the busses.

The security provided by this technique can be improved by using chip-specific scrambling. This is naturally not achieved by using a different set of exposure masks for the busses of each chip, since this is currently either not technically possible or affordable. Instead, scrambling is performed by randomizer circuits located just ahead of the memory. These can be driven by the chip serial number, for example. This technique is not difficult in terms of semiconductor technology, and it makes life considerably more difficult for someone who tries to tap the bus. Using variable input values for the randomizer makes it possible to achieve chip-specific and session-specific scrambling.

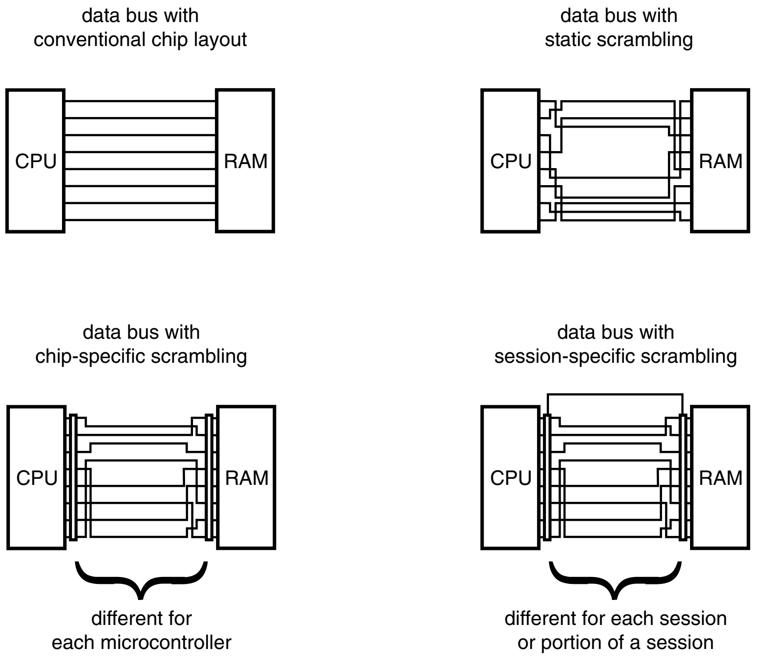


Figure 8.28 Bus scrambling in a smart card microcontroller, illustrated using an 8-bit data bus between the CPU and the RAM. The data bus lines shown here represent information flows rather than electrical leads. The encryption units are shown as separate components for the sake of clarity, but they are actually intermingled with the rest of the components in such a manner that they cannot be recognized as separate components, thus making them immune to attack

Protection: irreversible switching from the test mode to the user mode

All microcontrollers have a test mode that is used for verifying the chips during the fabrication process, and for executing internal test programs while the semiconductors are still in the wafer or after they have been packaged in modules by the manufacturer. The test mode allows types of access to the memory that are strictly forbidden when the chips are later in actual use. However, for technical production reasons, it is an unavoidable requirement to be able to read data from the EEPROM in this mode.

The change from the test mode to the user mode must be irreversible. This can be realized by using a polysilicon fuse on the chip. In this case, a voltage is applied to a test point on the chip that is provided for this purpose, and this voltage causes the fuse to melt through. The chip is thus switched into the user mode using hardware. Normally, this cannot be reversed. However, a fuse is by its nature a relatively large structure on the surface of the chip. It is conceivable that the fuse could be mechanically bridged after the removing the part of the passivation layer that covers the fuse. This would put the microcontroller back into the test mode, and the memory could be read out using the extended access options available in this mode. If the complete content of the memory is known, it is easy to clone the smart card that has been read out.

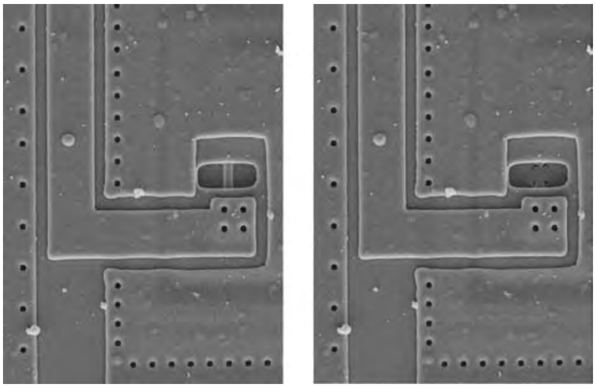


Figure 8.29 Photograph of a polysilicon fuse magnified 2000×. The picture on the left shows a fuse that is still intact, while that on the right shows a blown fuse (*Source:* Giesecke & Devrient)

In order to defend against this type of attack, most semiconductor manufacturers have adopted the practice of reserving a portion of the EEPROM for the switchover mechanism, in addition to using a fuse. If a certain unalterable value is located in this part of the memory, the chip has been irreversibly switched to the user mode. Even if the fuse is bridged over, the chip will not return to the test mode, since the additional logical switch in the EEPROM prevents this.

The security of the switchover from the test mode to the user mode can be increased even further by a very simple measure. If the microcontroller chip is laid out on the wafer such that the test pads needed to make contact with the chip for performing the tests are simply sawn off when the wafer is divided into individual dice, neither a fuse nor any EEPROM cells are needed to switch between the modes, since the elements needed for the test mode will no

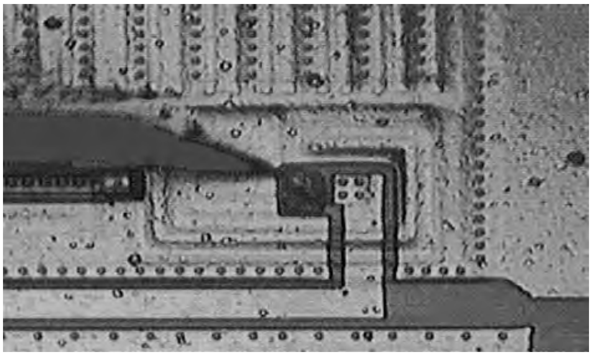


Figure 8.30 Photograph of a polysilicon fuse together with a microprobe needle, magnified 500×. A blown fuse could be bridged using a microprobe needle (*Source:* Giesecke & Devrient)

longer be present. It is also be possible to replace the fuse that switches from the test mode to the user mode by a track that is irreversibly broken when the dice are sawn from the wafer. With present-day technology, it is not possible to make a connection to a sawn-through track on the edge of a chip.

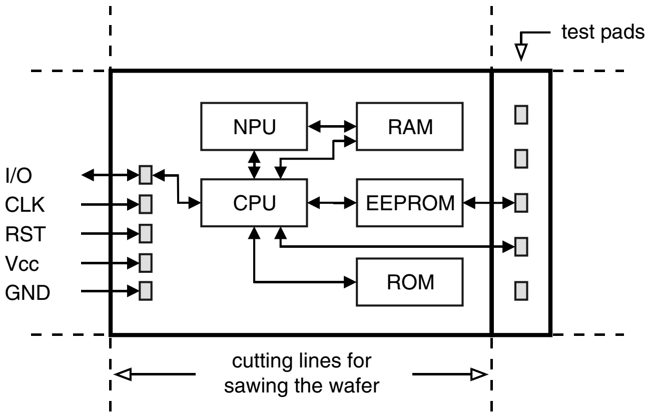


Figure 8.31 One of several possible ways to irreversibly remove the test pads used for testing the CPU and memory of a smart card microcontroller

Dynamic analysis and defense: tapping the memory busses of the microcontroller

Before the busses between the CPU and the memories of the microcontroller (ROM, EEPROM and RAM) can be tapped, the chip must be exposed and the passivation layer on the top surface of the chip must be removed. The passivation layer protects the chip against oxidation, but it also protects the chip against attack, since its integrity is monitored by sensors. According to Anderson and Kuhn [Anderson 96b], it can be removed by etching with hydrofluoric acid. In addition, a laser cutter[10] can be used to selectively cut openings in the passivation layer at the necessary locations.

After the passivation layer has been removed from the entire surface of the chip, or only from selected locations, it would be at least theoretically possible to make contact with the address, data and control busses for the memory using microprobe needles. If it is possible to make electrical connections to all the lines of these three busses, it is very easy to address the individual memory cells and to read any desired regions of the ROM and EEPROM. The chip does not have to be powered for this, and any desired type of connection jig can be used. The consequences of a successful attack using this method would be serious, since in principle it would make all the secret data in the non-volatile memory readable.

This method could be extended by making connections to the busses and then operating the chip in the normal manner. In this way, it would be possible to eavesdrop on the complete data traffic between the CPU and the memories, and this could be recorded using a sufficiently fast logic analyzer.

As already indicated, it is very difficult to make electrical contact with the individual tracks on the chip. With an 8-bit microcontroller, the number of connections needed for this attack is 16 for the address bus, 8 for the data bus and 1 to 4 for the control bus. In total, at least 25 simultaneous connections would have to be created between an external analysis computer and the tracks on the chip. Even with modern micromanipulator technology, this is currently not possible, due to the very small dimensions of the semiconductor structures. However, it would be possible to use a focused ion beam (FIB) generator, which is commonly used in the semiconductor industry, to implant a sort of electrically conductive contact surface for each bus line. These surfaces then could be used as contact points for microprobe needles. However, the effort required for this is enormous.

Even if an attacker succeeded in making these connections, he would still have to determine how the busses have been scrambled before he could successfully read the data. This is because the individual bus tracks are not arranged on the chip in an orderly fashion next to each other, but are instead arranged in an externally unrecognizable manner.

If markedly improved technology in the future should make it possible to make connections to the busses of current microcontrollers, that would probably not have any effect on security, since by that time semiconductor structures will have become significantly finer than they presently are. In addition, micromechanical technology will probably always lag behind semiconductor technology, which is based on optical processes. This means that even in the future, this sort of attack will probably not be suitable for significantly weakening the security of smart cards.

Dynamic analysis and defense: measuring the current consumption of the CPU

Already in 1995, in the first edition of this book, the following statement appeared at this point: 'The design of the processor is also crucial with regard to security. A smart card processor must have nearly the same current consumption for all machine instructions. Otherwise, conclusions can be drawn regarding the instruction being processed, based on the current consumption. A certain amount of secret information can be deduced from these conclusions.' The fact that it

[10] A laser cutter is a device for drilling and cutting using a high-power laser beam. It has an precision of a fraction of a micron

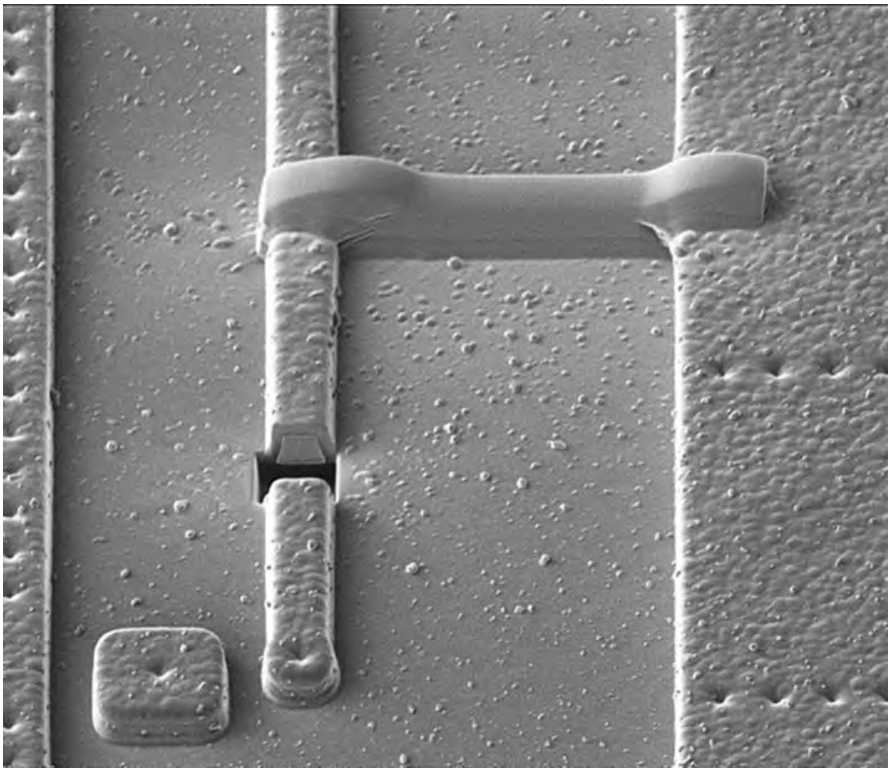


Figure 8.32 An example of using a focused ion beam (FIB) on a semiconductor chip. The track on the surface of the chip running from the top to the bottom of the picture has been separated using an FIB and then connected to a parallel track using a newly deposited metalization structure, which can be seen in the upper part of the picture. This structure was also created using the FIB (*Source:* Fraunhofer Institute for Integrated Circuits, Component Technology Group)

is possible to draw conclusions about the instructions being executed by a processor, and even about the data being processed, by analyzing the current consumption of the processor while it is executing instructions, was thus already known for several years when Paul Kocher, Joshua Jaffe and Benjamin Jun published a paper on simple power analysis (SPA) and differential power analysis (DPA) in June of 1998 [Kocher 98].[11]

The working principle of simple power analysis is relatively straightforward. The current consumption of the microcontroller is determined by measuring the voltage drop across a resistor connected in series with the power supply. Measurements are made at high time resolution using an analog-to-digital converter. With a high-performance processor, such as a Pentium or PowerPC, it would not be possible to draw any conclusions about the instructions being executed, due to the complexity of the internal processes. However, the relatively simple structures of the 8051 and 6085 CPUs used in smart card microcontrollers result in

[11] A detailed summary of this subject can be found in [Kocher 98b] and [Messerges 99]

measurable and thus interpretable variations in current consumption, according to the instructions and data being processed. To help clarify the principle, imagine that a particular program sequence with a particular set of data always produces the same plot of processor current versus time. If the same program is then run using different data, the plot of current versus time will be different. This variation is used to determine which data have been processed by the program.

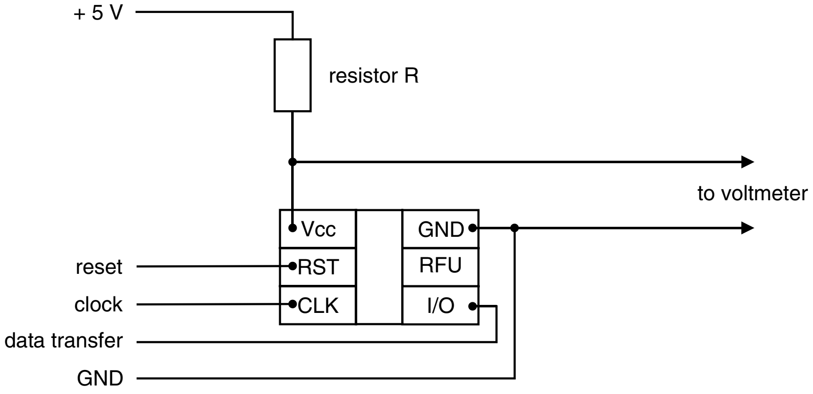


Figure 8.33 Circuit diagram of the connections to a smart card microcontroller needed to make simple current measurements using a series resistor

Differential power analysis (DPA) can reveal even finer differences in the current consumption of a microcontroller than simple power analysis. With the DPA technique, the current consumption is first measured while the microcontroller is processing known data, and then again while it is processing unknown data. The measurements are repeated many times, so that the effects of noise can be eliminated by taking average values. The differences are calculated once the measurements have been completed, and conclusions regarding the unknown data are drawn from the results.

In the paper by Kocher *et al.*, 'high-order differential power analysis' (HO-DPA) is mentioned as a further extension of DPA. This involves measuring not only the current consumption of the microcontroller, but also other variables that depend on the program being executed by the processor, such as the electromagnetic radiation of the chip. The measurement information collected in this manner using both known and unknown data can be used in the same way as in the DPA technique to calculate differences, which can then be used to compute the unknown data.

These three types of power analysis for smart card microcontrollers represent very serious forms of attack on hardware and software that have not been protected by suitable countermeasures. This is because the current consumption of some microcontrollers is definitely dependent on the machine instructions being executed and the data being processed by the instruction. In addition, the cost and complexity of the equipment needed for a successful attack using this method is relatively limited. However, there are several effective countermeasures based on suitably improved hardware and modified software.

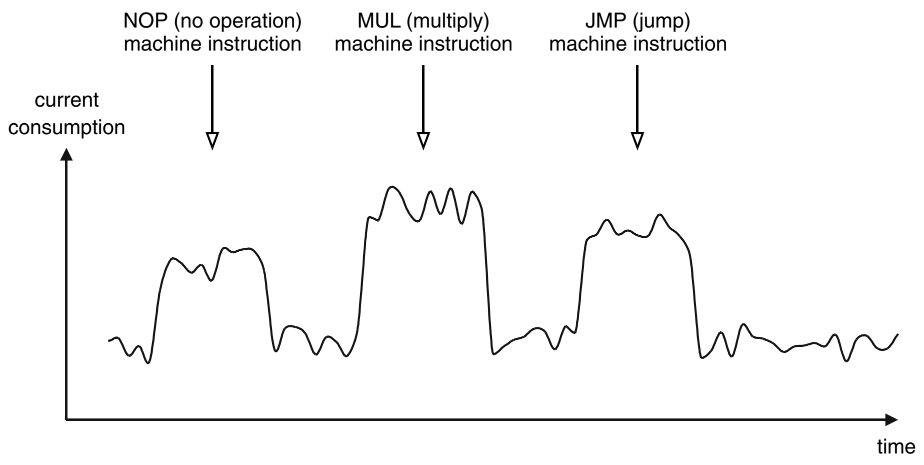


Figure 8.34 Simplified representation of variations in the current consumption of a smart card microcontroller while it is processing several different machine instructions. Besides being dependent on the machine instruction being processed, the current consumption of the processor may also depend on the data being processed

The simplest hardware solution is to incorporate a fast-acting voltage regulator in the chip that uses a sense resistor to monitor the current drawn by the microcontroller and ensures that it is independent of the instructions and data. Artificial noise current generators on the chip are also an effective solution. A technically more complicated solution is to use a modified processor design that always draws a constant current. However, all of these approaches slightly increase the power consumption of the microcontroller, which is undesirable in certain application areas, such as telecommunications. An alternative, simpler defense measure can be to activate certain components of the microcontroller that are not needed for the actual process while performing SPA/DPA-critical processes. The CRC checksum generator or numerical coprocessor could be used for this purpose, using random data as input values in order to generate artificial noise in the current consumption.

Using randomly generated delays (random wait states) in the processor considerably increases the difficulty of synchronizing the data obtained from current analysis, without increasing the chip's current consumption. A similar approach can be used with smart card microcontrollers that have their own on-chip clock generators, by continuously and randomly varying the clock frequency within certain limits.

There is presently an immense range of possible software countermeasures. Here we can describe a few representative examples. The simplest approach is to use only machine instructions that have very similar current consumptions. In this case, machine instructions whose current consumption is significantly different from the average level are not allowed to be used in the assembler code. Another approach is to have several different, randomly selected procedures for performing the same computations in cryptographic algorithms. This makes it considerably more difficult for the observer to recognize a correlation between known and unknown machine instructions or processed data. In order to make it more difficult to obtain the data needed to successfully perform a power analysis, all keys should be protected by irreversible

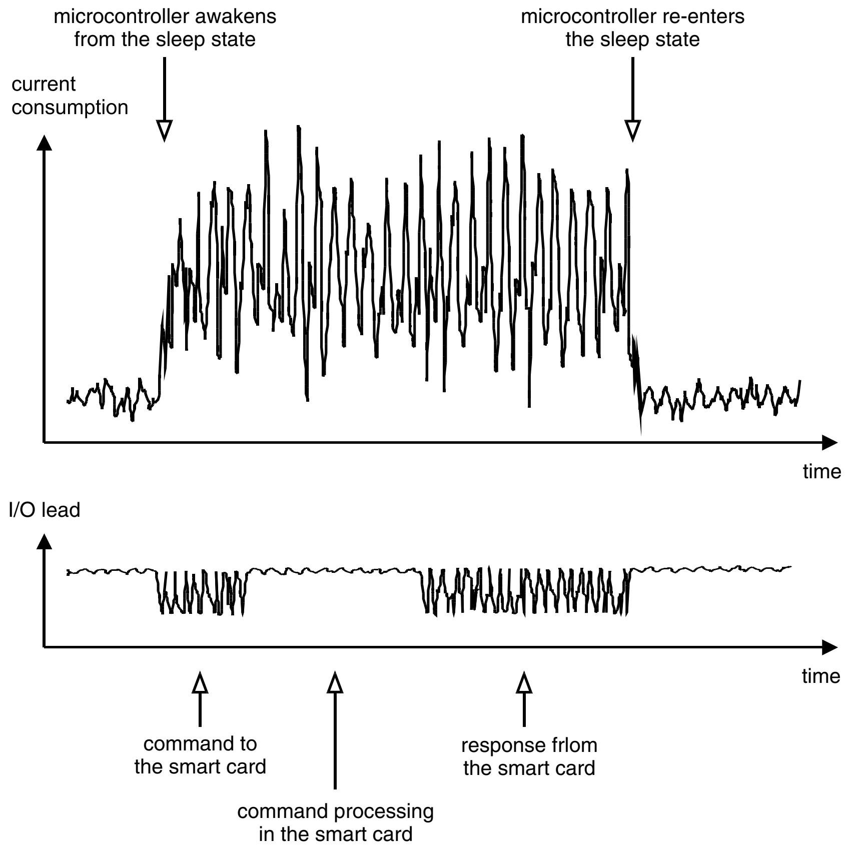

microcontroller awakens from the sleep state
microcontroller re-enters the sleep state
current consumption
time
I/O lead
time
command to the smart card
command processing in the smart card
response frlom the smart card

Figure 8.35 Simplified representation of the current consumption of a smart card microcontroller in the quiescent state and variations in its current consumption during operation. From the current drawn by the microcontroller, it is possible to recognize when it is awakened from the sleep state by the first falling edge on the I/O line, following which it exhibits a continuously varying current consumption that depends on the machine instructions being executed

retry counters. In addition, it is necessary to block free access to all commands (such as INTERNAL AUTHENTICATE) that can be used to pass any desired data through a cryptographic algorithm in the smart card. If it is essential to use commands of this sort for some reason, the smart card must test the authenticity of the terminal before executing them. Restricting the use of the available commands also makes it more difficult to collect reference data for a subsequent power analysis.

As a matter of principle, secret data should never be processed bitwise, since doing so considerably simplifies SPA/DPA analysis. When keys have to be loaded into the registers of a cryptoprocessor, in some implementations they are intermixed with random numbers that are also loaded in these registers as dummy values, in order to render the corresponding measurements meaningless. Of course, the true keys must be located in the registers at the end of the loading process.

SPA/DPA techniques are not just limited to ferreting out secret data stored in smart cards. They can also be used for purposes such as convincingly demonstrating that specific program code is used in a smart card. This is done by making an SPA analysis of the function in question in the smart card and comparing the current consumption plot obtained in this manner with the plot for a reference card. Even if the source code is not known, under favorable conditions this technique can for example be used to prove that segments of program code from an outside source are being used in a competitor's product. The technical basis for this is the fact that generally speaking, the machine code produced from the same source code by a given compiler will also be the same. The differences arising from the subsequent linking process, due to the almost certain differences in code localization in the memory, are relatively small.

Testing software in smart cards for resistance to SPA/DPA attacks has presently reached a high level of refinement and thus taken on the character of a specialist discipline. It has become common for measurements to be made periodically during software development, with the software being modified as necessary according to the results of the measurements in order to defeat SPA/DPA attacks. At the early stages of development, measurements are made with the software in EEPROM, and the analyses are repeated and refined when the first samples are obtained from the semiconductor manufacturer with the software in the ROM of the microcontroller. This is because experience has shown that this aspect is definitely significant with regard to SPA/DPA measurements.

By their nature, SPA and DPA can be used for more than just mounting attacks on cryptographic algorithms. Both methods are also very suitable for analyzing all activities of the processor. With suitable experience and equipment, it is even possible to determine the data involved in copy operations within the memory of a smart card that is not resistant to these types of attack.

Analysis and defense: measuring the electromagnetic radiation of the CPU

It is at least theoretically possible to draw conclusions about the internal processes of the smart card microcontroller from measurements of its electromagnetic radiation, in the same manner as with differential power analysis. Magnetic fields with small dimensions and strengths can be measured using SQUIDs (superconducting quantum interference devices). However, this is technically enormously difficult, and the knowledge of the internal structure of the semiconductor device that is indispensable for this method is not generally available. In addition, ICs can be very effectively protected against this sort of attack by stacking several traces on top of each other, so that even if a magnetic field can be measured, it is not possible to determine which of the tracks is actually carrying the associated current.

Manipulating the smart card microcontroller

Manipulation and defense: altering the memory content of the smart card microcontroller

Directly reading the memory content of a microcontroller is a possible attack scenario whose danger can be appreciated at first glance. A similar scenario that is almost as strong a form of attack is intentionally altering the data content in a memory of the smart card microcontroller. This does not mean randomly introducing errors in the computation process of a cryptographic algorithm, which forms the basis of differential fault analysis (DFA), but instead selectively changing the values of certain bits or bytes in the ROM or EEPROM.

Non-selective changes in all types of memory can be produced by (for example) exposing the module to X-rays or shining ultraviolet light on the exposed chip. EEPROM cells can be discharged by exposing them to ultraviolet light, which causes their contents to take on the value of the lowest-energy state. This process is exactly the same as erasing a conventional EPROM using an ultraviolet lamp. However, it cannot reasonably be used for an attack, since the attacker has no control over which EEPROM cells are switched.

However, the ultraviolet lamp can be replaced by a collimated beam of light or light from a laser, and this can be focused to a fine point. This could certainly be used to alter the contents of individual memory cells. The advantage of using a laser is that it can supply enough power to also modify the contents of ROM cells. A focused ion beam can also be used in a similar manner to change the contents of memory cells.

The changes that are possible can certainly be used for theoretically effective attacks. For example, the random number generator could be manipulated such that it no longer produced random numbers, but instead always supplied the same value. If this were possible, authentication of the terminal by the smart card could be broken by a replay attack using a previously employed value.

It is certainly possible to imagine other types of attacks that could be carried out if the contents of specific memory bits could be intentionally modified. For example, all S boxes of the DES algorithm could be intentionally changed to a uniform value of zero or one. This would mean that the DES algorithm would no longer act as an encryption algorithm, but only as a linear transformation [Anderson 96a].

If the exact location of the DES key in the EEPROM is known and it is also possible to modify individual bits in the EEPROM (using focused ultraviolet light, for example), it is naturally possible to utilize these conditions to mount an effective attack. This attack consists of setting an arbitrary bit of the key to 0 and then calling a command that uses the DES algorithm with the modified key. If the return code indicates a parity error in the key, the bit that has been modified was originally set to 1, while if no parity error is reported, the bit was already set to 0. The same procedure is then followed for the remaining 55 bits of the key, with the result that the secret key is known [Zieschang 98].

Many other types of attack along the same lines are possible, such as selectively modifying program processes or altering pointer values. These attacks may look very simple and attractive on paper, but it would be very difficult to carry them out in actual practice. The necessary conditions for a successful attack are not exactly easy to achieve, so this type of attack remains an interesting but theoretical concept.

In order to alter bits selectively, an attacker must have detailed knowledge of the physical addresses of the data and program code in the memory, and he must also know the scrambling and/or encryption schemes used for the memory in question. In addition, all data and routines that are significant with regard to security are protected using checksums that are always checked before using the data or routine. This means that the attacker would also have to selectively modify the checksum to match the modified data. You should also not overlook the fact that all protective layers covering the memory in question must be neutralized before any manipulation can take place. All of these considerations together reduce the attractiveness of this type of attack to almost nothing, even though it must be admitted that it sounds very attractive in theory.

Attacks at the logical level

The main prerequisite for attacks on the security of a smart card at the logical level is knowledge of the communications and information flow between the terminal and the smart card. In this case it is not particularly necessary to understand the processes occurring at the hardware level, but rather the software processes. In terms of information technology, the sample scenarios described here are located one level above attacks that primarily exploit the properties of the hardware.

Attack and defense: dummy smart cards

Probably the simplest imaginable type of attack is to use a smart card that has been custom programmed and includes additional logging and analysis functions. Up until a few years ago, this was practically unfeasible, since only a few companies had access to smart cards and the microcontrollers used to produce them. Nowadays, though, smart cards and configuration programs can be freely purchased from a number of companies. This naturally increases the options available to an attacker. Even without this, with a certain amount of effort and dexterity it is possible to assemble a working smart card using a plastic card and a standard microcontroller in an SMD package. Such a card can at least be made to imitate the electrical

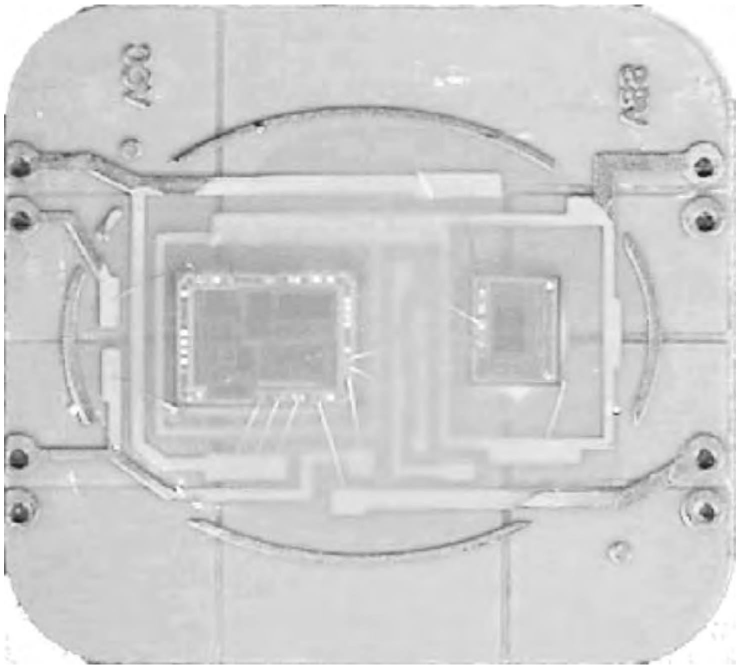


Figure 8.36 Rear view of an opened smart card module. The chip at the left is a standard PIC microcontroller that is connected to an EEPROM memory chip at the right by bonding wires and tracks. This type of chip module is typically used for cloned smart cards and other types of attacks on smart card systems

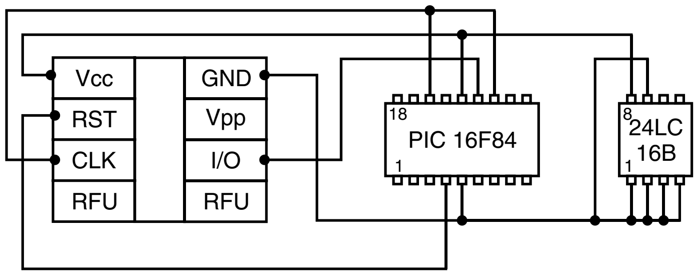


Figure 8.37 A typical substitute circuit for an smart card microcontroller built using standard discrete components (PIC 16F84 microcontroller and 24LC16B EEPROM memory chip). These components fit into a typical smart card module, so it is not possible to detect any difference from a genuine smart card microcontroller without investigating the module. This circuit and variations on it can be found on the relevant Internet sites

interface of a real smart card and to behave the same way for data transfers. It is now possible to obtain such cards from a wide variety of sources via the Internet. New possibilities are also offered by Java technology for smart cards, which makes it easy to generate programs and load them into dummy cards.

With such a dummy card, it would be possible to record at least a part of the communications with a terminal and subsequently evaluate this information. After several attempts, it would probably be possible to perform part of the communications in exactly the same way as a genuine smart card. Whether this can be put to advantage is doubtful, since all professionally designed applications have cryptographic protection for important activities. As long as the secret key is not known, the attack will not go any farther than the first authentication. Such an attack can only be successful if the secret key is known or the complete application runs without any cryptographic protection. Should such an application exist, it is highly doubtful that any benefits that could be obtained from this type of attack would be sufficiently large to justify the necessary effort.

Analysis: determining the command set of a smart card

The instruction classes and commands that are supported by a smart card are of course not often published, but it is very easy to determine what they are. This is more interesting with regard to completely determining the command set of a smart card than it is for an attack on the security of the smart card. However, it is conceivable that an attack could be mounted on the basis of this information.

The method used to determine the command repertoire is illustrated in Figure 8.38. The first step is to generate a command APDU and send it to the smart card using a freely programmable terminal. The class byte in the APDU is changed for each APDU to cover the range from '00' to 'FF'. As soon as a return code other than ‘invalid class’ is received, the first valid class byte has been determined. There are usually two or three valid instruction classes, which can then be used to try all possible instruction bytes in the next round. This consists of sending command APDUs with various instruction bytes to the smart card and noting the ones that yield a return code other than ‘unknown instruction’. If suitable software is available in the terminal, this method can be used determine which commands are supported by a particular smart card in one to two minutes. To a certain extent, a portion of the possible parameters of the commands so identified can also be determined in a similar manner.

This algorithm can be made considerably faster by using only the class byte codes allowed by the ISO/IEC 7816-4 standard and allowing the instruction byte to be an indexed variable. This strongly reduces the number space of the class byte by taking secure messaging and logical channels into account. A similar improvement can be made by using only even-valued instruction bytes, since the odd-valued codes contain only the V_{pp} control information, which is no longer used.

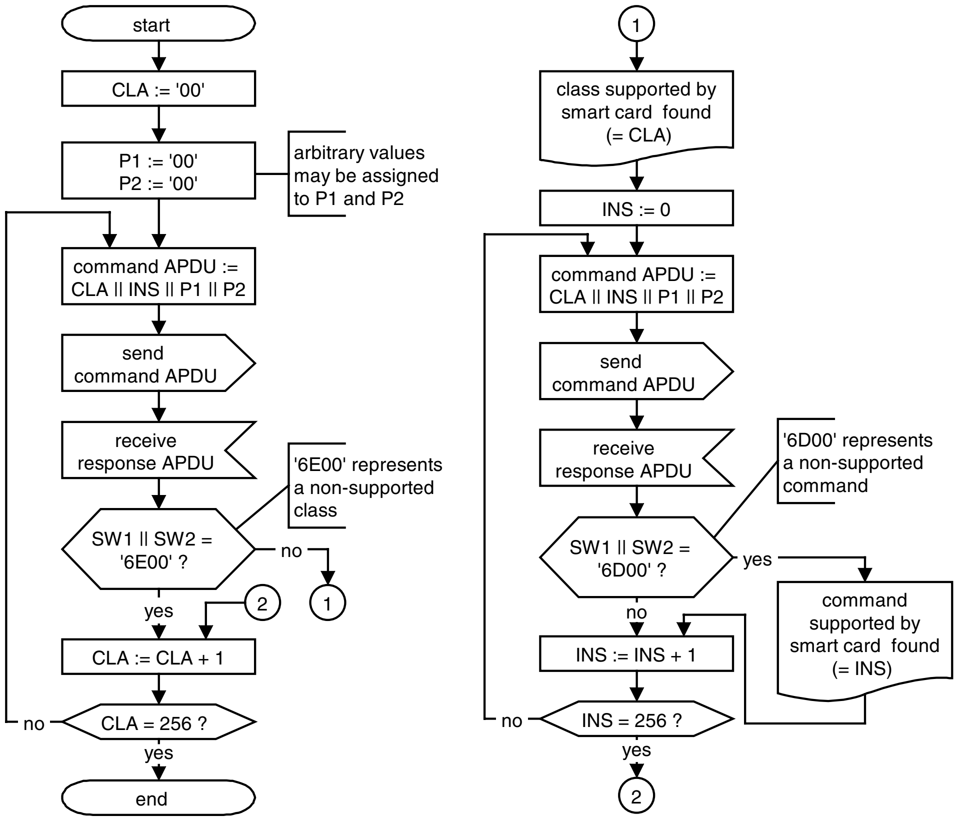

start
CLA := '00'
P1 := '00'
P2 := '00'
arbitrary values may be assigned to P1 and P2
command APDU := CLA || INS || P1 || P2
send command APDU
receive response APDU
'6E00' represents a non-supported class
SW1 || SW2 = '6E00' ?
no
yes
2
1
CLA := CLA + 1
CLA = 256 ?
no
yes
end
1
class supported by smart card found (= CLA)
INS := 0
command APDU := CLA || INS || P1 || P2
send command APDU
receive response APDU
'6D00' represents a non-supported command
SW1 || SW2 = '6D00' ?
yes
no
command supported by smart card found (= INS)
INS := INS + 1
INS = 256 ?
no
yes
2

Figure 8.38 Basic procedure for performing an exhaustive search for all commands supported by a smart card operating system. The results of the search will only be complete if command invocation is not controlled by a state machine. The procedure works on the principle of systematically testing all class byte (CLS) and instruction byte (INS) codes in turn, ignoring any command contents that may be present (secure messaging, logical channels, V_{pp} control and so on)

The reason that this simple search algorithm for instruction classes, commands and parameters can be so effective is that practically all command interpreters in smart card operating systems evaluate received commands by starting with the class byte and working through the following bytes. This process is terminated as soon as the first invalid byte is recognized, and a suitable return code is generated and sent back to the terminal. However, it can only work if the smart card does not have a global state machine that monitors the command sequence. If such a state machine is present, it is at least possible to use this procedure to determine the command sequence in a step-by-step manner.

The utility of such a procedure for an attacker may not appear to be that great, since the command set is usually not secret. However, it does at least provide a simple and fast means to determine all of the available commands. It is also a very useful means for determining whether the producer of the operating system has incorporated any undocumented commands in the software.

Attack: tapping data transmissions

A slightly modified smart card can be used to tap data transmissions during a session and manipulate the data as desired. The modifications consist of gluing an insulated dummy contact on top of the I/O contact, so that the original I/O interface is no longer connected to the I/O contact. The new (dummy) contact and the original I/O contact are then connected to a fast computer. With suitable programming, this computer can delete or insert any desired data within the communications between the terminal and the smart card. If the computer is sufficiently fast, neither the terminal nor the card will detect any difference between normal and manipulated communications.

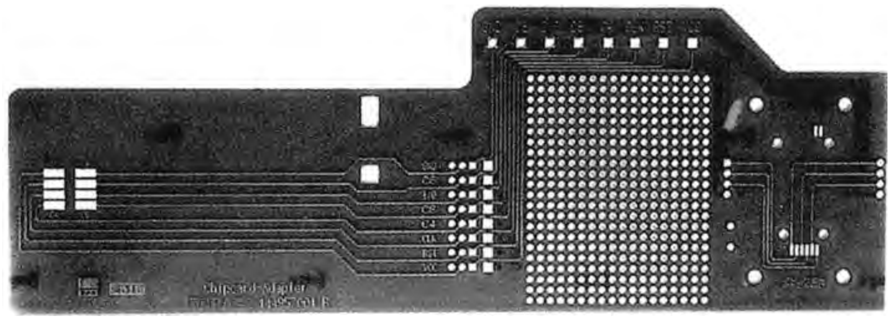


Figure 8.39 An adapter that can be used to extend a smart card outside of a terminal enclosure in order to allow measurements to be made on the card. The eight contacts can be seen on the left, and a prototyping area for electronic circuitry can be seen on the right

It is clear that the course of a session can be radically affected using this method. Whether an attacker can derive any benefit from this method depends primarily on the application in the smart card. A well-known design principle says that eavesdropping on communications or the deletion or insertion of data in the communications stream must not be allowed to impair security. If this principle is not observed, an attacker can certainly obtain an advantage using this method. There are known cases of fraud using simulated memory cards.

In order to provide protection against this type of attack, some terminals have shutters that cut off any wires attached to the smart card. Secure messaging can also be used very effectively here to allow any manipulation of the data during the data transmission to be reliably detected.

Many terminals may be used only under supervision, which makes it difficult to use manipulated cards with leads to an accompanying computer in such terminals. In summary, although this type of attack can be regarded as very interesting and quite promising in theory, in practice it achieves only modest success.

Attack and defense: power interruption

A type of attack that was successful with many smart cards until recently is to interrupt the power to the card at a particular time while a command is being executed. This type of attack is based on the fact that with conventional programming, all write operations to EEPROM pages are performed sequentially. If the programmer has not been clever in arranging the order of the write operations, an attacker can derive an advantage by cutting off power at the right time.

This can be briefly illustrated using a highly simplified example. In an electronic purse application, if the balance is increased before the log file is updated when processing a purse loading command, an attacker would have a good chance of being able to load a smart card for free. He would only have to switch off the power at the right time, or jerk the card out of the terminal with millisecond accuracy (!). The purse balance would then have been changed to the new value, but there would no log record for this transaction and no response to the command. With simple electronic purse systems in the past, such an attack was certainly a real possibility.

In order to determine the exact time to terminate processing, the attacker only has to use an electronic counter to count the number of clock pulses after the time when the command is sent and then perform a series of experiments with increasing clock counts to determine the proper time to interrupt power to the card. It hardly needs to be said that the entire procedure can be more or less automated using a computer.

	purse balance	purse balance file (in binary notation)
1. current purse balance	100 EUR	°0110 0100°
2. deduct 10 EUR	90 EUR	
3. erase the EEPROM	255 EUR	°1111 1111°
4. write the new purse balance	90 EUR	°0101 1010°

Figure 8.40 Example procedure for writing a new balance in an electronic purse. Here it is assumed that the erased state of the EEPROM represents a logic 1. Due to the way the EEPROM works, this means that the entire EEPROM page must be erased (which means setting all of its bits to 1) if only one bit in the page must be changed from 0 to 1. In this example, if the power for the smart card is cut off exactly after the EEPROM has been erased, which means after step 3, the purse balance would be set to its maximum value and the attacker would have effectively created money. This can be reliably prevented by using atomic operations

Although this type of attack sounds attractive and appears to be easy to copy, in practice there are several effective countermeasures. The simplest approach is arrange the EEPROM write instructions in a carefully considered order. The EN 1546 standard for multisector electronic purses is well worth examining in this regard, since all of the electronic purses described in this standard are explicitly protected against this sort of attack.

However, even a perfectly ordered sequence of write operations cannot by itself achieve absolute protection. This can be illustrated using another example. When the electronic purse of our previous example is being loaded, it may be necessary to erase the EEPROM before the write process. If the erased state of the EEPROM corresponds to the maximum value of the purse balance, which incidentally is the usual case, the purse can be artificially loaded to its maximum value by simply interrupting the power to the card at the right time. The proper moment is when the erase operation has just been completed and the write operation has not yet been started.

Operating system designers know an effective countermeasure for this type of attack, which is to use atomic operations as described in detail in Section 5.10. The characteristic of an atomic operation is that it is indivisible, which means that it is performed either completely or not at all. This provides fully adequate protection against the type of attack just described. Even the optimally ordered EEPROM write operations described in the EN 1546 standard require atomic operations in several locations to prevent this type of attack from being implemented.

Attack and defense: current analysis during PIN comparison

A technically very interesting type of attack on comparison features, such as PINs, can be carried out using a combination of physical measurement of a parameter and variation of logical values. This type of attack relates to all mechanisms in which data are sent to the smart card and compared in the card with corresponding values, with a retry counter being incremented according to the result of the comparison.

The attack works on the principle of measuring the current drawn by the card, for example by measuring the voltage drop across a resistor in the Vcc lead. If a suitable command containing the comparison data is sent to the card, it is possible to see from the current measurement whether the retry counter has been incremented, even before the return code has been received. If the return code is sent before the retry counter is written when the result of the comparison is positive, this method can be used to determine the value of the reference data. This is done by sending all possible variations of the comparison value to the smart card and cutting off power to the card before the retry counter has been incremented if the result is negative. A positive result can be clearly recognized from the associated return code, which is sent before the retry counter is written.

There are two basic ways to defend against this type of attack. The simplest defense consists of always incrementing the retry counter before making the comparison, and then decrementing it afterwards as appropriate. In this case, the attacker cannot obtain an advantage, regardless of when he interrupts power to the card, since the retry counter will have already been incremented. The second defense is more complicated, but it provides similar protection. In this approach, the retry counter is incremented after a negative comparison and written to an unused EEPROM cell after a positive comparison. Both of these write accesses occur at the same time in the process, so the attacker can draw no conclusions with regard to the result of the comparison. He learns the result of the comparison only after receiving the return code, and at this point it is too late to prevent a write access to the retry counter by cutting off the power.

Attack and defense: timing analysis of PIN comparisons

Programmers always give considerable attention to making programs execute as quickly as possible. Normally, this is also an important consideration. However, the fact that the execution time of a process has been optimized can be utilized for an attack that definitely has a good chance of success. If a PIN is sent to a smart card for comparison, the associated comparison routine normally compares the PIN it receives with the stored PIN value byte by byte. A programmer who is not security-conscious will program this routine such that the first difference between the two compared values causes the routine to immediately terminate and return to the calling program. This leads to minute variations in the execution time of the comparison process, which can nevertheless be measured using suitable equipment (such as a storage oscilloscope). This information could be used by an attacker to determine the secret PIN code in a relatively straightforward manner.

Up to a few years ago, this was still an effective type of attack on smart cards. However, it is now a known type of attack, and comparison routines are constructed such that all digits of a PIN are always compared. Consequently, there is no time difference between positive and negative comparison results.

Protection: noise-free cryptographic algorithms

The security of smart card applications is based on secret keys used with cryptographic algorithms. In order to access the card in certain ways or perform certain operations with the card, the terminal must always first authenticate itself using a secret key. Naturally, authentication of the terminal by the card represents an attractive target for an attacker. By contrast, authentication of the card by the terminal is not attractive with respect to an attack on the card, since a smart card can be manipulated as desired using a (dummy) terminal.

The smart card authenticates the terminal by sending it a random number, which the terminal then encrypts and returns to the card. The smart card then performs the same encryption and compares the result with the value received from the terminal. If the two values match, the terminal has been authenticated, and it receives a corresponding return code. If the authentication fails, the card sends a different return code. The starting point for the attacker is analyzing the processing time between when the command is sent and when the response is returned by the smart card.

As late as the early 1990s, cryptographic algorithms with significant differences in execution times for different keys and plaintexts were still sometimes used. The resulting reduction of the key space can be exploited by an attacker to search for the secret key using a brute-force attack. The duration of the search is strongly dependent on the noise level of the algorithm. The size of the key space becomes smaller as the variation in execution time increases, making it easier and faster to search for the key. If the exact implementation of the algorithm on the target computer is known, this information can also be included as reference data for generating the timing tables. This type of attack was made public under the name 'timing attack' in a publication by Paul Kocher in 1995 [Kocher 95], which primarily deals with the time dependencies of the RSA and DSS algorithms.

In principle, a timing analysis is a very dangerous threat to the security of a smart card. However, since this type of attack has been known for a relatively long time, all present-day smart cards use only noise-free cryptographic algorithms, which are algorithms for which the time required for encryption or decryption is independent of the input values. This blocks this sort of attack. However, the programmer has conflicting interests in this regard, since a noise-free algorithm usually requires more program code and is always slower than a noisy version. The reason for this is that a noise-free algorithm must be designed such that the path through the program has the same length for all combinations of plaintext data, ciphertext data and keys. This means that the longest necessary path is the reference value, and all other paths must be suitably modified to match this length.

To provide additional security, in some applications all authentication keys have their own retry counters, so that only a limited number of unsuccessful authentications can be performed. Once the retry counter has reached its maximum value, the smart card blocks all further attempts at authentication.

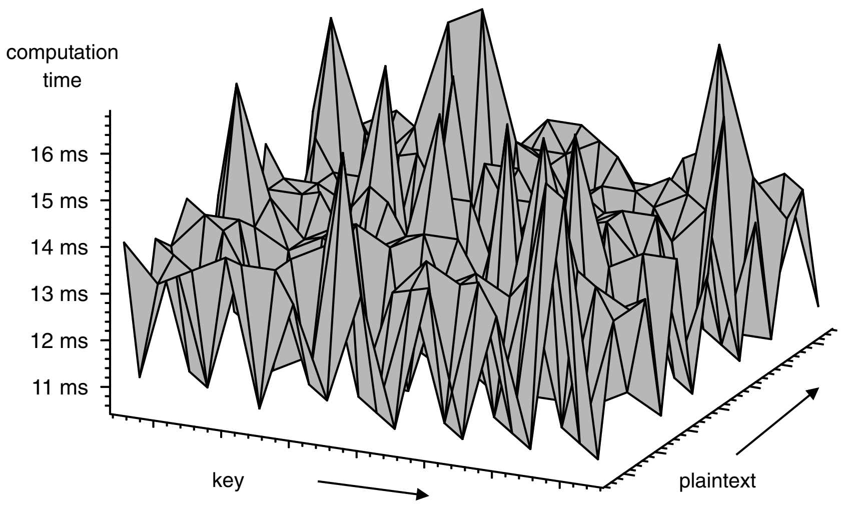


Figure 8.41 Example of the effects of ciphertext and plaintext data on a noisy encryption algorithm. This plot shows a portion of the plaintext / ciphertext space. It was generated using an old implementation of the DES algorithm, with 100,000 iterations per measurement value

Manipulation: differential fault analysis (DFA)

As is well known, the operation of electronic devices can be adversely affected by exposing them to electromagnetic interference. For instance, a mobile telephone can cause the processors of many types of small computer-controlled appliances to crash. The cause lies in the memory cells, whose contents can be altered by the high-frequency AC fields.

In 1996, Dan Boneh, Richard DeMillo and Richard Lipton published a study [Boneh 96] describing a theoretical method for determining the secret keys of asymmetric cryptographic algorithms by introducing scattered hardware errors. Since the three discoverers of this method worked at the Bell Communications Research (Bellcore) Laboratories at the time, this type of attack is often called the Bellcore attack.

Only two months later, Eli Biham and Adi Shamir published an extension of the Bellcore attack called *differential fault analysis* (DFA) [Biham 96], which also included symmetric cryptographic algorithms such as DES. This meant that, at least in theory, many smart card applications were exposed to a new and serious form of attack.

The basic principle of both of these types of attack is relatively simple. In the first step, an arbitrary plaintext is encrypted using the key to be broken, and the resulting ciphertext is saved. Following this, the operation of the card is disturbed while it is processing the cryptographic algorithm, for example by exposing it to ionizing radiation or high-frequency fields in order to alter a single bit of the key in a random location while the computation is being performed. This yields a ciphertext that is incorrectly encrypted, due to the altered bit. This process is repeated many times, and all the results are saved for analysis. The remainder of the procedure for determining the value of the secret key is purely mathematical, and it is fully described in the papers just mentioned.

The strength of this attack is primarily due to the fact that it is not even necessary to know the location of the altered bit in the secret key. Biham and Shamir state in their publication that with a single corrupted key bit, 200 ciphertext blocks are sufficient to compute the value of the secret DES key. If triple DES (with a 168-bit key) is used in place of simple DES, the number of required ciphertexts does not increase significantly. Even if more than one bit is altered, this attack remains effective; the only consequence is that more incorrectly encrypted ciphertexts are needed.

In practice, this type of attack is not as simple as it sounds. If at all possible, only one bit should be altered, or at least only very few bits. If the entire microcontroller is simply bathed in microwave radiation, usually so many bits will be altered that the processor will hopelessly crash. Consequently, an attempt is made to induce the processor to make isolated processing errors by injecting specially prepared glitches[12] into the power or clock lines. If the filter on the associated input leads cannot neutralize these glitches, they can produce the desired processing errors.

However, a smart card is not totally helpless in the face of a Bellcore attack or DFA if suitable precautions are taken. The simplest defense is to simply compute the cryptographic algorithm twice and compare the two results. If they match, no attempt has been made to alter any bits from outside the card. This defense assumes that intentionally introduced random errors can never alter the same bit twice in a row. This is a realistic assumption, since if it ever became possible to selectively alter specific bits in a smart card processor, attacks that are much simpler and faster than DFA would be possible. The main disadvantage of double computation is the additional time that it requires, which can cause problems. This applies primarily to attacks on time-intensive asymmetric cryptographic procedures, such as RSA and DSS.

Another effective defensive measure against differential fault analysis can be achieved by always encrypting different plaintexts. The simplest solution is to prefix the plaintext to be encrypted with a random number. This means that the cryptographic algorithm always encrypts different data, which prevents DFA from being used.

In summary, the Bellcore attack and differential fault analysis are unquestionably dangerous types of attack that can succeed with smart cards that do not incorporate adequate protective measures. However, all smart card operating systems and applications were modified to protect them against these types of attack shortly after they became known, so neither the Bellcore attack nor DFA currently represents a serious threat.

[12] A glitch is a very brief interruption or spike in the voltage or current

Attack and defense: disturbing the processor

A type of attack that is similar to using differential fault analysis to attack the secret key of a cryptographic algorithm consists of attempting to affect the execution of program code routines by disturbing the operation of the processor. A type of attack that has been known to manufacturers of smart cards and smart card microcontrollers since around 1998 is the 'light attack', which was described in mid-2002 by Sergei Skorobogatov and Ross Anderson [Skorobogatov 02] as an 'optical fault induction attack'.

This paper describes an arrangement in which a standard commercial flash unit is attached to the camera adapter flange of a conventional optical microscope. Following this, a highly restricted region of the RAM of a standard microcontroller (PIC16F84) is exposed to light from the flash unit. With microcontrollers that are not hardened to resist this type of attack, this arrangement can be used to selectively set certain bits in the RAM to the logic 0 or 1 states.

The operation of the processor can be disturbed by applying glitches to the supply lines, exposing the chip to flashes of light or using high-frequency radiation [Lamla 00], among other things. If the disturbance is triggered at the proper instant during the execution of the program, it can be used to intentionally influence a query operation, for instance. A simple example of this is shown in Figure 8.42. The task of the illustrated routine is to send the content of a transmit buffer, whose boundaries are specified by a start address and an end address. If the attacker succeeds in intentionally disturbing the query that determines the end address of the transmit buffer, data following the end of the transmit buffer will also be sent to the terminal. Should the workspace for a cryptographic algorithm be located in this region of memory, its keys could be illicitly read out in this manner.

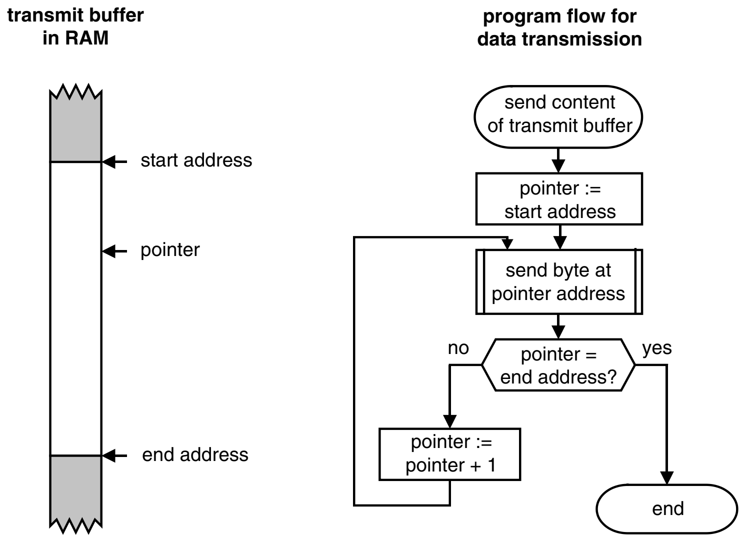


Figure 8.42 Example of a non-robust routine for sending the content of a transmit buffer, which can be successfully attacked by disturbing the processor

The defense against this attack involves several system levels. At the hardware level, it is important for the smart card microcontroller to have suitable sensors, so that it can detect all attempts to disturb the processor. These sensors can include voltage glitch detectors and a large number of suitable light sensors. In order to make it impossible to defeat a few light sensors by covering them with black ink, it is a good idea to use a relatively large number of sensors distributed over the surface of the chip. This by itself is sufficient to preclude many types of attack. An opaque chip encapsulation material provides only limited protection, since it can be removed relatively easily using chemical methods.

The second level of protection must be implemented in the software. The program code shown in the example can be made significantly more robust by using an 'equal to' query instead of a 'less than or equal to' query. Another countermeasure is to execute the query twice, with a random delay between the two queries. This requires the attacker to use two flashes of light to manipulate the query, and he will be additionally hindered by the fact that he cannot exactly predict the timing of the second flash.

In addition, all confidential data stored in RAM should be immediately deleted after they have been used, or they should be temporarily encrypted. In order to further reduce the consequences of this type of attack, it is also a good idea to encrypt all secret data (such as PIN codes and keys) located in EEPROM. Should an attacker succeed in reading out portions of the EEPROM by manipulating queries, he would then only obtain encrypted data, which would be of no use to him. If an MMU is present, it can also be configured to monitor compliance with certain boundaries for transmitting data from the card. Furthermore, modern processors can detect illegal machine instructions and invalid addresses and respond appropriately. As can be clearly seen from this defense scenario, an attack that unquestionably can be regarded as serious can be blocked by suitable combination protective measures in hardware and software.

Protective elements: smart card operating systems

Protective mechanisms in the hardware form the basis for protective mechanisms in the operating system software. No potential weakness may be overlooked, since the three components of the protective mechanisms – hardware, operating system and application – are linked in a logical AND relationship. This is similar to a chain, in which the weakest link determines its breaking strength. If a particular mechanism fails in a smart card, the entire security of the card collapses. The operating system in particular forms the basis for the actual application, whose information and processes must be protected.

The following material deals specifically with measures for protecting against typical attacks, rather than general smart card security functions. However, most of these general functions also contribute significantly to operational security and protection against attacks. For this reason, you are explicitly referred to the appropriate sections of Chapter 5.

Protection: hardware and software tests following a reset

When the operating system is initialized, at minimum the most important parts of the hardware must be tested to see if they are in proper working order. For instance, a RAM test is indispensable, since all access conditions are stored in the RAM while the chip is operating, and failure of a single bit could cause a complete security collapse. It is likewise necessary to compute and test the checksums for the most important portions of the ROM and EEPROM. The CPU is at least implicitly tested by sending the ATR, since the bulk of all possible machine instructions must be executed faultlessly for this to be possible. Explicit testing of the CPU and any NPU that may be present can usually be limited to sample testing, since completely testing all functions for flawless operation would take too much time and code.

If the operating system discovers a hardware error or checksum error, there are two possible ways to proceed. The first option is for the software to immediately jump to an endless loop, which means that an ATR cannot be sent and subsequent commands can no longer be received. The main disadvantage of this is that the cause of this behavior cannot be recognized from the outside. The problem might be a broken bonding wire, a fractured chip or a checksum error in the EEPROM, but this cannot be determined by the user. A better option is to have the smart card attempt to send a special ATR before disabling itself by entering an endless loop. The error ATR at least gives the outside world an indication of what has happened inside the smart card. However, it must not be overlooked that simply sending an error ATR requires a largely functional CPU, a few bytes of RAM and several hundred bytes of program code in the ROM.

Protection: layer separation in the operating system

Layer separation, with clearly defined parameters for transitions between the individual layers, is a sign of a stable and robust smart card operating system. The consequences of possible design or programming errors in the operating system are minimized by clean separation of the layers within the operating system. Of course, this does not mean that such errors will not occur, but the effects of the errors will not be as extensive as with an operating system programmed in very compact, condensed code. Layer separation makes it difficult for any error that occurs in one layer to propagate to other layers.

Protection: supervising data transmission

Another very important element of security is to supervise the data transmission process in order to protect the memory against unauthorized accesses. All communications to and from the smart card take place via an I/O interface supervised by the operating system. No other form of access is possible. This represents an effective form of memory protection in the smart card, since it ensures that the operating system always retains control over access to memory regions.

The transmission protocol, which is controlled by the transport manager, must intercept all possible incorrect inputs. There must be no possibility of influencing the data transmission process by manipulating transfer blocks in order to cause data to be illicitly sent from the memory to the terminal.

Protection: checksums for important memory contents

The file structure, and in particular the file headers (file descriptors), should be protected using checksums. This enables the operating system to at least detect any unintentional changes to data stored in memory. This requirement is especially important in light of the fact that the object-oriented access conditions for each file are stored in this part of the file.

All memory regions of the EEPROM that are vitally important for the smart card operating system must be protected using checksums (EDCs). Whenever such a region is accessed or the code it contains is called to be executed, the consistency of its contents must be verified before the access or code execution is allowed to proceed.

Protection: encapsulation of applications

Some operating systems encapsulate the individual DFs containing the applications and their files, so that individual applications are isolated from each other. However, this concept is based on software protection alone, with no support from the chip hardware. The amount of protection is thus not as great as it could be. Nonetheless, even this software approach to application encapsulation can be very beneficial in case of an error, since it makes it impossible for the file manager to exceed the boundaries of a DF without explicit prior selection. The effects of a memory error on a file are thereby at least limited to the DF in question.

If hardware support for the operating system is present in the form of a memory management unit (MMU), the various applications can be fully isolated from each other. In this case, even manipulated software within an application cannot obtain unauthorized access to the memory regions of other applications.

Protection: camouflaging the activities of the operating system

Whenever data must be written to the EEPROM, the charge pump in the chip must first be switched on. This increases the current consumption of the chip, and with some types of microcontrollers this can easily be detected using a suitable measurement setup. This means that the fact that it may be possible to externally determine when EEPROM write accesses occur must be taken into account in the design of the operating system. The software in the smart card must prevent an attacker from being able to take advantage of this knowledge.

It is very important that it should not be possible for an attacker to draw any useful conclusions about processes and decisions in the machine program by measuring the current drawn by the card. For instance, it would be fatal if it were possible to use such measurements to reliably judge the outcome of a PIN comparison before the completion of command processing and transmission of the return code, since this information could very easily be used to analyze the value of the PIN.

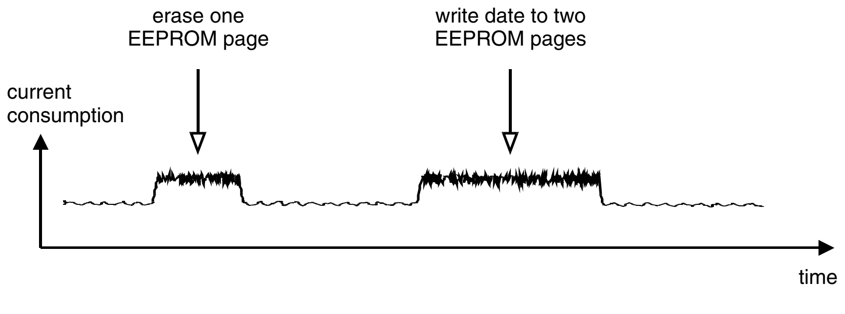


Figure 8.43 Approximate representation of the variation in the current consumption of a smart card when the charge pump is switched on

Protection: object-oriented access conditions

Early smart card applications were always based on a centrally managed access mechanism. One disadvantage of centralized access management mechanisms is that software or memory errors can affect the overall security of the smart card. Modern object-oriented file management systems, in which the access conditions are stored in the individual files, have the advantage that only a single file is affected by a memory error, with the security of all other files remaining intact. This is actually a fundamental property of all distributed systems. They are somewhat more difficult to program, but they provide significantly stronger security against attacks and errors, due to their self-sufficiency.

Protection: disabling the smart card

The operating system must allow the smart card to be fully disabled. This is very important for the final stage of the smart card life cycle. Using statistical methods, it is possible to perform very exact analyses of the software in the chip by collecting discarded but still fully functional smart cards. To prevent this, mechanisms for completely disabling the operating system and all of its routines must be available in the operating system, in order to make it impossible to analyze the electrical or runtime behavior of the cards.

Attack and defense: random number generator

The random numbers generated by the smart card are used in authentication to individualize a session, which means to make each session unique and different from all preceding and following sessions. The objective of this is to make it impossible to successfully replay data that have been obtained by tapping a previous session. Another form of the attack would be to have the smart card generate so many random numbers that their sequence becomes predictable. Yet another possibility is to keep requesting random numbers from the smart card until the EEPROM memory of the random number generator no longer works properly, so that the same number is generated over and over again.

Any of these attacks could, if successful, bypass the authentication of the terminal by the smart card. Without exception, they work only with the first generation of smart cards.

They will all fail with modern operating systems. The cycle length of current random number generators is so large that the same random number never appears twice within the lifetime of an individual smart card. It is also no longer of any benefit to generate so many random numbers that problems start to occur with the EEPROM. If this happens, random number generation is simply blocked, so further authentication is prevented.

A high-quality random number generator must meet some additional requirements, such as producing non-predictable random numbers and having a long cycle length (the number of values that are generated before the generator repeats itself). In addition, all smart cards within a particular application must generate different random numbers. This may sound extremely obvious, but problems have repeatedly occurred in the past in this regard! This different behavior is achieved by entering a starting value for the pseudorandom number generator when the smart card is initialized or personalized. This starting value is often called a *seed number*, in allusion to a biological seed that determines the growth of a plant. The design and evaluation criteria for pseudorandom number generators are extensively discussed in Section 4.9, ‘Random Numbers’, along with methods to measure the quality of random numbers.

Protective components of the smart card application

The protective mechanisms of the application are based on suitable mechanisms in the hardware and operating system. The application is dependent on having these two lower levels fully meet their obligations with regard to protection, since it cannot correct for any errors in the hardware or the operating system. For example, if it is possible to read the contents of the EEPROM using an analysis procedure, even the most complicated and secure encryption processes are of no use at all, since the keys can be taken directly from the EEPROM by an attacker. An application must nevertheless be constructed such that the entire system is not compromised in the event of a successful attack on an individual card.

Protection: simple mechanisms

In order to provide effective protection against attacks, all mechanisms of an application should be designed to be as uncomplicated as possible, and they should always conform to the generally applicable principle of 'keep it as simple as possible'. In the first place, this makes implementation easier, and later on, it makes it easier to test the protective mechanisms in order to verify that they are properly implemented and effective. It is extremely dangerous to assume that protection against all possible forms of attack can be obtained by simply making something sufficiently complicated. As a rule, exactly the opposite is true. A common consequence of using complicated processes and mechanisms is that various things are forgotten or overlooked, which makes things that much easier for an attacker.

Fundamentally, the available protective mechanisms in the operating system should always be utilized in the application. They have been tested for reliability, and the defense they provide starts at a lower software level than that of the application. This is not intended to mean that an application does not need to have any protective mechanisms of its own, but the mechanisms already present in the operating system should always be used.

Protection: conservative access privileges

In addition to the principle of 'keep it simple', there is a second generally valid rule. This is that access privileges for the files and commands of a smart card should be granted as conservatively as possible. Access should be generally prohibited, and only allowed if it is absolutely necessary.

The advantages of this approach are that it makes it less likely that access to important data and commands will be granted unintentionally, and it costs an attacker additional effort to obtain each piece of necessary information. This can considerably reduce the attractiveness of an attack, since it increases the overall amount of effort required.

Protection: state machines for command sequences

Attacking a smart card application is considerably more difficult if it is not possible to execute every command at any desired time and an unlimited number of times. This can be realized by using a state machine to specify the allowed sequences of commands. For example, if mutual authentication of the terminal and the smart card is specified as the first required action, an attacker will have to overcome this protective barrier before he or she can execute any further commands.

Protection: redundant access security

The attacker's job is made considerably more difficult if the smart card files are protected not only by access conditions stored in the objects, but also by using a state machine to specify the permitted commands and parameters. With this, the attacker cannot discover the specific features of the system by simply trying each command or combination of commands in turn. If the command sequences are supervised by a state machine, only the commands defined in the application can be executed in the smart card. All other commands will be blocked in principle by the state machine. This considerably reduces the scope of the possibilities available to an attacker with regard to command manipulation.

Protection: various test levels

It has been standard practice for many decades to support various test levels for bank notes. This involves security features that can be independently checked by different groups of people or different types of machines. For instance, many of the visual features, such as security threads and watermarks, can be checked by anyone on the street. For checking at the next level, an ultraviolet lamp is needed to allow the fluorescent pigments in the paper to be seen. The features belonging to the next higher level are used by automated equipment to verify that the notes are genuine. A typical example is the infrared characteristics of the bank note. Yet another level of independent features is provided for tests performed by the central bank.

This concept can easily be transferred to smart cards, with the logical consequence that not everybody or every piece of equipment can test all of the features. For example, a retail terminal for an electronic purse system might contain only some of the keys used for signature verification, rather than all of them. This would not weaken the system in a cryptographic sense, and it would have the advantage that an attacker could not compromise the entire system by learning the master key of a retail terminal. The only entity that would know all the keys in the system required for a complete transaction data set would be the system operator, who would always be able to recognize an attack due to the forged signatures, and who would thus be able to take appropriate countermeasures in case of an attack.

Protection: security features

Features incorporated in the microcontroller can offer additional operational security for smart cards. These features consist of additional functional units that are added to the microcontroller and can be tested by the terminal, along with testing the software in the chip. Both analog and digital components are used for this purpose. The security of these features is based on concealment and is different for each application, which means that the chips are application-specific.

Protection: secure data transmission

There are certain risks associated with transmitting data in an insecure environment. Using relatively simple technical manipulations of the interface between the terminal and the card, it is possible to insert or delete almost any desired data within the normal data steam during a session. If this happens while data related to security are being transmitted, an attacker could derive a benefit from such manipulations.

In order to prevent this type of relatively simple and easily executed attack, a secure messaging method can be employed. However, complete encryption all of transmitted data should be avoided as much as possible, with encryption being reserved for transmitting secret keys and similar items. One reason for not encrypting all of the data relates to data privacy legislation. Almost all information that is written to the memory of a smart card is public. If this information is encrypted, nobody can check what is actually written to or read from the card. In order to avoid any suspicions regarding encrypted data, which in principle would be justified, data should as much as possible remain unencrypted while being transmitted.

Protection: error recovery functions

If a session is prematurely terminated for an undefined reason, or there are fundamental questions regarding an earlier session, it is a major benefit to have application-specific log files in the smart card. Such files are maintained by the operating system, which updates them regularly during the session to reflect the current state of the application and any signatures or other data that may have been received from the terminal. The logged data are located in a cyclic file in which the oldest record is always overwritten each time a new entry is made, causing the content of the oldest record to be lost. For example, if a log file contains 20 records, information regarding the most recent 20 sessions can be stored for subsequent analysis of session history. This information can be used resolve many questions and unambiguously clarify contested transactions and sequences of events.

An reason for maintaining detailed log files in the smart card is the fact that they make certain error recovery functions possible. With a log file, it is possible to automatically restore the previous state of the card (a *roll back*) if a session is terminated in an undefined manner. This would otherwise require analyzing the exact process and sequence of events, which might require human intervention.

Protection: authentication

Unilateral authentication, which is well known due to its use with magnetic-stripe cards, basically amounts to nothing more than verification by the terminal that the card is genuine. A magnetic-stripe card, due to its passive nature, cannot verify the genuineness of a terminal. The introduction of smart cards has fundamentally changed this situation. Now the card can also test whether it has been inserted into a genuine terminal or is connected to a genuine background system. This has extensive consequences with regard to security, since it makes it possible for the card to also take active measures against unauthorized access attempts.

Numerous possibilities arise from the ability of the card and the terminal to perform mutual authentication, but they are usually not exploited to anywhere near their full potential. A smart card should at minimum refuse to allow any access attempts as long as the terminal cannot properly authenticate itself. This would make it impossible to undertake any sort of analysis of the smart card operating system in private, even if only to find out what commands are present.

Protection: online behavior

Terminals with integrated security modules can be used fully autonomously to operate applications using smart cards. Of course, periodic uploads and downloads to and from the background system are still necessary, but they usually occur only infrequently. However, in the case of a relatively large application with a large number of cards in circulation, it must at least be possible for a terminal to quickly make a connection to the background system if necessary, in order to provide direct end-to-end communication between a smart card and the background system. The importance of this increases with the size of the system and the scope of the benefit an attacker can obtain by means of fraud. This is because a direct communications link to the smart card allows the background system to access the current database of the card and block the card if necessary. In addition, the keys stored in the background system are significantly more secure than the keys stored in the many terminals in the field, even if the terminals have security modules. The background system can also produce good statistical evaluations of the card data it receives via sporadic end-to-end links to the smart cards.

All of these arguments are naturally particularly relevant to electronic purses based on smart cards. The ‘urge’ to go online can be triggered by random variables and timing windows stored in the smart card. An equally effective method is to use a counter in the card to demand an online connection with mutual authentication after a certain number of offline transactions have taken place or if the value of the offline transaction exceeds a certain level. At the end of the session, the background system can reset the counter or alter the values of the parameters that control the online behavior of the card.

Protection: blacklists

It is impossible to fully eliminate the possibility of counterfeit smart cards being used in a system, no matter how well the cards may be protected against attacks. A smart card system must also incorporate effective mechanisms to protect users by blocking stolen cards throughout the entire system. The methods used for this purpose are strongly dependent on the application in question and the design of the system, but they can all be reduced to a few basic techniques.

In order to prevent forged or lost smart cards from being used, it is necessary to maintain lists that identify either valid cards or invalid cards by means of some unique feature. This feature is usually a number, such as the card number. From the perspective of impeccable system design, which requires everything that is not explicitly permitted to be implicitly prohibited, a list of valid cards would be best. However, in a large system such a ‘whitelist’ would be awkwardly large and would require very frequent updating. This can be easily illustrated by noting that in a system with 10 million smart cards and an 8-byte card number, the whitelist would contain 80 MB of data.

This is why blacklists are used in practice. A blacklist records all cards that have been blocked. In the example just mentioned, the size of the list would be reduced to 800 KB if the number of blocked cards is 1 % of the total. However, if it is necessary to block significantly more than 1 % of the cards in the system, due to attacks or lost cards, the size of the list would quickly become impractical even with this approach.

In order to further reduce the number of data transfers and the amount of data that must be transferred between the system that maintains the list and the system that tests cards against the list, ‘red lists’ are occasionally used as well. A red list identifies cards that are demonstrably forged and thus should be immediately confiscated or at least blocked for all further transactions. The number of entries in such a list lies in the two- or three-figure range, even in large systems.

Smart cards can be checked against these lists in real time with systems that work online. With systems that work partially or fully offline, updated blacklists and red lists must be transferred to the terminals as often as possible. This should occur at least daily, since a protective mechanism based on a blacklist will otherwise not be effective.

Attack and defense: computer viruses and Trojan horses

Until recently, computer viruses were entirely unknown with smart cards, since there was no technical provision for downloading program code while the card was in use. Modern smart card operating systems, however, have mechanisms that allow program code to be downloaded to smart cards after they have been issued to cardholders, and then executed. This means that in principle, the conditions necessary for the existence of computer viruses in smart cards have been created. By definition, a computer virus is a program that can reproduce itself and thus spread to other computers. If such a program cannot reproduce itself, it is called a Trojan horse. Both types of program have in common that under certain circumstances they can perform unauthorized actions in the host computer. With a smart card, this could involve reading and outputting the values of secret keys.

Unlike the situation with normal PCs, it is not a straightforward task to load a program into the memory of a smart card and then execute it. There are security mechanisms in the card that prevent programs from being run without authorization. For example, some applications may require prior authentication of the terminal. In addition, it is usually necessary to use at least a MAC or a digital signature to load program code into a smart card. Some smart card operating systems also use software or hardware to mutually isolate the memory regions used by individual applications, so that the applications in the smart card cannot affect each other. As a result of these strong security measures, it is unlikely that computer viruses or Trojan horses that are unintentionally downloaded when the card is already in use will be able to impair the functions or security of any applications within the foreseeable future.

Attack and defense: exhaustive key search

One possible type of attack at the cryptographic level is an exhaustive search for a key. For this, the attacker needs a plaintext–ciphertext pair (or better yet, several pairs), and naturally he has to have the appropriate cryptographic algorithm. He or she then encrypts the given plaintext using each possible key in turn until the given ciphertext is obtained. This key can then be tested with all other plaintext–ciphertext pairs on hand. If correct encryption can be performed in each case, the key that has been identified is most likely the correct key. This procedure is basically suitable for all encryption algorithms, although it is not always the fastest method for determining the value of the secret key.

As early as 1993, Michael Wiener published plans for a special computer with a stated cost of one million dollars that could test all DSS keys for a given plaintext–ciphertext pair within seven hours [Wiener 93]. This would allow the value of a 56-bit DES key to be determined in 3.5 hours on average. A few years later, in 1997, the DES key for a plaintext–ciphertext pair provided by RSA Inc was determined in 97 days by systematic searching, using more than 70,000 computers interconnected via the Internet [RSA 97]. The search rate during the final phase of this experiment amounted to around 0.7 % of the DES key space every 24 hours. Another example of the large processing capacity that can be obtained by interconnecting computers via the Internet is the SETI@Home initiative for searching for extraterrestrial life. The EFF 'DES Cracker', which was built as a massively parallel computer in 1998, required only 56 hours to determine an unknown DES key [EFF 98].

In practice, several different approaches are taken to counter such attacks. The simplest and best-known measure is to make the key space of the cryptographic algorithm so large that it is not possible to perform a systematic search within an acceptable length of time, even with very high processing capacity. This is why the DES algorithm has now been replaced by triple DES as a matter of principle. Compared with currently available processing capacity, the key space of the DES has simply become too small.

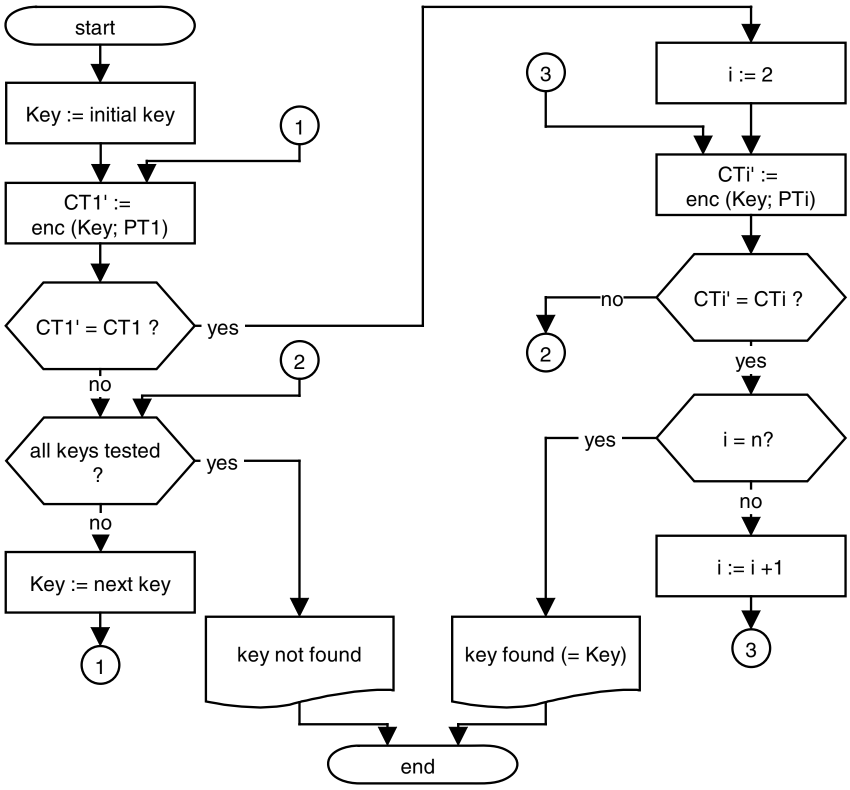


Figure 8.44 A basic procedure for performing an exhaustive search for a key, with a given cryptographic algorithm and several plaintext–ciphertext pairs, according to a proposal by James Massey [Massey 97]. The following abbreviations are used: CT = ciphertext; PT = plaintext; (CTi, PTi) = plaintext–ciphertext pair i; n = number of plaintext–ciphertext pairs

Another defensive measure can be created very easily by constructing the application protocol such that pairs of plaintext and ciphertext do not occur. In smart card applications, in most cases it is not even necessary to encrypt the data, since it is sufficient to secure the data using a MAC. Since the mapping of multiple plaintext blocks onto a MAC is not unique, a brute-force attack using a MAC is a great deal more arduous than the same type of attack using a plaintext–ciphertext pair.

If a random number is prefixed to the plaintext in the smart card (which is called 'salting') and the resulting data are encrypted or used to compute a MAC before being transmitted, the data to be encrypted will be different each time the function is used, so the results will also be different each time. This also makes an exhaustive search more difficult, since in many cases the random number does not have to be public. For example, it could be a secret shared by the security module and the smart card. Incidentally, a random number prefixed to the data to be encrypted within the smart card also provides very good protection against attacks using differential fault analysis (DFA) and power analysis (SPA/DPA), even if the random number is public.

The task of the attacker can also be made more difficult by using dynamic keys (session keys), which are different for each encryption operation. In this case, even if the attacker manages to determine the value of the key by some happy accident, it will not be of any use to him, since the key will have changed again before the next transaction.

9 Quality Assurance and Testing

Quality assurance, with its associated test procedures and methods, is particularly important for smart cards. A smart card manufacturer must fabricate its products in very large numbers at high quality and low cost. In contrast to other branches of the semiconductor industry, these products also contain relatively complicated and sensitive microcontrollers together with software that generally cannot be modified afterwards.

If we compare this situation with that for standard PC software, for example, the basic difference is obvious. In the latter case, it has become standard practice to replace the first release of new software (usually identified by a '0' at the end of the version number) within a short time, ranging from a few weeks to at most one or two months, by revised and improved versions (with version numbers ending in 'a', 'b', 'c' and so on). This would be impossible with smart cards. Their mask-programmed software is by nature unalterable, and it is not feasible to replace a large number of issued cards using any sort of recall campaign. Even with cards that are not used in the particularly sensitive area of financial transactions, such a campaign would cause lasting damage to the reputation of the card issuer, and the costs would be immense.

This is why quality assurance and testing are of fundamental importance in the production of smart cards. After the cards have been manufactured and distributed, it is simply not possible to 'stuff in' an improved version of the software a short time later. This naturally means that a large amount of effort must be expended to produce a product that has as few errors as possible.

With regard to the various tests, a basic distinction must be made between qualification tests and production tests. Qualification tests are used to make a basic decision about whether the smart card in question can be used at all. These tests are usually performed before introducing a new card body, chip, module or operating system. If the new or modified product meets the specified requirements, it is then qualified for production and can be manufactured in large numbers. After this, qualification tests are performed only infrequently on random samples.

A different sort of testing method is used for production tests. These tests can usually be executed quickly without using complex equipment or procedures, in order to meet the inescapable demand of mass production for short turnaround times and high throughput. They primarily involve only simple measurements of general mechanical and electrical parameters, together with sending suitable test commands to the smart card microcontroller.

Smart Card Handbook, Third Edition. W. Rankl and W. Effing
© 2004 John Wiley & Sons, Ltd ISBN: 0-470-85668-8

Many test specifications for large smart card applications are primarily designed with interoperability between smart cards and terminals in mind. A good example is the GSM 11.17 specification, entitled 'Subscriber Identity Module (SIM) Test Specification', which occupies around 100 pages. It describes detailed tests for GSM smart cards, which cover aspects ranging from the card body and general electrical parameters (including the supply voltage and current consumption) to data transmission protocols, commands and files. The GSM 11.17 tests are organized as follows:

- physical characteristics
- electrical signals and transmission protocols
- logical model
- security functions
- functions
- commands
- file contents.

The organization of the individual tests in this specification is equally clear and practical. Each individual test consists of four parts. The first part contains a formal definition of the test and specifies its application. The second part lists the requirements to be satisfied, and the third part describes the objective of the test in detail. The final part specifies the actual test procedure.

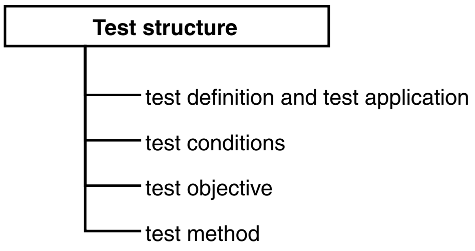


Figure 9.1 Basic organization of a GSM 11.17 test. This structure has been kept fairly general to allow it to be used in principle for all smart card tests

9.1 CARD BODY TESTS

There is presently only one international standard for testing cards with and without chips, which is the ISO/IEC 10373 standard. In Europe, there is also the EN 1292 standard, but this deals exclusively with smart cards and terminals, including their general electrical requirements. Standards relating to cards also often include individual tests and test procedures for checking the properties defined in the standard.

On the following pages, many of the usual tests and verifications for smart cards are briefly described in alphabetical order. The testing laboratories of card manufacturers usually have a repertoire of 120 to 150 different tests for cards.

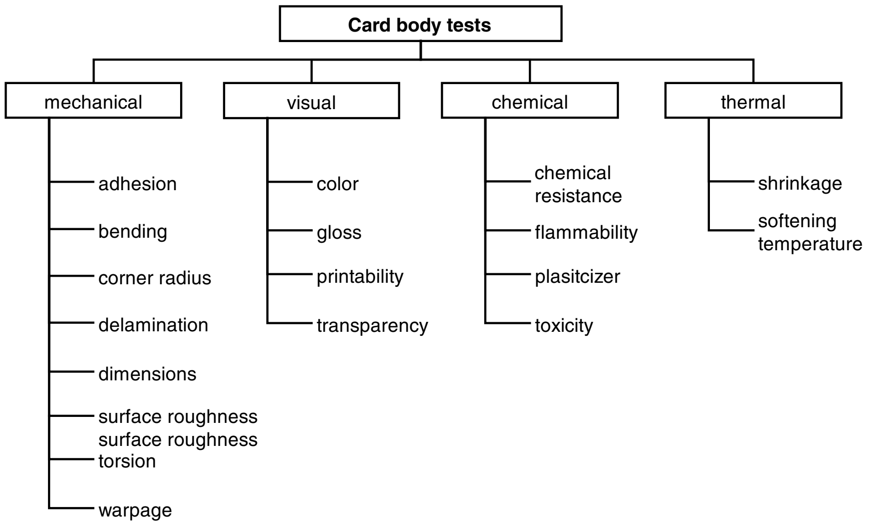


Figure 9.2 Classification of a selection of commonly used card body tests. A series of tests is necessary for each of the individual card components (hologram, magnetic stripe, chip and so on)

Standard ambient conditions are a fundamental requirement for the test environment, which means that a temperature of 23° C ± 3° C and a relative humidity of 40–60 % must be maintained in the test laboratory. The cards to be tested must be appropriately acclimatized to these conditions for at least 24 hours before the actual testing takes place.

Adhesion or blocking

(Basis: ISO 7810; test regulation: ISO/IEC 10373)
This test verifies whether the card's behavior changes when it is stored under certain ambient conditions. Five non-embossed cards are stacked together and uniformly subjected to a pressure of 2.5 kPa at 40° C with 90 % relative humidity, for 48 hours. After this, the cards are inspected for delamination, discoloration, surface changes and other visible changes.

Amplitude measurement

(Basis: ISO 7811-2; test regulation: ISO/IEC 10373)
This measurement verifies the signal amplitude and resolution of the magnetic stripe coding. A standard read/write head that is passed along the magnetic stripe at a precisely specified speed is used to make the measurement.

Bending stiffness

(Basis: ISO 7810; test regulation: ISO/IEC 10373)
In order to determine whether the card has the required bending stiffness, the left-hand side of the card is clamped to a depth of 3 mm with the card facing downwards. The amount of bending

is first measured with no load. A load of 0.7 N is then applied to the outer end of the card, and the difference between the amount of bending under load and the amount of bending with no load is measured. The result indicates the stiffness of the card. The bending stiffness test is often also performed at temperatures lower or higher than the usual testing temperature of 23° C.

Card dimensional stability and warpage with temperature and humidity

(Basis: ISO 7810; test regulation: ISO/IEC 10373)
Both the shape and the size of certain types of plastic change markedly in response to variations in atmospheric humidity. Consequently, the ability of the card to meet the standards must also be tested under these conditions. For this test, the card is placed flat on a surface and the temperature and humidity are varied. The testing conditions are –35° C, +50° C and +25° C at 5 % relative humidity and +25° C at 95 % relative humidity. The size and warping of the card are verified with respect to the standard values after it has been exposed to each of these conditions for 60 minutes.

Card dimensions

(Basis: ISO 7810; test regulation: ISO/IEC 10373)
This test measures the height, width and thickness of a non-embossed card. A force of 2.2 N is applied to the card, and its height and width are measured using a profile projector. For measuring the thickness, the card is divided into four equal rectangles, and the thickness of each rectangle is measured at the center using a micrometer at an applied force of 3.5 N to 5.9 N. The measured maximum and minimum values are compared with the standard thickness.

Card warpage

(Basis: ISO 7810; test regulation: ISO/IEC 10373)
This test measures the amount of warpage of the card. The card is placed on a flat surface and the warpage is measured using a profile projector. This test is primarily intended to be used for cards that are stamped from base material supplied in roll form.

Delamination

(Basis: ISO 7810; test regulation: ISO/IEC 10373)
This test is only meaningful for multilayer cards, which are assembled by laminating several layers of plastic. The cover foil is separated from the core foil at one point using a sharp knife. Starting with this separation, the tester attempts to pull the two laminated foils apart. The necessary force is measured and compared with reference values.

Dynamic bending stress

(Basis: ISO 7816-1; test regulation: ISO/IEC 10373)
The dynamic bending test is illustrated in Figure 9.3. The card is flexed at a rate of 30 times per minute (0.5 Hz) with a deflection f of 2 cm across its length or 1 cm across its width. The

card must remain undamaged after being flexed at least 250 times in each of the four possible directions (a total of 1000 bending cycles).

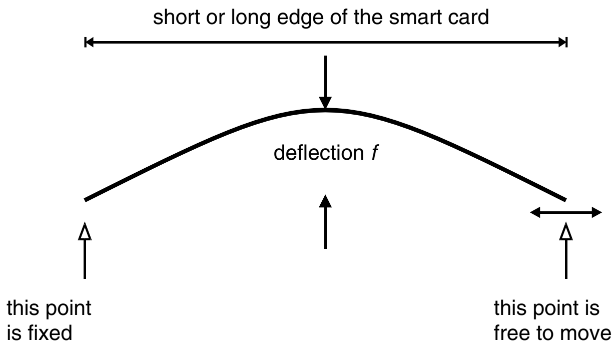

short or long edge of the smart card
deflection *f*
this point is fixed
this point is free to move

Figure 9.3 Schematic diagram of how the card is loaded for the dynamic bending test

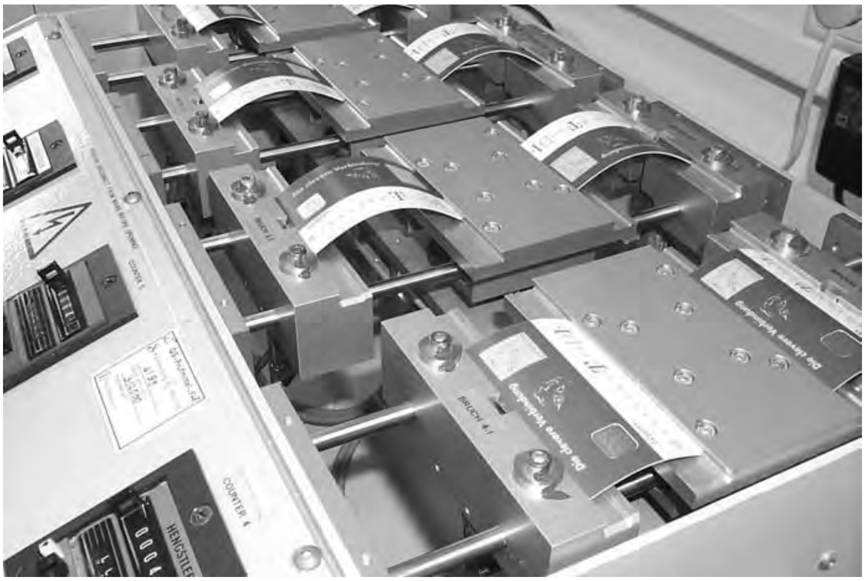


Figure 9.4 A machine for conducting dynamic bending tests on smart cards

Dynamic torsion stress

(Basis: ISO 7816-1; test regulation: ISO/IEC 10373)
In the dynamic torsion test, the card is twisted ±15 degrees about its longitudinal axis at a rate of 30 twists per minute (0.5 Hz). The standard requires 1000 torsion cycles without functional chip failure or visible mechanical damage to the card.

Electrical resistance and impedance of contacts

(Basis: ISO 7816-1/2; test regulation: ISO/IEC 10373)
The electrical resistance of the contacts is an important criterion for the reliability of the supply of electrical power to the microcontroller in the card and data transmission to and from the microcontroller. The resistance is measured using two test probes applied to two opposite corners of the smallest allowable contact rectangle with a force of 0.5 N ± 0.1 N. The resistance between the two test probe contacts, which are gold-plated and rounded to a radius of 0.4 mm, must be less than 0.5 Ω.

Electromagnetic fields

(Basis: ISO 7816-1; test regulation: ISO/IEC 10373)
In this test, the card is moved into a static electromagnetic field with a strength of 1000 Oe (79.6 H) at a maximum speed of 1 cm/s. The memory contents of the card must not change.

Embossing relief height of character

(Basis: ISO 7811-1; test regulation: ISO/IEC 10373)
In this test, the thickness of the card where it is embossed is measured using a micrometer, with an applied force between 3.5 N and 5.9 N.

Flammability

(Basis: ISO 7813; test regulation: ISO/IEC 10373)
The flammability of the card is measured by holding one edge at an angle of 45° in a specified Bunsen burner flame for 30 seconds (diameter 8.5 mm, height 25 mm).

Flux transition spacing variation

(Basis: ISO 7811-2; test regulation: ISO/IEC 10373)
This test determines whether the magnetic flux transitions that encode the individual bits in the magnetic stripe are uniform and sufficiently strong. A read head is passed along the stripe and the field variations are recorded. The measured results are compared with the values specified in ISO 7811-2.

Height and surface profile of the magnetic stripe

(Basis: ISO 7811-2/4/5; test regulation: ISO/IEC 10373)
This test measures the height and uniformity of the surface of the magnetic stripe. It generates a height profile using a special measuring device that is described in detail in the standard.

Light transmittance

(Basis: ISO 7810; test regulation: ISO/IEC 10373)
Some cards have an optical barcode on an embedded foil. This test is suitable for determining the optical transparency of the covering layer and the rest of the card body. One side of the card is illuminated with a light source, and the light transmission is measured on the other side with a detector that is sensitive to 900-nm light.

Location of contacts

(Basis: ISO 7816-2; test regulation: ISO/IEC 10373)
This test is used to measure the locations of the contacts. The card is placed on a flat surface and subjected to a force of $2.2\ \mathrm{N} \pm 0.2\ \mathrm{N}$. Following this, the positions of the contacts relative to the edges of the card are measured using any desired method that has an accuracy of at least 0.5 mm.

Resistance to chemicals

(Basis: ISO 7810, ISO 7811-2; test regulation: ISO/IEC 10373)
The chemical resistance of the card body and the magnetic stripe are investigated using these tests. Different cards are placed in the following precisely specified liquids at a temperature between 20° C and 25° C:

- 5 % aqueous solution of sodium chloride
- 5 % aqueous solution of acetic acid
- 5 % aqueous solution of sodium carbonate
- 60 % aqueous solution of ethyl alcohol
- 10 % aqueous solution of sugar
- gasoline (according to ISO 1817)
- 50 % aqueous solution of ethylene glycol.

Each card is removed from the solution after one minute and either visually examined or tested using a magnetic stripe reader.

Static electricity

(Basis: ISO 7816-1; test regulation: ISO/IEC 10373)
This test, which is only meaningful for smart cards, checks the chip's robustness with regard to electrostatic discharge (ESD). A 100-pF capacitor that has been charged to +1500 V and –1500 V in turn is discharged through a 1500-Ω current-limiting resistor into the chip's various contacts. There must be no damage to the functionality of the chip and no change to the contents of its memory due to the discharges.

Surface profile of contacts

(Basis: ISO 7816-1/2; test regulation: ISO/IEC 10373)
This test compares the surface profile of the individual contacts with the surface of the rest of the card. It is intended to ensure that the contacts lie in approximately the same plane as the overall card surface.

Surface roughness of the magnetic stripe

(Basis: ISO 7811-2; test regulation: ISO/IEC 10373)
The surface roughness of the magnetic stripe is measured using the same device as for the height and surface profile. However, it is used with a special probe tip that allows the surface roughness of the magnetic stripe to be determined. The reason that this test is important is that surface roughness is one of the major factors in the wear of the read/write heads in magnetic-stripe readers.

Ultraviolet light

(Basis: ISO 7816-1; test regulation: ISO/IEC 10373)
Since (E)EPROM memories lose their contents when they are exposed to ultraviolet light, there is a special test to determine whether the smart card is sensitive to ultraviolet light. The card is irradiated for 10–30 minutes by ultraviolet light with a wavelength of 254 nm and an energy density of 15 Ws/cm^2. The (E)EPROM data contents must not change as a result.

Vibration

(Basis and test regulation: ISO/IEC 10373)
Since cards are often subjected to severe vibrations during transport and use (e.g. mobile phones in cars), an appropriate test is also necessary. It requires the card to be tested on a vibration table in each of the three axes with an amplitude not exceeding 1.5 mm over a frequency range of 10 Hz to 500 Hz. The functionality and memory content of the chip must not be adversely affected as a result.

Wear test for magnetic stripe

(Basis: ISO 7811-2; test regulation: ISO/IEC 10373)
In order to determine how the magnetic stripe responds to wear, test data are first written to the stripe. A dummy read/write head with a hardness of 110–130 HV and a radius of curvature of 10 mm is then passed back and forth along the stripe 1000 times with an applied force of 1.5 N. Following this, the data are read again. The signal amplitude must lie within the limits specified in ISO 7811-2.

X-ray test

(Basis: ISO 7816-1; test regulation: ISO/IEC 10373)
The contents of (E)EPROM memory cells can be altered by X-ray irradiation, just as with ultraviolet light. In order to test the X-ray resistance of the memory, the chip is irradiated by Xrays with an energy of 70 kV. The memory contents are then examined for any changes, or the memory is tested to see whether it can still be written.

There are naturally many other things that can be tested, such as the number of insertion cycles, the wear resistance of the inks, the stability of the plasticizer and resistance to perspiration and saliva. Depending on where and how the card will be used, the appropriate tests must be selected and performed.

9.2 MICROCONTROLLER HARDWARE TESTS

Besides ensuring the quality of the card body, one of the primary tasks of quality assurance is to ensure that the microcontroller is in good working order. The microcontroller is the most important and most vulnerable component of a modern smart card.

The CPU and memory are subjected to a variety of tests starting with the semiconductor fabrication stage. In order to allow these tests to be run, every microcontroller has a test ROM containing various programs that support external access to the CPU and memory. In addition, there are sometimes special pads (contacts) that allow free access to the central buses of the processor. During fabrication, needle probes are used to contact the appropriate pads on the chip to allow the necessary test programs to be run. These pads are cut off when the chips are sawn from the wafer, in order to prevent them from later being used for attacks. This means that it is no longer possible to access the internal buses of the chips.

Once the die has been packaged in the module, another test is naturally performed using the module contacts. This often only amounts to performing an activation sequence and seeing whether an ATR can be received. If this is possible, it is assumed that the chip has not suffered any serious damage while being packaged into the module and that all bonding wires are correctly connected. A similar ATR test is also made immediately after the module has been embedded in the body of the card. This test checks whether the module has been damaged by being briefly heated during the embedding process.

The microcontroller is meticulously tested before the smart card is initialized. Test commands that are specifically allowed for this processing step are used.[1] After successful completion of the test program, which lasts between 10 and 100 seconds, these commands are irreversibly blocked against further use. It is possible to perform these time-consuming tests at this stage without reducing throughput by using a large number of initialization machines operating in parallel, so that the duration of the test does not have a significant effect. The tests used here, to give some examples, check whether all EEPROM bytes can be written and again erased and whether the RAM is fully operational. If the chip is scratched during the bonding process, this could prevent some EEPROM cells from being properly written or cause certain regions of the ROM to have incorrect contents.

[1] See also Section 7.11, ‘Commands for Hardware Testing’

Various final tests are performed after the card has been initialized and personalized, depending on the manufacturer. This is usually done using fully automatic, self-calibrating testers that can configure themselves by reading data relevant to the tests from the smart card, following which they carry out the tests accordingly.

In addition to the relatively simple and quickly executable tests undergone by all cards, there are also random sample tests that are only performed on individual cards. These cards, which are taken from regular production, can naturally also be subjected to destructive testing if necessary.

Qualification testing and continuous random sample testing have also been addressed by the publication of the EN 1292 standard. It defines many different test procedures for microcontrollers. Typical sample and qualification tests for microcontrollers are:

- rise and fall times at the I/O contact (EN 1292)
- number of possible write/erase cycles in the EEPROM
- EEPROM data retention
- CLK overfrequency and underfrequency detection
- Vcc overvoltage and undervoltage detection
- I/O contact voltage (EN 1292)
- current consumption at the CLK input (EN 1292)
- current consumption at the reset contact (EN 1292)
- current consumption at the Vcc input (EN 1292)
- current consumption at the Vpp input (EN 1292).

Naturally, every card manufacturer also employs its own supplementary tests to cover special features of the embedded microprocessors. For example, there are special tests for the various sensors on the chip to allow each of them to be suitably tested.

9.3 EVALUATING AND TESTING SOFTWARE

Physical components, such as the bodies and modules of smart cards, can largely be tested using conventional methods. Electrical characteristics can also be measured in a satisfactory manner using automated test equipment. However, the situation with regard to the microcontroller software is somewhat different. Although the methods used to test software for errors have been steadily refined during the past 40 years, since the appearance of the first programs, and there are many recognized good methods for producing programs with a low number of errors, it is still true that in everyday practice, software errors show up relatively frequently.

This is not a serious problem in most applications, since a revised version of the software can quickly be issued to correct the errors. This cannot be done as easily with smart cards, since most of the software is located in the ROM of the microcontroller. A new version of the software necessitates a completely new production run by the semiconductor manufacturer, which takes around 8–12 weeks. If the smart cards have already been put into service, it is practically impossible to modify the existing software. It follows from these very strict constraints that software for smart card microcontrollers must have an extremely low number of errors. Software that is truly 'error-free' would be even better, but given the present state of software development, this remains a distant goal.

As is well known, the subject of software testing is extremely extensive. It is described in many books in all of its variations and orientations. We can only present a short sketch of this subject, which by now has become almost an independent branch of information technology. Consequently, in the following sections we discuss only certain special aspects of testing software for smart card microcontrollers. Glenford J. Myers' book [Myers 95] can be considered to be representative of the literature on this subject. We would also like to point out that military standards, in particular, contain many good and well-proven methods for generating and testing software.

9.3.1 Evaluation

Due to their ability to store data securely, smart cards are primarily employed in security-sensitive areas. However, smart cards can be used to advantage not only for the secure storage of data, but also equally well for the secure execution of cryptographic algorithms.

The field of electronic payments, in particular, is an expanding market for smart cards. Since enormous amounts of money flow in a widely distributed system, the application provider or card issuer must have a high degree of confidence in the semiconductor manufacturer, the producer of the operating system and the smart card personalizer. The application provider must be able to be absolutely certain that the software in the smart card performs the required financial transactions without any errors and that the software is free of security leaks, not to mention trapdoors deliberately introduced into the software.

For example, suppose a secret command could be sent to the smart card to read out the PIN and all secret keys. In the case of a GSM or Eurocheque card, the attacker would then be able to clone any number of cards and sell them in perfect working order.

These security requirements relate not only to manufacturing the smart cards, but equally well to initialization and personalization of the cards, since the secret keys and PIN are loaded into the cards in these stages. The card issuer must place a high degree of trust in the card provider with regard to security.

This also applies to the fundamental security of the software in the smart card. Problems can arise even if a 'trap door' has not been intentionally included in the software to allow data to be spied out of the card. Faulty operation of the software could very well make it possible to read data from the card or write data to the card using a combination of commands that is not used in normal processes. Although the likelihood of such a coincidence is extremely low, it is nevertheless well known that given the current state of software technology, it is impossible to guarantee that programs are free of errors under all conditions. It is certain that in the future, companies that produce software for smart cards will no longer be able to deny all responsibility on the basis of such legalistic formulations.

There are only two ways in which the application provider can test the trustworthiness of a product. He can either test all possible variations the smart card software himself, or he can have the software tested by a trustworthy party. The first option is frequently possible only to a limited degree, since the provider usually does not have all the necessary technical expertise and capabilities. The second option, which is assigning the tests to another party, is currently regarded by all concerned as an acceptable solution.

This same problem has existed for many years with software and systems developed for military use. It is thus not something that is new in the smart card world. In order to establish metrics for the trustworthiness of software products, which means to make it objectively measurable, the US National Computer Security Center (NCSC) issued a catalog of criteria for evaluating the trustworthiness of information technology systems in 1983. NCSC was founded in 1981 by the American Department of Defense (DoD). The publication of 'Trusted Computer System Evaluation Criteria' (TCSEC) followed in 1985. This book had an orange binding, so it has come to be generally known as the 'Orange Book'. These criteria serve as guidelines to the NCSC for the certification of information technology systems.

The TCSEC has become an international model for practically all criteria catalogs in the information technology field. In Europe, specifically European criteria have been defined, although they are based on the TCSEC. They were first published in 1990 as the 'Information Technique System Evaluation Criteria' (ITSEC), and a revised version was issued in 1991.

The Common Criteria (CC) were created in order to provide a uniform standard for testing the correctness of software. They can be regarded as representing the essential elements of the TCSEC and the ITSEC. The Common Criteria are also better organized for the evaluation of software than the TCSEC or the ITSEC. Although the first version of the Common Criteria was published as early as 1996, it has not yet supplanted the TCSEC or the ITSEC.[2] The Common Criteria have also been published as an international standard (ISO 15408). In contrast to the ITSEC, which has six levels, the Common Criteria have seven levels of trustworthiness. It is relatively easy to make the transition from an evaluation based on the TCSEC or the ITSEC to one based on the Common Criteria, since all of these catalogs have many features in common. However, since in the smart card field in particular the ITSEC is still used as the essential basis for software evaluation, we refer only to this catalog in the following description.

Occasionally, the requirements of the FIPS 140-2 standard are taken into account in performing evaluations, in addition to the ITSEC and the CC. This standard specifies four possible security levels for security modules, which can be considered to include smart cards, and provides detailed descriptions of seven requirement areas related to security. The contents of this standard are very practically oriented and also deal with details of technical implementation, such as criteria for the quality of random-number generators.

Regardless of the method used, an evaluation process has four characteristics. First, it must be unbiased, which means that the evaluator must not have any preconceived ideas regarding the item to be evaluated or its producer. The second characteristic is that the evaluation process must be objective and structured to minimize the significance of personal opinions. The third characteristic is that the same result must be obtained if the evaluation process is repeated. The final characteristic is that the evaluation process must be reproducible, which means that a different tester or testing agency must reach the same conclusions.

One of the most important considerations in any evaluation is defining the security targets for the target of evaluation (TOE). The target of the evaluation is the object to be tested,

[2] The TCSEC, ITSEC and CC are available at no charge from many Internet sites (e.g., the CC at [NIST])

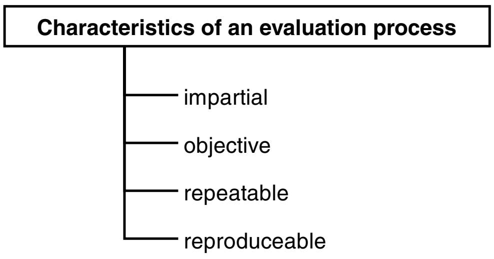


Figure 9.5 The four characteristics of an evaluation process

and the security targets describe the mechanisms to be tested. Incidentally, an evaluation can be dramatically simplified by carefully selecting the security targets, since elements that are critical with regard to security can thereby be excluded. This is just a trick that can be used to achieve a high evaluation level in the quickest and least costly possible manner. Naturally, the actual security can only suffer as a result.

The ITSEC

Since the ITSEC is supposed to be valid for all possible information technology systems, and the document is only around 150 pages long, the security criteria must be described in a very abstract form. It is consequently very difficult to read and, like legislation, it occasionally requires outright interpretation.

The ITSEC is based on the idea that there are three fundamental threats to any system, which are unauthorized access to data (breach of confidentiality), unauthorized alteration of data (breach of integrity) and unauthorized impairment of functionality (breach of availability). The security criteria are based on these three threats.

The threats can also be diminished outside the system by various measures. For example, an insecure computer system can be made considerably more secure if physical access to the system is controlled by security gates. Such aspects represent general conditions that also must be taken into consideration during an evaluation. The resulting residual threat can then be assessed using the ITSEC criteria catalog.

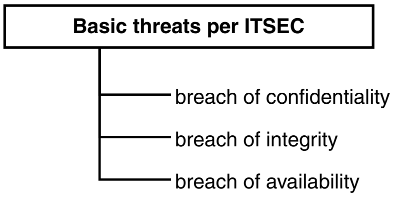


Figure 9.6 The three basic threat areas according to the ITSEC

The basic procedure for evaluating a system in terms of the ITSEC is to rate the mechanisms that it uses to maintain security with regard to the three defined basic threats. The ratings are made in a manner that is similar to grading a school assignment. The mechanisms used must be both correct and effective. The mechanisms are also judged with regard to basic functions that are relevant to security, which are also referred to as 'generic categories'.

Table 9.1 Generic categories of mechanisms according to the ITSEC

Category
Identification and authentication
Access control
Securing evidence
Log evaluation
Reprocessing
Genuineness
Reliability of service
Transmission security

Table 9.2 lists the requirements for evaluating software for smart cards. The amount of effort does not increase linearly for each successive level, but instead nearly quadratically.

Table 9.2 Summary of the requirements for smart card software development as a function of the ITSEC quality level

Necessary information	Evaluation level					
	E1	E2	E3	E4	E5	E6
Security targets	yes	yes	yes	yes	yes	yes
Formal model of security policy	no	no	no	yes	yes	yes
Descriptions of functions	informal	informal	informal	informal & semi-formal	informal & semi-formal	informal & formal
Architectural design	informal	informal	informal	semi-formal	semi-formal	formal
Detailed design	no	no	informal	semi-formal	semi-formal	semi-formal
Implementation	nothing required	nothing required	nothing required	source code testing	source code testing	source code & object code testing
Operation	yes	yes	yes	yes	yes	yes
Level of rigor	– state description –		– presentation –		– explanation –	

This means that it takes twice as much effort to go from level E2 to level E3 as it does to go from level E1 to level E2. The consequences of this are naturally most pronounced for evaluation levels E4 through E6. Complete evaluation of a medium-sized smart card operating system at the ITSEC E6 level can easily take several years and cost several million euros.

In an evaluation, a fundamental distinction is made among informal, semi-formal and formal methods. An informal description of a function can perfectly well be compared with a textual description in this book. The Small-OS operating system presented in Section 5.11 is basically described semi-formally. A formal description, by contrast, can be logically tested and is produced using a formal notation (such as predicate logic).

In the ITSEC, the levels of security against threats are divided into several classes by function. Each of these classes is a collection of a particular combination of security functions, and it indicates the level of threat for which the system has a secure defense. There are six quality levels or evaluation levels (E1 through E6). E1 is the lowest level, while E6 is the highest level.

In all of the functional classes, the formal requirements imposed on the development process and the development environment are prescribed in a very abstract manner. Furthermore, the functional classes contain information and specifications regarding operating documentation and the eventual operating environment. This information is presented in a form that can be applied to all possible technical variations of software development.

An ITSEC evaluation frequently includes an appendix that relates to the 'strength' of the mechanisms, which means their robustness against attacks. Three levels of strength are defined: low, medium and high. 'Low' characterizes protection against random, unintentional ingress into a secure environment. 'Medium' means that there is protection against attackers having limited resources. The top level, 'high', means that there is protection against attackers having very good technical knowledge and resources. Normally, the highest level of mechanism strength is used with ITSEC level E4 and higher. It should also be used as a matter of principle for smart cards, in the interest of system security.

The ZKA criteria

In addition to the international evaluation standards such as TCSEC, ITSEC and CC, there are also specific evaluation standards that have become established for large smart card applications. Two examples are the Visa criteria for security testing and the German ZKA criteria. The ZKA criteria are mandatory in Germany for Eurocheque cards with chips, and all smart card operating systems for this application are tested against these criteria by authorized evaluation bodies. The ZKA criteria are listed and explained in Table 9.3.

In testing based on the ZKA criteria, the prepared documentation as well as the software and hardware are reviewed. This is the main advantage of the smart card-specific ZKA criteria in comparison with the general TCSEC, ITSEC and CC criteria, which apply to all types of software. Tables 9.3 and 9.4 list the 13 ZKA criteria along with associated explanations.

In summary, the ZKA criteria presently probably offer the best security for smart cards, since they meet the stringent demands of a large financial transaction system and are specially tailored to the particular interests of a system based on smart cards.

Table 9.3 The ZKA criteria for evaluating a smart card system (from Stefan Rother [Rother 98b])

Criterion	Explanation
Component authentication	It must be possible to authenticate the security components of the system.
Message integrity	Information relevant to security that is exchanged between the components must be protected against manipulation.
User authentication	Certain functions may be executed only after the user's PIN has been correctly entered in order to authenticate the user.
PIN and key secrecy	PINs and keys may never be transferred in plaintext outside of a secure area. To the extent that the PINs and keys are processed or stored in components of the operating system, these components must be protected against unauthorized reading or modification. The system must prevent carrying out an exhaustive PIN search.
Logging	All events relevant to security must be logged in the components that are involved. The logs must be protected against manipulation.
Key management	There must be mechanisms for the distribution, administration and exchange of keys. Only temporary keys may be used with symmetric cryptographic procedures, and keys must be kept separate according to their purposes.
Hardware	All operations relevant to security must be protected against unauthorized accesses.
Organizational precautions relating to manufacturing and personalization	Only the evaluated program code with the described parameter values may be used in actual operation.
Process security	The correctness of the processes and data flows within the evaluated programs is ensured by a source code test.
Other applications	Additional applications must not have any effects that endanger the security.
Encryption methods	The cryptographic algorithms that are used must represent the current state of the technology and are not allowed to be based maintaining the secrecy of the method.
Unambiguous representation	Each security component must be clearly identifiable in the overall system.
Personnel requirement	Only suitable persons should be allowed access to the components and mechanisms that are relevant to security.

Table 9.4 Investigating the security of the specifications and source code of a smart card operating system based on the ZKA criteria (from Stefan Rother [Rother 98a])

Investigation of specifications and source code	
Operating system commands	Encryption and decryption
Application commands	Key derivation
Communications mechanisms	Signature generation and testing
Memory and resource management	Key administration
File attributes and access privileges	Authentication mechanisms
Checksum algorithms	Random number generator

Summary

The procedure with a real smart card project is as follows. The card issuer first works with the operating system producer to define the operational requirements and the threats that must be taken into account. Following this, an agreement is reached regarding the necessary evaluation level. Next, the operating system is configured according to this evaluation level and the corresponding evaluation process, and the necessary documentation is provided. In the final step, the completed operating system with all its components can be evaluated by a suitable independent organization, which is the evaluation body, according to the previously agreed level.

An ITSEC evaluation has certain benefits for the card issuer and application provider, since they can be assured that the design and implementation of many security aspects are both clearly defined and effective. However, in the dynamic smart card market this assurance comes at the price of certain drawbacks. Software development time is considerably increased by the need to conduct an evaluation, even at the lowest evaluation levels. The additional effort needed to produce the necessary documentation also increases development costs, which ultimately must be reflected in the price that the producer of the operating system charges for its product.

However, the major disadvantage is something completely different. Even with an evaluated system, it is not possible to guarantee that all processes and mechanisms function exactly as they are described in the relevant documentation. Evaluation does not mean that the party performing the evaluation completely tests the product, but only that it reviews the documentation, and possibly the source and object code, that it has received for the product in question. It is equally important for the target hardware to guarantee a level of security equivalent to that of the evaluation. There is no benefit in having error-free, secure software if it is possible to use the hardware as a 'back door' to bypass the software.

These four reservations with regard to evaluation must be kept in mind and carefully considered in each individual case.

9.3.2 Test methods for software

There is one thing that should be clearly stated at the beginning with regard to software testing, even though it may sound obvious. This is that testing a program does not mean conducting a wholesale search for errors in the program, but instead performing tests according to a test specification, with the objective of discovering whether there are errors in the program being tested. Testing thus has nothing to do with debugging, since tests are performed by testers and debugging is performed by software developers.

As we have repeatedly emphasized elsewhere, programs for smart card microcontrollers are not very large compared with other types of software. Nevertheless, they have their own special features. This can be illustrated by considering some of the general specifications of a typical operating system.

In a microcontroller having 16 kB of ROM and 8 kB of EEPROM, the software requires around 20 kB of memory. This leaves 4 kB of EEPROM available for any applications. If this software is programmed in assembler, it will amount to around 30,000 lines of source code, which would fill 500 sheets of paper if printed at 60 lines to the page. The number of conditional branches will be around 2000, and even an experienced programmer needs around nine months to generate 20 kB of assembler code. Presently, very high-performance smart card operating systems occupy up to 12,000 lines of source code in C, which corresponds to approximately 2000 sheets of paper.[3]

This numerical example clearly demonstrates that smart card operating system software is rather complex. On top of this, the software must be used almost exclusively in areas where security is a consideration. This means that the demand for a low level of errors cannot be met by simply employing a few homemade tests during or following software development. Instead, a suitable testing strategy is necessary.

With smart cards, as in other areas, the trend is shifting from assembly language programming to high-level programming languages, such as the C language, which is relatively close to the hardware level, and Java, which is an object-oriented language. Using a high-level programming language is necessary for large smart card operating systems with more than 30 kB of code, not only due to the resulting decrease in implementation time, but also because of the need to minimize the number of errors.

This can be illustrated as follows. It can be assumed that the number of errors per line of source code is nearly the same for almost all programming languages. Since the functional level of a high-level language is significantly higher than that of assembler, this means that the error density is lower for the same amount of executable program code.

For example, if we assume the entirely realistic value of 1.5 errors for every 100 lines of program code, and in addition we assume that only half of all errors can be found with an acceptable amount of effort, then a tested program will still have 0.75 errors for every 100 lines of program code. With Java, the relationship between lines of source code and machine code (bytecode) is around 1:6. This means that the functionality of one line of Java code roughly corresponds to that of six lines of machine code. If we assume that all other conditions are the same and that the compilation process is largely error-free, the number of errors in a program can be reduced by a factor of six by using a more powerful language. Even if the actual value is lower than this in practice, this is still an exceptionally strong justification for using powerful programming languages.

9.3.2.1 Fundamentals of smart card software testing

It is necessary to consider the life cycle of smart card software before even starting to define a test strategy. The waterfall model proposed by W. W. Royce, which has been known since 1970 and has been published in many forms, can be used for this purpose. It is relatively well suited to mask-programmed smart card operating systems. However, since it is also designed to be used with very extensive software projects for PCs and mainframe computers, here we use a simplified version specially adapted to smart cards.[4]

The five steps described here are normally performed in sequence. However, it is certainly possible for a problem encountered in a certain step to make it necessary to go back and repeat one or more steps. This should be avoided as much as possible, since each iteration costs time and money.

In order to meet economic demands, such as time-to-market and short software development time, it is often necessary to overlap the steps to a certain extent, instead of performing them

[3] See also Section 5.1, ‘Historical Evolution of Smart Card Operating Systems’
[4] An extensive description of the life-cycle model usually used for the development of smart card operating systems and applications can be found in Section 15.7, ‘Life-Cycle Models’

in strict sequence. With this method, which is known as simultaneous engineering, sections of the software are split into individual modules as early as possible. These modules are then launched down the waterfall concurrently. It thus can happen that smart card software containing only a data transmission protocol may already be at the system integration level while the cryptographic algorithm for the same application is still being specified.

Analysis

The analysis stage includes establishing the basic definition of the objective and compiling the requirements in the form of a formal requirements specification document that defines all the requirements that must be satisfied by the smart card software to be developed. The analysis stage also allows for the generation of proposed solutions in the form of preliminary designs. Put simply, this stage defines *what* the finished software has to do.

Design

Analysis is followed by the design stage, which establishes *how* the software will do its job. For this purpose, it is necessary to generate precise specifications that are not subject to interpretation and that completely define one of the various possible solutions to the requirements produced in the analysis stage. Formally constructed specifications are best, since they allow the features, functions and processes of the software to be clearly and unequivocally defined. Specifications written in pseudocode, which can be tested for consistency and freedom from errors using computer programs, are well suited to this purpose.

In a computer-aided software engineering (CASE) environment, such specifications can be used directly to generate the source code and test programs. They are sometimes referred to as 'executable specifications', which means specifications produced in a form that can be interpreted and further processed by computers.

Implementation and test

Once the specifications have been finalized and accepted, the program flowcharts for assembler or C programming may be generated. This is followed by programming and associated testing. The outcome of this stage is a fully programmed and tested smart card operating system.

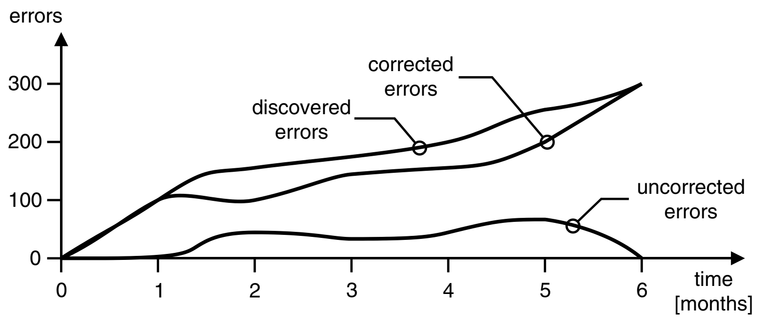


Figure 9.7 Typical curves of discovered, corrected and uncorrected errors in the course of a typical development project for a smart card operating system

System integration

Since smart cards can only function as part of a larger system, the various system components must be integrated in this stage. The results of system integration are the complete and error-free interaction of all parts of the system and the final documentation for the entire system.

Maintenance

This final stage of software development can only be used to modify any general parameters located in issued cards. Large-scale software upgrades or modifications are no longer possible at this stage.

In the future, the ability to easily and quickly program smart cards using a high-level language such as Java will make it possible to use evolutionary life-cycle models[5] in addition to the traditional waterfall model. With an evolutionary model, the analysis, design, and implementation and test stages, and in part even the system integration stage, are iterated several times with increasingly improved results. The objective is to quickly arrive at an optimum solution by expending minimal effort on generating specifications and working with fully functional prototypes.

It is a well-known fact of long standing that in all kinds of projects, and particularly in the case of software development, the cost of correcting an error increases as the project progresses. This fact should lead to the expenditure of an appropriate amount of time and effort in the initial stages of the project, as represented by the waterfall model. If the design is incomplete or the specification is faulty, the cost of remedying the problem rises exponentially in the subsequent stages of the project.

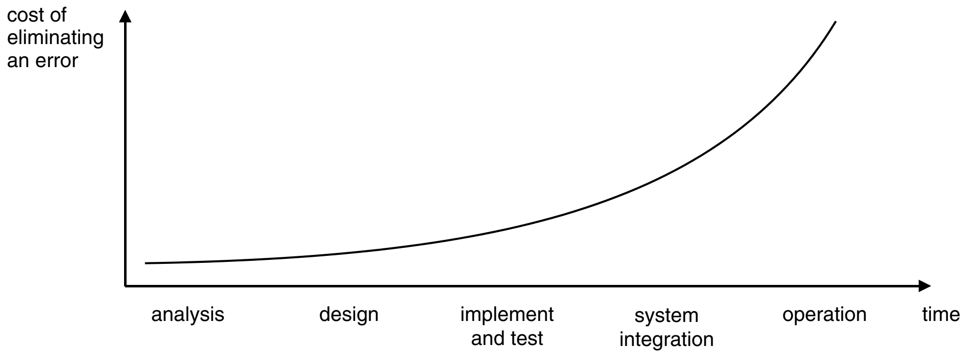


Figure 9.8 The cost of correcting an error as a function of the time when it is discovered

9.3.2.2 Test procedures and test strategies

Nowadays it is impossible to keep track of all the different methods and procedures that are available for testing software. However, only a few well-proven methods are necessary

[5] See also Section 15.7, 'Life-Cycle Models'

for testing smart card programs. It is possible to draw on decades of experience and a large number of publications on the subject of testing. Incidentally, software testing always means attempting to discover errors in the program, not demonstrating that the program is correct.

All test procedures can be divided into static and dynamic types. In a static procedure, the program code is analyzed and evaluated using various methods, either manually or automatically. The two most commonly used static testing methods are program assessment and review, which are briefly described and explained below.

Static program assessment using software tools

This consists of analyzing various properties of the program code using static techniques. The properties that can be analyzed include the following:

- number of lines of code (LOC)
- number of lines of comments
- ratio of the amount of comments to the amount of program code
- structure of the program code
- number of functions
- nesting depth
- ‘dead’ code

Review

Review consists of the formal analysis and evaluation of program modules by a team of assessors. This is sometimes referred to as a ‘code walkthrough’ or a ‘code inspection’.

In contrast to static methods, dynamic program analysis methods test the program while it is in operation, either manually or with the aid of computers. There are two fundamentally different approaches (blackbox and whitebox testing), plus a third, hybrid approach (graybox testing).

Blackbox test

A blackbox test is based on the idea that tester knows nothing about the internal processes, functions and mechanisms of the program to be tested. This means that all that can be done is to examine the input and output data with regard to their relationship to each other, as defined in the specifications.

Blackbox tests are the standard for smart card operating systems. They are also used for security modules for terminals and computer systems. However, it is often incorrectly assumed that these tests can discover Trojan horses or similar items that may be present, in addition to errors in the software. This assumption is used as an argument for dispensing with relatively time-consuming and expensive program code analysis. Although a blackbox text may allow the tester to detect simple, unsophisticated trapdoors programmed into the system or ones that have been inadvertently generated, an experienced programmer can easily create access

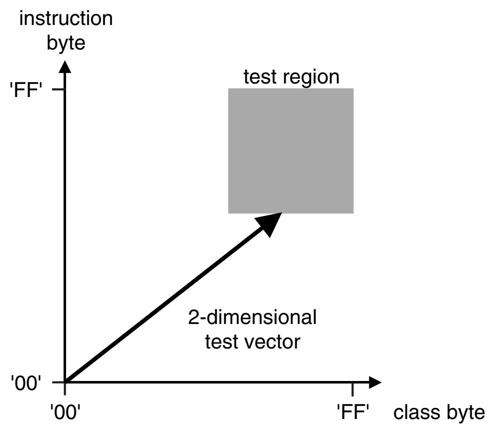


Figure 9.9 Example of a two-dimensional test vector for a blackbox test, which defines a specific test region by means of equivalence classes. Test vectors for complex test cases can easily have 10 or more dimensions. Due to the limited amount of time available for testing, the test region or regions must therefore be suitably restricted

possibilities that can never be detected by a blackbox test. This can be illustrated using the following simple example. It is not meant to serve as a model for a Trojan horse, since this has already been known for a long time, but rather to enhance the awareness of the necessity of code inspections in security analyses.

Almost all smart card operating systems contain a command for generating and issuing random numbers (GET CHALLENGE). This command could be modified such that only the first 8-byte number that it issues is actually generated by the pseudorandom number generator. Each of the subsequent 'random' numbers would then consist of an 8-byte value taken from the EEPROM and XORed with the first random number. A simple external program could then be used to read out the entire memory contents, including all the keys. Incidentally, this is a very good example of applied steganography in smart cards.

With a blackbox test, there is no way to determine whether a Trojan horse is concealed behind this command. Even a statistical analysis of the random numbers obtained would not detect any significant deviation from the normal pseudorandom numbers. The only way to recognize such a manipulated program is to inspect the entire code of the operating system. This example illustrates only one of many possible ways in which a normal command can be modified in order to obtain the contents of the memory. Since only a few lines of program code are needed for this modification, the only effective way to combat such a possibility is to completely reveal and analyze the source code.

Abort tests are used to test the functional viability of atomic operations. Such tests are also called 'recovery tests'. In such a test, a suitable command is sent to the smart card to cause an atomic operation to be initiated in the card. While the atomic operation is being executed, power to the card is interrupted at a specific time. Following this, a check is made to see whether the processed data have been maintained a consistent state by the atomic operation. In such tests, power is not just interrupted at one particular time. Instead, the test is run may times with the power being interrupted at various times distributed over the entire duration of the atomic operation. In order to obtain valid test results, the abort time is displaced in steps, each of which is approximately equal to half of the EEPROM write time for a single page. The functionality of atomic operations can be tested very well in this manner. However, the number of tests required is fairly large. If a typical time increment of 1 ms is used, thoroughly testing command processing in a smart card over an interval of 100 ms would require 100 tests.

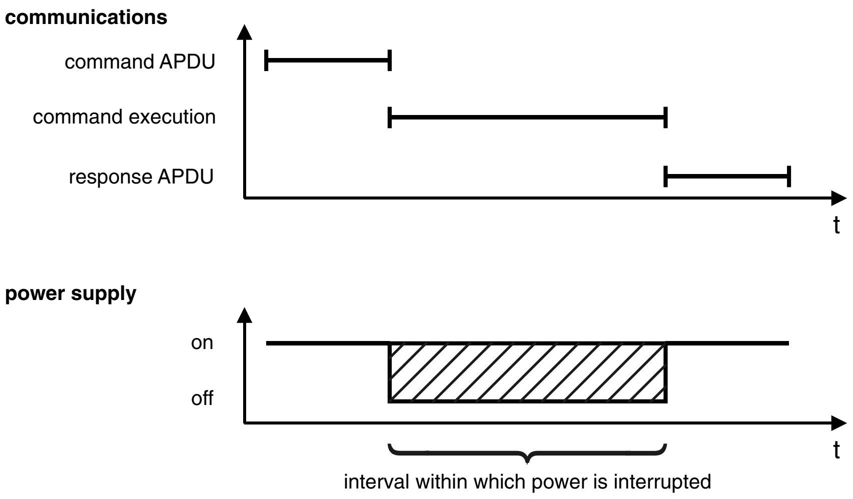


Figure 9.10 Timing diagram of an abort test for testing an atomic operation

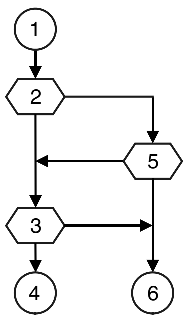
Whitebox test

A whitebox test is often called a ‘glassbox test’, which clearly describes the concept. With this type of testing, all internal data structures and processes are known to the tester and can be completely understood. The relevant program documentation is used to design and generate the tests, but the specification is always the sole authority. For decades, program flowcharts and Nassi–Schneidemann diagrams (structograms) have commonly been used to document programs, and they also form the basis for evaluating the internal functions of the software in a whitebox test. With object-oriented languages such as Java, the unified modeling language (UML) has become the prevalent form of representation. The various description variants of UML are also very well suited to the architectural description of smart card software.

Since the exact program sequences are known, it is natural for the tester to want to test all possible execution paths through the software. There are several ways to do this. One of them is statement coverage, in which every instruction in the program is executed at least once. This makes it very easy to discover whether the program contains dead code, which is code that is never used, but it is not capable of ensuring that the desired functionality is present. A better method for this is decision coverage, which involves traversing all decision nodes in the program code at least once in each of their possible options.

In order to be able to recognize internal program processes during dynamic testing, it is necessary to have a sophisticated emulator for the smart card microcontroller in question or to ‘instrument’ the program being tested. An instrumented program has special program code inserted just before every jump instruction, branch instruction and function call. This code collects location and parameter information when the program is run. An analysis program can be used to statistically and graphically evaluate this information. Unfortunately, the additional program code alters the timing relationships of the program, and in the worst case, it can even cause the behavior of the program to change. This must be borne in mind whenever this technique is used.

An extension of the decision coverage criterion is to traverse all program decisions in all possible combinations once for each combination. This covers all possible execution paths. However, the limited amount of time available for testing means that this is possible only with very small programs consisting of a few hundred bytes of code. Even with programs on the order of 1000 bytes, it is not possible to test all possible combinations in a reasonable length of time.



instruction coverage	1-2-3-4
	1-2-5-6
decision coverage	1-2-3-4
	1-2-5-6
	1-2-5-3-4
	1-2-3-6
decision coverage in all combinations	1-2-3-4
	1-2-5-6
	1-2-5-3-4
	1-2-5-3-6
	1-2-3-6

Figure 9.11 Example of the number of possible execution paths through a program flowchart, for testing using statement coverage and decision coverage

Table 9.5 clearly illustrates this in summary form, using a typical smart card command interpreter as an example. The function of this program module is to identify a command located in the card's input buffer by means of the class and instruction bytes, and then to check the P1, P2, Lc and Le parameters. This size of the program code for this routine is around 200 bytes, and it contains 18 branches. The possible output values consist of five return codes and calls to 26 different command procedures.

Two other path coverage criteria are used in particular for testing smart card operating systems: input coverage and output coverage. The objective is to generate all possible input and output values. The output values are often restricted to the available return codes, since otherwise the number of variations would be too large.

Since the number of possible input values can also quickly reach a magnitude that makes testing impractical, due to the multitude of input values or the amount of time required, equivalence classes are usually employed. This reduces the large number of possible input values to a relatively small number that can be tested in a reasonable length of time. Equivalence classes are formed by selecting boundary cases on either side of the decision range, together with a value in the middle of the range. For example, if the smart card command interpreter allows a range of 20 through 50 for the value of the P1 byte, the equivalence class would be formed using the values 19, 20, 50 and 51 for the boundary values and 35 (for example) as the midrange value. This set of values verifies the essential query conditions of the program. After

Table 9.5 The number of possible test cases for various coverage methods in a whitebox test. A 200-byte smart card command interpreter is used for this example. An average processing time of 30 ms, including data transmission, was used to calculate the test durations

Coverage method	Possible test cases	Test duration
One million random input values	1,000,000	≈8 h
Command coverage	10	≈0.3 s
Decision coverage	50	≈1.5 s
Decision coverage in all possible variations	50,000,000	≈17 days
Input coverage in all possible variations (5-byte header)	$\approx 1.1 \times 10^{11}$	≈1000 years
Input coverage with equivalent classes	15	≈0.5 s
Output coverage of the return codes	6	≈0.2 s

this test, it could be assumed with a relatively high level of confidence that parameter range checking has been correctly implemented.

Particularly with assembly language programming, it is unfortunately necessary to take the properties of the target hardware into account when defining the equivalence classes. For instance, all arithmetic operations that can cause an overflow or underflow in the processor due to the architecture of the arithmetic unit (8-, 16- or 32-bit width) must be taken into account when forming the equivalence classes. Only then is it possible to be sure that underflows and overflows are correctly handled in the program.

Whitebox tests are often used for module testing during smart card development. In such tests, finished software modules located in smart cards are fed data from outside using special test commands, following which the results of the actions of the software modules are determined from outside using test commands. The actual results are then compared with the expected results outside the card.

Graybox test

A graybox test represents a hybrid combination of blackbox and whitebox tests. With such a test, only some parts of the software are known, such as internal program processes. Graybox tests are primarily used in the integration phase with smart cards, since they allows errors in the interaction of the individual components to be very quickly and effectively detected and corrected. Naturally, appropriate test keys (which are public) are needed from the key management facility. Once this part of the integration tests is successfully concluded, the results can be checked using the real keys (life keys).

9.3.3 Dynamic testing of operating systems and applications

It is important to realize from the start that program testing can only demonstrate the presence of errors, not their absence. If we assume that only roughly half of the errors that are present are actually found, it can be safely concluded that an average program still contains a number of weaknesses.

In practice, an error rate of 0.7 per 100 lines of code, after testing, is often assumed for assembly-language programming. If we take the previously mentioned value of 30,000 lines of source code for a smart card operating system as representative and subtract two-thirds of these as comments, we can calculate that there are still around 70 errors in a fully tested and released operating system. In the areas of military and medical technology, where security is critical, it is assumed that there are still four undiscovered errors for every 10,000 lines of source code, despite the tremendous amount of effort expended on testing and quality assurance [Thaller 93]. Although most undiscovered errors will never manifest themselves, in the right circumstances a single error is sufficient to bypass all the security barriers of a smart card operating system. It is highly beneficial to always bear this in mind as a motivation for careful and well-considered testing.

Of course, there are natural limits to testing. Particularly in commercial projects, in contrast to research projects, the amount of time available and the maximum affordable cost are strongly limiting factors. In addition, testing becomes increasing more difficult and demanding as the number of errors present in the program decreases. The search for the last few errors must at some point come to an end, since the time and resources that can be expended on it are fundamentally limited.

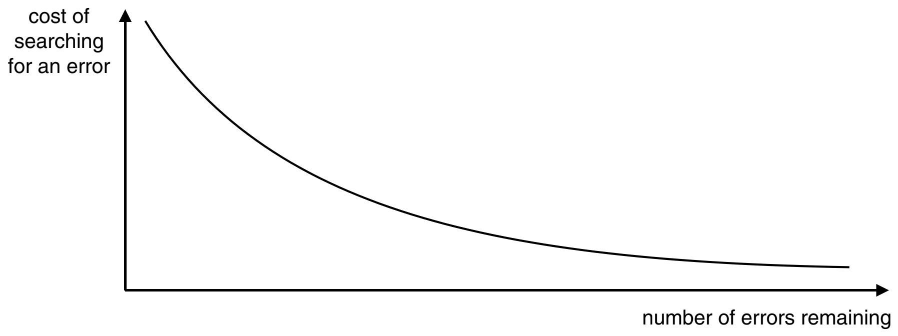


Figure 9.12 The cost of searching for errors as a function of the number of remaining errors

When a new version of the software is released, it can generally be assumed that it will contain fewer errors, since there has been an opportunity to analyze errors discovered in use and eliminate them. Interestingly enough, this reduction in the number of errors does not continue indefinitely. Instead, the number of errors is usually seen to reach a minimum around the second version, following which it generally increases. This comes about simply because the necessary corrections are based on the original specifications and source code. After a certain time, which can vary, it is likely that correcting one error will produce one or more new errors. This leads to the curve shown in Figure 9.13. After a certain number of versions, it is thus significantly better to make a completely fresh start than to continue building on outdated concepts and repeatedly revised source code. Incidentally, this is true in almost all fields of technology.

In accordance with the IEEE 1008 standard ('Standard for Software Unit Testing'), three test levels can be distinguished for dynamic testing. The first of these is the *basic test* level, which essentially tests the basic functions and successful execution of the individual commands. The

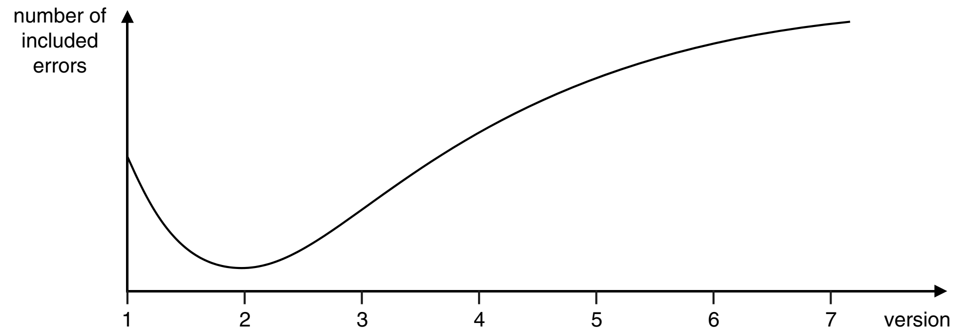


Figure 9.13 Empirically determined dependence of the number of residual errors in a program on the version number

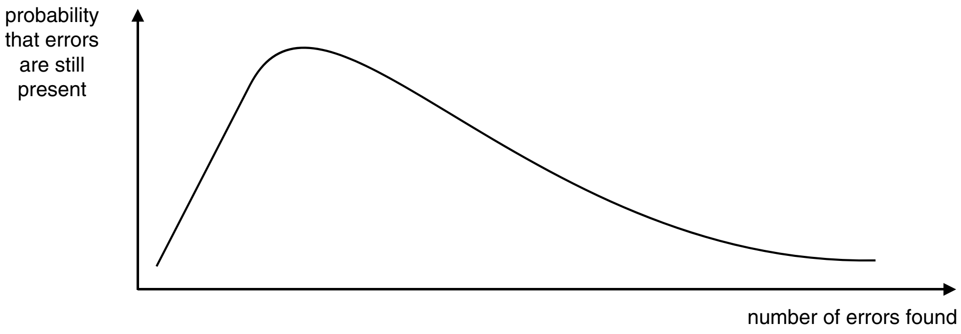


Figure 9.14 Empirically determined dependence of the number of residual errors in a program on the number of errors already found

second level is the *capability test*, which encompasses boundary values and non-successful execution. The third level is the *behavior test*, in which commands are tested in combination with each other.

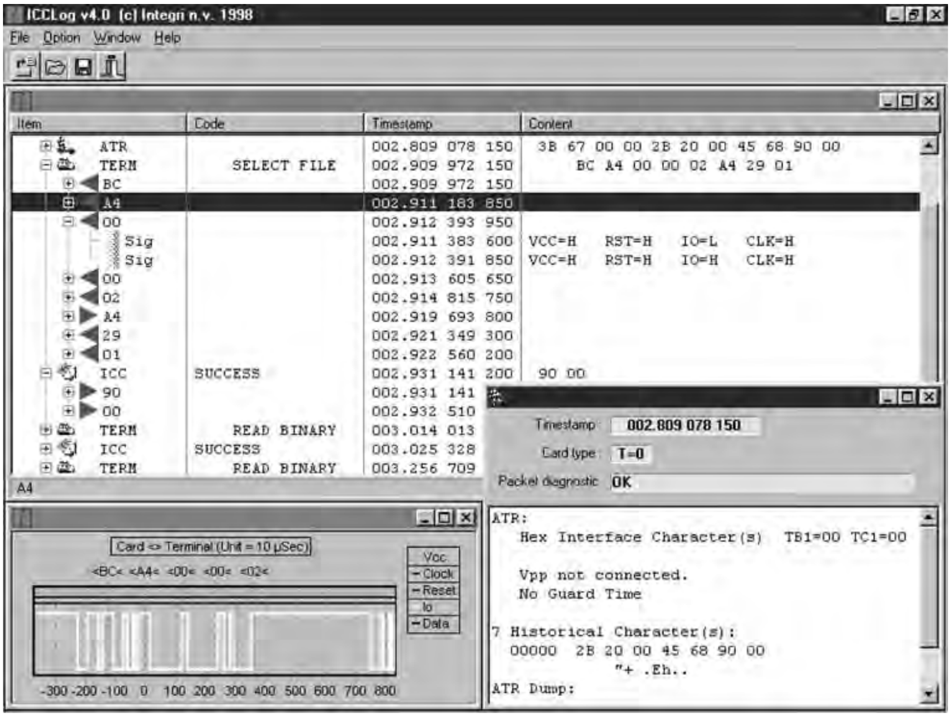
Test methodology

There is a major difference between testing a new operating system and testing a new application. When a smart card operating system is tested, the entire program code must be tested for a wide variety of application cases. This requires a large number of different tests. In the case of a new application, which consists of only a DF and several EFs, the number of tests is reduced to match the amount of additional data and the identification and authentication procedure defined for the application.

If a new operating system must be tested, several test applications that are similar to some typical real applications are usually generated. This essentially amounts to creating equivalence classes for the usual applications. These equivalence classes form the basis for the individual tests that are subsequently performed.

The approach to testing new smart card operating systems described here has become established in the course of several years in a wide variety of projects. Testing always starts with the data transmission functions, since they form the basis for all further activities. Following this, all available commands are tested. If an application is involved, the next stage is file tests. If all these tests are completed successfully, testing of defined procedures can begin.

There are currently only a few international standards that govern the construction and execution of tests for smart card operating systems and applications. A European standard (EN 1292) defines a few tests for the ATR and the T = 1 transmission protocol. For GSM smart cards, relatively extensive tests for the operating system and application are defined in the GSM 11.17 specification.



```
Item          Code            Timestamp         Content
ATR                           002.809 078 150   3B 67 00 00 2B 20 00 45 68 90 00
TERM          SELECT FILE     002.909 972 150      BC A4 00 00 02 A4 29 01
  BC                          002.909 972 150
  A4                          002.911 183 850
  00                          002.912 393 950
    Sig                       002.911 383 600   VCC=H   RST=H   IO=L   CLK=H
    Sig                       002.912 391 850   VCC=H   RST=H   IO=H   CLK=H
  00                          002.913 605 650
  02                          002.914 815 750
  A4                          002.919 693 800
  29                          002.921 349 300
  01                          002.922 560 200
ICC           SUCCESS         002.931 141 200   90 00
  90                          002.931 141
  00                          002.932 510
TERM          READ BINARY     003.014 013
ICC           SUCCESS         003.025 328
TERM          READ BINARY     003.256 709
A4
```

```
Timestamp          002.809 078 150
Card type          T=0
Packet diagnostic  OK
```

```
ATR:
  Hex Interface Character(s)  TB1=00 TC1=00

  Vpp not connected.
  No Guard Time

7 Historical Character(s):
 00000  2B 20 00 45 68 90 00
             "+ .Eh..
ATR Dump:
```

Figure 9.15 Screen display of a tool for verifying the communications between a terminal and a smart card on the physical and logical levels (Copyright © Integri [Integri])

In order to provide an overview, a selection of possible tests in a conventional sequence is presented below. This list does not pretend to be complete, and it is only meant to serve as a detailed illustrative example. The purpose of the listed tests is to test the essential general parameters of a new operating system, including one or more applications.

Data transmission tests

- ATR (parity error detection, and if T = 0 is present, character repetition and ATR structure and contents)
- PTS (PTS structure and contents)
- Data transmission test at OSI layer 2
  (start bit, data bits and stop bit, divider, and data transmission convention)
- T = 0 transmission protocol
  (parity error detection and character repetition, various processes)
- T = 1 transmission protocol
  (CWT, BWT, BGT, resync, error mechanisms, various processes)
- Secure messaging

Testing available commands

- Test all possible class bytes
- Test all possible instruction bytes
- Test all available commands using equivalence classes for the supported functions

Testing available files

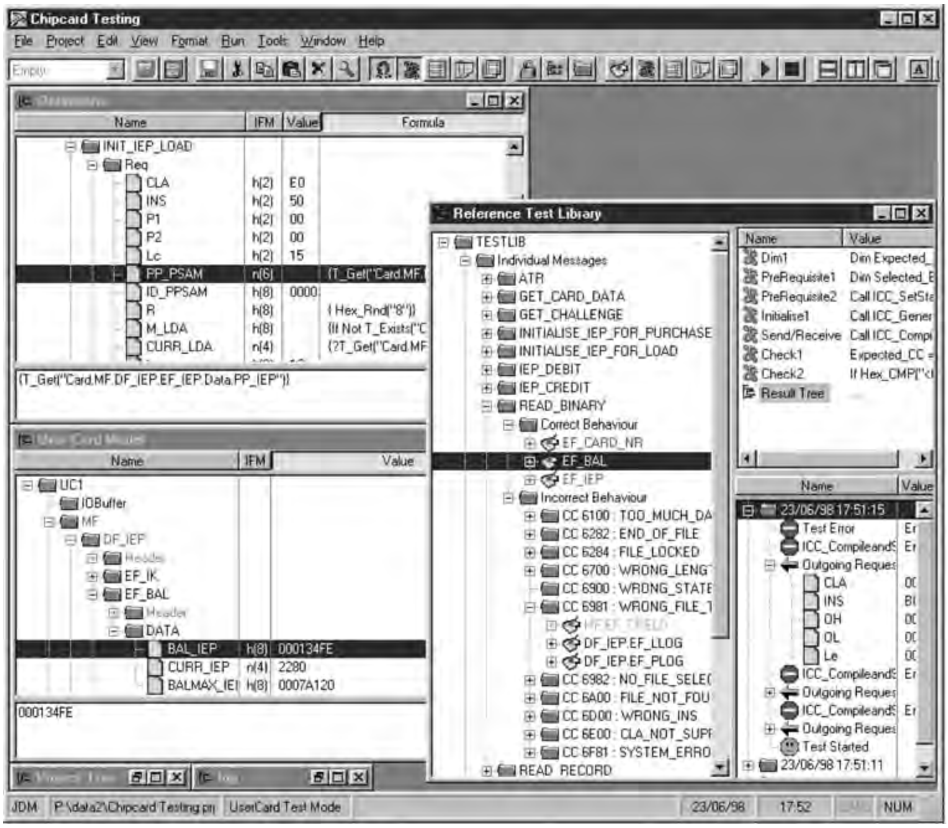
- Test whether all files are present in the correct locations (MF, DF)
- Test for correct file size
- Test for correct file structure
- Test for correct file attributes
- Test for correct file contents
- Test the defined access conditions (read, write, block, unblock etc.)

Testing available processes

- Test the defined state machines (e.g., the command sequence)

As can easily be imagined, even if equivalence classes are generated and various other minimization techniques are used, a relatively large number of individual tests are required. It can be assumed that 4000 to 8000 different tests must be prepared to cover the essential test cases for a 20-kB smart card operating system, with tests that perform the same operation many times in a single loop (such as sending several hundred different values to the smart card) being counted as single tests. The number of commands sent to the smart card using these tests can easily be on the order of 40,000. The amount of time required to perform all these tests is in the range of one to two days. The only way to manage such a large number of tests with a reasonable amount of effort is to use a suitable database, which can also store the test results.

The 'tree and tabular combined notation' (TTCN), which is standardized in ISO/IEC 9646-3, is one of the techniques that can be used to formally describe the tests. Any desired test case can be described in a general and standardized form using this notation. An interpreter can then use this description to automatically generate the command APDUs for the card being tested. This allows largely automated test procedures to be defined.


Chipcard Testing
File Project Edit View Format Run Tools Window Help
Empty
Reference Test Library
JDM
P:\data2\Chipcard Testing.prj
UserCard Test Mode
23/06/98
17:52
NUM

Name	IFM	Value	Formula
INIT_IEP_LOAD			
Req			
CLA	h(2)	E0	
INS	h(2)	50	
P1	h(2)	00	
P2	h(2)	00	
Lc	h(2)	15	
PP_PSAM	n(6)		(T_Get("Card.MF.
ID_PPSAM	h(8)	0000	
R	h(8)		(Hex_Rnd("8"))
M_LDA	h(8)		(If Not T_Exists("C
CURR_LDA	n(4)		(?T_Get("Card.MF

(T_Get("Card.MF.DF_IEP.EF_IEP.Data.PP_IEP"))

Name	IFM	Value
UC1		
IOBuffer		
MF		
DF_IEP		
Header		
EF_IK		
EF_BAL		
Header		
DATA		
BAL_IEP	h(8)	000134FE
CURR_IEP	n(4)	2280
BALMAX_IEI	h(8)	0007A120

000134FE

```
TESTLIB
  Individual Messages
    ATR
    GET_CARD_DATA
    GET_CHALLENGE
    INITIALISE_IEP_FOR_PURCHASE
    INITIALISE_IEP_FOR_LOAD
    IEP_DEBIT
    IEP_CREDIT
    READ_BINARY
      Correct Behaviour
        EF_CARD_NR
        EF_BAL
        EF_IEP
      Incorrect Behaviour
        CC 6100 : TOO_MUCH_DA
        CC 6282 : END_OF_FILE
        CC 6284 : FILE_LOCKED
        CC 6700 : WRONG_LENG
        CC 6900 : WRONG_STATE
        CC 6981 : WRONG_FILE_T
          MF.EF_TRELD
          DF_IEP.EF_LLOG
          DF_IEP.EF_PLOG
        CC 6982 : NO_FILE_SELE(
        CC 6A00 : FILE_NOT_FOU
        CC 6D00 : WRONG_INS
        CC 6E00 : CLA_NOT_SUPI
        CC 6F81 : SYSTEM_ERRO
    READ RECORD
```

Name	Value
Dim1	Dim Expected_
PreRequisite1	Dim Selected_E
PreRequisite2	Call ICC_SetSta
Initialise1	Call ICC_Gener
Send/Receive	Call ICC_Compi
Check1	Expected_CC =
Check2	If Hex_CMP("<l
Result Tree	

Name	Value
23/06/98 17:51:15	
Test Error	Er
ICC_CompileandS	Er
Outgoing Reques	
CLA	0(
INS	B(
OH	0(
OL	0(
Le	0(
ICC_CompileandS	Er
Outgoing Reques	
ICC_CompileandS	Er
Outgoing Reques	
Test Started	
23/06/98 17:51:11	

Figure 9.16 Screen display of an object-oriented, database-supported tool for testing smart card operating systems and applications. The definition of the data elements of a command APDU (INITIALIZE IEP for Load) is shown at the upper left. Below this is the associated reference simulation of the smart card. Part of the tree structure that defines the individual tests is displayed on the right (Copyright © Integri [Integri])

The structure of a test tool for smart cards is shown in Figure 9.17. The specification of the card's software, which is written completely in pseudocode, is contained in an appropriate database. If the specification changes, the necessary modifications to the tests are made automatically. Another database contains all of the tests, which are defined in a high-level language that can also be directly read by a computer. The two databases feed a test pattern generator, which generates the commands (i.e., TPDUs or APDUs as appropriate) for the card being tested. A simulation of the real card, which is largely defined by the specification, is run in parallel. Since there are incompletely predictable processes in the real card (e.g., generating a random number), additional data must be sent to the simulated card. The real and simulated cards send their command responses to a comparator. If they are the same, the real card has provided the correct result, insofar as the simulation is the proper reference. All the data generated during a test run are stored in a log database so they can later be manually evaluated.

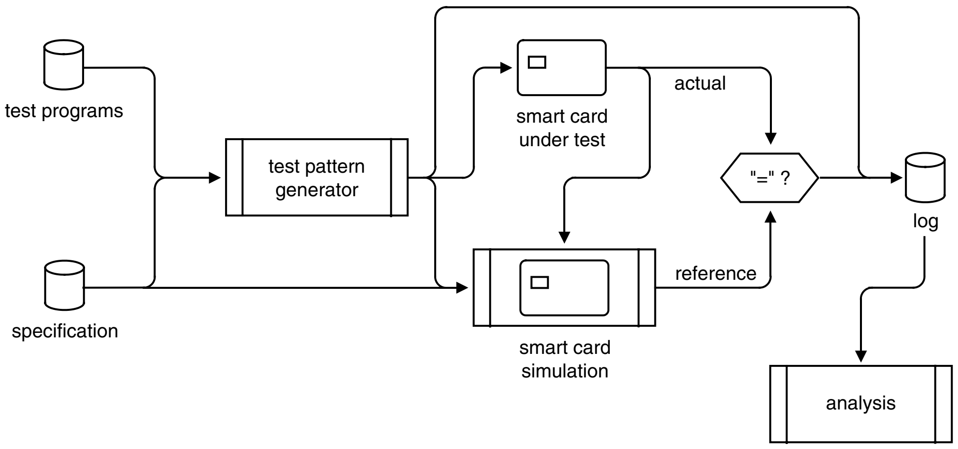


Figure 9.17 Basic structure of a tool for testing smart card operating systems and applications

10
The Smart Card Life Cycle

This chapter presents the life history of a smart card, from the origin of the semiconductor chip through the production of the card to the recycling of the card materials. One section of this chapter is also dedicated to the life cycle of smart card applications, since this is very important with regard to multiapplication cards, which are becoming increasingly numerous. Separate chapters of this book are dedicated to the most important stages in the life of a smart card, such as the basic aspects of smart card operating systems. There are explicit references to these chapters in the appropriate locations. Regarding the materials used for the card body, the various types of modules and other card components, we refer you directly to the corresponding sections of Chapter 3.[1]

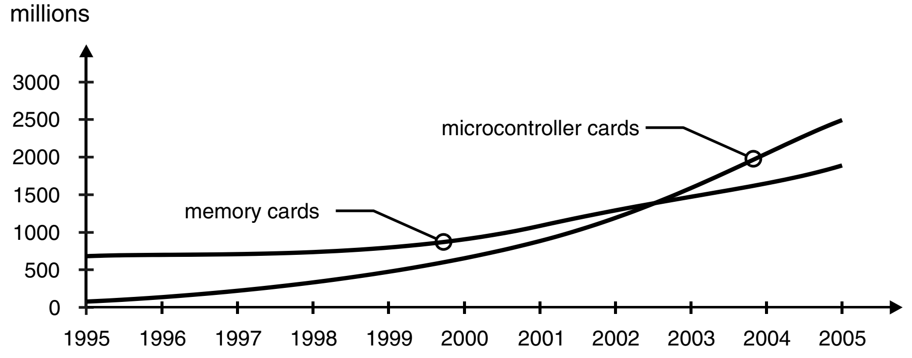


Figure 10.1 International production of memory and microcontroller cards. The numbers are estimated values, since the various sources differ considerably. Average values have been used here

A smart card basically consists of two completely different components. The first component is the card body, with its printing, security features and possibly a magnetic stripe. The second component, which is what makes the card body into a proper smart card, is the module

[1] See Section 3.2, 'The Card Body'

incorporating the chip. This division of the card into two components applies equally well to memory cards and microcontroller cards.

The manner in which data are transferred also affects the structure of the card. Smart cards with contacts make electrical connections to the terminal by means of six or eight externally visible contacts. Contactless smart cards contain coils of various sizes within the card body, which are connected to the chip module that is also embedded in the card body. This structure naturally has a significant effect on the manufacturing of this type of card.

The manufacturing process also depends to a considerable degree on other elements of the card, such as the material used for the body of the card, the methods used for applying text to the card and the security features. However, regardless of all these options, there is one thing that has absolute priority: cost optimization. Manufacturing smart cards is a mass production process, in which lot sizes start at around 10,000 pieces and can certainly reach the level of 10 million. A highly optimized production process is the most important prerequisite for the cost-effective manufacturing of high-quality card products.

10.1 THE FIVE PHASES OF THE SMART CARD LIFE CYCLE

In addition to the manufacturing process, the life cycle of a smart card depends on the application in which it is used. A smart card for the GSM mobile telecommunications system, for example, has a considerably different career after manufacturing than a credit card containing a chip. Nevertheless, the various types of cards still have much in common.

Table 10.1 Summary of the individual life-cycle phases according to the ISO 10202-1 standard

Life cycle phase	Typical activities
Phase 1: Production of the chip and the smart card	• designing the chip • generating the smart card operating system • fabricating the chips and modules • producing the card body • embedding the module in the card body
Phase 2: Card preparation	• completing the smart card operating system
Phase 3: Application preparation	• initializing the application(s) • personalizing (individualizing) the application(s), both visually and electrically
Phase 4: Card usage	• activating the applications • deactivating the applications
Phase 5: Termination of card usage	• deactivating the applications • deactivating the card

The ISO 10202-1 standard attempts to define a card life cycle that is equally valid for all manufacturing methods and a wide variety of applications. This standard is very strongly oriented towards financial transaction applications and the information technology used in these applications, rather than the actual production of card bodies and chips. Nevertheless, it represents a quite successful attempt to provide a structured description of the life history of smart cards from the beginning to the end. This is why it is used here as the basis for describing the smart card life cycle.

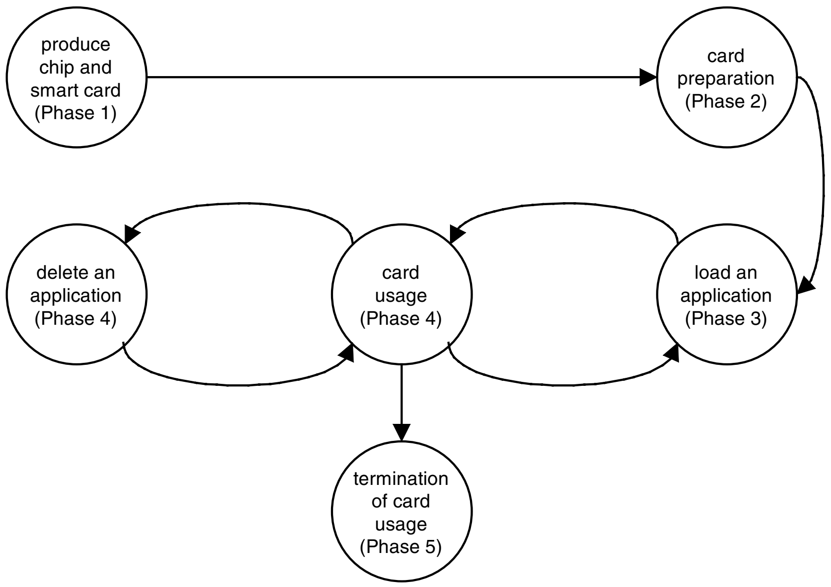

produce chip and smart card (Phase 1)
card preparation (Phase 2)
delete an application (Phase 4)
card usage (Phase 4)
load an application (Phase 3)
termination of card usage (Phase 5)

Figure 10.2 The life cycle of a smart card according to the phase model of the ISO 10202-1 standard. Loading and deleting applications, which is possible with multiapplication smart cards, takes place in phases 3 and 4

According to the ISO 10202-1 standard, the life of a card is divided into five phases, which are interconnected by precisely specified transitions. All stores of cards required by the technical implementation of the production process and all transportation paths between the various firms that perform the various production operations must be physically or cryptographically secured in order to preclude the manipulation or theft of partly finished products.

All production steps must naturally be accompanied by appropriate quality assurance. Since smart cards are normally used in areas in which security is an issue, it is presently common to guarantee the traceability of the manufacturing process in accordance with the ISO 9000 family of standards. At minimum, this means that all production steps must be logged using batch and chip numbers. It must be possible to reconstruct the production steps undergone by each individual smart card, at any desired time after it has been manufactured. This makes it easier to analyze the cause of any manufacturing faults that may show up. Since each individual chip has a unique chip number, no two microcontrollers are identical following the semiconductor fabrication process, which makes it relatively easy to implement traceability on the basis of chip numbers. Manufacturing traceability can be implemented either by storing the relevant information in a manufacturing database or by writing all the information relevant to the manufacturing of each chip in the chip itself. The ISO 10202-1 standard recommends storing the manufacturing data in the chips, which has certain advantages compared with storing the data in a database. If the data are stored in the chips, the manufacturing data for any chip can be obtained without having to access a database, although this comes at the cost of valuable space in the microcontroller EEPROM.

Table 10.2 Manufacturing data typically stored in chips. This information can be written only once, after which it can only be read (the file or data object has the WORM attribute)

Phase of the smart card life cycle	Typical fabrication data
Phase 1: chip and smart card production	• ID of the chip manufacturer • ID of the fabrication line • unique chip number • chip type • ID of the module embedder • date and time of embedding the chip in the module
Phase 2: card preparation	• ID of the initializer • ID of the finishing machine • date and time of initialization
Phase 3: application loading	• ID of the personalizer • ID of the finishing machine • date and time of personalization

10.2 PHASE 1 OF THE LIFE CYCLE IN DETAIL

The first phase of the ISO 10202-1 standard life cycle can be subdivided into two parts. The first of these covers the generation of the smart card operating system and the semiconductor manufacturing process for the microcontroller, while the second part covers all of the technology for producing the card body.

10.2.1 Generating the operating system and producing the chip

Operating systems and other software for smart card microcontrollers are so complex that we have devoted a separate chapter to them, in which all aspects of the subject are described in detail.[2] However, we must not overlook the fact that a significant part of the technical basis for the security of the remainder of the card's life cycle is established in the fabrication of the chip. No matter how high the quality of the operating system may be and how much cryptographic protection is used, they are of little use if all the secret data can be read from the chip thanks to an error in the design or fabrication of the chip.

Semiconductor chips are usually produced in protected facilities with restricted access. Restricted access is relatively easy to achieve with cleanrooms, which can anyway only be entered via interlocked doorways. However, this is also important with regard to security, since it is the only way to guarantee that no ICs containing Trojan horses in their software can be smuggled into the system during chip fabrication or after the dice have been separated. This would otherwise be a very serious and relatively dangerous form of attack on the security of smart card applications.

[2] See Chapter 5, 'Smart Card Operating Systems'

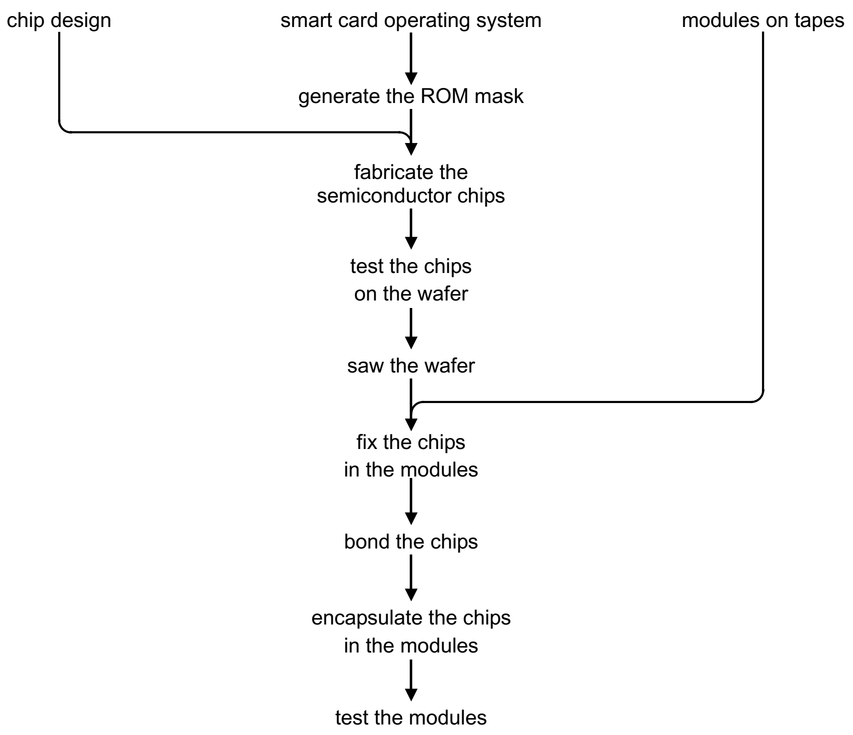

chip design
smart card operating system
modules on tapes
generate the ROM mask
fabricate the semiconductor chips
test the chips on the wafer
saw the wafer
fix the chips in the modules
bond the chips
encapsulate the chips in the modules
test the modules

Figure 10.3 The production of chips and modules in the first phase of the smart card life cycle

Chip design

The geometric structure of a chip for a memory or microcontroller card should be square or as nearly possible square, since this minimizes the risk of the chip being broken by the stresses that arise when the card is bent. Complete protection of the chip against bending stresses is in principle technically possible with an extremely stiff module package, but this is not desirable in practice. Such a stiff module would eventually cause the card body to crack, due to the alternating bending stresses to which the card is recurrently exposed.

The semiconductor components used for the chip, such as the CPU and numeric coprocessor, are normally standard components[3] that have been technically modified for increased security. Semiconductor components for the automotive industry are often used for this purpose, since they must be designed to meet similarly severe environmental and reliability requirements. However, such components must be modified as necessary to fully adapt them to the security requirements imposed on smart card microcontrollers.

In the chip design process, the first step after establishing the functional specification is generating a general chip architecture, with a block diagram of the circuit and a rough layout of the future microcontroller. Following this, the overall block diagram is refined step by step

[3] See also Section 3.4, 'Smart Card Microcontrollers'

into logic blocks, gate-level functions, transistors and ultimately the geometric structures of the individual exposure masks. Each step is accompanied by circuit simulation and extensive testing. This is a complex process, consisting of many individual steps, and a fair amount of experience is necessary to arrive at the optimum arrangement of the elements of the chip. At the end of this process, sample chips are produced on a test fabrication line in a semiconductor manufacturing plant. These are the first reference devices, which are very precisely measured and exercised. A security assessment is often performed in parallel, although this assessment cannot be completed any earlier than the time when the first regular chips are produced.

The process of designing a chip can take several months to a year before a fully operational chip is obtained that can meet all the necessary requirements for mass production. The fact that it takes so much time and effort to design a chip is the reason that the interval between successive generations of smart card microcontrollers is two to three years. Due to the high cost of making significant changes to an existing chip, the most substantial modifications are predominantly ‘shrinking’ the chip, in order to better utilize the wafer area, and making minor improvements or extensions to the hardware.

Smart card operating systems

Software for smart card operating systems and applications based on these operating systems must be written using either assembly language or the C language, due to the small memory capacities of the microcontrollers. Using these languages, which are relatively close to the hardware level, naturally tends to disproportionately prolong the duration of the entire software development process, and thus significantly increases its cost.

The tests for the software, most of which is located in the ROM of the microcontroller, are very thorough and comprehensive, since it is almost impossible to correct any residual errors in this software after the chips have been manufactured.[4] Chip production always involves generating ROM masks, which essentially represent the software that will later be located in the ROM of the microcontroller, where it cannot be subsequently modified. If any software error is detected in the following production steps, it can only be corrected by repeating all of the preceding steps.

In order to make the best possible use of the available memory space in the microcontroller, the program code must be adapted to the specific type of chip that is used. Porting the software to another type of chip is thus only possible at additional effort and expense. Consequently, the time required to generate a complete ROM mask is around nine months. This can be significantly reduced if it is possible to use program code that is already on hand (in the form of software libraries). Once the development of the ROM mask has been completed, it can be formally handed over to a semiconductor manufacturer.

ROM masks and fabrication of the semiconductor chips

From the software that it receives in an EEPROM, on a diskette or via data telecommunications, the semiconductor manufacturer generates an exposure mask for the ROM of the

[4] See also Chapter 9, ‘Quality Assurance and Testing’

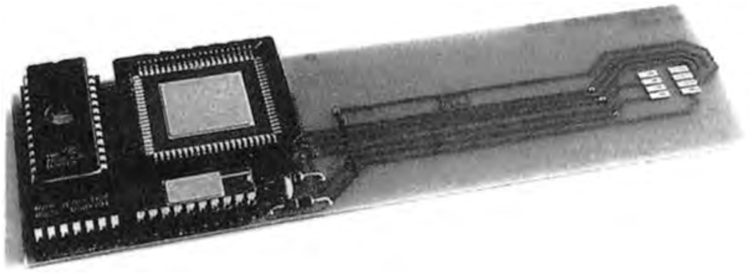


Figure 10.4 A mini emulator for a smart card, in which the mask-programmed ROM has been replaced by a removable EPROM in a DIL package. The large IC is a smart card microcontroller with all busses freely accessible (a 'bond-out' chip)

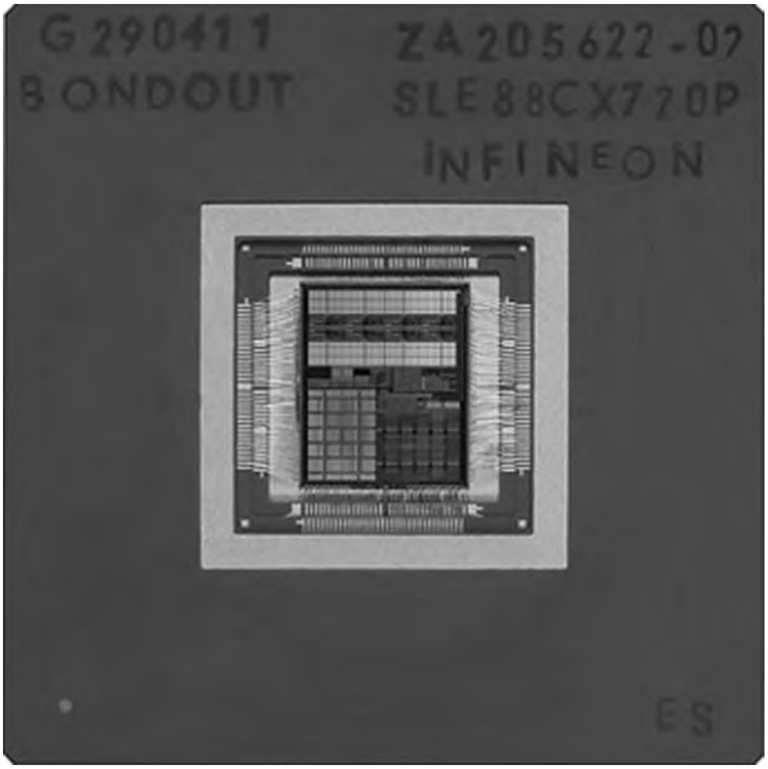


Figure 10.5 Example of a bond-out chip without final shielding, for use in a smart card microcontroller emulator. (*Photo:* Infineon Technologies)

microcontroller. This mask, which contains the program code, is called the 'ROM mask' by operating system designers, or often simply 'the mask'. If the structures are reduced in size when the mask is imaged onto the wafer, it is instead called a 'reticule'. The ROM mask is only one of approximately 20 masks needed to produce the microcontroller. The structures of the microcontroller chips are produced on suitably prepared high-purity silicon disks called wafers. The diameter of wafers used for smart card microcontrollers is presently usually 6 to 8 inches (15.2–20.4 cm). With 0.8-µm technology, around 700 microcontrollers will fit on a 6-inch wafer.

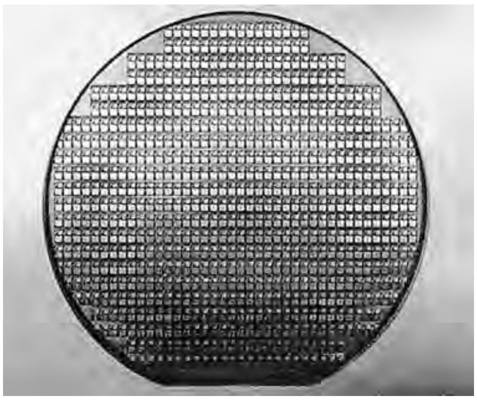


Figure 10.6 A 6-inch diameter wafer containing approximately 700 SLE44C80 microcontrollers. The milled straight edge (primary flat) is used to mark the orientation of the wafer during fabrication (*Source:* Infineon)

In the semiconductor industry, the trend for fabrication processes in the coming years is towards larger wafers and smaller structures. It can be assumed that in a few years, 12-inch (30.6-cm) wafers and 0.13-µm technology will become the prevailing standard for producing smart card microcontrollers. The cost of a fabrication plant at this level of technology is on the order of one billion euros.

Up until only a few years ago, full-wafer masks were used, with all 700 microcontrollers being exposed at once. Contact exposure was normally used with such masks. As the dimensions of the chip structures became increasingly smaller, this was no longer possible, since the yield was not acceptable. In all new production methods, the set of photomasks represents only a single chip, instead of an entire wafer. These very delicate masks are made from plates of quartz glass, which is transparent to ultraviolet light. These plates hold the patterns for a chip in the form of chrome-metal tracks. The track patterns are transferred to the glass plates by first coating the plates with a photosensitive layer and then using an electron-beam writer to expose the patterns. Following this, the sensitive layer is developed and the unexposed regions are removed by etching.

The photomask is produced at a scale that is 5 or 10 times greater than the actual scale of the chip, which allows image-enhancing reduction to be used when the wafer is exposed. The machines that are used to expose the wafers, which are called 'steppers' in the trade, are high-precision optical devices that can focus the image of the mask on the wafer with an accuracy of a fraction of a micrometer. They can also reposition the wafer with an equal level of precision. After the ultraviolet light exposure process for one chip has been completed, the wafer is moved by one step to the position of the next chip, and the exposure process is repeated at this position. The entire wafer is thus exposed one step at a time, until all the microcontrollers that it will contain have been exposed.

The entire wafer is coated with a light-sensitive lacquer called the photoresist. Where the photoresist has been exposed to light, the lacquer is removed by etching, and the underlying

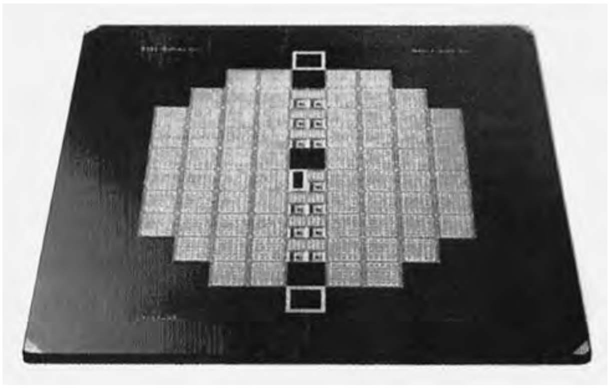


Figure 10.7 Quartz-glass photomask for simultaneously exposing an entire wafer (*Source:* Philips)

wafer surface is then doped with impurity atoms. After the wafer has been cleaned several times and recoated with a new layer of photoresist, it is ready to be exposed using the next of the approximately 20 masks. Depending on the particular manufacturer and the fabrication process used, producing a finished wafer involves around 400 processing steps and takes six to 12 weeks, although the The actual processing time is less than a week. The very long lead times encountered in practice are primarily due to the queuing technique commonly used in the mass production of semiconductor devices.

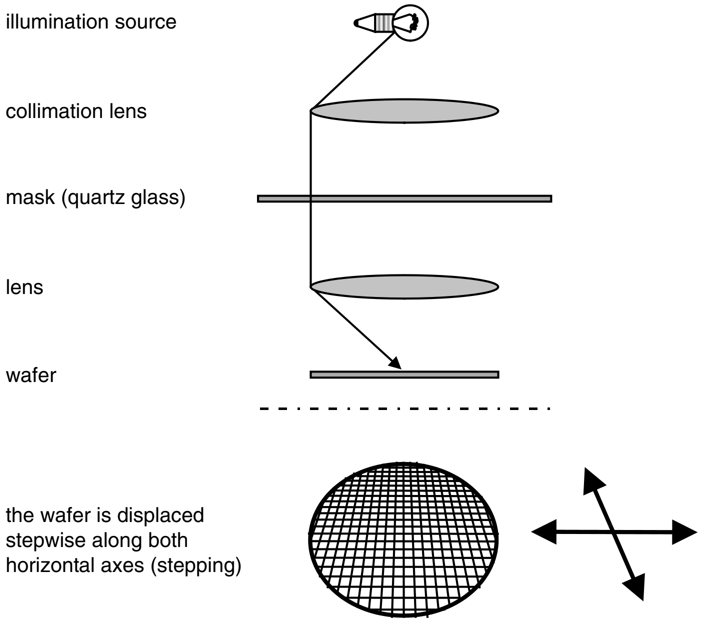

illumination source
collimation lens
mask (quartz glass)
lens
wafer
the wafer is displaced stepwise along both horizontal axes (stepping)

Figure 10.8 Operating principle for exposing individual chips on a semiconductor wafer using a stepper

In order to make the production process more economical, a group of several wafers (a batch) is always passed through the semiconductor fabrication machine each time. A typical batch consists of 12 wafers, which is commonly the minimum production quantity. This corresponds to approximately 10,000 chips for almost all semiconductor manufacturers. It takes a considerable amount of extra effort to process less than a full batch, so the production costs would be just as high as for a full batch. However, some fabrication equipment allows shared batches or multi-project wafers, in which different types of microcontroller chips with different ROM masks are produced on a single wafer. This allows smaller lots than the otherwise obligatory 10,000 pieces. However, by no means all types of microcontrollers can be produced in this manner, and not every fabrication machine can handle shared batches.

The overall yield from the fabrication process is around 80 % with a well-tuned process. This means that only around 560 of the original 700 chips on the wafer can be used in the following production steps. With a relatively new production process, the long-term yield can easily be as low as 60 %. This has a very negative effect on production volume and profitability.

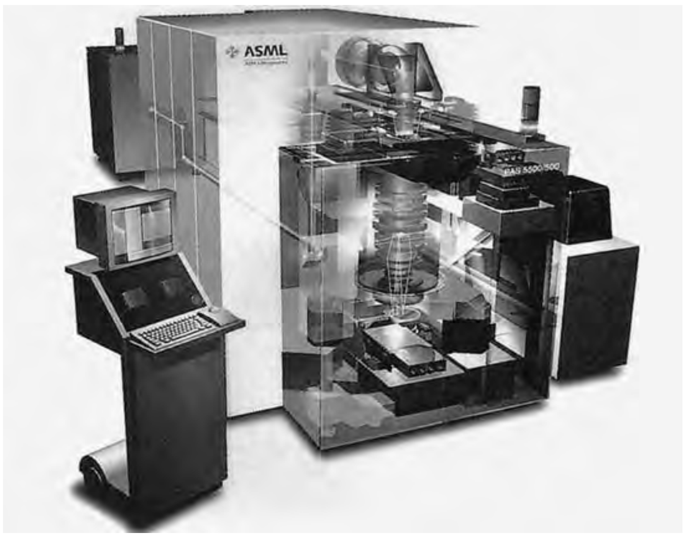


Figure 10.9 Cross-sectional drawing of a stepper for the stepwise optical exposure of individual semiconductor chips (*Source:* ASM Lithography)

Chip testing on the wafer

In the next production step, the microcontrollers on the wafer are contacted using metal probes and individually tested. This requires making contact with each of the 700 microcontrollers on the wafer, either individually or in groups of up to eight, and then performing an electrical function test. Since there are usually not any supplementary contacts for the microcontrollers, even at this production stage only the five contacts that will later be used in the smart card can be used for testing.

The functional elements are tested on the wafer significantly more intensively and extensively than later on, since the microcontrollers are still in the test mode at this stage. In the test mode, all of the memories (RAM, ROM and EEPROM) can be read and/or written without any restrictions. Any microcontrollers that fail this test are marked with a small colored dot. This allows non-functional chips to be optically identified in the following steps, so that they can be discarded after the wafer has been sawn into individual dice.

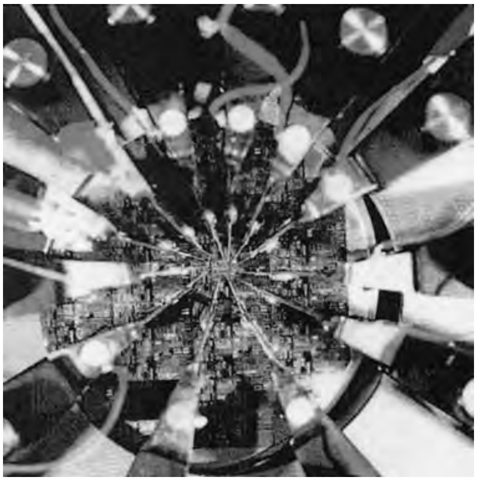


Figure 10.10 Testing a microcontroller on the wafer. Needle probes are used to make connections to the IC, and each IC is individually tested (*Source:* Philips)

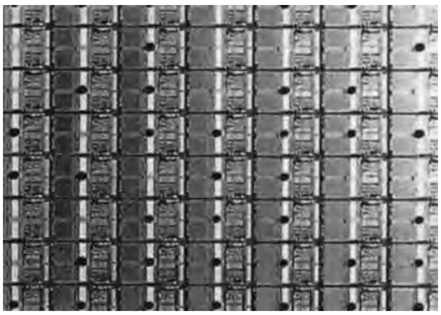


Figure 10.11 Portion of a wafer with microcontroller ICs. The dots on top of some of the chips mark defective devices

In addition, the ability to freely access the memory in the test mode is exploited to write chip-specific data to the EEPROM. This includes a serial number that must be used only once, so that it is unique for each chip. This individualizes each chip, and thereby each smart card. The benefit of this, in addition to certain security aspects, is that traceability as defined by the ISO 9000 family of standards is guaranteed by the unique chip number.

Sawing the wafer

After the chips have been tested on the wafer, the next step is to separate them. A thin self-adhesive film is applied to the back of the silicon disk, so that the individual chips will remain in position after the wafer has been cut. Special saws with blades that are around 25 μm thick and spin at more than 30,000 rpm are used to cut the wafer into pieces. The wafer is sawn such that each resulting piece holds a single microcontroller. These small pieces of silicon, with a maximum area of 25 mm^2, are called dice ('die' in the singular). Each die holds a microcontroller that will ultimately be incorporated into a smart card. Once the wafer has been separated into dice, the defective dice, which are marked with colored dots, are separated from the good dice and destroyed.

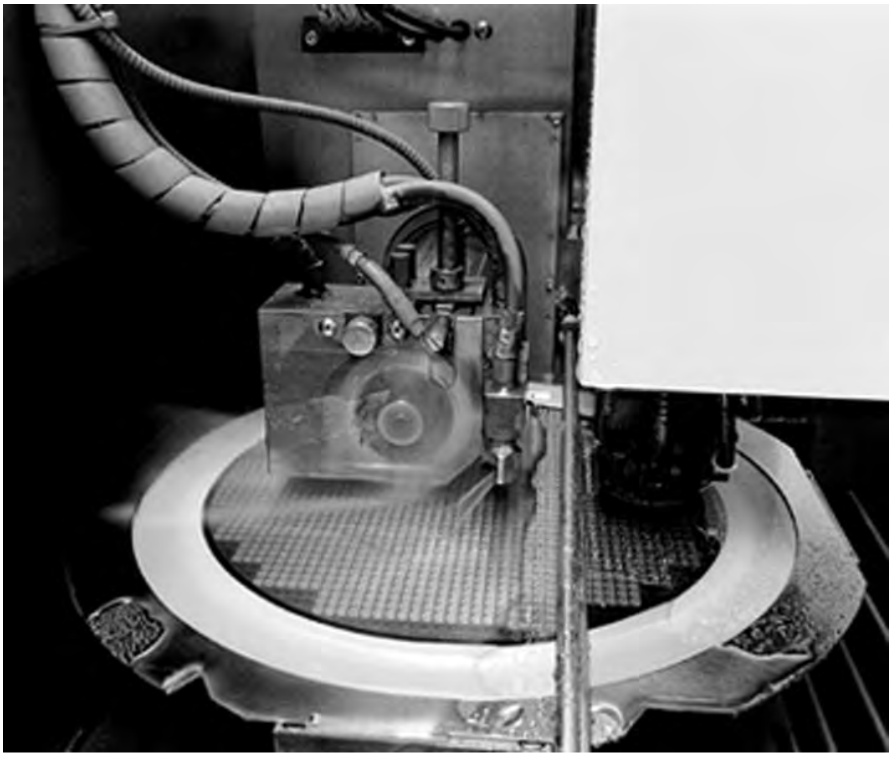


Figure 10.12 Sawing a wafer into dice (*Source:* Renesas)

Up to this point, it is not possible to tell whether the ROM software has been copied without any errors. For this reason, around 10 dice are removed from the batch at this stage and mounted in ceramic DIL packages. The software producer receives these first sample devices and uses his test facilities to determine whether the software in the ROM functions correctly. The entire chip can also be tested. If an error in the software or hardware is detected at this point, the production process must be stopped, and the entire batch has only scrap value. After the error has been corrected, the production process must be started again from the beginning, with the generation of a new ROM mask by the semiconductor manufacturer. The lost time cannot be recovered, even with accelerated handling in the other production phases.

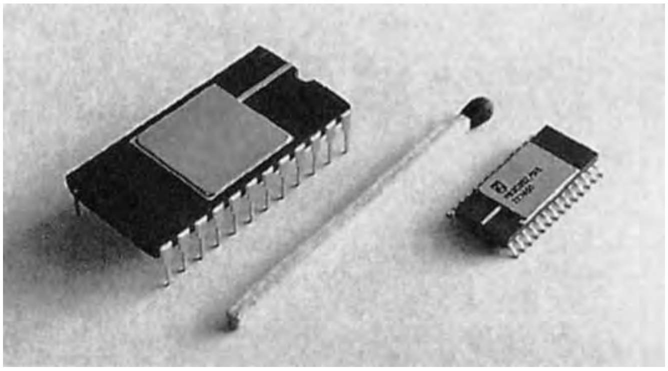


Figure 10.13 Examples of microcontrollers mounted in various types of ceramic packages, which are used for software testing

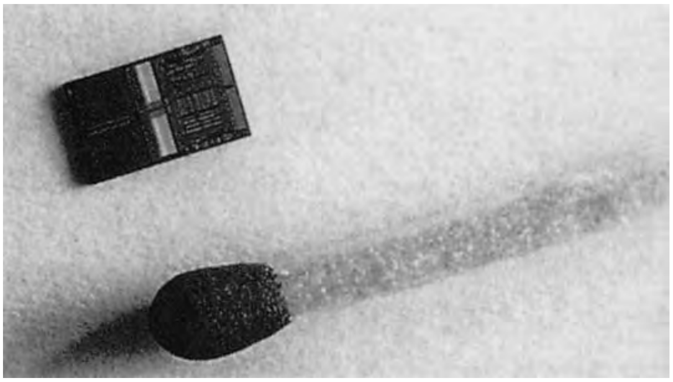


Figure 10.14 An individual die with a single match for size comparison. The die is approximately 0.12 mm thick

Attaching chips to modules

The next step in the production process, after the dice have been sawn from the wafer, is to mount them into modules. The modules increase the resilience of these very fragile bits of quartz crystal, and the electrical contacts on their top surfaces will later be used to make the connections between the card and the terminal.

Chip modules are usually supplied on rolls of 35-mm plastic tape with perforated edges, which carry modules in adjacent pairs. Depending on the size of the module, a single roll can hold from 10,000 to 20,000 modules. The 35-mm plastic tape is simply called 'tape' by insiders, and this type of packaging is called 'chip on tape' (COT).

Incidentally, the width of the tape is the same as that of 35-mm photographic film, which is commonly used in still and motion-picture cameras. The reason for using this format originates from the early days of smart card manufacturing. At that time, the 35-mm film format was chosen for the module carrier to allow inexpensive transport and packaging methods to be used with a minimum of new development, since it allowed existing commercial spools and winding equipment for film to be used for module tapes. Since changing to a different format was no longer feasible after this format had reached a certain level of general use, it is still employed today.

The bottoms of the dice (the silicon base material) are permanently bonded to the bottoms of the modules. The dice can then be electrically connected to the contact surfaces of the modules in subsequent production steps.

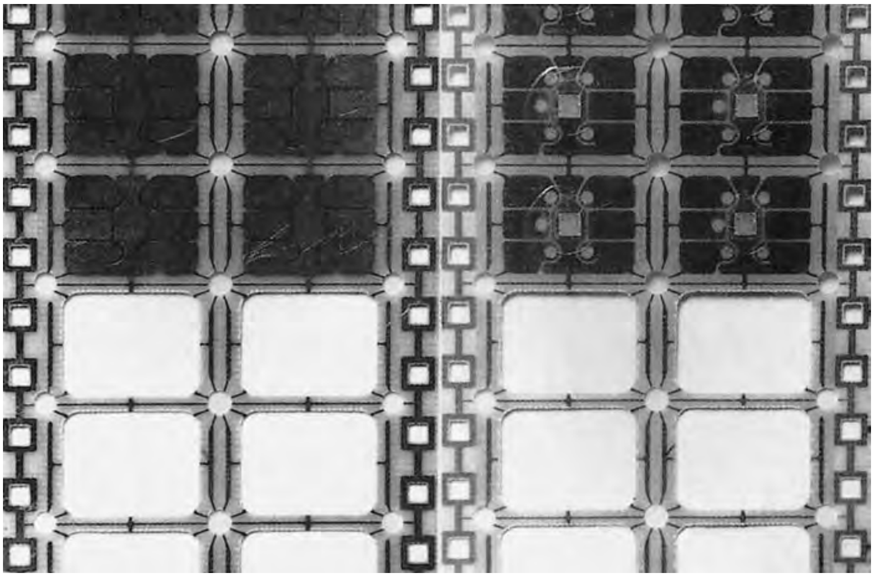


Figure 10.15 Example of a 35-mm plastic tape with attached pairs of modules. The front of the tape is shown on the left, while the rear is shown on the right. The holes left by modules that have already been punched out of the tape can be seen in the lower portions of the pictures

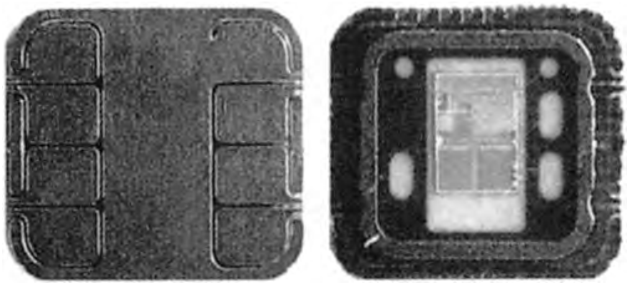


Figure 10.16 Example of the layouts of the front and rear sides of a module

Electrically bonding the chips

After the dice have been glued into the modules, the next step is to make the electrical connections to the rear surfaces of the contacts. This is done using very fine gold wire, which is welded to the aluminum contact pads on the die and the corresponding contact surfaces on the rear of the module. To prevent the bonding wires from being broken by temperature variations, each wire is formed into a loop. However, the loops must not be too large, since otherwise the bonding wires would not be fully covered by the plastic resin that is later poured over the chip. This would increase the risk of corrosion of the wires.

Encapsulating the chips in the modules

After the chip has been bonded, a black epoxy resin is poured over the chip and the rear surface of the module. This resin protects the fragile crystal against environmental influences such as humidity, twisting and bending. An opaque resin is used, because semiconductor devices are normally very sensitive to light and electromagnetic energy in the near-visible part of the spectrum.

After the chips have been encapsulated, the carrier tapes with the modules are wound onto large spools and packed into cardboard boxes. For small production runs, it is also possible to package the modules individually in plastic containers. However, this is avoided when there are large piece counts, since it makes it difficult for the module implanter to use automated processing equipment.

If a new microcontroller is to be introduced into the market or modified chip hardware has to be tested, the production process is often complete when the modules have been encapsulated. In this case, the modules are then passed through suitable testing and qualification stages. Only after these have been completed with no errors is it OK to start a new batch, with suitably modified software, for mass production. A similar situation exists when there is a new version of an operating system, in which case the production process also ends at this point. This is followed by the necessary qualification testing, which can take weeks or even months. If necessary, another pass through the revision loop with an improved version of the operating system may then take place.

Module testing

As a consequence of the production steps up to now – sawing the wafer, attaching and bonding the chips and encapsulating the chips in the modules – 3 to 7 % of the dice will have become unusable. An additional test is therefore usually performed before the modules are packaged and delivered. For this test, each module must be connected to the tester via the contacts on the front of the module. The first thing the tester does is to switch the microcontroller from the test mode to the user mode by blowing the polysilicon fuse and writing a special byte value to a specific location in the EEPROM. After this, it is no longer possible to externally access the memory for reading or writing without first satisfying specific security conditions.

The test computer next carries out an ISO activation sequence and attempts to detect a valid ATR. If this is possible, it then tests the chip hardware using the commands integrated in the mask-programmed software. If all these tests are successful, the module has not been damaged by any of the previous production steps, so it can be built into a smart card.

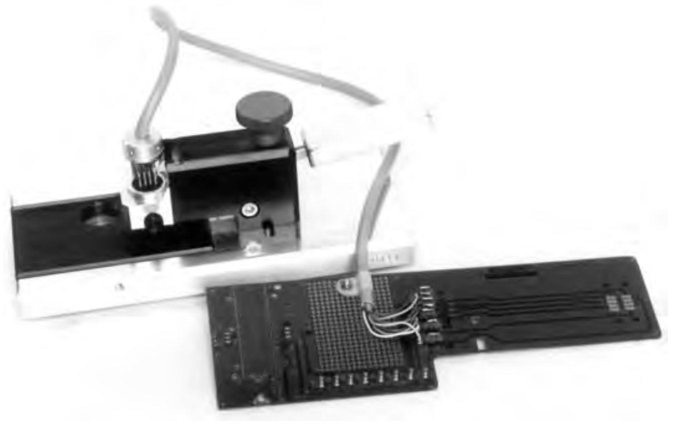


Figure 10.17 An adapter in the commonly used ID-1 format for modules on tape

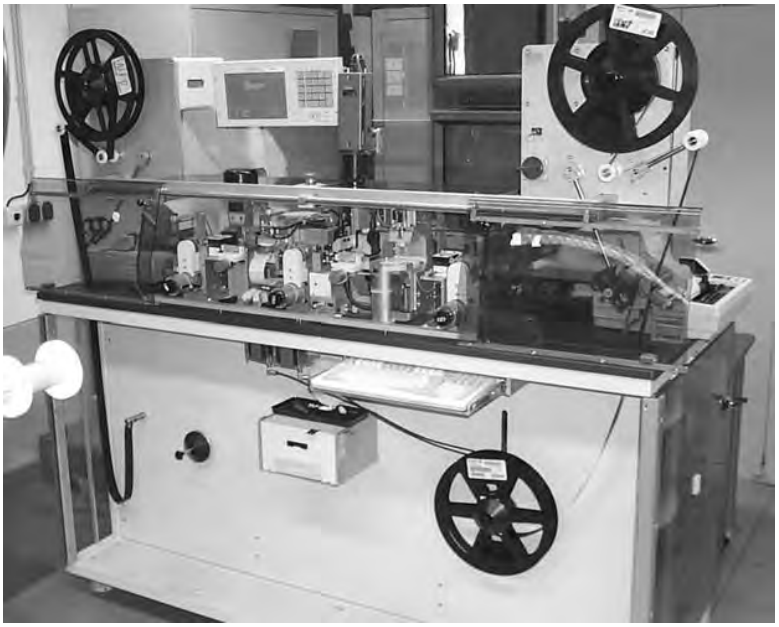


Figure 10.18 An incoming product inspection machine for tape-mounted modules (COT)

10.2.2 Producing card bodies without integrated coils

Card bodies for smart cards that do not have integrated coils can be mass-produced using three different basic processes. These processes differ in terms of the durability of the card, surface features and allowed card components. Many card manufacturers often offer only one type of process, rather than the full range of options.

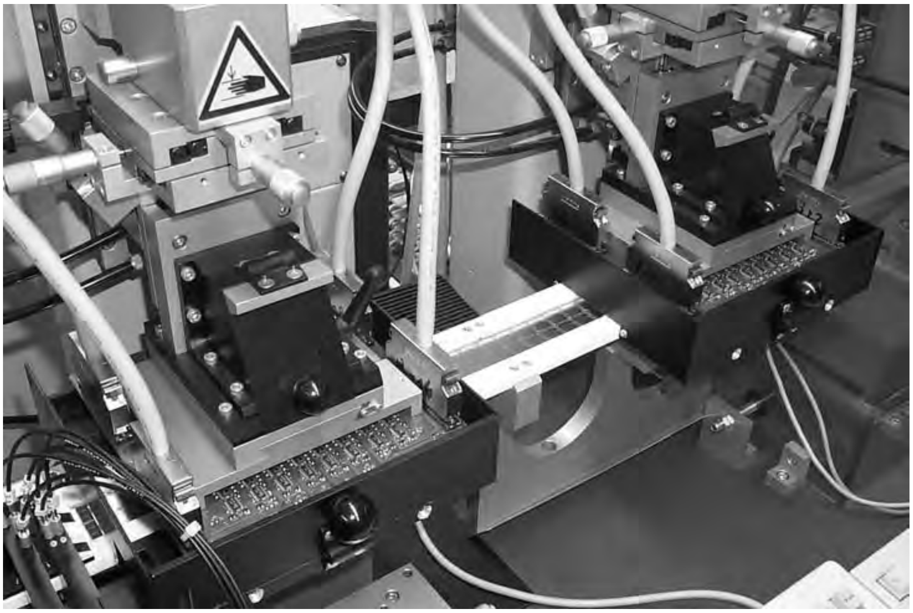


Figure 10.19 Detail photo of an incoming product inspection machine for chip modules. Here 16 modules at a time are electrically contacted by the two contact heads and tested. Machines of this sort can also be used for initializing microcontrollers for smart cards

Laypersons often regard the manufacturing of card bodies as an uncomplicated, easily mastered technology that essentially only amounts to punching out a few pieces of plastic foil and gluing them together. However, this is by no means true. The mass production of high-quality card bodies involves a multitude of complex manufacturing steps, and it demands outstanding mastery of the chemical processes needed to produce and use the plastic materials and associated inks.

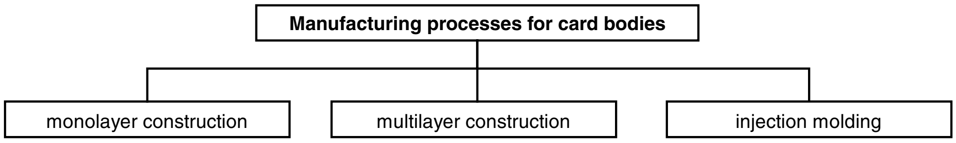


Figure 10.20 Classification of the basic manufacturing processes for plastic card bodies

The technically most elaborate process is to construct the card body from several layers of plastic that are thermally bonded. This is called a multilayer construction, and the process of bonding the layers using heat and high pressure is called lamination. The thickness of the plastic foils used for the inner part of the card (the core foils) ranges from 100 μm to 600 μm, while the thickness of the outer or cover foils (overlay foils) ranges from 25 μm to 300 μm. A card body constructed in this manner allows a great degree of freedom in the form and layout of the card components, is very stable and also allows security features to be placed between the layers. For example, this technique is used in the MM process for German Eurocheque cards.[5]

Monolayer construction of the card body, using a single piece of 800-μm plastic sheet (monofoil), is a simplified version of multilayer construction. This process is less expensive, but the cards are less stable than multilayer cards. Above all, they allow significantly fewer options for the design and layout of the card components. For instance, with monolayer construction it is not possible to have a transparent cover layer to protect the printed elements against scratching and rubbing.

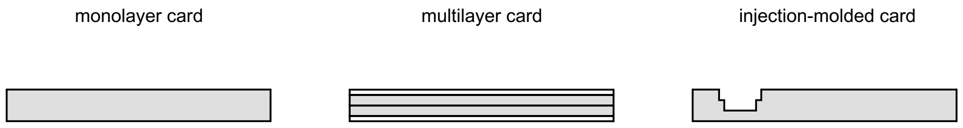


Figure 10.21 Overview of the commonly used types of card construction. A multilayer card is constructed using external cover layers (overlay foils) and internal layers (core foils)

The third process that can be used to produce a plastic card body is injection molding. This essentially results in a monolayer card body, with all of its advantages and disadvantages. However, there is one small but significant difference. A thin printed foil (approximately 80 μm thick) can be placed in the mold, which allows injection-molded cards to be produced with printing right from the mold. This process, which is called in-mold labeling, has its limitations compared with offset printing or silk-screening in terms of layout and the inks that can be used. However, it has the advantage that it is not necessary to run the cards through a single-card printing machine after they have been molded. An additional feature of the injection molding process is that the cavity for the chip module can be formed in the molding process, so it does not have to be milled out afterwards. There are also processes newly available in which the chip module is placed in the mold, so that it is anchored to the card body when the card is molded. This method also makes it unnecessary to perform many of the steps described below.

Standard injection molding machines have a capacity of approximately 2000 card bodies per hour. Although injection molding may appear to be inexpensive, it is usually more expensive than stamping single-layer card bodies from a large sheet of plastic if very many cards have to be produced (more than one million). This is primarily due to the time-consuming handling of individual cards, which is one of the primary cost factors.

Printing the foils

With regard to printing the card body, there is usually no difference between multilayer and monolayer cards. In the sheet printing process, large sheets of plastic are printed on multiple-copy sheets, and the individual cards are then punched from the sheets. The sheets are normally big enough to allow 24 to 48 images (cards) to be printed on each sheet, which is fed one or more times through the individual inking stations of an offset or silk-screen printing press. The front and rear sides of the card bodies must be printed separately.

[5] See Section 3.1.2, 'Card components and security features'

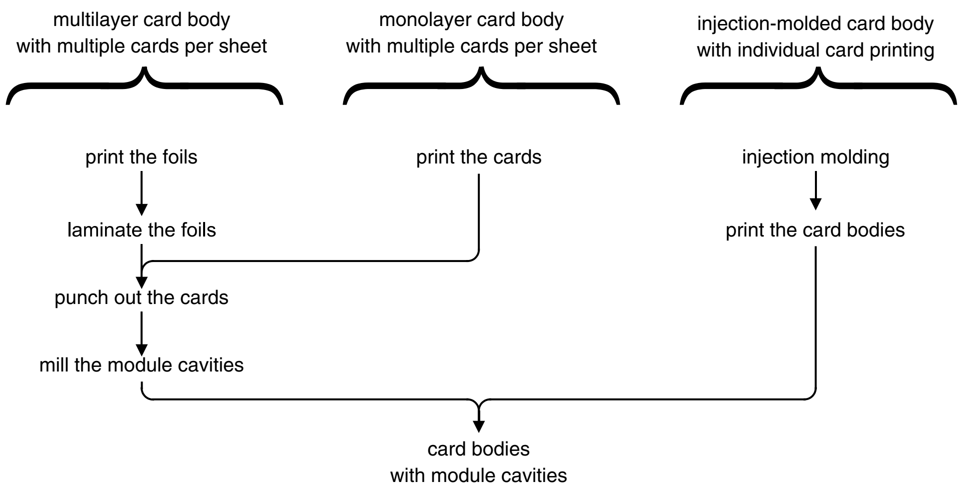


Figure 10.22 Producing card bodies for smart cards in Phase 1 of the smart card life cycle

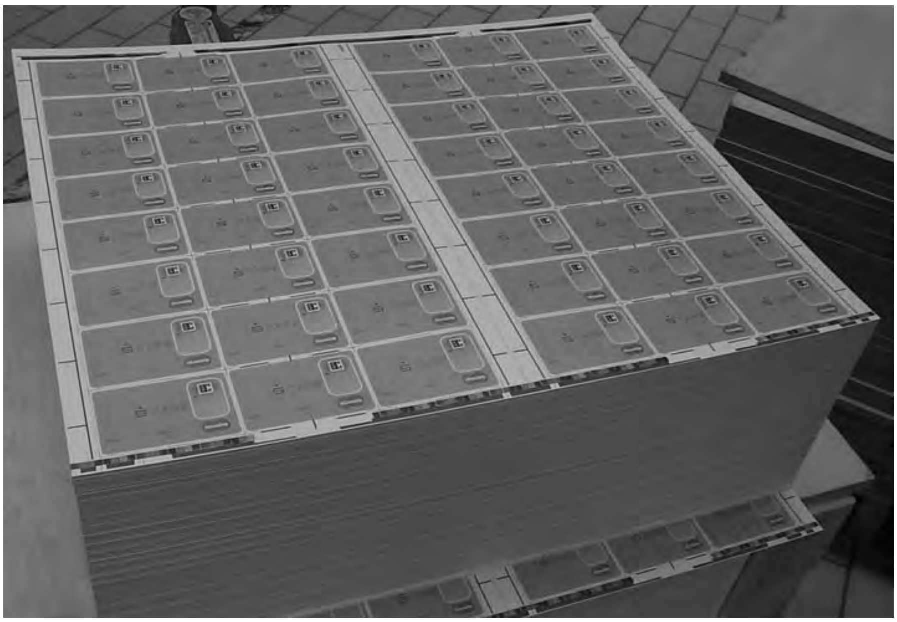


Figure 10.23 A stack of printed multiple-copy sheets, each of which holds 48 card bodies

A basic distinction is made between offset printing and silk-screen printing with regard to processes used for printing cards. Finer details can be printed on the card with offset printing than with silk-screen printing. In addition, the inks used for offset printing are hardened using ultraviolet light. This occurs immediately following printing, which has the advantage that the printed cards can be stacked right away. However, it is not possible to apply holograms or magnetic stripes on top of UV-hardened coatings using the hot-stamp process,[6] since they are not thermoplastic. A similar consideration applies to internal foils for laminated card bodies, since the lamination process requires the foils to be thermoplastic. Consequently, the silk-screen printing process is used, in which the inks harden by the evaporation of a solvent and remain thermoplastic. Additional card elements can easily be added on top of surfaces printed in this manner.

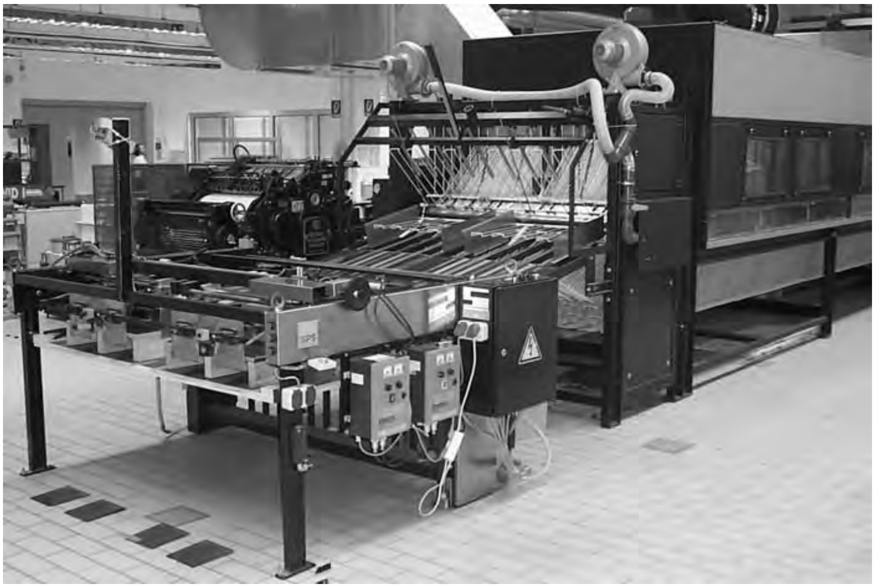


Figure 10.24 A silk-screening machine for sheet printing, which can print a multiple-image sheet for 48 card bodies in a single operation

In practice, the two printing processes are often used in combination. For example, large single-color areas and the background for the magnetic stripe and hologram can be printed using silk-screening, while the fine details can be applied in a second step using offset printing. Silk-screen printing cannot achieve such a high level of detail.

In summary, offset printing is ideal for colored subjects with high resolutions and large piece quantities. However, if additional functional elements (such as holograms) must be permanently applied to the surface of the card, either the entire card or at least the backgrounds for these elements must be printed with thermoplastic inks using the silk-screen process. Inks for offset printing cannot be used for this purpose, since they are hardened using ultraviolet light and are not thermoplastic.

A third process that is sometimes used is thermal-transfer printing, in which a piece of colored foil is released by heating and transferred to a white card body. The colored foil bonds to the surface of the card. This process is frequently used as a purely black-and-white printing process to apply serial numbers to cards, but in principle, it can be used to reproduce all colors

[6] This is a hot-gluing technique that requires a thermoplastic substrate

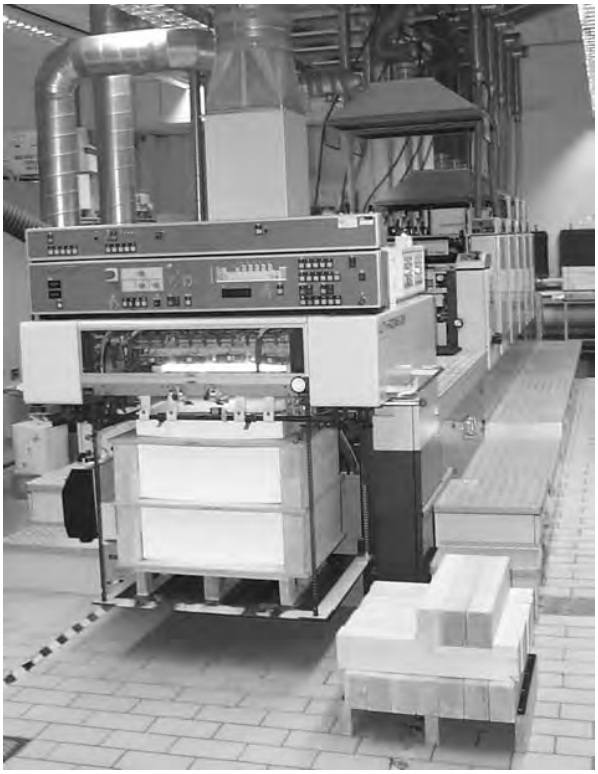


Figure 10.25 An offset printing machine for sheet printing

and shadings. However, it is slow and expensive, so it is primarily used for small quantities of cards in desktop personalization machines. It has a resolution of up to 300 dpi. An additional disadvantage of this process is that the applied colors are only bonded to the surface of the card, so they can be scratched off.

The disadvantages of colors that are only bonded to the surface of the card can be avoided by using thermal dye-sublimation printing. In this process, a print head heated to nearly 200° C presses hot dye into the top plastic layer of the card body. The maximum penetration is 5 µm, which is sufficient to make the printing scratchproof. Thermal dye-sublimation printing otherwise has essentially the same characteristics as thermal transfer printing, including high costs, so it is also suitable only for relatively small quantities of cards. Even then, both processes can generally only be used to print on suitably prepared card surfaces, which limits their use. Nevertheless, they represent an entirely worthwhile complement to just-in-time printing for small and medium-sized quantities of cards.

Laminating the foils

With a multilayer card body, after the foils have been printed they are laminated at a temperature of 100–150° C and the required features are integrated. The printed foils are protected against

Table 10.3 Summary of the most commonly used processes for printing cards

Properties	Offset printing	Silk-screen printing	Thermal transfer	Thermal dye sublimation
Sheet printing possible for mass production	yes	yes	no	no
Resolution	very good	moderate	good	good
Surface coverage of the ink or dye	good	very good	satisfactory	satisfactory
Printed surface suitable for lamination	no	yes	uncommon	uncommon
Printing resistant to scratching	no	only with a laminated overlay foil	no	yes
Card-specific printing	no	no	yes	yes
Cost	low	low	high	high

scratching and wear by supplementary transparent overlay foils laminated onto the front and rear surfaces, which are often called overlay foils. Depending on customer requirements, signature panels, magnetic stripes and security features are also embedded in multilayer cards or laminated on top of the cards at this stage.

It is certainly possible to use foils with different thicknesses for the front and rear sides of the card. In general, though, it is better to use a symmetrical construction for mechanical reasons. This means that foils with the same thickness are used for the front and back of the card. This avoids possible problems with a 'bimetallic' effect that causes the card to warp.

Punching the foils

Once the individual foils have been laminated, they must be brought to the desired card format. This is achieved using a punching process. The machines used for this have an hourly throughput of 4000–8000 cards. The burr that can be seen or felt at the edges of some cards is due to a worn punch and die set.

Milling the module cavity

A necessary part of the process of producing a card body is milling a cavity to receive the module. There are also processes in which the foils are punched out in advance to produce an opening to receive the module when the card body is laminated. The cavity is also already present in the card body if it is produced by injection molding.

Since the underside of the module has a bump where the encapsulated die is located, a matching cavity must be formed in the card body by milling out a recess. A single-level cavity is generally unsatisfactory for modern types of modules, so it is common practice to mill a cavity with two or even three levels. This provides a larger contact surface between the card body and the module, which allows the module to be durably attached to the card body. In

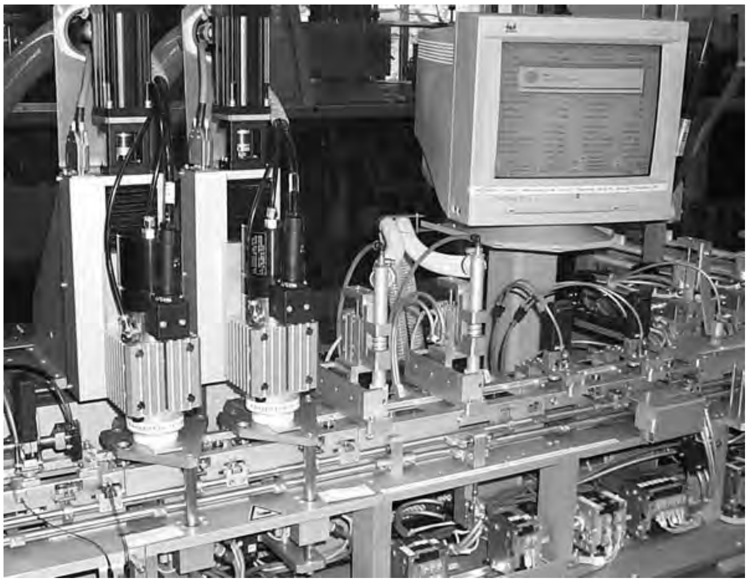


Figure 10.26 A milling machine for machining cavities in card bodies. The two upright elements on the left are electrically operated milling spindles with integrated milling swarf aspiration

addition, it is mechanically significantly better if only the rim of the module is firmly attached to the card body, with no physical contact between the die on the rear side of the module and the card body. In a manner of speaking, the module is fitted into the card so that it 'floats' in the card body.

The first step in making the cavity is to mill a recess that is as large as the contact assembly of the module and just as deep as the contact assembly is thick. Following this, an additional recess is milled in middle of the first recess to provide room for the encapsulated die. This yields a cavity with two steps.[7]

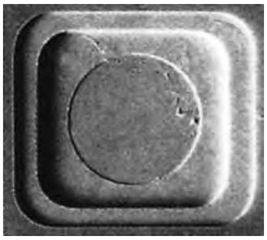


Figure 10.27 Example of a milled module cavity in a card body

The milling must be performed very precisely, since the thickness of the remaining card material at the deepest part of the cavity is only 0.15 mm. If the milling machine vibrates

[7] See also Section 3.2.2, 'Chip modules'

or rocks, the card body could be milled all the way through and thus be rendered useless. If the cavity is not deep enough, the module will stand proud of the surface of the card, which is only allowed within very narrow limits. This tricky production step is performed by fully automated machines, with the card bodies fed in from one bin and passed out to another bin. The throughput of a single machine is around 1000 cards per hour.

Printing the card body

A second type of printing process, in addition to sheet printing, is individual card printing. With this process, the cards are printed after they have been separated. When cards are printed individually, this always takes place before the cavity is milled. The throughput of machines for printing individual cards ranges up to 12,000 cards per hour. A variation of individual card printing is thermal transfer or thermal dye-sublimation printing in desktop personalization equipment. The throughput of such equipment is significantly lower, and is around 300 cards per hour at most.

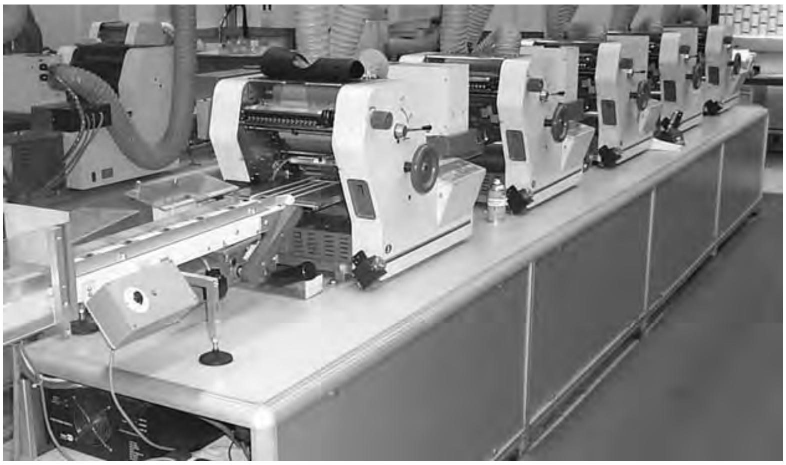


Figure 10.28 Five single-card silkscreen printing machines arranged in series

Applying card components to the card body

After the card bodies have been trimmed to size, various components such as holograms and magnetic stripes are applied to them. Holograms, which are supplied in rolls, are permanently bonded to the card body using a thermal bonding technique (hot stamping or hot rolling).[8] After this, any attempt to remove this security feature will destroy it. Magnetic stripes are applied to the card bodies using lamination or hot stamping.

[8] See also Section 3.1.2, 'Card components and security features'

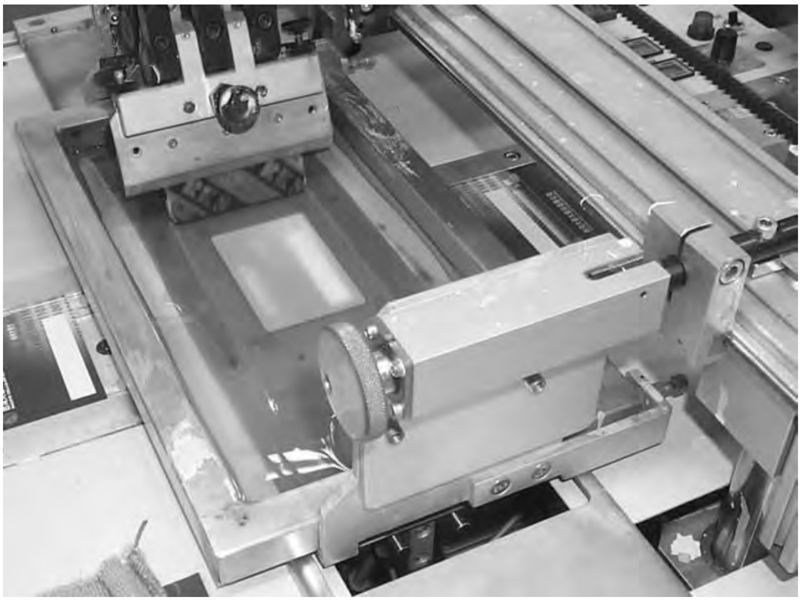


Figure 10.29 Detail of a single-card silkscreen printing machine

10.2.3 Producing card bodies containing integrated coils

Contactless smart cards need coils for transferring energy and data. At high frequencies, the coil can be made so small that it can be integrated into the chip module. With such a contactless card, the production process is thus nearly the same as for cards with contacts. The chip module with the coil is simply laminated between two or more plastic foils, or set into a cavity.

However, relatively low frequencies are used for most present-day contactless smart cards, which means that larger-diameter coils are necessary. These coils are usually rectangular with rounded corners and measure approximately 75 mm by 45 mm. This means they are only slightly smaller than an ID-1 format card body. They normally have four turns, an inductance of around 4 µH and an electrical resistance of a few ohms. Printed coils are an exception, since they have a resistance of around 300 ohms.

Producing special card bodies with integrated coils requires the standard production process to be adapted to meet the modified requirements. However, some general principles remain the same. For example, contactless smart cards are also mostly made using multiple copes for 48 cards printed on large sheets, rather than individually, since this is significantly more economical.

Connecting the chip to the coil

In the currently standard technology, the die to be connected to the coil is attached to a lead frame and electrically connected to two contacts on the lead frame, which are used to make

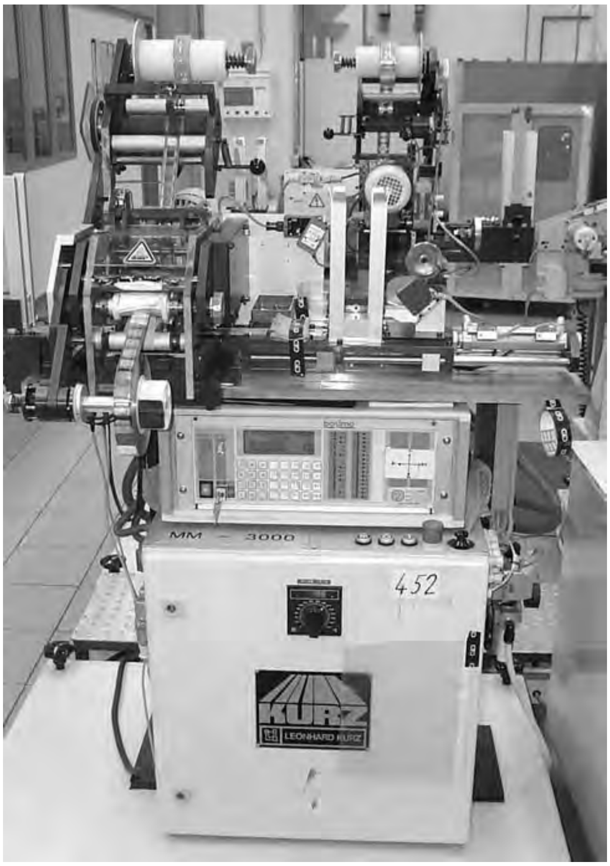


Figure 10.30 A machine for applying holograms to card bodies using the hot-stamping process

electrical connections to the two ends of the coil. The reason for using a lead frame is that the electrical connections to the coil do not have to be positioned as precisely as they would have to be if the coil were connected directly to the die. Although this method is more expensive than connecting the coil directly to the die, the latter method requires very precise positioning and high-quality bonding technology.

If the coil is integrated into the card body, two different methods are used to make the connections between the chip and the lead frame or coil. In the widely used wire bonding method, the chip is connected using fine bonding wires. A significantly more elegant and less expensive solution is die bonding, in which the chip is pressed and glued against the lead frame or coil to make a direct electrical contact. This requires the chip to be flipped over (relative to the wire bonding technique), so this method (and sometimes the chip itself) is called ‘flip-chip’. With both of these methods, the chip is covered with a protective layer of plastic resin after the connections are made.

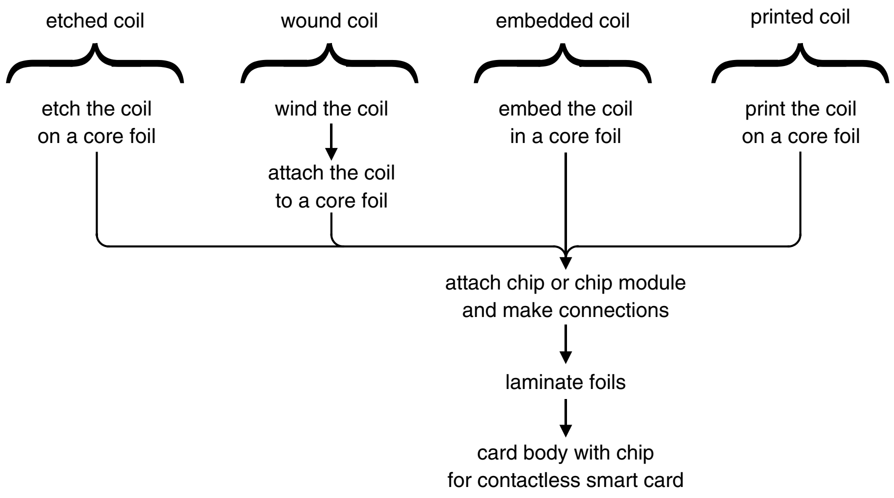

etched coil
wound coil
embedded coil
printed coil
etch the coil on a core foil
wind the coil
attach the coil to a core foil
embed the coil in a core foil
print the coil on a core foil
attach chip or chip module and make connections
laminate foils
card body with chip for contactless smart card

Figure 10.31 Producing card bodies for smart cards in Phase 1 of the smart card life cycle

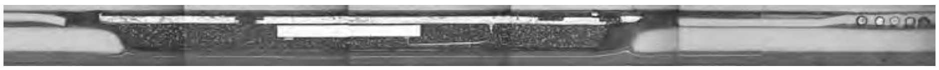


Figure 10.32 Cross-section of a contactless smart card with a module and a wound coil embedded in the card body. The visible edge of the chip inside the module is 2 mm long (*Source:* Giesecke & Devrient)

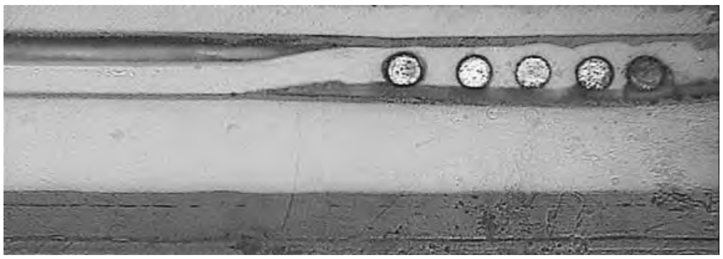


Figure 10.33 Detail of the cross-section of a contactless smart card with an a wound coil embedded in the card body (*Source:* Giesecke & Devrient)

Etched coils

There are several ways to integrate a coil into a card body. In the first process that was developed for producing contactless smart cards, plastic foils coated with a 35-µm film of copper are exposed and then etched to produce a coil in the copper film. The etched track is around 100 µm wide. After the etching, a chip is placed at the terminals of the coil and electrically connected to them. This assembly is then treated as an internal foil, to which overlay foils are laminated

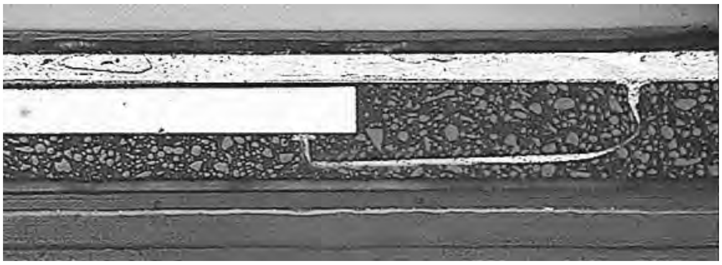


Figure 10.34 Detail of the cross-section of a chip module for contactless smart cards embedded in a card body. The bonding wire connecting the chip to the coil can be clearly seen (*Source:* Giesecke & Devrient)

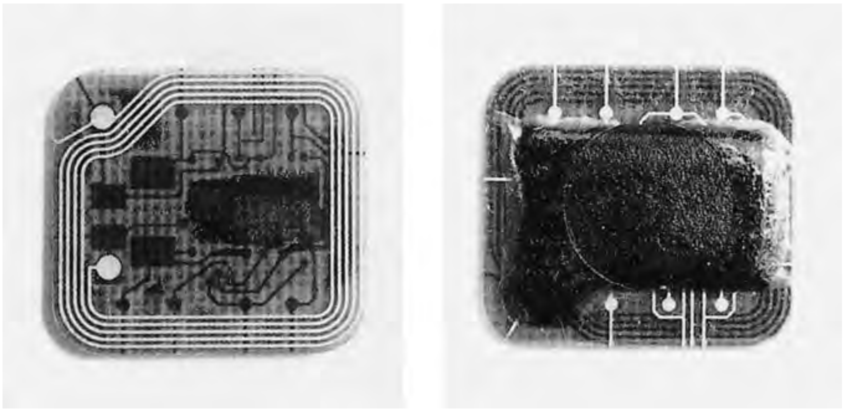


Figure 10.35 Front and rear views of a module with an integrated coil for a microcontroller chip

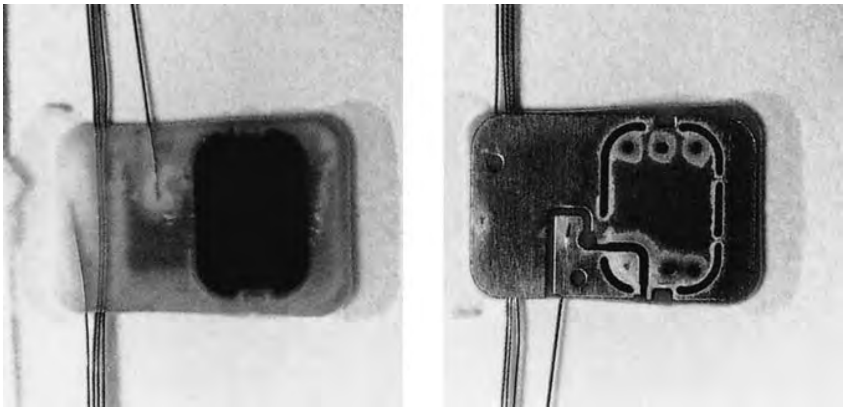


Figure 10.36 Front and rear views of a lead-frame module for a contactless smart card with the module connected to a wound coil

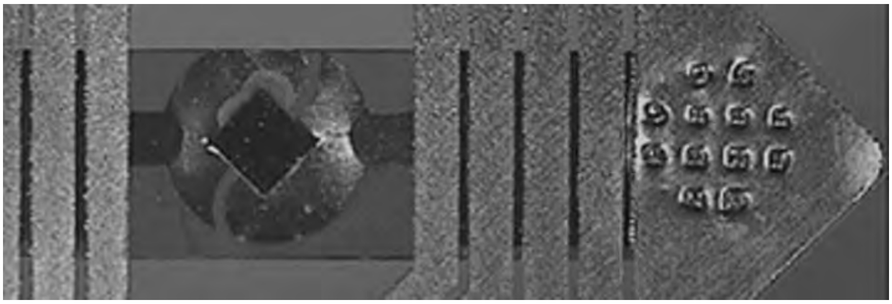


Figure 10.37 Detail of the electrical connection of a small chip to an etched coil using die bonding

on the front and rear sides. The smart card is then finished. This type of card is referred to as a contactless smart card with an etched coil.

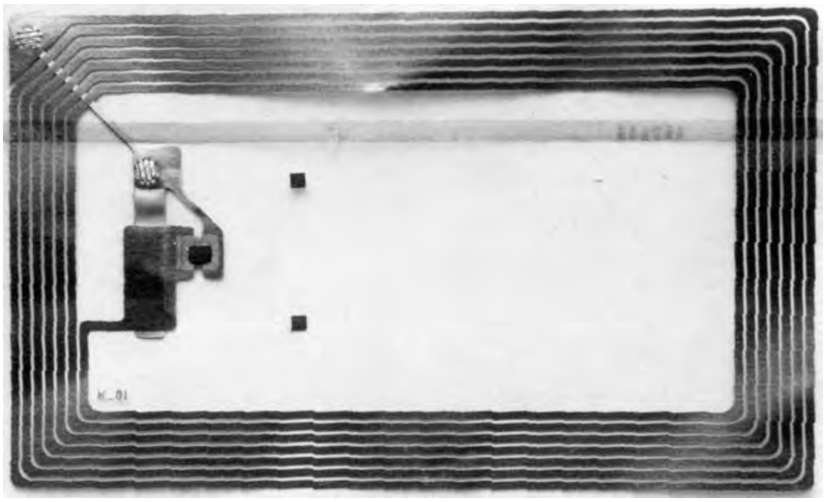


Figure 10.38 An etched coil for a contactless smart card

Wound coils

The wound coil is a further development of the etched coil. Copper wire with a diameter of 150 µm is wound on a tapered coil form and then slid from the form onto a thin foil, to which it is attached by thermoplastic welding using heat and pressure. Following this, the chip or chip carrier is put in place and the foils are laminated. This method is distinctly less expensive than etched coils, so it is more suitable for large quantities.

Embedded coils

Another method for producing wire coils is called the embedded coil method. This works in a relatively simple manner. A coil of 150 µm-diameter copper wire is laid out directly on a

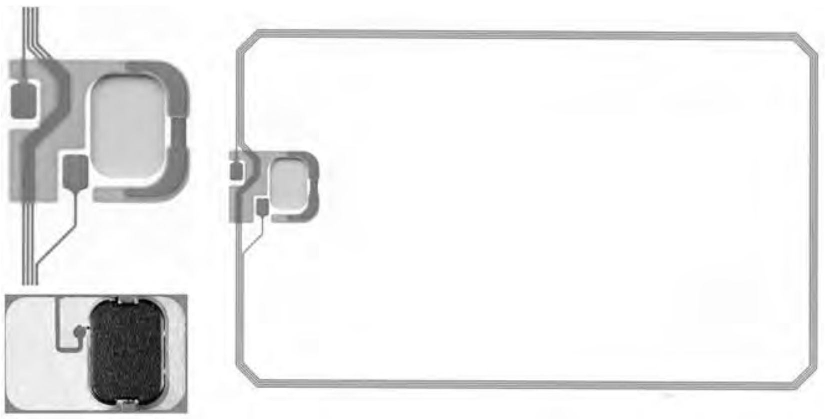


Figure 10.39 Right: inlay foil for a contactless smart card with an etched coil. Top left: enlarged detail of the module cavity and contact surfaces for connecting the coil to the module. Bottom left: corresponding module with an encapsulated chip

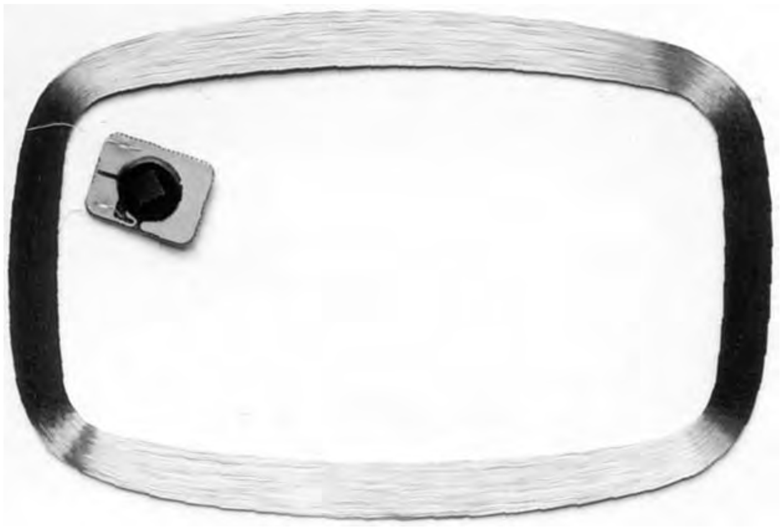


Figure 10.40 A wound coil made from copper wire together with a module, for use in a contactless smart card. This method is now obsolete, but the photo clearly shows the construction of the coil and the module

plastic foil and simultaneously bonded to the foil using ultrasonic energy. A device called a ‘sonotrode’ is used to make the coil; it mechanically guides the wire and at the same time ultrasonically welds the wire to the plastic foil. After the coil has been made, the chip or chip carrier is connected to the coil and a cover foil is laminated on top.

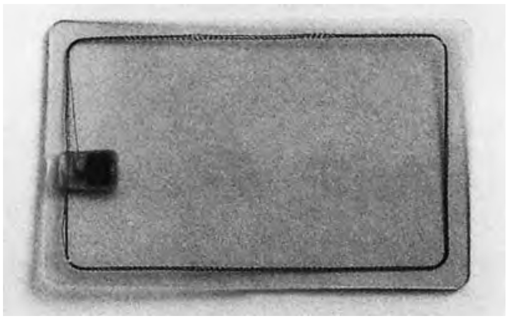


Figure 10.41 Example of a wound coil with a lead-frame module, located in a card body that has been kept transparent for demonstration purposes

Figure 10.42 Coils for contactless smart cards made using the embedded coil technique, located on a multiple-copy sheet

Printed coils

Of all of the possible methods, the printed coil method is the most highly developed and the least expensive for mass production. The windings of the coil are printed on an internal foil by silk screening using conductive ink. Silk screening must obtain an ink thickness of approximately 50 μm, so that the electrical resistance of the coil is not excessively high. After the coil has been printed, the chip is connected by die bonding and encapsulated using epoxy resin. The final production step is laminating a protective cover foil on top of this assembly. The main advantage of this method is that it permits high throughput due to the technical simplicity of the printing process. It is thus exceptionally well suited to producing large numbers of cards. However, it takes a considerable amount of expertise to achieve the necessary level of quality in printing and die bonding.

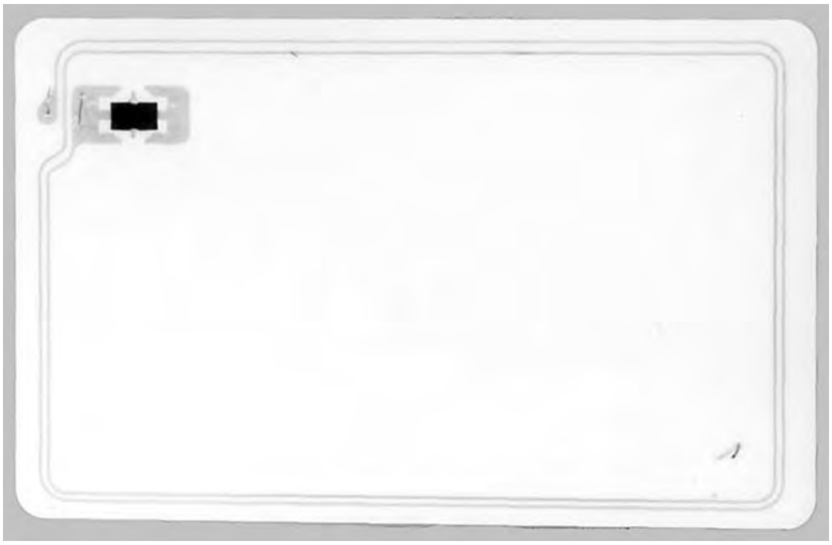


Figure 10.43 Printed coil for a contactless smart card

10.2.4 Combining the card body and the chip

The final step in the production process is implanting the modules from the semiconductor manufacturer or module producer into the prefabricated card bodies from the card manufacturer. The mechanical aspects are the most important in this step. Nonetheless, a certain amount of specialized expertise is needed to durably fit the modules into the cavities of the card bodies. This is not as simple as just pasting clippings into a scrapbook.

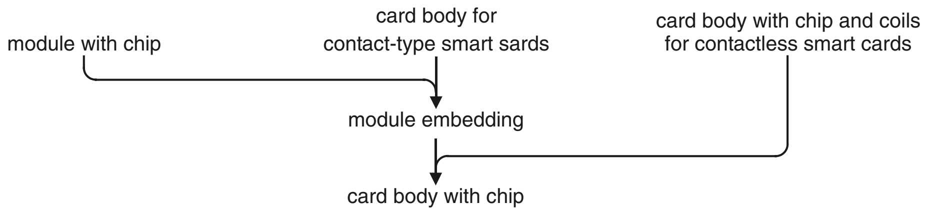

module with chip
card body for contact-type smart sards
card body with chip and coils for contactless smart cards
module embedding
card body with chip

Figure 10.44 Implanting the module in the card body for a contact-type smart card, in Phase 1 of the smart card life cycle

Implanting the module

Regardless of the method used to produce the card body and create a cavity for the module, the module must be embedded in the card body in the next step of the production process.

Normally, a piece of double-sided hot-melt adhesive tape is used to attach the module to the card body. Only the supporting surface around the rim of the module is glued to the card body, with the encapsulated die in the middle of the module remaining free. The module is thus attached to the card body such that it 'floats' within the card body. To achieve this, the adhesive tape must be pre-punched and then applied to the modules on the 35-mm carrier tape so that it covers only the edges of the modules. After this, the individual modules are separated from the carrier tape and glued into the card bodies using the attached adhesive. The durability of the bond depends primarily on the proper combination of heat, pressure and time.

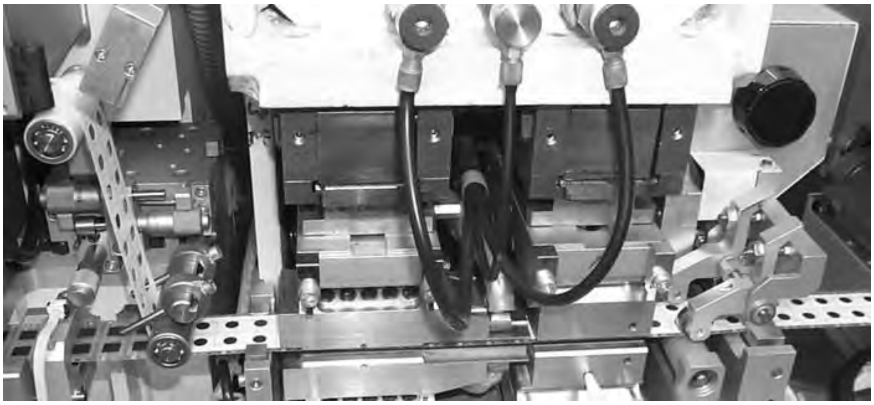


Figure 10.45 A machine for laminating adhesive tape onto modules on 35-mm carrier tape

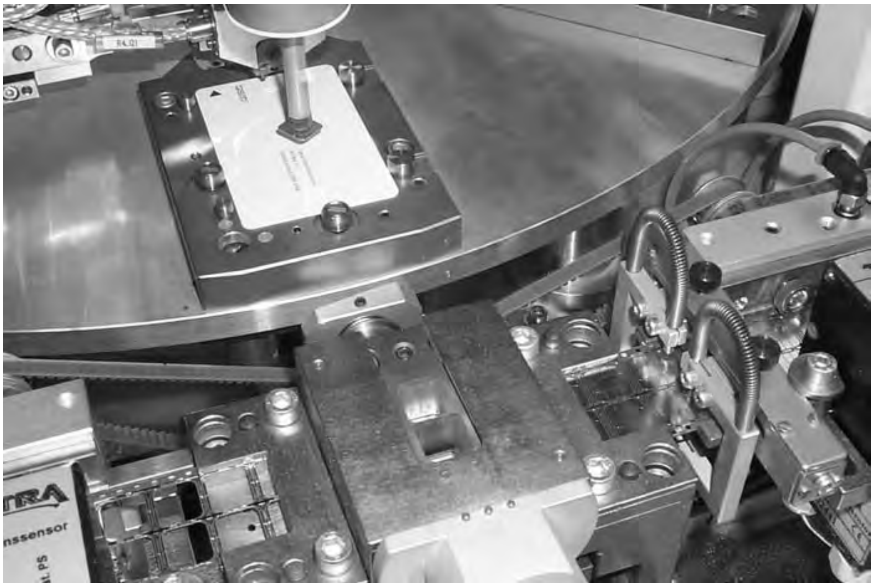


Figure 10.46 A rotary-table machine for implanting chip modules. The modules, whitch are arranged in pairs on the 35-mm carrier tape, can be seen in the foreground together with a small punch, from which they are lifted by a suction device and placed in the cavity of the card body

The problem with this hot gluing process, which requires considerable expertise, is that the modules are briefly heated to around 180° C. This normally lasts approximately one second, but if it lasts too long the modules will be destroyed by being overheated. In any case, this brief heating artificially ages the chips, although this normally does not have any negative effect. The implanting machines used for card production can process around 2000 modules per hour, which amounts to one embedding operation every 1.8 seconds.

Other methods, such as using liquid cold-setting glues, are also used, but the hot-gluing method is still considered to be very reliable. The main problems with using liquid glues that are injected into the milled cavity are the lack of a clearly defined adhesion surface and the tendency of the glue to harden over time.

Once the module has been implanted in the card body and all the non-personal features and printing have been applied to the card, the mechanical production of the smart card is complete.

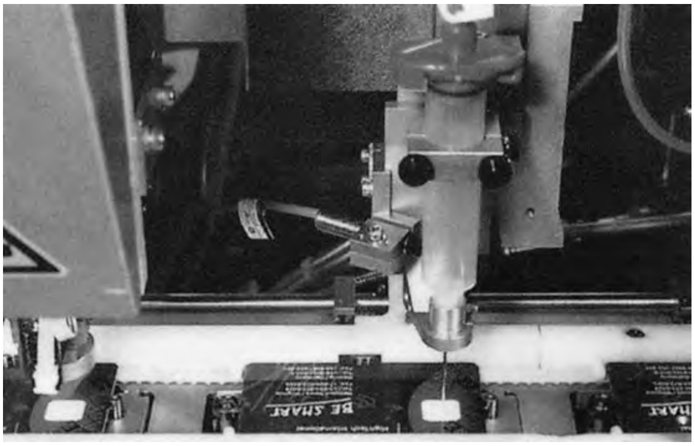


Figure 10.47 Injecting liquid glue into a pre-milled module cavity in an implanting machine (*Source:* Mühlbauer)

10.3 PHASE 2 OF THE LIFE CYCLE IN DETAIL

According to the ISO 10202-1 standard, Phase 2 of the smart card life cycle describes the loading of all data that are not card-specific as well as implanting the chips in prepared card bodies. Phase 2 and Phase 3 are frequently carried out by a single firm, although in such firms the two phases are normally fully separated, both organizationally and physically, for reasons of security.

A production planning and control (PPC) system is frequently used to coordinate these complicated production processes. The various finishing machines draw their data from this system, and in parallel with this, they report current processing status to a central control station. This minimizes the time and costs involved in controlling the mass production of smart cards. An additional benefit of the PPC system is that networking the processing equipment makes the data needed for quality assurance and testing available for near-real-time evaluation.

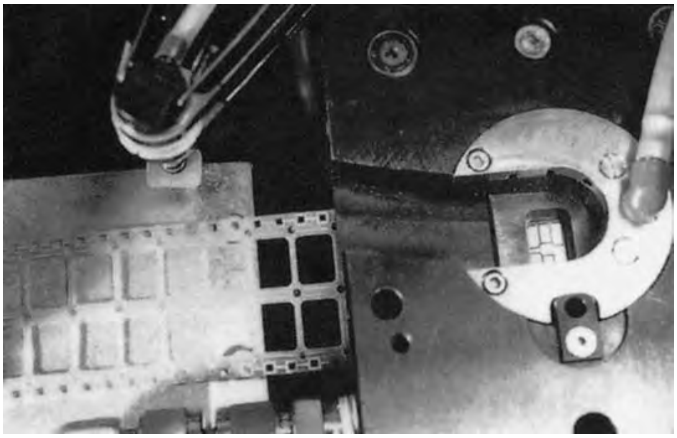


Figure 10.48 Stamping a module from the tape in an implanting machine (*Source:* Mühlbauer)

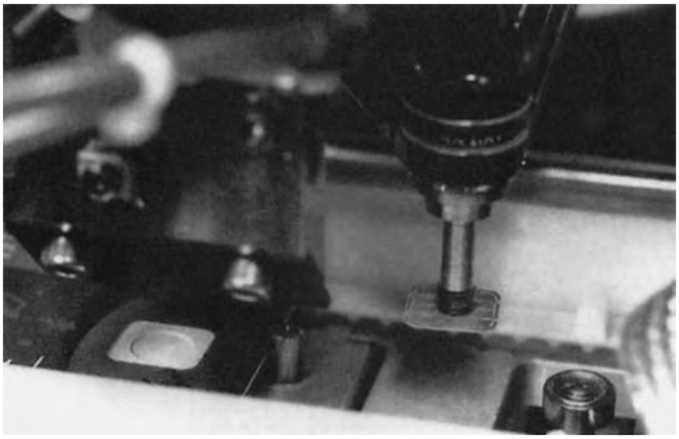


Figure 10.49 Placing a module in the module cavity in an implanting machine (*Source:* Mühlbauer)

Data transfer

The card issuer or application provider must provide the card personalizer with all the data related to his application. This includes information such as the name of the application, the structure of the file tree, the required files and the file structures. This information is loaded into the cards when they are initialized. Furthermore, the personalizer also needs all customer-specific and system-specific data, such as secret card-specific keys and the names and addresses of the cardholders. This information is transferred using diskettes, magnetic tapes or data telecommunications.

The personalization data are almost always sensitive with regard to security, which means that the transport path and data transfer must be suitably protected. Consequently, the data are normally encrypted. The associated decryption key is naturally transported to the personalizer

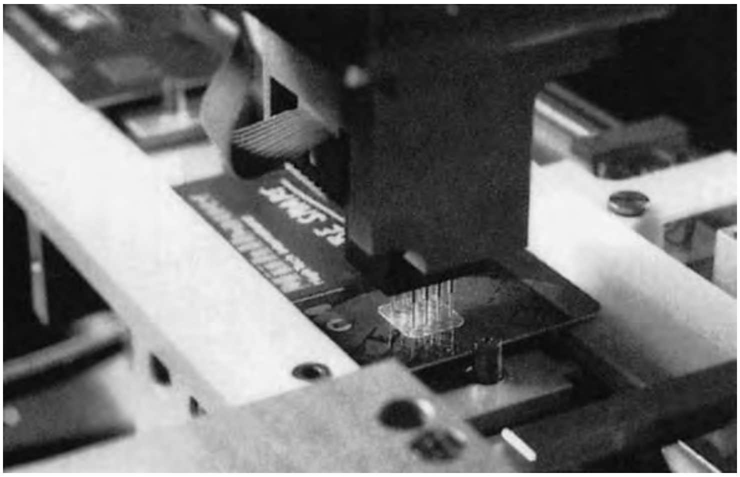


Figure 10.50 Electrically testing a module in an implanting machine after it has been glued into the card body (*Source:* Mühlbauer)

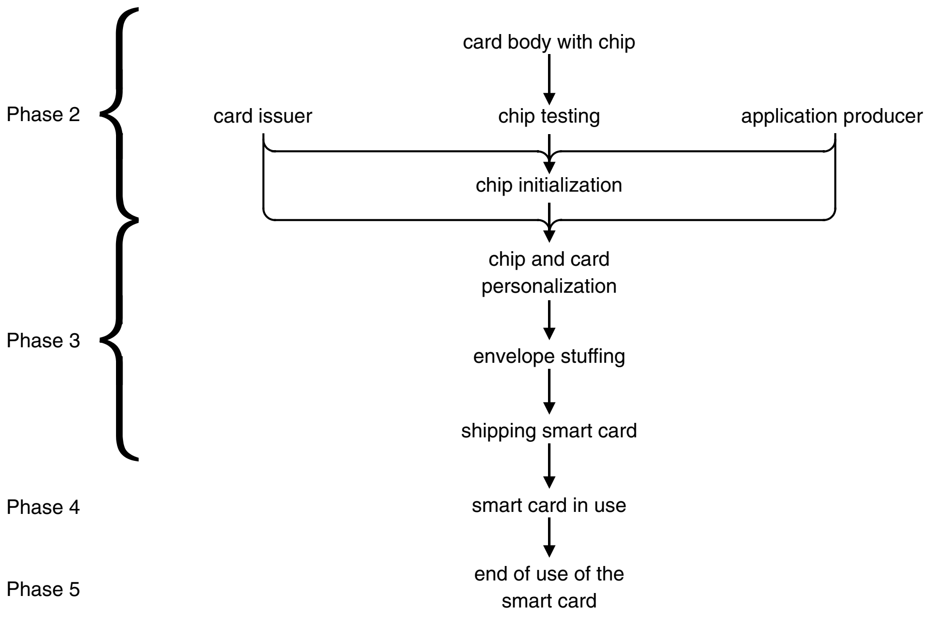


Figure 10.51 Phases 2 through 5 of the smart card life cycle: testing, initialization, personalizing, use and end of use

via a different route than the data. This means that the personalization data are worthless if they are lost, since it is not possible to decrypt them without the key.

However, there are many smart card applications in which no transfer of card-specific data takes place. The best-known example is SIMs for the GSM mobile telecommunication system, which are not manufactured for a particular card producer, but instead contain only individual card data and keys. The data sets needed for this purpose are usually generated directly by the card producer and reported back to the application operator, so that the latter knows which cards have been produced. The only sense in which a data transfer takes place is that the card producer receives the data that are the same for all cards and the initial and final values for the card-specific data. The data sets for each of the individual cards are then generated in security modules located in the finishing equipment.

Electrical testing

The first production step of this phase is an electrical test of the smart card. A basic test is made by performing the ISO smart card activation sequence, to which the card must respond with a valid ATR. If the ATR can be received and it meets expectations, it is certain that at least the core of the microcontroller is operational. Following this come special tests for the hardware components, such as the ROM, EEPROM and RAM. Special machines that can process multiple cards in parallel are used to achieve high throughput with these tests, some of which can take up to several seconds. Machines with a throughput of up to 6000 cards per hour are typically used.

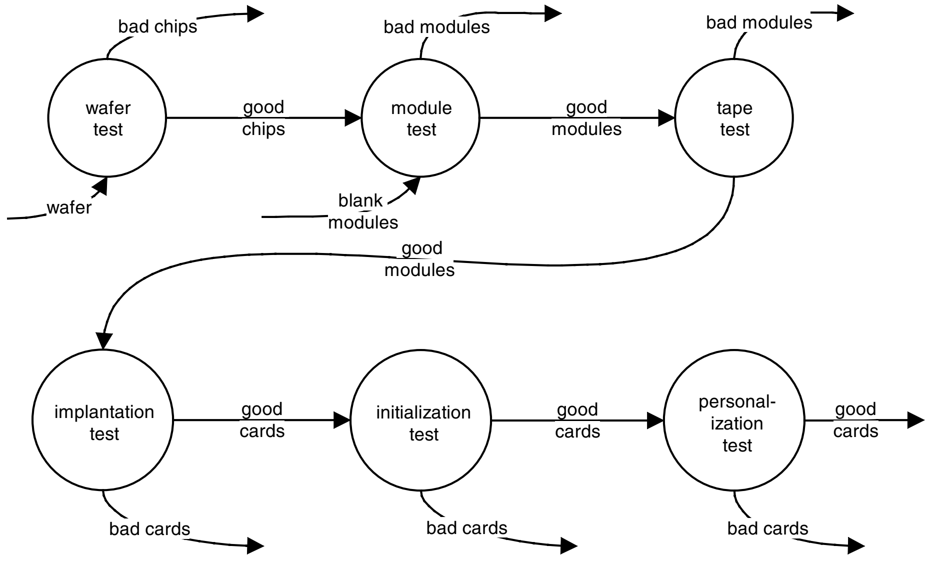


Figure 10.52 Material flow diagram showing typical electrical tests for smart card microcontrollers that are performed during production

Wafer test:	wafer test performed by the semiconductor manufacturer
Module test:	test performed by the module implanter after bonding the chip
Tape test:	testing the modules on the 35-mm tape
Implantation test:	testing the smart card after the module has been implanted
Initialization test:	testing the smart card after the card has been initialized
Personalization test:	testing the smart card after the card has been personalized

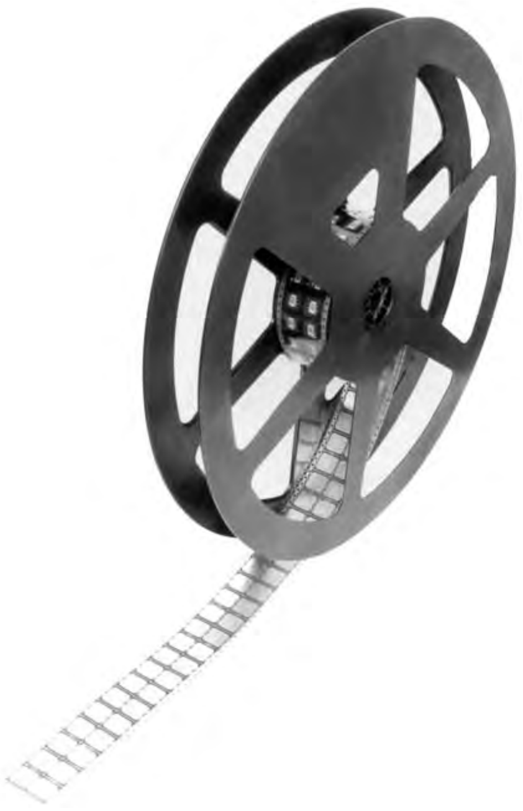


Figure 10.53 A reel with a partial roll of tape carrying paired modules. This is the typical manner in which modules are supplied to the smart card manufacturer by the module manufacturer

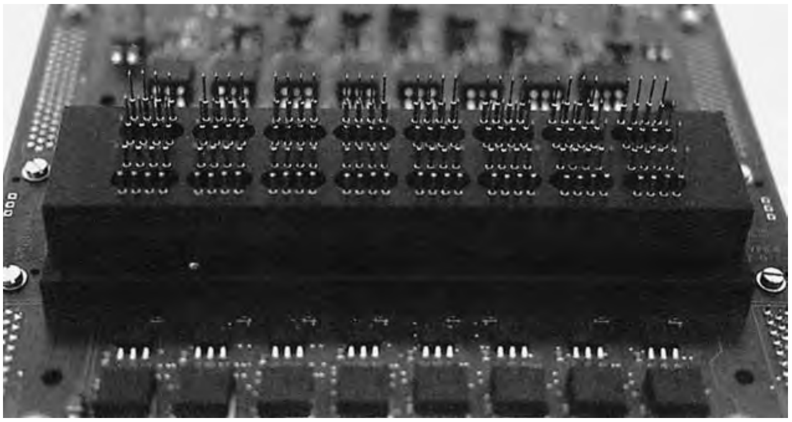


Figure 10.54 Contact head of an incoming inspection machine for modules on 35-mm tape. This machine can process 16 modules in parallel

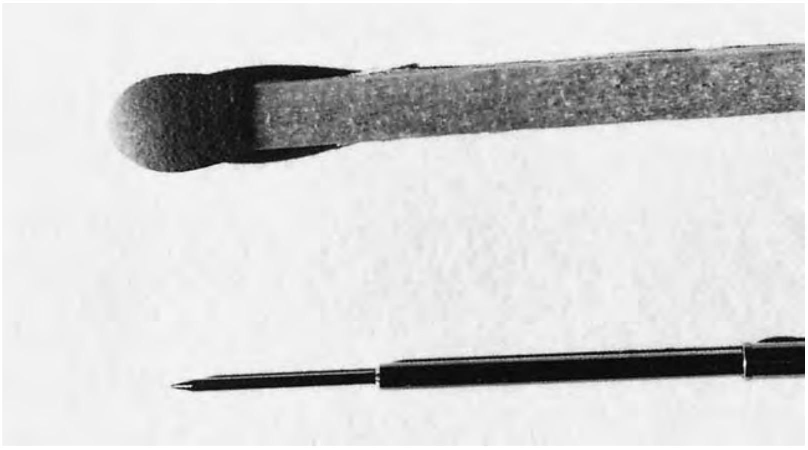


Figure 10.55 Detail view of a probe needle for the contact unit of a high-performance smart card testing station. Although the shape of the probe needle does not conform to the ISO/IEC 7816-2 standard, this type of contact element is often used in production facilities due to its reliability

The preferred way to test the operation of the EEPROM is to write a ‘checkerboard’ pattern, such as 'AA' (°1010 1010°) or '55' (°0101 0101°), to the individual bytes. However, since this would take a long time, particularly with large EEPROMs, a trick is sometimes used to shorten the test. Instead of using the specified EEPROM write time, which might for example be 3.5 ms per page, only one-tenth of this time is used (350 μs in this example). Data will be retained in the EEPROM for only a few minutes when such a short write time is used, but this does not cause any problems here, since the checkerboard-pattern memory test is completed a few seconds after the data have been written. The advantage of this dynamic form of EEPROM programming is that it significantly speeds up testing without reducing the quality of the testing. The same technique is sometimes used when the transmit and receive buffers of the I/O manager are located in EEPROM instead of RAM. In this case, the reduced write time yields a marked increase in the effective data transmission rate.

There is another interesting trick that is used in electrical testing. In order to reduce the amount of time required to load data in subsequent production steps, a final test pattern (such as '00') is written to the entire EEPROM using the normal write time. Since the value already stored in the memory is known in the subsequent processes of completion, initialization and personalization, only the data that are different from this value have to be actually written to the EEPROM. A similar technique can be used to set the contents of the EEPROM to a value that makes it unnecessary to first erase the page to be written for subsequent write operations. Both of these tricks distinctly reduce the times required to carry out subsequent production steps in which data are written to the EEPROM.

Completion

Most operating systems are only partially contained in the mask-programmed ROM of the smart card. The link tables and portions of the program code are loaded into the EEPROM of the smart card only after an authentication using a secret key. The process of loading the EEPROM portion of the operating system is called completing the operating system.[9]

This approach allows minor modifications to be made to the ROM program code, in order to correct errors or adapt the code to special applications without being forced to generate a new ROM mask. The smart card operating system is not fully present in the smart card until the EEPROM data have been written to the card. After this, it is possible to execute all application commands, such as SELECT and READ RECORD.

Card completion, which involves data that are the same for all cards for a particular application, is performed using high-throughput machines that process multiple cards in parallel, just as with the incoming inspection of cards.

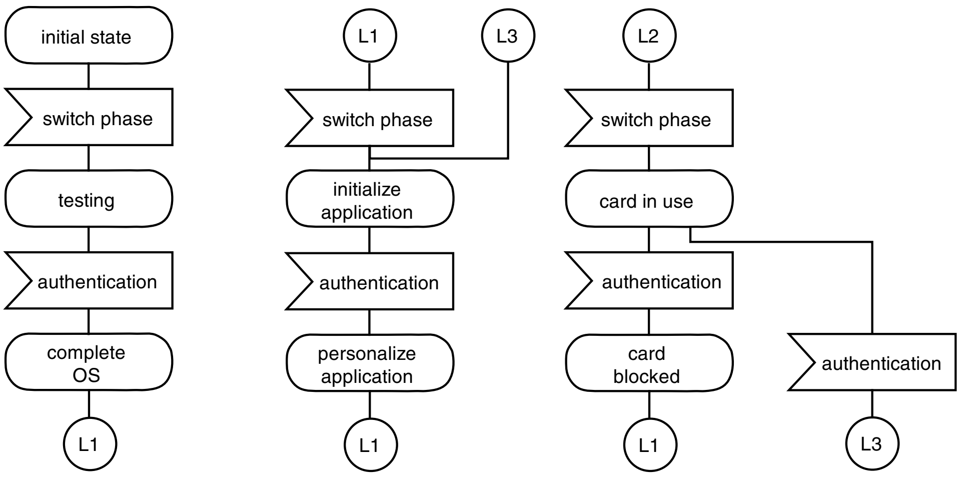


Figure 10.56 Smart card operating system state machine for implementing a five-phase life cycle

Initialization

Completing the card provides it with the software that is necessary for the next production step, which consists of loading all the data belonging to an application that are the same for all smart cards used with that application. This consists of the application data that do not vary from card to card and all other non-personal data that are the same for every smart card. This step is called initialization.

At the file level, initialization consists of creating all necessary files (MF, DFs and EFs) and filling them as much as is possible with the application data. In many cases, the file contents are predefined by the applicable specifications (such as GSM 11.11). With modern operating systems, initialization is performed using the CREATE, UPDATE BINARY and UPDATE RECORD commands. This is the last processing step in which all smart cards are treated the

[9] See Section 5.4, 'Completion'

same. Consequently, initialization can also be performed using fast parallel machines. Card-specific application data and personal data are not loaded into the smart card until the following step, which is called personalization.

Figure 10.57 An initialization machine that can process 40 cards in parallel (*Source:* Mühlbauer)

The reason for distinguishing between general, global data and specific, personal data in the finishing process relates to minimizing production costs. Personalization machines that can write specific data to each individual smart card under the required security conditions are technically complex and have a throughput of around 700 cards per hour. They are also usually equipped with relatively slow labeling units for the card bodies. This results in high unit costs for loading data into the cards. Consequently, an attempt is always made to load all global data, which does not differ from card to card, into the cards using simpler and faster initialization machines, which can process around 3500 cards per hour.

The bottleneck for both initialization and personalization is transmitting the data to the card and writing it to the EEPROM. The time required for write accesses to the EEPROM cannot presently be reduced, due to technical limitations. However, the time required to transmit the initialization and personalization data can be drastically reduced by increasing the clock rate and reducing the divider value. For example, many initialization and personalization machines use data transmission rates of up to 115 kbit/s, instead of the usual value for smart cards of 9600 bit/s. This can reduce the initialization or personalization time by a factor of nearly two.

The following sample calculation clearly illustrates that even small time optimizations can be worthwhile in the mass production of smart cards. Here we assume that one million cards are to be initialized with 4 kB (4096 bytes) of data each, using two initialization machines operating for two shifts (16 hours) per day. We also assume that initialization is performed using 40 commands and the T = 1 transmission protocol, with 12 data bits for each byte of transmitted data. In addition, the EEPROM write cycle time is 3.5 ms for a 4-byte page, and a prior erase operation is not necessary. The transport time for the initialization machines, which are not equipped with terminals for parallel processing, is 1 second per smart card, and any dead time that may occur (for loading or emptying bins, for example) is not taken into account. The resulting cycle time is thus the sum of the EEPROM writing time, the data transmission time and the transport time.

Using the formulas given in Section 15.2 ('Formulas for Estimating Processing Times') with a data transmission rate of 9600 bit/s, we obtain a processing time of 90.7 days. If the data transmission rate is increased to 38.4 kbit/s, the time required to process one million cards drops to 52.5 days. A data transmission rate of 115 kbit/s would be ideal, since at this rate card production could be completed more than 46 days earlier than at 9600 bit/s.

Table 10.4 Processing time for initializing smart cards using various data transmission rates. The basic assumptions and general conditions are described in the text

Data transmission rate:	9600 bit/s	38,400 bit/s	115,200 bit/s
EEPROM writing time:	3584 ms	3584 ms	3584 ms
Data transmission time:	5870 ms	1468 ms	489 ms
Resulting cycle time:	10,454 ms	6051 ms	5073 ms
Resulting processing time:	90.7 days	52.5 days	44.0 days

From this example, it is clear that particularly when a large amount of data must be stored in the smart cards, it is worthwhile to invest time and effort in optimizing the processing. The described increases in the data transmission rate depend only on the smart card operating system and do not require any special chip hardware, such as would be necessary for writing the data to the EEPROM faster. Consequently, it is possible to reduce the initialization time for all suitably prepared smart cards.

10.4 PHASE 3 OF THE LIFE CYCLE IN DETAIL

Phase 3 primarily covers the part of the life cycle consisting of the visual and electrical personalization of the smart card. As with Phase 2, this phase normally occurs in a highly automated production environment that is designed for processing large numbers of cards.

Generating card-specific secret data

As a rule, the individual data for personalization are provided by the card issuer on a data storage medium or via data telecommunications. However, a special method is often used for providing secret data, such as PINs and keys, since such data must remain secret under all circumstances and are only allowed to be generated in highly secure environments. There are four methods that are used in practice for PINs.

The simplest option is to generate a trivial PIN, which the cardholder must change to a PIN of his or her choice the first time the card is used (and before actually using the card for a valid transaction). However, for a variety of reasons this method cannot be implemented in all systems, even though it has the advantage of not requiring the printing and posting of PIN letters.

A somewhat more elaborate option for producing PINs is for the card issuer to generate the PINs using a good random number generator, followed by secure transfer of the PINs to the card personalizer. The latter then writes the PINs to the cards to be personalized using the usual secure mechanisms and generates the associated PIN letters. A variation of this option is to generate the PINs in the cards, followed by the secure transfer of the PINs to the personalization machines for use in further processing.

The third possibility is generation of random PINs by the card personalizer. These PINs, which are generated in a secure environment, are written to the appropriate data fields in the smart cards, as with the previous options. In parallel with this, PIN letters are generated and sent to the cardholders. The associated smart cards reach the cardholders via a separate path. If the card issuer needs to have the PINs that have been generated in this manner, they can be provided to him in a secure manner. Otherwise it is generally not necessary to store the generated PINs anywhere except in the smart cards.

Another way to generate PINs is to use an algorithm, which may be a cryptographic algorithm, to compute card-specific PINs using data present in the cards and a master key. The drawback of this method is that the master key and (in some cases) the algorithm must be kept secret.[10]

If the secret data to be generated are not PINs, but instead keys for cryptographic algorithms, essentially similar methods can be used. The principal difference is that in this case it is not necessary to generate PIN letters, although the keys must be provided to the system operator in a secure manner. This is done using what is called the 'response data', which are transferred from the party that generates the keys to the system operator in a cryptographically secured form via data telecommunications or a physical data storage medium.

Transferring data to the smart card

There are two fundamentally different methods that can be used to store the initialization data in the memory of the microcontroller. The first method, which aims to avoid direct physical addressing of the memory, uses only logical addresses in the microcontroller for initialization and personalization as much as possible. From a purely theoretical perspective, this is the preferable method, since it avoids the need to use physical addresses outside of the smart card. This automatically eliminates many potential sources of errors, and within certain limits it also makes loading data into the smart card independent of the type of microcontroller present in the smart card. The drawback of this approach is that it significantly increases the time required for initialization and personalization, and particularly in the case of mass production, time is a very critical factor.

Consequently, there is a second method that is used in practice to load data into smart cards, which involves writing the initialization data directly to the microcontroller memory using externally specified physical addresses. This significantly reduces the amount of time required compared with a method based on logical addresses. Unfortunately, with this approach it is necessary to work with physical addresses external to the card, which carries corresponding drawbacks with regard to susceptibility to error and general usability. In practice, the method used is generally determined on a case-by-case basis. If the number of smart cards to be produced is sufficiently large, the increased cost of the software for the initialization machinery and the necessarily complicated testing can be justified.

[10] An example of this method for generating PINs is given in Section 8.1.1, 'Verifying a secret number'

In order to write data directly to physical addresses, the data must be suitably prepared in advance. One way to do this involves mimicking the complete file management system of the smart card operating system in the form of a simulation. A conversion program can then be used to load the data to be written into the appropriately coded file bodies of the simulation and provide them with their associated file headers. After this, all that is necessary is to relocate the files constructed in this manner to the proper addresses in memory. Naturally, the entire process must be performed without errors and in a manner that is matched to the operating system in question. Following this, the data can be read from the simulation and directly written to physical memory addresses in the smart card, using the usual commands.

Unfortunately, this approach has not proved to be worthwhile in practice, since its costs far outweigh its potential benefits. In addition, the cost of testing to ensure the absence of errors, using expensive black-box tests, would be excessive. Consequently, this method is rarely used.

The commonly used method is much simpler. A smart card containing a dump routine in an otherwise unused area of memory is first initialized using file management commands, which use logical addresses. The initialized memory is then read out using the dump routine, and the data so obtained are written to the physical addresses of the smart card to be initialized. This allows initialization and personalization times to be reduced by as much as 30 %. In principle, this method can be considered to be cloning. Its major advantage is that it is simple and robust, and the only critical aspect is that the smart card containing the dump routine must never be allowed to leave the processing facility. If this smart card were fully personalized, the dump routine it contains could be used to read out all of the secret data. Consequently, this smart card has suitable mechanisms to prevent it from ever being misused for reading out memory as the result of an exchange or an attack. This can be achieve relatively simply, for example by having the dump routine automatically delete itself the next time the card is reset. In this case, the smart card can be used only during a single session, since it will lose its dump capability the first time its supply voltage is interrupted.

Personalization / individualization

The next step in producing a smart card that is ready to be sent to the user is personalization, which is sometimes called individualization. In a more general sense, personalization means loading all data assigned to a particular person or a particular card into the smart card. This might be a name and address, for example, but it could also be card-specific keys. What is important is that the data are specific to a particular card.

A basic distinction is made between visual and electrical personalization. The embossing characters, as well as text or pictures applied to the card using laser engraving, constitute the visual part of personalization. The electrical part consists of loading personal data into the microcontroller and writing data to the magnetic stripe. The processing time for visual personalization depends very strongly on the specific features and cannot be generally stated. Electrical personalization usually takes between 5 and 20 seconds, depending on the amount of data.

Embossing names and similar card-specific, character-based information is performed by a machine in which metal letter punches are hammered against the rear of the card at great speed and with considerable force. Since this is a relatively simple procedure, but one that is very loud and produces a lot of vibration, the embossing machines are usually physically separated

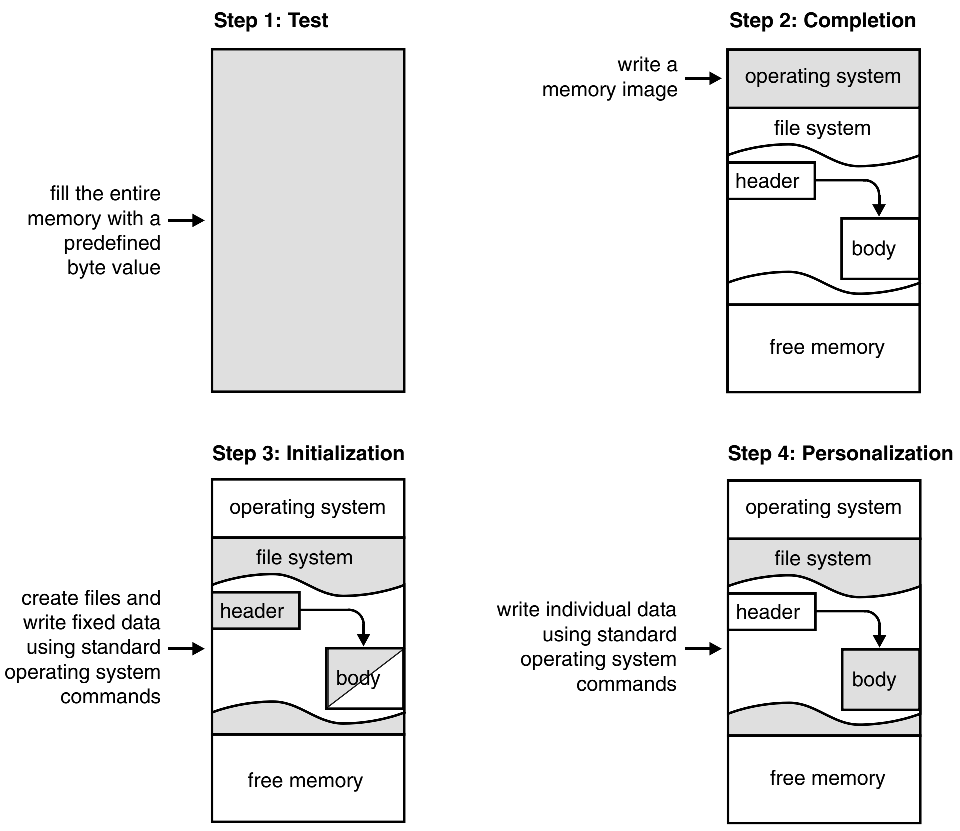


Figure 10.58 Schematic representation of the most important steps for loading common and individual data into a smart card using file management commands such as CREATE, UPDATE BINARY and UPDATE RECORD. The shaded regions mark the data written in each step. For simplicity, this diagram depicts the loading of only one file system. Similar steps would be used to load other items, such as Java applets

from the rest of the processing equipment. Laser engraving equipment, which can be used to darken regions just below the overlay foil of the card body using a laser beam, is very often employed instead of mechanical embossing. This technique is also useful if it is necessary to have a black-and-white picture on the card body.

The data for the chip are written to the memory in the same way as for initialization. However, to the extent that this involves secret keys, cryptographically protected data transmission[11] is often used to prevent an attacker from deriving any benefit from tapping the data line. For cards that are used for financial transactions, an even more complex method is sometimes used. This involves using a special security module in the personalization machine to re-encrypt the encrypted personalization data received from the card issuer and then load it directly into the

[11] See also Section 6.6, 'Securing Data Transmission'

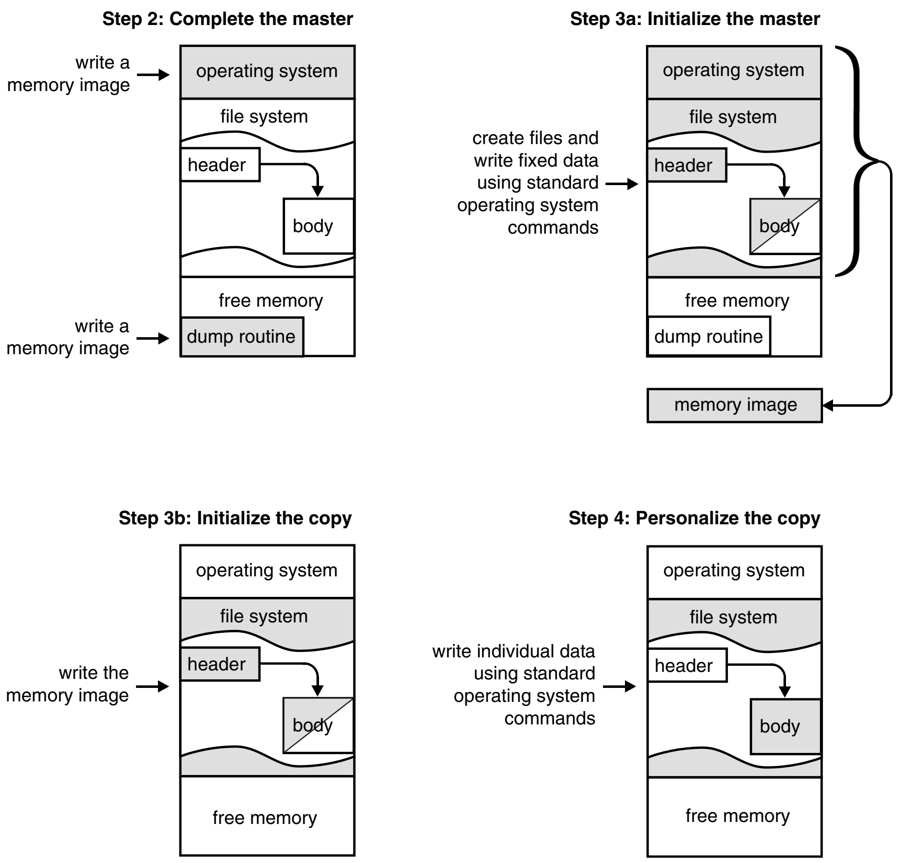


Figure 10.59 Schematic representation of the principal steps for loading common and individual data into a smart card by physically copying the data from a master version previously generated using file management commands. Figure 10.58 shows further details of the processes used to generate the master version and the copy. The numbering scheme used here is also taken from that figure. The shaded regions mark the data written in each step. For simplicity, this diagram depicts the loading of only one file system. Similar steps would be used to load other items, such as Java applets

smart card. The advantage of this method is that the personalizer does not know the secret data in the card and also has no possibility of spying it out by tapping the data lines.

The technical trend in smart card personalization is increasingly heading in the direction of using a process that is cryptographically fully secured. This means that in principle the work can be performed by inexpensive service firms in non-secure facilities. Nowadays, there are also processes in which the personalizer receives the card-specific data recorded on a CD-ROM. In this case, the production data set with its associated card-specific key is inseparably linked to the unique chip number of the microcontroller. Among other things, this makes it impossible for the personalizer to produce duplicates of smart cards, unless he can somehow manipulate the operating system. However, this method has the disadvantage that some of the

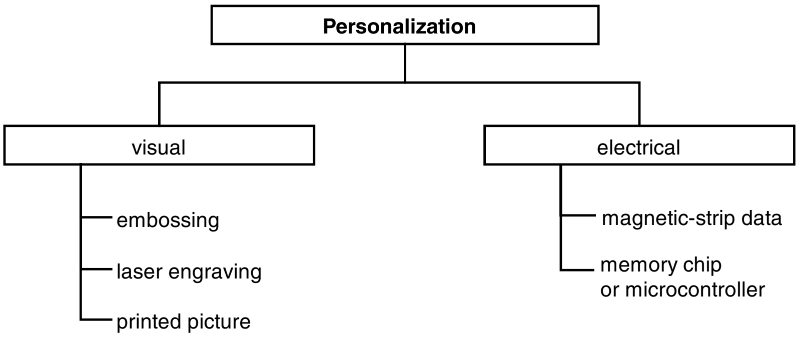


Figure 10.60 Classification of the elements of a smart card that can be personalized

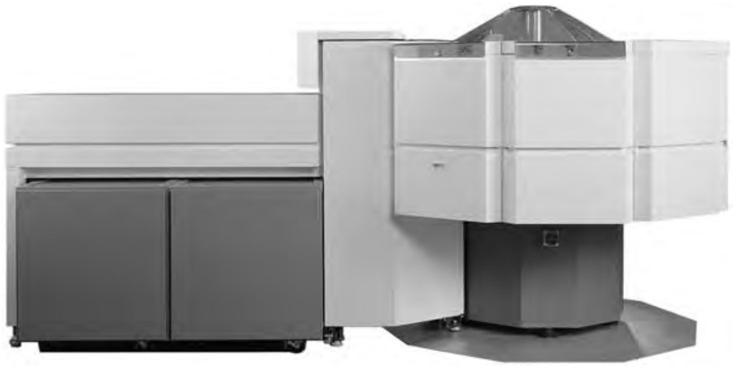


Figure 10.61 A modular personalization system (Luchs 5000), which includes an integrated laser labeling unit (*Source:* Giesecke & Devrient)

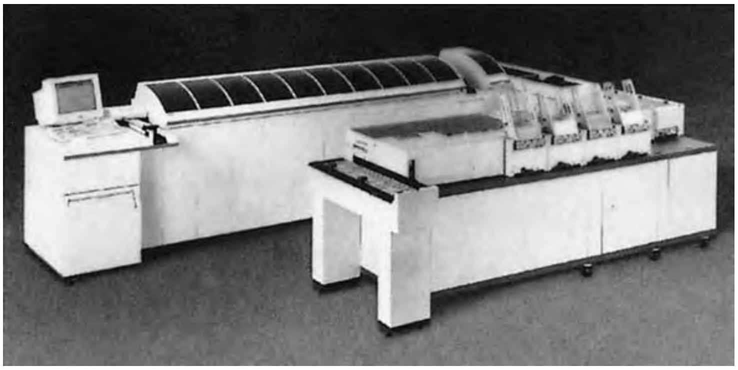


Figure 10.62 A modular personalization system (Data Card 9000), which includes integrated postal processing (*Source:* Data Card)

delivered data sets cannot be used if any of the chips are faulty, since the defective chips are no longer available. If this method is used, the personalizer must always report back to the party that generated the data to inform them which chips have actually been processed. This is not necessary with the personalization methods that are presently in common use, since it is easy to reproduce a faulty card. Incidentally, this is also why the personalization facilities of card producers are always secure areas.

Unfortunately, the cryptographic procedures and security measures used in the realm of personalization are largely secret, so it is not possible for us to describe any specific application. However, Figure 10.63 shows an example of an initialization process followed by a personalization process, as seen from a cryptographic perspective. For the cryptographic protection to be effective, these two production steps must take place in separate rooms using separate personnel.

The illustrated procedure works as follows. During initialization, a card-specific key (KD) is derived in a security module using a unique chip number and a master key (KM). This key is sent as plaintext to the card, where it is stored. Naturally, a lot of other data must be written to the smart card during the initialization, but generating and storing the card-specific key KD is the only cryptographically relevant step.

Following this, the card is personalized. This can be done immediately following the initialization, but it may also be done several weeks later. The important factor is that personalization must be completely separate from initialization, in order to prevent a KD that has been illicitly acquired during initialization from being used during personalization to decrypt the card-specific data.

In the personalization process, the personalization data that have been encrypted using a shared key are decrypted for each individual card by the security module. This is necessary because the producer of the personalization data does not know the individual chip numbers, which are independently generated by the semiconductor manufacturer. The security module then computes the card-specific key (KD) from the card number that it receives from the smart card and the master key (KM). Now the security module and the smart card have a shared secret in the form of KD. This is used to encrypt the personalization data, which are then transferred in encrypted form to the smart card, where they are decrypted and written to the appropriate locations in the EEPROM. This process provides complete cryptographic protection of the personalization procedure. It protects the data to be used for personalization against being spied out, as long as the key (KD) that is written to the card during the initialization remains secret.

Figure 10.64 shows an alternative method for securing loading data into smart cards, in which the first step consists of having the smart card and the terminal agree on a common secret key by means of a Diffie–Hellmann key exchange. After this, the data are transmitted to the smart card in encrypted form using this key. The major advantage of this method is that it never involves transmitting a secret key in non-encrypted form.

At the conclusion of the personalization process, the personalization machine runs several quality control tests on the finished smart card. In the latest machines, for example, each card is scanned by a camera and the visual personalization is evaluated by a computer and checked against a production database. In case of an error, the card is ejected into a faulty-card bin and a new copy of the card is automatically produced. Normally, the personalization data in the microcontroller are also checked. However, this is technically difficult to do, since read

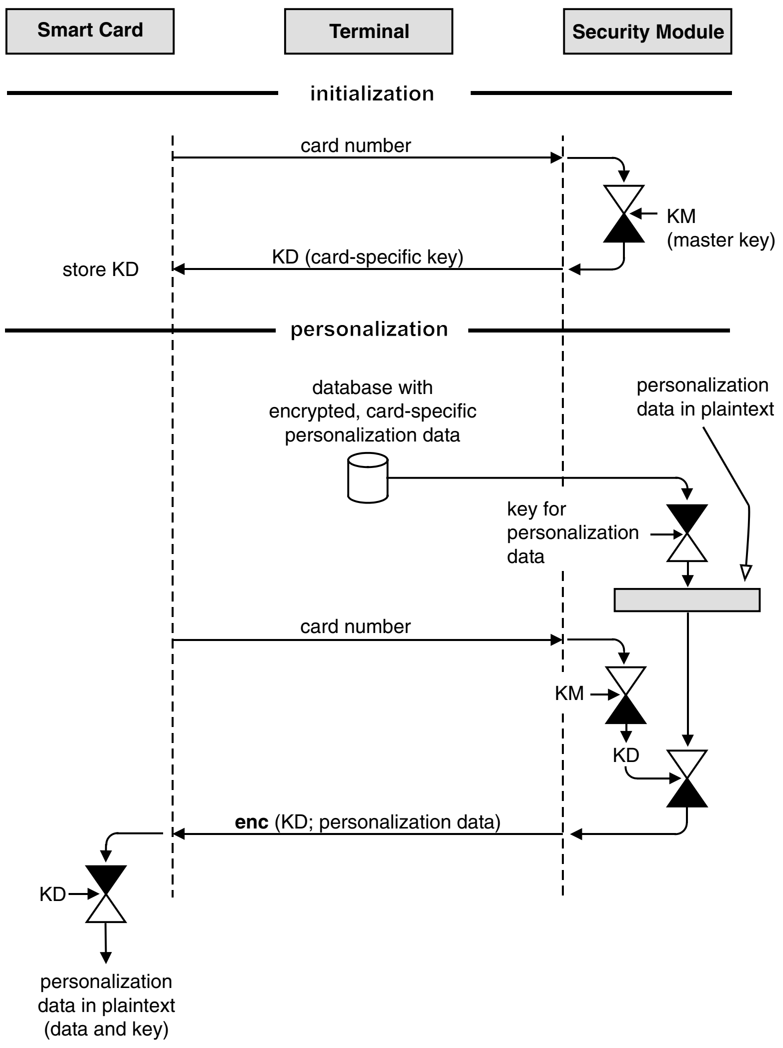


Figure 10.63 Schematic representation of a typical initialization and personalization procedure using cryptographically secured transmission of data and keys. 'KM' designates the master key, which is used to derive the card-specific keys (KD). Only the cryptographically relevant processes are shown

access to many of the files is no longer allowed. Consequently, special security modules for these tests are frequently present in personalization machines. These modules contain secret master keys with which the personalized keys in the smart cards can be tested for correctness, possibly via an authentication.

Another approach is to provide the personalizer with command strings and corresponding response strings for each individual card. The personalizer then sends these commands in the

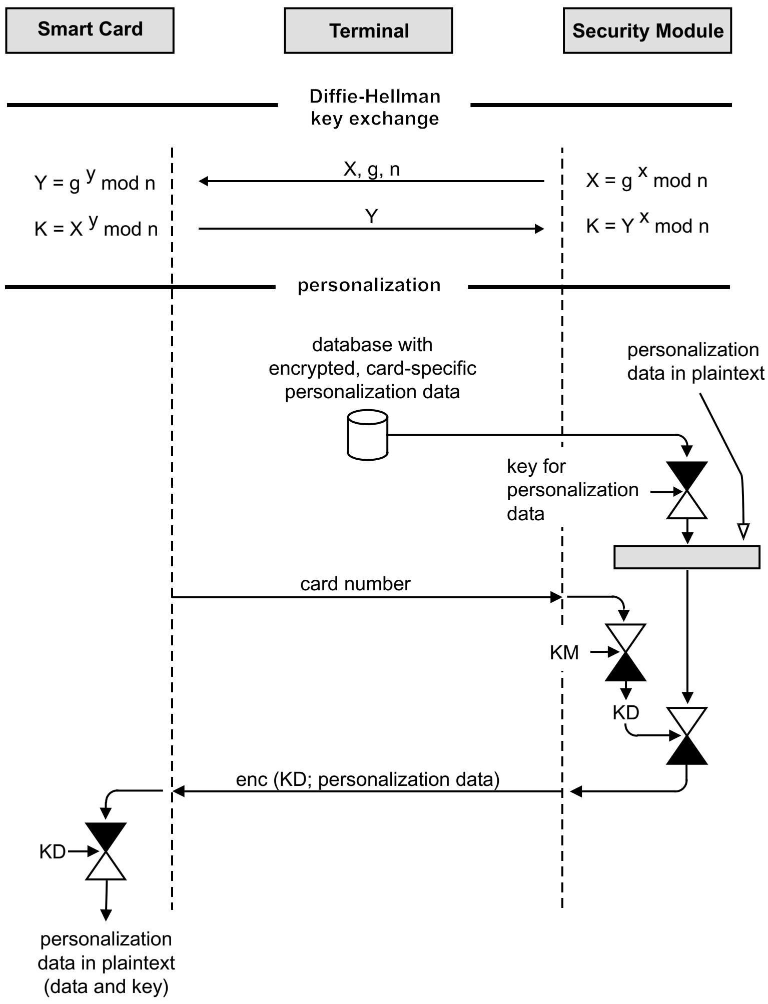

Smart Card
Terminal
Security Module
Diffie-Hellman key exchange
Y = g^y mod n
K = X^y mod n
X, g, n
Y
X = g^x mod n
K = Y^x mod n
personalization
database with encrypted, card-specific personalization data
personalization data in plaintext
key for personalization data
card number
KM
KD
enc (KD; personalization data)
KD
personalization data in plaintext (data and key)

Figure 10.64 Schematic representation of a possible procedure for personalization using cryptographically secured transmission of data and keys. In this special procedure, the keys for loading the data in encrypted form are negotiated in advance using a Diffie–Hellman key exchange. This eliminates the need to transmit a previously stored symmetric personalization key to the smart card in cleartext in a separate step. Only the cryptographically relevant processes are shown

correct sequence to the smart card and compares the responses received from the card with the responses accompanying the commands. If they do not match, the smart card is not behaving as expected and a personalization error must have occurred. With this method, it is not necessary to have a special security module for the tests in the personalization machine.

Once a smart card has been personalized, it is generally not possible to reverse the process, which means that an incorrectly personalized smart card is worthless. Of the various processes, electrical personalization is the most prone to errors, and any errors that occur in the personalization of a large batch of cards would result in major financial losses and delays. Consequently, there are a few smart card operating systems that allow the complete personalization to be fully deleted following a suitable authentication. With regard to the operating system, the smart card afterwards behaves the same as after semiconductor fabrication or completion. This capability is sometimes used for test cards, since it makes it possible to modify the software in the card instead of scrapping the card every time the software changes. Occasionally, such smart card operating system mechanisms are enabled for regular cards, thus allowing cards to be depersonalized if necessary.

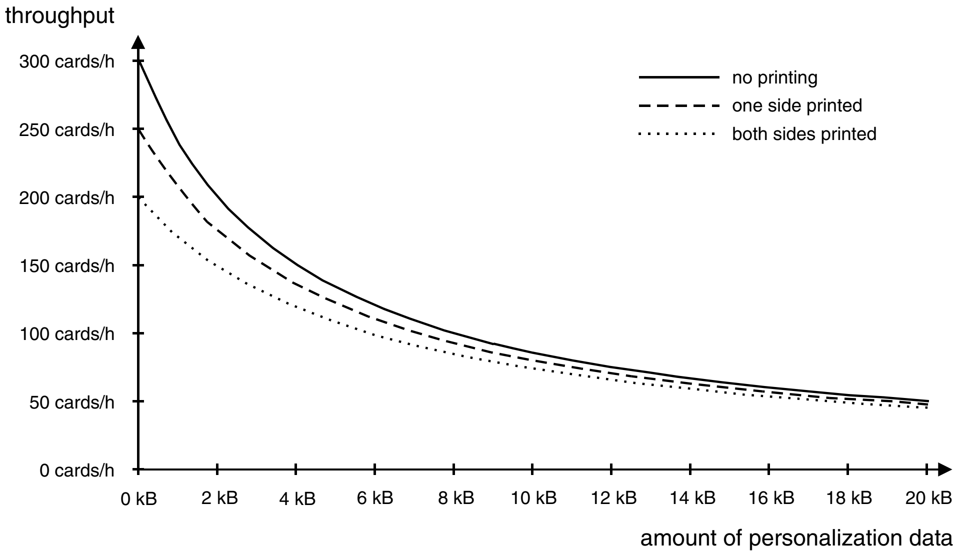


Figure 10.65 Throughput diagram for electrical personalization with single-sided and double-sided card printing using a desktop personalization machine

Generally speaking, smart card personalization is not performed for quantities less than (typically) 10,000 cards. However, many applications require the ability to reproduce individual, customer-specific smart cards. For instance, it must be possible to replace a defective or lost Eurocheque smart card within a few days, since otherwise the cardholder will no longer be able to obtain money from cash dispensers. With an increasing level of customer friendliness, there is an increasing demand for this sort of just-in-time personalization equipment. It is usually installed alongside the mass-production personalization equipment, receives card data via data telecommunications and uses smart cards that have already been initialized and held as partly-finished products. With this sort of card production, provision of a replacement card to the end user (the cardholder) within 24 hours can be guaranteed, should this be necessary. Such equipment, which is designed for fast turnaround, is naturally not suitable for the mass production of smart cards.

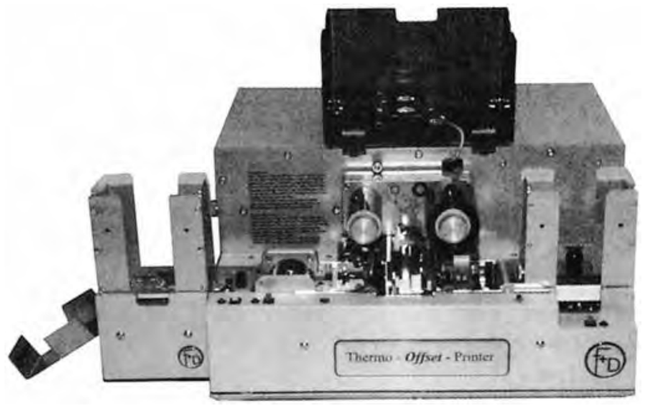


Figure 10.66 Example of a desktop personalization machine for electrical personalization and double-sided color printing with a resolution of 300 dpi. The input stack of cards is located on the right-hand side, while the stacks of good and rejected cards are located on the left (*Source:* F + D)

Envelope stuffing and shipping

The final processing step in the production of smart cards is packing and shipping the cards. This is not necessary with some types of cards, such as pre-paid phone cards, which are frequently supplied *en masse* to the card issuer. However, with more sophisticated and expensive cards it is common for the cardholder to receive a personalized letter containing his or her new card. With some applications, such as credit cards, the cardholder also receives a letter with the PIN. For reasons of security, this is sent separately and a few days later than the card. The area in which all of these activities take place is often called the lettershop.

The envelope of the PIN letter is made with a carbon-paper coating on the inside. This allows a slip of paper inside the envelope to be printed from the outside using a dot-matrix impact printer. The envelope is constructed such that an unauthorized person cannot read the printed PIN code without visibly damaging the envelope. These measures ensure that it is not possible for someone to spy out PIN codes without being noticed, even while the PIN letters are being generated. High-performance printing systems for PIN letters can print up to 34,000 documents per hour.

For posting the cards, the personal information (such as the cardholder's name and address) is either read from the card or retrieved from the production database, depending on the card type. This information is printed on a 'card carrier', which is a pre-printed letter, using a high-throughput laser printer. The letter may have two punched slots to hold the corners of the card. Alternatively, a strip of easily removable adhesive material is often used to attach the card to the letter. Following this, the card carrier is folded and inserted into an envelope. After the envelope has been franked, the smart card with the personalized letter is ready to be posted to the cardholder. High-performance envelope stuffing machines have a throughput of around 7000 letters per hour.

The final quality control step is to automatically weigh the finished letters containing the cards. The weight of the card, which is around 6 grams, is easily sufficient to ensure reliable verification that each envelope actually contains a card.

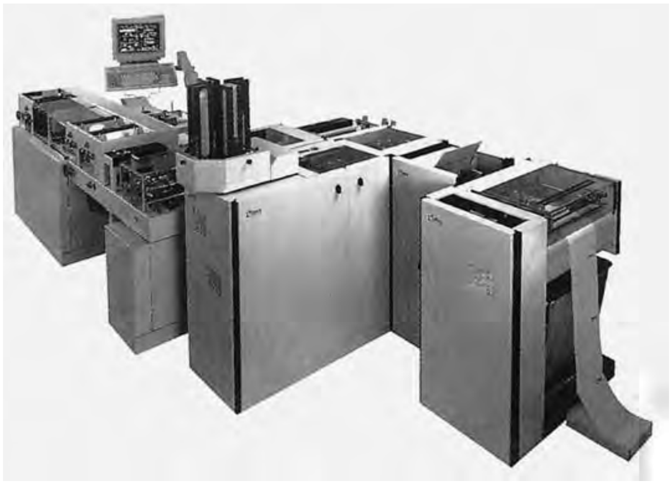


Figure 10.67 A system for attaching cards to their associated letters, which are then stuffed into envelopes along with any necessary attachments. This machine can prepare and stuff up to 7000 envelopes per hour (*Source:* Böwe Systec)

In order to minimize postage costs, it is common to presort the letters by postal code before handing them over to the post office. This optimization is most easily realized by producing the cards in the order necessary to satisfy the postal sorting criteria (such as a regional code followed by a local code).

Practical experience with even such simple things as sending cards by post repeatedly brings new and interesting problems to light. For instance, one time a major producer of smart cards was confronted with sudden failures in smart cards sent by post. When the cause of these failures was investigated, it was discovered that the responsible postal distribution center had changed the arrangement of the feed rollers in the sorting machine. With the new arrangement, the letters containing the smart cards were bent so severely during sorting that the chips inside the modules broke in some of the cards. The problem was solved by shifting the position of the card on the carrier by a few centimeters. For this and other, similar reasons, a few hundred test letters are often posted in the target region and then analyzed prior to a

Table 10.5 Summary of the relative cost factors for two types of smart cards containing microcontrollers with different memory capacities

Component or production step	Smart card with: ≈6 kB ROM ≈1 kB EEPROM ≈128 bytes RAM	Smart card with: ≈16 kB ROM ≈8 kB EEPROM ≈256 bytes RAM
Die:	50.0 %	65.0 %
Module:	25.0 %	15.0 %
Card body:	12.5 %	10.0 %
Initialization and personalization:	12.5 %	10.0 %

major mailing, in order to ensure that the smart cards will not be damaged during transport or sorting.

The production steps and phases that have been described thus far represent a mass production process, which is standard for cards such as GSM cards and credit cards with chips. Other applications or card issuers may have other basic requirements with regard to card production. For example, some GSM smart cards are personalized 'on site' in the shop and then handed directly to the customer. The customer naturally receives a favorable impression of the competence and capability of the shop if he or she can receive a personalized card immediately after subscribing and paying. However, this depends very strongly on the marketing policy and security requirements of the card issuer. In contrast to this example, producing card bodies and modules is basically independent of the ultimate card issuer or his marketing aspects, and thus largely the same for all applications.

10.5 PHASE 4 OF THE LIFE CYCLE IN DETAIL

Phase 4 of the life cycle of a smart card is well known to normal card users from daily experience with their own cards. New applications can be downloaded or activated, and applications already present in the card can be deactivated if necessary. Since the majority of this book addresses this phase, it is not described any further here, with the exception of card management systems.

Card management systems

Administrative systems for cards have been used by a variety of card issuers for many years already. However, up to now the emphasis has primarily been on inventory management and associating cards with specific persons. With the increasingly widespread use of smart cards that support modifying, downloading and deleting applications, the functions of card management systems have been fundamentally altered, since they must also deal with the aspects of card-specific applications. Such systems are called card management systems (CMS), applet management systems (AMS) or sometimes file management systems (FMS). The term 'card management system' is used here.

A functional card management system first requires a high-performance database system containing all necessary information about issued cards, as well as at least occasional online connections to the cards to be managed. For these reasons, existing smart cards used in telecommunications applications are quite suitable for use with card management systems, since they are continuously connected online to the background system while in use. In payment systems that operate partially offline, it is still possible to utilize temporary online connections to the background system, such as when a card is used with a cash dispenser or merchant terminal. An essential prerequisite for any sort of online connection is a secure end-to-end connection between the smart card and the management system.

A card management system can have a very broad range of functions. The simplest function is updating the contents of files in specific smart cards, using standard smart card commands that are sent to the cards via secure channels. A somewhat more complicated function is file management, which means deleting existing files and creating new files, using mechanisms that are similar to those used for updating file contents. All of these operations on files are referred to as ‘remote file management’ (RFM).

Significantly larger data volumes are involved in storing a new application in a smart card. If the application is file-based, all of the corresponding files must be created in the smart card and then filled with data. If the new application is program-code based, the program must be loaded into the smart card. In the case of Java Card, this is primarily done using the OP loader.[12] However, it can sometimes be necessary to replace an application by a different application or a new version of the same application. In preparation for this, the data for applications present in the smart card must be secured. Following this, the application in question must be deleted and the new application must be created in the smart card. Finally, the secured data must be loaded into the application, which may involve converting the data to a different format.

The card management systems described above relate to the period after the smart card has been issued to the end user. However, the functions of a card management system can be significantly expanded to cover the entire life cycle of the smart card. This is referred to as life-cycle management. It begins with the completion of the smart card operating system and extends over the initialization and personalization of the smart card through its actual use and any subsequent deactivation of the card that may be necessary at some time, including transferring the data to a new smart card.

Naturally, this manifold of functions causes card management systems to be quite complex. Furthermore, it should be noted that it is extremely rare for the set of smart cards being managed to be homogeneous. The most common situation is a highly heterogeneous hodge-podge of different smart card operating systems in various versions running on a variety of hardware platforms with different memory sizes. The applications to be managed will also have a certain range of versions.

As an example that illustrates the resulting complexity, we can consider the situation of an operator of a telecommunications network using SIMs having three different versions of the operating system running on three different hardware platforms with three different versions of the application. In the worst case, the card management system will have to perform 27 ($= 3^3$) different types of access to the application. The card user, by contrast, sees all of these 27 variants as only a single application in his SIM.

Besides the large number of variants that can quite easily arise, another consideration is that the smart cards to be managed must meet certain general conditions. In principle, the entire administrative process must be performed in an atomic manner by the card management system, since if it is somehow possible to prevent administration operations from being fully completed by means of some sort of interruption to the process, it must be possible to restore the original state. For example, consider downloading a Java applet into a SIM via the air interface. If the connection is broken, for instance because there is a coverage gap in a tunnel, this must not be allowed to have any sort of technical consequences for the existing functionality of the SIM. All of this can be technically achieved using existing mechanisms and procedures, but it requires substantial effort.

There are commercially available card management systems that can provide several of the previously described functions. However, if smart cards are used on a large scale in a system in

[12] See also Section 5.11, ‘Open Platform’

which it is necessary to dynamically manage applications, major extensions to certain aspects of existing card management systems will be necessary, regardless of the nature of the functions.

10.6 PHASE 5 OF THE LIFE CYCLE IN DETAIL

Phase 5 of the life cycle of smart cards according to the ISO 10202-1 standard defines all measures relating to terminating the use of the card. Specifically, these measures consist of deactivating the application(s) in the smart card, followed by deactivating the smart card itself. However, both of these processes are purely theoretical with most smart cards. In practice, cards are either thrown into the trash or carefully labeled and filed away by collectors for some indeterminate length of time. Generally speaking, it is quite rare for cards to be returned to the card issuer.

Nevertheless, there are commands that can be used to deactivate individual applications and the complete smart card. The ISO/IEC 7816-9 commands DELETE FILE, DEACTIVATE FILE, TERMINATE DF and TERMINATE CARD USAGE are explicitly intended to be used to herald the final stage of the life cycle of an application.[13]

These commands are primarily essential for managing individual applications in multiapplication cards, but they are rarely used with present-day smart cards, which mostly incorporate more or less only one application. The easiest way to end the life of a smart card is to simply cut it into pieces using a pair of scissors. Anyone can do this, and some card issuers recommend this method for ‘terminating’ smart cards.

Nevertheless, in some cases it would certainly be justified for reasons of security to return smart cards to their issuer. Some of them still contain valid secret keys, and if a potential attacker could manage to acquire several hundred or even a thousand cards, he would have a significantly larger pool of data for analyzing the hardware and software of the smart cards than if he had only a few cards. Statistical investigations based on a large number of cards will always yield more information than those based on individual cards.

For this reason, as well as well-known environmental considerations, some card issuers collect expired cards when they issue new cards. In addition, collection bins for empty telephone cards are often placed next to card phones. Effective recycling of cards is only possible after the cards have first been collected.

Recycling

We must honestly admit that little progress has been made in the recycling of smart cards. For one thing, presently there are simply not enough cards collected for a proper recycling process, and the amount of material to be recycled is anyhow not all that large. In 1997, approximately 40,000 metric tons of plastic were used in the whole world for the production of smart cards. Even under the fully idealistic assumption that an equal weight of cards could be separately collected and fed back into a recycling process, this is a vanishingly small amount compared with the total amount of plastics produced worldwide, which for PVC alone amounted to approximately 13 million metric tons in the same year.

[13] See also Section 7.8, ‘File Management Commands’

Nevertheless, this will change with the increasingly widespread use of cards. Recycling smart cards is a particularly difficult problem. The card body, which is laminated from several layers of various types of plastic, is a highly heterogeneous material. In addition, the cards are printed with several different kinds of ink and contain holograms, signature panels and magnetic stripes, all of which add to the number of different materials in the mix. Highly homogeneous materials can only be accumulated during card production, for instance as scrap resulting from punching cards from single-layer sheets. It is relatively easy to reuse these materials, and many card manufacturers already do so.

In the case of discarded smart cards, on the other hand, it is currently practically impossible to separate the cards into homogeneous sorts of material. The presently proposed recycling method is to punch the modules out of the cards and then shred the rest of the card bodies. The plastic shreddings can be used to produce low-quality plastic items (garden ornaments are a typical example of this type of recycling). The modules can also be finely ground, and the metals that they contain can be recovered using electrolytic processes. However, such methods are presently not used anywhere on a large scale. In addition, it is not entirely clear that this sort of complex recycling truly protects the environment better than simple incineration or burial.

In the case of contactless smart cards with coils of copper wire or conductive ink embedded in the card body, it is effectively impossible to separate the material of the card into individual types of plastic.

Particularly in the case of multilayer cards, the only practical approach is high-temperature incineration, which some people rather arrogantly refer to as ‘energy recycling’. If the temperature is sufficiently high, relatively few harmful materials are released. It remains to be seen whether this solution will be considered to be acceptable in the long term. In any case, even though a single smart card weighs only 6 grams, the net weight of one million such cards is still 6 metric tons.

Table 10.6 Summary of the major components of smart cards, in terms of weight

Component	Material	Weight
card body	various plastics (e.g. PVC, PC, ABS)	4.400 g
inks on the card body	resins and pigments	very low
magnetic stripe	iron oxide and similar materials, ink and adhesive	very low
hologram	aluminum and adhesive	very low
microcontroller (10 mm^2)	silicon with various doping elements	0.009 g
bonding wires	gold or aluminum	very low
encapsulation blob for the microcontroller	epoxy resin	0.010 g
adhesive to hold the module in the card body	epoxy resin	very low
module with six contacts	epoxy resin, glass fibers, nickel, aluminum, gold	0.170 g
module with eight contacts	epoxy resin, glass fibers, nickel, aluminum, gold	0.180 g

11 Smart Card Terminals

The only connection between a smart card and the outside world is the serial interface. There is no other way in which data can be exchanged, so an additional device that provides electrical connections to the card is necessary. In this book, such a device is always referred to as a terminal. However, other terms are used, such as interface device (IFD), chip-accepting device (CAD), chip-card reader (CCR), smart card reader[1] and smart card adapter. The basic functions, which are to supply power to the card and to establish a data link, are the same for all of these devices.

Any terminal that consists of more than just a contact unit, a voltage converter and a clock generator always has its own processor (usually with an 8- or 16-bit architecture) and associated memory. In simple equipment, the processor can be part of a microcontroller, but it is often a component of a single-board computer. Terminals are usually programmed only by terminal manufacturers using C, C++ or Java [JavaPOS]. In mobile telephones, which are also smart card terminals, a variant of Java (Java 2 Micro Edition, or J2ME) will attain considerable importance in the future as a programming language.

Terminals do not have their own hard disk drives, which means that they must store their programs and data in battery-backed RAM, EEPROM or Flash EEPROM. The amount of available memory is usually on the order of a few megabytes.

The problems related to allowing third parties to program terminals have been solved in the same manner as for smart cards by using executable program code, so here the solutions will most likely lead to the same sorts of developments. The Europay Open Terminal Architecture (OTA), with a Forth interpreter, was one of the first attempts at a solution in 1996, and Java for terminals is the next step. The EMV specification also explicitly includes a concept for downloadable program code.

In contrast to smart cards, which all have very similar constructions, terminals are built in many different ways. A fundamental distinction can be made between portable and stationary terminals. Portable terminals are battery-powered, while fixed terminals are preferably powered from the mains network or the data interface. Terminals can also be classified by their user

[1] The terms ‘card reader’ and ‘smart card reader’ should not be understood to mean that data can only be read from the card using such devices. Write accesses are naturally also possible

Smart Card Handbook, Third Edition. W. Rankl and W. Effing
© 2004 John Wiley & Sons, Ltd ISBN: 0-470-85668-8

interfaces. Portable devices in particular may have displays and simple keypads to allow their most important functions to be used on site. Although fixed terminals also often have displays and keypads, they have permanent links to higher-level computer systems as well. A terminal lacking a man–machine interface (i.e., display and keypad) must have a direct connection to a computer in order to provide a link between the smart card and the user.

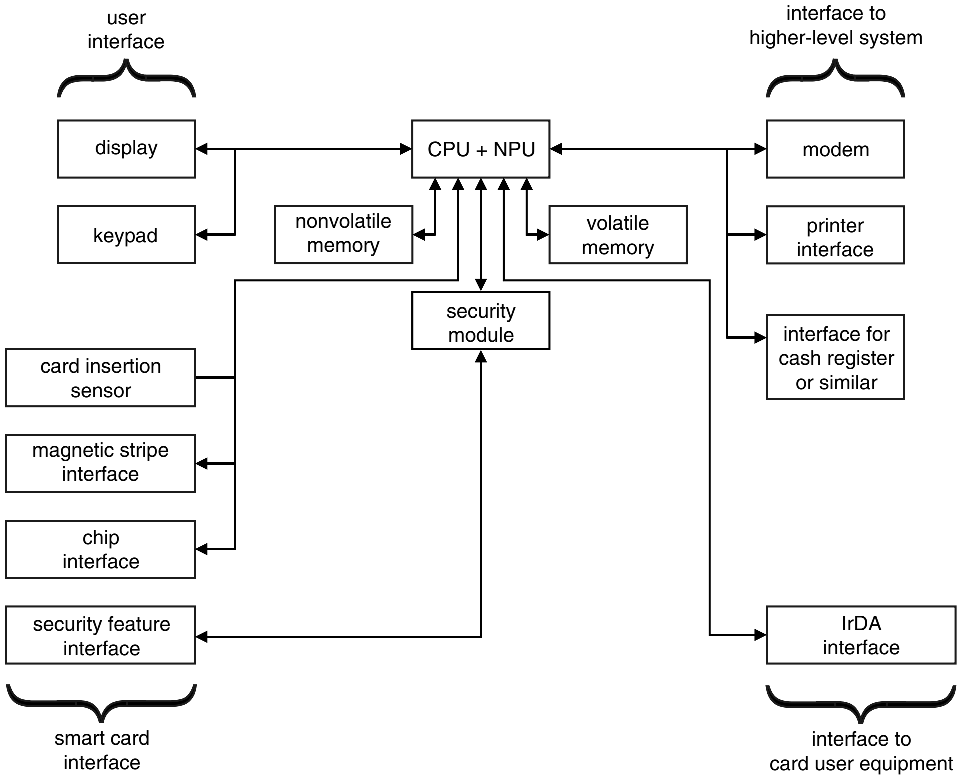


Figure 11.1 Typical architecture of a smart card terminal with a display, keypad, magnetic-stripe reader and security module. Such terminals are often used at point-of-sale locations to allow payments to be made using a wide variety of cards (credit cards, debit cards and electronic purses). A keypad that is specially protected against manipulation (a PIN pad) can be used if necessary. This diagram shows the basic energy and data flows and is not a schematic diagram

There is a general and very practical characterization of classes of terminals in one of the specifications of the German ZKA, which divides terminals into four classes. A Class 1 terminal is one that essentially consists of a contact unit without any supplementary functional elements, along with an interface to another system (e.g., USB). Class 2 includes all of the capabilities of Class 1, with the addition of a keypad. A Class 2 terminal need not have its own keypad if it is connected between a contact unit and a PC. A Class 3 terminal has a display, in addition to the elements of Class 2. Class 4, which is the most elaborate, has all of the functional elements of Class 3 as well as a hardware security module (HSM) with RSA capability.

Table 11.1 Classification of terminals according to the ZKA

Class	Functional elements
1	Contact unit and interface to other systems
2	Class-1 functional elements + keypad
3	Class-2 functional elements + display
4	Class-3 functional elements + security module

There are also a few terminals equipped with Infrared Data Association (IrDA) or Bluetooth interfaces. Such terminals can be used for direct communication between the terminal and a personal digital assistant (PDA) or a mobile telephone. The advantage of this is that the user, who can assume that his or her own device is trustworthy, does not have to enter data (such as a PIN) using a 'foreign' terminal.

The division into portable and fixed terminals leads to a further distinguishing feature, which is how the terminal is used. An *online* terminal has an uninterrupted connection to a remote computer during operation, and this computer assumes part of the control function. A typical example is a terminal used for physical access control, which is completely controlled by a background system to which it is permanently connected.

The opposite type of terminal is an *offline* terminal. Such a terminal works completely independently of any higher-level system. However, although there are very many types of online terminals, there are practically no 'pure' offline terminals. All offline terminals occasionally exchange data with a background system, if only to request a new blacklist or an updated version of the terminal software.

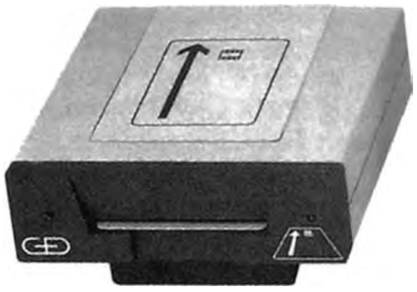


Figure 11.2 A typical smart card terminal for connection to a computer via a serial interface (Giesecke & Devrient model CCR2)

In typical applications within a building, the physical link between the terminal and the remote computer is either an electrical cable or a fiber-optic cable. However, the link can also be formed by a telephone connection to the nearest computer center, as is the case with point-of-sale terminals for electronic payments. This may involve a dial-up link or a permanent link (leased line), depending on the application. Since leased lines are expensive, there is an increasing tendency to use the telephone line only as necessary, in order to reduce operating costs. This means that the terminal must be equipped with a dial-up modem.

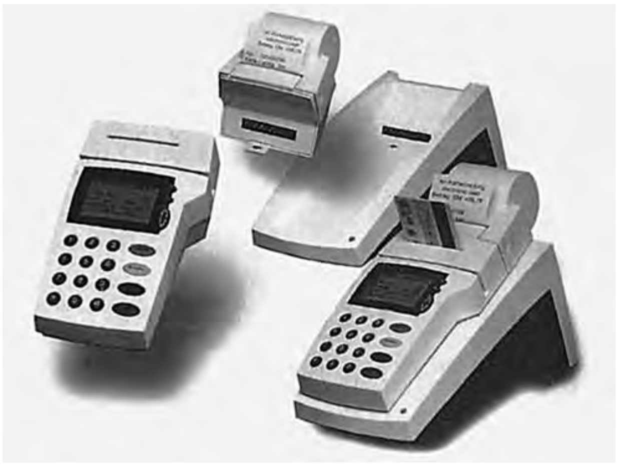


Figure 11.3 Example of a portable smart card terminal for electronic payments using credit cards, debit cards and electronic purses (Giesecke & Devrient model ZVT 900). This terminal has an integrated security module and a printer, and it can be used offline.

Smart card terminals in the form of PC cards (formerly called PCMCIA cards) do not readily fit into the above classification scheme. They can be used both online and offline, and with both desktop and portable computers. In principle, such terminals are just simple and usually inexpensive hardware interfaces between a smart card and a computer. The only prerequisite for using a PC-card terminal is a PC card slot, which must be either a type I slot (3.3 mm high) or type II slot (5 mm high), depending on the manufacturer. Some PC-card smart card terminals contain expansion memory for the smart card and coprocessor ICs for mass data encryption and decryption, in addition to the smart card interface. These terminals, which are only a few millimeters thick, are certainly the most versatile of all. They open up application areas for smart cards that in some cases are totally new. With such terminals, it is now possible for smart cards to work together with standard PCs and standard software without additional cables, power supplies or external hardware. The spectrum of possible applications is very wide. It includes access protection for specific PC functions, software copy protection and e-mail transfers protected by digital signatures.

'Diskette terminals' are also available. They provide a simple means to exchange data between a smart card and a PC. Such a terminal has the form of a 3.5-inch diskette and contains a very thin contact unit, card-activation electronics, a battery and a coil for transferring data to and from the read/write head of the diskette drive. There is enough room in a 3.3-mm thick diskette terminal to insert a smart card. On the PC side, all that is needed is a suitable software driver to handle data exchange. This is one way to integrate smart cards into existing systems in an uncomplicated and economical manner, although in practice this solution has not achieved widespread acceptance.

Many years of R&D activity lie between the earliest two-chip smart cards and the modern-day versions, which are equipped with very powerful microcontrollers. Terminals have undergone a similar technical evolution over the same period. The first terminals often had very primitive mechanical and electrical constructions, partly due to lack of experience. The consequence of this was that smart card microcontrollers were frequently damaged and thus failed

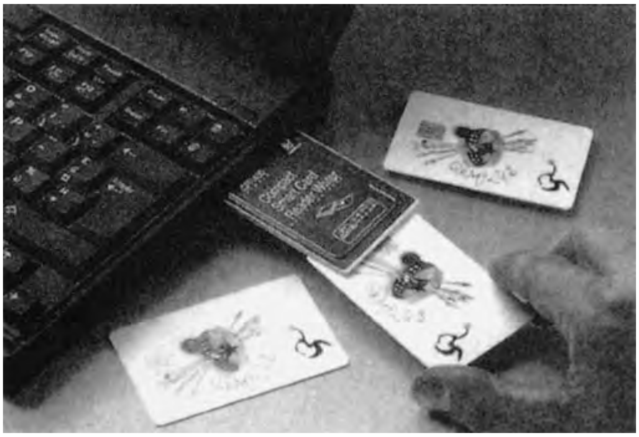


Figure 11.4 A typical smart card terminal in PC-card format (Gemplus model GPR400)

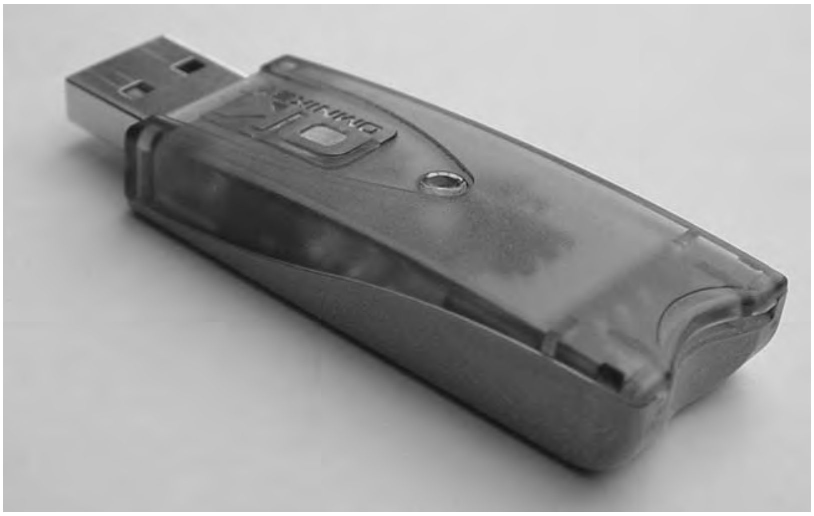


Figure 11.5 A smart card terminal in the form of a USB plug, for use with cards in the ID-000 (plug-in) format

prematurely. Since then, most terminal manufacturers have overcome these ‘teething troubles’, and a development stage has been reached in which external design is a more important factor in the buyer’s choice of terminal than technical features and specifications, which are generally similar for all terminals and manufacturers.

In functional terms, a smart card terminal consists of two parts: a contact unit for the card and a terminal computer. The card reader, into which the smart card is inserted so that it can be electrically contacted, essentially has only a mechanical function. The terminal computer is needed to electrically drive the contacting unit, manage the user interface and establish a link to a higher-level system. In the simplest case, it can be a single microcontroller, while in technically more sophisticated solutions, it is a single-board computer.

11.1 MECHANICAL PROPERTIES

When a smart card is inserted into a terminal, two things happen in a mechanical sense. First, the card's contacts must be electrically connected to the terminal computer. This is the task of the contact unit. Second, the terminal must detect the fact that a card has been inserted. This can be handled by a microswitch or an optical sensor (light barrier). One drawback of the latter is that its reliability can be affected by dirt or cards with transparent bodies. A mechanical switch is generally the most effective solution.

Terminals differ very greatly in terms of the contact units and contacts that are used. The GSM 11.11 specification imposes certain limits on the insertion force and the shape of the contacts, and almost all terminals use these values. According to this specification, the tips of the contact elements in the terminal should be rounded rather than pointed, with a radius of curvature of at least 0.8 mm. This largely prevents scratching the contact surfaces of the card. In addition, the force required to insert the card into the contact unit is significantly lower if the contact elements have rounded leading edges than if they are pointed.

According to the GSM specification, the maximum force exerted on a single contact must not exceed 0.5 N under any circumstances (the EMV specification allows 0.6 N). This is intended to protect the chip located beneath the contacts, since this piece of silicon crystal could break under greater stress.

Although the location of the contacts on the card is internationally standardized by ISO and should thus be the same everywhere, a French national standard (AFNOR) has the chip nearer the top edge of the card. Consequently, there are terminals that have two contact heads. This allows both ISO and AFNOR contact locations to be supported. This technically complicated solution is of interest in systems in which smart cards with ISO and AFNOR contact positions are used together. This is only a transitional situation, since ISO specifies that the AFNOR location should no longer be used. Several French banking applications, for example, employ terminals with dual contact heads. This allows both the old AFNOR cards and the newer ISO cards to be used during the transition period.

Problems can occur with the electrical contacts between the terminal and the smart card, especially with portable terminals and terminals installed in vehicles. Such terminals, in particular those in vehicles, are often subjected to high accelerations, which can cause the contacts to briefly separate from the card's contact surfaces. You can imagine that a vehicle traveling over cobblestones at a certain speed can cause the spring-loaded contacts to oscillate at their resonant frequency. If the card is electrically activated at the time, it is simply impossible to predict what will happen.

In the extreme case, when all contacts simultaneously lift free and then reconnect with the card, the card would probably execute an activation sequence and then send an ATR. However, in this situation it is certain that the electrical activation sequence will not comply with the ISO standard, which means that this can eventually lead to chip failure if it is frequently repeated. In any case, this brief power interruption will naturally result in the loss of all states that have been achieved in the card during the current session. Depending on the application, it may thus be necessary to enter the PIN again or re-authenticate the user.

If only one contact lifts free, the consequences strongly depend on which contact it is. If it is the I/O contact, the only consequence is a temporary disturbance to the communications link. This disturbance can be handled using standard error recovery mechanisms. If a different contact lifts free, the card will be reset. In this case, the communications link must be re-established from the very beginning.

In order to prevent the contacts from lifting free due to acceleration forces, the contact force can be increased, but the upper limit is still 0.5 N per contact. There is no simple satisfactory technical solution to this problem, but the probability of contact separation can be minimized by sensible placement of the terminal. For example, the terminal can be mounted so that the contacts are perpendicular to the main axis of acceleration.

In any case, the terminal software must be able to independently re-establish communications if the contacts have briefly lifted free of the card. The millions of GSM telephones in daily use demonstrate that smart cards can be used in portable equipment without any problems.

The service life of the contacts and the technical construction of the terminals vary immensely. The service life is also strongly affected by environmental conditions, such as temperature, humidity and the like. An MTBF (mean time between failures) of 150,000 insertion cycles, however, is considered to be a normal value for a terminal.

Contact units with wiping contacts

The technically simplest terminals, which are thus the least expensive, have only wiping contacts in the form of leaf or disc springs. No other mechanical contact elements are present in these simple terminals. However, with such a simple spring-based contact unit, the contact surfaces and part of the card are always dragged across the contacts when the card is inserted and withdrawn, which produces scratch marks. These are undesirable for both aesthetic and technical reasons.

Repeated scratching of the gold-plated contact surfaces of the card gradually wears away the protective gold layer, and the exposed metal underneath this plating will then oxidize. This adversely affects the electrical connection. The user may have to insert and remove the card several times in order to rub off the oxide layer so that a satisfactory electrical connection can be made.

Mechanically driven contact units

The next higher class of terminals does not have fixed sliding contacts, but instead a mechanism that presses the contact unit against the contact surfaces of the card when the card is inserted in the terminal. A lever mechanism converts the force used to insert the card into a force perpendicular to the contact surfaces.

An optimally designed mechanism also produces a very small amount of movement of the contact unit along the length of the card while the contacts are being applied to the card. This ensures reliable electrical contact with the card, since the sliding motion rubs away any light soiling on the contact surfaces. The contact pins are also individually spring-loaded, in order to ensure a well-defined contact pressure for each contact surface.

Electrically driven contact units

The technically most complex solution, which is also the best mechanical solution, is a terminal with an electrically driven contact unit. Here a set of parallel contact pins is driven by a motor or solenoid to make perpendicular contact with the card from above, with a slight lateral motion.

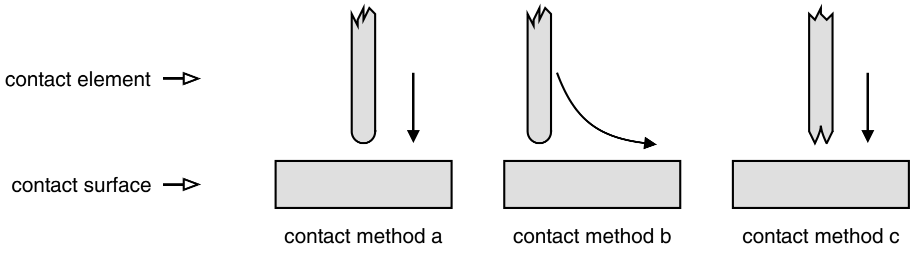


Figure 11.6 Methods for making electrical contact with smart cards. Method (a) with rounded contact pins is unfavorable, since soiling of the contact surface will adversely affect the reliability of the electrical contact. Methods (b) and (c) represent good solutions for the two types of contact pins illustrated. The sharp-edged contact pins shown in (c) slightly penetrate the contact surface, which can be seen under a microscope as small surface nicks

Due to the complexity of this electromechanical construction, the terminal is relatively large. However, this type of terminal is quite suitable for use in professional applications, in which many millions of contact cycles must be made without maintenance. It is therefore typically used in automated teller machines (ATMs) and personalization machines employed in smart card manufacturing.

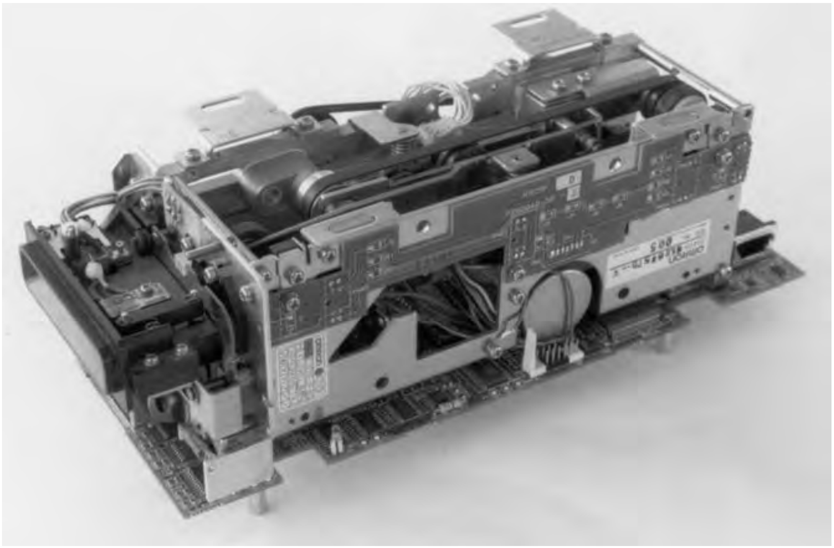


Figure 11.7 A typical self-feeding reader for cash dispensers, with a shutter and magnetic-stripe reader

Card ejection

The smart card is normally inserted manually, which means without any assistance from the terminal. Only ATMs have self-feeding card readers, which use a conveyor mechanism to feed the card to the contact unit within the machine. Ordinary terminals do not have such mechanisms, and they differ only in the manner in which the card is ejected. Simple terminals do not automatically eject the card, which means that the card must be manually removed from the reader. Two different techniques are used for this, called 'push–push' and 'push–pull'. With a push–push contact unit, the card is inserted by hand as usual, and it must removed by again pushing it and then pulling it. This is not ergonomically desirable, since this sequence of motions is unnatural, so people often forcibly pull the card out of the terminal. This causes the contact pins to be scraped over the contact surfaces and the body of the card, since the contacts have not yet been released by the mechanism. Push–pull contact units better match normal motion sequences, since the card is simply pushed into the terminal to insert it and pulled out of the terminal to remove it.

Terminals that automatically eject the card have a spring that is tensioned by inserting the card. This can be released by the terminal computer via a solenoid. This causes the card to be partially extended from the terminal, rather than fully ejected, so that the user can grasp it and pull it out completely.

Card-ejecting readers have one major advantage relative to other types. Ejection of the card very clearly signals the end of the session to the user, while also reminding the user not to forget the card in the terminal. This reminder is often emphasized by an audible beep. This practical argument is the main reason for using card-ejecting readers.

Cash dispensers in particular are usually able to retain smart cards if necessary. Since they routinely have self-feeding card readers, it is naturally technically feasible to route the card to a special retention bin in the machine if necessary, rather than to the exit slot. From a technical viewpoint, retaining cards presents no major problems, as long as the terminal is large enough to hold the extra mechanism and the retention bin. In certain circumstances, however, there can be legal problems if the card user is also the legal owner of the card.

Ease of card withdrawal

The reliability of a system based on smart cards can suffer severely if users can withdraw their cards from the terminal at any time during a session. For one thing, this causes the card to be disconnected from the power supply without following the prescribed deactivation sequence. It could also interrupt EEPROM read or write operations, causing the content of a file to be undefined. This could cause the card to fail completely. For these reasons, it is advantageous to use terminals with card-ejecting readers that are designed such that it is impossible to manually pull the card out of the terminal. A hidden mechanical emergency ejector can be provided to remove a smart card from the terminal in case of a power failure. However, under normal circumstances the terminal can determine when to return the card to the user, thus preventing the user from interfering with ongoing processes.

11.2 ELECTRICAL PROPERTIES

With the exception of the contact unit, a terminal primarily consists of electronic components. These are used to provide the interfaces to the user and the background system, and to electrically drive the contacts. The terminal's electromechanical parts and the smart card itself must be supplied with electrical signals. The only information that is directly provided by the contact unit is whether a card has been inserted. The only signal that is sent directly to the contact unit is the signal to actuate the automatic card ejector, if such a device is present.

The card interface consists of the five contacts for the ground, supply voltage, clock, reset and data signals. Once the electrical connections have been made, it is very important with regard to the service life of the card for the activation sequence specified by ISO/IEC 7816-3 to be followed exactly. Otherwise, the chip may be electrically overstressed, which will increase the failure rate. It is also important to observe the proper deactivation sequence, since otherwise the same problems may occur.

In this regard, there is an important consideration with simple terminals that allow the user to remove the card manually. Whenever the contact unit detects that the card is being withdrawn, the terminal's electronic circuitry must immediately execute a deactivation sequence. This is the only way to prevent the contacts from sliding across the contact field of the card while they are possibly still energized, which would produce results that have little in common with a standard deactivation sequence. However, the consequences of such an unallowed card withdrawal can be even more serious, since shorts may occur between the leads if the contacts are worn or slightly bent. The mild sparking due to the discharge of capacitors in such a situation will damage both the contact elements and the contact surfaces of the card.

With regard to the electric circuitry, almost all terminal manufacturers have realized by now that short-circuit protection is indispensable. If this point is neglected, a single smart card with shorted contact surfaces can cause the electrical demise of very many terminals. Incidentally, shorted cards crop up regularly, partly due to vandalism and partly due to technical defects.

Short-circuit protection should extend to the point that every contact can be connected to any other contact or group of contacts without any repercussions. Ideally, the circuitry that drives the smart card should be fully electrically isolated from the remaining circuitry of the terminal. This is standard practice in public card phones in Germany, since it also largely protects the equipment against externally applied voltages as well as shorts.

The voltage needed for writing and erasing EEPROM pages is generated by the microcontroller via a charge pump on the chip. This can draw currents of up to 100 mA for intervals of a few nanoseconds. The same effect, in a reduced form, can be produced by transistor switching processes in the CMOS integrated circuits. Even very fast regulator circuits in the power supply cannot handle these short spikes, with the consequence that the supply voltage for the card collapses due to the heavy current load and the EEPROM write or erase cycle fails. In extreme cases, the voltage dropout can be so severe that the processor lands outside of its stable operating area and a system crash occurs.

The remedy is to connect a capacitor as close as possible to the contacts for the smart card. A ceramic capacitor of about 100 nF is suitable, as it can release its charge very quickly. The leads to the smart card must be as short as possible, so that lead resistance and inductance do not significantly affect the ability of the circuit to meet the increased current demand within the necessary interval. A brief increase in current demand can be met by drawing charge from the capacitor until the voltage regulator can respond to the change. This is a simple and economical way to avoid power supply problems.

Particularly for electronic payment systems, it is nowadays standard to equip the terminal with a real-time clock. This is required for reasons of traceability and user protection. According to the EMV specification for credit card terminals, the clock may not be off by more than 1 minute per month. This is not technically difficult, since suitably accurate clock components are available as single-chip solutions. In addition, the clock can be adjusted every time the terminal makes an online connection to the higher-level system. Radio time signal receivers in terminals have so far not achieved any practical importance, since signal reception is too strongly affected by the screening effects of the site where the terminal is installed. Standard time code signals usually cannot be received inside a reinforced concrete structure, for example.

11.3 SECURITY TECHNOLOGY

Terminals may contain a very large variety of security mechanisms. The spectrum ranges from mechanically protected enclosures to security modules and sensors for the various card features. In pure online terminals, whose only function is to convert the electrical signals that pass between the background computer system and the smart card, there is normally no need for additional built-in security technology. In such cases, security is handled entirely by the computer that controls the terminal.

However, as soon as data must be entered into the terminal or the terminal must operate independently of the higher-level system, it is necessary to incorporate suitable mechanisms to provide additional system security. The possibilities are almost unlimited, but they depend very strongly on the smart card in question and its security features.

With a typical smart card, whose body is very simple and only serves as a carrier for the microcontroller, there are usually no security features on the card body. There is thus no need for the terminal to check such features. In contrast, smart cards for financial transactions are usually hybrid cards, which means that they have a magnetic stripe in addition to a chip, in order to maintain compatibility with older systems. However, hybrid cards also possess the usual features that enable the terminal to check their genuineness independently of the chip. Suitable sensors must therefore be present in the terminal.

Terminals that work offline, either completely or occasionally, must contain master keys for the cryptographic algorithms that are used, since card-specific keys cannot be derived without these keys. These master keys are very sensitive with regard to security, since the entire security of the system is based on them. In order to guarantee their security and confidentiality at all times, they are not stored in the normal electronic circuitry of the terminal, but in a separate security module within the terminal that has special mechanical and electrical protection.

This security module can for example be a single-board computer encapsulated in epoxy resin, which can exchange data with the actual terminal computer only via an interface. The secret master keys are never allowed to leave the security module, but are used only internally to perform computations. In a typical application example, the security module receives an individual card number or chip number from the smart card via the terminal computer, and it uses this number to derive a card-specific key. This key is then used within the security module to compute a signature or perform authentication.

Modern versions of this module, which is normally the size of a matchbox, contain extensive sensor systems for detecting attacks. They are also largely self-contained electrically, so they can actively resist attacks, even if denied an external source of power. If an attack is detected, the usual defense is to erase all keys, so that an attacker is left with only a circuit board circuit encased in epoxy resin inside a metal case, with no data worth analyzing.

Due to the high cost of good security modules, the trend in recent years is to use smart cards instead. Although this leads to certain restrictions in terms of memory size, sensors and self-reliance, the level of security is generally adequate, even for electronic payment applications. Cards in the IC-000 format (plug-in) are used to limit the physical size.

Since security modules in smart-card format are not permanently built into terminals, but can be exchanged, they are ideally suited to extending terminal hardware, as illustrated by the following example. Static unilateral RSA authentication will become increasingly important in the next few years, partly because it is prescribed in the international EMV specification for credit cards with chips. Since RSA authentication is so computer-intensive that it cannot be performed by the processors normally used in terminals within an acceptable length of time, permanent built-in security modules represent a problem. However, if a plug-in smart card is used as a security module in the terminal, it can easily be exchanged. Relatively expensive smart cards containing supplementary arithmetic coprocessors can then be used for the security modules, which can perform RSA computations at high speed once the terminal software has been suitably modified.

In the future, a variety of card issuers will market debit and credit cards containing chips. All of these cards will use different keys and different methods for key derivation and authentication. Furthermore, it is unlikely that all card issuers will be willing to reveal secret data and methods to manufacturers of security modules. In all probability, the approach that will be taken is for a card issuer or group of card issuers to issue a common ‘terminal card’ that can perform all of the processes relevant to the security of their collective systems and can execute these processes within the terminal. This card will be accessed using one of the two standard transmission protocols ($T = 0$ or $T = 1$), and it will largely behave just like a standard smart card. The only difference will be that the terminal card will contain functions related to secret master keys, key derivation procedures and collecting security-related data (such as sales balances). The terminal will only look after the user interface and uploading or downloading data to or from the background system. All security-related functions will be handled by the terminal card. This means that the terminal must be able to work with several different terminal cards, rather than only one. A particular card will be automatically selected according to the card issuer and the selected function. The demand for several independent terminal cards has been taken into account in the latest terminals. Some of them have up to four contact units for plug-in cards. They can thus use terminal cards from several different card issuers in parallel, without mutual interference.

One of the commonly used security measures, besides providing mechanical protection for the terminal by using a robust housing that can only be opened using special tools and incorporating a security module in the terminal, is to provide mechanical protection against unauthorized tapping of data transmissions to and from the smart card. This consists of a sort of guillotine arrangement that cuts through any wires that may run from the card to the exterior of the card reader after the card has been inserted. The purpose of this device, which is called a shutter, is to prevent tapping or manipulation of the messages sent between the card and the terminal. It can be actuated either electrically or simply by inserting the card. If the wires cannot be cut, due to their thickness or composition, the shutter will not close completely. This is detected by the terminal electronics, and no power is applied to the card, so no communication takes place.

Communication between the terminal and the smart card must fundamentally be designed such that tapping or manipulation cannot impair the security of the system. Shutters should thus not actually be necessary. Nevertheless, security can certainly be increased somewhat if things are made more difficult for a would-be attacker. It makes a big difference whether an

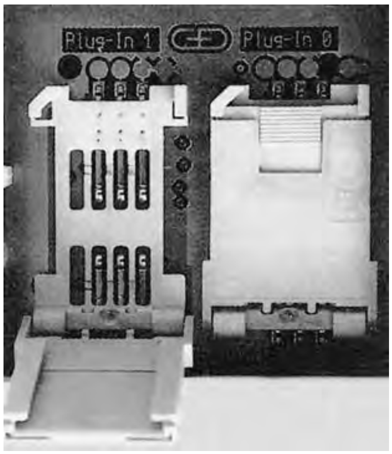


Figure 11.8 Example of two contact units for plug-in format security modules, located side by side in a smart-card terminal

attacker can readily tap the data exchange or first has to overcome a few hurdles. However, shutters make terminals bigger and more expensive, and very few of them still close precisely after several thousand operating cycles. The system design should therefore not rely entirely on this sort of mechanical protection.

11.4 CONNECTING TERMINALS TO HIGHER-LEVEL SYSTEMS

For smart cards to be used in a PC environment, it is necessary to have a terminal that is connected to the PC and to have support from the PC software. The difficulty here is naturally that in the past, each type of terminal required its own software driver to be installed in the PC. Each driver in turn had its own software interfaces, so in practice it was not possible to generate terminal-independent software. In the mid-1990s, work began on developing specifications for terminal-independent integration of smart cards into PC programs. This occurred in various countries and was performed by a wide variety of organizations. Internationally, two industrial standards have come to prevail: Personal Computer / Smart Card (PC/SC) and Open Card Framework (OCF). In Germany, as well as other countries, the *Multifunktionales Kartenterminal* (MKT) specification has been in place for some time. It has achieved surprisingly widespread used within the German-speaking realm. All three of these specifications are described in summary form below, and they can also be obtained free of charge via the Internet.

11.4.1 PC/SC

The first efforts to generate an international specification for linking cards with PC began in May 1996. The companies Bull, Hewlett-Packard, Microsoft, Schlumberger, Siemens

Nixdorf, Gemplus, IBM, Sun, Verifone and Toshiba participated in the development of this specification.

Version 1.0 of the 'Interoperability Specification for ICCs and Personal Computer Systems' was published in December 1997. It consists of eight parts, which are described in Table 11.2. The working group was known as PC/SC (for 'personal computer / smart card'), and this abbreviation is also used to refer to the specification. It can be obtained via the Internet from the WWW server of the specification group [PC/SC].

Table 11.2 Summary of the eight parts of the PC/SC specification

PC/SC specification	Content
Part 1: Introduction and Architecture Overview	This is the basis for all other parts of the specification. It identifies the relevant standards, summarizes the system architecture and the hardware and software components, and lists definitions and acronyms.
Part 2: Interface Requirements for Compatible IC Cards and Readers	This defines the physical characteristics of contact-type smart cards. It specifies basic electrical properties, such as the power supply and the reset behavior, and defines the data elements, structures and allowed processes of the ATR and PTS. There is a summary of the basic aspects of data transfers at the physical level, and the T = 0 and T = 1 protocols are both described.
Part 3: Requirements for PC-Connected Interface Devices	The requirements imposed on the terminal and the supported terminal features (display, keypad and so on).
Part 4: IFD Design Considerations and Reference Design Information	Information for designing terminals, with reference to PS/2 keyboard interfaces and USB interfaces.
Part 5: ICC Resource Manager Definition	Detailed descriptions of the technical aspects of the ICC Resource Manager, including the associated classes.
Part 6: ICC Service Provider Interface Definition	Detailed descriptions of the technical software aspects of the ICC Service Provider and Crypto Service Provider, including the associated classes.
Part 7: Application Domain and Developer Design Considerations	Description of the utilization of the PC/SC specification from the application perspective.
Part 8: Recommendations for ICC Security and Privacy Devices	Compilation and definition of recommended functions and mechanisms that should be supported by a PC/SC Smart Card. This includes the file system (MF, DF and EF), associated file access conditions, necessary system files in the smart card (for keys, PINS and so on), commands, return codes and cryptographic algorithms.

At least in principle, PC/SC is platform-independent, since it works on all Windows-based PCs, and these make up the majority of personal computers. It allows smart cards to be integrated into any desired application in a manner that is largely independent of programming language, since it supports widely used languages such as C, C++, Java and Basic. The only prerequisites are that a suitable driver must be available for the terminal to be used and the smart card must be PC/SC-compatible. However, this compatibility requirement is reasonably non-critical, since the scope has been kept relatively broad.

The easiest way to gain an overall understanding of the PS/SC specification is to view it in terms of the defined hardware and software components. The following seven components are described in terms of their functions and mutual interfaces:

- ICC-aware application
- ICC service provider
- Crypto service provider
- ICC resource manager
- IFD handler
- IFD
- ICC

The tasks and functions of each of these components are briefly described below, in the order in which they are listed above.

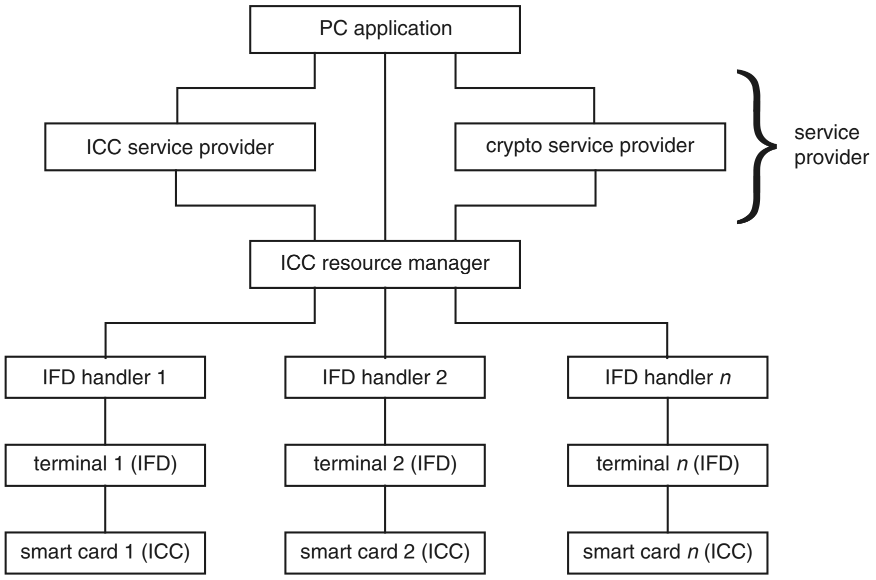


Figure 11.9 Overview of the software architecture of the PC/SC specification for linking smart cards to PC operating systems

ICC-aware application

This is an application that runs on a PC and that wishes to use the functions and data of one or more smart cards. It can also be an application that runs under a multiuser operating system with multitasking and multithreading.

Service provider

The function of the service provider is to encapsulate the individual functions of a smart card, independent of whatever operating system is used in the smart card. For example, it is possible to select a file via the API of the service provider without knowing the smart card command used for this purpose or having any idea of the coding of this command.

The service provider component is split into the ICC service provider and the crypto service provider. This is done to avoid problems with export restrictions that many countries have with regard to cryptographic algorithms. The separate crypto service provider, which handles all functions that can cause export problems, can be omitted. In this case, the PC/SC interface can be used for all functions except cryptographic functions.

The service provider does not have to be a single piece of software. It can also consist of multiple software components linked by a network. For example, it is possible to locate the crypto service provider on a cryptographically secure or high-performance computer that is isolated from the remainder of the PC/SC components.

ICC resource manager

The ICC resource manager is the most important component of the PC/SC architecture. It manages all resources that are necessary to integrate smart cards into the operating system. It must provide three important functions.

First, it is responsible for recognizing connected terminals and smart cards. It must also recognize when a smart card has been inserted or removed from a terminal, and respond to such events by providing suitable messages.

Its second function is to manage the allocation of terminals to one or more applications. For this purpose, a terminal resource can be exclusively assigned to a particular application. However, if several applications access the same terminal simultaneously, this terminal must be identified and managed by the ICC resource manager as a shared resource.

The third function is to provide transaction primitives. A transaction primitive is formed by binding the commands related to a particular function into a group. This ensures that these commands will be executed in an uninterrupted sequence. Otherwise, it would be possible for two uncoordinated applications to concurrently access a smart card, each using its own sequence of commands. The problems that this would cause can most easily be illustrated by the following example. In a smart card, only one file can be selected at a time. If two different applications attempt to select different files at the same time using SELECT FILE commands and then read data from the smart card using read commands (such as READ BINARY), it is completely undefined which file will actually be read. This depends only on the order in which the commands arrive at the smart card. A much more complicated situation, but no less tricky, arises when it is necessary to perform complex procedures involving several applications interacting with a single smart card (such as paying using an electronic purse). The ICC resource manager ensures that command sequences that belong together cannot be split up or interrupted by other commands, and so ensures that the individual procedures are executed one after the other.

IFD handler

The IFD handler is a sort of driver that is specific to a particular terminal. Its tasks are to link the terminal to the specified interface of the PC and to map the individual characteristics of the terminal onto the PC/SC interface. In a manner of speaking, the IFD handler represents a data channel from the PC to a particular terminal.

IFD (interface device)

The IFD component of the PC/SC specification is a terminal connected to the PC via an interface. The interface is arbitrary, so the terminal can for example be connected to the computer via an RS232 interface, a universal serial bus (USB) interface or a PC-card interface. The terminal must meet the ISO/IEC 7816-1/2/3 standards, which among other things means that it must support both of the asynchronous data transmission protocols (T = 0 and T = 1). Optionally, it may support synchronous transmission protocols (2-wire, 3-wire and I^2C bus) for memory cards, as specified by the ISO/IEC 7816-10 standard. In the terminal, in addition to a display, the PC/SC specification supports a numeric keyboard, a fingerprint scanner and other biometric sensors for user identification.

ICC (integrated chip card)

Microprocessor smart cards that are compatible with the ISO/IEC 7816-1/2/3 standards are required to be supported by the PC/SC specification. Memory cards that comply with the ISO/IEC 7816-10 standard may also be used, if this is allowed by the terminal.

11.4.2 OCF

The Open Card Initiative [OCF] was founded in 1997 by a group of more than 10 companies active in the smart card and PC areas. The objective was to create a smart card interface on PCs that was independent of the operating system of the PC (Windows, Unix etc.) and independent of whatever application might be present in the smart card. The result is Open Card Framework (OCF), a Java-based interface on PCs that can be used to allow applications running on PCs to access applications in smart cards. OCF has become an industry standard in the Java environment.

11.4.3 MKT

In Germany, work on generating a specification for linking smart cards to PCs via software began at a relatively early date. This led to the *Multifunktionales Kartenterminal* (MKT) specification, which has been published in various versions by Teletrust Deutschland since 1994. It is primarily oriented toward the interests of the health care field, but it is now used as a basis for many other types of terminals within Germany.

The MKT specification is composed of seven parts. Part 1 describes the basic MKT concept, which contains a basic overview of the software architecture and the MKT terminal. Part 2 specifies the 'card terminal – integrated chip card' (CT-ICC) interface. This is the interface for contact-type smart cards using synchronous and asynchronous data transmission.

Part 3 contains a description of an application-independent interface for terminals, which is called the 'card terminal application programming interface' (CT-API). This interface is independent of any particular programming language and has a procedural structure. It provides the following three functions: 'CT_init' for initializing a connection, 'CT_data' for data exchange using an existing connection and 'CT_close' for closing a connection.

This is complemented in Part 4 by the specification of several basic, application-independent commands for controlling terminals, which are called the 'application-independent card terminal basic command set' (CT-BCS).

Part 5 describes the ATR and general data fields for smart cards using synchronous data transmission. Part 6 contains the associated transmission protocols, as well as corresponding general commands to be sent to the terminal. Based on this, Part 7 specifies the translation of ISO/IEC 7816-4 commands into commands for smart cards using synchronous data transmission, which means memory cards.

The MKT specification was one of the first documents of its type in the world, and it has been given an extremely broad basis in Germany by thousands of terminals used with 80 million medical insurance cards. Although it certainly no longer represents the technical state of the art, it will remain a national industry standard for many years to come.

11.4.4 MUSCLE

Suitable drivers are required for using smart cards with Linux, as with all other types of PC operating systems. However, for a long time such drivers were not available, which made it rather cumbersome to use smart cards with Linux for operations such as logging on.

The first version of MUSCLE (Movement for the Use of Smart Cards in a Linux Environment), which is intended to fill exactly this gap, was published in 2000. With regard to its architecture, MUSCLE is strongly based on PC/SC, but in contrast to PC/SC the source code is openly available under a GPL license [MUSCLE], which means that it can also be modified and further developed by third parties. MUSCLE defines a Linux API that allows smart cards to be accessed in a relatively uncomplicated manner using a connected terminal.

12
Smart Cards in Payment Systems

The original primary application of smart cards with microcontrollers was user identification in the telecommunications sector. In recent years, however, smart cards have established themselves in another market sector, namely electronic payment systems. Due to the large number of cards in use, the market potential of this sector is enormous. This is underscored by the fact that more than one billion credit cards have been issued throughout the world.[1] The future applications of electronic purses include replacing conventional means of payment (banknotes and coins), shopping via global networks and pay-per-view television.

Smart cards are by nature particularly suitable for payment system applications. They can easily and securely store data, and their convenient size and robustness make them easy for everyone to use. Since smart cards can also actively perform complicated computations without being influenced by external factors, it is possible to develop totally new approaches to performing payment transactions. This is very clearly illustrated by electronic purses in the form of smart cards, which are possible only with this medium.

Electronic payment systems and electronic purses offer significant benefits to everyone involved. For banks and merchants, they reduce the costs associated with handling cash. Offline electronic purses largely eliminate the costs of data telecommunications for payment transactions. The risk of robbery and vandalism is reduced, since electronic systems contain no cash to be stolen. For merchants, the fact that transactions are processed more quickly is also a persuasive argument, since it means that cash management can be optimized. Vending machines and ticket dispensers can be made simpler and cheaper, since assemblies to test coins and banknotes are not needed. Electronic money can be transferred via any desired telecommunications channel, so it is not necessary to regularly collect money from the machines. Customers also benefit from the new payment methods, although to a lesser degree. It is not necessary to always have change on hand, and it is possible to pay quickly at a vending machine or ticket dispenser.

Ultimately, the success or failure of a payment system is determined by its potential users. If the benefits for them are too marginal, they will not use the system and will choose other means

[1] As of the summer of 2002

Smart Card Handbook, Third Edition. W. Rankl and W. Effing
© 2004 John Wiley & Sons, Ltd ISBN: 0-470-85668-8

of payment. After all, an electronic purse is just a new means of payment that complements rather than replaces other existing means of payment, such as credit cards and cash. There is no reason to fear that these means of payment, which have provided reliable service for many years, will be entirely supplanted by electronic purses in the form of smart cards.

12.1 PAYMENT TRANSACTIONS USING CARDS

The simplest approach to using cards for payment transactions is to use magnetic-stripe cards holding data for online authorization. After the user's card has been checked against the blacklist and solvency has been verified, funds can be transferred directly from the cardholder's bank account to that of the merchant. With smart cards, the scenario is slightly different, but in principle it remains the same. The smart card is logically linked to a bank account, and after unilateral or mutual authentication of the background system and the card, a previously entered amount is transferred. Naturally, PIN verification is also performed in the smart card or background system during the transaction.

Both of these scenarios are based on a background system that makes all of the decisions. They do not by any means fully exploit the capabilities of smart cards. However, there are other means and methods of making payments that can be implemented by exploiting these capabilities. Some of them are described in this chapter.

12.1.1 Electronic payments with smart cards

There are three fundamental models for electronic payments using smart cards: (a) credit cards, in which payment is made after a service is rendered (*pay later*), (b) debit cards, in which payment is made when the service is rendered (*pay now*) and (c) electronic purses, in which payment is made before the service is rendered (*pay before*).[2] These models are described below, as well as a variation on them.

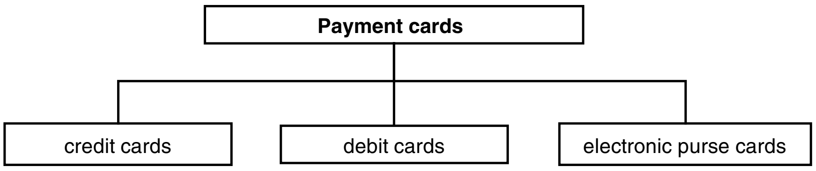


Figure 12.1 Classification of payment cards

Credit cards

The original idea of using a plastic card to pay for goods or services comes from credit cards. The principle is simple: you pay using the card, and the corresponding amount is later debited from your account. The cost of this process is borne by the merchant, who usually pays a

[2] This classification can be augmented by the category 'pay never', which relates to fraud

fee that depends on the amount of the transaction. This fee is usually around 2 to 5 % of the purchase price.

Up to now, most credit cards have not included chips. The disadvantage of such cards is that they have a relatively low level of protection against forgery. Consequently, card issuers experience significant losses due to counterfeit cards, since the merchant is guaranteed payment. Evidently, up to now, these losses have been lower than the cost of introducing cards with chips. However, credit cards will probably be supplemented with chips in the not too distant future, in order to reduce the steadily increasing cost of fraud.

Debit cards

The country in which debit cards are most widely used is Germany. A debit card, which may be a magnetic stripe card or a smart card, allows the amount of the payment to be transferred to the account of the merchant or service provider as a direct part of the payment process. With both debit cards and credit cards, the actual payment process is normally authorized by a credit check via a background system. There is usually a threshold level above which this must occur, so it is not always necessary to make a connection to the background system for small purchases. The threshold level is on the order of €200.

Electronic purses

With an electronic purse, 'electronic money' is loaded into the card before any payment is made. This can be done in exchange for cash or using a cash-free process. When a purchase is actually made, the balance in the card is reduced by the amount of the payment, and at the same time the balance of the electronic purse of the second party (who is usually the merchant) is increased by the corresponding amount. The merchant can later submit the electronic money received in this manner to the operator of the electronic purse system and be credited with the corresponding amount of real money. The user of an electronic purse thus exchanges real money for an electronic form of money that is loaded in his or her smart card. When a purchase is made, the cardholder exchanges this electronic money for goods or services.

This system has three significant drawbacks for the user. The first is that when the card is loaded, the user receives electronic money in exchange for real money. Financially, the user thus gives the operator of the purse system an interest-free loan, since it could take several weeks for the user to actually spend the electronic money, while the real money immediately becomes the property of the system operator. The amount of interest may be small for an individual user, but in total it represents a substantial source of supplementary income for the operator of the purse system. In many field trials conducted up to now, it has been found that in industrialized countries the average amount in an individual electronic purse is around 75 euros. The total average amount of money in an electronic purse system is called the 'float'. Assuming that 10 million cards are in use and the interest rate is 5 %, the total annual interest on the float amounts to 37.5 million euros, without any offsetting costs. In this example, the amount of interest lost by an individual cardholder is only 3.75 euros, which he or she will not regard as a major disadvantage. In addition to the interest income from the float, the purse system operator receives additional income in the form of unspent electronic money, due to cards that end up in collections and defective cards that are not returned for refund.

A second drawback is that a real problem arises if the purse operator goes bankrupt. This is because the card user has exchanged real money, whose value is guaranteed by the state within certain limits, for electronic money in a smart card. If the purse operator goes bankrupt, the electronic money can suddenly become worthless, and the user will have lost his or her money. Consequently, efforts are now being made in some countries to restrict the operation of electronic purse systems to banks and similar institutions. At minimum, lodging a security deposit with a government agency is required, so that the amount of money loaded in the smart cards is covered in the event that the card issuer goes bankrupt.

There is yet a third significant drawback for the user. What can the holder of an electronic purse do if it no longer works? If the purse is anonymous, not even the purse system operator can determine the amount of money that was last loaded into the card. The purse holder will also find it practically impossible to provide convincing proof of how much money was still in the card. If the chip is ruined, the electronic money is thus irrevocably lost. Unfortunately, a smart card is much less robust than banknotes or coins, for understandable reasons.

In practice, a compromise is presently used to deal with this problem. Since the last amount loaded into the card online is known, as well as the purse balance at the time of this transaction, the approximate amount in the purse can be calculated. This amount is then paid to the client. However, if a particular client frequently makes claims due to faulty smart cards, the system operator will curb his goodwill. The customer, who ultimately bears the risk, is thus denied any further compensation in the hope that he or she will take better care of the smart card in the future.

Open and closed system architectures

A distinction must be made between open and closed architectures for electronic payment systems. An open system is fundamentally available to multiple application providers, and it can be used for general payment transactions among various parties. In contrast, a closed system can be used only for payments to a single system operator.

The technical aspects of this can be briefly illustrated using a telephone card with a memory chip as an example. With memory cards, all that happens when a payment is made is that a counter is irreversibly decremented. The terminal does not have to keep an exact account of the number of units that have been deducted; it only has to ensure that the counter in the card is always properly decremented whenever the service is used (that is, whenever a call is made using the card). In this case, the terminal is a sort of machine for destroying units of electronic money. Of course, in practice a balance is kept for each terminal, but the deducted amounts are only booked to the internal accounts of the purse system operator. Fraud in settlement of the deducted amounts between the terminal owner and the purse system operator is impossible in principle, since both parties are part of the same organization (in this case, the telephone company).

In an open system, the terminal owner and purse system operator can be completely different bodies. The purse system operator must therefore be able to verify that the accounts for the terminal receipts are correct and not manipulated. This must be taken into consideration from the very beginning in the system design, since otherwise account settlement between the terminal owner and purse operator will be very difficult or impossible. In the above example using a memory card, the system concept makes it impossible for the terminal operator to convincingly guarantee the purse system operator that the claimed amount is correct. This is because the terminal operator can only present an invoice for a certain number of units, instead of forgery-proof signatures for the amounts paid, as would be possible with a genuine electronic purse system.

System architecture and terminal connections

The system architecture of an electronic payment system using smart cards can be either centralized or decentralized. With payment systems in particular, system security is the most important issue. There is thus frequently a tendency to use centralized systems, since this gives the system operator complete control of the system.

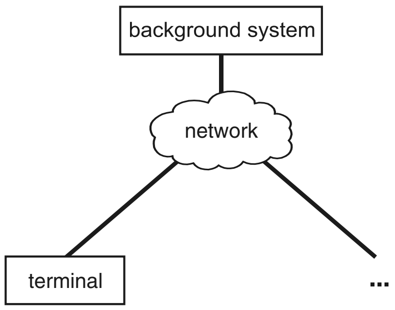


Figure 12.2 The basic architecture of a centralized system for electronic payments. All of the illustrated connections are permanent

In concrete terms, a centralized system means an online system in which every payment transaction is performed directly and online by the background system. If a communications link cannot be established, payment is not possible. Nevertheless, a centrally operated system has certain advantages. For instance, incoming transactions can be directly compared with the current blacklist in real time. Key exchanges can be carried out directly by the background system without any delays. The software in the terminals and the general parameters in the cards can be updated directly and with little additional effort, since a direct link to the background system must be established for each transaction.

However, these advantages are offset by several major disadvantages. In many countries, telecommunication charges are so high that it is not reasonable for merchants to have permanent links to background systems or to dial up a background system for each transaction. In some areas, the telephone network is not sufficiently reliable to allow an online link to the higher-level computer to be established at any desired time.

Due to their active nature, smart cards are excellent for use in decentralized systems, since they contain part of the system security 'in house'. This is also their main advantage relative to passive magnetic-stripe cards, which cannot force the system to perform specific procedures.

In particular, using electronic purses with automated equipment, such as vending machines and ticket dispensers, compels the use of a decentralized system, since electronic purses can operate completely independently for weeks or months and do not have any means to connect to an existing communications system. A decentralized system is thus often preferred. In addition, a decentralized system has significantly better characteristics with regard to robustness. If the background system fails in a centralized system, all electronic payments are blocked. In a decentralized system, by contrast, the consequences of a temporary failure usually do not even reach as far as the merchant terminals.

Decentralized systems also have certain disadvantages, primarily in the area of system management. This is because online connections can only be established at certain times, and as a rule only by the terminals. However, it is essential for system security that the terminals always use the current blacklist. This is one of the reasons why many systems require each terminal to establish an online connection to the background system at least once a day. This is used to transmit the accumulated transaction data to the background system, with various types of administration data being transmitted to the terminal in return. Some examples of this administration data are new terminal software, new key sets, the current blacklist and data to be loaded into customers' cards.

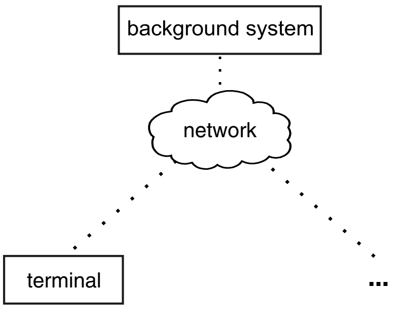


Figure 12.3 The basic architecture of a decentralized system for electronic payments. All of the illustrated connections can be established as needed

In practice, mixed solutions that are neither fully centralized nor fully decentralized are often used, in order to combine the advantages of the two architectures while avoiding their disadvantages. A mixed solution consists of allowing both the terminals and the smart cards to compel online connections under certain conditions. If an online connection cannot be established, the payment does not take place. Some typical conditions are: (a) online authorization is required for payments above a certain amount, which can usually be set individually for each smart card by the system operator; (b) the number of offline transactions and the amount of time since the last online transaction can be used to decide whether to go online; (c) a random number generator can be used to force a certain percentage of all transactions to take place online. Some systems also have a special button on the terminal that forces an online transaction. This button can be pressed by the sales staff if they suspect that the customer is using a manipulated card.

All of these criteria ensure that on average, every card makes a direct connection to the background system within a defined and statistically computable time interval. The system operator thus recovers direct control over the system, which he initially lost by using a decentralized

Table 12.1 Typical actions and conditions that trigger an online connection between a smart card and the background system

Action or condition	Usual value
Transaction type (e.g., cash disbursement)	—
POS conditions (e.g., PIN pad present)	—
Parameter-driven random selection	10 %
Manual request at the merchant terminal	1 %
First use of the smart card	—
Number of offline transactions performed since the last online transaction	10
Accumulated offline amount since the last online transaction	500 euros
Time since the last online transaction	7 days
Payment amount exceeding a configurable threshold value	200 euros

system. Terminals and automated machines having only a small turnover can be excluded from these online constraints, since even in the case of fraud only small losses can occur. This saves the cost of a link to a communications network, since data exchange can be performed manually by service personnel.

12.1.2 Electronic money

Electronic money must have certain properties if it is to be used with the same flexibility as normal money. If these properties are wholly or partially absent, the capabilities of electronic money are necessarily more or less limited. The essential properties necessary to minimize the difference between electronic money and real money are described below.

Processable

An important, although in principle trivial, property of electronic money is that it can be completely and automatically processed by machines. This is the only way in which large systems can be operated economically.

Transferable

Electronic money must not be bound to a particular medium, such as smart cards. It must be possible to transfer electronic money using any desired medium, such as a network or computer.

Divisible

Electronic money must be divisible, so that any desired amount can be paid without recourse to using normal money. This is similar to normal money, which although not arbitrarily divisible, is available in a sufficient number of different denominations that normal purchases can be made using a small number of coins and banknotes.

Decentralized

Payment systems with centralized architectures can be easily monitored by the purse system operator, with opportunities for fraud being very limited. The best example of such a system is the online authorization of credit card purchases. However, centralized systems suffer from many drawbacks. They are expensive, vulnerable to technical disturbances, inflexible and difficult to modify or extend. Systems with decentralized architectures minimize these drawbacks. This can be seen very clearly with payments in the private sphere, in which money changes hands without any involvement by a central body. Electronic money should also have this property, since it otherwise cannot compete with normal money. For an electronic payment system, in concrete terms this means that it must be possible to make offline payments and to make payments directly from one purse to another one. The property of allowing direct payments between purses (purse-to-purse transactions) is sometimes called ‘transferability’.

Monitorable

Despite the demand for anonymity, electronic money must allow the purse system operator to monitor the system, since this is the only way in which manipulations and security gaps can be recognized and eliminated. This is exactly the same as the situation with normal money, in which every citizen is obliged to immediately report counterfeit money to the appropriate authorities. In the case of electronic money, the purse system operator is responsible for guarding against fraud and forgery, and he can and must monitor the consistency of payment flows.

Secure

A fundamental property of electronic money must naturally be security against forgery. Any system will collapse within a short time if it is possible to forge or duplicate money in any form or manipulate payment flows. This is why cryptographic functions are used so extensively in the field of electronic payments, since this is the only way to achieve the required level of security.

Anonymous

Anonymity means that it is impossible for anyone to associate payments with particular persons. The value of this requirement is very much a question of perspective. From a technical perspective, the purse issuer desires a system with as little anonymity as possible, so he can monitor the system in the best possible manner. The possibility of fraud is very limited in non-anonymous systems, since anyone who commits a fraud can quickly be identified. Government agencies, such as the police and tax authorities, have similar interests. Non-anonymous electronic money would give them considerably more scope for monitoring financial transactions than they have enjoyed up to now with normal money.

The position of purse users is diametrically opposite. They consider current payment methods using normal money to represent an excellent state of affairs, and they regard complete anonymity and non-traceability of payment transactions as the optimum solution.

Particularly with regard to anonymity, operators of electronic purse systems often choose a compromise solution in the interest of system security. For instance, in most systems payments are anonymous, but loading electronic purses is not. This allows the system to be monitored reasonably well in a simple manner at a relatively low cost.

At first sight, some of these properties appear to be contradictory. For instance, in many cases complete anonymity and optimum system monitoring are mutually exclusive. However, this field is in the early stages of development, and there are already systems being planned in which these two properties can definitely be realized simultaneously.

There are two properties of real money that are not mentioned above, although they are highly significant. The first is that real money is legal tender that must be accepted by everyone in a particular country. In almost all countries, vendors of goods or services are obliged to accept the legal currency of that country as a means of payment. The second property relates to the stability of the currency. Except for a few countries with high rates of inflation, the legal currency in circulation has a stable value. If this is not the case, people resort to barter or using foreign currencies.

12.1.3 Basic system architecture options

Electronic payment systems based on smart cards can be constructed in a wide variety of manners. For economic reasons, they are often based on existing systems, most of which are based on magnetic-stripe cards. However, there is no single basic model that applies to all payment systems, since the requirements vary too widely. We can therefore only describe the basic principles of such systems in terms of their essential components.

Large smart card payment systems basically consist of four different components. These are the background system, the network, the terminals and the cards.

Background system

The background system consists of two parts: *clearing* and *management*. The clearing subsystem maintains the accounts of all of the banks, merchants and cardholders participating in the system, and it books all incoming transaction data. It also provides the system monitoring functions. A simple example of such a function is maintaining a running balance to check whether the total of the amounts submitted to the clearing system exceeds the total amount of money in the electronic purses. If it does, an attacker has loaded money into smart cards without the knowledge of the background system.

The management part of the background system controls all administrative processes, such as distributing new blacklists, switching to new key versions, sending software updates to the terminals and so on. This subsystem also generates data sets for personalizing smart cards.

The background system has complete control of the electronic payment system, regardless of the system architecture. Even with systems that work completely offline, the background system establishes the global system parameters and monitors the security and operation of the system.

Network

The network links the background system to the terminals. The connections may be circuit-switched (e.g. ISDN) or packet-switched (e.g. X.25). As a rule, the network is totally transparent to the data traffic, which is passed unmodified from the sender to the receiver.

Terminals

The various types of terminals can be classified as either loading terminals or payment terminals, according to their functions with respect to payments. They can also be classified as automated terminals or attended terminals. The classic example of an automated terminal is a cash dispenser (ATM). In electronic purse systems, automated terminals are primarily used only to load cards. It would naturally also be conceivable to allow an electronic purse to be emptied using such a terminal, with the balance being paid out in cash. Attended terminals are typically located at supermarket checkouts and in retail shops. They are always used to pay for goods. In some systems, terminals in banks can also be used to load smart cards in exchange for cash payments.

Smart cards

Smart cards are the most widely distributed component of the system. They can be used as electronic purses, but they can also be used as security modules in various types of terminals. Another use is transporting data between various system components. Cards for this purpose, which are called transfer cards, are used to manually transfer transaction data from a terminal that works completely offline to one that works online (such as a cash dispenser).

The example system shown in Figure 12.4 illustrates the system components and their logical connections. The background system, which may be the background system of a different operator or a component of the system itself, is connected to the other components via a transparent network.

Electronic purses are must commonly loaded using cash dispensers, most of which operate online, although they can also operate offline for a limited time in the event of a network failure. For this reason, they have their own security modules, which hold all of the keys necessary for normal operation and key derivation.

There are also electronic purse payment systems that operate fully offline. Two examples are parking meters and terminals in taxis. In such cases, transfer cards can be used to transport the transaction data from the security modules to a cash dispenser, from which they reach the

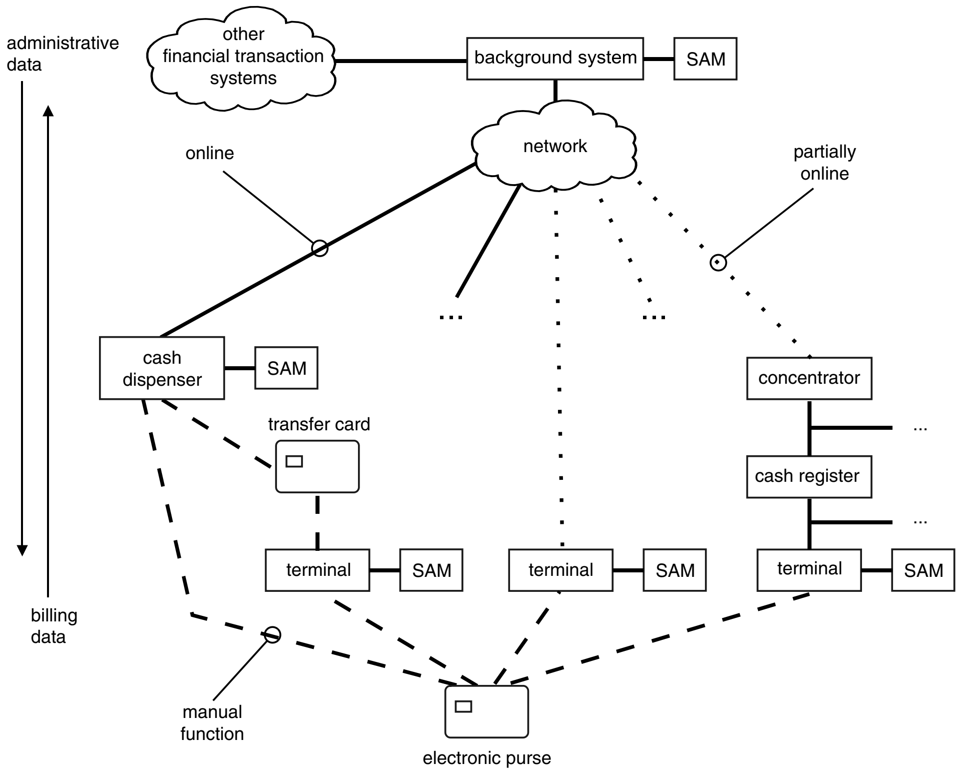


Figure 12.4 Example architecture of an electronic purse system (SAM = security module)

background system via the network. In exchange, the terminals receive current administration data, such as blacklists and software updates.

A second type of payment terminal is one that is connected to the network via an online connection that is established as necessary. This type of terminal normally works offline, but it periodically connects to the background system in order to exchange any available billing and administrative data.

A third type of payment terminal has no direct connection to the network. For example, it could be connected to a supermarket cash register that in turn is connected to a concentrator located in the facility. This concentrator, which is normally a PC acting as a server, might connect to the background system once a day via the network. The necessary data exchanges occur during this connection.

The Quick electronic purse system in Austria and the Geldkarte system in Germany are similar to the example system just described, and many parts of the Visa Cash electronic purse system correspond to what has just been described. For large applications, it is quite common to use a distributed system architecture consisting of several different background systems operating in parallel. With such an architecture, several different purse systems with more than one system operator can be operated with mutual compatibility.

12.2 PREPAID MEMORY CARDS

With regard to electronic payment systems, we must not neglect memory cards. They are produced in very large numbers in the form of prepaid electronic purses, and they are used in many applications. This situation will certainly not change over the next few years. Although memory cards will slowly but surely be replaced by microcontroller smart cards, their strength lies in their unparalled low price. The most common application for prepaid electronic payment cards is prepaid telephone cards, which are widely used in many countries and simply discarded once they have been used up.[3]

A memory card[4] only has to contain some control logic and an irreversible down counter to allow it to do its job. More recent versions also support unilateral authentication of the card by the terminal. For this purpose, the logic unit has been enhanced by adding a simple encryption

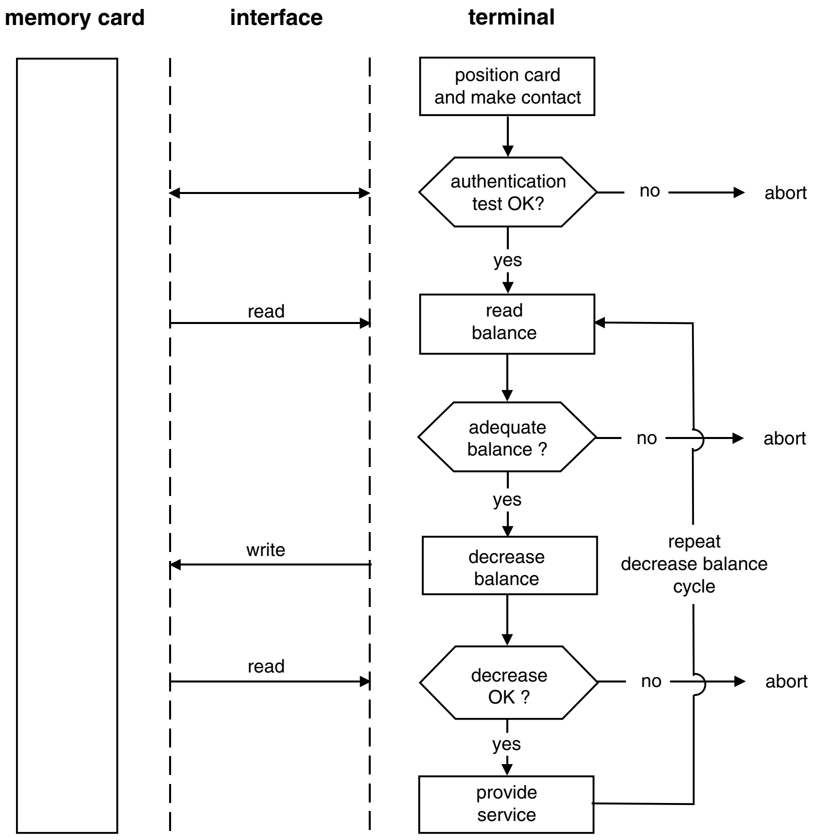


Figure 12.5 The devaluation cycle of a prepaid memory card as seen by the terminal

[3] See also Section 13.6, 'Public Card Phones in Germany'
[4] Chip architecture for memory cards; see also Section 2.3.1, 'Memory cards'

function, whose task is to encrypt a random number received from the terminal using a secret key stored in the card and return the result to the terminal. This is the only way the terminal can be sure that the memory card being used is genuine.

Beside the fact that it is difficult to ensure the authenticity of memory cards, they have another drawback that limits their use as a general-purpose medium for electronic payments. Since such cards can easily be manipulated, it is difficult to construct a payment system that can support a variety of independent service providers (such as kiosks and taxis). This is because secure communications between the terminal and the card are not possible, so proper accounting of the amounts paid, and thus overall system monitoring, are in principle only possible using indirect methods.

Due to their low cost, memory cards enjoy advantages with respect to microcontroller cards in certain very restricted application areas, but they are not particularly suitable for open applications in payment systems. Future electronic payment systems will doubtless use predominantly microcontroller smart cards, since they are significantly more versatile.

12.3 ELECTRONIC PURSES

The idea of implementing an electronic purse in a smart card goes back to the early days of smart card technology. However, only since the mid-1990s has this concept been realized, since that was when the development of large systems first began.

If we take a normal purse containing coins and banknotes as a reference, we can easily see which properties electronic money has in the eye of the user. Money must be put into a purse before it can be used, which means it cannot be used like a debit or credit card, where payment is only made on or after receipt of the goods or services. Instead, it is used like a telephone card, which must be paid in advance. The actual payment process must be quick and simple, since otherwise the level of user acceptance will be low. Furthermore, all payments from a purse are anonymous, which means that it is not possible to reconstruct who bought what at which time. The most annoying characteristic of a purse becomes apparent if it is lost, since the money in it is then irretrievably gone. However, this does not necessarily have to be true of an electronic purse.

The primary advantage of a purse, or rather of the money it contains, is that it is accepted everywhere within a given country. It is precisely this factor that is missing in most existing electronic purse systems. With a telephone card, all you can do is make telephone calls, and nothing else. This is typical of a closed application. An ideal electronic purse, on the other hand, can be used in more than one sector, and thus allow its user to make payments in many different businesses using a single purse.

12.3.1 The CEN EN 1546 standard

The European Commission decided in 1990 to have the *Comité Européen de Normalisation* (CEN) produce a European standard for a multisector electronic purse system. Work on the standard started in 1991. Up to 1998, the various project teams had spent around eight man-years on developing this standard. Since a number of independent experts have participated in this effort, it very unlikely that there are any security gaps in this standard. It is thus already

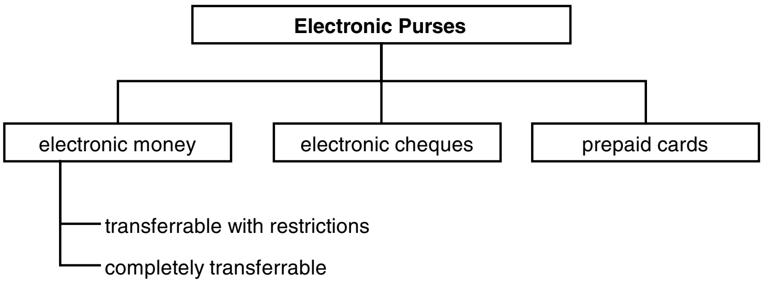


Figure 12.6 Classification of electronic purse systems based on smart cards. Electronic money is not tied to a particular card and can be denominated as desired. By contrast, electronic checks have fixed denominations. Prepaid cards (electronic purses) contain electronic money that is tied to the card and can be freely denominated

quasi-evaluated. The essential parts are presently no longer subject to change, and they will be published as a European standard after the final vote takes place.

The EN 1546 standard is a public standard, and the processes of its individual functions are described in great detail. It is therefore very suitable for demonstrating payment and loading processes in an electronic purse system. With many existing systems, these processes cannot be described in the same level of detail, since the relevant commands, processes and internal functions are confidential. This standard is thus very useful for illustrating the fundamental external and internal processes of an electronic purse system.

The typical application areas are clearly indicated by the first systems based on this standard. The Danish system operator Danmønt has introduced a purse compliant with this standard into its existing system. In Austria, the Eurocheque card issued throughout the country includes an electronic purse (called Quick) based on the EN 1546 standard, in addition to other applications. However, the largest international application of this standard is the Visa Cash system. This is one of several electronic purses offered by Visa. The essential features of the international specification for electronic purse systems, which is titled 'Common Electronic Purse Specifications' (CEPS), are also based on EN 1546.

The EN 1546 standard is titled 'Inter-Sector Electronic Purse' and is divided into four parts. The first part, 'Definition, concepts and structures', describes the overall system. This basic document defines and explains, in abstract form, all of the logical components and their interconnections. In the second part, these basic concepts are used to describe the security architecture of the overall system and its individual components. This includes not only mechanisms for maintaining security, but also possible attacks and the corresponding necessary countermeasures.

Part three, 'Data elements and interchanges', contains descriptions and definitions of the data elements needed for the electronic purse system. It also describes the commands related to the smart cards and security modules and their associated responses.

The final part describes the state machines and states for the devices used. It employs a symbolic representation similar to well-known flowchart diagrams. This formal notation, which is known as SDL notation, is derived from the CCITT Z.100 recommendation.[5]

[5] See also Section 4.2, 'SDL Notation'

This standard, which encompasses around 300 pages, thus contains a complete description of an electronic purse system, including the smart cards, the terminals with their security modules and the background and clearing systems. Its objective is to establish a common standard for large electronic purse systems with very many smart cards and wide geographical distribution.

The advantage of a general standard for electronic purse systems is primarily that it allows individual, independently operated systems to be mutually compatible. As with GSM, this gives the user the option of being able to use his or her card in the future to make payments using the systems belonging to other purse providers. This is an essential prerequisite for the success of this sort of payment system.

However, a small comment is in order at this point. The EN 1546 standard provides a large amount of freedom with regard to actual implementation, and it regards itself as more of a framework than a precise specification of individual bits and bytes. It is thus perfectly possible for two different systems to be fully compliant with this standard but mutually incompatible, for example because they use different cryptographic algorithms.

The basic elements of the system architecture are shown in Figure 12.7. The purse provider bears the overall responsibility for the system and is also the system manager. He is comparable to a GSM network operator. The term 'purse holder' is defined in the standard to refer to the user of the electronic purse. This is the person who makes payments using the electronic purse application in the card and receives goods or services in return. There are also three other parties that perform functions in the system. The 'service provider' offers goods

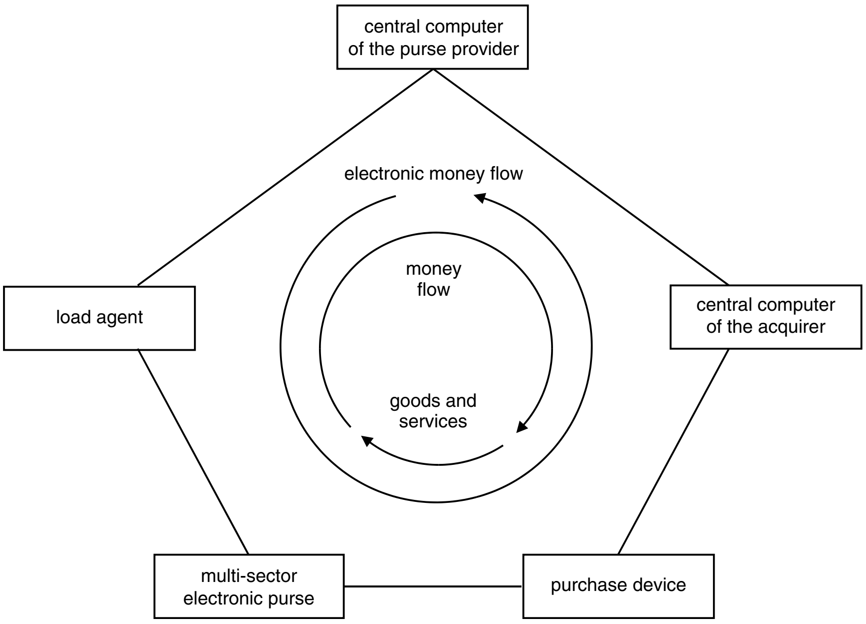

central computer of the purse provider
electronic money flow
money flow
goods and services
load agent
central computer of the acquirer
multi-sector electronic purse
purchase device

Figure 12.7 Basic structure of an intersector electronic purse system and associated payment flows according to EN 1546

or services that are accepted by the user and paid using an electronic purse. The 'acquirer' is responsible for establishing and managing the data links between the purse issuer and the service providers. He may also consolidate the individual transactions arriving from the payment facilities, so that the purse provider only receives collective certificates. The 'load agent' is the counterpart of the service provider, since he can reload the electronic purse in exchange for a payment.

These five parties need not all be real persons or firms; they may also be virtual. However, real technical components are allocated to each of them, classified according to their level of security. Components that are regarded as secure prevent any external manipulation of the data that are processed or stored within them. With components regarded as non-secure, such manipulation is at least theoretically possible. However, the system as a whole is designed such that the manipulation of any of the components identified as non-secure in Figure 12.8 will not affect the overall security of the system.

Here the abbreviation 'IEP' stands for 'inter-sector electronic purse' and refers to an intersector electronic purse application in a smart card. A purchase device is used to pay for received goods or services. It is a terminal with keypad and display, and it must also have a security module. The term 'secure application module' (SAM) is used in the standard to refer to all types of security modules. A SAM contains all secret keys necessary for transactions between the IEP and the central computer of the purse provider. Naturally, the keys never leave

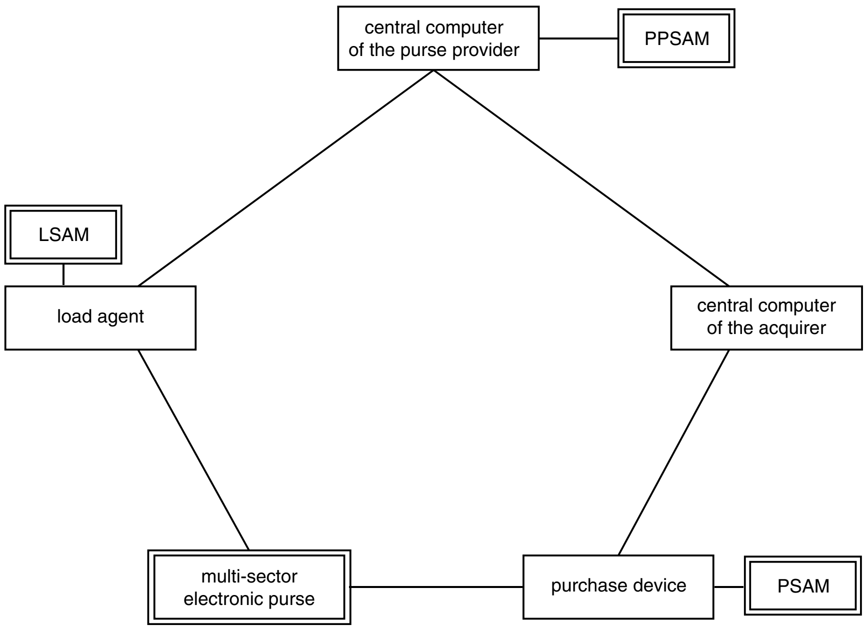

central computer of the purse provider
PPSAM
LSAM
load agent
central computer of the acquirer
multi-sector electronic purse
purchase device
PSAM

Figure 12.8 Components and connections of electronic purse systems according to EN 1546. The components with a single outline are not secure, while those with a double outline are secure

the security module, but are used only inside the SAM by the cryptographic algorithms.[6] In many cases, therefore, the links between the system's secure components are all direct. The non-secure components are only used to transparently relay sensitive data.

Smart cards have made it possible to convert the idea of an electronic purse system into reality. They thus form the central subject of the system description in the CEN EN 1546 standard. All of the associated files, commands, states and processes are defined and described in this standard. In order to define the entire system, there are similar parts that address the security module and the other components.

However, since this is a standard rather than a specification, the purse provider is naturally given a great degree of freedom and many options. A variety of functions can be used to construct the purse. For example, a simple system that only allows loading and paying with the purse can be easily implemented. This can be further enhanced with functions for canceling payments, modifying purse parameters and converting currencies. The exact selection of the many complex options is largely left to the purse provider, who must choose the options that best meet his particular needs.

The most important aspects of an intersector electronic purse system with regard to the IEP, which is the smart card, are described below.

EN 1546 data elements

Designations for all data elements were introduced to allow the data used in the entire system for the electronic purse application to be referred to unambiguously. Data flows and data processing can be represented, simply and unambiguously, in a mathematically correct notation using these very short designations. The standard also contains a simple data dictionary, which describes the corresponding data contents and associated formats for the standardized data elements.

Files

The complete electronic purse application is contained in a dedicated DF in the smart card. All of the files necessary for proper operation are contained in this DF. In addition, information relating to the card, the chip, other applications and the like is stored in several files directly under the MF. The data elements needed for operating the electronic purse are contained in six EF files located in a DF for the purse. These files are listed and briefly described in Table 12.3.

The EF_{IEP} file specifies the general parameters of the purse, which form the basis for all transactions that take place. The EF_{IK} file contains specific information for every available key. EF_{BAL} contains the amount that is currently in the purse and available to the user. The log files are used exclusively to record all transactions, separated by function. Only with these files is it possible to cancel a payment or handle errors. There are separate log files for loading, paying, modifying the purse parameters and making currency conversions. All log files have a cyclic structure, in order to record the most recent transactions.

[6] A possible key management system for an electronic purse system that is compliant with EN 1546 is described in Section 4.8, 'Key Management'

Table 12.2 Summary of the most important standard data elements of EN 1546

Data element	Description
ALG_{IEP}	cryptographic algorithm used by an IEP
AM_{IEP}	authentication mode required by an IEP
AP_{IEP}	application profile of an IEP
BAL_{IEP}, BAL_{PSAM}, BAL_{PPSAM}	balance of an IEP, PSAM or PPSAM
$BALmax_{IEP}$	maximum balance of an IEP
CC_{IEP},CC_{PSAM},CC_{PPSAM}	completion code from an IEP, PSAM or PPSAM
CT	collection status
$CURR_{IEP}$, $CURR_{LDA}$, $CURR_{PDA}$	actual currency for an IEP, LDA or PDA
$DACT_{IEP}$	activation date of an IEP
DD	discretionary data
$DDEA_{IEP}$	deactivation date of an IEP
$DEXP_{IEP}$	expiry date of an IEP
ID_{IEP}, ID_{PSAM}, ID_{PPSAM}	identifier for an IEP, PSAM or PPSAM
IEP	intersector electronic purse
IK_{IEP}, IK_{PSAM}, IK_{PPSAM}	key information for an IEP, PSAM or PPSAM
LDA	load device application
LSAM	load SAM
M_{LDA}, M_{PDA}	transaction amount for load or purchase
$MTOT_{IEP}$, $MTOT_{PSAM}$	total transaction amount for a purchase
NC	number of collections (designates a final sum)
NI	number of individual transactions
NT_{IEP}, NT_{LSAM}, NT_{PSAM}	transaction number for an IEP, PSAM or PPSAM
PDA	purchase device application
PP_{IEP}, PP_{PSAM}, PP_{PPSAM}	purse provider identifier for an IEP, PSAM or PPSAM
PPSAM	purse provider SAM
PSAM	purchase SAM
R	random number
S_1	IEP signature
S_2	PSAM or PPSAM signature
S_3	IEP signature
S_4	PSAM signature
SAM	secure application module
TM	total amount
TRT	transaction type and status

Table 12.3 The files and data elements needed for an electronic purse, in accordance with EN 1546. The log files for exchange rate calculations and parameter changes are not shown

File	Function	Data elements	Descriptions
EF_{IEP}	Fixed data and parameters for the purse	PP_{IEP}	IEP purse provider identifier
		ID_{IEP}	IEP identifier
		$DEXP_{IEP}$	IEP expiry date
		$DACT_{IEP}$	IEP activation date
		$DDEA_{IEP}$	IEP deactivation date
		AM_{IEP}	IEP authentication mode
		AP_{IEP}	IEP application profile
		DD	user-specific data
EF_{IK}	Information relating to all keys	ALG_{IEP}	IEP cryptographic algorithm
		IK_{IEP}	IEP key information
		DD	user-specific data
EF_{BAL}	Purse balance	BAL_{IEP}	EP balance
		$CURR_{IEP}$	IEP currency
		$BALmax_{IEP}$	IEP maximum balance
		DD	user-specific data
EF_{TFIELD}	Transaction field	NT_{IEP}	IEP transaction number
EF_{LLOG}	Log file for loading	TRT	transaction type and status
		NT_{IEP}	IEP transaction number
		BAL_{IEP}	IEP balance (new balance)
		M_{LDA}	transaction amount for loading
		$CURR_{LDA}$	currency for loading
		ID_{PPSAM}	purse provider identifier for PPSAM
		CC_{IEP}	IEP completion code
		DD	user-specific data
EF_{PLOG}	Log file for payment	TRT	transaction type and status
		NT_{IEP}	IEP transaction number
		BAL_{IEP}	IEP balance (new balance)
		M_{PDA}	transaction amount for purchase
		$CURR_{PDA}$	currency for purchase
		ID_{PSAM}	purse provider identifier for PSAM
		NT_{PSAM}	PSAM transaction number
		CC_{IEP}	IEP completion code
		DD	user-specific data

Commands[7]

The files form the foundation of the purse, and the commands are built on this foundation. Eight of these commands are needed for operating the purse system. Three commands belong to the ISO/IEC 7816-4 standard: SELECT FILE, READ BINARY and READ RECORD.

[7] For detailed descriptions of the commands, see Section 7.14, 'Commands for Electronic Purses'

These commands are only used to select the electronic purse application using its AID and subsequently read various data from the purse files as necessary.

The other five commands were developed specifically for use with electronic purses. They are always used in pairs for individual transactions, since in principle they act as a sort of mutual authentication. During authentication, data needed for the transaction are also exchanged. The corresponding commands and responses are naturally structured such that any manipulations at the interface between the card and terminal can be immediately detected, resulting in immediate termination of the transaction and logging of the event.

All purse commands directly access data elements in the purse files for both reading and writing. The files are automatically selected by the operating system prior to these accesses. For instance, basic purse data are occasionally needed while a command is being processed. In this case, the operating system selects the EF_{IEP} file, and the desired data element is provided to the command. All transactions, as well as the most important data, are recorded in suitable log files during the command–response cycle.

EN 1546 defines the commands listed in Table 12.4 and specifies their functions inside the card in detail.

Table 12.4 Specific commands for electronic purses as defined in EN 1546

Command	Function
INITIALIZE IEP	Initialization for a subsequent purse command
LOAD IEP	Loading the purse, canceling a previous payment and error recovery
DEBIT IEP	Paying using the purse and confirming payment
CONVERT IEP CURRENCY	Converting currencies
UPDATE IEP PARAMETER	Modifying general purse parameters

The standard does not provide commands for verifying or changing PINs, since these functions are not needed for the proper operation of the purse. However, additional commands for PIN verification and management can be included in the purse application as necessary, without causing any interference or problems with the existing purse commands.

States

As may already be apparent from the command summary, each transaction consists of an introductory initialization command and a subsequent command that completes the transaction. In order to fix the sequence of commands, state diagrams are used to define the necessary states and state transitions in the application. This naturally requires the card to contain a state machine. Depending on its current state, the card will accept or reject various commands.

Cryptographic algorithms

The entire security of the system is based on a cryptographic algorithm. The messages exchanged between the components all have appended signatures to allow manipulations to be

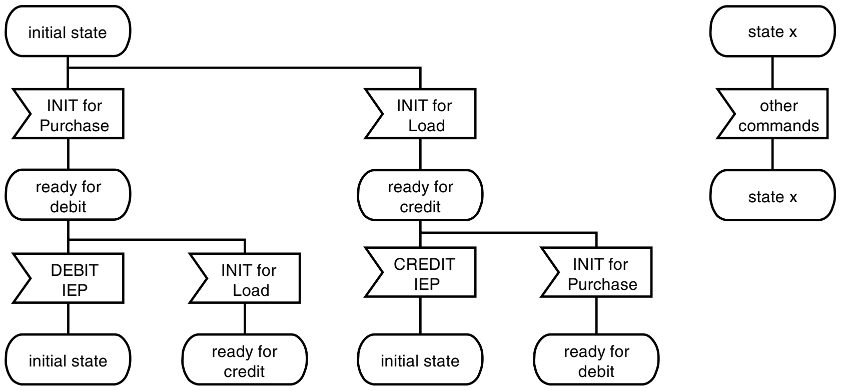


Figure 12.9 Simplified state diagram for loading and paying with an EN 1546 electronic purse

detected. This is the only protection for messages, which are always exchanged in plaintext.[8] The message exchange is structured such that any desired type of cryptographic algorithm can be used to generate the signature. The symmetric DES algorithm is currently most commonly used, but the standard also allows asymmetric algorithms such as RSA or DSS. This algorithm independence is a great advantage, since it considerably extends the useful life and flexibility of the standard.[9]

Procedures

The standard does not just specify files, commands and states, but also describes and explains the associated procedures. These are specified in detail in terms of their data elements, using a pseudolanguage similar to Basic. This is necessary because the security of some processes strongly depends on the sequence of the operations used for processing commands inside the card. During a transaction, for example, the appropriate log file must always be updated before the response is sent to the terminal. The processes and transactions for all components are also precisely specified. Descriptions are provided for the following electronic purse processes for smart cards:

- loading
- paying
- canceling a payment
- correcting an error
- converting currencies
- changing the purse parameters.

[8] If the DES algorithm is used, a 4-byte MAC is provided as a signature at the end of the message
[9] An example of a possible key hierarchy with key derivation is described in Section 4.8.6, 'Key management example'

The commands for reading files can of course also be used for monitoring purposes. However, the actions involved may vary, depending on the purse provider and the objective of the monitoring.

The structure of all specified procedures is determined by a fundamental principle, which is that it must never be possible to create electronic money by manipulating data transmissions between the components or by abruptly terminating a transaction. The worst consequence of such actions is limited to the destruction of electronic money. This automatically excludes certain types of attack. Incidentally, this principle is incorporated in the designs of nearly all electronic purse systems.

Each process is basically divided into three phases. Complete initialization of the participating components occurs in the first phase. Actual execution of the transaction takes place in the second phase. The third phase, which is optional, is used to confirm the previous actions. Successful completion of the first two phases amounts to unilateral (or optionally, mutual) authentication of the two components.

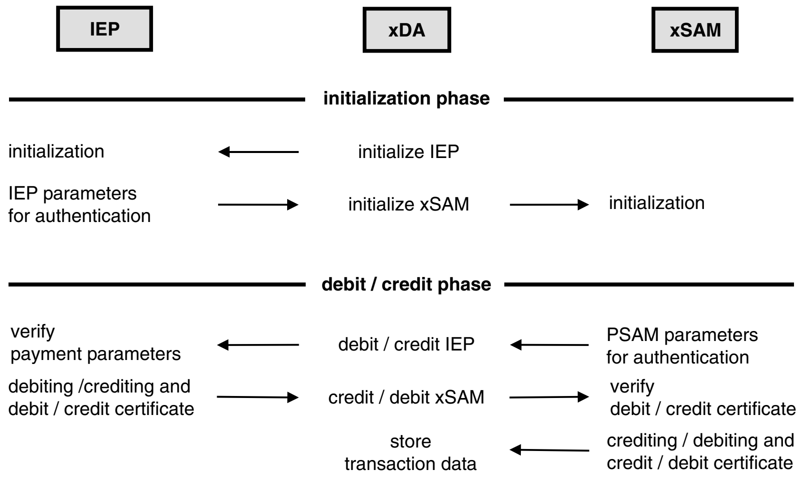

IEP
xDA
xSAM
initialization phase
initialization
initialize IEP
IEP parameters for authentication
initialize xSAM
initialization
debit / credit phase
verify payment parameters
debit / credit IEP
PSAM parameters for authentication
debiting /crediting and debit / credit certificate
credit / debit xSAM
verify debit / credit certificate
store transaction data
crediting / debiting and credit / debit certificate

Figure 12.10 The basic sequence of an EN 1546 electronic purse transaction (phases 1 and 2)

In all of the procedures, unilateral or mutual authentication is interleaved with the actual purse functions (payment, loading, etc.). This minimizes the time required for the purse transactions and increases security, since it significantly reduces the number of commands needed to perform the functions. This is similar to the situation with the standard ISO commands INTERNAL AUTHENTICATE and EXTERNAL AUTHENTICATE, which can be replaced by the non-standard command MUTUAL AUTHENTICATE without affecting functionality or security.

Procedure for loading a purse

Before an electronic purse can be used to make payments, it must first be loaded. The procedure for this is shown in Figure 12.11, which illustrates the option in which the electronic purse is loaded online using a terminal that is directly connected to a background system with its associated security module (PPSAM). Other options are also possible according to the standard, such as loading the purse using a security module in a terminal (LSAM). However, the procedure described here is commonly used in current systems, since it gives the system operator complete control over loading.

Table 12.5 Abbreviations of functions and procedures used in Figures 12.11 and 12.12, which illustrate payment transactions according to the EN 1546 standard

Abbreviation	Meaning
Ax	Unique label for an action
Cx	Unique label for a command
Parameters (. . .)	Request to a participant using the indicated data elements
Response (. . .)	Response to previous request, with the indicated data elements
Rx	Unique label for a response
Sign (. . .)	Generate a signature for the indicated data elements
Verify (. . .)	Verify the indicated data elements or function
Write (. . .)	Write the indicated data elements to a file

In the example shown in Figure 12.11, an electronic purse (IEP) is loaded by the background system via a terminal (LDA) using a security module (PPSAM). The card user first inserts his card into the terminal, which executes a reset. In the ATR, the card sends the terminal various general parameters for the subsequent communication process. After this, the terminal selects the electronic purse DF in the card. Once this has been done successfully, the card user inserts the amount of money to be loaded into the terminal, using an acceptable currency. This information is then sent to the PPSAM via the first purse command. The PPSAM verifies the indicated currency and the amount still allowed to be loaded. In response, it returns three data elements to the terminal.

The terminal appends the load amount (M_{LDA}) and the associated currency ($CURR_{LDA}$) to the data elements received from the PPSAM and sends all of this information to the IEP using the 'INITIALIZE IEP for Load' command. The IEP then checks, among other things, whether the purse balance after the load amount is added would exceed the maximum allowable amount in the purse ($BALmax_{IEP}$). If it would not, the IEP increments a transaction counter (NT_{IEP}), computes the session key ($KSES_{IEP}$) and generates a signature (S_1). These are returned to the terminal, along with several other data elements.

Following this command, the terminal simply relays the received data elements to the PPSAM. Here they are checked against the permitted range of values, and a card-specific key (KD_{PPSAM}) and a session key ($KSES_{PPSAM}$) are generated. If the subsequent verification of signature S_1 is successful, the card has been authenticated, since it must know the secret key for computing S_1. The PPSAM then generates signature S_2 and sends it to the terminal, along with the key information (IK_{PPSAM}). The terminal again only relays these data elements to the card, this time using the command LOAD IEP.

The IEP now verifies signature S_2. If this is successful, the PPSAM has also been authenticated by the IEP. The balance in the purse (BAL_{IEP}) is then increased. The IEP next generates a third signature (S_3), which is sent to the terminal for confirmation that the balance has been successfully increased. The final command transfers this signature to the PSAM, which completes the entire loading transaction.

IEP		**LDA**		**PPSAM**
R1: Response (ATR)	← →	C1: RESET		
R2: Response (CC_{IEP})	← →	C2: SELECT Parameters (DF_{IEP})		
		A1: Input ($M_{LDA} \Vert CURR_{LDA}$)		
		C3: INITIALIZE PPSAM for Load Parameters ($M_{LDA} \Vert$) $CURR_{LDA}$)	→ ←	A2: Verify ($CURR_{LDA}$) Verify ($BAL_{PPSAM} \geq M_{LDA}$) Generate R R3: Response ($PP_{PPSAM} \Vert$ $ID_{PPSAM} \Vert R$)
A3: Verify (PP_{PPSAM}) Verify ($CURR_{LDA}$) Verify ($BAL_{IEP} + M_{LDA} \leq$ $BALmax_{IEP}$) $NT_{IEP} := NT_{IEP} + 1$ $KSES_{IEP} = f(KD_{IEP},$ $DEXP_{IEP}, NT_{IEP})$ S_1:=Sign ($PP_{IEP} \Vert ID_{IEP} \Vert$ $DEXP_{IEP} \Vert NT_{IEP} \Vert$ $M_{LDA} \Vert CURR_{LDA} \Vert$ $BAL_{IEP} \Vert ID_{PPSAM} \Vert R$) Write ($EF_{LLOG}$) R4: Response ($PP_{IEP} \Vert ID_{IEP} \Vert$ $ALG_{IEP} \Vert IK_{IEP} \Vert$ $DEXP_{IEP} \vert NT_{IEP} \Vert$ $BAL_{IEP} \Vert S_1 \Vert CC_{IEP}$)	← →	C4: INITIALIZE for Load Parameters ($PP_{PPSAM} \Vert$ $ID_{PPSAM} \Vert R \Vert M_{LDA} \Vert$ $CURR_{LDA}$)		

Figure 12.11 Procedure for loading an electronic purse (IEP) online via a terminal (LDA) using a security module (PPSAM)

		C5: DEBIT PPSAM Parameters (PP_{IEP} || ID_{IEP} || ALG_{IEP} || IK_{IEP} || $DEXP_{IEP}$ || NT_{IEP} || BAL_{IEP} || S_1)	→ ←	A4: Verify (PP_{IEP}) Verify (ID_{IEP}) Verify (ALG_{IEP}) Verify (IK_{IEP}) Verify ($DEXP_{IEP}$) $KD_{PPSAM} = f(ID_{IEP}, VK_{IEP}, KM_{PPSAM})$ $KSES_{PPSAM} = f(KD_{PPSAM}, DEXP_{IEP}, NT_{IEP})$ Verify (S_1) S_2 := Sign(PP_{PPSAM} || ID_{IEP} || NT_{IEP} || M_{LDA} || $CURR_{LDA}$) $BAL_{PPSAM} := BAL_{PPSAM} - M_{LDA}$ R5: Response (IK_{PPSAM} || S_2 || CC_{PPSAM}
A5: Verify (S_2) $BAL_{IEP} := BAL_{IEP} + M_{LDA}$ S_3 := Sign(PP_{IEP} || ID_{PPSAM}, R, CC_{IEP}) Write (EF_{LLOG}) R6: Response (S_3 || CC_{IEP})	← →	C6: CREDIT IEP Parameters (IK_{PPSAM} || S_2)		
		C7: PPSAM Load Acknowledgement Parameters (S_3, CC_{IEP})	→ ←	A6: Verify (S_3) Verify (CC_{IEP}) R7: Response (CC_{PPSAM})

Figure 12.11 (*Cont.*)

The procedure just described is one of many possible options. It is frequently used in practice, since it is very common to perform purse-loading transactions online. EN 1546 also includes options for loading via a special loading security module (LSAM). Such decentralized modules can be built into special loading terminals, such as cash dispensers.

Procedure for paying with a purse

The following example, which is illustrated in Figure 12.12, demonstrates a payment procedure using the components required for this function: the electronic purse (IEP), the terminal (PDA) and the security module in the terminal (PSAM).

After the purse card has been inserted in the terminal, the terminal executes a reset in order to request ATRs from the PSAM and the IEP. If either of these ATRs does not match the expected value, the terminal aborts the payment procedure. If the ATRs match their expected values, the terminal selects the purse DF in the IEP. If this file cannot be selected, the procedure is also aborted. For reasons of clarity, however, these processes and general error handling are not shown here.

After selecting the purse DF in the IEP card, the terminal sends the initialization command 'INITIALIZE IEP for Purchase'. The IEP receives this command, increments the transaction counter, computes a key ($KSES_{IEP}$) and generates a signature (S_1) for the various data elements. It then sends these data elements and the signature to the terminal.

The terminal next sends the initialization command 'INITIALIZE PSAM for Purchase' to the PSAM. This command simply relays the data elements received from the card to the PSAM. The PSAM verifies these data elements, which means that the expiry date ($DEXP_{IEP}$), currency ($CURR_{IEP}$), cryptographic algorithm used (ALG_{IEP}) and the other received data are compared with values stored in the PSAM. If all of the comparisons are successful, the transaction counter (NT_{PSAM}) is incremented. If any of the comparisons fails (e.g., if the expiry date of the IEP

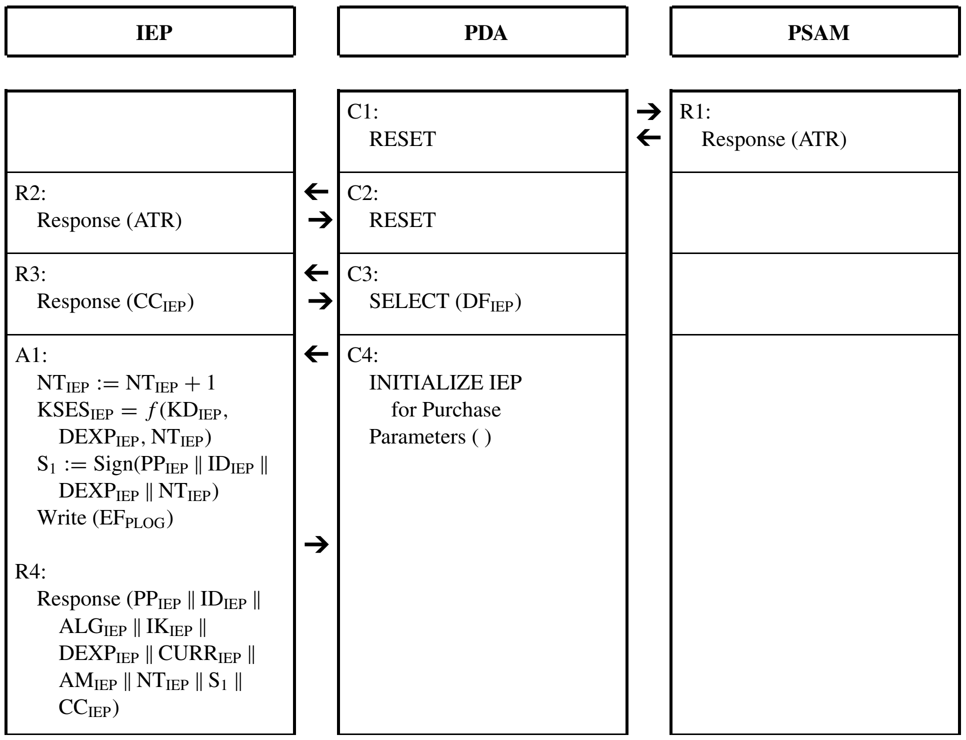

IEP
PDA
PSAM
C1: RESET
R1: Response (ATR)
R2: Response (ATR)
C2: RESET
R3: Response (CC_{IEP})
C3: SELECT (DF_{IEP})
C4: INITIALIZE IEP for Purchase Parameters ( )
A1:
$NT_{IEP} := NT_{IEP} + 1$
$KSES_{IEP} = f(KD_{IEP}, DEXP_{IEP}, NT_{IEP})$
$S_1 := Sign(PP_{IEP} \| ID_{IEP} \| DEXP_{IEP} \| NT_{IEP})$
Write (EF_{PLOG})
R4:
Response ($PP_{IEP} \| ID_{IEP} \| ALG_{IEP} \| IK_{IEP} \| DEXP_{IEP} \| CURR_{IEP} \| AM_{IEP} \| NT_{IEP} \| S_1 \| CC_{IEP}$)

Figure 12.12 Transaction procedure for making a payment using an electronic purse (IEP) with a terminal (PDA) and security module (PSAM) in accordance with EN 1546

		C5: INITIALIZE PSAM for Purchase Parameters (PP_{IEP} || ID_{IEP} || ALG_{IEP} || IK_{IEP} || $DEXP_{IEP}$ || $CURR_{IEP}$ || AM_{IEP} || NT_{IEP} || S_1)	→ ←	A2: Verify (PP_{IEP}) Verify (ID_{IEP}) Verify (ALG_{IEP}) Verify (IK_{IEP}) Verify ($DEXP_{IEP}$) Verify ($CURR_{IEP}$) Verify (AM_{IEP}) $NT_{PSAM} := NT_{PSAM} + 1$ $KD_{PSAM} = f(ID_{IEP},$ $VK_{IEP}, KM_{PSAM})$ $KSES_{PSAM} f(KD_{PSAM},$ $DEXP_{IEP}, NT_{IEP})$ Verify (S_1) $MTOT_{PSAM} := 0$ S_2 := Sign(PP_{PSAM} || ID_{PSAM} || NT_{PSAM} || $MTOT_{PSAM}$ || ID_{IEP} || AM_{IEP} || NT_{IEP}) R5: Response (ID_{PSAM} || NT_{PSAM} || IK_{PSAM} || S_2 || CC_{PSAM})
		A3: Input (M_{PDA} || $CURR_{PDA}$)		
A4: Verify ($CURR_{PDA}$) Verify ($BAL_{IEP} \geq M_{PDA}$) Verify (S_2) $MTOT_{IEP} := MTOT_{IEP} +$ M_{PDA} S_3 := Sign(PP_{IEP} || ID_{IEP} || AM_{IEP} || NT_{IEP} || ID_{PSAM} || NT_{PSAM} || $MTOT_{IEP}$ || $CURR_{IEP}$) $BAL_{IEP} := BAL_{IEP} - M_{PDA}$ Write (EF_{PLOG}) R6: Response (S_3 || CC_{IEP})	← →	C6: DEBIT IEP Parameters (ID_{PSAM} || NT_{PSAM} || IK_{PSAM} || S_2 || M_{PDA} || $CURR_{PDA}$)		

Figure 12.12 (*Cont.*)

	C7: CREDIT PSAM Parameters (S_3, M_{PDA})	→ ←	A5: Verify (S_3) $MTOT_{PSAM}$:= $MTOT_{PSAM}$ + M_{PDA} R7: Response (CC_{PSAM})
	C8: PSAM Complete Purchase Parameters ( )	→ ←	A6: TM := TM + $MTOT_{PSAM}$ S_4 := Sign(PP_{PSAM} || ID_{PSAM} || NC || NI(NC) || TM(NC) || CURR(NC) || CT(NC)) R8: Response (S_4, CC_{PSAM})

Figure 12.12 (*Cont.*)

has been reached), command processing is immediately terminated and an appropriate return code is sent to the terminal (PDA).

The PSAM next generates a derived key using the data provided by the IEP and generates a session key, and then it checks signature S_1. If the signature is correct, it follows that all of the transferred data are authentic, and the IEP has also been authenticated by the PSAM. In other words, the PSAM knows that the card containing the electronic purse is genuine.

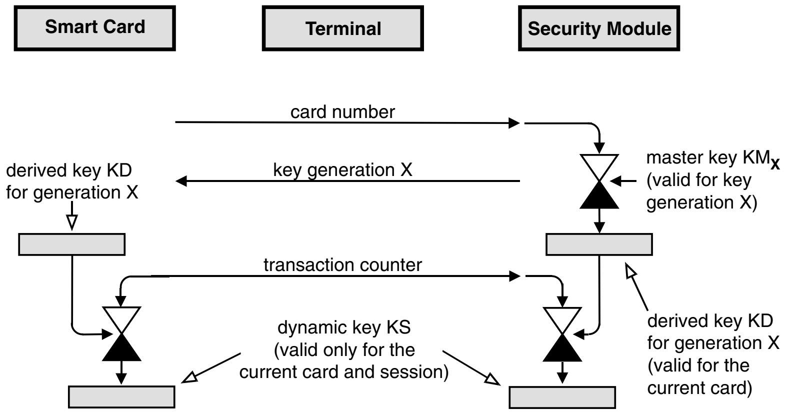

Smart Card
Terminal
Security Module
card number
derived key KD for generation X
key generation X
master key KM_X (valid for key generation X)
transaction counter
dynamic key KS (valid only for the current card and session)
derived key KD for generation X (valid for the current card)

Figure 12.13 A possible key derivation process for an EN 1546 electronic purse system. The key depends on the combination of a card-specific key from a certain generation that is passed to the card and a session-specific transaction counter. The key that is so generated may be used for making payments or debiting monetary units

Next, the PSAM also generates a signature (S_2), which is sent to the terminal along with several other data elements. The user now enters amount to be paid (M_{PDA}) and the associated currency ($CURR_{PDA}$) at the terminal. The terminal then sends the entered amount (M_{PDA}) and the data elements previously received from the PSAM to the card, using the DEBIT IEP command. The IEP now checks whether there is enough money in the purse to make the payment. If there is, it verifies signature S_2. If the signature is correct, the data have not been manipulated during transmission, and the PSAM has also been authenticated by the IEP, since only a genuine PSAM can possess the secret key needed to generate signature S_2. The appropriate amount is subtracted from the purse balance, a third signature (S_3) is generated to confirm the debit transaction just performed and the log file is updated.

Signature S_3 and the debited amount are sent via the terminal to the PSAM, which verifies S_3. If this signature is correct, the amount debited in the IEP is added to an internal data element ($MTOT_{PSAM}$). The following command, PSAM Complete Purchase, updates the PSAM balance by adding $MTOT_{PSAM}$ to the purse balance (TM). Finally, the PSAM receives a signature (S_4) to confirm that the payment transaction has been successfully completed.

The procedure described above is a very simple example of the various payment procedures described in EN 1546. There are also other possibilities, including an especially fast debiting procedure for card phones and a procedure that allows a receipt to be generated at the end of the transaction.

Files, commands and procedures are also specified for all other important system components, just as they are specified for the cards as described above. This primarily applies to the security modules, since system security relies solely on these modules. Statistical methods may be employed to monitor the overall operation of the system, which in the case of large applications may consist of tens of thousands of terminals and several hundred thousand smart cards. Maintaining full accounting for every individual card would conflict with the demand for anonymity, and would anyhow require far too much computation. However, as tests have shown, the security of the overall system can be continuously monitored at an acceptable cost using random samples.

The European EN 1546 standard established one of the foundations for multisector smart card electronic purse systems using smart cards. Nearly all procedures and functions that were in common use when the standard was generated and that were considered worthwhile are included in the standard. There is only one function that has not yet been described, although it is very important for card users. This is the 'purse-to-purse transaction', which means transferring electronic money directly from one purse to another. There is presently no description of this type of money transfer in the EN 1546 standard.

12.3.2 Common Electronic Purse Specifications (CEPS)

In the mid-1990s, many electronic purse systems based on smart cards were developed independently of each other in many European countries. Some typical examples are Quick in Austria, Geldkarte in Germany and Proton in Belgium and the Netherlands. All of these purse systems have similar functionality, but they are all mutually incompatible. The need to make these purse systems compatible with each has become increasing compelling, in part due to the introduction of a common European currency in 2002. Since all electronic purse cards are generally only valid for a period of three years, it is in principle possible to make gradual modifications to the purse systems over the course of several years using a migration path that is yet to be defined.

The fundamental prerequisite for achieving mutual compatibility among several electronic purse systems is a document specifying the features that the systems must have for compatibility. This document bears the name ‘Common Electronic Purse Specifications’ (CEPS), and the first version was published in 1999 by CEPSCO [CEPSCO]. In an earlier specification stage, the focus of CEPS was on an internationally interoperable electronic purse system, rather than one limited to European interests.

CEPS includes the standard functions for modern electronic purse systems, such as offline payment, online loading and online currency conversion. It is based on the European standard for electronic purses, EN 1546,[10] but it contains several extensions and modifications with respect to this standard. For instance, in contrast to EN 1546, RSQ-based certificates are used for authentication of terminals and smart cards. Triple DES is recommended as the cryptographic algorithm.

CEPS, like many electronic purse systems, is optimized for simple smart card microcontrollers. A typical implementation of CEPS in assembler or C requires 8 kB of ROM, 4 kB of EEPROM, 1 kB of RAM and a numeric coprocessor for the asymmetric cryptographic algorithm. In the future, most European electronic purse systems will be compatible with CEPS, so in the medium term it should be possible to make payments in various European countries using a single purse card.

12.3.3 Proton

Proton is an internationally used electronic purse system, which up to now has been developed almost exclusively by Bull, starting as early as 1995. It originates from Belgium and the Netherlands (where it is known under the brand name ‘Chipknip’), which is also where it is most widely used and presently has the status of a national electronic purse system. There are also relatively large purse systems based on Proton in Switzerland and Sweden (under the ‘Cash’ brand name). As of the spring of 2002, there were approximately 40 million cards issued internationally and around 360,000 terminals.

This electronic purse system was originally called CC 60, which is a name that originates from Bull. The current version is designated R3, and the next generation, which is already available in initial versions and has been strongly extended, is designated R4. R3 is a purse system that is optimized for inexpensive smart card microcontrollers, and it can be readily implemented in chips having 16 kB of ROM, 6 kB of EEPROM and 256 bytes of RAM. R4 is an extended version of R3 and is compatible with CEPS.[11]

Besides the actual purse system, the specifications for R4 define an operating system for multiapplication smart cards that includes debit and credit capabilities in accordance with the EMV specification,[12] as well as a digital signature application and Java functionality. There is also a contactless version of Proton, which is primarily intended to be used in the local public

[10] See also Section 12.3.1, ‘The EN 1546 standard’
[11] See also Section 12.3.2, ‘Common Electronic Purse Specifications (CEPS)’
[12] See also Section 12.4, ‘The EMV Application’

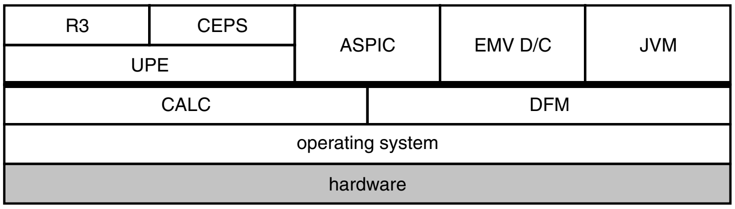


Figure 12.14 Schematic representation of the basic architecture of the Proton R4 smart card operating system. The following abbreviations are used: CALC (card application life cycle), DFM (data file management), UPE (universal purse engine), ASPIC (application for secure personal identification = PKI functionality), EMV D/C (EMV debit/credit application) and JVM (Java virtual machine)

transportation sector. The publisher of the specification, which is confidential, is Proton World [Proton].

The electronic purse system includes the usual functions, such as loading, individual payment, incremental payment ('sliced payment') and refunding a payment. The main use for incremental payment is public card phones, which require small amounts to be repeatedly debited from the purse balance at short intervals during a session. Transactions are stored in record-oriented log files. The purse parameters are also stored in files. In order to allow the system to be implemented using inexpensive microcontrollers, which have relatively little memory, it is allowed to place the files directly below the MF without a DF. Both DES and triple DES are used as cryptographic algorithms in R3. Many of the smart card commands used in the system are based on or compatible with the ISO/IEC 7816-4 and EN 1546 standards. They are supplemented by several application-specific commands. The actual purse function is related to the standard EN 1546[13] procedures in many aspects. However, it is readily apparent that Proton is several years older than EN 1546.

12.3.4 The Mondex system

There are presently several large payment systems in the world that use smart cards as a key component. Very few of these systems are based on an electronic purse, in which monetary units are stored directly in the card and not in a background system. Of these, there is only one system that can claim to allow electronic payments that correspond to payments using normal money. This is the Mondex system [Mondex].

The idea behind this concept, which is currently unique, was born in 1990. After five years of development, the first field trial was carried out in July 1995 in the southern English city of Swindon. A wide variety of shops were included in this trial, including newsstands, snack bars, supermarkets and travel agents, as well as filling stations and telephones. The maximum amount in the purse was set at £500 (approximately €550) for the trial, but this value can in principle be set to any desired level. Following this trial, which was widely reported in the press, there have been additional field trials in many different regions, but up to now the system has not been introduced in any country as a national system.

[13] See also Section 12.3.1, 'The CEN EN 1546 standard'

Mondex was a consortium of three firms: British Telecom, National Westminster Bank and HSBC. Its purpose was to create a means of payment that can be used like cash but does not have the disadvantages of cash. The result of this technical development was intended to be franchised to banks and other firms. After having had several intermediate owners, Mondex now belongs to the credit-card company MasterCard.

Mondex is one of the few electronic purse systems to be offered as a complete system, from the cards to the background system. Despite immense marketing expenditures, the high initial expectations with regard to widespread use of the system have failed to materialize. Currently, Mondex is an electronic purse application in the Multos smart card operating system,[14] and it is used in a few locations throughout the world.

The smart card operating system used for Mondex is not limited to electronic purse systems. It is a multifunctional, general-purpose system that can be used for multiple applications in a single smart card. This operating system is called Multos, and it is marketed internationally by Maosco [Maosco], primarily in the card-based payment systems sector. A special feature of Multos is that it supports downloading software to cards in the field. This software is written using a language similar to C, called Multos Executable Language (MEL), which is processed by an interpreter in the smart card.

The system

Since the Mondex purse is designed to behave in the same way as real money, purse-to-purse transactions are naturally possible. This allows cardholders to make payments among themselves without the intervention or knowledge of a bank or similar organization. The system is completely open and anonymous, and as many participants as desired can be involved. Figure 12.15 shows the system participants and the possible money flows.

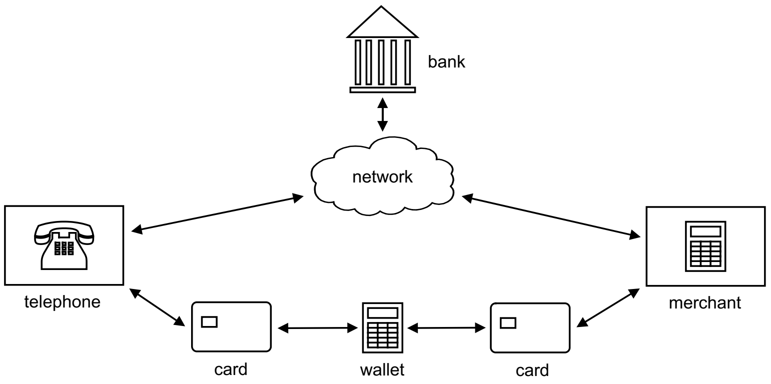


Figure 12.15 Possible money flows and participants in the Mondex electronic payment system

The electronic purse is located in the chip of a conventional ID-1 card with contacts. A matchbox-sized key fob with a display can be used to view the balance in the purse. If the card

[14] See also Section 5.14.2, 'Multos'

is inserted in this mini-terminal, the current purse balance and the last 10 transactions can be viewed. A 'wallet' is needed to transfer electronic money to the purse of another cardholder. This device, which resembles a pocket calculator, has a small keypad and display. It also has a built-in security module and a terminal for the electronic purse. To perform a purse-to-purse transaction, the user inserts the first smart card into the wallet and enters the amount to be transferred. This amount is then transferred from the electronic purse to the wallet's security module. The second card is then inserted into the wallet, and the amount is transferred to it from the security module. This completes the transaction.

Another device in this payment system is a telephone with a built-in terminal. It allows money to be transferred over the telephone line during a call. A typical application is ordering goods from a mail-order catalogue. In this case, payment can be made when the order is placed. Naturally, this technique can also be used to load the purse via the telephone, or to perform a transaction between two cardholders. If the card is loaded from a bank account, a four-figure PIN must of course be entered for security reasons, in order to protect the account holder against unauthorized withdrawals.

Each electronic purse can accept up to five different currencies. As soon as the balance for a particular currency reaches zero, a different currency can be loaded into the card. The purse can be blocked with a simple command and unblocked by entering a four-digit PIN, in order to prevent unauthorized use.

The merchant terminals contain security modules that use the same type of smart card as those used by customers. It would thus be possible to use such a security module to pay for other goods. Interestingly enough, this could make the theft of such a card worthwhile, as it could then be used just like a normal purse. However, this problem was recognized early on, and preventive measures were taken. Merchant cards can be configured to allow them to only receive electronic money, with debiting of the card only being possible during an online transaction with the merchant's bank. As can be seen, electronic money is not necessarily immune to theft. It all depends on whether it can be used by a thief. If a merchant terminal has online access to the bank (possibly via a dial-up link), it can be configured to automatically transfer money from the merchant card to the merchant's bank account whenever a particular balance is reached.

Security mechanisms and the payment procedure

All specifications related to transaction processes and the security model of the Mondex system are confidential. This makes it very difficult to obtain detailed technical information about the system and its individual components. We can therefore provide only a broad technical summary that illustrates some of the mechanisms and procedures used in the system.

The microcontroller that is used is a Renesas H8/3102. For mass production, a processor specially developed for Mondex is planned, with a numerical coprocessor and a suitable amount of memory, since the application requires around 5 kB in EEPROM. A symmetric cryptographic algorithm, such as DES, is probably used. As a special processor with a numerical coprocessor will be used in the future, it can be assumed that this will be replaced by an asymmetric algorithm for increased security. The RSA algorithm could be used, for example. In principle, though, the system is independent of the cryptographic algorithm used. It does not rely on special properties of a particular algorithm, but only uses (digital) signatures to protect data transmissions. In this regard, it differs little from multi-sector European electronic purse systems that are compliant with EN 1546.[15]

Since the Mondex system is operated in a completely decentralized manner, there must be a special procedure for switching key versions and algorithms. Each issued card contains at least two totally different cryptographic algorithms with several associated keys. If it is necessary to switch to another key version, or even to use a different algorithm, an appropriate parameter is set in all smart cards that make an online connection to the background system. These cards can in turn set the same parameter in all cards with which they conduct payment transactions. This snowball effect produces a system-wide switch to the new general parameters within a very short time, due to the exponential increase in the rate of data propagation. This would happen even if the background system only modified the parameter in a single card. This is a very effective, fast and simple method of changing global data in a decentralized payment system.

Naturally, it must be possible to isolate particular cards in the system. This can be done in three different ways. First, suspect cards identified by blacklists can be recognized and retained by the machine into which the card is inserted, although this is usually only possible with cash dispensers, since only they have the technical resources to retain cards. Second, the blacklists are loaded into all of the terminals, which can block cards so that they can no longer be used for transactions. Third, all issued electronic purse cards allow only a certain number of transactions to occur, after which they are automatically blocked. This block can be removed by an online query after the card has been checked against the blacklist, so the card does not have to be replaced. This ensures that a card with an electronic purse cannot be used indefinitely without any control by the background system.

A typical payment transaction between two smart cards in the Mondex system is divided into two stages, which are shown graphically in Figure 12.16. In the first stage, the current transaction is registered, which involves exchanging all of the data needed for the subsequent money transfer. This is followed by the second stage, in which the second smart card sends the desired amount to the first smart card. The complete data set is digitally signed, so it cannot be manipulated during the transfer. After receiving the data, smart card 1 checks the signature to verify both the authenticity of smart card 2 and the authenticity of the transferred data. If all of these verifications are successful, the desired amount is debited from smart card 1 and sent to smart card 2, together with a digital signature. Smart card 2 checks this signature to eliminate the possibility that the data have been manipulated, which also allows it to authenticate smart card 1. If all of these verifications are successful, the amount is credited to the purse. Following this, smart card 2 generates a confirmation that the amount was properly credited, adds a digital signature and sends this information to smart card 1. The transaction is completed when this confirmation of payment has been received and successfully verified.

Both cards contain log files, and they have suitable mechanisms to allow a transaction to be correctly resumed from the appropriate point if it is interrupted. These error recovery mechanisms are very important, since otherwise electronic money would be destroyed if a transaction were interrupted. Each of the participating cards has three separate log files for storing transaction-related data. The first is the transaction log, which stores various data related

[15] See also Section 12.3.1, 'The CEN EN 1546 standard'

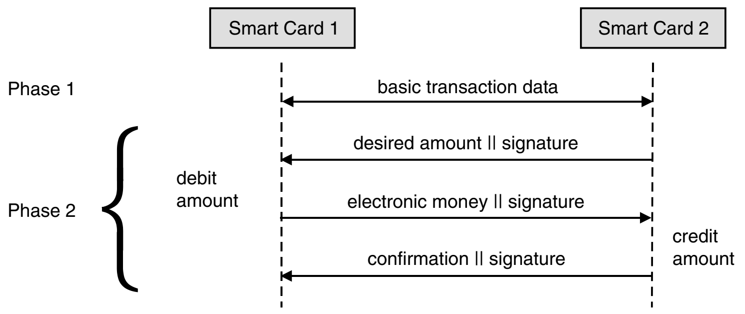


Figure 12.16 Information flow for a transaction between two smart cards in the Mondex system

to the 10 most recent successful transactions. The second is the pending log, which contains all of the data accumulated during a transaction that will be needed if error recovery becomes necessary. The third is the exception log, which stores all transactions that are not completed successfully. If all of the records in this file have been written, the smart card is automatically blocked. The cardholder must then unblock it via an online transaction, during which the log file entries are loaded into the background system and analyzed. After this, these entries are deleted.

Summary

The Mondex system is currently the only completely open electronic payment system using electronic purses. It supports all types of transactions that are possible with normal cash. In addition to this, it allows payments to be made via various telecommunications media, such as the telephone system. If the card containing the purse is lost, the money held in it is naturally also lost, just as with a real purse containing cash. However, this makes the system completely anonymous, which is sure to boost user acceptance. To a certain extent, the Mondex system is a simulation of a real money circuit. Since many central banks and government bodies have strong reservations with regard to direct card-to-card money transfers, a version of Mondex in which purse-to-purse transactions are blocked has also been developed. This yields a money circuit for the electronic purse system that is similar to that of EN 1546.

Since it is in principle impossible to demand a fee for each individual transaction with such a system, a question that quite quickly arises is how the system operator can generate any revenue. After all, the investments needed to establish and operate the system are not exactly trivial. In the Swindon field trial, each electronic purse user was charged a relatively low fee of £1.50 (≈€1.70) per month. The merchants naturally also paid fees. Although it would be possible to charge clearing fees for the merchants' turnovers, the completely open nature of the system naturally leaves merchants free to use their accumulated electronic money to make purchases from each other. The system operator could also generate revenue by offering various services to cardholders and merchants.

A major advantage of Mondex is that the clearing costs are nearly zero, since clearing in the usual sense is not necessary. Particularly with very low-value payments ('micropayments'), clearing costs in many systems can be very large relative to actual turnover. In some electronic purse systems, for example, the complete clearing costs, including transaction logging using shadow accounts, is around 5 eurocents per transaction. This means that the clearing cost for a pack of chewing gum bought from a vending machine (for 20 eurocents) is a hefty 25 % of the purchase price, which is totally unacceptable for the merchant. In the Mondex system, these costs would not arise.

In the coming years, the Mondex system will influence the market for electronic payment systems in many ways. At the international level, several large banks are considering the introduction of such a payment system. The possibility of making card-to-card transactions, which is viewed by many national banks as a security risk, can be disabled in the latest version of the system. This has strongly increased its level of acceptance. We can hardly wait to see what will develop.

12.4 THE EMV APPLICATION

Specifications and standards for using smart cards in a wide variety of application areas have been available for many years. However, there has traditionally been a strong concentration on telecommunications applications (telephone cards and GSM cards). This situation has changed markedly since the mid-1990s. The European electronic purse standard (EN 1546) is a good example of this trend, but the most important specification in the field of payment systems is the EMV specification. This specification, which is named after its three initiators (Europay, MasterCard and Visa), contains detailed descriptions of all aspects of credit cards containing microcontroller chips. A corresponding specification for matching terminals is also now available.

In the autumn of 1993, the three internationally active credit card companies Europay, MasterCard and Visa started work on a specification titled 'IC Card Specifications for Payment Systems'. Version 1 was published relatively soon afterwards, in October 1994. In mid-1995, a revised specification (Version 2) was completed. The final version of this specification, called 'EMV '96', was released at the end of June 1996. It is backward compatible with Version 2. After ambiguities had been cleared up and small errors corrected, a completely revised and compatible version of the EMV specification (Version 3) was published in the summer of 1998. At the end of 2002, the version number was increased to 4 with a few modifications. This is the presently valid version, and its official name is 'EMV 2000'. It is also available via the Internet [EMV], and it can be recommended without reservation to interested parties as a worthwhile subject of study.

Several factors motivated these credit card issuers to prepare a specification for credit cards with chips within such a short time. First, existing credit cards with magnetic stripes can be very easily forged. Nowadays, the only real obstacle is the hologram, which is still moderately secure against forgery. All other card features can be copied relatively easily. The second important factor is the value-added services that a microcontroller card can offer. Electronic purses, bonus points and telephone functions are only some of the possibilities.

The specification for credit cards with chips is divided into four parts, which are called 'books'. Book 1, *Application-independent ICC to terminal interface requirements,* draws heavily on the ISO/IEC 7816-1/2/3 family of standards. It describes the electromechanical characteristics, logical interface and data transmission protocols, which are the application-independent parameters. According to this part of the specification, the smart card has an ID-1 format[16] with ISO contact locations. It must have a supply voltage of 5 V $\pm$ 0.5 V, a maximum current consumption of 50 mA and a clock rate of 1–5 MHz, among other things. The contact force must not exceed 0.6 N per contact. Data transmission at the physical level is essentially identical to ISO/IEC 7816-3. This applies to the time interval for individual bits,[17] the ATR[18] and the two transmission protocols[19] ($T = 0$ and $T = 1$).[20] The specification of the APDU[21] is identical to that in the ISO/IEC 7816-4 standard.

The second part of the specification (Book 2, *Security and key management*) defines the security mechanisms and key management. Book 3, *Application specification*, contains the data elements, commands and processes needed for debit and credit transactions in accordance with the EMV specification. Book 4, *Cardholder, attendant, and acquirer interface requirements*, contains many general specifications for EMV-compliant payment systems that are related to users and merchants.

Since typical credit cards are mass-produced articles, their manufacturing costs must naturally not be too high. As these costs predominantly depend on the embedded microcontroller, the credit card application was designed from the start to use a minimum amount of memory. A conventional EMV application without supplementary functions thus fits into a processor with 6 kB of ROM, 1 kB of EEPROM and 128 bytes of RAM. Even in the smart card area, these values represent the lower end of the range of available chips, but they do make for an inexpensive product.

To a certain extent, the EMV specifications are basic documents that specify the minimum requirements for the various card issuers. The current version leaves a number of issues open. For example, risk management of the terminal and the smart card during transactions is not yet precisely specified. Consequently, the EMV application is only described here in outline, since many details must be specified by the card issuer.

Files and data elements

The specification for credit cards with chips only states that a tree structure must be used for the files.[22] The actual application is located in its own DF, which is selected using an application identifier (AID) and which contains all of the data elements for the credit card application. These data elements are stored in EFs with standard file structures in accordance with ISO/IEC 7816-4. EFs can normally be selected implicitly using short FIDs. The main difference with respect to similar specifications is that no particular EFs or FIDs are specified. However, this is not necessary for the functions used for payments, since all of the necessary data can be processed using existing commands. There is only one file directly below the MF (EF_{DIR} per

[16] See also Section 3.1.1, 'Card formats'
[17] See also Section 6.1, 'The Physical Transmission Layer'
[18] See also Section 6.2, 'Answer to Reset'
[19] See also Section 6.4, 'Data Transmission Protocols'.
[20] See also Section 6.5, 'Message Structure: APDUs'
[21] See also Section 6.5, 'Message Structure: APDUs'
[22] See also Section 5.6, 'Smart Card Files'

ISO/IEC 7816-5), which contains all of the information related to the applications present in the card.

All data elements in the terminal–card system are specified using unambiguous templates and tags. They can be addressed within the application using the specified commands, without any knowledge of their precise locations in the file tree or a particular file. This makes it possible to leave the definition of the file structure to the card issuer, since it does not affect the execution of the transactions.

Commands

Strictly speaking, only three commands are necessary for performing the actual payment functions. Additional commands are needed for personalization, management, special functions and value-added services, but they fall outside the scope of the EMV specification. According to the requirements of the EMV specification, the following commands must be available, with all return codes being similar to ISO/IEC 7816-4: [23]

- APPLICATION BLOCK (specific to EMV)
- APPLICATION UNBLOCK (specific to EMV)
- CARD BLOCK (specific to EMV)
- EXTERNAL AUTHENTICATE (as a subset of ISO/IEC 7816-4)
- GENERATE APPLICATION CRYPTOGRAM (specific to EMV)
- GET CHALLENGE (ISO/IEC 7816-4)
- GET DATA (specific to EMV)
- GET PROCESSING OPTIONS (specific to EMV)
- PIN CHANGE/UNBLOCK (specific to EMV)
- READ RECORD (as a subset of ISO/IEC 7816-4)
- SELECT (as a subset of ISO/IEC 7816-4)
- VERIFY (as a subset of ISO/IEC 7816-4)

Cryptographic mechanisms

The cryptographic mechanisms used in an application are naturally highly dependent on the associated general requirements. This can be seen especially well in the EMV application. A basic initial premise for the system design was that terminals do not necessarily contain security modules, depending on the system operator. This makes it impossible to use symmetric cryptographic algorithms, since the keys cannot be kept secret. The reasons for not using

[23] See also Section 7.10, 'Commands for Credit and Debit Cards'

security modules are that they significantly increase the cost of the terminal, and that an international system for managing keys in terminals, some of which operate offline, would be very complicated and expensive.

In order to make smart card authentication by the terminal possible under these conditions, an asymmetric cryptographic algorithm must be used. EMV uses static unilateral authentication with card-specific keys.[24] This does not allow the card to check the authenticity of the terminal, but this is not essential in the EMV application, since debiting is not performed in the card. The card only generates a transaction certificate for the terminal. This transaction certificate is not anonymous with respect to the cardholder, and it can be submitted to the relevant card issuer only by an authorized (known) merchant. This largely excludes most possible forms of fraud, since a valid transaction certificate can be 'converted' into money only by an authorized merchant known to the card issuer.

In principle, any desired cryptographic algorithm can be used, since both the associated data elements and the algorithm itself are unambiguously identified by TLV-coded data structures. Version 2 of the EMV specification allows either the SHA-1 (as per FIPS 180-1) or the ANSI X9.30-2 hash function to be used.[25] The cryptographic algorithm used is not DSS, as might be expected from the use of SHA-1, but instead the RSA algorithm[26] (ANSI X9.31-1). The length of the key can vary, depending on the card issuer, and it is indicated by an appropriate code (signature tag). Small numbers, such as 3 or $2^{16} + 1$, are recommended for the public key in order to minimize computation time.

If the smart card has established an online link to the background system, it is possible to protect the data transmission by secure messaging[27] as specified in ISO/IEC 7816-4. A symmetrical cryptographic algorithm is used for this end-to-end communication, namely triple DES. In this case, it is possible to do so without compromising system security, since both the background system and the card can securely store the secret key.

System architecture and transaction processes

Traditionally, a highly centralized system architecture is used for payment systems in the credit card sector. There are usually several background systems, which are either individually or collectively responsible for a certain region (such as Germany). The computer centers for the background systems, which are equipped with high-performance computers, are interconnected by the independent network of the card issuer. This network supports data exchanges for clearing and increases operational reliability, since if one center fails, its activities can be taken over by other centers. Individual terminals are connected to the background system via the public telephone system and data networks, such as ISDN and X.25. An acquirer, who routes and bundles transaction data, may be located between the terminals and the background systems. However, this strongly depends on the particular card issuer and country. At the terminal level, there are two different options: data may be exchanged directly with the acquirer,

[24] See also Section 4.11.3, 'Static asymmetric authentication'
[25] See also Section 4.9, 'Hash Functions'
[26] See also Section 4.7.2, 'Asymmetric cryptographic algorithms'
[27] See also Section 6.6, 'Securing Data Transmission'

or a concentrator belonging to a merchant or chain of shops may be used. Both of these options are possible, and both are used in practice.

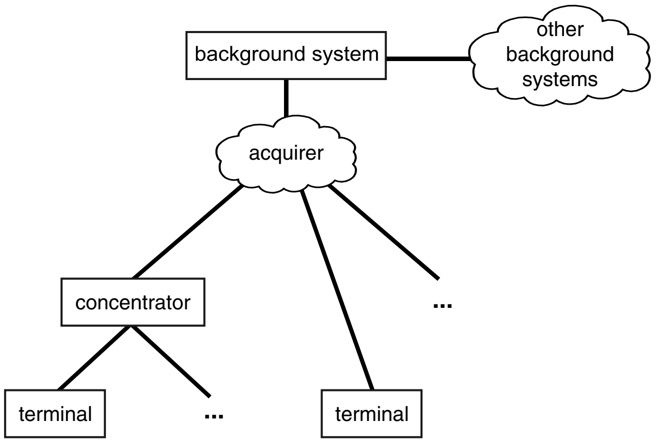


Figure 12.17 Basic architecture of the payment system for EMV cards

The process of making a payment with a ‘smart’ credit card is not much different from making a payment using a traditional credit card. The customer presents his or her card to the cashier, and it is inserted into a smart card terminal. If a terminal is not available, payments can be made in the conventional manner using the magnetic stripe or embossed characters, which are still present on the card. However, even if a terminal is present, it is still possible to verify the identity of the card user by means of a signature. In this case, the associated transaction receipt has a marker that indicates that the card user has been identified in this manner. The other option is for the card user to enter a four-digit PIN. This can be checked online by the background system or offline by the smart card. If a PIN is used for identification, the transaction certificate indicates which type of PIN check was performed. The payment process is shown graphically in Figure 12.18.

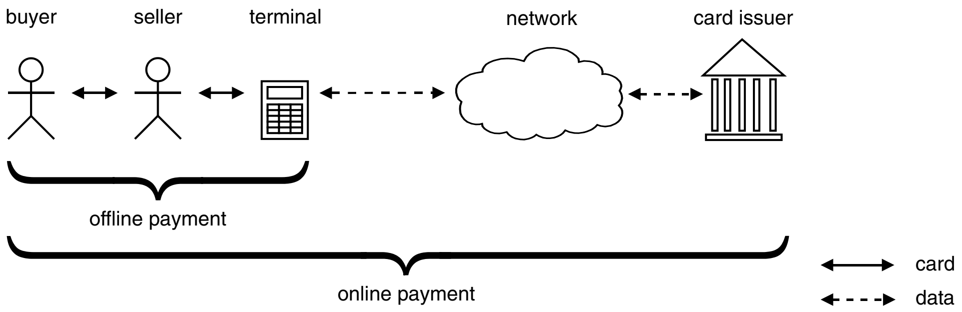


Figure 12.18 Basic infrastructure of a payment process using a smart card, according to the EMV specification

At the detail level, the individual functions and the process of a successful payment transaction using a smart card are naturally somewhat more complicated. An example is shown in Figure 12.19, which illustrates the fundamental mechanisms. Both the card and the terminal determine the exact course of the transaction based on various transaction data, such as the amount involved. For instance, if the amount to be paid exceeds a certain sum, the card requests online authorization of the payment by the background system. The terminal must then establish a link to the background system and have the payment approved. Only after the background system has approved the request does the card generate a valid transaction certificate, which the merchant can submit for clearing. The purpose of online authorization is to minimize the financial risk to the card issuer. Since a 'smart' credit card must regularly report to the background system in accordance with a number of conditions, the maximum loss in the event that a card is stolen or manipulated can be held within precisely calculable limits.

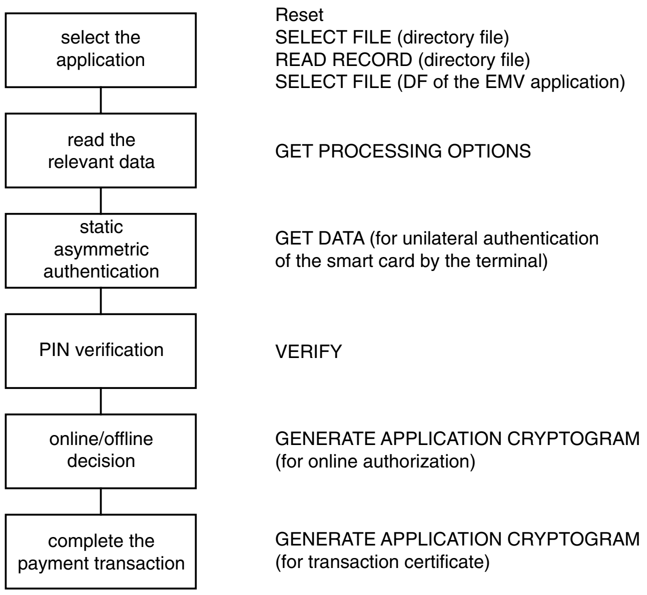

select the application
Reset
SELECT FILE (directory file)
READ RECORD (directory file)
SELECT FILE (DF of the EMV application)
read the relevant data
GET PROCESSING OPTIONS
static asymmetric authentication
GET DATA (for unilateral authentication of the smart card by the terminal)
PIN verification
VERIFY
online/offline decision
GENERATE APPLICATION CRYPTOGRAM (for online authorization)
complete the payment transaction
GENERATE APPLICATION CRYPTOGRAM (for transaction certificate)

Figure 12.19 A high simplified portrayal of the payment process with a smart card according to the EMV specification, as seen by the terminal

Future developments

In terms of its structure and contents, the EMV specification allows the card issuer considerable room for individual initiative, which makes further developments and individual versions possible. This flexibility will doubtless be utilized extensively by various firms. For instance, each of the three credit card issuers involved in drawing up the EMV specification has generated a specification for an electronic purse [28] that can also be included in the smart card as

[28] See also Section 12.3.1, 'The CEN EN 1546 standard'

necessary. Particularly with vending machines or small purchases, a prepaid electronic purse definitely has advantages, since the fee that is otherwise charged for a credit card payment is not applicable.

In the near future, the increasing commercial use of international networks, such as the Internet, will require secure means of payment that are internationally available and widely accepted. Credit cards with chips and a few value-added services would be eminently suitable for this, since they have already achieved international acceptance and widespread use, independent of any particular country or currency. Work is presently being carried out in this area. The difference between this and the payment process described above is not great, since the Internet or the like could take the place of the acquirer. In principle, all that is necessary is that the customer has access to a smart card terminal.

The EMV specification has achieved the same significance in the financial transactions field as GSM 11.11 has for smart cards used in telecommunications applications. There are no modern smart card operating systems on the market that do not claim to be EMV-compatible. Due to the number of credit cards with chips that can be expected to be in circulation in the future, this specification represents the minimum standard for all future applications.

12.5 THE EUROCHEQUE SYSTEM IN GERMANY

Germany is different from other countries with regard to card-based payment systems, in that traditionally debit cards (Eurocheque cards) have been used much more than credit cards. This type of card can be used in many places to make payments after the user has entered a four-digit PIN. The amount to be paid is immediately deducted from an account associated with the card. The merchant must pay a fixed fee for each payment transaction, but it is not particularly high. With credit cards, the fee is a percentage of the revenue, and for this reason credit cards have achieved only moderate acceptance in Germany. There is also a widely used option called *POS ohne Zahlungsgarantie* (POZ), in which the customer consents to have the amount of the purchase transferred from his bank account to that of the merchant via direct debit. In this case, the Eurocheque card serves only to provide a reference to the customer's bank account, which is checked online to see whether the balance is sufficient to cover the amount of the purchase. However, in this case the merchant does not receive a payment guarantee, as he would with a credit card transaction or a normal Eurocheque card transaction (ec-Cash or Geldkarte).

Since Eurocheque transactions usually have to be authorized online by a background system, the merchant must also pay the costs of the individual data transmissions or the rental for a leased line. Since this is only worthwhile for the merchant if there is a large sales volume and many purchases are made using Eurocheque cards, improvements to this system have been sought for some time. The acceptance of Eurocheque cards among merchants would increase dramatically if the high telecommunications costs could be eliminated. This means that a system that can work offline is needed.

In 1993, the *Zentraler Kreditausschuss* (ZKA), which is a working group of the national associations of the German banking industry, issued a call for tenders for the design of a multifunctional chipcard (MFC)[29] that would be suitable for electronic payment systems.

[29] The term 'multifunctional chip card' was coined at this time and in this context

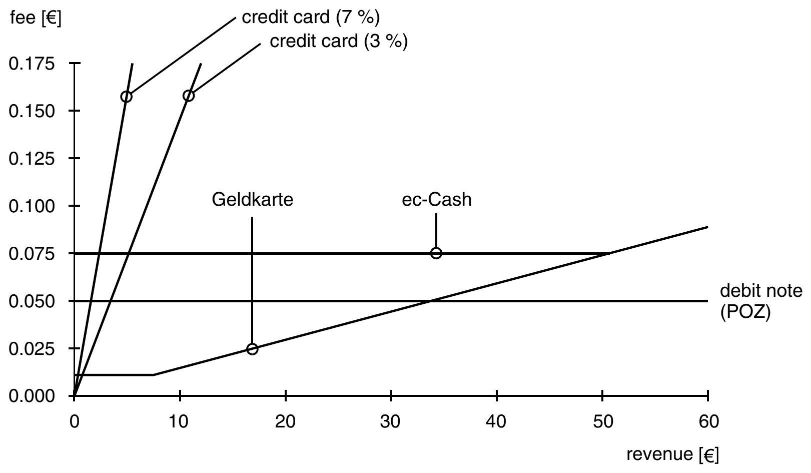


Figure 12.20 Merchant fees for electronic payments in Germany. With the debit note method (POZ), the fee is eurocents per transaction, independent of the revenue. With this method, the customer signs a debit note authorization, which allows the merchant's bank to debit the amount of the purchase from the customer's account. When a purchase is made using a Geldkarte, the merchant must pay at least 0.3 % of the purchase amount, with a minimum fee of 1 eurocent. If the customer uses ec-Cash, the merchant must pay at least 0.3 % of the received revenue, with a minimum fee of 7.5 eurocents. With a credit card, the merchant must pay between 3 % and 7 % of the revenue, depending on the contract, possibly with a contractual minimum revenue

Several firms offered solutions corresponding to the requested functionality, and one of them was selected and awarded the rights to the design by the ZKA. Due to changes in the general technical requirements, this design was then extensively revised. As a result of these revisions, there is now a family of specifications for Eurocheque cards with chips, each of which addresses a particular area. Unfortunately, these specifications are confidential, so it is not possible to publish detailed information. However, we can present brief summaries of the various areas, which are:

- the SECCOS operating system
- Geldkarte
- ec-Cash with chips
- digital signatures
- EMV
- electronic driver's license
- electronic marketplace
- personalization

These documents describe a payment card whose functionality corresponds to that of the current Eurocheque card and which also contains an electronic purse. It is also possible to load any desired supplementary applications into the card after it has been issued.

Prior to the nationwide introduction of the new card, a large-scale field trial was conducted in a region around Ravensburg and Weingarten (near Lake Constance), which has a trading area with 250,000 inhabitants. The trial involved around 100,000 Eurocheque cards and approximately 500 terminals. Following this, wide-scale introduction of the card throughout Germany started in the fall of 1996. Up to now, around 50 million Eurocheque cards with chips have been issued in Germany, all of which must be replaced every three years. At the end of 1998, 250,000 terminals had been installed in Germany, with 300 million transactions taking place each year with these terminals. In 1999, the combined turnover for magnetic-stripe and chip-based transactions using ec-Cash and Geldkarte amounted to approximately 19 billion euros. Nearly all of the 41,000 automated cash dispensers in Germany are equipped with smart card terminals. There are presently around 30,000 loading terminals for Geldkarte in Germany. This field trial thus proved to be the precursor to one of the largest smart card payment systems in the world.

In 2001, a statistical survey of the Geldkarte electronic purse system was conducted. The figures from the survey give a clear picture of the current usage of the system. Of the 50 million cards that have been issued with the Geldkarte application, only 4 million are actively used, resulting in approximately 14 million transactions in half a year. The average load amount was 30 euros, and the average payment amount was 2.30 euros.

User functions

The German Eurocheque card with a chip has a variety of user functions. It can be used to make online or offline payments at a suitable terminal after entering a PIN code. The amount to be paid is then deducted from the associated account by the bank that issued the card. This application is referred to in this chapter as 'ec-Cash', although there are also other designations for it.

Naturally, it is also possible to use a Eurocheque card with a chip to obtain cash from the cash dispensers of various banks, but in functional terms, this actually belongs to the realm of ec-Cash. It goes without saying that the smart card also supports a variety of lobby-machine functions, such as printing an account statement. The card also contains a prepaid electronic purse called Geldkarte, which can be used to make payments without entering a PIN code. This purse is available in anonymous and non-anonymous versions. It can be repeatedly reloaded using suitable loading terminals (located at bank counters and cash dispensers), either against a cash payment or via an ec-Cash transaction.

One of the capabilities of the Eurocheque card is downloading additional applications, with all of their associated files and commands, after the card has been issued to the cardholder. However, this capability has not been used very much up to now, since the coordination requirements and conditions for a new application are relatively complex.

The overall system in brief

As is usual with large payment systems, there is no central clearing system for German Eurocheque smart cards. There are four computer centers for processing settlements among the accounts of the merchants, cardholders and participating financial institutions. Figure 12.21 shows an overview of the clearing process for the non-anonymous version of the electronic purse application. The clearing body in Germany is called the *Börsenevidenzzentrale* (BEZ). There are also approximately 25 loading centers that perform loading operations for Geldkarte cards in coordination with the BEZ.

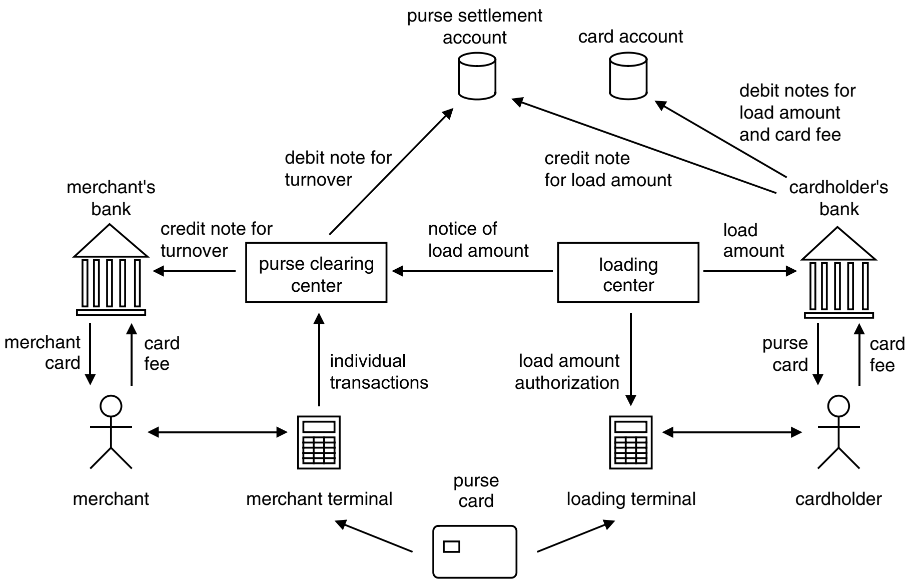


Figure 12.21 Basic architecture of the payment system for the German Geldkarte, which is an account-linked electronic purse system and thus not anonymous

All transactions are based on two accounts: the purse settlement account, which always reflects the current balance of the electronic purse in the card, and the card account, which for example may be the cardholder's current account. The purse settlement account is thus a shadow account, which is maintained in parallel with the electronic purse. If an amount of money is loaded into the electronic purse, a corresponding booking is made to the purse settlement account at the same time, since this transaction must always be performed online. Depending on whether the payment transaction occurs online or offline, the amount of the payment is booked against the purse settlement account either at the same time or at a later time. The main advantage of this account-linked system is that the purse balance can be reconstructed after a certain amount of time if the electronic purse is lost or becomes unusable. Of course, it is not possible to ensure that payments are anonymous with this approach. However, there is also an anonymous version of the electronic purse that uses a shadow account, with no reference to a customer account.

Eurocheque smart cards

Several different types of cards are used in the German Eurocheque system. However, smart cards for normal bank customers can be classified into several categories, as follows:

- Eurocheque cards (ec-Cash and Geldkarte, account-linked and thus not anonymous)
- Geldkarte linked to an account (non-anonymous)
- Geldkarte not linked to an account (anonymous)

There are two DFs in a Eurocheque card, one of which holds the ec-Cash application and the other the Geldkarte[30] application. If the card contains an electronic purse application, the DF for ec-Cash is not present. The account reference, which determines whether the card is anonymous, is provided using only certain data elements.

For merchants, there is merchant card in ID-000 format (plug-in) for use in their terminals. It contains all of the commands and files needed to conduct payment transactions. This card can be regarded as the security module of the terminal. One of the unusual and technically interesting features of this system is that it has both real and virtual merchant cards. A real merchant card is a normal smart card in plug-in format. A virtual merchant card is simply a software simulation of a real merchant card, which runs in the protected environment of a security module (SAM) in a merchant terminal.

This solution was originally a compromise to allow terminals without sockets for plug-in cards to be used in the new system. In the meantime, it has turned out to have some very positive technical features. For instance, a virtual merchant card can easily be replaced via remote maintenance, since it consists only of software. In addition, its useful life is significantly longer than that of a real card, since it is not subject to the detrimental effects of a limited number of EEPROM write cycles. Finally, a good hardware security module is at least as secure as a smart card, since its security mechanisms are always active, thanks to its built-in power buffer.

The entire informatics concept and the security module of the card are strongly based on the ISO/IEC 7816 family of standards. The original version, which is now called Type-1 Geldkarte, included a few application-specific mechanisms, but they have been eliminated in more recent versions. A complete smart card operating system with PKI functions has now been specified. It is called Security Card Operating System (SECCOS), and it supports all of the essential mechanisms of the ISO/IEC 7816 standards. The security and access mechanisms of ISO/IEC 7816-9 have also been included in a very elaborate form, with the result that as of now, the German Eurocheque smart card probably represents the most complete implementation of this standard in the world. For reasons of compatibility, elements of the EMV specification for credit cards also contributed to the specification of the Eurocheque card.

In terms of the general technical parameters prescribed by the specification, the card is based on many previously existing standards. Naturally, its dimensions match those of the ID-1 format and are thus the same as the present Eurocheque card. In addition, it is constructed

[30] Unfortunately, the term ‘Geldkarte’ is ambiguous, so it must always be understood in context. It can refer to either an electronic purse as an application in a smart card, or the smart card itself

as a hybrid card, with both a chip and a magnetic stripe, in order to avoid compatibility problems during the transition from terminals with magnetic-stripe readers to new terminals with smart-card contact units. The card uses the T = 0 transmission protocol with PPS. The triple-DES algorithm is used for the cryptographic processes. One of the interesting security aspects is that the entire software and hardware of the smart card must be certified in accordance with the ZKA criteria catalog.[31]

The file management system supports several levels of DFs, as well as file selection using short FIDs, FIDs and AIDs. The usual file structures (transparent, linear fixed and cyclic) are supported, and they can be implicitly selected in the appropriate commands. The maximum size of the two record-oriented file structures is 254 records of 255 bytes each. Naturally, no presently existing application fully exploits this maximum size.

The file management system uses a special mechanism to assign EFs to specific applications. This function, which is implemented using two non-standard commands, allows EFs to be assigned to applications across DF boundaries. This makes it possible to use a short FID within a particular DF to select an EF located in a different DF. Consequently, a particular EF can be assigned to several different DFs. This corresponds in principle to the alias mechanism used in many PC operating systems. The objective is to make EFs containing general information available to several applications across application boundaries without using complicated selection procedures. An EF assigned to several applications in this manner can be selected using a short FID or a FID, and then read or written after the necessary security state has been attained. All files have specific access conditions, which makes reading and writing dependent on previously attained states (such as PIN entry). With this object-oriented system, it is also possible to make access to files depend on secure data transmission. This means that there are file attributes that can compel the use of secure messaging for any access.

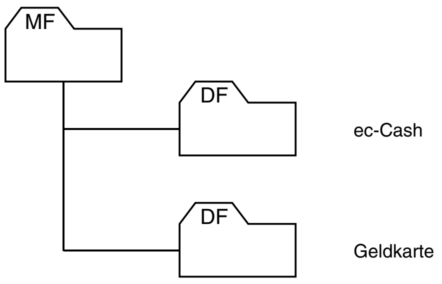


Figure 12.22 Basic file tree of a German Eurocheque smart card containing both the ec-Cash and Geldkarte applications

The available commands can be divided into four classes. The first class consists of commands that are compliant with ISO/IEC 7816-4, although they have reduced functional scope compared with the standard. The second class consists of Eurocheque-specific administration commands, which are used for management purposes in the card. They can be used to

[31] See also Section 9.3.1, 'Evaluation'

create new files, delete existing files and enter new commands into the card. The third class is extension commands, which are used to achieve the functionality needed for the ecCash and Geldkarte applications. The administration and extension commands are purely specific to Eurocheque cards, and in principle they have no connection to any international standards. The fourth class consists of the initialization and personalization commands.

As can be seen from this brief description, the Eurocheque card has a relatively large range of functions. This unavoidably results in a large memory requirement. Consequently, the presently used target hardware predominantly consists of microcontrollers with 100 kB of ROM, 32 kB of EEPROM and 2 kB of RAM. A pure electronic purse card without the ec-Cash functions needs only roughly 48 kB of EEPROM, 16 kB of EEPROM and 1 kB of RAM. Altogether, such memory requirements mean that relatively large chips must be used to hold the extensive amount of program code, along with the 2.3 kB of application data for the Geldkarte electronic purse and 1.6 kB for ec-Cash.

Value-added services

The operating system of the Eurocheque card includes commands and mechanisms for downloading executable program code. However, this code must be tailored to each type of microcontroller and operating system being used, since only machine code (which is address dependent) can be downloaded. The resulting amount of logistical overhead for downloading new commands is the main reason why this mechanism is presently not used.

However, value-added services do not necessarily require loading programs into the smart card. In most cases, it is sufficient to have files available that have suitable access privileges. The Eurocheque card specification includes commands for creating files. However, the administrative overhead for implementing supplementary applications for individual cards via a clearing center is very large, so this mechanism is also very seldom used. Instead, several files are stored in the Eurocheque cards when they are personalized, in order to provide space for storing new applications some time after the cards have been issued. The savings bank association has given the name ‘Space Manager’ to this technique for managing files for supplementary applications.

Summary

The German Eurocheque card system is presently one of the largest and most complex payment systems using smart cards. This applies not only to the transaction procedures, but also to the logistics of chip fabrication, card personalization and card distribution. After all, every three years approximately 30 million cards must find their way into customers’ hands within less than three months.

The security evaluations of the microcontroller hardware, operating system software and application software have also set new standards, since the acceptance criteria are severe and are constantly adapted to new circumstances (such as DFA, DPA and the like). Another interesting aspect is the technically sophisticated compatibility tests, which must ensure that software produced using a variety of masks on a variety of microcontrollers works smoothly with a wide variety of terminals.

The original figures of more than 20 different masks used with more than 10 different microcontrollers have now been reduced to only six masks and three microcontrollers, and in all likelihood only two mask makers will survive. This is a clear sign of the tendency toward consolidation exhibited by all large systems.

Field trials for Geldkarte have been conducted or are planned in France, Luxembourg and Iceland. An additional prospect for this system is shown by several pilot experiments in which Geldkarte is being used as a payment medium in the German portion of the Internet. All that the customer needs is a simple, inexpensive terminal connected to his home PC, along with related software. The merchant's counterpart is a security module or a special terminal connected to the customer's PC via the Internet.

The fact that more than 50 million smart cards are in use in the German Eurocheque system has an effect on all payment system projects based on smart cards. The experience gained from using this multiapplication smart card in Germany will provide the stimulus for considerable further refinement of smart card operating systems and related microcontroller hardware.

13
Smart Cards in Telecommunications

Telecommunications, or communications technology, is the technology used to exchange messages over arbitrary distances between persons and/or machines. Over the centuries, it has developed from human messengers and visual signaling techniques to communications using wire-bound and wireless electrical signals. The pioneers in this field were primarily military organizations and businessmen, for whom it was a matter of survival to be able to transmit messages as quickly as possible. Nevertheless, the biggest impetus to the development of telecommunications has come only in recent years, during which it has become a true mass application as the result of various favorable conditions, such as deregulation of the market, inexpensive manufacturing of terminal equipment and a general economic and technical boom.

Although the first proposals for using electricity to convey messages were published as early as 1753, the first telegraph line between Paris and Lille did not go into service until 1794. In the following decades, inventions related to conveying messages using electricity were made by numerous persons in many countries. The most important and most interesting of these developments were the construction of the electrochemical telegraph by S. Soemmerring in 1809, the invention of induction telegraphy by Carl Friedrich Gauß and Wilhelm Weber in 1833 and the construction of the recording telegraph by Samuel Morse in 1835 (along with the publication of the Morse code).

However, all of these inventions and constructions, many of which were quite sophisticated, could only transmit coded electrical signals, rather than the human voice. The latter first became possible around 1860. As so often with major inventions, it is not possible to credit a single person with the invention of the telephone. Still, there were essentially two persons who publicly demonstrated devices that could transmit speech[1] using electrical signals: Johann Philip Reis, a teacher at a school for the deaf, and Alexander Graham Bell, a teacher of the deaf and dumb. Both of them originally conceived their inventions as teaching aids for helping deaf persons learn to speak. The major strength of Alexander Graham Bell was his aggressive marketing of

[1] According to an anecdote that is now impossible to verify, the first words that Johann Philip Reis transmitted by telephone in 1863 were 'Das Pferd frisst keinen Gurkensalat' ['Horses do not eat cucumber salad']. He presumably chose this somewhat remarkable sentence, rather than something that the person at the other end could readily guess, in order to be sure that the transmission of human speech actually worked. Something like the presently common 'Hello world' could have easily been guessed by the listener

Smart Card Handbook, Third Edition. W. Rankl and W. Effing
© 2004 John Wiley & Sons, Ltd ISBN: 0-470-85668-8

his invention, with the result that he came to be the better known of the two inventors of the telephone.

The first local telephone network was constructed in 1878 in New Haven in the USA. In Berlin, the first local public telephone network was put into operation in 1881. It had human operators, and naturally it had all the characteristics of the simple electronic technology of the time. However, the first automatic telephone exchange in Germany, using rotary lifting selector switches, was installed as early as 1908 in Hildesheim. The worldwide installation and interconnection of what is known in technical terms as the ‘public switched telephone network’ (PSTN) began at this date.

After Guglielmo Marconi succeeded in making the first wireless transmission of data over a distance of several kilometers in 1896, it took only five years for the Atlantic Ocean to be bridged by wireless telegraphy. The invention of amplifier tubes (valves) stimulated the technical development of transmitters and receivers for voice communications, with the result that the first successful transmission of speech over the Atlantic Ocean took place in 1915. At the 1929 radio exhibition in Berlin, the first ‘picture phone’ was presented. It had the dimensions of three contemporary telephone booths.[2]

In the following years, the technical development of telecommunications and mobile radios accelerated, leading to the inauguration of the first mobile telecommunications network at a relatively early date.

The first public land mobile networks (PLMNs) came into being around 1950, and at that time they were reserved for a small and primarily well-to-do social class. For instance, the first public mobile telephone network in Germany (the A-Netz) started operation in 1958. It had human operators and used analog signals in the 150MHz band. With a geographic coverage of approximately 80 % of the Federal Republic of Germany, it had a maximum capacity of 10,500 subscribers. This first mobile telephone network in Germany did not have a cellular architecture, but instead consisted of a few distributed high-power transmitter and receiver facilities within whose coverage area it was possible to make telephone calls. It was also necessary to know the approximate location of the mobile telephone subscriber when placing a call to the subscriber, so that the proper radio zone could be selected by the human operator. The A-Netz remained in operation until 1977, when it was replaced by the B-Netz and subsequently the C-Netz, both of which are also analog systems.

At the beginning of the 1980s, mobile telephone networks with cellular architectures simultaneously came into being in many parts of the world. However, these systems were all mutually incompatible, and due to the expense of the terminal equipment (mobile telephones) and high usage charges, they were only suitable for a relatively well-to-do clientele. These networks are presently referred to as the first-generation (1G) mobile telecommunications networks. They were cell-based, but data transmission at the ‘air interface’ was still analog. Subscribers were identified by personalized mobile telephones, which means that each mobile telephone had a fixed association with a particular subscriber. One of the first 1G systems, the German C-Netz (which is also designated C-450), supported the use of cards. Magnetic-stripe cards were still used in the first C-Netz mobile telephones, but they were quickly supplanted by smart cards. This led to a distinction between the telephone and the subscriber, with the result that the personalization of telephones, which was standard at that time, was no longer necessary. As a consequence, the telephones, which are quite expensive compared with the

[2] A good condensed summary of the history of the methods used in the past for transmitting messages can be found in (among others) the book *Nachrichtentechnik* by Oskar Blumtritt [Blumtritt 97]

smart cards, became readily interchangeable. Due to strong competition, the prices of mobile telephones decreased rapidly, which was in the interest of network operators with regard to having the largest possible potential market.

The heterogeneous first-generation mobile telecommunication systems in Europe ultimately led to a desire among European postal authorities in the early 1980s to unify the many different country-specific systems. The intention behind this was to make it technically possible to use mobile telephones in more than one country, which above all would allow the prices of system components and terminals for a common European system to be drastically reduced as the result of economies of scale.

The end result of several years of work was a specification for the international Global System for Mobile Communications (GSM).[3] The first GSM systems underwent trials in 1991 and were put into regular service in 1992, and they have quickly spread far beyond their original European boundaries. However, in many parts of the world there are still other types of mobile telecommunication systems that are incompatible with GSM. Consequently, as early as 1985 the International Telecommunications Union (ITU) [ITU], with an eye to the future, started working on the international unification and functional extension of all existing mobile telecommunication systems. This concept was called 'Future Public Land Mobile Telecommunication Service' (FPLMTS). However, it failed to produce an internationally standardized solution, as was originally intended, so in 1995 it was converted into the IMT-2000 concept ('International Mobile Telecommunication at 2000 MHz'). IMT-2000 provides considerably more room for maneuvering with regard to implementation, and it has come to form the basis for the current UMTS and other third-generation mobile telecommunication systems.

It is common throughout the world to classify the technology of mobile telecommunications networks using a generation number. The generations are counted starting at '1', and they include only networks with cellular architectures. According to this scheme, early examples of mobile telecommunication networks, such as the German A-Netz and B-Netz, would belong to generation zero, but this designation is not commonly used. The designation 'first generation' (abbreviated as '1G') is applied to cellular mobile telecommunication networks with analog air interfaces. Some typical examples of 1G networks are AMPS and the German C-Netz. A second-generation (2G) system is understood to be a cellular mobile telecommunication network with digital data transmission on the air interface. The two most widely used 2G systems are GSM and CDMA. Functional extensions of GSM, such as the General Packet Radio System (GPRS) and EDGE ('Enhanced Data Rates for GSM and TDMA Evolution'), which head in the direction of the third generation, are typically referred to as 2.5G systems. The third generation (3G) also encompasses cellular mobile telecommunication networks with digital air interfaces, but with major extensions with regard to mobile data communications and Internet-compatible services compared with 2G systems. Some typical 3G systems are UMTS and CDMA 2000. Both of these systems are in turn members of the IMT-2000 family.

Details relevant to smart cards with regard to setting up GSM and UMTS mobile telecommunication systems are described in the sections of this chapter that are dedicated to these two systems.

The first efforts to develop concepts for the fourth generation (4G) of mobile telecommunication networks are presently underway. Future telecommunication systems will doubtless have significantly greater bandwidth efficiency than present systems, since frequency spectra

[3] A summary of the development of the GSM is presented in Section 13.2, 'The GSM System'

are a limited resource, as well as being a very expensive resource in some countries due to the way in which they are auctioned. In the GSM system, a frequency bandwidth of approximately 5 Hz is required for a transmission bandwidth of 1 bit/s, resulting in a frequency efficiency of 0.2 bit/s per Hz. In the UMTS system, this is already improved to 1 bit/s per Hz, and in systems still in the research stage, efficiencies of up to 30 bit/s per Hz have already been achieved.

There are many additional ideas and proposals for 4G systems, although none of them have yet achieved a confirmed status or a uniform development direction. Nevertheless, the basic features of the two main constraints have already been established: international interoperability of the terminal devices and high data transmission rates as needed. Interestingly enough, these two desires were already stated in the mid-1980s as major requirements for FPLMTS and IMT-2000.

Table 13.1 Some general technical parameters of the most important mobile telecommunication systems. One possible criterion for differentiating mobile telecommunication systems is the actual data transmission rate as a function of the system generation. With 2G systems, it is around 10 kbit/s, with 2.5G systems it lies in the range of 64–144 kbit/s, and with 3G systems it is 384–2000 kbit/s

System	Generation	Usual frequency range	Air interface	Special features
AMPS	1	≈850 MHz (824–894 MHz)	analog, FDMA	No smart card, data transmission rate 2.4 kbit/s.
C-Netz (C-450)	1	≈450 MHz (450–466 MHz)	analog, FDMA	With smart card, data transmission rate 2.4 kbit/s.
GSM	2	900 MHz (890–960 MHz), 1800 MHz (1710–1880 MHz), 1900 MHz	digital, TDMA with FDMA	With smart card (SIM), data transmission rate 9.6–14.4 kbit/s.
D-AMPS (IS-54)	2	≈850 MHz (824– 894 MHz)	TDMA	No smart card, extension of AMPS, data transmission rate 8 kbit/s.
CDMA (IS-95)	2	≈850 MHz (824–894 MHz)	CDMA	No smart card.
GPRS, EDGE	2.5	*see* GSM	*see* GSM	Extension of existing GSM networks for packet-switched services. Data transmission rate up to 384 kbit/s.
UMTS	3	2000 MHz (1900–2170 MHz)	digital, WCDMA	With smart card (USIM), data transmission rate up to 2 Mbit/s.
CDMA 2000	3	2000 MHz	digital, CDMA	Optional smart card (R-UIM), data transmission rate up to 2 Mbit/s.

13.1 SURVEY OF MOBILE TELECOMMUNICATION SYSTEMS

This section provides a technical summary of current mobile telecommunication systems, to the extent that this is necessary for understanding the use of smart cards in this area. Significantly more detailed descriptions of all of the technical aspects of currently used mobile telecommunications networks can be found in Jörg Eberspächer *et al.* [Eberspächer 00], Bernhard Walke [Walke 00] and Raymond Steele *et al.* [Steele 2001].

In this chapter, the term 'mobile telecommunication system' is used instead of 'mobile telephone system', since in all recent systems simple voice transmission is only one of many possible services, with the transmission of various types of data becoming increasingly more prominent.

13.1.1 Multiple-access methods

The frequency bandwidth available to a mobile telecommunication system, which is also called its frequency spectrum, is typically limited to a few tens of megahertz. In order to make this limited bandwidth quasi-concurrently available to as many subscribers as possible, 'multiple-access' methods must be used. The purpose of such methods is to allow the greatest possible number of mobile stations within a cell to access the network with acceptable quality by suitably exploiting radio transmission techniques and information technology.

There are basically four different types of multiple access methods. They differ in their cost of implementation and the efficiency with which they utilize the available bandwidth. These four methods are called frequency-division multiple access (FDMA), time-division multiple access (TDMA), code-division multiple access (CDMA) and space-division multiple access (SDMA). They are briefly described below.

FDMA (frequency-division multiple access)

With frequency-division multiple access, each transmitter is assigned a reserved frequency band within the total available frequency range. The transmitter is allowed to continuously and exclusively transmit within its assigned frequency band. With FDMA, each transmitter within a cell transmits on a different frequency. Incidentally, this is also the most commonly used method for conventional radio equipment, which uses a single common channel (a half-duplex link) for communications. If a full-duplex link is used (i.e., simultaneous uplink to the base station and downlink to the mobile station), which is usually the case for telephony, two frequency channels are naturally required to handle each call. Due to its limited technical complexity, FDMA is relatively well suited to mobile telecommunications using analog data transmission.

For instance, frequency-division multiple access was used for the air interface between fixed and mobile stations in the German C-Netz. In this system, separate 4.44-MHz frequency bands were reserved for uplink and downlink, with each band being divided into 222 frequency channels, each 20 kHz wide.

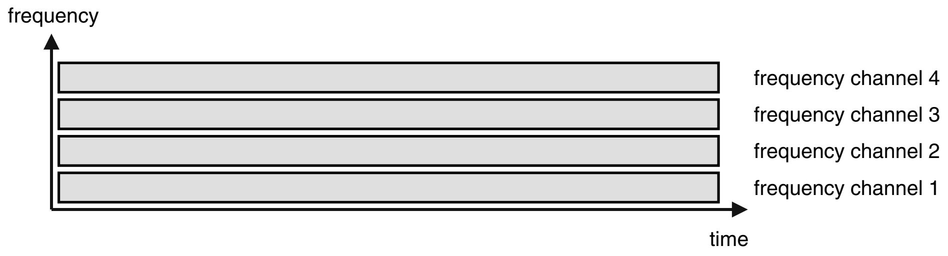


Figure 13.1 Frequency–time diagram for frequency-division multiple access (FDMA). In this example, each frequency channel corresponds to one transmission channel. The regions between the frequency channels are guard bands for suppressing interference between individual frequency bands

TDMA (time-division multiple access)

With time-division multiple access, data are transmitted quasi-concurrently from several transmitters to a single receiver on a single frequency. Each transmitter is assigned a particular time slot, within which it is allowed to transmit exclusively but not continuously. In the GSM system, for example, the time slot available for a signal burst is 577 μs (15/26 ms), of which 546 μs are occupied by the signal burst to be sent within this interval. The difference between these two values (31 μs) is used as a guard time to accommodate small timing variations. Maintaining the necessary exact timing of the time slots requires very precise and technically complex synchronization between the transmitter and the receiver. Furthermore, the signal propagation time between the transmitter and the receiver must be taken into account when time-division multiple access is used. For example, the difference in signal propagation time between mobile stations in the immediate vicinity of a base station and mobile stations 30 km from the base station is approximately 100 μs. In practice, these propagation time differences must be offset by 'premature' transmission, so that the signals transmitted by the mobile stations always arrive at the base station exactly within the time slots reserved for them.

Incidentally, the need to offset the transmission time in order to compensate for propagation time differences is what determines the maximum diameter of a cell in the GSM system. The maximum allowable interval for equalizing propagation times between the base station and the mobile station is 116.3 μs. This is the maximum time that a transmission can be sent prematurely and still arrive at the receiver within the prescribed time slot. This yields a maximum cell radius in the GSM system of approximately 35 km. Premature transmission is also called 'timing advance'.

In order to reduce the effects of frequency-selective interference, time-division multiple access can be combined with frequency hopping, in which both the transmitter and the receiver change frequency channels after each time slot in a predefined sequence. As a result, there is a high probability that interference in particular frequency ranges will only affect isolated signal bursts. In many cases, the results of such interference can be compensated using error-correcting transmission codes.

An example of the use of time-division multiple access in combination with frequency-division multiple access is the air interface between fixed and mobile stations in the GSM system. In this case, the available frequency band of 25 MHz is divided into 24 individual channels, each having a bandwidth of 200 kHz. Each of these frequency channels in turn

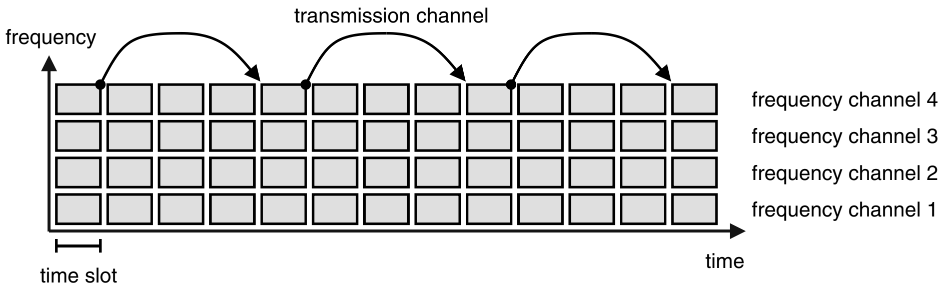


Figure 13.2 Frequency–time diagram for time-division multiple access (TDMA). In this example, each frequency channel is allocated four transmission channels. Each of the gray rectangles represents one signal burst. The regions between the frequency channels and time slots are guard bands and guard times for suppressing interference between the individual signal bursts

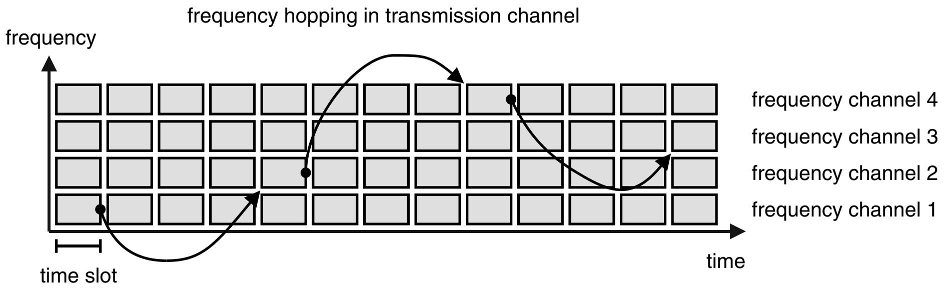


Figure 13.3 Frequency–time diagram for time-division multiple access (TDMA) with frequency hopping. Here each frequency channel is allocated four transmission channels. Each of the gray rectangles represents one signal burst. The regions between the frequency channels and time slots are guard bands and guard times for suppressing interference between the individual signal bursts

is allocated eight call channels. This means that up to eight mobile stations can concurrently transmit on a single frequency channel, with each mobile station having access to the frequency channel for an interval of 0.577 ms every 4.615 ms.

CDMA (code-division multiple access)

Code-division multiple access is a multiple access method in which data are transmitted to a receiver by multiple transmitters that concurrently transmit signals within the entire available frequency spectrum. Code-division multiple access is based on spread-spectrum technology, in which an original narrow-band signal is expanded into a wide-bandwidth radio signal using a transmitter-specific mapping law and then transmitted as a wideband signal. This wideband signal is received by the receiver, where it can be transformed back into the original narrow-band signal by employing the known mapping law used by the transmitter. In the wideband code-division multiple access (WCDMA) variant, two separate frequency bands are used for uplink and downlink, for which reason this CDMA variant is often referred to as frequency-division/code-division multiple access (FD/CDMA). In the time-division/code-division multiple access (TD/CDMA) variant, the uplink and downlink are separated by using different time slots.

Code-division multiple access has the advantage of bring highly insensitive to frequency-selective interference. It also provides weak protection against unauthorized eavesdropping if the transmitter-specific mapping law is not known to the attacker.

CDMA is used in the UMTS system in the WCDMA variant, using a bandwidth of 5 MHz each for uplink and downlink.

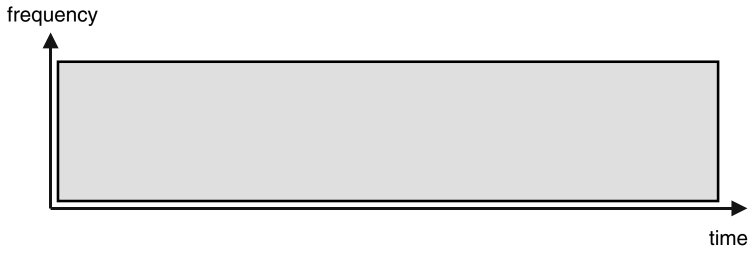


Figure 13.4 Frequency–time diagram for code-division multiple access (CDMA). Within the gray rectangle representing the available frequency spectrum, several transmitters transmit concurrently using transmitter-specific spread-spectrum signals, which can be converted back into the original signals by the receivers

SDMA (space-division multiple access)

Space-division multiple access is a multiple access method for transmitting data in parallel from multiple transmitters to a receiver using a single frequency. For this purpose, the transmitters use directionally selective (adaptive) aerials aimed at specific receivers. This requires a relatively high level of technical complexity, so this method is presently used only to a limited degree for base stations in the mobile telecommunications sector. The directionally selective aerials are usually antenna arrays with electronic beam-steering capability. This makes it unnecessary to physically aim the aerial towards the receiver. Space-division multiple access can basically be combined with other multiple access methods, but it is presently seldom used in the mobile telecommunications sector due to its unfavorable cost/benefit ratio.

13.1.2 Cellular technology

The frequency band available to a mobile telecommunications system must serve a very large number of mobile stations. It is thus not sufficient to simply use one or more multiple-access methods to attempt to service a relatively large region containing a large number of potential subscribers.

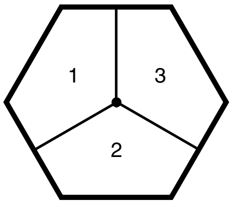


Figure 13.5 Schematic representation of space-division multiple access (SCMA) with a single radio cell. The base station located in the middle of the cell has three array aerials arranged at 120 degrees to each other. Each aerial covers one-third of the cell, with overlapping coverage areas

Taking the GSM system as an example, this can be briefly illustrated using a few key figures. Based on the bandwidth available to the GSM system and the multiple-access method used, it can be calculated that in theory, a maximum of approximately 1000 subscribers could be conducting telephone conversations at the same time (128 channels with eight time slots each yields 992 concurrent call channels). In practice, the number is significantly smaller, since some channels must be used for other purposes, such as signaling.

In order to allow several million subscribers to concurrently converse with each other, the entire mobile telecommunications system of a network operator is organized into cells, with a certain number of frequency channels being allocated to each cell. This technique, which was developed around 1970, is called cellular technology. Each set of neighboring cells is assigned several different frequency channels from the available frequency spectrum, thus avoiding interference due to the concurrent use of identical channels in adjacent cells. These frequency channels can be used again in other cells separated by one or more adjacent cells, without giving rise to interference problems. Frequency reuse is one of the fundamental principles of cellular technology. However, a useful side effect of cellular technology is that the transmit power required from the mobile telephone decreases as the cell size is reduced, which reduces both power consumption and the amount of 'electronic smog' generated by mobile telephones.

A group of several cells with different assigned frequency channels can be considered to form a cluster. Clusters of 3, 4, 7, 12 and 21 cells are customary. In network planning, the entire region to be covered by the network is blanketed using these clusters. In this process, the individual cells are represented in simplified form as hexagons instead of circles, so the network plan resembles a honeycomb.

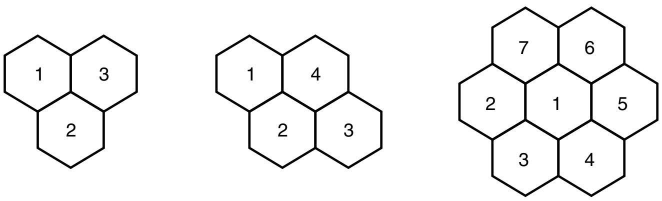


Figure 13.6 Graphic representation of typical clusters with three, four and seven cells. Each of the sequentially numbered cells has a certain number of assigned frequency channels, which differ from those of its neighboring cells

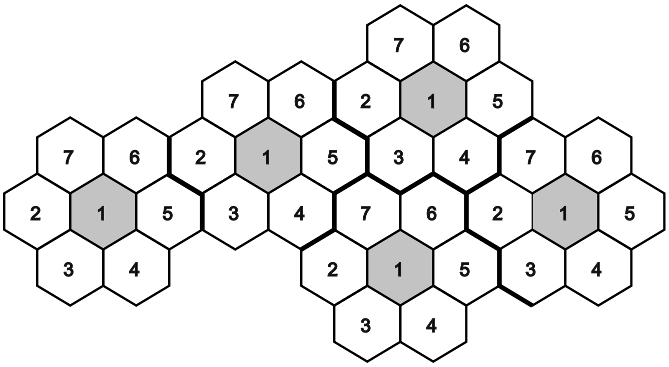


Figure 13.7 Graphic representation of full radio coverage of a region with cells using seven cells per cluster. It can be clearly seen that any two cells having the same frequency channels are separated by at least two other cells having different frequency channels

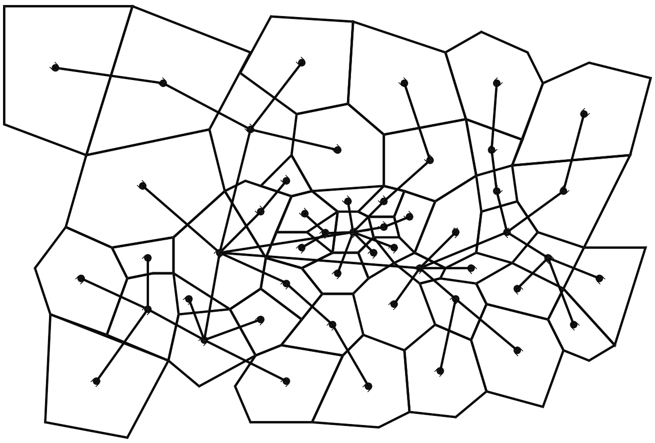


Figure 13.8 Sample section of a cell plan for an actual network. Different sizes of cells are used to ensure that an acceptable level of network coverage is achieved despite variations in network loading (which are usually due to high subscriber densities). In this cell plan, for instance, a traffic intersection with a high density of mobile telephones is located in the middle

13.1.3 Cell types

A decisive factor with regard to obtaining an economical network structure with the best possible network coverage is the types of cells that are used. The following specifications for cell dimensions are rough approximations and are primarily intended to illustrate the basic uses of the different types of cells.

For the large-scale coverage of areas with low incident network loading and terrain that provides little screening, large cells with a typical diameter of 10 to 40 km are used. Such cells are commonly found in rural areas with low population densities and flat terrain. The next smaller size of cell is a macrocell, which typically has a diameter of 1 to 10 km. Macrocells are commonly used in the centers of relatively small settlements and in many suburbs of agglomeration regions.

Microcells, with a typical diameter of 0.1 to 1 km, are frequently used at traffic intersections and within inner-city regions with high call volumes. The smallest type of cell is called a picocell, and it has a diameter of 50 to 100 m. Such cells are often used within buildings, such as inside offices and meeting rooms.

A special type of cell is the umbrella cell which is a cell containing one or more relatively small cells that service an area that cannot be covered by the smaller cells it contains. This special type of cell can be used where there is a highly localized high level of call traffic in a region that otherwise has a low network load. Naturally, the umbrella cell and the smaller cells covered by it must use different frequency channels.

An example of using an umbrella cell would be an alpine region in which there is a single ski hut with snack and drink bars located in a large ski area. In the network, a picocell would be provided to service the ski hut, with the rest of the surrounding ski area being serviced by an umbrella cell. In some ski areas, it would be a good idea to install at least one picocell for each of the lower lift stations, so that skiers can while away the time spent waiting for the lifts in (telephone) conversation.

An additional subtype of cell is the selective cell, which covers only a sector of a circle instead of providing uniform circular coverage over 360 degrees. This type of cell is typically mounted facing an underpass, and thanks to its radiation pattern, which resembles that of a spotlight, it can provide very good mobile telecommunications coverage within the underpass. Selective cells are also a proven means to provide coverage inside tunnels.

Approximately 3000 base stations would be needed to provide full coverage for approximately 5 million subscribers in a country the size of Germany (all 16 federal states), which has an area of approximately 360,000 km^2. With 20 million subscribers, the number of base stations would rise to approximately 12,000 for nearly 100 % coverage. With a network of this magnitude, approximately 60 mobile switching centers (MSCs) would be used. Incidentally, as of mid-May 2001, the four German GSM networks had a combined total of approximately 52,000 base stations and 164,000 cells.

13.1.4 Bearer services

The function of bearer services in a mobile telecommunication system is to transport voice and data between the terminal and the background system. For the acoustic transmission of voice at the currently standard telephone quality, a minimum frequency bandwidth of 3100 Hz is necessary, which is designated 'telephone bandwidth'. This bandwidth results from the fact that the human voice primarily uses the frequency range between 300 Hz and 3400 Hz. In the GSM system, a compressed, discontinuous data stream is generated from this continuous signal, and in the full-rate mode this data stream requires a data transmission rate of 13 kbit/s. This transmission rate is sufficient for a half-duplex telephone connection with standard telephone quality, which means for an uplink or a downlink. If both parties to the conversation wish to be able to speak at the same time as usual, a full-duplex link is required, as is commonly used in telephony. This results in a net transmission rate of 26 kbit/s for the two directions. In practice, the required transmission rate is rendered considerably greater by control data and sophisticated error correction codes. In the GSM system, the required gross transmission rate for uplink and downlink is approximately 34 kbit/s.

Ideally, bear services for data transmission allow data exchange between the two communicating parties to be transparent, which means the these parties do not have to give any consideration to the communications protocols used by intermediate parties. In the GSM system, SMS, USSD, CSD, HSCSD and GPRS are used as bearer services. The availability of these services depends on the technical capabilities of the particular mobile telephone and the GSM system into which it is logged in.

Short message service (SMS)

If the short message service (SMS) is used as a bearer service, the data to be transmitted must be packaged in SMS messages. These messages can have a maximum length of 176 bytes, and the maximum length of a data packet is 140 bytes. The difference of 36 bytes is needed for the transmission data. Unfortunately, SMS does not guarantee that a series of short messages sent one after the other will arrive in their original order. A guaranteed transmission rate cannot be specified with this packet-switched data transmission service, in part because it is strongly dependent on actual network loading and in part because it depends on whether the mobile station is logged in to its home network or some other network. Typical elapsed times for short messages range from 0.5 seconds to 20 seconds. Expressed in terms of a transmission rate, this corresponds to a range of 2.8 kbit/s to 94 bit/s.

Unstructured supplementary services data (USSD)

The design of the USSD bearer service is similar to that of the SMS. The principal difference is that USSD is a circuit-switched service with a maximum message length of 182 bytes.

Circuit-switched data (CSD)

In the GSM system, the circuit-switched bearer service CSD simply consists of transmitting data instead of voice across the air interface. For this purpose, the data are fed into the mobile station using a modem. The gross transmission rate is typically either 9.6 kbit/s or 14.4 kbit/s.

High-speed circuit-switched data (HSCSD)

HSCSD is a circuit-switched bearer service defined in GSM 02.34, with a technical structure similar to that of CSD. Theoretically, it has gross data rate of up to 76,800 bit/s for combined uplink and downlink transmissions.

General packet radio system (GPRS)

GPRS is a packet-switched bearer service defined in GSM 01.60 and GSM 02.60. It supports a theoretical maximum data transmission rate of up to 115.2 kbit/s for uplink and downlink.

13.2 THE GSM SYSTEM

The smart card used in GSM mobile telephones, which is called the 'subscriber identity module' (SIM), was and still is the pioneer in terms of functionality and memory capacity. This is in part due to the fact that smart cards used in mobile telephones, whose manufacturing costs are several hundred euros, are significantly less price sensitive than other types of smart cards, such as those used for electronic payments or medical applications. Another decisive factor with regard to smart card technology is the generally high rate of evolution of the entire telecommunications sector. The pioneering position with regard to technology and standardization that is presently held by the SIM, in comparison with all other smart card applications, is the reason why this topic is described here in such great detail.

GSM, which was commercially inaugurated in 1992, became the international standard for mobile telecommunications systems within only a few years. This includes transmitting not only voice but also data, which are presently still primarily transmitted in the form of 'short messages' using SMS. In mid-2001, there were a total of 400 mobile telecommunications networks in 171 countries based on the GSM standard, with more than 565 million subscribers. More than 20 billion short messages are transmitted every month.[4] Mobile telecommunications networks based on the GSM standard often have country-specific designations. In Germany, for instance, the four operational GSM networks are called the D-Netz (900-MHz and 1800-MHz GSM variants) and the E-Netz (1800-MHz variant), and in Austria the GSM network is in part also referred to as the A-Netz.

Specification of the GSM system started in 1982 under the auspices of the *Conférence Européenne des Postes et Télécommunications* (CEPT). The objective was to generate a specification for a transnational, interoperable mobile telecommunications network. In the course of time, these efforts led to the conclusion that it was possible to draft specifications for a transnational, interoperable and ISDN-compatible digital cellular mobile telecommunication system operating in the 900-MHz band. The Groupe Spécial Mobile was founded for this purpose, which gave rise to the original abbreviation 'GSM'. In 1986, the GSM Permanent Nucleus was established, with headquarters in Paris, to coordinate the generation of the specification. It was later also responsible for specifying a wide variety of tests for system components. From a technical perspective, it is interesting to note that a number of the technologies that were chosen for GSM at that time were fully new and untested in practice. For instance, the air interface using a combination of time-division multiple access with frequency-division multiple access and digital data transmission was totally unexplored territory for large-scale mobile telecommunication applications. These decisions led to many technical problems, particularly in the system development stage, but from the present perspective they can be regarded as a fortunate choice, since GSM proved to be an innovative system that was not burdened with the technical ballast of the early days of mobile telecommunications.

[4] A good overview of current statistical figures and network operators can be found at GSM World [GSM]

The common contractual basis for operators of GSM networks is the Memorandum of Understanding (MoU), which was first signed by 15 European network operators in 1987. The GSM Association is an internationally active body, with offices in Dublin and London, for the coordination of mobile telecommunications systems. It was founded in Copenhagen in 1987, and it is responsible for the development and application of the GSM standards. The GSM Association represents more than 500 network operators, manufacturers and suppliers in the GSM industry. In 1989, the specifications developed by the various working groups under the leadership of the GSM Permanent Nucleus were incorporated into the newly founded European Telecommunication Standards Institute (ETSI), where they have since been further developed. In 1990, all of the GSM Phase 1 specifications were complete in an acceptable form and were frozen.

In 1998, the Subscriber Identity Module Expert Group (SIMEG) started work on the specification for the GSM smart card, which is called the ‘subscriber identity module’ (SIM). This group was composed of representatives of card manufacturers, manufacturers of mobile telephones and network operators. Working under the auspices of the ETSI, the SIMEG generated the specification for the interface between the smart card and the mobile telephone. This specification bears the name ‘GSM 11.11’. In 1994, the SIMEG was transformed into the newly founded Special Mobile Group 9 (SMG9), which retained the duties and authorities of the original group. The SMG9 was given the mandate of further developing and maintaining all of the SIM specifications up to 2000. In 2000, the SMG9 was dissolved, and its responsibilities were divided between two newly founded expert groups. The ETSI Project Smart Card Platform (EP SCP) expert group handles all generic issues in the area of smart cards for telecommunications, while the 3GPP expert group is responsible for the application-specific interface between the mobile telephone and the SIM or USIM.[5]

The first operating GSM network was demonstrated at the ITU Telecommunications Fair in Geneva in 1991. During the fair, approximately 11,000 calls were routed without any major problems. In 1992, the first GSM systems were put into regular service in several European countries (Denmark, Finland, France, Germany, Italy, Portugal and Sweden). At that time, there were approximately 250,000 subscribers. Also in that year, the first ‘roaming agreement’ between two network operators was signed, and the first non-European network operator signed the MoU, which meant that it officially decided to use the GSM system. Only one year later, at the end of 1993, the millionth subscriber was registered. In that year, the first GSM-1800 network began operation in Great Britain. In 1995, the first GSM-1900 network went into operation in the USA, and at the end of July 1998, the 100-millionth GSM subscriber was registered. In mid-2001, there were 500 million subscribers throughout the world, and it is anticipated that there will be 1 billion subscribers in 2005.

The GSM specifications were extended in 1991 and 1992. They now also cover the 1800-MHz frequency band (1710–1785 MHz uplink, 1805–1880 MHz downlink; wavelength approximately 16.6 cm) with GSM 1800 (previously called Digital Cellular System, or DCS) and the 1900-MHz band (1850–1910 MHz uplink, 1930–1990 MHz downlink; wavelength approximately 15.8 cm) with GSM 1900 (previously called Personal Communication System, or PCS). Since then, GSM in the original 900-MHz frequency band (880–915 MHz uplink,

[5] A comprehensive overview of the interesting history of the expert groups for the standardization of the SIM, USIM and UICC can be found in an article by Klauss Vedder entitled ‘The Subscriber Identity Module: Past – Present – Future’, in [Hillebrand 02]

925–960 MHz downlink; wavelength approximately 33.3 cm) is referred to as GSM 900. Due to the higher frequencies and lower levels of transmitted power, the maximum diameter of a cell in the higher-frequency systems is only 20 km. Consequently, they are primarily used in regions with high subscriber density, and less often in regions with low subscriber density. The principal difference between GSM 1800 and GSM 1900 is found in the transmitter and receiver components on either side of the air interface.

The low data transmission rate of the GSM system, which is 9600 bit/s or 14,400 bit/s with an improved codec for the air interface, has relatively quickly proven to be a weakness of the system. The rapidly increasing demand of mobile subscribers for transferring large volumes of data has further exacerbated this shortcoming. Consequently, an evolution path leading to increased data transmission rates, and particularly packet-switched transport services, has been specified. The next stage in the development of GSM is the circuit-switched HSCSD (high-speed circuit-switched data) service. With HSCSD technology, a theoretical data transmission rate of up to 76,800 bit/s (8×9600 bit/s) for uplink or downlink can be achieved by using 'channel bundling' to combine several existing time slots in the air interface. Existing GSM networks can be extended to support HSCSD with relatively little effort by modifying the base stations and using special mobile telephones. However, the disadvantage of this approach is that the number of transmission channels can be increased by at most a factor of 8, so HSCSD will probably not become a major success.

The packet-switched General Packet Radio System (GPRS) service is the following step in the evolution of GSM. It provides a packet-switched connection with a data transmission rate of up to 115.2 kbit/s (downlink or uplink) by bundling the eight existing 14.4-kbit/s time slots. A mobile telephone with GPRS technology is constantly logged in to the network for data transport, and thus always available for data transmission. For this reason, GPRS is also quite suitable for discontinuous data transmission. A drawback is the relatively high cost of upgrading the base stations. GPRS is regarded as a G2.5 technology, and it has a good chance of becoming a significant factor in extending the lifetime of existing GSM systems.

The final planned stage of enhancement for GSM networks is EDGE ('Enhanced Data Rates for GSM and TDMA Evolution'). Using the existing network infrastructure, GSM mobile telephones with EDGE technology can be connected to base stations with a data transmission rate of up to 384 kbit/s by using a different modulation method for the air interface. The extent to which EDGE technology will play a significant role in the future, when it will have to compete with 3G systems such as UMTS that will then be available, cannot presently be foreseen.

The designated successor to GSM is UMTS, whose basic architecture is based on GSM. It thus does not represent a fundamentally new mobile telecommunications technology, as did GSM at the time it was developed.

13.2.1 Specifications

A large number of interrelated and mutually dependent specifications were necessary to fully describe the GSM system in technical terms. In total, there are approximately 130 individual specifications, with a total size of more than 6000 pages.

Particularly in connection with the GSM system, the terms 'specification' and 'standard' are often used interchangeably. In the case of GSM, both of these terms are actually justified.

Since they are published by the ETSI standardization organization, the specification documents formally have the status of standards. However, their technical descriptions are so strict that practically all implementations based on them are mutually compatible, which is a typical characteristic of a specification. For this reason, in this book we consistently refer to the GSM numbering scheme (e.g., GSM 11.11) that is commonly used in technical circles, rather than the corresponding ETSI standards (e.g., TS 100977), which are identical in content.

The course of development of the GSM system is characterized by a series of phases building on top of each other. The basic services (voice transmission, call forwarding, roaming and the SMS message service) were implemented in Phase 1, which began in 1992. In Phase 2, which began in 1996, supplementary services were added, including conference calls, call handover, call number negotiation and GSM in the 1800-MHz frequency band. This was followed by Phase 2+, in which these services were augmented with the functions of the SIM Application Toolkit, HSCSD and GPRS (among others).

As is usual with specifications, the GSM specifications employ their own technical vocabulary. This vocabulary is precisely defined in technical terms in various lists of abbreviations and glossaries, and is only applicable to the GSM field. A summary is provided by GSM 1.04 ('Abbreviations and acronyms'). Due to this technical vocabulary, it is generally relatively difficult for newcomers to become familiar with GSM, since it is constantly necessary to consult the explanations of the abbreviations in the appropriate places when studying the specifications.

The specification forming the basis for the GSM security module in the mobile telephone is designated GSM 02.17 ('SIM Functional Characteristics') and contains a relatively abstract description of the functional requirements for the SIM. The most important card-specific document in the GSM system, GMS 11.11 ('Specification of the Subscriber Identity Module – Mobile Equipment (SIM – ME) interface'), is based on this specification. In more than 170 pages, GSM 11.11 precisely and unambiguously specifies the interface to the SIM. This is a pure interface specification that does not contain any details about the actual implementation.

The specifications of the electrical parameters of smart cards using 3-V and 1.8-V technology, which supplement GSM 11.11, are contained in GSM 11.12 ('Specification of the 3 Volt Subscriber Identity Module – Mobile Equipment (SIM – ME) interface') and GSM 11.18 ('Specification of the 1.8 Volt Subscriber Identity Module – Mobile Equipment (SIM – ME) interface').

Besides these specifications, which primarily describe the basic functionality of the SIM, there is also GSM 11.14 ('Specification of the SIM Application Toolkit for the Subscriber Identity Module – Mobile Equipment (SIM – ME) interface'), which describes a platform for secure supplementary services in the SIM. These are referred to as the SIM Application Toolkit (SAT). This specification was published in 1996, and it primarily offers network operators the possibility of loading their own applications into the smart card for controlling the mobile telephone. GSM 11.14 specifies in detail how functions such as driving the display, polling the keypad, sending short messages (SMS) and other functions related to suitable value-added applications must be implemented in the SIM.

The requirements specifications GSM 02.48 ('Specification of security mechanisms for the SIM application toolkit, stage 1') and GSM 03.48 ('Specification of security mechanisms for the SIM application toolkit, stage 2'), which is based on GSM 02.48, introduce two important security mechanisms for the SIM. The first item that they address is specifying security mechanisms for end-to-end communications between the background system and the SIM that are protected against eavesdropping and manipulation. In practice, these mechanisms are primarily used for secure data transmission via the air interface (‘over the air’, or OTA). The second item, which is addressed by GSM 03.48, is a description of the basic mechanism for remote file management (RFM) and remote applet management. This description is in principle bearer-independent, but in GSM 03.48 it is presented using transport via SMS as an example.

Particularly in the telecommunications environment, smart cards with Java have become established very quickly, which is why the effects of such cards were seen relatively early in the GSM specifications. The basis for all smart card operating systems with executable program code is formed by the GSM 02.19 specification. It contains a list of all basic services for a language-independent API for executable program code in the SIM. Based on this standard, GSM 03.19 specifies a detailed implementation of a Java Card API for SIMs based on the Java Card 2.1 specification. This standard is the key document for using Java Card with GSM. It is supplemented by GSM 11.13, which specifies the test environment, test applications, test procedures, test coverage and individual test cases. The described tests are aimed exclusively at the IT aspects of a Java Card implementation for GSM.

The GSM specifications related to the SIM are not being developed any further, since the functionality of the SIM is fully adequate for the current needs of the GSM system. The only modifications that are still routinely made to the relevant specifications involve clarifications of passages that are subject to interpretation. Since 1999, the focus has been on standardizing

Table 13.2 The most important standards for the SIM and SIM-related services[6]

GSM 02.09	Security Aspects
GSM 02.17	SIM Functional Characteristics
GSM 02.19	Subscriber Identity Module Application Programming Interface (SIM API); Service description; Stage 1
GSM 02.48	Specification of security mechanisms for the SIM application toolkit, Stage 1
GSM 03.19	Digital cellular telecommunications system (Phase 2+); Subscriber Identity Module Application Programming Interface (SIM API); SIM API for Java Card; Stage 2
GSM 03.48	Specification of security mechanisms for the SIM application toolkit, Stage 2
GSM 09.91	Interworking aspects of the Subscriber Identity Module – Mobile Equipment (SIM – ME) interface between Phase 1 and Phase 2
GSM 11.11	Specification of the Subscriber Identity Module – Mobile Equipment (SIM – ME) interface
GSM 11.12	Specification of the 3 Volt Subscriber Identity Module – Mobile Equipment (SIM – ME) interface
GSM 11.13	Test specification for SIM API for Java card
GSM 11.14	Specification of the SIM Application Toolkit for the Subscriber Identity Module – Mobile Equipment (SIM – ME) interface
GSM 11.17	Subscriber Identity Module (SIM) conformance test specification
GSM 11.18	Specification of the 1.8 Volt Subscriber Identity Module – Mobile Equipment (SIM – ME) interface

[6] A general list of all GSM standards related to the SIM is provided in the directory of standards in Chapter 16. All of the GSM standards can also be obtained free of charge from the ETSI web server [ETSI]

the UICC ('universal integrated circuit card') with the USIM ('universal subscriber identity module') application, which is primarily being pursued under the auspices of the 3GPP.

13.2.2 System architecture and components

Every GSM network can be divided into three general subsystems, which are described in general terms in the GSM 01.02 specification ('General description of a GSM Public Land Mobile Network (PLMN)'). These three subsystems are the radio subsystem (RSS), the network and switching subsystem (NSS) and the operation subsystem (OSS).

The radio subsystem is composed of the mobile telephone, which is called the mobile station (MS), and the base station subsystem (BSS). The mobile station consists of two physically and logically separate components, which are called the mobile equipment (ME) and the subscriber identity module (SIM). The mobile equipment is the radio and encryption component with the user interface, while the SIM is the correct designation (in GSM nomenclature) for a GSM-specific smart card. These two components together form the operational mobile telephone.

As a rule, the base station subsystem is formed by the base stations located at the center of each cell. The functions of the base station subsystem are to establish contact with the mobile telephones via the air interface and to supply data to the higher-level components of the network. A base station consists of one or more base transceiver stations (BSTs) and a base station controller (BSC). The base station transceiver, with its aerial and associated radio-frequency components, is the actual transmission and reception component. A typical receiver module for a base station transceiver has eight 200-kHz channels, so in theory it can concurrently maintain eight active links to mobile stations. In practice, only seven active links are usually used, since one channel is usually reserved for administrative communications. One, three or six receiver modules are usually fitted in each base transceiver station. One or more base transceiver stations are in turn managed by a base station controller. A typical setup consists of three base station transceivers arranged at 120 degrees to each other, all connected to a base station controller. If a mobile station moves from the send/receive region of one base transceiver station into that of another base transceiver station, and both base transceiver stations are assigned to the same base station controller, the base station controller can independently initiate the handover after signaling this to the responsible mobile switching center.

Data transmission via the air interface is encrypted and has a net transmission rate of 13 kbit/s in full-rate mode. It employs a lossless compression method with technically sophisticated error correction mechanisms, such as frequency hopping, convolutional coding and interleaving.

The network and switching subsystem essentially consists of the mobile switching center and the visitor location register (VLR). A mobile switching center (MSC) manages multiple base station subsystems. It forms the link between the base station subsystems connected to it, other mobile switching centers and, of course, the public switched telephone network. The mobile switching center is responsible for setting up, managing and shutting down connections, handling call charges and supervising supplementary services, such as call forwarding, call blocking and conference calling. The visitor location register (VLR) contains information about all mobile stations currently within range of the associated mobile switching center. This information is needed for functions such as routing a call to a particular mobile telephone via the proper base station subsystem and radio cell. The VLR also maintains a list of mobile stations belonging to subscribers of other networks that have logged into the network of the associated mobile switching center via roaming.

The topmost hierarchical level in a GSM system is the operation subsystem. It consists of the operation and maintenance center (OMC), the authentication center (AuC), the home location register (HLR) and the equipment identity register (EIR). The operation and maintenance center is responsible for regular network operation, subscriber administration and call billing. The authentication center is the security component on the network side, and in a manner of speaking it is the counterpart to the SIM on the mobile side. It generates and manages all keys and algorithms needed for operating the system, especially for authentication of the mobile stations (i.e., the SIMs). Another central component is the home location register, which contains all of the subscriber data as well as the localization data for each of the mobile stations. The equipment identification register is the complement to the HLR for mobile stations instead of subscribers. It contains essential data, such as the serial numbers of all mobile stations represented in the network.

13.2.3 Important data elements

This section describes a selection of important data elements that are primarily related to the SIM and its functions. The coding of the described data elements can be found in the descriptions of the typical data of a SIM.

Coding of alphanumeric characters

In the original middle-European GSM system, alphanumeric characters were and still are coded using a 7-bit code derived from the ASCII code. This code is defined in the GSM 03.08 specification. However, the spread of the GSM to other countries made it necessary to extend the character set. Consequently, the UCS-2 16-bit subvariant of the Universal Character Set (UCS) is used for characters that cannot be represented using the west-European character set as defined by GSM 03.38. The UCS-2 character set allows the most important characters of all living languages to be represented.[7]

Three different schemes are defined for character coding using the USC-2 character set, in the interest of minimizing memory space. The preferred scheme (Scheme 1), which, however, requires the most memory, is identified by having a value of '80' for the first byte. This is followed by the 16-bit USC-2 character code, with the most significant byte first. Unused bytes are set to 'FF'.

Scheme 2 is identified by having a value of '81' in the first byte. The second byte contains the number of characters in the character string. The two following bytes represent a 16-bit pointer to the UCS character set, which is used to select a language-specific character within the UCS. Bits 1–7 and bit 16 of this pointer are set to '0'. If bit 8 of one of the following bytes has a value of °1°, bits 1–7 of that byte are added to the pointer value, with the resulting pointer value then indicating the actual character in the UCS character set. If bit 8 has a value of °0°, the character in question is a member of the 7-bit character set defined by GSM 03.38.

[7] See also Section 4.2, 'Coding Alphanumeric Data'

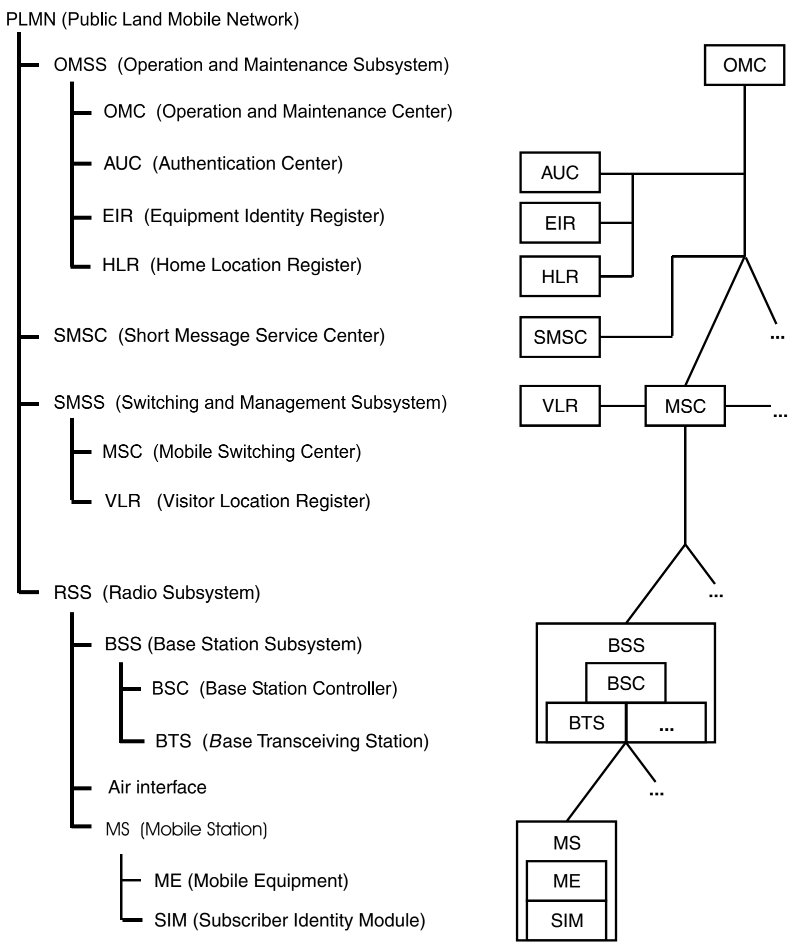

PLMN (Public Land Mobile Network)
OMSS (Operation and Maintenance Subsystem)
OMC (Operation and Maintenance Center)
AUC (Authentication Center)
EIR (Equipment Identity Register)
HLR (Home Location Register)
SMSC (Short Message Service Center)
SMSS (Switching and Management Subsystem)
MSC (Mobile Switching Center)
VLR (Visitor Location Register)
RSS (Radio Subsystem)
BSS (Base Station Subsystem)
BSC (Base Station Controller)
BTS (Base Transceiving Station)
Air interface
MS (Mobile Station)
ME (Mobile Equipment)
SIM (Subscriber Identity Module)
OMC
AUC
EIR
HLR
SMSC
VLR
MSC
BSS
BSC
BTS
MS
ME
SIM

Figure 13.9 Basic architecture of a typical mobile telecommunications system compliant with the GSM 01.02 specification. In this example, the EIR and HLR databases are centralized. Since many aspects of the PLMN configuration are left to the operator of a particular network, the databases may be decentralized and distributed among several MSCs if necessary (e.g., due to high network loading). For ease of understanding, a link to a short message service center (SMSC) is also shown here, although this is not a direct GSM system component. One or more radio subsystems may be combined to form a location area (LA), and one or more network and switching subsystems may be combined to form a service area (SA)

Scheme 3 is identified by having an initial byte value of '82'. As with Scheme 2, the second byte contains the length of the character string, while the third and fourth bytes represent a complete 16-bit pointer to the USC character set table. If bit 8 of the following byte has a value of °1°, the following seven bits must be added to the pointer value to uniquely determine the UCS character. If bit 8 has a value of °0°, the character in question is a member of the 7-bit character set defined by GSM 03.38.

Table 13.3 The databases essential to the operation of a GSM system, and the most important data elements in these databases

Database	Data elements
HLR (home location register)	Subscriber information IMSI (international mobile subscriber identity) MSISDN (mobile station ISDN number) Service restrictions (e.g. roaming not allowed) Subscribed services Parameters for supplementary services Information about the subscriber's equipment Authentication data (i.e. RAND, SRES, Kc triplet) (implementation-dependent) Localization data (mobile location information) MSRN (mobile station roaming number) Address of current VLR (if available) Address of current MSC (if available) TMSI (if available)
VLR (visitor location register)	Subscriber information IMSI (international mobile subscriber identity) MSISDN (mobile station ISDN number) Parameters for supplementary services Information about the subscriber's equipment Authentication data (i.e. RAND, SRES, tuple) (implementation-dependent) Localization data (mobile location information) TMSI (temporary mobile subscriber identity) MSRN (mobile station roaming number) LAI (location area information) TMSI (if available)
EIR (equipment identity register)	IMEI (international mobile equipment identity) of all mobile stations (white list) IMEI of mobile stations to be reported (graylist) IMEI of blocked mobile stations (blacklist)

SIM service table (SST)

The SST contains a table of services that can be used with or enabled in addition to voice service, such as short message service or fixed dialing number service.

Fixed dialing numbers (FDN)

Fixed dialing numbers are a special type of dialing numbers that can be selected even when all other dialing numbers are blocked in the mobile telephone.

ICC identification (ICCID)

The ICCID is a unique identification number for the smart card. It is BCD-coded and 10 bytes long, and it can be right-padded with 'F' as necessary.

International mobile equipment identity (IMEI)

The IMEI is a unique device number for the mobile station. It contains 15 digits and thus usually occupies eight bytes. It is composed of the six-digit type approval code, two manufacturer code digits, a six-digit serial number and a check digit. The IMEI is stored in the mobile telephone and in the equipment identification register (EIR) in a central location.

International mobile subscriber identity (IMSI)

The IMSI is the unique subscriber identity within the GSM system. It is BCD-coded and has a length of nine bytes, which may be right-padded with 'F' as necessary. It consists of the mobile country code (MCC), the mobile network code (MNC) and a serial number assigned by the network operator. The IMSI is normally never transmitted over the air interface in cleartext, in order to prevent the location of a mobile station from being illicitly traced. Instead of the IMSI, the TMSI is normally used together with the LAI for identification purposes.

Individual key (Ki) and cipher key (Kc)

The keys Ki and Kc are secret keys for symmetric cryptographic algorithms. Ki is the card-specific key for the cryptographic computation of the authenticity of the SIM, and Kc is used for encrypting data transmitted between the mobile station and the base station via the air interface.

Short message service (SMS)

The short message service allows messages with a maximum length of 160 alphanumeric characters to be transmitted between the network and the mobile station via the signaling channel. SMS service is used not only for conveying short messages for subscribers, but also as a bearer service for transmitting data to the mobile telephone or the SIM, for instance for the WAP and OTA services.

Abbreviated dialing numbers (ADN)

Abbreviated dialing number are dialing numbers stored in the mobile telephone or the SIM along with supplementary information, which can be easily and quickly selected using a menu or special buttons.

Location area information (LAI)

The LAI is the unique position information of the mobile station. It is used in combination with the TMSI to generate a unique subscriber identity. The LAI consists of a three-digit country code (CC), a two-digit mobile network code (MNC) and a location area code (LAC), which has a maximum length of five digits.

Mobile station ISDN number (MSISDN)

The MSISDN is the dialing number of the mobile station. It is independent of the subscriber identity (IMSI).

Temporary mobile subscriber identity (TMSI)

The TMSI is a temporally and spatially limited subscriber identity with a length of four bytes. It is used to protect the true subscriber identity. The TMSI is only unique in combination with the location area information (LAI). The TMSI is assigned by the VLR, where it is also stored.

13.2.4 The subscriber identity module (SIM)

The SIM is a mandatory security module located in the mobile telephone of a GSM system as an exchangeable component. It is defined as follows in the GSM 02.17 specification: 'The SIM is an entity that contains the identity of the subscriber. The primary function of the SIM is to secure the authenticity of the mobile station with respect to the network'.

Besides its primary functions of holding the identity of the subscriber, which is realized using a PIN, and authenticating the mobile station with respect to the network, the SIM also performs a number of other functions. It allows program execution to be protected against manipulation, and it makes it possible to store data such as dialing numbers, short messages and personal configuration settings for the mobile telephone. In addition, it is the bearer for secure supplementary services used with mobile telecommunications.

Two different SIM formats are used in the GSM system. In mobile telephones designed to allow the SIM to be exchanged relatively often, the ID-1 format is used. This is based on the idea of a company or family telephone with a separate card for each user. Mobile telephones with small dimensions, whose SIMs are intended to be exchanged only rarely, use plug-in SIMs in the ID-000 format. However, the only difference between the two types of SIMs is the physical size of the card. Their logical and physical characteristics are otherwise fully identical. Since the mid-1990s, mobile telephones have become more or less personal accessories. This has had an effect of the size of card used, since it is no longer necessary to exchange the card depending on who is using the telephone. Already in 1995, half of all ID-1 cards sold were punched to allow a card in ID-000 format to be broken loose, and since 1998 practically all cards have this feature.

Communications between the mobile equipment and the SIM use the $T = 0$ protocol with the standard parameters, as specified in ISO/IEC 7816-3. The data transmission convention can

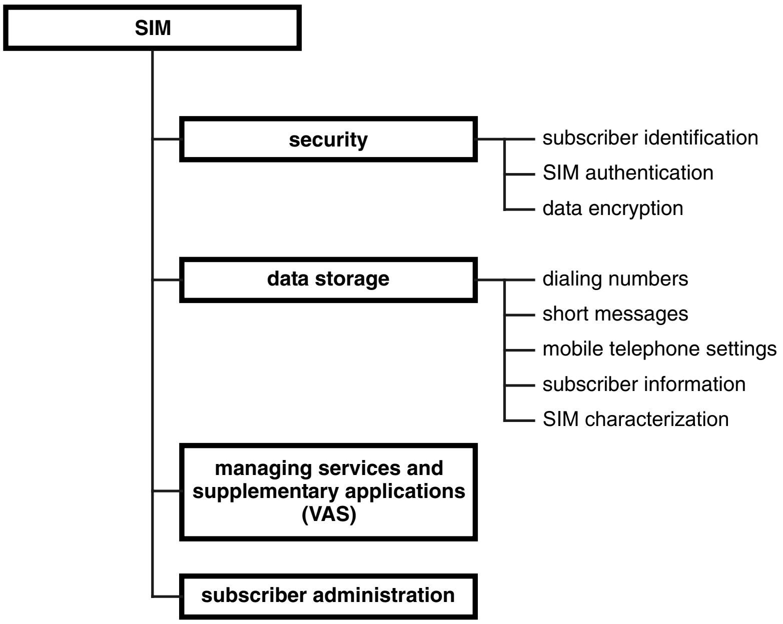

SIM
security
subscriber identification
SIM authentication
data encryption
data storage
dialing numbers
short messages
mobile telephone settings
subscriber information
SIM characterization
managing services and supplementary applications (VAS)
subscriber administration

Figure 13.10 Classification of the basic functions of the SIM in the GSM system

be freely selected by the card via the ATR. There is provision for a PPS, and this capability is frequently used to increase the data transmission rate. The divider value (clock rate conversion factor) typically used by many mobile telephones is 64, which yields a data transmission rate of 78 kbit/s with a 5-MHz clock rate. In isolated cases, a value of 31 ($\approx$156 kbit/s at a 5-MHz clock rate) is even used. For historical reasons, the T = 0 communications command GET RESPONSE is incompatible with ISO/IEC in two regards.[8] If data can be fetched from the terminal using GET RESPONSE, the SIM indicates this by putting '9F' in the SW1 byte, rather than '61' as specified in ISO/IEC 7816-3. GET RESPONSE also has another special feature. According to GSM 11.11, the data provided by the SIM can be fetched a byte at a time using GET RESPONSE, and a transmit buffer pointer is maintained in the SIM for this purpose. This is not possible with ISO/IEC 7816-3, which specifies that the data to be fetched using GET RESPONSE can only be requested starting with the first byte or as an entire block. However, these two incompatibilities do not lead to any problems in practice.

In 1998, on the occasion of the tenth anniversary of the standards for the SIM, the SMG9 published the slogan shown in Figure 13.11. This was the first statement to quite clearly show the significance and size already achieved by the GSM system at that time, as well as the pride felt for one of the essential components of the system, the SIM.

The specifications for the SIM formed the basis for many other specifications for smart cards used in the field of mobile telecommunications. The most important of these specifications are briefly described below.

[8] Strictly speaking, in this case the ISO/IEC 7816-3 standard is *de facto* incompatible with GSM 11.11, since the latter was chronologically first. However, in terms of standardization, an ISO/IEC standard has a higher rank, so GSM is *de jure* incompatible

Billions of Calls
Millions of Subscribers
Thousands of Different Types of Telephones
Hundreds of Countries
Dozens of Manufacturers ...

... and only one Card
The SIM

Figure 13.11 The slogan of the SIM standardization group SMG9 for its tenth anniversary in 1998

In 1992, as part of the European standardization of 'digital enhanced cordless telecommunications' (DECT) by ETSI (DECT is a standard for cordless telephones using cellular technology operating in the 1.9-GHz band), the first version of a specification for the DECT authentication module (DAM) was published. This specification was frozen in 1995 under the ETSI number ETS 300 331. Unfortunately, the DAM is specified as being optional, with the result that it was never converted into an actual product; it fell victim to the cost reduction programs of all manufacturers of cordless telephones.

The digital terrestrial trunked radio system TETRA [TETRA] also has provision for an optional smart card called the TETRA-SIM, whose specifications are based on the SIM for GSM mobile telephones. The EN 300 812 specification for the TETRA-SIM also allows it to be implemented as an application in the UICC if so desired. Since the TETRA-SIM is optional, it can also exist as a software implementation in the terminal device.

Another type of smart card whose specification is based on the SIM is the R-UIM (removable user identity module) in 3G mobile telecommunications systems, which was defined by the Third Generation Partnership Project 2 (3GPP2). The R-UIM is envisaged as an optional component of the associated terminals, and its functionality is similar to that of the SIM. It is specified in the TIA/EIA/IS-820 and TIA/EIA/IS-839 standards. A significant difference between the R-UIM and the SIM is that the former includes the CAVE ('cellular authentication, voice privacy and encryption') cryptographic algorithm, which as its full name suggests, can be used in the R-UIM for a wide variety of cryptographically secured functions. A UIM application toolkit (UATK), which borrows heavily from the SIM application toolkit, has also been specified for the R-UIM.

In the satellite-based Inmarsat mobile telephone network [Inmarsat], which has been in operation since the early 1989s, modified GSM cards are also used now as a means of establishing subscriber identity. Another extension of the SIM, which features a few additional files and a special cryptographic algorithm, is the smart card for the international Iridium mobile telecommunications system [Iridium]. In its ultimate form, this system is intended to consist of 66 satellites orbiting at a height of 780 km that are equivalent to GSM base stations. The frequency used for the air interface between the mobile stations and the satellites is 1616 MHz. The medium- to long-term survival of this technically interesting and unquestionably sophisticated mobile telecommunications system depends on the somewhat precarious financial situation of its operating consortium.

SIM commands

The GSM 11.11 specification defines 22 operational commands for the SIM, which are identified by a class byte value of 'A0'.[9] The commands can be classified into commands related to security, commands for operations on files and commands belonging to the SIM Application Toolkit. Table 13.4 contains a summary of these commands.[10]

Table 13.4 The commands specified for the SIM in GSM 11.11

Command	Brief description
Security commands	
CHANGE CHV	Change the PIN
DISABLE CHV	Disable PIN queries
ENABLE CHV	Enable PIN queries
RUN GSM ALGORITHM	Execute the GSM-specific cryptographic algorithm
UNBLOCK CHV	Reset the PIN retry counter from its terminal count
VERIFY CHV	Verify the PIN
Commands for operations on files	
INCREASE	Increase the value of a counter in a file
INVALIDATE	Reversibly block a file
READ BINARY	Read from a file with a transparent structure
READ RECORD	Read from a file with a record-oriented structure
REHABILITATE	Unblock a file
SEEK	Seek a text string in a file with a record-oriented structure
SELECT	Select a file
STATUS	Read various data from the currently selected file
UPDATE BINARY	Write to a file with a transparent structure
UPDATE RECORD	Write to a file with a record-oriented structure
SIM Application Toolkit commands	
ENVELOPE	Pass data to a value-added service of the SIM forming part of the SIM Application Toolkit
FETCH	Retrieve a SIM Application Toolkit command from the SIM in the mobile equipment
TERMINAL PROFILE	List all functions of the mobile equipment with respect to the SIM Application Toolkit
TERMINAL RESPONSE	Convey the response of the mobile equipment to a previous SIM Application Toolkit command of the SIM
Miscellaneous commands	
GET RESPONSE	Command specific to T = 0 for requesting data from the smart card
SLEEP	Obsolete command for putting the smart card into a low-power state

There is a special feature with regard to entering the four-digit PIN, which incidentally is designated ‘cardholder verification’ (CHV) in GSM. The user can disable further queries for the user PIN by using a special command (DISABLE CHV) together with the proper PIN,

[9] See Section 6.5.1, ‘Structure of the command APDU’, for a description of the command structure
[10] See Chapter 7, ‘Smart Card Commands’, for descriptions of typical smart card commands

thus making it unnecessary to enter the PIN before logging in to a mobile telecommunications network. The disadvantage of this, which is that a lost card can be used illicitly for telephoning until it has been blocked by the network operator, falls under the responsibility of the user. Another command (ENABLE CHV) can be used as desired to again enable PIN queries.

A SIM usually has two CHVs. The idea behind this is to differentiate between the card user and the cardholder, in order to distinguish the functions that can be used or allow only the cardholder to use certain functions. This can be briefly explained using the EF_{FDN} file holding the fixed dialing numbers as an example. The card user only knows CHV 1, which is sufficient for dialing the numbers stored in EF_{FDN}. However, the cardholder also knows CHV 2, so he or she can alter the entries in EF_{FDN}, since the access conditions for UPDATE RECORD require successful verification of CHV 2. One example of how this feature can be used is restricting the numbers that children can call with the mobile telephone to the numbers stored in EF_{FDN}, since they only need to know CHV 1 in order to use the telephone. Their parents, who also know CHV 2, can edit the numbers stored in EF_{FDN}.

For reasons of compatibility, all SIMs also support the SLEEP command, although it has been obsolete for many years. Its original function, which was to save energy in terminal equipment, has now been taken over by the hardware of the smart card microcontroller or the operating system.

The STATUS command is used for two purposes. The first is requesting information about the currently selected file, while the second is verifying that a SIM is present. The mobile equipment periodically sends a STATUS command to the SIM at an interval of approximately 30 seconds, in order to confirm that the SIM is present. If no response to the STATUS command is received from the SIM within five seconds, the SIM is deactivated and the call is terminated. In addition, there is usually some sort of mechanical contact present to detect or prevent exchanging the SIM while the mobile telephone is in use.

The relevant GSM specifications do not specify any administrative commands for file management. Such commands were originally not necessary, since for a long time smart card operating systems did not support creating or deleting files, due to a lack of memory space. However, this situation has fundamentally changed, with the result that these file management functions, which in principle are very important, are now available. They can be used to download files into SIMs at any desired time using remote file management functions, assuming sufficient free memory space is available. The administrative commands are described in the TS 102.222 specification, which originates from the 3GPP environment and was originally conceived for the USIM. However, since the administrative commands for smart card file systems do not exhibit any fundamental differences between SIM and USIM, in practice this standard has also become firmly established in the SIM environment.

SIM files

The SIM has a hierarchical file system, with an MF and two DFs directly below the MF. EFs containing data for the application are located under the MF and in the DFs. The EFs may have transparent, linear fixed or cyclic file structures.

The file identifiers (FIDs) of the SIM files have a special feature, which is that the first byte of each DF under the MF always has the value '7F', DFs directly below the GSM DF have the value '5F' and EFs have the value '5F'. EFs directly below the MF must have the value

Table 13.5 Smart card file management commands specified in TS 102.222

Command	Brief description
ACTIVATE FILE	Unblock a file
CREATE FILE	Create a new file
DEACTIVATE FILE	Reversibly block a file
DELETE FILE	Delete a file
TERMINATE CARD USAGE	Irreversibly block a smart card
TERMINATE DF	Irreversibly block a DF
TERMINATE EF	Irreversibly block an EF

'2F' in the first byte of their FIDs, EFs under the TELECOM DF must have the value '6F' and EFs in an MExE EF must have the value '4F'. These conventions are largely a remnant of the early days of smart card microcontrollers, and they have long since ceased to have any practical significance.

The access conditions for all files are state-oriented and are individually specified for the four file access commands READ, UPDATE, INVALIDATE and REHABILITATE for each file. There are 16 different states for file access, which are numbered from 0 to 15 in increasing order of security. State 0 as an access condition means **always**, which means that the file may always be accessed using the associated command. State 15 represents the other extreme, which is that the file may **never** be accessed using the associated command. State 1 means that access is allowed following successful verification of CHV 1, which means PIN number 1. Similarly, State 2 requires successful verification of CHV 2 prior to access to the file. State 3 is not presently used and is reserved for future use. States 4 through 14 are reserved for administrative use, which means that the network operator can access files using these access conditions with special PINs or authentications.

All EFs containing general information about the smart card, such as a unique card serial number (ICCID), are located directly below the MF. All EFs relevant to the GSM system are located in the TELECOM DF. A typical example of such EFs is the EF containing the abbreviated dialing numbers. The GSM DF, by contrast, contains all EFs holding information specific to the network operator, such as the IMSI.

In total, 70 different EFs are defined in the GSM 11.11 specification, of which only 12 (with a total content of approximately 110 bytes of data) are obligatory. The rest of the EFs are optional, so their presence in the file system of the SIM depends on the network operator and the services that are provided. In addition to the files defined in the specification, the network operator can place its own files in the SIM file tree for maintenance or administrative purposes. In practice, intensive use is made of this possibility, with the result that typically around 40 files containing approximately 12 kB of user data are present in the SIM.

Some of the EFs in the file tree of the SIM must be written especially often. One example is the LOCI (location information) EF. This file stores the currently valid temporary mobile subscriber identity (TMSI) along with the supplementary location area information (LAI). The data in this file must be modified for each change in base station and each new call. Consequently, a SIM operating system must support a special file attribute called ‘high update activity’. The technical implementation of this involves storing several file bodies under a

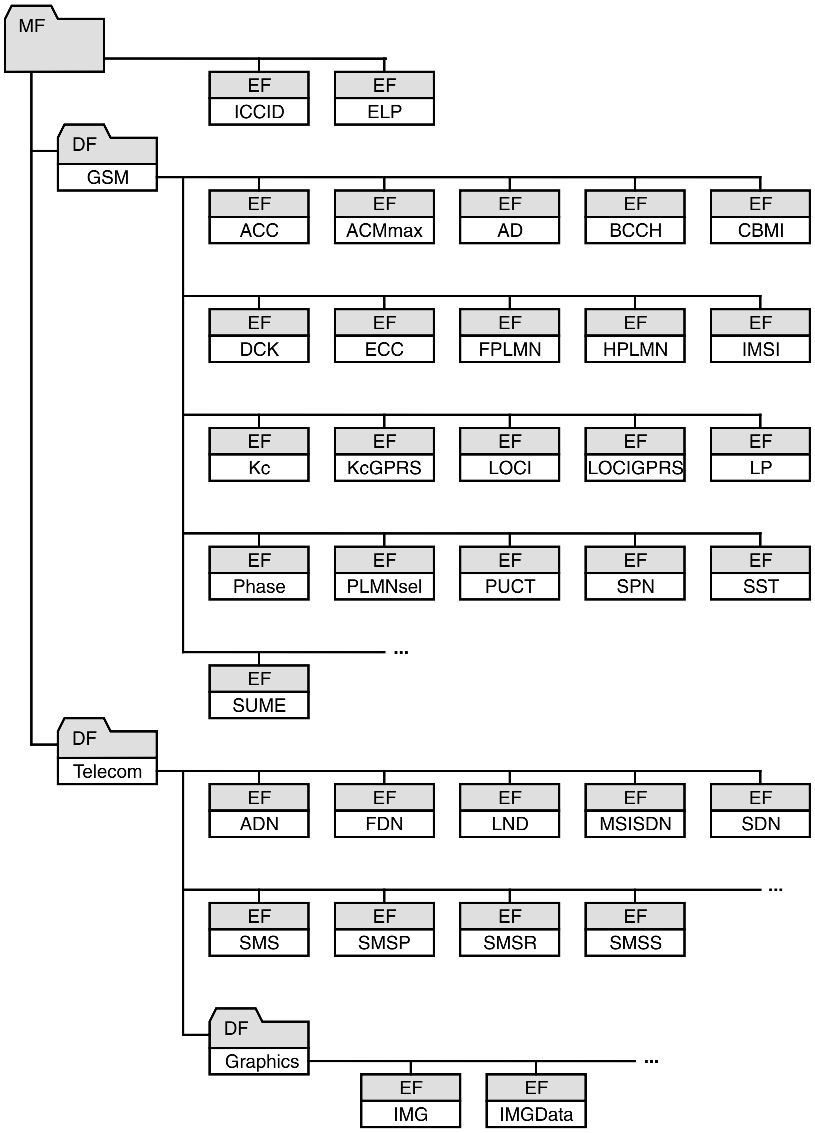


Figure 13.12 Overview of the most important SIM files

single file header. If an error occurs in a file body, the operating system automatically switches to a replacement file body. This file attribute came into existence at a time when EEPROM pages had only 10,000 guaranteed write/erase cycles. However, technical refinements have increased the number of cycles to around half a million, which means that this attribute has effectively become obsolete. Nevertheless, it is still present in the GSM specification, although

Table 13.6 Typical SIM files according to GSM 11.11, with the coding of the data elements and illustrative decoded examples

MF		**Root directory**
	Description:	This is the source directory for the entire SIM.
	File:	FID = '3F00'
DF$_{\text{TELECOM}}$		**Telecom directory**
	Description:	This directory holds all files specific to the services.
	File:	FID = '7F10'
DF$_{\text{GSM}}$		**GSM directory**
	Description:	This directory holds all files specific to the GSM network.
	File:	FID = '7F20' (or '7F21' for compatibility with older-model GSM 1800 mobile telephones)
DF$_{\text{GRAPHICS}}$		**Graphics directory**
	Description:	This directory holds all files containing graphics information.
	File:	FID = '5F50'
MF.EF$_{\text{ELP}}$		**Extended language preference (ELP)**
	Description:	This file holds an extended list of the preferred languages for the user interface.
	File:	FID = '2F05'; structure: transparent, file size: 2*n* bytes; accesses: READ: always, UPDATE: CHV 1
	Coding:	Each country code consists of two alphanumeric characters according to ISO 639, using the GSM 03.38 alphabet. bytes 1& 2: highest-priority language . . . bytes 2 (*n* – 1) & 2*n*: lowest-priority language
	Example:	'64 65' ⇒ highest-priority language: German '65 6E' ⇒ second highest-priority language: English '66 72' ⇒ third highest-priority language: French '65 73' ⇒ lowest-priority language: Spanish

Table 13.6 (*Cont.*)

MF.EF$_{\text{ICCID}}$	**ICC identification (ICCID)**
Description:	This file holds a unique identification number for the smart card.
File:	FID = '2FE2'; structure: transparent, file size: 10 bytes; accesses: READ: always, UPDATE: never
Coding:	sequential number, BCD-coded, left-justified and padded with 'F' as necessary byte 1, bits 1– 4: digit 1 byte 1, bits 5–8: digit 2 byte 2, bits 1–4: digit 3 etc.
Example:	'98 94 20 00 00 10 81 85 39 11' ⇒ 89 49 02 00 00 01 18 58 93 11
DF$_{\text{GSM}}$.EF$_{\text{ACM}}$	**Accumulated call meter (ACM)**
Description:	This file holds the number of call charge units accumulated since a particular starting time.
File:	FID = '6F39'; structure: cyclic, 3*n* bytes; accesses: READ: CHV 1; UPDATE: CHV 2
Coding:	bytes 1–3: accumulated number of call charge units
DF$_{\text{GSM}}$.EF$_{\text{ACMmax}}$	**Accumulated call meter maximum (ACM)**
Description:	This file holds the maximum amount of call charge units.
File:	FID = '6F37'; structure: transparent, 3 bytes; accesses: READ: CHV 1; UPDATE: CHV 2
Coding:	bytes 1 –3: maximum amount of call charge units
DF$_{\text{GSM}}$.EF$_{\text{FPLMN}}$	**Forbidden public land mobile network (FPLMN)**
Description:	This file holds a list of forbidden network operators.
File:	FID = '6F7B'; structure: transparent, 12 bytes; accesses: READ: CHV 1; UPDATE: CHV 1
Coding:	bytes 1–3: forbidden PLMN no. 1 bytes 4–6: forbidden PLMN no. 2; etc. See EF$_{\text{PLMNsel}}$ for the data structure and an example.

(*Cont.*)

Table 13.6 (*Cont.*)

Example:	'FF FF FF FF FF FF FF FF FF 62 F2 20' '62 F2' ⇒ MCC ⇒ '262' ⇒ Germany '10' ⇒ MNC ⇒ '01' ⇒ Germany D1
$DF_{GSM}.EF_{HPLMN}$	**Home public land mobile network search period (HPLMN)**
Description:	This file holds a time interval for searching for the home network.
File: Coding:	FID = '6F31'; structure: transparent, 1 byte; accesses: READ: CHV 1; UPDATE: ADM Time interval for searching for the home network in minutes; coding: per GSM 02.11
Example:	'05' ⇒ search for home network every 5 minutes
$DF_{GSM}.EF_{IMSI}$	**International mobile subscriber identity (IMSI)**
Description:	This file holds the international subscriber identity number.
File:	FID = '6F07'; structure: transparent, file size: 9 bytes; accesses: READ: CHV 1; UPDATE: ADM
Coding:	byte 1: length of the IMSI in bytes byte 2, bits 1–3: °100° byte 2, bit 4: parity of the IMSI, coded per GSM 04.08 byte 2, bits 5 –8: digit 1 of the IMSI bytes 3–9: digits 2 –10 of the IMSI IMSI = MCC || MNC || serial number of the network operator, BCD-coded and right-padded with 'F' as necessary Coding: from MCC; for MNC see $EF_{PLMNsel}$
Example:	'08 92 62 01 71 00 10 92 67' '08' ⇒ length ⇒ 8 bytes '9' || '2 62' ⇒ MCC ⇒ Germany '01' ⇒ MNC ⇒ Germany D1 '71 00 10 92 67' ⇒ serial number of the network operator
$DF_{GSM}.EF_{KC}$	**Kc key**
Description: File:	This file holds the key Kc for encrypting data on the air interface. FID = '6F20'; structure: transparent, file size: 9 bytes; accesses: READ: CHV 1; UPDATE: CHV 1

Table 13.6 (*Cont.*)

	Coding:	bytes 1–8: key Kc byte 9, bits 1–3: serial number of the key
DF$_{GSM}$.EF$_{LOCI}$		**Location information (LOCI)**
	Description:	This file holds current location information for the mobile telephone.
	File:	FID = '6F7E'; structure: transparent, file size: 11 bytes; accesses: READ: CHV 1; UPDATE: CHV 1
	Coding:	bytes 1–4: TMSI (temporary mobile subscriber identity) bytes 5–9: LAI (location area information) byte 10: TMSI TIME (not used from Phase 2 onwards) byte 11: Location update status: b3–b1 = °000°: updated b3–b1 = °001°: not updated b3–b1 = °010°: forbidden PLMN b3–b1 = °011°: forbidden location area Coding: per GSM 04.08
	Example:	'5F 40 96 46 62 F2 10 80 04 FF 00' '5F 40 96 46' ⇒ TMSI '62 F2 10 80 04' ⇒ LAI 'FF' ⇒ TMSI TIME ⇒ not used '00' ⇒ location update status ⇒ updated
DF$_{GSM}$.EF$_{LP}$		**Language preference (LP)**
	Description:	This file holds a list of the preferred languages for the user interface.
	File:	FID = '6F05'; structure: transparent, file size: n bytes, $n \geq 1$; accesses: READ: always, UPDATE: CHV 1
	Coding:	per GSM 03.38 byte 1: highest-priority language . . . byte n: lowest-priority language
	Sample languages:	Valid for the GSM alphabet according to GSM 03.38 '00': German '01': English '02': Italian '03': French '04': Spanish '05': Dutch '06': Swedish '07': Danish '08': Portuguese '09': Finnish '0A': Norwegian '0B': Greek '0C': Turkish '0D': Hungarian '0E': Polish '0F': Unspecified language

(*Cont.*)

Table 13.6 (*Cont.*)

Example:	'00 01 03 05' '00' ⇒ highest-priority language: German '01' ⇒ second highest-priority language: English '03' ⇒ third highest-priority language: French '05' ⇒ lowest-priority language: Dutch
$\text{DF}_{\text{GSM}}.\text{EF}_{\text{PHASE}}$	**Phase information**
Description:	This file holds information about the phase supported by the SIM.
File:	FID = '6FAE'; structure: transparent, file size: 1 byte; accesses: READ: always, UPDATE: ADM
Coding:	byte 1: 00 = Phase 1; 02 = Phase 2
Example:	'02' ⇒ Phase 2
$\text{DF}_{\text{GSM}}.\text{EF}_{\text{PLMNsel}}$	**Public land mobile network selector (PLMNsel)**
Description:	This file holds a list of the preferred network operators.
File:	FID = '6F30'; structure: transparent, 3*n* bytes ($n \geq 8$); accesses: READ: CHV 1; UPDATE: CHV 1
Coding:	bytes 1–3: PLMN for the highest selection priority bytes 4–6: PLMN for the second-highest selection priority Data structure: 2 bytes MCC (mobile country code) || 1 byte MNC (mobile network code), BCD-coded per GSM 04.08, high and low nibbles swapped; 'FFFFFF' ⇒ entry not used
Sample MCC codes:	262: Germany 208: France 234: Great Britain 222: Italy 232: Austria 310: USA
Sample MNC codes for Germany:	01: Germany D1 02: Germany D2 03: Germany E-plus 07: Germany Viag Intercom

Table 13.6 (*Cont.*)

Example:	'62 F2 20 72 F0 10 32 F4 01 32 F2 30 32 F0 10 62 F2 10 62 F0 20 42 F0 10 22 F8 10', remainder 'FF' '62 F2' ⇒ MCC ⇒ '262' ⇒ Germany '20' ⇒ MNC ⇒ '02' ⇒ Germany D2 etc.
$DF_{GSM}.EF_{PUCT}$	**Price per unit and currency table (PUCT)**
Description:	This file holds the price per call unit and the currency, for the current summary of call charges.
File:	FID = '6F41'; structure: transparent, file size: 5 bytes; accesses: READ: CHV 1; UPDATE: CHV 1 or CHV 2
Coding:	bytes 1 –3: currency code, character coded using the GSM alphabet bytes 4 & 5: price per unit = EPPU$\times$10 ^{EX} EPPU: elementary price per unit; EX: exponent EPPU component: B5.b1: 2^0 B5.b2: 2^1 B5.b3: 2^2 B5.b4: 2^3 B4.b1: 2^4 B4.b2: 2^5 B4.b3: 2^6 B4.b4: 2^7 B4.b5: 2^8 B4.b6: 2^9 B4.b7: 2^{10} B4.b8: 2^{11} Exponent component (EX): B5.b6: 2^0 B5.b7: 2^1 B5.b8: 2^2 B5.b5: sign of the exponent: 0: +, 1: –
Examples:	'44 45 4D 01 57' '44 45 52' ⇒ currency code ⇒ "EUR" '01 57' = °0000 0001° || °0101 0001° ⇒ price per unit ⇒ $17 \times 10^{-2} = 0.17$
$DF_{GSM}.EF_{SPN}$	**Service provider name (SPN)**
Description:	This file holds the name of the service provider.
File:	FID = '6F46'; structure: transparent, file size: 17 bytes; accesses: READ: always, UPDATE: ADM
Coding:	byte 1: conditions for display '00': display of PLMN name not required '01': display of PLMN name required bytes 2–17: service provider name, coded per GSM 03.38, left-justified and right-padded with 'F' as necessary
Example:	'01 50 72 6F 76 69 64 65 72 20 41' '01' ⇒ display of PLMN name required '50 72 6F 76 69 64 65 72 20 41' ⇒name of service provider ⇒ "Provider A"

(*Cont.*)

Table 13.6 (*Cont.*)

$DF_{GSM}.EF_{SST}$	**SIM service table (SST)**
Description:	This file holds a table of available and activated services supplementary to the voice service.
File:	FID = '6F38'; structure: transparent, file size: $\geq$ 2 bytes; accesses: READ: CHV 1; UPDATE: ADM
Coding:	byte 1, bits 1 & 2: service no. 1 byte 1, bits 3 & 4: service no. 2 byte 1, bits 5 & 6: service no. 3 byte 1, bits 7 & 8: service no. 4 byte 2, bits 1 & 2: service no. 5 etc. Bit coding: b1, b3, b5, b7 = 1 / 0: service available / not activated b2, b4, b6, b8 = 1 / 0: service enabled / not activated
Sample services:	Service no. 1: disable CHV testing Service no. 2: abbreviated dialing numbers (ADN) Service no. 3: fixed dialing numbers (FDN) Service no. 4: short message service (SMS) Service no. 18: service dialing numbers (SDN) Service no. 35: status report for short messages Service no. 38: GPRS Service no. 39: image (IMG)
Example:	'DF 3F DF FF 03' = °1101 1111° || °0011 1111° || °1101 1111° || °1111 1111° || °0000 0011° °11° $\Rightarrow$ disable PIN available and activated °11° $\Rightarrow$ abbreviated dialing numbers available and activated °01° $\Rightarrow$ fixed dialing numbers available and not activated °11° $\Rightarrow$ short message service available and activated etc.
$DF_{GSM}.DF_{GRAPHICS}.EF_{IMG}$	**Image (IMG)**
Description:	This file holds references to files containing graphics that can be shown on the display of the mobile telephone.
File:	FID = '4F20'; structure: linear fixed, ($9n + 2$) bytes; accesses: READ: CHV 1; UPDATE: ADM

Table 13.6 (*Cont.*)

Coding:	byte 1: number of references to image files bytes 2–10: description of the reference to image file 1 bytes 11 –19: description of the reference to image file 2 . . . byte $9n + 2$: RFU
Coding of the references:	byte 1: width of the image in pixels byte 2: height of the image in pixels byte 3: image coding scheme bytes 4 & 5: FID of $EF_{\textbf{IMGData}}$ bytes 6 & 7: offset to the image data in $EF_{\textbf{IMGData}}$ bytes 8 & 9: size to the image data in $EF_{\textbf{IMGData}}$ in bytes
$DF_{GSM}.DF_{GRAPHICS}.EF_{IMGDattaX}$	**Image data (IMGData)**
Description:	Each of these files holds a bitmapped graphic that can be shown on the display of the mobile telephone.
File:	FID = '4Fxx'; structure: transparent, n bytes; accesses: READ: CHV 1; UPDATE: ADM
Coding:	bytes 1 – n: image data
$DF_{TELECOM}.EF_{ADN}$	**Abbreviated dialing numbers (ADN)**
Description:	This file holds the abbreviated dialing numbers. Each record contains a name and the associated dialing number.
File:	FID = '6F3A'; structure: linear fixed, record size: $n + 14$ bytes; accesses: READ: CHV 1; UPDATE: CHV 1
Coding:	bytes 1 – n: name coded in characters per GSM 03.38 byte $n + 1$: length of the BCD-coded dialing number in bytes byte $n + 2$: type of dialing number, coded per GSM 04.08 e.g.: '81' = unknown type of dialing number, ISDN dialing number scheme '91' = international type of dialing number, ISDN dialing number scheme bytes $(n + 3) – (n + 12)$: BCD-coded dialing number with upper and lower nibbles swapped in byte bytes $(n + 13) – (n + 14)$: pointer to supplementary data for this entry in $EF_{\textbf{CCP}}$ and EF_{EXT1}, generally not used (i.e. 'FF') Unused bytes are set to 'FF'

(*Cont.*)

Table 13.6 (*Cont.*)

Example 1:	Record content: '57 4F 4C 46 47 41 4E 47 FF FF FF FF
	FF FF FF FF 07 91 94 98 69 35 24 46 FF FF FF FF FF FF'
	'57 4F 4C 46 47 41 4E 47' ⇒ "Wolfgang"
	'FF FF FF FF FF FF FF FF' ⇒ not used
	'07' ⇒ length of the dialing number (7 bytes)
	'91' ⇒ international dialing number, ISDN dialing number scheme
	'94 98 69 35 24 46' ⇒ dialing number 49 89 96 53 42 64
	'FF FF FF FF' ⇒ not used
	'FF FF' ⇒ EF_{CCP} and EF_{EXT1} not used
Example 2:	Record content: '57 4F 4C 46 47 41 4E 47 FF FF FF FF
	FF FF FF FF 07 91 94 98 69: '57 4F 4C 46 47 41 4E 47
	FF FF FF FF FF FF FF FF 07 81 80 99 56 43 62
	F4 FF FF FF FF FF FF'
	'57 4F 4C 46 47 41 4E 47' ⇒ "Wolfgang"
	'FF FF FF FF FF FF FF FF' ⇒ not used
	'07' ⇒ length of the dialing number (7 bytes)
	'81' ⇒ unknown type of dialing number, ISDN dialing number scheme
	'80 99 56 43 62 F4' ⇒ dialing number 089 96 53 42 64
	'FF FF FF FF' ⇒ not used
	'FF FF' ⇒ EF_{CCP} and EF_{EXT1} not used
$DF_{TELECOM}.EF_{FDN}$	**Fixed dialing numbers (FDN)**
Description:	Fixed dialing numbers can be stored in this file as needed. These dialing numbers are used when the subscriber is only allowed to dial certain numbers.
File:	FID = '6F3B'; structure: linear fixed, record size: $(n + 14)$ bytes; accesses: READ: CHV 1; UPDATE: CHV 2
Coding:	same as EFADN
Example:	see EFADN
$DF_{TELECOM}.EF_{LND}$	**Last number dialed (LND)**
Description:	The most recently dialed numbers are stored in this file.
File:	(optional file) FID = '6F44'; structure: cyclic, record size: $(n + 14)$ bytes; accesses: READ: CHV 1; UPDATE: CHV 1
Coding:	same as EF_{ADN}

Table 13.6 (*Cont.*)

$\text{DF}_{\text{TELECOM}}.\text{EF}_{\text{MSISDN}}$		**Mobile station ISDN number (MSISDN)**
	Description:	This file holds the dialing number of the mobile station.
	File:	FID = '6F40'; structure: linear fixed, record size: ($n + 14$) bytes; accesses: READ: CHV 1; UPDATE: CHV 1
	Coding:	same as EFADN
$\text{DF}_{\text{TELECOM}}.\text{EF}_{\text{SDN}}$		**Service dialling numbers (SDN)**
	Description:	This file holds the service dialing numbers, which may for example be dialing numbers for directory information or schedule information.
	File:	FID = '6F49'; structure: linear fixed, record size: ($n + 14$) bytes; accesses: READ: CHV 1; UPDATE: ADM
	Coding:	same as EFADN
$\text{DF}_{\text{TELECOM}}.\text{EF}_{\text{SMS}}$		**Short message service (SMS)**
	Description:	This file belongs to the short message service. It holds the short messages sent to and received from the network.
	File:	FID = '6F3C'; structure: linear fixed, record size: 176 bytes; accesses: READ: CHV 1; UPDATE: CHV 1
	Coding:	byte 1: status of the record in question: '00' = free record; '01' = message coming from the network and read; '03' = message coming from the network and still to be read; '05' = message sent to the network; '07' = message to be sent to the network; bytes 2–176: message coded per GSM 03.40; unused bytes at the end of the message are set to 'FF'

(*Cont.*)

Table 13.6 (*Cont.*)

Coding of a message from the network to the mobile telephone	byte 2:	number of bytes in the SMSC dialing number, including the dialing number type
	next 2–12 bytes:	SMSC dialing number: '81' = unknown type of dialing number (no "+"), '91' = international type of dialing number ("+"), data nibblewise swapped
	next byte:	control information (generally '04')
	next byte:	number of digits in the dialing number of the sender, excluding the dialing number type
	next 2–12 bytes:	dialing number of the sender, with data nibblewise swapped
	next byte:	protocol tag ('00' = text message)
	next byte:	data coding ('00' = GSM standard alphabet)
	next 7 bytes:	SMSC time stamp, data nibblewise swapped: year || month || day || hours || minutes || seconds || time zone ('00' = GMT)
	next byte:	number of characters in the message
	next 1 –140 bytes:	message (if the GSM standard alphabet is used, the text portion is compressed, which means the 7-bit codes are continuously packed into bytes)
Coding of a message from the mobile telephone to the network	byte 2:	number of bytes in the SMSC dialing number, including the dialing number type
	next 2–12 bytes:	SMSC dialing number: '81' = unknown type of dialing number, (no "+") '91' = international type of dialing number, ("+"), data nibblewise swapped
	next byte:	relative time of the mobile telephone (generally 'FF')
	next byte:	message reference
	next 2–12 bytes:	dialing number of the destination, with data nibblewise swapped
	next byte:	protocol tag ('00' = text message)
	next byte:	data coding ('00' = GSM standard alphabet)
	next X bytes:	term of validity of the message: 1–143: t = (X + 1) × 5 min; 144–167: t = 12 h + (X – 143) × 30 min; 168–196: t = (X – 166) × 1 day; 197–255: t = (X – 192) × 1 week
	next byte:	number of characters in the message
Sample SMS message from the network to a mobile telephone	'01 07 91 94 71 01 67 05 00 04 0C 91 94 71 71 46 53 42 00 00 00 60 52 31 63 15 00 17 C8 A0 93 28 AC 0E 91 20 62 51 0A 1A 22 93 D0 65 50 4A 2D 3A 01' || remainder of record is 'FF' '01' ⇒ message coming from the network and read '07' ⇒ number of bytes in the SMSC dialing number, including the dialing number type	

Table 13.6 (*Cont.*)

	'91 94 71 01 67 05 00' ⇒ SMSC dialing number = +49 17 10 76 50 00 '04' ⇒ no further messages '0C' ⇒ 12 ⇒ number of digits in the dialing number of the sender, excluding the dialing number type, is 12 '91 94 71 71 46 53 42' ⇒ sender dialing number = +49 17 17 64 35 24 '00' ⇒ test message '00' ⇒ GSM standard alphabet '00 60 52 31 63 15 00' ⇒ SMSC time stamp = 00 06 25 13 36 51 00 ⇒ 25.06.0013 : 36 : 51,time zone 0 (GMT) '17' ⇒ 23 ⇒ number of characters in the message is 23 'C8 A0 93 28 AC 0E 91 20 62 51 0A 1A 22 93 D0 65 50 4A 2D 3A 01' ⇒ message: "Handbuch der Chipkarten"
Sample SMS message from a mobile telephone to the network	'07 02 81 F0 11 FF 00 81 00 00 00 08 D7 27 D3 78 0C 3A 8F FF' || remainder of record is 'FF' '07' ⇒ message to be sent to the network '02' ⇒ number of bytes in the dialing number, including this length specification ☎ '81' ⇒ unknown dialing number ☎ 'F0' ⇒ control information '11' ⇒ relative time of mobile telephone 'FF' ⇒ message reference ☎ '00' ⇒ length of the dialing number of the destination = 0 ☎ '81' ⇒ unknown dialing number ☎ '00' ⇒ test message '00' ⇒ GSM standard alphabet '00' ⇒ validity interval ☎ '08' ⇒ number of characters in the message is 8 'D7 27 D3 78 0C 3A 8F FF' ⇒ message: "WOLFGANG" Note 1: The record structure depends on the implementation in the actual mobile telephone and is not universally valid. Note 2: After this SMS record has been read from the SIM, the data elements above marked with ☎ are expanded before being sent from the mobile telephone. After the message has been sent to the network, the first byte of this data set is changed from ‘07’ to ‘05’.
$\text{DF}_{\text{TELECOM}}.\text{EF}_{\text{SMSP}}$	**Short message service parameters (SMSP)**
Description:	This file belongs to the short message service. It holds the settings for sending short messages.

File:	FID = '6F42'; structure: linear fixed, record size: $(28 + n)$ bytes;
	accesses: READ: CHV 1;
	UPDATE: CHV 1

(*Cont.*)

Table 13.6 (*Cont.*)

$DF_{TELECOM}.EF_{SMSS}$	**Short message service status (SMSS)**
Description:	This file belongs to the short message service. It holds the status of the stored short messages.
File:	FID = '6F43'; structure: linear fixed, record size: $(2 + n)$ bytes; accesses: READ: CHV 1; UPDATE: CHV 1
Coding:	byte 1: last used SMS message reference number per GSM 03.40 byte 2: b1 = 0: no space for the message in the SIM memory b1 = 1: enough space for the message in the SIM memory b2 – b7: RFU; set to '1'
Example:	'70 FF' '70' $\Rightarrow$ last used SMS message reference number 'FF' $\Rightarrow$ memory space available in the SIM

current smart card operating systems do not treat files having this attribute any differently than files that do not have it.

It was originally planned to replace GSM smart cards every two years in order to avoid failures due to the limited number of EEPROM write/erase cycles. However, since practically no problems have arisen in this regard up to now, most network operators replace smart cards only in the event of actual failure. This yields considerable cost savings for the provider, since his logistics only have to deal with replacing defective cards. The number of cards that have to be replaced is also considerably reduced by the fact that the useful life of most cards is significantly longer than two years. This markedly decreases procurement costs, since it is only necessary to replace smart cards when they no longer work properly. Practical experience has shown that cards must be replaced every five to seven years.

Authenticating the SIM

Besides storing data, one of the primary functions of the SIM is performing authentication with respect to the GSM network. This involves a unilateral authentication of the SIM by the background system. The SIM thus does not test whether the background system is authentic; instead, the background system only tests whether the SIM is authentic. If the authenticity of the SIM is confirmed, the network operator knows that it can bill the call to the owner of the mobile telephone. However, this unilateral authentication has the disadvantage that the user of the mobile telephone cannot be certain that he is connected to an authentic network instead of a counterfeit network. As a consequence, it is possible to eavesdrop on calls using a suitable piece of equipment, called an IMSI catcher, without knowing the secret keys. The operating principle of the IMSI catcher is based on having the device establish its own radio cell by acting as a counterfeit base station, which allows it to interpose itself in the air interface between a genuine base station and the mobile telephones by representing itself as a base station to the mobile telephones and as a mobile telephone to the base station. Such an attack would not be possible with mutual authentication followed by encryption of all call data between the SIM and the background system.

The SIM is identified using a number that is unique within the entire GSM system. This number, which has a maximum length of eight bytes, is called the ‘international mobile subscriber identity’ (IMSI). The subscriber can be identified using the IMSI in all GSM networks throughout the world. In order to keep the identity of the subscriber as confidential as possible within the network, whenever possible a temporary mobile subscriber identity (TMSI) is used instead of the IMSI. The TMSI is generated from the visitor location register (VLR) and is thus valid only within a portion of the GSM network in question. Nevertheless, in combination with the location area information (LAI) the TMSI is unique within the entire GSM network. For all further identification transactions, only the TMSI is used once it has been assigned. The relationship between the IMSI and the TMSI is stored in the visitor location register (VLR) for the duration of its actual use. In the exceptional case that the TMSI is not known in the VLR, the IMSI must be transmitted in cleartext over the air interface in order to identify the subscriber.

The card-specific keys for authentication and encrypting data on the air interface can be derived from the IMSI. However, the SIM cannot encrypt data for the air interface, since the processing and data transmission capacity of a smart card are not adequate for real-time encryption of voice data. Instead, the SIM computes a derived temporary key for transmission encryption and passes it to the mobile equipment. The mobile equipment has a high-performance encryption unit in the form of a signal processor, which can encrypt and decrypt voice data on the air interface in real time. The encrypted data on the air interface are usually decrypted back into cleartext by the base station controller (BSC).

If a subscriber wishes to make a call, his mobile telephone establishes a connection to the base station with the best reception and gives it the TMSI from the SIM memory along with the LAI, or in exceptional cases the IMSI. If the subscriber is located in the region of his or her home network, a ‘triple’ of authentication and encryption data is generated by the authentication center (AuC). This data set includes the ciphering key (Kc) for encrypting data on the air interface, a random number (RAND) and the resulting signed response (SRES). The advantage of this procedure is that the secret individual key (Ki) and the authentication algorithm, which is partly confidential, never have to leave the authentication center. This triple is then passed to the home location register (HLR).

If the mobile telephone is logged in to its home network, the triple (Kc, RAND and SRES) is sent to the appropriate visitor location register (VLR). There the result of encrypting the random number (SRES) is requested from the SIM by the mobile switching center (MSC) and compared with the result received from the AuC (SRES’). If the two results match, the SIM has been authenticated and the system can start encrypting the data on the air interface using the A5 cryptographic algorithm and associated key (Kc).

On the other hand, if the mobile telephone is logged in to a foreign network the triple is passed to the foreign network, where it can be used in the same manner as in the home network. This situation clearly shows the cleverness of this authentication and encryption scheme, since the A3 and A5 cryptographic algorithms are specific to individual network operators and cannot be computed in a foreign network, even if the secret key is known. Only the A5 cryptographic algorithm, which is used for encrypting data on the air interface, is common throughout the GSM system, in order to allow these data to be given suitable cryptographic protection if the key Kc is known.

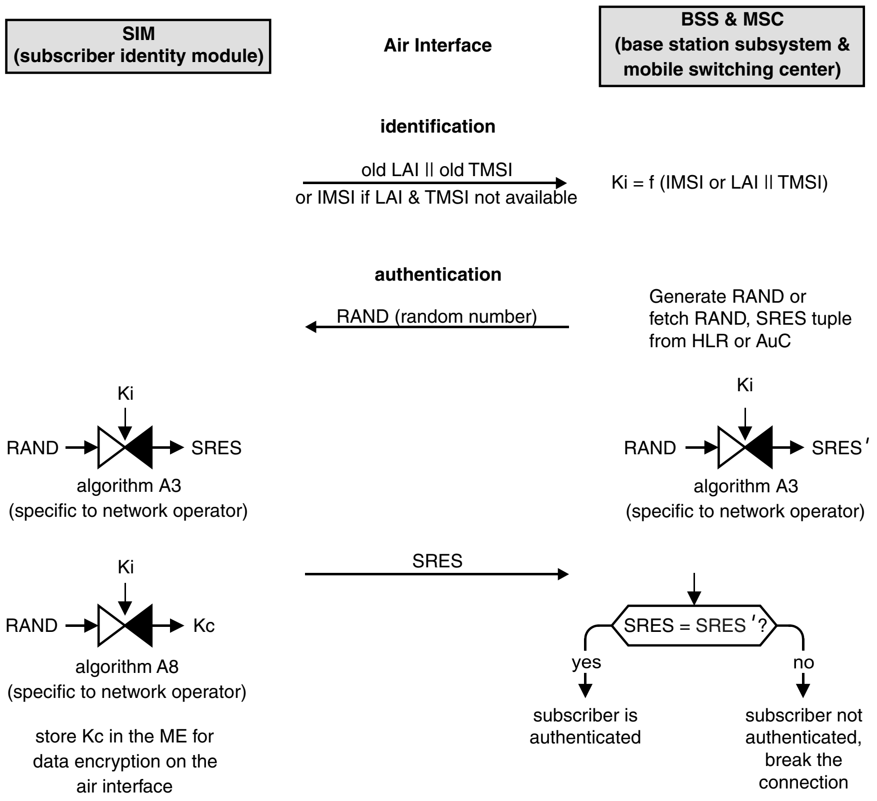

SIM
(subscriber identity module)
Air Interface
BSS & MSC
(base station subsystem & mobile switching center)
identification
old LAI || old TMSI
or IMSI if LAI & TMSI not available
Ki = f (IMSI or LAI || TMSI)
authentication
RAND (random number)
Generate RAND or fetch RAND, SRES tuple from HLR or AuC
Ki
RAND
SRES
algorithm A3
(specific to network operator)
Ki
RAND
SRES'
algorithm A3
(specific to network operator)
SRES
Ki
RAND
Kc
algorithm A8
(specific to network operator)
store Kc in the ME for data encryption on the air interface
SRES = SRES '?
yes
no
subscriber is authenticated
subscriber not authenticated, break the connection

Figure 13.13 Procedure for the identification and subsequent authentication of the SIM by the GSM background system using the A3 and A8 cryptographic algorithms, which are specific to the individual network operator. Key Kc is later used for encrypting the data transmitted between the mobile station and the base station via the air interface

The cryptographic algorithms used in the GSM system are generally confidential, which is the only departure from Kerckhoff's principle[11] in this system. All other information about the system is publicly accessible. Originally, an algorithm called COPM 128 was often used for the A3 and A8 cryptographic algorithms, which are specific to individual network operators. However, this algorithm was cracked in 1998, since its key was too short. In retrospect, this shows the value of Kerckhoff's principle, since cryptologists would have probably recognized that the key was too short if the algorithm had been made public. The COMP 128 cryptographic algorithm is still presently used, but in an improved form called COMP 128-2. The A5 cryptographic algorithm, which is the same throughout the GSM system, is a stream cipher consisting of three linear feedback shift registers (LFSRs) with lengths of 19, 22 and 23 [Anderson 01], incremented by the TDMA frame number.

[11] See also Section 4.7, 'Cryptology'

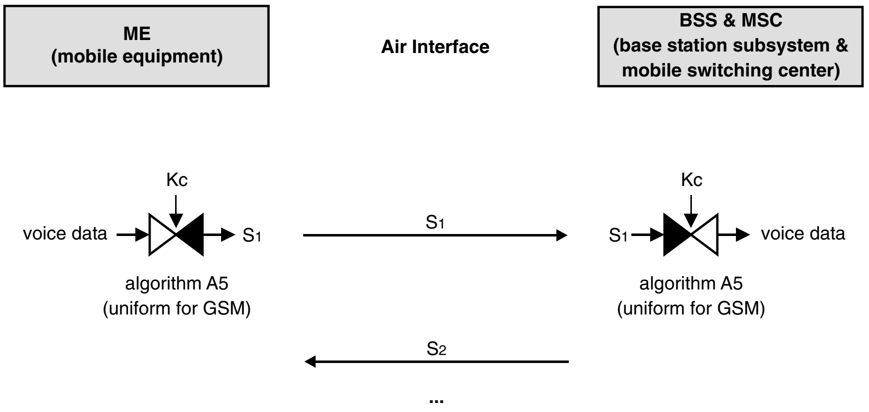


Figure 13.14 Data transmitted between the mobile station and the base station via the air interface are encrypted using the A5 cryptographic algorithm and the secret key Kc. This process must be preceded by authentication of the SIM by the GSM background system

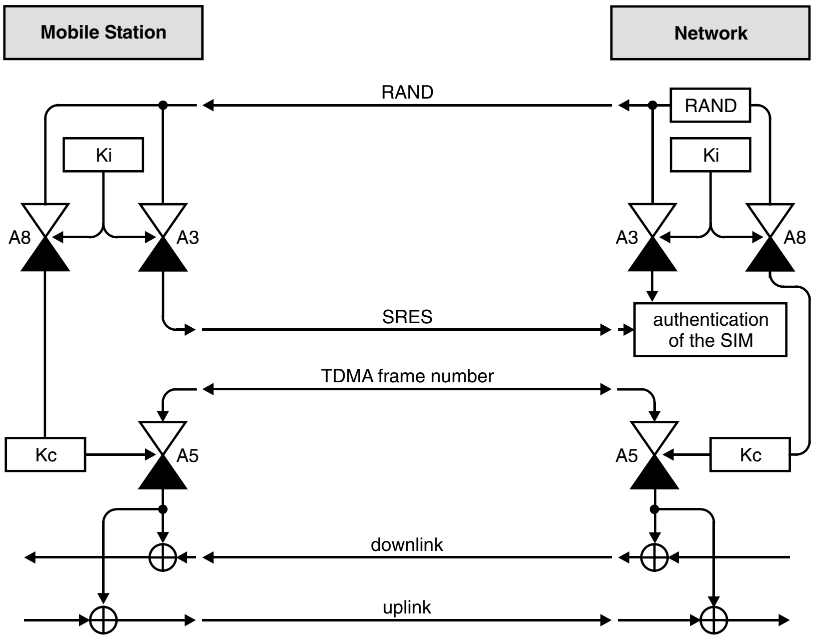


Figure 13.15 Functional overview of the cryptographic functions of the SIM, mobile equipment and background system in the GSM system

Switch-on and switch-off procedures for the mobile telephone

The procedures associated with switching a mobile telephone on and off are briefly described below, with a strong focus on the role of the SIM.

When the mobile telephone is switched on, hardware self-tests are first run and the operating system, which occupies several megabytes, is then started up. In order to distract the impatient user during the several seconds taken by this process, entertaining animations are often shown on the display. Once the operating system is fully active, one of the next steps is to initiate the activation sequence for the SIM. The activation sequence is followed by several measures for configuring the optimum transmission parameters, such as analyzing the ATR and executing a PPS procedure.[12] Following this, it is now common practice to create a virtual SIM in the memory of the mobile telephone. For this purpose, the mobile telephone reads a large amount of data from the files in the SIM, such as the abbreviated dialing numbers, and stores them in appropriate data fields in the mobile telephone. The objective of this is to ensure fast read and write access to the SIM data, which would otherwise not be possible due to the low data transmission rate between the mobile telephone and the SIM and the amount of time taken for EEPROM write accesses. Consequently, most mobile telephones primarily work with the copies of the SIM data that are located in their memories. Of course, this technique cannot be used with all of the data in the SIM. Activities such as PIN verification and authentication must always be performed in combination with the SIM, since the data needed for these activities are not allowed to leave the SIM.

One of the side effects of using a virtual SIM is that it considerably increases the life expectancy of the SIM, since a large number of EEPROM write accesses that would otherwise be necessary simply never occur. The stress on certain files within the SIM resulting from frequent write accesses is thereby considerably reduced. A typical example of this is the EF_{LOCI} file, which contains information about the current location of the mobile telephone. The EEPROM locations containing this file are especially heavily stressed in mobile telephones that frequently change GSM cells, for which reason this file has the attribute 'high update activity'. If the data in this file are primarily updated in the RAM of the mobile telephone, the problem of an excessive number of write accesses to the EEPROM of the SIM is rendered insignificant.

The data in the virtual SIM in the memory of the mobile telephone are written back to the SIM following critical operations, so the data stored in the files in the SIM are again current data following the writeback operation. This is frequently performed asynchronously by the mobile station using a low-priority operating system task, so the user is not aware that it is happening. Updating the SIM at critical points in time is also important because the SIM should always hold essentially current data in its EEPROM in the event of a sudden loss of power, such as may happen when the batteries are removed. For instance, it would be extremely annoying if removing the batteries resulted in the loss of all of the dialing numbers painstakingly entered into the telephone since the last time the mobile telephone was switched on.

When the mobile telephone is switched off, the user usually sees only a brief sequence of animated characters on the display. However, all the files in the virtual SIM are written to the physical SIM while this is happening, in order to bring it up to date. After this, a SIM

[12] See also Section 6.2, 'Answer to Reset (ATR)', and Section 6.3, 'Protocol Parameter Selection (PPS)'

Listing 13.1 Typical activities of a mobile telephone that are related to the SIM, shown in proper temporal sequence. The portrayed activities and command sequences correspond to a typical mobile telephone, although it must be borne in mind that the GSM specifications generally leave the relevant details to the manufacturer of the mobile telephone. In this example, the SIM used in the mobile telephone essentially has only the functionality necessary for making telephone calls, with the exception of the files for abbreviated dialing numbers and short messages. In the case of a SIM or mobile telephone with a greater range of functions, the activities of the two communicating parties would increase accordingly.

The user switches on the mobile telephone.	
Perform SIM activation sequence	Activate the SIM.
Receive ATR	Determine whether a SIM is present and ascertain the parameters of the transmission protocol.
Execute PPS	Modify the transmission protocol parameters as necessary.
The mobile telephone has now established a working communications link with the SIM.	
SELECT DF_{GSM} GET RESPONSE	Select the GSM directory and retrieve information about the directory.
SELECT EF_{PHASE} READ BINARY	Select and read the EF containing the phase data.
SELECT EF_{LP} GET RESPONSE READ BINARY	Select the language preference EF, retrieve information about the file structure and read the file.
The user enters a PIN. VERIFY CHV STATUS	Test the PIN, and then query the state of the retry counter.
SELECT EF_{SST} GET RESPONSE READ BINARY	Select the SIM service table EF, retrieve information about the file structure and read the file.
TERMINAL PROFILE	Transfer information about the properties of the mobile telephone to the SIM. (important for SIM Toolkit applications)
SELECT MF	Select the root directory.
SELECT EF_{ICCID} GET RESPONSE READ BINARY	Select the ICC identification number EF, retrieve information about the file structure and read the file.
SELECT DF_{GSM}	Select the GSM directory.

SELECT EF_{IMSI} GET RESPONSE READ BINARY	Select the international mobile subscriber identity EF, retrieve information about the file structure and read the file.
SELECT EF_{AD} GET RESPONSE READ BINARY	Select the administrative data EF (which contains the administrative data for the mobile station), retrieve information about the file structure and read the file.
SELECT EF_{LOCI} READ BINARY	Select and read the location information EF.
SELECT EF_{KC} READ BINARY	Select and read the cipher key EF.
SELECT EF_{BCCH} READ BINARY	Select and read the broadcast control channels EF, which contains network-specific information.
SELECT EF_{FPLMN} READ BINARY	Select and read the forbidden PLMN EF.
SELECT EF_{HPLMN} READ BINARY	Select and read the HPLMN search period EF.
SELECT $DF_{TELECOM}$	Select the telecom directory.
SELECT EF_{SMSS} GET RESPONSE READ BINARY	Select the SMS status EF (which contains information about stored messages), retrieve information about the file structure and read the file.
SELECT EF_{SMSP} GET RESPONSE READ BINARY	Select the SMS parameters EF, retrieve information about the file structure and read the file.
SELECT EF_{SMS} GET RESPONSE n × READ RECORD	Select the SMS EF, retrieve information about the file structure and read all *n* records of the file.
SELECT EF_{ADN} GET RESPONSE n × READ RECORD	Select the abbreviated dialing numbers EF, retrieve information about the file structure and read all *n* records of the file.
The mobile telephone is now ready to make a call or transmit data.	
The user makes a call.	
SELECT DF_{GSM}	Select the GSM directory.
RUN GSM ALGORITHM GET RESPONSE	Authenticate the SIM with respect to the background system.

SELECT EF_{KC}
UPDATE BINARY
	Select the cipher key (Kc) EF and write the updated cipher key to the EF.

SELECT EF_{LOCI}
UPDATE BINARY
	Select the location information EF and write the updated location information to the EF.

SELECT EF_{BCCH}
UPDATE BINARY
	Select the broadcast control channels EF and write network-specific data to the EF.

The user switches off the mobile telephone.
SELECT EF_{LOCI}
UPDATE BINARY
	Select the location information EF and write the updated location information to the EF.

SELECT EF_{BCCH}
UPDATE BINARY
	Select the broadcast control channels EF and write network-specific data to the EF.

deactivation sequence is executed and the operating system of the mobile telephone is then shut down.

The procedures and mechanisms just described are not part of the GSM specification. Consequently, they are generally implemented in completely different manners in different types of mobile telephones. What has been described here should be regarded as only a possible and technically effective implementation. With the GSM system in particular, it should also be borne in mind that it is quite common for mobile telephones that are already 10 years old to still be in use. It can confidently be assumed that such telephones do not have virtual SIMs, but instead perform all read and write operations directly in the SIM.

Example of a typical command sequence

Reading dialing numbers from an EF with a record-oriented structure, such as EF_{ADN}, is a practical example of a typical command sequence. The first step is to select the appropriate file in the proper directory. Since the number of records in the file is left up to the network operator, the first thing that must be done is to determine the size of the file. The number of entries is then calculated from the file size and the record length. After this, each record containing a dialing number can be read using READ RECORD with the number of the record in question. This process is shown in detail in Figure 13.16.

SIM Application Toolkit

In the original specifications for the GSM system, the GSM card was simply seen as a means to identify the user using PIN and an authentication token, in the interest of billing security, that was independent of the mobile telephone. However, in the course of time the desire to utilize the GSM card for additional functions, particularly supplementary services, became increasingly pronounced. For instance, a mobile telephone is also a competent medium for checking the balance of a bank account or receiving vital news, such as football scores and

Terminal		SIM
SELECT FILE	→	
Command [$DF_{TELECOM}$]		return code := file selection result
IF (return code = OK)	←	*Response* [return code]
THEN file successfully selected		
ELSE abort		
SELECT FILE	→	
Command [EF_{ADN}]		return code := file selection result
IF (return code = OK)	←	*Response* [return code]
THEN file successfully selected		
ELSE abort		
GET RESPONSE	→	Ascertained file size s
		Ascertained record length m
IF (return code = OK)	←	*Response* [s || m || return code]
THEN command successfully executed		
ELSE abort		
Computer number of records n		
n := s ÷ m		
VERIFY CHV	→	Test CHV
Command [CHV 1]		return code: = result of CHV testing
IF (return code = OK)	←	*Response* [return code]
THEN CHV testing successful		
ELSE abort		
FOR x := 1 TO n {		
READ RECORD	→	
Command [record number n]		
IF (return code = OK)	←	*Response* [record data || return code]
THEN record successfully read		
ELSE abort }		

Figure 13.16 Basic command sequence for reading the abbreviated dialing numbers from the EF_{ADN} file. The illustrated sequence shows only the essential aspects of the process and assumes that all commands are successfully executed

daily horoscopes. However, the modest capabilities of the GSM were not sufficient to permit the technical implementation of these value-added services (VAS). The response to this was the development of the GSM 11.14 specification, entitled ‘SIM Application Toolkit’ (SAT). The first version of this specification was published in 1996 by ESTI.

The SIM Application Toolkit enables the SIM to directly access functions of the mobile station, such as driving the display, polling the keypad, sending short messages and other functions needed in connection with a value-added service. Ultimately, the SIM Application Toolkit is a construction kit that allows almost any desired application to be implemented in a SIM.

A number of new commands had to be defined for the SIM Application Toolkit. A noteworthy feature of these commands is that they are sent to the mobile equipment by the SIM, which requires a certain change in mental attitude. The data part of these ‘proactive’ commands is

BER-TLV coded.[13] This makes it possible to easily achieve expansion capability while ensuring downward compatibility. However, the greatest advantage of this is the enormous flexibility obtained by using TLV-coded data.

With the SIM Application Toolkit, it was necessary to devise a way to circumvent the usual master–slave arrangement between the terminal and the smart card for the SIM, but for reasons of compatibility, modifying the transmission protocol was not allowed. The solution to this problem was relatively simple. In a process called 'polling', the mobile equipment sends the query command STATUS to the SIM at a definable regular interval (such as every 20 seconds), and if necessary the SIM can indicate in its response that a command for the mobile equipment is ready to be sent and should be fetched from the SIM. In practice, the polling interval is not maintained all that exactly by the mobile equipment, but this is not critical. This circumvention of the master–slave principle is designated 'proactivity of the SIM', and the associated commands are called 'proactive commands'.

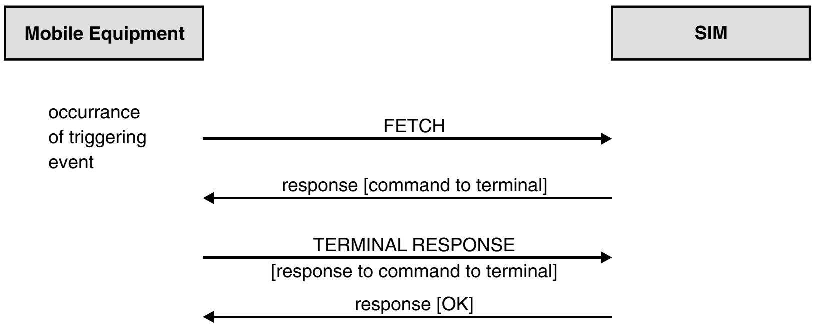

Mobile Equipment
SIM
occurrance of triggering event
FETCH
response [command to terminal]
TERMINAL RESPONSE
[response to command to terminal]
response [OK]

Figure 13.17 The extended protocol process between the mobile equipment and the SIM for the proactive commands of the SIM Application Toolkit, as specified in GSM 11.14. The response of the smart card to a command contains a command to the terminal in the data part. The terminal executes this command and returns the associated response to the smart card in the data part of a command. The sequence shown here is based on the transmitted APDUs and shows only successful results

This technique effectively reverses the master–slave relationship between the mobile equipment and the SIM. This makes it possible for the card, acting on its own initiative, to poll the keypad, show its own data and menu structures on the display of the mobile telephone and emit a beep sound. The SMS mechanism can also be used to exchange data between the SIM and the GMS background system via the air interface. For instance, a news server can be regularly polled in this manner, with the result being presented on the display of the mobile telephone as an e-mail or short message.

The commands that make this mechanism possible are FETCH, TERMINAL RESPONSE and ENVELOPE. The mobile equipment uses FETCH to retrieve a command from the SIM. After processing this command, the mobile equipment returns the associated result to the SIM using TERMINAL RESPONSE. The ENVELOPE command allows data to be transferred to the SAT application of the mobile equipment.

In addition to this proactivity, the SIM can also inform the mobile equipment of certain events for which the SIM must be immediately notified if they occur.

[13] See also Section 4.1, 'Structuring Data'

Table 13.7 The proactive SIM smart card commands specified for the SIM Application Toolkit in GSM 11.14. Note that the commands listed here are sent to the terminal by the smart card, rather than from the terminal to the smart card as usual. Certain commands can only be used if they are supported by the hardware configuration of the mobile equipment

Command	Brief description
User interface	
DISPLAY TEXT	Show a text or icon passed with the command on the display of the mobile station.
GET INKEY	Show a text or icon passed with the command on the display of the mobile station, followed by requesting a character from the keypad.
GET INPUT	Show a text or icon passed with the command on the display of the mobile station, followed by requesting one or more characters from the keypad.
LANGUAGE NOTIFICATION	Advise the mobile equipment of the language used by the SIM Application Toolkit in the text fields.
PLAY TONE	Instruct the mobile equipment to issue a tone.
SELECT ITEM	Transfer a selection list to the mobile equipment with the instruction that the user is to select an item.
SET UP IDLE MODE TEXT	Show a text or icon passed with the command on the display of the mobile station while the mobile station is switched on but not in use.
SET UP MENU	Transfer a menu list to the mobile equipment with the instruction to integrate it into the menu structure of the mobile equipment.
Second card terminal	
GET READER STATUS	Request the status of a supplementary card terminal in the mobile station.
PERFORM CARD APDU	Send an APDU to the smart card located in a supplementary card terminal in the mobile station.
POWER OFF CARD	Deactivate the smart card located in a supplementary card terminal in the mobile station.
POWER ON CARD	Activate the smart card located in a supplementary card terminal in the mobile station.
Network interface	
CLOSE CHANNEL	Instruct the mobile equipment to close a data channel.
GET CHANNEL STATUS	Instruct the mobile equipment to return the status of a data channel.
OPEN CHANNEL	Instruct the mobile equipment to open a data channel.
RECEIVE DATA	Instruct the mobile equipment to receive data via an open data channel.
RUN AT COMMAND	Transfer an AT command to the mobile equipment and execute the command in the mobile equipment, followed by passing the result back to the SIM.
SEND DATA	Instruct the mobile equipment to transmit data via an open data channel.
SEND DTMF	Transmit a DTMF during a current voice connection.
SEND SHORT MESSAGE	Transmit a short message.

Table 13.7 (*Cont.*)

Command	Brief description
SEND SS	Transmit a supplementary service (SS) message, which is a control sequence, to the network.
SEND USSD	Transmit an unstructured supplementary services data (USSD) message, which can be used to send any desired type of data.
SET UP CALL	Establish a connection.
Miscellaneous	
MORE TIME	Request the mobile equipment to give the SAT application more time for processing.
POLL INTERVAL	Start cyclic polling of the SIM and specify the interval.
POLLING OFF	Stop cyclic polling of the SIM.
PROVIDE LOCAL INFORMATION	Request the mobile equipment to provide current location information to the SIM.
REFRESH	Advise the mobile equipment that the data content of the SIM has changed, so it should read this data anew.
SET UP EVENT LIST	Transfer an event list to the mobile equipment with the request to inform the SIM if one of these events occurs.
TIMER MANAGEMENT	Start, end or configure the eight possible timers in the mobile equipment that can generate an event.
LAUNCH BROWSER	Start a microbrowser supported by the smart card operating system.

There are several different ways to launch SAT-based supplementary services in the SIM. The simplest manner involves an action on the part of the user. For example, if the user selects a certain function from the menu of the mobile telephone and this function is based on a supplementary SIM application, a corresponding command is sent to the SIM by the mobile equipment. The further course of events is then determined by the value-added service in the SIM. However, certain events in the mobile telephone, such as call setup, call termination or changing network cells, can be used to invoke a SAT-based application in the SIM. The simplest method for invoking a SAT application in the SIM is cyclic polling of the SIM by the mobile equipment. In practice, it is possible to implement SIM-based value-added services at a relatively moderate cost using these three basic invocation methods.

The actual capability for controlling supplementary services in the SIM Application Toolkit is achieved using executable program code, which can be generated using any desired programming language, such as assembler, C or Java.

The typical sequence of events with a SIM Application Toolkit application is as follows: first, following the activation sequence of the SIM, various types of data are read by the mobile equipment, including the EF_{Phase} file, which indicates which GSM phase the SIM supports. If the code for Phase 2+ is stored in the EF_{Phase} file, the terminal concludes that the SIM Application Toolkit is fully supported. Following this, the terminal uses the TERMINAL PROFILE command to inform the SIM of its properties that are relevant to the SIM Application Toolkit. This completes the initialization, and any other commands related to the GSM application that do not belong to the SIM Application Toolkit can then be sent as necessary.

Typically, the next process is installing a selection menu in the mobile equipment. This is done by placing BER-TLV coded data for the menu in the response to a FETCH command requested by the SIM and sent by the mobile equipment. The mobile equipment then integrates the new selection menu into its menu structure and acknowledges having done so with a confirmation in the subsequent TERMINAL RESPONSE command. The selection menu is thus installed in the mobile equipment and activated. After this, the usual GSM commands can be exchanged and processed. As soon as the user of the mobile telephone selects a menu entry, the ENVELOPE command is sent to the SIM with information about the selected menu entry. The SIM confirms receipt of the command and can then start a wide variety of processes belonging to the application and user selection.

For example, a share price on the stock exchange could be requested as the result of selecting a SIM application. This function can be implemented in a wide variety of manners. One simple method would be to send an SMS message to a server of the network operator with a request for the current share price for a particular company. If this request is successfully processed, the server could then send a SMS reply message to inform the application in SIM of the share price, and the application could then advise the mobile telephone user of the current share price using a DISPLAY TEXT command.

This is only a very simple example of what can be done using the SIM Application Toolkit, but clearly shows that the SIM Application Toolkit is a very powerful tool for producing value-added services in the SIM, and that it is relatively easy to implement such services. Things can start to become difficult when a value-added service must be implemented using functions of the mobile equipment that are not supported by the SIM Application Toolkit. Other well-known hindrances to SAT-based applications are the large variety of methods for presenting data on the display and fundamental incompatibilities or implementation errors in the mobile equipment. However, all of these hurdles can be overcome with a certain amount of effort and experience. In summary, it can thus be said that the SIM Application Toolkit is still the most technically mature and secure means to implement value-added services in mobile telephones. The SIM Application Toolkit forms a very powerful interface for value-added services in the SIM, and it can be integrated into the existing system without any modifications.

The ETSI Project Smart Card Platform (EP SCP) expert group is in the process of defining a generic foundation for all application toolkits for smart cards in mobile telecommunications, based on the SIM Application Toolkit. This toolkit will be called the Card Application Toolkit (CAT), and it will form the basis for the SIM Application Toolkit (SAT), the USIM Application Toolkit (USAT) and the UIM Application Toolkit (UATK).

Over-the-air (OTA) communication

After a SIM has been issued, it is sometimes necessary to establish a direct connection from the background system to the SIM. This type of communication is particularly essential for managing existing applications and generating new value-added services in the SIM. Consequently, mechanisms have been created in the GSM 03.48 specification to allow secure end-to-end communications to be established between the background system and the SIM via the air interface.

Since this requirement was not dealt with by the original GSM specifications and it is nearly impossible to make changes in a system of this magnitude, a trick is used for end-to-end

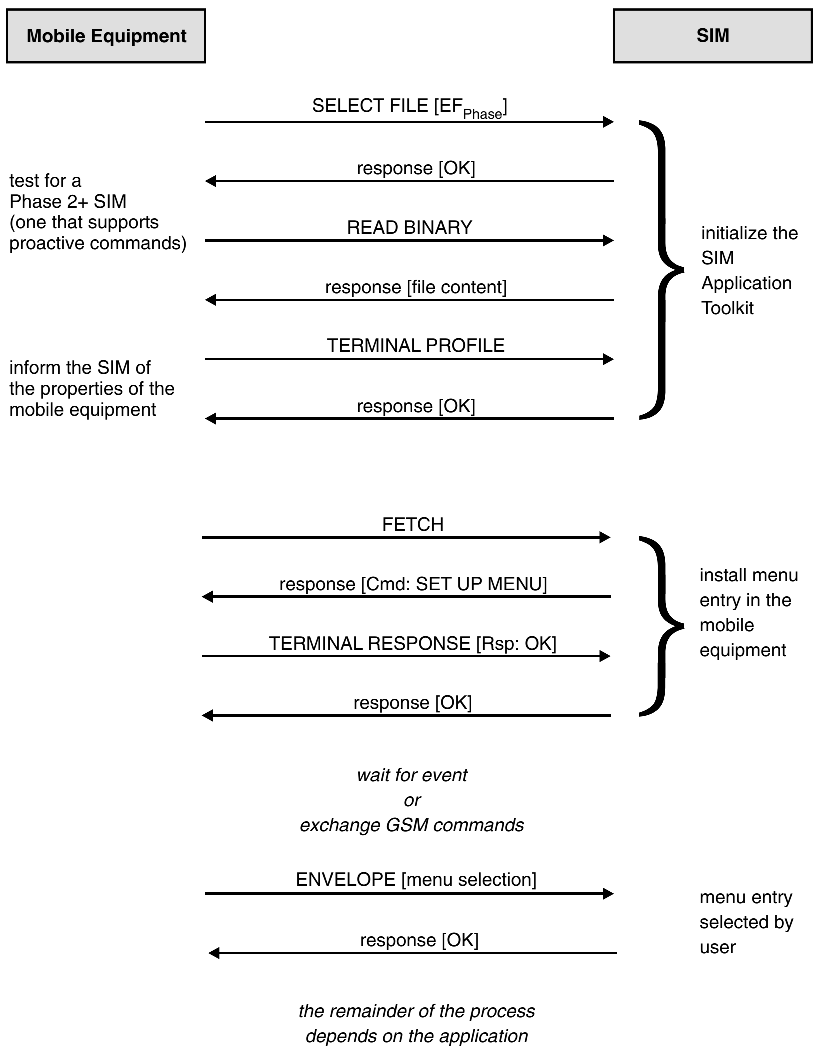

Mobile Equipment
SIM
SELECT FILE [EF$_{Phase}$]
response [OK]
test for a Phase 2+ SIM (one that supports proactive commands)
READ BINARY
response [file content]
TERMINAL PROFILE
inform the SIM of the properties of the mobile equipment
response [OK]
initialize the SIM Application Toolkit
FETCH
response [Cmd: SET UP MENU]
TERMINAL RESPONSE [Rsp: OK]
response [OK]
install menu entry in the mobile equipment
wait for event or exchange GSM commands
ENVELOPE [menu selection]
response [OK]
menu entry selected by user
the remainder of the process depends on the application

Figure 13.18 Typical example of using the SIM Application Toolkit to install a supplementary menu entry in the mobile equipment. The procedure illustrated here is based on the transmitted APDUs and shows only successful command execution

communication with the GSM card. The short messages available in the system are used as containers for messages to and from the SIM. All that this requires is modifications to the background system and issuing new smart cards, with all intermediate systems remaining unchanged. Nevertheless, short messages are presently only one of several possible bearers for OTA, although they are the most widely used type.

OTA communication offers a relatively wide range of protective mechanisms for the transmitted data. For instance, the simplest security level consists of using a CRC checksum to protect the data against transmission errors. In the realm of cryptographic protection, it is also possible to provide the data with a send sequence counter and encrypt them using DES or triple DES (with two or three keys). If necessary, a MAC or digital signature can also be computed for the data to be transmitted.

The operating principle of using the SMS as a bearer service is as follows. If the background system wishes to send a command (for example) to a particular SIM, it generates a short message to the card in question and embeds the command in the message, using the necessary cryptographic protective mechanisms. As soon as a mobile station containing the SIM in question logs in to the system, the short message is transmitted via the signaling channel. This does not require establishing a voice connection via a traffic channel. Based on the coding of the message as specified in GSM 03.40, the mobile equipment recognizes that the message contains SIM-specific data and uses the SIM Application Toolkit ENVELOPE command to send it to the SIM. The message is thus not automatically stored in the EF_{SMS} file by the mobile equipment using the UPDATE RECORD command, as is otherwise usual. The SIM stores the message received via the ENVELOPE command in a separate buffer. If the message is part of a set of chained messages, the next task of the SIM is to restore the correct sequence of the messages, since as is well known, SMS does not ensure that messages are received in the proper order.

The SIM then interprets the received message, extracts the command or commands from it and processes it or them. A SMS message may optionally be generated by the SIM as a response. This message is transferred to the mobile equipment in the response to a FETCH command requested by the SIM, and the mobile equipment forwards it to the background system in the usual manner via the service channel.

With this trick, it is possible to establish a bidirectional end-to-end link between the background system and the SIM that is fully transparent to all of the intermediate system components. This allows the SIM to be addressed just as though it were located in a terminal connected to a PC. This communications channel can be used for tasks such as modifying existing data in files as part of remote file management. A common use for OTA communication is updating the service dialing numbers stored in the SIM. It can also be used to carry out significantly more complex tasks. For instance, it can be used to download executable program code in the form of applets for supplementary applications in SIMs based on Java Card. The possibilities that OTA communication offers to the network operator are immense. Unfortunately, many parts of GSM 03.48 do not represent a specification, which is precisely defined at the bit level, but instead a standard, which offers a wide range of options, not all of which are specified in detail, that can be used by individual card manufacturers for their SIMs in the manner that best suits their purposes. This naturally has detrimental consequences for the mutual compatibility of SIMs from different manufacturers, which must be compensated in normal network operation by libraries in the background system of the network operator that are specific to the various smart card manufacturers.

Remote file management (RFM)

The mechanisms provided by OTA allow direct end-to-end communication between the background system and the SIM. This forms the basis (with regard to data transmission technology) for the remote management of the files in the SIM, which is called remote file management (RFM) in GSM terminology. This bearer-independent basic

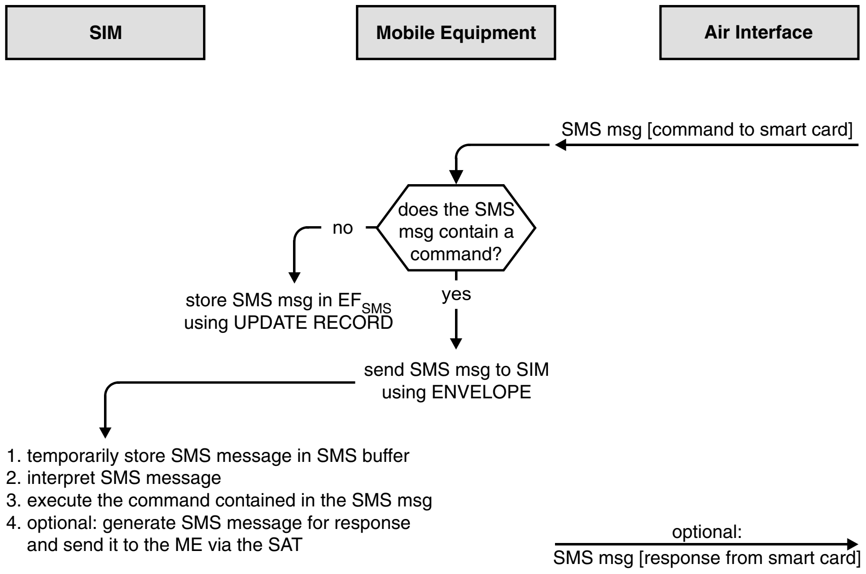

SIM
Mobile Equipment
Air Interface
SMS msg [command to smart card]
does the SMS msg contain a command?
no
yes
store SMS msg in EF_{SMS} using UPDATE RECORD
send SMS msg to SIM using ENVELOPE
1. temporarily store SMS message in SMS buffer
2. interpret SMS message
3. execute the command contained in the SMS msg
4. optional: generate SMS message for response and send it to the ME via the SAT
optional:
SMS msg [response from smart card]

Figure 13.19 Procedure for exchanging data using SMS messages passed between the background system and the GSM card. This procedure is commonly called 'over the air' (OTA) communication

functionality is specified in GSM 03.48, which is in turn based on the requirements of GSM 02.48.

Only certain SIM commands are allowed to be used for remote file management, but it is possible to achieve a broad scope of functionality using these commands. They are divided into input commands, which send data to the SIM, and output commands, which request data from the SIM. The background system is allowed to send not only individual commands to the SIM within an OTA message, but also lists containing several commands. However, such lists are subject to the restriction that only the final command in the list is allowed to request data from the SIM. The reason for this restriction, which does not cause any difficulties in practice, is primarily that it significantly simplifies the remote file management software in the SIM. Due to this restriction, several OTA messages must be sent to the SIM if several files or records have to be read.

Table 13.8 Smart card commands allowed to be sent to the SIM for remote file management, as specified by GSM 03.48

Input commands		Output commands
SELECT	VERIFY CHV	READ BINARY
UPDATE BINARY	CHANGE CHV	READ RECORD
UPDATE RECORD	DISABLE CHV	GET RESPONSE
SEEK	ENABLE CHV	
INCREASE	UNBLOCK CHV	
	INVALIDATE	
	REHABILITATE	

The operating principle of remote file management using SMS as a bearer service can briefly be explained using a typical practical example. If a background system wants to modify an abbreviated dialing number stored in the EF_{ADN} file, it can proceed as follows. In the first OTA message, which may consists of a series of SMS messages, it selects the EF_{ADN} file by means of a SELECT command specifying the path within the $DF_{Telecom}$ directory. The final command in this OTA message is a READ RECORD command with a record number known to the background system, which causes the service number to be read from the file and returned via OTA. If this service number is not current, an UPDATE RECORD command is sent to the SIM using another OTA message, and the appropriate record is overwritten with the new number.

Naturally, caution must be exercised in using remote file management to modify files that are significant for an open session. A typical example of such files is EF_{SST}, which contains the SIM service table. This table lists all the available and potentially activated services of the SIM. Under certain conditions, modifying the content of this file can cause the mobile telephone to behave unpredictably, and in the worst case it can render the SIM unusable.

The SIM also contains two EFs for which modification is simply forbidden. These are the EF_{ICCID} file, which holds the identification number of the smart card (ICCID), and the EF_{Kc} file, which holds the key for encrypting data transmitted between the mobile station and the base station via the air interface. From a logical perspective, it makes no sense to modify either of these files, since the ICCID is a unique identification number for the smart card and plays no role in normal operation. The key (Kc) is always computed by the SIM for each session, so it would be pointless to modify it via RFM. If it is nevertheless modified during an open session, the connection to the network might be broken, since the mobile equipment would use an incorrect key for encrypting data on the air interface.

Remote applet management

The GSM 03.48 specification also contains a section related to remote applet management, which is similar to remote file management. Remote applet management makes it possible to manage applications based on Java Card via a direct end-to-end link between the background system and the SIM.

A general prerequisite is that the smart card in question must be a SIM that is compliant with GSM 03.19, which is essentially based on the Java Card 2.1 specifications.[14] All management commands for applets and packages are based on the Open Platform specification, which is effectively the industry standard for these mechanisms.[15]

The application management functions include loading, installing, deleting, locking and unlocking Java applets in the SIM and retrieving parameters from these Java applets. Similar mechanisms are defined for loading packages into the SIM and deleting packages from the SIM.

All of the procedures and mechanisms are basically independent of any particular bearer service, but the SMS is presently the most commonly used means for managing applets and packages in SIMs via OTA. If SMS is used as the bearer, data transmission to and from the SIM takes place in exactly the same manner as described above for remote file management using OTA.

[14] See also Section 5.14.1, 'Java Card'
[15] See also Section 5.11, 'Open Platform'

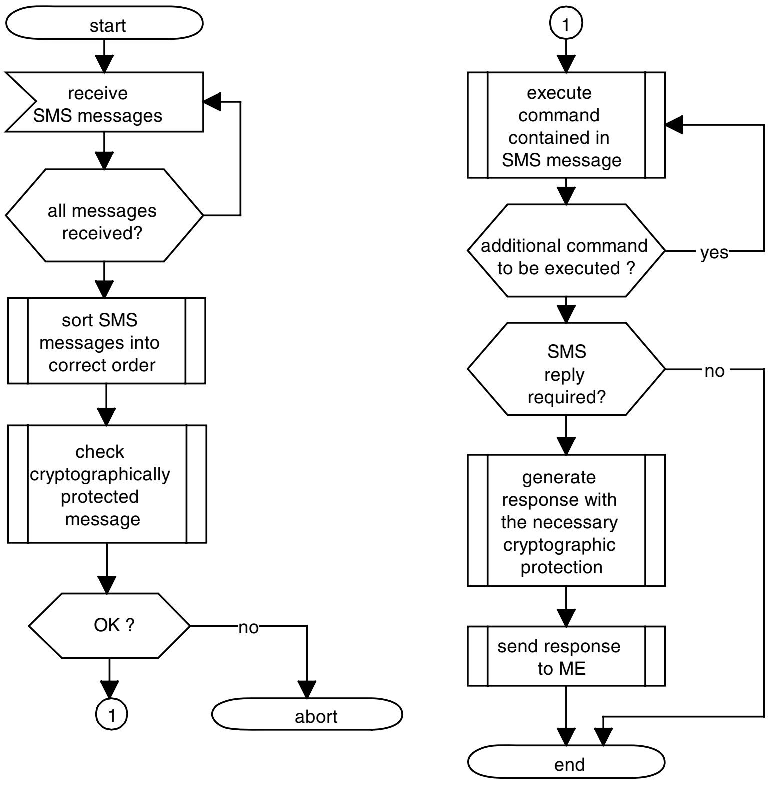

start
receive SMS messages
all messages received?
sort SMS messages into correct order
check cryptographically protected message
OK ?
no
1
abort
1
execute command contained in SMS message
additional command to be executed ?
yes
SMS reply required?
no
generate response with the necessary cryptographic protection
send response to ME
end

Figure 13.20 Flow chart of the basic program sequence for remote file management (RFM) via the air interface using OTA, as specified in GSM 03.48. Remote file management is essentially performed by the processes shown in the upper right branch of the flow chart, with the remainder of the processes serving to establish secure communications in accordance with GSM 03.48

Table 13.9 Smart card commands allowed to be sent to the SIM for remote applet management via the air interface, as specified by GSM 03.48. These commands correspond to the Open Platform specification with regard to functionality and coding

Input commands	Output commands
DELETE	READ BINARY
SET STATUS	READ RECORD
INSTALL	GET RESPONSE
LOAD	
PUT KEY	

Dual IMSI

In the commercial realm, accrued call charges for a mobile telephone belonging to a company are usually paid by the company in question. However, if a person having such a mobile telephone also uses it for private calls, he or she must later settle the charges for such calls with the company, which is only possible with an itemized telephone bill. In practice, such a person will probably make private calls at the expense of the company, otherwise he or she must carry two telephones in order to make both business and private calls. Needless to say, this is too much to expect. This scenario is the reason why there are 'dual-IMSI' mobile telephones, which contain SIMs having an additional file holding two IMSIs and possibly also two Ki keys, depending on the implementation. This file is actually not part of any standard. A supplementary SAT-based application in the SIM generates a new menu entry in the mobile equipment that allows the user to select whether he or she wishes to make a business call or a private call. Depending on the user's selection, a particular IMSI is copied to the EF_{IMSI} file, and if necessary a different, related Ki key is used. This means that the mobile telephone has two different identities in terms of the IMSI, and the network operator can generate two separate bills. This makes it possible to charge separately for business calls and private calls.

This capability can be further extended, for example by activating the fixed dialing numbers for the IMSI used for business calls, thus limiting the possible dialing numbers to a particular set of numbers. This means that the user of such a mobile telephone cannot charge private calls to the company, since only the numbers related to the company can be dialed. If the second IMSI is activated using the value-added service in the SIM, the fixed dialing numbers are deactivated and the company is not billed for the calls. In this way, a single mobile telephone can be used to make both business and private calls without creating problems in settling the calling charges.

There is yet another use for multiple IMSIs, which it is not very elegant from a technical perspective. If IMSIs for several different network operators are stored in a SIM such that they can be individually selected by the user via a menu, the mobile telephone can be used for manual roaming. The user selects the IMSI belonging to the network in whose territory he or she happen to be located, and can then log into this network using the selected IMSI. The user will then receive a telephone bill from each network operator as appropriate. This sort of roaming using multiple IMSIs is practiced in large parts of India, for example, since regular roaming agreements between some of the network operators do not exist.

Implementing a home zone

Some network operators offer person-specific special rates for restricted local regions. Such a region is usually an approximately circular zone defined around the place of residence of the subscriber, within which calls are charged at the rate for the fixed network instead of the more expensive rate for the mobile network. For the user, the primary benefits of this type of location-specific service are that he or she no longer needs a connection to the fixed telephone network, and that it eliminates the need to coordinate two sets of abbreviated dialing numbers (one for the regular telephone and another one for the mobile telephone).

The most suitable way to implement such a service is to use a value-added service in the SIM and the functions of the SIM Application Toolkit. This also ensures that all information about the home zone is stored in the person-specific SIM, rather than somewhere else such as in the mobile telephone.

In the GSM system, each base station continuously transmits a unique identifier via the air interface. This identifier consists of the location area information (LAI) and a cell identity (CI). One approach to implementing home zone capability would be to have the mobile telephone pass this information to the SIM, where it could be compared with one or more stored values. These values would preferably be stored in a file, so that they could be modified or updated at any desired time using remote file management. The drawback of this approach is the relatively large amount of memory needed in the SIM, since it is fundamentally necessary to accommodate regions with a high density of base stations, which would lead to large LAI and CI lists.

A similar approach would be to use the advance timing information of the air interface. With this approach, the current location of a mobile station could be determined to within significantly less than 10 meters by using cross-polling. This is more than adequate for implementing a home zone.

However, in practice a different solution is often preferred, in which all of the base stations belonging to a network operator periodically transmit their location coordinates on the signaling channel using the cell broadcast service.[16] The SIM has an EF containing reference values, which are read by the mobile equipment and compared with the received location coordinates. If the mobile equipment determines that the mobile telephone is located within the home zone, a suitable symbol (such as a small house icon) is shown on the display. Since the background system knows the location of the mobile telephone, it can switch incoming and outgoing calls over to a more favorable rate. The data for the coordinates of the home zone are stored in an EF in the SIM, so they can be easily modified using remote file management. This also allows home zones to be conveniently established or changed to a different location using remote maintenance. The drawback of this solution is that it requires special software in the mobile equipment, instead of being implemented as a value-added service in the SIM using the SIM Application Toolkit.

Operating principle of SIM Lock

SIM Lock is the name given to a technique for binding the mobile equipment to a particular SIM or group of SIMs. The SIM Lock function is used by network operators to bind mobile telephones subsidized by a network operator to a particular SIM and its payment mode for a certain length of time. It is based on the GSM 02.22 specification. The operating principle of the SIM Lock is always based on data that are stored in both the SIM and the mobile equipment and are compared by one of the two components each time the mobile telephone is switched on, with the telephone only being enabled for use if the two sets of data match.

There are two practical implementations of the SIM Lock function. With the more commonly used option, the mobile telephone reads certain data from the SIM and compares them with data stored in the mobile telephone. This usually consists of the group identifiers, which are stored in the EF_{GID1} (group identifier level 1) and EF_{GID2} (group identifier level 2) files. These

[16] Harald Bögeholz and Dusan Zivadinovic, 'Telefon-Zellen', c't 1999, Volume 18

group identifiers can be used to specify classes of SIMs, which can then be used to specify class-based pairings of particular SIMs to particular (subsidized) items of mobile equipment. The advantage of this variant is that the SIM and the mobile equipment do not have to be individually 'married'. The IMSI from the EF_{IMSI} file, or other static SIM-specific data stored in EFs, is sometimes used instead of the group identifiers as a reference value for forming pairs.

The second option, which is rarely used in practice, involves having the SIM use the SIM Application Toolkit to read unique data from the mobile equipment and compare it with stored data. If these data match, the mobile telephone can be used to make the desired call after being enabled by the SIM.

It is usually possible to disable the SIM Lock function, either via the air interface or by entering a secret key into the mobile telephone, in order to allow other SIMs to be used in mobile equipment previously protected by a SIM Lock. The reference value for this is usually the individual mobile equipment identity (IMEI) of the mobile equipment in question.

Operating principle of prepaid systems

The proportion of prepaid SIMs ranges from around 30 % to as much as 80 %, depending on the country. The principal reasons why people use prepaid SIM are that they provide better control of costs and avoid the need to pay subscription charges.

The operating principle of a system designed to work with prepaid SIMs is generally as follows. A card-shaped voucher, which often has the dimensions of an ID-1 card but is not as thick, has a 13-digit number printed underneath a rub-off coating, which acts as a seal. If a user wishes to 'reload' his mobile telephone, he or she purchases a voucher, whose integrity can be verified by the fact that the rub-off coating is still intact. After rubbing off the coating covering the number, the user must enter the now-visible number into her mobile telephone using a special menu. This reference value is immediately passed to the background system, where it is compared with the reference value for the issued voucher, which is stored in a database. If the result of the comparison is positive and if the voucher has not already been used, the load amount associated with the reference value is credited to the SIM in question.

At this point, the possible implementations diverge. The solution originally envisaged in the GSM specifications was a units counter in a file (the accumulated call meter file EF_{ACM}), whose value would be continuously updated by advice of charge data from the mobile equipment and compared with the value stored in the 'accumulated call meter maximum value' file (EF_{ACMmax}) by the SIM. If the actual value reached the maximum value, the mobile telephone would prohibit further calls until the actual value was again reset, which could be done via remote file management or some other means. Although this solution is certainly technically feasible, it is not used in practice, since communications between the mobile equipment and the SIM are not secure and thus could be manipulated relatively easily.

In practice, prepaid SIMs are managed by a centralized system, with two different approaches being used. The first approach entirely dispenses with using supplementary data in the SIM and runs entirely in the background system. The drawback of this approach is that the background system computer must have real-time capability, which increases its cost. With the second approach, a suitable value-added service must be present in the SIM, but there is no need for general real-time capability in the background system. The disadvantage of this approach is that when the prepaid amount has been used up, there may be a delay before the connection is broken.

A typical background system for prepaid SIMs, which does not necessarily have to have real-time capability, can be described using the components shown in Figures 13.21 and 13.22. A number of supplementary components must be integrated into an existing GSM system in order to support prepaid SIMs. In this example, one of these components is the call management subsystem, which is a supplementary component of the mobile switching center (MSC) that can route or prohibit calls in real time, and which maintains an interface to the prepaid system. The prepaid system is the central component of the system, whose task is to coordinate the call management subsystem and the billing system. In the billing system, the credit balances in the individual SIM accounts are managed using a database. With this arrangement, the SIM contains only a few special commands along with corresponding data.

When a call to a mobile telephone arrives in the background system, the first thing that happens is that the call management subsystem advises the prepaid system that a call to particular mobile telephone having a particular SIM is pending. The prepaid system then has the billing system calculate the maximum allowable length of the call, based on the outstanding credit balance for the SIM, and passes this information to the call management subsystem. If the credit balance is sufficient, the call management subsystem routes the call. It will also interrupt the call if the maximum call duration is reached. On completion of the call, the call management subsystem informs the prepaid system of the duration of the call, and the prepaid system uses this information to update the credit balance of the account via the billing system. The updated balance can then be shown on the display of the mobile telephone.

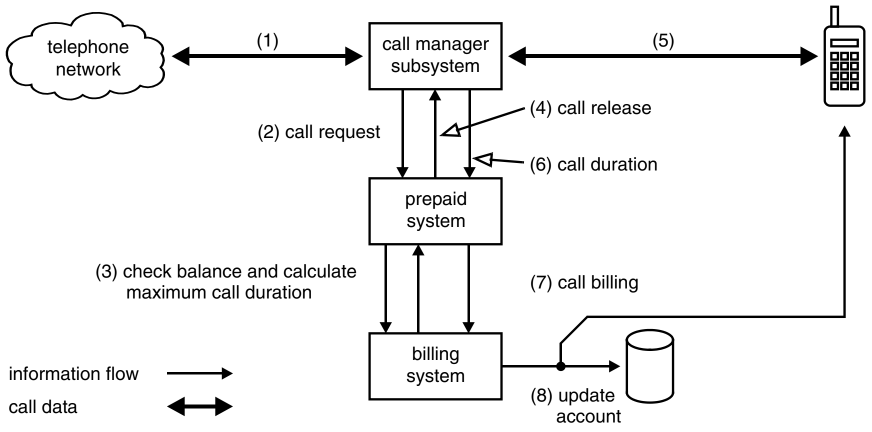


Figure 13.21 Basic architecture of a system for prepaid SIMs using the SICAP Prepaid Roaming solution as an example. This diagram shows the progress of a call made to a mobile telephone, with the numbers in parentheses indicating the sequence of events. The call management subsystem is part of the GSM background system, and may for example be a component of the mobile switching center (MSC)

When a call is made from a mobile telephone, a similar process occurs. First, the maximum allowable call duration is computed via a USSD query to the prepaid system and the billing system and then passed to the call management subsystem. If the maximum duration is reached during the call, the call is interrupted. Otherwise, the balance of the call account is updated on completion of the call and the corresponding amount is stored in the database. Naturally, the remaining credit can also optionally be displayed on the mobile telephone.

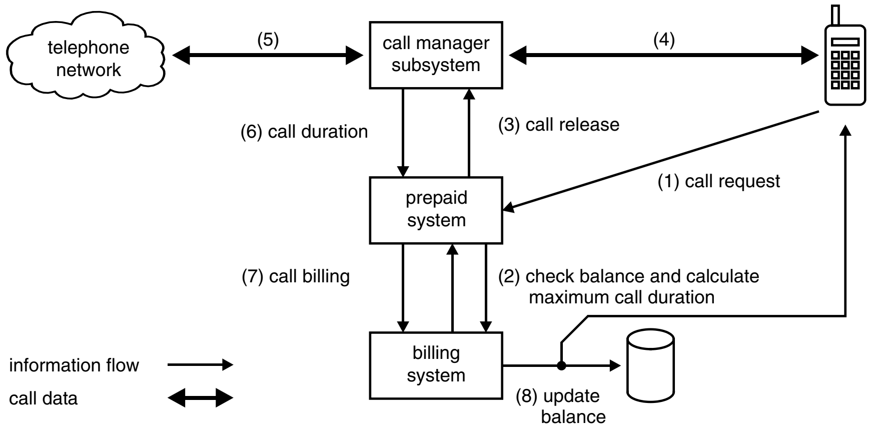


Figure 13.22 Basic architecture of a system for prepaid SIMs using the SICAP Prepaid Roaming solution as an example. The diagram shows the progress of a call originating from the mobile telephone, with the numbers in parentheses indicating the sequence of events. The call management subsystem is part of the GSM background system, and may for example be a component of the mobile switching center (MSC)

13.2.5 General Packet Radio System (GPRS)

The General Packet Radio System (GPRS) is an extension of the original GMS system. It has been defined as an ETSI standard, and its purpose is to provide a packet-switched data service with a high data transmission rate, as specified in GSM 01.60 ('Requirements specification of GPRS') and GSM 02.60 ('Service description; Stage 1'). GPRS can be dynamically adapted to actual capacity demand, so only the actually necessary capacity is used. A maximum data transmission rate of 115.2 kbit/s can be achieved by bundling the eight available time slots, each of which has a capacity of 14,400 kbit/s. A mobile telephone with GPRS technology is constantly logged in to the network with respect to data transport, and thus always available for data transmission without requiring a connection to first be established for this purpose. Consequently, GPRS is highly suitable for discontinuous data transmission. GPRS also forms the basis for mobile telecommunications services based on the Internet protocol (IP).

With regard to system architecture, GPRS is based on a GSM system augmented by several new components. The serving GPRS support node (SGSN), which coordinates the exchange of data packets with the mobile equipment at the MSC level, is analogous to the MSC. The SGSN is subordinate to a gateway GPRS support node (GGSN), whose primary function is to provide an interface to other packet-switched data services, such as X.25 and IP. The GGSN transforms GPRS-specific data packets into packets corresponding to the other packet-switched services

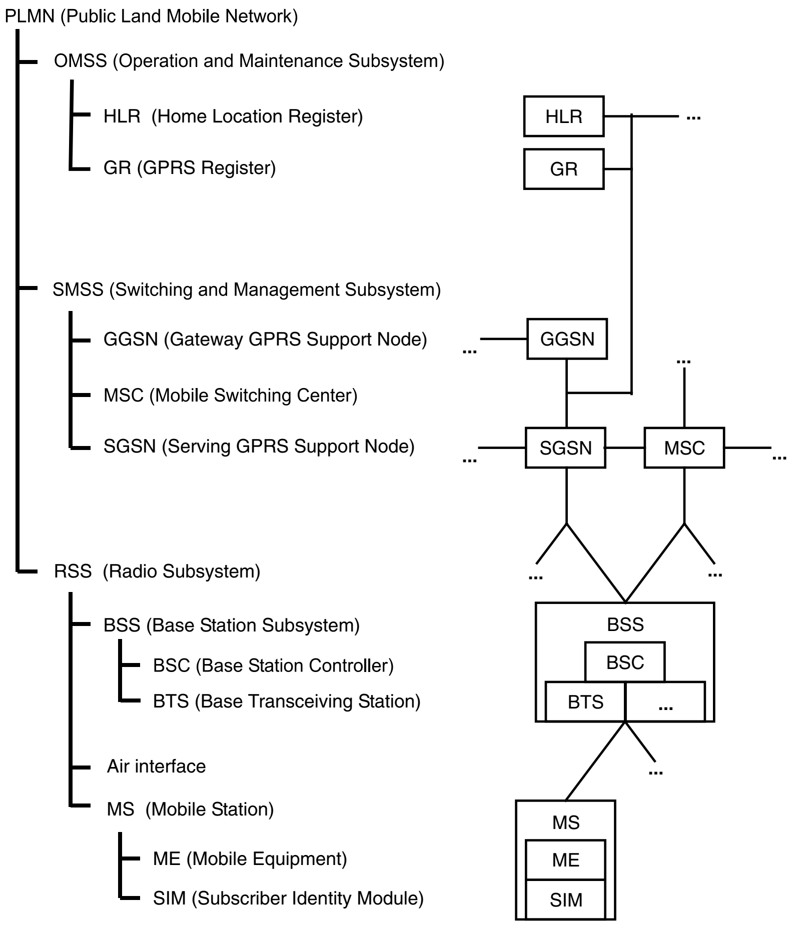

PLMN (Public Land Mobile Network)
OMSS (Operation and Maintenance Subsystem)
HLR (Home Location Register)
GR (GPRS Register)
SMSS (Switching and Management Subsystem)
GGSN (Gateway GPRS Support Node)
MSC (Mobile Switching Center)
SGSN (Serving GPRS Support Node)
RSS (Radio Subsystem)
BSS (Base Station Subsystem)
BSC (Base Station Controller)
BTS (Base Transceiving Station)
Air interface
MS (Mobile Station)
ME (Mobile Equipment)
SIM (Subscriber Identity Module)
HLR
GR
GGSN
SGSN
MSC
BSS
BSC
BTS
MS
ME
SIM

Figure 13.23 Architecture of a portion of a GSM network belonging to a single network operator, with a superimposed GPRS network as specified by GSM 01.60 and GSM 02.60

and vice versa. The central component of the system is the GPRS register (GR), which is analogous to the HLR and manages all of the data related to specific GPRS subscribers.

13.2.6 Future developments

The GMS application represented the international breakthrough for smart cards, and it is still *the* standard for smart cards and smart card operating systems. Compared with the latest developments in the smart card world, some of the commands and mechanisms in the GSM realm may appear outdated, but GSM was and still is the pioneer for large international smart card applications. Ultimately, all subsequent applications can only learn and benefit from the experience gained and problems encountered using this application. In many respects, GSM in the form of the GSM 11.11 and 11.14 specifications forms the foundation for all more recent and more sophisticated smart card applications.

Recent models of mobile telephones are incorporating an increasing number of the functions of personal digital assistants (PDAs), in addition to pure telephone functions. Since it is relatively difficult to externally manipulate the software of a mobile telephone, it can be considered to be a trusted device. The consequences of this can be seen in many service functions and telephones with hardware extensions. For example, there are mobile telephones with IrDA-compliant infrared interfaces or Bluetooth interfaces, as well as mobile telephones with larger and more powerful displays.

This makes it technically possible to use mobile telephone to make payments from an electronic purse at a suitably equipped POS station. If the user has to enter a PIN, in the future he can do so using his relatively tamper-proof telephone keypad instead of an unfamiliar terminal. The corresponding data can be exchanged using an infrared or Bluetooth interface, with no need to establish (and pay for) a telephone connection. The potential uses of such capabilities are extremely varied, so they can only be outlined in broad terms at present.

For a variety of reasons, dual-slot mobile telephones have failed to achieve widespread use. This is probably more due to the business strategies of network operators than technical reasons, such as the size of the mobile telephone. Up to now, network operators have shown little interest in encouraging the use of third-party applications in the smart cards of their highly subsidized mobile equipment. Presently, the development trend is focused on value-added services in SIMs. The wide-scale introduction of digital signature applications as part of WIM, which despite its name cannot be used for WAP, at least creates the necessary technical conditions for the entire spectrum of mobile business applications.

A microbrowser implemented in the SIM will doubtless continue to form the basis for secure data-based applications in the coming years, which could inevitably lead to a market shakeout between this technology and GSM-capable Java cards. However, in the first instance the primary uses for the latter types of cards will be in the area of value-added services based on program code.

MExE (Mobile Station Execution Environment) is a framework for integrating procedures defined by the network operator and executable program code into the mobile station. Stage 1 of MExE specifies the integration of WAP browsers for the WML markup language in the mobile equipment. The subsequent step, Stage 2, adds a Java virtual machine (JVM) to these functions. This allows Java programs to be loaded into mobile telephones and run there, and it allows value-added services to be implemented directly in the mobile telephone, rather than in the SIM (as is presently common).

CAMEL (Customized Applications for Mobile Enhanced Logic) provides the GSM network with a new option that extends functionality in the direction of intelligent networks (IN). With CAMEL, it is for example possible for the network to modify dialing numbers during call setup. This would permit services such as international roaming with prepaid cards or standardized international service numbers to be implemented significantly more simply than at present.

Even an established system such as GSM must be further developed in order to meet new requirements and satisfy additional customer desires. This is presently taking place in small steps, and it has led to modifications and extensions such as proactive SIM commands, OTA, WIM, microbrowsers and RFM, as well as extended capabilities such as HSCSCD, GPRS and EDGE. Nevertheless, at some point in time it will be necessary to make a major evolutionary step in order to convert all of these extensions, modifications and special cases into a new system that is once again self-contained. This new system will be the Universal Mobile Telecommunication System (UMTS), which provisionally can be expected to exist alongside the GSM system for many years, and which may at some time supplant the GMS system.

13.3 THE UMTS SYSTEM

In 1998, a group of five standards organizations consisting of ANSI T1 (USA), ARIB (Japan), ETSI (Europe), TTA (Korea) and TTC (Japan) initiated the Third Generation Partnership Project (3GPP), whose purpose was to specify a successor to the GSM in the form of an international IMT-2000-compliant mobile telecommunications system based on the GSM specifications. This mobile telecommunications system is generally known throughout the world as a 3G (third-generation) system, but in Europe it has predominantly come to be known as the Universal Mobile Telecommunication System (UMTS).[17] This system initially enjoyed widespread public interest due to the enormous license fees paid for the necessary frequency spectrum, rather than because of the new capabilities it will offer. In Germany alone, network operators paid approximately 50 billion euros for the UMTS frequency spectrum in an auction, and the total amount paid for the spectrum in Europe was 112 billion euros. Constructing the network will soak up another 30 billion euros in Germany alone.

A relatively short time later, the first UMTS network went into operation in Japan at the end of 2001. The development of UMTS was primarily pushed by a few countries, such as Japan, in which subscriber density is so high that there was no point in further extending existing mobile telecommunications systems. In the remainder of this section, some of the essential differences between UMTS and GSM are described.

For the air interface, which is called the 'UMTS radio access network' (UTRAN), UMTS uses code-division multiple access (CDMA) for communication between base stations and mobile stations. The transmission frequency is in the 2000-MHz band (wavelength approximately 15 cm), in compliance with IMT-2000. The architecture is very similar to that of GSM, with the main difference being that the components of the UMTS system are linked via IP.

From the smart card perspective, the greatest difference between GMS and UMTS is that UMTS uses a completely redefined security module called the 'universal subscriber identity module' (USIM). This security module is based on the ISO/IEC 7816 family of standards. It is thus the first such module in the world of smart cards for mobile telecommunications to guarantee compatibility with other smart cards specified in accordance with these standards, such as EMV-2000 compliant smart cards used in electronic payment systems.

At this point, we recommend that you carefully read Section 13.2 ('The GMS System') before continuing, since the SIM and USIM smart cards are very similar, and in the following material we primarily concentrate on the differences. As can be readily seen, the UMTS system is based on the GMS system, and many of the proven principles and mechanisms of the GMS system have been incorporated into the UMTS system.

[17] The term 'UTMS' is always used in this book instead of '3G', since it unambiguously describes a particular mobile telecommunication system that is only one of several 3G systems. For instance, CDMA-2000 is also a 3G system, but it has an optional smart card called the removable user identity module (R-UIM), which differs from the USIM for UMTS in many respects

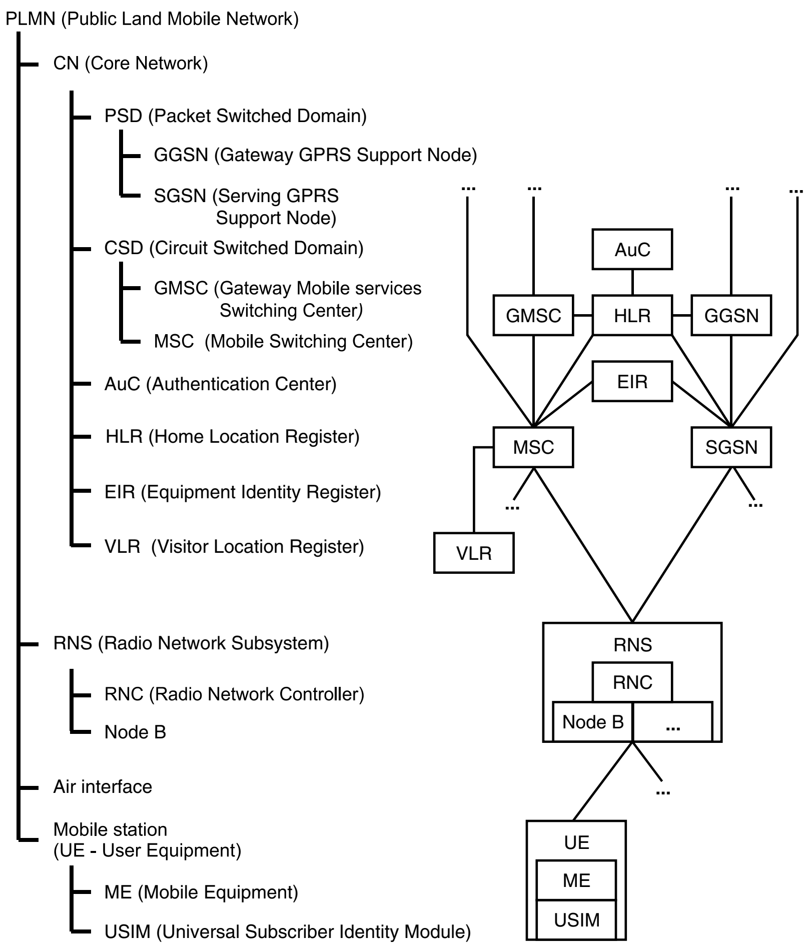

PLMN (Public Land Mobile Network)
CN (Core Network)
PSD (Packet Switched Domain)
GGSN (Gateway GPRS Support Node)
SGSN (Serving GPRS Support Node)
CSD (Circuit Switched Domain)
GMSC (Gateway Mobile services Switching Center)
MSC (Mobile Switching Center)
AuC (Authentication Center)
HLR (Home Location Register)
EIR (Equipment Identity Register)
VLR (Visitor Location Register)
RNS (Radio Network Subsystem)
RNC (Radio Network Controller)
Node B
Air interface
Mobile station (UE - User Equipment)
ME (Mobile Equipment)
USIM (Universal Subscriber Identity Module)
AuC
GMSC
HLR
GGSN
EIR
MSC
SGSN
VLR
RNS
RNC
Node B
UE
ME
USIM

Figure 13.24 The basic architecture and most important designations for the components of a typical mobile telecommunications system that is compliant with the TS 123.002 UMTS standard

'USIM' is the usual designation for the smart card application for UMTS, and this application resides in a UICC. Nevertheless, in practice the term 'USIM' is used not only for the application but also for the UMTS smart card, even though this is not entirely correct. The USIM is primarily the bearer of the identity of the subscriber, and its principal function is to ensure the authenticity of the mobile station with respect to the network and vice versa.

The specification for the USIM is based on the TS 102.221 specification, which is the fundamental specification for telecommunications smart cards and characterizes the physical and logical parameters of a 'universal integrated circuit card' (UICC). Based on this specification for a general-purpose smart card for telecommunications applications, the requirements

Table 13.10 The most important standards for the USIM

Standard	Title
TS 21.111	USIM and IC card requirements
TS 31.102	Characteristics of the USIM Application
TS 31.110	Numbering system for telecommunication IC card applications
TS 31.111	USIM Application Toolkit (USAT)
TS 31.121	USIM Application Test Specification
TS 31.122	USIM Conformance Test Specification
TS 102.221	Physical and Logical Characteristics
TS 102.222	Administrative Commands

specification TS 21.111 defines the basic requirements for the USIM application, which are in turn described in detail in TS 31.102. These specifications are complemented by TS 31.111, which contains an extensive description of the USIM Application Toolkit (USAT). The commands for managing applications in a UICC are contained in TS 102.222, which is now also used as a quasi-standard specification in the SIM environment. Finally, TS 31.122 contains specifications for conformity tests.

Table 13.11 The characteristic physical, electrical and logical properties of a USIM based on the UICC specification

Property	Remark
Card format	ID-1 or ID-000
Supply voltage	UICC: 1.8 V and/or 3 V and/or 5 V In practice, USIMs typically have the following voltage ranges: (1.8 V & 3 V) or (3 V & 5 V)

Transmission protocol	PPS (mandatory)
	T = 0 (mandatory)
	T = 1 (optional)

Logical channels	Up to 4
Commands	See the summary of commands in Table 13.12
Access conditions for files	As specified in ISO/IEC 7816-9

An operating system for UICCs must comply with the ISO/IEC 78126 family of standards in all of its essential properties, which means it must be fully multiapplication capable. This applies in particular to the commands, file management and the file access conditions, which are rule-based in accordance with ISO/IEC 7816-9, with access being controlled by an 'access rule reference' EF (EF_{ARR}). However, there are many similarities to the SIM with respect to commands and file management. With regard to file types, USIM has a special feature in the form of the 'application dedicated file' (ADF) type. This is a special type of DF containing all of the DFs and EFs for a particular application that does not have the MF as its root directory.

An ADF can thus be considered to be similar to an MF in this regard, since it does not have any higher-level file type. An ADF is selected using an AID stored in the EF_{DIR} file.

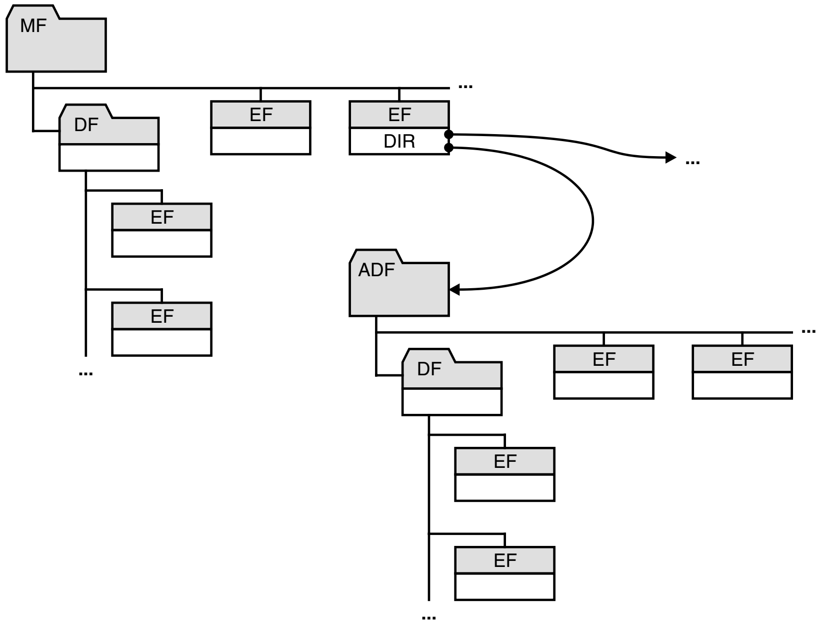


Figure 13.25 The relationships between the MF, DFs and ADFs in the UICC or USIM. An ADF can be selected using an AID stored in the EF_{DIR} file, and it contains all the files for an application

In the UICC, the USIM represents nothing more than one of many possible applications located in their own ADFs. Due to the multiapplication capability of the UICC, it is not particularly difficult to implement a SIM in addition to the USIM and thereby create a smart card that can be used in both UMTS and GSM systems. In fact, this will probably be the standard configuration for the foreseeable future, since logistics costs for the network operator can be reduced by having a single smart card for the two different mobile telecommunications technologies.

Most of the files for a USIM application can also be found in the same or similar form in a SIM. The only significant modification that has been made relates to the storage of dialing numbers. For USIM, a relatively obscure concept involving optional and mandatory files linked by pointers is specified. A wide variety of data for dialing numbers and subscribers can be stored in these files.

A USIM has two PIN codes called PIN 1 and PIN 2, which are fully analogous to CHV1 and CHV 2 in a SIM. The access conditions for the files are specified such that the card user knows PIN 1 and the cardholder knows PIN 2. This allows the majority of the functions of the mobile equipment to be flexibly managed.

Several cryptographic functions named f1, f2, f3, f4 and f5 ('function 1' through 'function 5') are used for USIM authentication in the UMTS system. They are used to authenticate

Table 13.12 Smart card commands for the USIM as specified in TS 102.221

Command	Brief description
Security commands	
CHANGE PIN	Change the PIN
DISABLE PIN	Disable PIN queries
ENABLE PIN	Enable PIN queries
UNBLOCK PIN	Reset the PIN retry counter from its terminal count
VERIFY PIN	Verify the PIN
AUTHENTICATE	Authentication of the USIM by the outside world
Commands for operations on files	
INCREASE	Increment a counter in a file
READ BINARY	Read from a file with a transparent structure
READ RECORD	Read from a file with a record-oriented structure
SEARCH RECORD	Search for a text string in a file with a record-oriented structure
SELECT	Select a file
STATUS	Read various data from the currently selected file
UPDATE BINARY	Write to a file with a transparent structure
UPDATE RECORD	Write to a file with a record-oriented structure
DEACTIVATE FILE	Reversibly block a file
ACTIVATE FILE	Unblock a file
USIM Application Toolkit commands	
ENVELOPE	Pass data to a value-added service of the USIM in the USIM Application Toolkit environment
FETCH	Retrieve a USIM Application Toolkit command from the USIM and provide it to the mobile equipment
TERMINAL PROFILE	List all functions of the mobile equipment in the USIM Application Toolkit environment
TERMINAL RESPONSE	Convey the response of the mobile equipment to a previous USIM Application Toolkit command of the USIM
Miscellaneous commands	
GET RESPONSE	Command specific to $T = 0$ for requesting data from the smart card
MANAGE CHANNEL	Control logical channels

the network and the USIM, as well as for establishing cryptographically secured communication via the air interface. The kernel of these security functions is a symmetrical cryptographic algorithm that can be parameterized using additional linked input values. The USIM specification proposes the MILENAGE algorithm as a sample algorithm.

Just as the SIM has a SIM Application Toolkit, an application toolkit called the USIM Application Toolkit (USAT) is specified for the USIM. It is nearly identical to the SAT, and like the SAT, it is used for implementing value-added services in the USIM. ETSI also specified a microbrowser called 'USAT Interpreter' for the USIM. Despite what may be suggested by its name, this microbrowser cannot be implemented in SIMs as well.

Table 13.13 The mandatory application-independent files for a USCC, which must therefore be present in a USIM

MF.EF$_{\text{DIR}}$		Application directory file (DIR)
	Description:	This file contains information about the applications present in the smart card.
	File:	FID = '2F00'; structure: linear fixed, number of records: ≥1, file size: n bytes; accesses: READ: always; UPDATE: ADM
	File content:	Each record contains a BER-TLV coded data object in accordance with ISO/IEC 7816-5 that describes an application in the smart card, with the mandatory specification of its AID.
MF.EF$_{\text{ICCID}}$		**ICC identification (ICCID)**
	Description:	This file holds a unique identification number for the smart card.
	File:	FID = '2FE2'; structure: transparent, file size: 10 bytes; accesses: READ: always; UPDATE: never
MF.EF$_{\text{PL}}$		**Preferred language (PL)**
	Description:	This file holds a list of the preferred languages for the user interface.
	File:	FID = '2F05'; structure: transparent, file size: 2*n* bytes; accesses: READ: always; UPDATE: PIN
MF.EF$_{\text{ARR}}$		**Access rule reference (ARR)**
	Description:	This file holds a list of the access rules for files directly below the MF.
	File:	FID = '2F06'; structure: linear fixed, file size: *n* bytes; accesses: READ: always; UPDATE: ADM
	File content:	Each record holds an access rule according to ISO/IEC 7816-9.

13.4 MICROBROWSERS

A significant portion of the success of the World Wide Web can without doubt be attributed to the web browsers. They made it possible to view stored content in the form of hypertext documents, navigate among these documents and run program code embedded in hypertext documents without undesirable side effects in the client computer, all without requiring computer-specific programs to be installed in the remote servers.

Browsers with simple structures and requiring only small amounts of memory and processing power are often referred to as 'microbrowsers'. Generally speaking, such browsers cannot

Table 13.14 The mandatory files for a USIM in addition to the mandatory application-independent files. For the coding of the data elements and illustrative decoded examples of the files, see Section 13.2.4, 'The subscriber identity module (SIM)'

MF		**Root directory**
	Description:	This is the source directory for the entire USIM.
	File:	FID = '3F00'
MF.DF$_{\text{GSM}}$		**GSM directory**
	Description:	This directory holds a collection of files specific to the GSM network.
	File:	FID = '7F20'
MF.DF$_{\text{TELECOM}}$		**Telecom directory**
	Description:	This directory holds all files specific to the services.
	File:	FID = '7F10'
MF.DF$_{\text{PHONEBOOK}}$		**Telephone book directory**
	Description:	This directory holds all files belonging to the telephone book.
	File:	FID = '5F3A'
ADF$_{\text{USIM}}$		**USIM application directory**
	Description:	This directory holds all files belonging to the USIM application.
	File:	AID = RID = 'A0 00 00 00 87'
ADF$_{\text{USIM}}$.DF$_{\text{PHONEBOOK}}$H		**Telephone book directory**
	Description:	This directory holds all files belonging to the telephone book
	File:	FID = '5F3A'
ADF$_{\text{USIM}}$.DF$_{\text{GSM - ACCESS}}$		**GSM directory**
	Description:	This directory holds all files needed for access to the GSM network.
	File:	FID = '5F3B'

(*Cont.*)

Table 13.14 (*Cont.*)

$ADF_{USIM}.DF_{MExE}$		**Mobile station execution environment (MExE) directory**
	Description:	This directory holds all files specific to MExE.
	File:	FID = '5F3C'
$ADF_{USIM}.EF_{IMSI}$		**International mobile subscriber identity (IMSI)**
	Description:	This file holds the international subscriber identity.

	File:	FID = '6F07'; SFI = '07';
		structure: transparent,
		file size: 9 bytes;
		accesses: READ: PIN; UPDATE: ADM

$ADF_{USIM}.EF_{Keys}$		**Keys**
	Description:	This file holds the encryption key CK (ciphering key), the integrity testing key IK (‘integrity key’) and the key identifier KSI (key set identifier).

	File:	FID = '6F08'; SFI = '08';
		structure: transparent,
		file size: 33 bytes;
		accesses: READ: PIN; UPDATE: PIN

$ADF_{USIM}.EF_{KeysPS}$		**Keys for packet-switched services (packet-switched domain)**
	Description:	This file holds keys for the packet-switched services, consisting of the encryption key CKPS (ciphering key packet switched domain), the integrity testing key IKPS (integrity key packet switched domain) and the key identifier KSIPS (key set identifier packet switched domain).

	File:	FID = '6F09'; SFI = '09';
		structure: transparent,
		file size: 33 bytes;
		accesses: READ: PIN; UPDATE: PIN

$ADF_{USIM}.EF_{HPLMN}$		**Home public land mobile network search period (HPLMN)**
	Description:	This file holds a time interval for searching for the home network.

	File:	FID = '6F31'; SFI = '12';
		structure: transparent, 1 byte;
		accesses: READ: PIN; UPDATE: ADM

Table 13.14 (*Cont.*)

$\text{ADF}_{\text{USIM}}.\text{EF}_{\text{UST}}$		**USIM service table (UST)**
	Description:	This file holds a table of available and activated services supplementary to the voice service.

	File:	FID = '6F38'; SFI = '12';
		structure: transparent,
		file size: $\geq$ 1 byte;
		accesses: READ: PIN; UPDATE: ADM

$\text{ADF}_{\text{USIM}}.\text{EF}_{\text{ACC}}$		**Access control class (ACC)**
	Description:	This file holds data regarding network accesses attempts and access priorities.

	File:	FID = '6F78'; SFI = '12';
		structure: transparent, 1 byte;
		accesses: READ: PIN; UPDATE: ADM

$\text{ADF}_{\text{USIM}}.\text{EF}_{\text{FPLMN}}$		**Forbidden public land mobile network (FPLMN)**
	Description:	This file holds a list of forbidden network operators.

	File:	FID = '6F7B'; SFI = '0D';
		structure: transparent, $3n$ bytes;
		accesses: READ: PIN; UPDATE: PIN

$\text{ADF}_{\text{USIM}}.\text{EF}_{\text{LOCI}}$		**Location information (LOCI)**
	Description:	This file holds current location data for the mobile telephone.

	File:	FID = '6F7E'; SFI = '0B';
		structure: transparent,
		file size: 11 bytes;
		accesses: READ: PIN; UPDATE: PIN

$\text{ADF}_{\text{USIM}}.\text{EF}_{\text{AD}}$		**Administrative data (AD)**
	Description:	This file holds the administrative data for the USIM.

	File:	FID = '6FAD'; SFI = '03';
		structure: transparent,
		file size: $(4 + n)$ bytes;
		accesses: READ: always; UPDATE: ADM

(*Cont.*)

Table 13.14 (*Cont.*)

$ADF_{USIM}.EF_{ECC}$		**Emergency call codes (ECC)**
	Description:	This file holds the emergency dialing numbers.

	File:	FID = '6FB7'; SFI = '01';
		structure: linear fixed,
		file size: (4+*n*) bytes;
		accesses: READ: always; UPDATE: ADM

$ADF_{USIM}.EF_{PSLOCI}$		**Packet-switched location information (PSLOCI)**
	Description:	This file holds information about the current location of the mobile telephone for the packet-switched services.

	File:	FID = '6F73'; SFI = '0C';
		structure: transparent,
		file size: 14 bytes;
		accesses: READ: PIN; UPDATE: PIN

$ADF_{USIM}.EF_{START - HFN}$		**Initial value of the hyperframe number**
	Description:	This file holds information for data transmission via the air interface.

	File:	FID = '6F7B'; SFI = '0F';
		structure: transparent,
		file size: 6 bytes;
		accesses: READ: PIN; UPDATE: PIN

$ADF_{USIM}.EF_{THRESHOLD}$		**Maximum value of the hyperframe number**
	Description:	This file holds information for data transmission via the air interface.

	File:	FID = '6F5C'; SFI = '10';
		structure: transparent,
		file size: 3 bytes;
		accesses: READ: PIN; UPDATE: ADM

$ADF_{USIM}.EF_{ARR}$		**Access rule reference (ARR)**
	Description:	This file holds a list of access rules for the files in the ADF.

	File:	FID = '6F06'; SFI = '17';
		structure: linear fixed,
		file size: n bytes;
		accesses: READ: always; UPDATE: ADM

Table 13.14 (*Cont.*)

MF.EF$_{NETPAR}$		**Network parameters (NETPAR)**
	Description:	This file holds data related to the physical parameters of data transmission via the air interface.

	File:	FID = '2FC4';
		structure: transparent,
		file size: $\geq$ 46 bytes;
		accesses: PIN: always; UPDATE: PIN

run program code embedded in hypertext documents, and they are also adapted to a particular target system with regard to memory usage and the required processing capacity. A microbrowser can be regarded as sort of extension to a smart card operating system, in which case it is categorized as a 'SIM microbrowser'. The alternative is a microbrowser integrated into the software of the mobile telephone, which is called a 'mobile equipment (ME) microbrowser'. Probably the best known example of such a microbrowser is the WAP browser, which runs fully in the mobile telephone and only optionally accesses a smart card application called 'wireless identification module' (WIM)[18] in order to use cryptographic functions.

SIM-based microbrowsers were first announced for the GMS environment in 1998 by the Across Wireless, which is now known as Smarttrust [Smarttrust]. These microbrowsers can interpret a WML dialect that has been optimized to meet the needs of SIMs, and they also allow supplementary functions, such as generating digital signatures using the RSA algorithm, to be retrofitted by means of downloadable plug-ins. The size of the program code of a typical microbrowser is around 15 to 25 kB, with an additional 1 to 2 kB being required in RAM or EEPROM for working buffers.

In 1999, the SIM Alliance [SIM Alliance] consortium was founded by Gemplus, Giesecke & Devrient, ORGA and Schlumberger, with the objective of allowing services developed for WAP to also be used in mobile telephones lacking WAP capability. At that time, this was still a very attractive possibility, since the market penetration of WAP-enabled mobile telephones was rather small. For the realization of such capability, the SIM must have a SIM-Alliance compatible microbrowser (also called a 'S@T browser'), while the mobile telephone only needs to support GSM Phase 2+. This gives the SIM sufficient control over the mobile telephone via the SIM Application Toolkit to allow portions of WML contents and their functionalities to be reproduced.

A major advantage of microbrowsers in the SIM relative to microbrowsers in the mobile equipment is their secure execution environment. It is certainly conceivable that a WAP browser that runs in the mobile equipment could have its various functions manipulated to the point that it would be possible to make both the processor and the memory accessible to manipulation. A SIM microbrowser, by contrast, is fully integrated into a SIM and can thus take advantage of all protective mechanisms inherent to smart cards. With a SIM microbrowser, an end-to-end link at the application level can be implemented between an application on the application server and the SIM without any loss of security. This is not possible with a WAP browser.

The Smarttrust browser and the SIM Alliance browser are both company- or consortium-specific browsers that do not originate from a standardization body. They are also not mutually

[18] See also Section 13.5, 'The WIM'

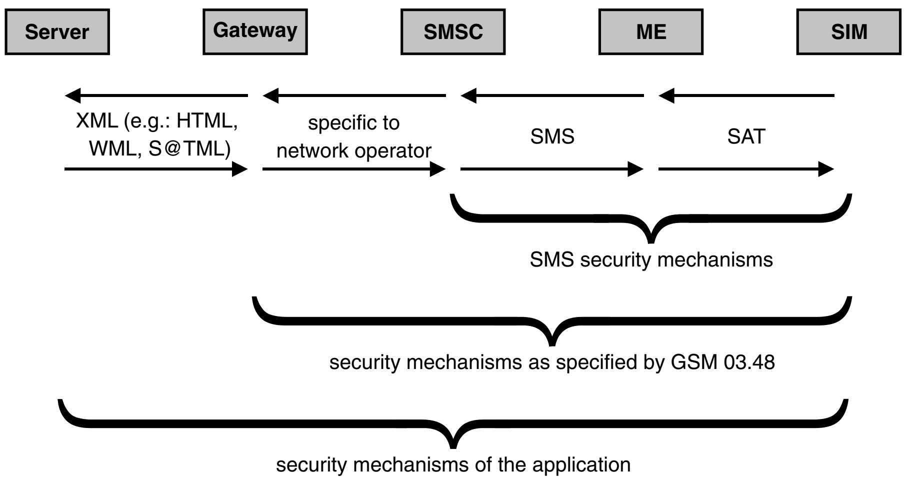

Server
Gateway
SMSC
ME
SIM
XML (e.g.: HTML, WML, S@TML)
specific to network operator
SMS
SAT
SMS security mechanisms
security mechanisms as specified by GSM 03.48
security mechanisms of the application

Figure 13.26 Schematic representation of the basic data transmissions and hierarchically structured security mechanisms of microbrowser applications

Listing 13.2 Example of a simple WML application. The WML document is located on a server. Using a converter, it is transformed into a 20-byte WML bytecode and then transmitted to the microbrowser in the SIM using a SMS message. The microbrowser interprets the WML bytecode and then uses the SIM Application Toolkit DISPLAY TEXT command to output the text 'Hello World' to the display of the mobile equipment

`<?xml version="1.0"` `encoding="ISO-8859--1"?>`	WML version and character encoding specification
`<wml>`	Start of the WML document
`  <card>`	Start of a new card
`    Hello World`	Text to be displayed
`  </card>`	End of the card
`</wml>`	End of the WML document
`'00 12 02 0C 48 65 6C 6C 6F 20 57 6F 72 6C 64 20 06 00 06 00'`	WML document converted to WML bytecode

compatible. However, at the end of 2001 ETSI published the first version of the specification for the USIM Application Toolkit Interpreter (USAT Interpreter), which represents the result of a working group founded in mid-2000 to address this set of issues. Contrary to what its name suggests, the USAT Interpreter can quite definitely be implemented in SIMs as well as USIMs. It is likely that the USAT Interpreter will fully supplant both the Smarttrust browser and the SIM Alliance browser in the medium term. The name 'interpreter' is also based on the new capability of incorporating branches in interpreted bytecode. There are also a variety of variable types that can be used from within bytecode for intermediate storage across session boundaries. Browsers are by their nature typical examples of the 'pull principle', which means transferring information by fetching information from a higher-level system. Nevertheless, the USAT Interpreter also includes a push function, which refers to the transfer of information by sending information from a higher-level system to a lower-order system. This is very attractive for certain applications, since it allows users of mobile telephones to be explicitly informed of certain things by the application server.

Table 13.15 The most important standards for the USAT Interpreter

Standard	Title
TS 22.112	USAT Interpreter – Stage 1
TS 31.112	USAT Interpreter Architecture Description; Stage 2
TS 31.113	USAT Interpreter Byte Codes
TS 31.114	USAT Interpreter Protocol and Administration

Generally speaking, an online system (such as GSM or UMTS) must be available before a microbrowser can be used in a smart card. The contents to be displayed or executed are located in an application server belonging to the content provider. This server is connected to a gateway server by means of the usual Internet protocols and secure communication mechanisms (such as SSL and TSL), which guarantee the application server the appropriate communications security. In the case of communication with the SIM via SMS messages, the gateway server is in turn connected to a short message service center (SMSC). The data to be sent to the SIM are reformatted into GSM 03.48-compliant data packets, and at the same time they are provided with the required cryptographic features for secure communications. The data packets are then sent transparently to the SIM via the GMS system using SMS messages. The SIM recognizes that these messages contain GSM 03.48-compliant data, decodes the data accordingly and if necessary reassembles the original message from several short messages. If no errors have occurred, the result is then passed to the SIM microbrowser, which interprets the message

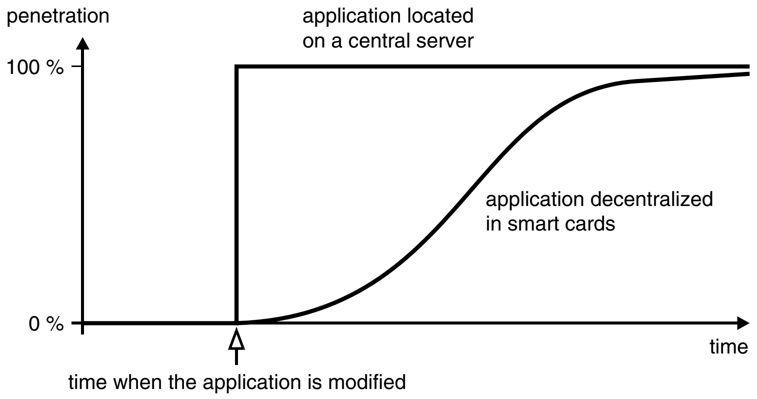


Figure 13.27 Penetration curves for a new application, showing the difference between centralized storage of the application on a server and decentralized storage in SIMs. Depending on the particular mobile telecommunications network and the application in question, it can take up to several weeks to achieve 90 % penetration

and executes the corresponding SIM Application Toolkit commands. Finally, a response may optionally be generated and returned to the application server using the reverse sequence.

There are cases in which the microbrowser does not receive its messages in the form of online content from an application server connected via the network, but instead in the form of XML-compliant bytecode stored in a file in the SIM. This offline content provides an elegant way to store frequently used applications locally for rapid access, while also reducing the network load in the process. However, the drawback of this approach is that these locally stored applications must be explicitly updated by the background system. The principal advantage of applications that are centrally stored on a server is that such individual updating is not necessary.

13.5 THE WIRELESS IDENTIFICATION MODULE (WIM)

In 2001, 'WAP' was widely extolled in the mass media as the long-awaited new Internet technology for mobile telephones. For a variety of reasons, including a lack of content, poor availability of suitable mobile telephones, low data transmission rates and disappointing reproduction of content on the displays of mobile telephones, some of which were text-based and monochrome, WAP was unable to gain a foothold among end users. 'WAP' has now come to be regarded as a synonym for useless multibillion-euro investments in technologies that miss the mark with end users. However, it is entirely possible that following this phase of practically boundless disappointment, sooner or later a sense of reality will prevail and WAP will come to be regarded and used as a simple means to view contents that happen to be largely text-based.

The term 'WAP', which is short for 'wireless application protocol', actually refers to a number of specifications for implementing connections between mobile terminals (mobile telephones, PDAs etc.) and a server via a wireless network (such as GSM, CDMA or TDMA), for the direct exchange of data. Presently, the most common application for WAP is implementing Internet services in mobile telephones in a manner that is largely independent of the mobile telecommunications standard used. Nevertheless, besides the technology referred to in the common usage of the term, the designation 'wireless application protocol' also refers to the protocol used between the terminal device and the background system.

The internationally active standardization body for WAP is the WAP Forum [WAP], which was founded in June 1997 by Phone.com, Ericsson, Motorola and Nokia and is presently composed of representatives of more than 350 companies.

Since June 2000, the specifications for WAP also include a security module in the mobile device called the 'wireless identification module' (WIM). Only with such a module is it possible to establish secure communications with applications in the background system in a reasonable manner. Two possible versions of the WIM have been specified: as an application in a dedicated smart card, or as a supplementary application in a SIM, USIM or UICC. Since dual-slot mobile telephones have failed to become established in the market, the second version, in which the WIM is a supplementary application in an existing smart card, will come to prevail.

The WIM specification is based on the PKCS #15 specification [RSA] for digital signature applications in smart cards, which are now also addressed in a compatible form by ISO/IEC 7816-15. In order to understand the following material, you should have at least a cursory knowledge of PKCS #15-compatible signature applications. If necessary, you should read Section 14.5, 'PKCS #15-Compliant Signature Applications', before continuing.

The primary task of the WIM is to provide cryptographic functions for securing WAP data transmissions at the transport level in the form of 'wireless transport layer security' (WTLS).

Interestingly enough, these functions of the WIM can be used not only in the WAP context, but also in the environment of the SIM Application Toolkit. This makes the WIM a general-purpose signature application for telecommunications cards. Consequently, it has become a standard component in many SIMs and USIMs.

The WIM specification contains a number of primitives in abstract form that describe basic functions. Table 13.16 presents a summary of these generic functions, which are used in the WTLS protocol. In a subsequent stage, these primitives are linked to suitable commands or command sequences.

Table 13.16 The primitives defined in the WIM specification and related smart card commands

Primitive	Smart card command
Device control	
Open Service	MANAGE CHANNEL
Close Service	MANAGE CHANNEL
Verification	
Perform Verification	VERIFY
Disable Verification Requirement	DISABLE VERIFICATION REQUIREMENT
Enable Verification Requirement	ENABLE VERIFICATION REQUIREMENT
Change Reference Data	CHANGE REFERENCE DATA
Unblock Reference Data	RESET RETRY COUNTER
Data access	
Open File	SELECT
Close File	—
Read Binary	READ BINARY
Update Binary	UPDATE BINARY
Cryptography	
Compute Digital Signature	MANAGE SECURITY ENVIRONMENT PERFORM SECURITY OPERATION
Verify Signature	MANAGE SECURITY ENVIRONMENT PERFORM SECURITY OPERATION
Get Random	ASK RANDOM
Key Transport	MANAGE SECURITY ENVIRONMENT PERFORM SECURITY OPERATION
Key Agreement	MANAGE SECURITY ENVIRONMENT PERFORM SECURITY OPERATION
Derive Master Secret	MANAGE SECURITY ENVIRONMENT
PHash	MANAGE SECURITY ENVIRONMENT PERFORM SECURITY OPERATION
Decipher	MANAGE SECURITY ENVIRONMENT PERFORM SECURITY OPERATION
Exception	
Exception	This can apply to any type of command.

WIM files generally have a transparent structure in order to simplify access and avoid creating unnecessary overhead. From a software engineering perspective, this does not make access any more difficult, since the pointer structure on which PKCS #15 is based ensures that access to the desired data objects can be made in a relatively easily understood and extensible manner. All data are described in ANS.1 format and coded using the Distinguished Encoding Rules (DER).

Besides the usual PKCS #15 files and data objects, the WIM also has a ‘peers’ data object and a ‘sessions’ data object. These two data objects are used to store specific linkage data in order to allow an abbreviated authentication handshake to be used in a subsequent session.

With regard to its physical properties, the WIM must comply with the applicable GSM specifications, such as GSM 11.11. It must support the T = 0 transmission protocol; the T = 1 protocol can also be optionally provided by the WIM. This can be selected as necessary by the terminal via a PPS process. Table 3.17 provides a summary of the basic properties of a WIM.

Table 13.17 Basic properties of a WIM

Property	Remark
Card format: ID-000	Optional
Card format: ID-1	Optional
Commands	See command summary in Table 13.18.
Logical channels	Optional
T = 0 transmission protocol	Mandatory
T = 1 transmission protocol	Optional
3-V supply voltage	Mandatory
5-V supply voltage	Optional

The commands needed for executing the functions with the WIM application are fully compliant with ISO/IEC 7816-4 and -8. This makes it relatively easy to implement a WIM in a smart card having a standard-compliant operating system, such as a UICC.

In order to invoke the WIM application, in practice a new logical channel is opened to the smart card (if this is supported by the operating system). The first step is then to select the application using its AID. In this case the RID has the value 'A0 00 00 00 63', and the associated PIX is the ASCII encoding of "WAP-WIM", which has the hexadecimal value '57 41 50 2D 57 49 4D'. After the application has been selected, the usual command sequences for WLTS or the signature application are used.

13.6 PUBLIC CARD PHONES IN GERMANY

Starting in the summer of 1989, the German state telephone company (Telekom) began introducing public card phones throughout the entire country. Prior to this, several field tests were carried out using systems from various manufacturers. These started as early as 1983 in four different regions in Germany with different characteristics (conurbations, cities and rural areas). Several different types of cards were also tested in these trials, including magnetic-stripe cards, hologram cards and cards made from different types of material (paper and plastic).

The conclusion drawn from these initial field tests was that plastic memory cards were the most suitable for use with card phones. The decisive factors were the achievable level of security and upward compatibility, as compared with other types of cards. In December of 1986 and July of 1987, large-scale field trials of a system based on memory cards were started in 16 major cities. These were successfully concluded in May of 1989. In the spring of 1999, there were more than 90,000 public card phones in Germany, and more than 300 million prepaid phone cards had been sold since the system was introduced. Internationally, the numbers at that time were 8.3 million card phones and more than 1100 million cards.

The card phones installed by Telekom in Germany can in principle be used with two different types of cards. The first type is the phone card, which is produced and used in very large numbers. In financial terms, this is an electronic purse that the customer purchases before using it. In technical terms, it is a memory card with an irreversible counter, a security feature and synchronous data transmission.

Table 13.18 Summary of the smart card commands defined in the WIM specification

Command	Brief description
Logical channels	
MANAGE CHANNEL	Control logical channels; optional command per ISO/IEC 7816-4.
Verification	
VERIFY	Verify passed-in PIN data; command per ISO/IEC 7816-4.
ENABLE VERIFICATION REQUIREMENT	Enable PIN queries; optional command per ISO/IEC 7816-8.
DISABLE VERIFICATION REQUIREMENT	Enable PIN queries; optional command per ISO/IEC 7816-8.
CHANGE REFERENCE DATA	Change the PIN; command per ISO/IEC 7816-8.
RESET RETRY COUNTER	Reset the retry counter; command per ISO/IEC 7816-8.
Data storage	
SELECT	Select a file; command per ISO/IEC 7816-4.
READ BINARY	Read from a file with transparent structure; command per ISO/IEC 7816-4.
UPDATE BINARY	Write to a file with transparent structure; command per ISO/IEC 7816-4.
Cryptographic operations	
MANAGE SECURITY ENVIRONMENT	Modify the parameters for using cryptographic algorithms in the smart card; command per ISO/IEC 7816-8.
PERFORM SECURITY OPERATION	Run a cryptographic algorithm in the smart card; command per ISO/IEC 7816-8.
ASK RANDOM	Request a random number from the smart card; command per ISO/IEC 7816-4.
Miscellaneous	
GET RESPONSE	T = 0 command for requesting data from the smart card; command per ISO/IEC 7816-4.

The other type of card is the telephone charge card, which is not as widely used. It resembles a credit card in that the user pays for the services (telephone calls) only after they have been received, by means of a monthly settlement against his or her bank account. In technical terms, a charge card is a smart card with a microcontroller. Data transmission to the card phone takes place using the block-oriented, asynchronous T = 14 transmission protocol. Since this type of card is not particularly important, we will not discuss it any further.

The entire card phone system is constructed as a distributed (decentralized) system with several successive layers of computers. In the event of a system breakdown, the lower levels can work fully autonomously for several days without affecting normal telephone operations. The two parts of the system that the normal user sees, which are the phone cards and the card phones, are described here.

Table 13.19 Examples of typical files in a WIM[19]

MF.DF$_{\text{WIM}}$		**WIM application**
	Description:	This directory holds all files belonging to the PKCS #15-compliant WIM application.
	File:	AID = 'A0 00 00 00 63 57 41 50 2D 57 49 4D'
MF. DF$_{\text{WIM}}$.EF$_{\text{TokenInfo}}$		**General information file (TokenInfo)**
	Description:	This file holds general information about the WIM and the functions supported by the WIM.
MF. DF$_{\text{WIM}}$.EF$_{\text{ODF}}$		**Object directory file (ODF)**
	Description:	This file holds a directory of all files containing data objects that are present in the card, with corresponding pointers to these files.
MF. DF$_{\text{WIM}}$.EF$_{\text{AODF}}$		**Authentication object directory file (AODF)**
	Description:	This file holds a list of the available authentication objects and corresponding pointers to these objects.
MF. DF$_{\text{WIM}}$.EF$_{\text{PrKDF}}$		**Private key directory file (PrKDF)**
	Description:	This file holds a list of the available private keys and corresponding pointers to these keys in EF$_{\text{PrKDFData\_1}}$ and EF$_{\text{PrKDFData\_2}}$.
MF.DF$_{\text{WIM}}$.EF$_{\text{PrKDFData\_1}}$		**Private keys for authentication and key exchange**
	Description:	This file holds the private keys for authentication and key exchange.

[19] See also Section 14.5, 'The PKCS #15 Signature Application'

Table 13.19 (*Cont.*)

MF.DF$_{\text{WIM}}$.EF$_{\text{PrKDFData\_2}}$		**Private keys for digital signatures**
	Description:	This file holds the private keys for digital signatures.
MF.DF$_{\text{WIM}}$.EF$_{\text{CDF\_1}}$		**Certificate directory file (CDF)**
	Description:	This file holds a list of the available private certificates for the user and corresponding pointers to these certificates in EF$_{\text{CDFData\_1}}$ and EF$_{\text{CDFData\_2}}$.
MF.DF$_{\text{WIM}}$.EF$_{\text{CDFData\_1}}$		**Authentication and key exchange certificates for users**
	Description:	This file holds the certificates for authentication and key exchange for the user.
MF.DF$_{\text{WIM}}$.EF$_{\text{CDFData\_2}}$		**Signature certificates for users**
	Description:	This file holds the certificates for digital signatures for the user.
MF.DF$_{\text{WIM}}$.EF$_{\text{CDF\_2}}$		**Certificate directory file (CDF) – for certificates from the certification authority**
	Description:	This file holds a list of the available certificates from the certification authority that can be modified by the user, as well as corresponding pointers to these certificates in EF$_{\text{CDFData\_3}}$.
MF.DF$_{\text{WIM}}$.EF$_{\text{CDFData\_3}}$		**Certificates from the certification authority for the user**
	Description:	This file holds the certificates of the certification authority that can be modified by the user.
MF.DF$_{\text{WIM}}$.EF$_{\text{CDF\_3}}$		**Certificate directory file (CDF) – for read-only certificates from the certification authority**
	Description:	This file holds a list of the available certificates from the certification authority that cannot be modified, as well as corresponding pointers to these certificates in EF$_{\text{CDFData\_4}}$.
MF.DF$_{\text{WIM}}$.EF$_{\text{CDFData\_4}}$		**Read-only certificates from the certification authority**
	Description:	This file holds the certificates of the certification authority that cannot be modified.

(*Cont.*)

Table 13.19 (*Cont.*)

MF.DF$_{\textbf{WIM}}$.EF$_{\textbf{DODF}}$		**Data object directory file (DODF)**
	Description:	This file holds a list of the available master secrets for the WTLS session and corresponding pointers to these objects.
MF.DF$_{\textbf{WIM}}$.EF$_{\textbf{DODFData\_1}}$		**Data objects**
	Description:	This file holds the available master secrets for the WTLS session.
MF.DF$_{\textbf{WIM}}$.EF$_{\textbf{Peers}}$		**Peers file**
	Description:	This file holds data for the WTLS authentication handshake.
MF.DF$_{\textbf{WIM}}$.EF$_{\textbf{Sessions}}$		**Sessions file**
	Description:	This file holds data for the WTLS authentication handshake.
MF.DF$_{\textbf{WIM}}$.EF$_{\textbf{UnusedSpace}}$		**Free memory directory file**
	Description:	This file holds a list of memory locations that are available in the EFs of the WIM application but are not used.

In order to survive as long as possible under the severe operational conditions to which it is exposed, the card phone has a sturdy metallic enclosure with openings for the controls and indicators, such as the keypad and display. The terminal is fully electrically isolated from the rest of the electronic assemblies. It is also short-circuit-proof, in order to protect it against vandalism and other attempts to disturb its operation by shorting its contacts.

The card phone is controlled by a powerful microcontroller. The control software can be remotely updated, which means that it is not necessary for a service technician to update the software on site by replacing EPROMs. The control processor of the card phone can also directly exchange information with the higher-level computer systems while a conversation is taking place, using a data-over-voice (DOV) modem. This is primarily necessary for telephone charge cards, since the accumulated charges are immediately sent to the background system for billing.

The main control processor of the card phone also handles communication with the phone card or charge card. With a charge card, it communicates using the T = 14 protocol, which is specified by Telekom and is used only in Germany. Otherwise, it uses the synchronous protocol employed by the memory card in question.

The built-in terminal can supply the memory card with an external programming voltage that is adjustable between 5 V and 25.5 V in 255 steps. However, since practically all new memory cards have an internal voltage converter that generates the programming voltage from the normal supply voltage, this is actually no longer technically necessary. It is only present for compatibility with older generations of phone cards, whose contents are still good.

If a charge card is used (a ‘real’ smart card), the terminal can choose a clock frequency ranging from 1.2 to 9.8 MHz. With synchronous cards, the clock frequency must be reduced to around 20 kHz in order to obtain usable communications.

The terminals have massively constructed shutters for protection against ‘dummy’ cards. The shutter has an impact cutter that slices through any wires or cables that pass through the card slot. This prevents tapping or manipulating the communications between the card and the terminal.

The chips that are used for phone cards have both ROM (which can be mask programmed by the semiconductor manufacturer) and EEPROM. These hold card-specific data and the counter for the card balance.

A charge pump in the chip generates the programming voltage for the EEPROM, so it is no longer necessary to supply this voltage externally. This means that the chip needs only the normal supply voltage. For protection against fraud using counterfeit cards, the chip has a hardware security feature whose operation is secret. In the future, memory chips that allow unilateral authentication of the card by the outside world will also be used.

Modern phone card chips have only six contacts. This is because only five contacts are actually necessary for the full functionality of the memory chip, with all other contacts being unused. Using eight contacts would increase production costs, since the module would be larger and thus more expensive. It would also take longer to mill the cavity for the module in the card body, thus reducing the throughput of the production equipment and increasing the unit cost. Consequently, practically all new phone cards have only six contacts. The six or eight leads brought out from the chip to the contacts are assigned as shown in Figure 13.28.

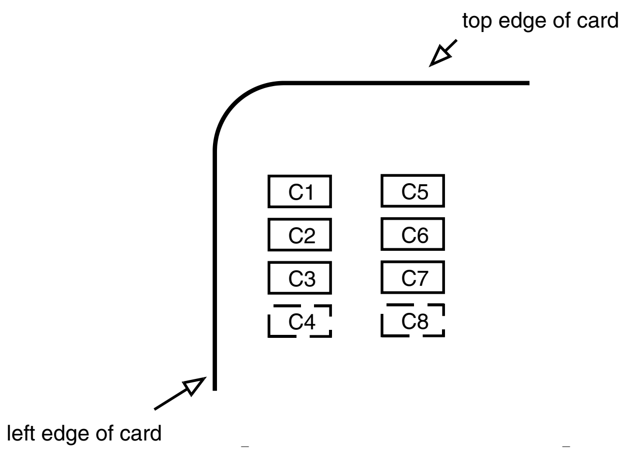


Figure 13.28 Contact assignments for a phone card. The contacts that are not listed are not used.

C1 supply voltage (5 V ± 5 %)	C5 ground
C2 control input	C7 data transmission
C3 clock input (≈20 kHz)	

Prepaid Telekom phone cards contain the data described below in the chip memory.

Serial number Each phone card has a 7-digit serial number. This allows individual cards to be blocked if there is suspicion of fraud. The serial number is stored in a part of the EEPROM that is blocked against overwriting or erasing.

Date of manufacture When the chip is manufactured, the year and date of its fabrication are permanently written to the EEPROM. This date refers only to the embedded chip, rather than the card.

Manufacturer code After the chip has been embedded in the card body, the card manufacturer writes a specific code to the memory of the card. This identifies the manufacturer, and it cannot later be altered.

Initial value The initial value of the card is also stored in every phone card, in addition to the current value.

Remaining balance This is the only data element in the card that is functionally necessary for a prepaid phone card. The card phone deducts the individual billing units one after the other from a five-digit irreversible octal counter in the EEPROM. The counter thus consists of five bytes of eight bits each. This gives a maximum count capacity of 32,768 (8^5). When the card is completed, the counter is set to the desired initial value of the card, which means that its count is equal the value of the card in units of eurocents. After this, the counter can be decremented toward zero whenever the card is used. Once the counter has reached zero, the card has been fully used up.

Transaction process

If you insert a phone card in a card phone, the first thing that happens is that the shutter closes the card slot, and at the same time the terminal's contacts are applied to the contact surfaces of the card. After this, the card phone executes an ISO activation sequence to initialize the card. Following this, the terminal sets the address pointer in the card to zero and reads the first 16 bits. If this is successful, the phone card is in good working order. The 16 bits are interpreted by the terminal as a sort of ATR containing various operating parameters for the card. The terminal next checks these parameters to see whether the inserted card is one of the allowed card types whose functions it can properly use. The card number, which the terminal has also read from the card, is at the same time sent via the DOV modem to the next higher level of the system. Here it is compared with the blacklist of blocked cards. If the blacklist contains no entry for this card, the card may be used, and the card phone receives a corresponding response.

The address pointer is now set to point to the region of the tariff counter, and the current value of the card is read from the EEPROM and shown on the display. After the user has entered the telephone number and a connection has been successfully made, the card phone is regularly advised via the DOV modem when it is time to deduct charge units. Each time this occurs, the card phone decrements the counter in the memory card and immediately reads out its new value. This allows it to check whether the amount was correctly deducted. If this is not the case, the connection is immediately broken. If the card phone calculates that the remaining balance of the card will be used up within the next 20 seconds, it generates a warning tone, and the card can be exchanged for a new one without interrupting the conversation.[20]

[20] Devaluation cycle for a memory card; see also Section 12.2, 'Prepaid Memory Cards'

14 Sample Applications

In this chapter, we present some sample smart card applications and describe them in broad terms. They are all based on the previous chapters, which deal with all relevant aspects of smart card technology, and they illustrate the extent and complexity of relatively large smart card applications.

Another objective of this chapter is to show smart card systems in which the cards are only one of several components. In such systems, the functionality, user friendliness and (in most cases) nearly all of the system security depend on the smart cards used in the system. However, such systems should always be viewed as a whole, since they operate satisfactorily only when all of their components work together harmoniously.

14.1 CONTACTLESS MEMORY CARDS FOR AIR TRAVEL

The system described in this section differs from the usual applications for smart cards in several basic aspects. These relate to the fundamental system architecture and data transmission mechanism, as well as how the cards are powered. This system is the ticket-free flight system of the German airline Lufthansa. It is based on contactless memory cards, which do not have to be inserted into a terminal.[1] The basic system concept also differs from that of all other smart card applications described here, in that the cards are only used to identify the users, with all application data being held in the background system.

For some time now, Lufthansa has been issuing frequent-flyer cards and cards for its bonus system. Originally, they used embossed cards and cards with magnetic stripes for automatic processing. In addition, certain types of cards could also be provided with a chip, so that they could be used with German public card phones and have a supplementary credit card function. This existing family of cards was to be extended by adding supplementary applications for boarding and ticketing in an upwardly compatible manner. In addition to the requirement for compatibility, a second objective was to make as few modifications as possible to existing systems. An equally important consideration was that the new cards should be easy for customers to use.

[1] See also Section 2.3.3, 'Contactless smart cards'

Smart Card Handbook, Third Edition. W. Rankl and W. Effing
© 2004 John Wiley & Sons, Ltd ISBN: 0-470-85668-8

In the ultimate stage of development, this would result in a multiapplication card with a magnetic stripe, embossing, a hologram, a memory chip with contacts and a contactless memory chip. Such a card could provide a large variety of functions without creating any compatibility problems with previously issued cards.

Before the planned system-wide introduction of the new cards, a pilot project was conducted on the Frankfurt–Berlin route. The new cards were issued to 600 customers of all different types, and many thousands of flights were made on this route between May and December of 1995. To allow customers to use the cards, suitable automated service kiosks were installed at both airports. Each kiosk was a PC-based system with a touch-sensitive color screen, a printer and a transceiver for contactless smart cards.

Applications in the card

The new card offers customers a wide variety of applications. The following descriptions relate to the fully equipped version of the card. Naturally, there are also simpler versions, such as cards having only the functions of the contactless memory chip. With such a card, travelers can check themselves in at a service kiosk. This naturally leads to higher throughput, with shorter waiting times and/or faster processing. Lufthansa's bonus system is also integrated into the contactless-chip version of the card, making it unnecessary to use any other card or enter additional information.

Using suitable automated equipment located in the air terminal, a traveler can check in and receive a printed 'boarding information' form. This form contains all of the essential information regarding the booked flight, similar to what is on the actual ticket. If necessary, a seat selection for the booked flight can be made at the kiosk. Flights can also be booked by telephone using the number embossed on the card. In the future, Lufthansa plans to manage all of this without paper tickets, since all relevant information can be retrieved from the background system via the card. Naturally, this does not rule out making flight reservations by telephone or fax, which will continue to be possible.

In addition to these functions, the card can also serve as a credit card by incorporating a hologram and a magnetic stripe with the necessary data. It is also possible to embed a contact-type chip in the ISO location. This is currently used to implant a phone-card chip.

The overall system

All of the applications based on the contactless chip are structured very simply with respect to the smart card. The card is used only for identification, with the memory chip being used to allow dynamic authentication by the background system.[2] After authentication, all of the information stored in the card is read out. Currently, this consists of the customer number, customer name and customer profile. With this information, the background system can match the card to a particular person, after which the booking data, bonus system points and all other functions are available. The card is thus used only as a kind of key, with the data and mechanisms belonging to the various applications remaining in the background system. This

[2] See also Section 4.11.1, 'Symmetric unilateral authentication'

has considerable advantages in this case, since the background system and all of its required databases, programs and interfaces are already well established.

Another advantage of this system is the way it deals with lost or defective cards. This is a difficult issue, which up to now has always been neglected in other systems using multi-application cards. If applications have been loaded into a card after personalization, when the cardholder receives a new card, he or she must individually contact all of the application providers in order to have them reload their applications into the card. Thanks to its centralized system architecture, the Lufthansa smart card system does not have this problem. If a card is lost or becomes defective, the customer receives a new card and the old card is blocked system-wide by means of its number. This does not require a large amount of logistical effort, since all airports served by Lufthansa have access to the necessary data via the well-established Lufthansa network.

The contactless card

The smart card has the internationally standard ID-1 format. The memory chip embedded in the card body uses inductive coupling, with a single coil for both power and data transfers. With this technology, the terminal can both read and write data at a distance of up to 10 cm.[3] If the card is in an ordinary purse, it is even possible to exchange data with the terminal without removing the card from the purse. It is only necessary to hold the purse next to the terminal. Since the chip and the coil are both located inside the card, the graphic layout need not be affected by these components. Furthermore, contactless technology eliminates the problem of contact wear, since there simply aren't any contacts.

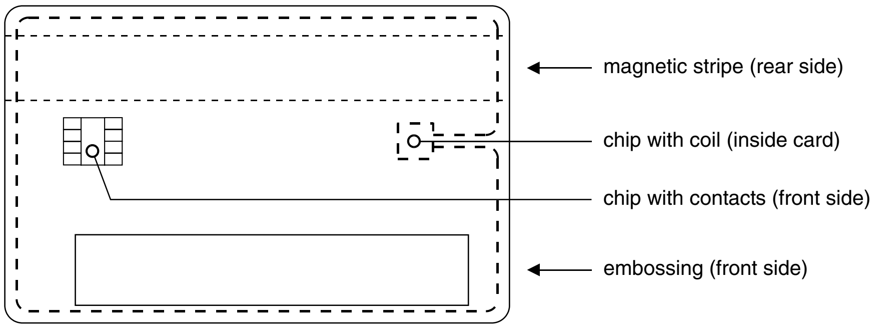

magnetic stripe (rear side)
chip with coil (inside card)
chip with contacts (front side)
embossing (front side)

Figure 14.1 Physical layout of the Lufthansa smart card in the full-up version

The typical transaction time between the terminal and the card in this system is around 100 ms, and a clock frequency of 13.56 MHz is used. The memory chip used (SLE 44R35) contains 1 kB of EEPROM and can be unilaterally authenticated by the outside world. Since presently only 48 bytes of data are stored in the memory (the customer number, customer name and customer profile), other applications could be incorporated in the future – although in this system, new applications naturally do not require additional memory in the smart card.

[3] See also Section 3.6, 'Contactless Cards'

Summary

The field trial on the Frankfurt–Berlin route was very successful. Nevertheless, implementation of the system in this form was not further pursued.

Although this smart card application is atypical with regard to system architecture, it brings an interesting new perspective to the smart card world. Since the application data are located in the background system, all aspects related to protection of personal data are also shifted to this system. This means that data privacy legislation, regardless of which country is involved, cannot have any influence on the data stored in the card. This approach also circumvents the well-known memory space problems that arise when several applications are located in a single card, since the issue of which user is allowed to write which data to the card simply does not exist. The separation of applications on the card is also complete, since the system design avoids any possible interference between individual applications. Finally, it can be remarked that the data that are valuable to the system operator are always securely stored in his background system, rather than in smart cards where they can potentially become lost.

This new system is also an exemplary illustration of a seamless transition from one card technology to the following one. This sort of 'soft' migration from one stage of technology to the next one is a very important consideration, since it preserves previous investments and avoids the need to construct an entirely new system.

14.2 HEALTH INSURANCE CARDS

In Germany, each member of a public health insurance plan was issued a health insurance card (*Krankenversichertenkarte* or KVK) by the end of 1994. In 1996, the private health insurance plans also began to issue their own smart cards, which are compatible with the public KVKs. These cards, of which more than 72 million have been issued, have thus achieved a level of penetration in the whole of the German population that exceeds that of phone cards.

Originally, it was only planned to introduce a magnetic-stripe card, but in consideration of possible future developments it was decided to use smart cards instead. The least costly solution was to use memory cards, since at that time microcontroller cards were much more expensive. However, the system is designed such that memory cards can be replaced by 'real' smart cards in further development stages over the course of several years. The result is a nationwide system that can form the basis for a future smart card health-care system, should this be necessary.

The health insurance member card has two basic functions for the insured person. Its first function is to identify the person to the doctor who treats that person. It thus replaces the paper health insurance card. Its second function is to act as a machine-readable data storage medium for the computer in the doctor's clinic. Usually, the terminal is connected to a PC in the clinic, which also controls the terminal. The card can be read using the terminal, and the billing data obtained in this manner can be further processed automatically. If the doctor manages his or her practice using traditional methods (that is, without a computer), the terminal can directly transfer the data from the card to a printer and thereby to a printed form.

Three different entities can access the health insurance card. The first is the doctor's clinic, where data can only be read from the card. Here there is no intention of allowing data to be written to the card, and the terminal software prevents such access. The second entity is the health insurance organization, which again can only read the data in the card. Here the insured person can read and check the personal information stored in the card. The health insurance organization also has special terminals that allow data to be written to the card. This can for example be necessary if the insured person moves to a new address. However, many insurers simply issue a new card to the insured person instead of modifying the data in the existing card, and request the cardholder to destroy the old one. This is logistically significantly simpler and thus less costly.

In the initial phase of the KVK project, consideration was given to storing a wide variety of patient information in the card. Some people wanted to include all possible information in the card, ranging from the blood type to allergies, so it would be a sort of emergency card. However, after all the objections related to the protection of personal data had been resolved, only the personal information listed in Table 14.1 was left to be stored in the chip in the card. The information contained in the card is essentially also present on the outside of the card, so the insured person knows his own information – although only to the extent that it is fixed and person-specific. The address is held only in the memory of the card, so that in principle it is not necessary to generate a new card if the insured person changes addresses.

Table 14.1 Data elements and TLV coding of German health insurance cards as defined in the technical specifications for German health insurance cards (1993)

Data element	Length	Tag
Checksum for the entire template (XOR)	1 byte	'8E'
Date of birth of the insured person (DDMMYYYY)	8 bytes	'88'
Expiry date of the card (MMYY)	4 bytes	'8D'
Federal state	1–3 bytes	'8A'
Given name of the insured person	1–28 bytes	'85'
Insurance card number (VKNR)	5 bytes	'8F'
Legal system (east/west)	1 byte	'90'
Name extensions of the insured person	1–15 bytes	'86'
Name of the city or town	2–22 bytes	'8C'
Name of the insurer	2–28 bytes	'80'
Number of the insured person	6–12 bytes	'82'
Number of the insurer	7 bytes	'81'
Postal code	4–7 bytes	'8B'
Status of the insured person	4 bytes	'83'
Street name and house number	2–28 bytes	'89'
Surname of the insured person	2–28 bytes	'87'
Template for data regarding the insured person	70–212 bytes	'60'
Title of the insured person	2–15 bytes	'84'

The requirement that the information in the card be generally known was also one of the prerequisites for the approval of the overall system. No information that is secret or not known to the insured person is allowed to be present in the card. It must also not be possible to write additional data to the card at a later date without authorization. To exclude the possibility of writing data to the card, neither doctors nor insurance organizations receive terminals that have this capability. Only a few administrative terminals located in the insurance organizations can write data to the cards. However, no special authentication key is needed for this, so data could easily be written using any suitably equipped terminal.

From a purely external perspective, a health insurance card behaves as though it contains only a single transparent file. Data can be freely read from this file using offset and length parameters. Certain administrative terminals can also write data to the memory, but for reasons of personal data privacy this only occurs as an exception.

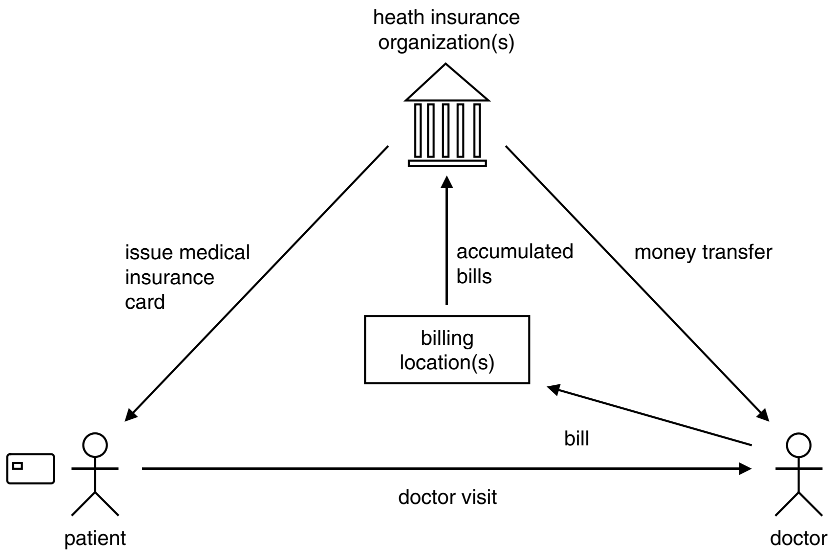


Figure 14.2 Basic architecture of the German health insurance card system

When the arrangement of the data elements was specified, it was very important for it to be possible to make future extensions or modifications without creating any compatibility problems. Consequently, all personal data in the health insurance card are structured using the ANS.1 data description language. They are stored in the card's memory in a TLV structure. This makes it possible to add other data objects in the future or change the codes used for existing data objects. The tags to be used are prescribed by a specification, so the data elements of all health insurance cards are structured in the same manner.

The health insurance card is not a microprocessor card. It is a memory card, with hardware similar to what has been used for years in phone cards. The EEPROM that is used must have a capacity of at least 256 bytes. This is equal to the amount of all necessary data located in the health insurance card. If the EEPROM is exactly this large, the necessary data just fit and it is physically impossible to write any additional data to the card in violation of data privacy legislation.

The clock-synchronous data transmission protocols depend on the specific type of chip that is used. Each terminal must therefore be able to fully process all possible protocols. The card body can be manufactured using injection molding or a multilayer technology. The useful life of the health insurance card is specified to be six years. After this time, the insured person automatically receives a new card. This means that around 15 million new cards must be issued each year.

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0y	'A2'	'13'	'10'	'91'	'46'	'0B'	'81'	'15'	'44'	'45'	'47'	'2B'	'44'	'32'	'88'	'F1'
								ATR and manufacturing data								
1y	'9F'	'61'	'0B'	'4F'	'06'	'D2'	'76'	'00'	'00'	'01'	'01'	'53'	'01'	'07'	'60'	'81'
				T	L				AID			T	L	V1	T	L1
2y	'88'	'80'	'18'	'56'	'65'	'72'	'65'	'69'	'6E'	'74'	'65'	'20'	'4B'	'56'	'20'	'20'
	L2	T	L	"V"	"e"	"r"	"e"	"i"	"n"	"t"	"e"	" "	"K"	"V"	" "	" "
3y	'20'	'20'	'20'	'20'	'34'	'30'	'33'	'33'	'35'	'33'	'31'	'81'	'07'	'30'	'30'	'30'
	" "	" "	" "	" "	"4"	"0"	"3"	"3"	"5"	"3"	"1"	T	L	"0"	"0"	"0"
4y	'30'	'30'	'30'	'30'	'8F'	'05'	'30'	'30'	'30'	'30'	'30'	'82'	'07'	'38'	'31'	'36'
	"0"	"0"	"0"	"0"	T	L	"0"	"0"	"0"	"0"	"0"	T	L	"8"	"1"	"6"
5y	'34'	'35'	'30'	'31'	'83'	'04'	'30'	'30'	'30'	'32'	'90'	'01'	'31'	'85'	'09'	'41'
	"4"	"5"	"0"	"1"	T	L	"0"	"0"	"0"	"2"	T	L	"1"	T	L	"A"
6y	'6C'	'65'	'78'	'61'	'6E'	'64'	'65'	'72'	'87'	'05'	'52'	'61'	'6E'	'6B'	'6C'	'88'
	"l"	"e"	"x"	"a"	"n"	"d"	"e"	"r"	T	L	"R"	"a"	"n"	"k"	"l"	T
7y	'08'	'30'	'31'	'30'	'31'	'31'	'39'	'36'	'35'	'89'	'15'	'50'	'72'	'69'	'6E'	'7A'
	L	"0"	"1"	"0"	"1"	"1"	"9"	"6"	"5"	T	L	"P"	"r"	"i"	"n"	"z"
8y	'72'	'65'	'67'	'65'	'6E'	'74'	'65'	'6E'	'73'	'74'	'72'	'2E'	'20'	'32'	'30'	'30'
	"r"	"e"	"g"	"e"	"n"	"t"	"e"	"n"	"s"	"t"	"r"	"."	" "	"2"	"0"	"0"
9y	'8B'	'05'	'38'	'30'	'30'	'30'	'30'	'8C'	'07'	'4D'	'7D'	'6E'	'63'	'68'	'65'	'6E'
	T	L	"8"	"0"	"0"	"0"	"0"	T	L	"M"	"ü"	"n"	"c"	"h"	"e"	"n"

Figure 14.3 Sample KVK data set for a privately insured person, with all data elements decoded. The data shown here correspond to a non-existent person, and the XOR checksum is intentionally incorrect. The following abbreviations are used: T: tag; L & L2: length of the following data; L1: tag within the length code indicating that the next byte (L2) is the actual length code; NU: not used; V1: personalizer tag

Ay	'8D'	'04'	'30'	'31'	'30'	'37'	'8E'	'01'	'42'	'C0'	'54'	'20'	'20'	'20'	'20'	'20'
	T	L	"0"	"1"	"0"	"7"	T	L	XOR	T	L					
By	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'
Cy	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'
	unused memory region (filled with space characters)															
Dy	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'
Ey	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'
Fy	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'20'	'00'
																NU

Figure 14.3 (*Cont.*)

If the terminal is connected to a computer, it is controlled by the computer using the T = 1 transmission protocol as specified in ISO/IEC 7816-3. There is one restriction in this regard, which is that data chaining may not be used in the protocol. Doing so would not add any functionality to the application, but it would increase the amount of memory needed in the terminal. Incidentally, this is a typical example of the fact that real applications often use only the necessary parts of standards, and it is uncommon for all of the functions specified in a standard to actually be implemented.

There are only three possible commands that the terminal can execute. The first command is a reset to the health insurance card, followed by reception or reading of the ATR. This command is always used at the start of a session to activate the health insurance card. The second command is READ BINARY with the ISO coding, which can be used to read selected portions of the data or all of the data via the terminal. The third command is WRITE BINARY, also in accordance with ISO/IEC 7816-4, although this command is only available in administrative terminals. It is blocked in all other types of terminals. The health insurance cards of the private insurers have write protection implemented using PINs that are known only to the insurance organization. The cards for the public health insurance plans can be freely written if the necessary commands are known.

If the terminal is directly connected to a computer, its essential function is only to provide a conversion between the T = 1 protocol and the hardware-dependent synchronous protocol of the health insurance card. Nevertheless, it can be clearly seen that it would be possible to switch to microprocessor cards without undue effort or expense. With microprocessor cards, the only function of the terminal would be to transparently relay the commands received from the clinic computer. The response from the card could also be transparently returned to the control computer, without any processing by the terminal.

14.3 ELECTRONIC TOLL SYSTEMS

In some countries, it is common to charge a toll for using certain roads. In contrast to flat-rate fees collected by selling windscreen stickers, tolls are collected according to usage. In other words, the amount collected depends on the frequency of use and the type of vehicle. Up to now, tolls have usually been paid in cash at tollbooths. In a few isolated cases, electronic systems using various types of cards have been used over the past few years in toll systems (road-pricing systems). However, all systems used up to now have the drawback that they significantly impede the flow of traffic, since vehicles must either stop or slow to a walking pace. The toll stations also require a large amount of space.

In light of this, the German traffic ministry decided in 1993 to start a large-scale field trial of automatic toll collection systems. The road selected for this test was the A555 *Autobahn* between Cologne and Bonn. Various systems provided by 10 different firms were tested on this stretch of road between May 1994 and June 1995.

The systems that were tested were evaluated in terms of several basic requirements. First, traffic should be able to flow normally and unimpaired past the toll collection points. In Germany ‘normal traffic flow’ can mean anything up to 250 km/h. Therefore, it should not be possible to evade toll collection by driving very fast. Furthermore, tollbooths or collection baskets were not allowed be used for paying tolls, since the equipment of the desired system had to be suitable for mounting on flyovers and sign bridges. In addition, the systems had to support both single-lane and multi-lane traffic. It is technically undesirable to channel traffic into individual lanes, since this strongly restricts traffic flow.

A supplementary requirement, which was not originally foreseen in this form, arose during the course of the project. This was complete anonymity with respect to toll collection, and this increasingly came to be regarded by the general public as the crucial factor for the entire trial. It should not be possible to generate vehicle movement profiles or monitor the routes traveled by specific vehicles. All proposed systems had a payment option that maintained vehicle anonymity as long as the toll was paid. As soon as a toll collection failed, the vehicle was photographed. The registered owner could then be tracked down and sent a suitable penalty notice. Naturally, this did not actually occur during the trial, since only ‘play money’ was used.

Almost all of the proposed automated toll collection systems used smart cards to hold the electronic currency. That is why we address this subject here, since it may well become important to the smart card industry in the future.

All tested systems have a device mounted in the vehicle called either the OBU (onboard unit) or IVU (in-vehicle unit), as well as additional equipment as necessary. This equipment is powered by the car battery. Inside the passenger compartment, each system has a smart card terminal, a display and a simple keypad. There is also a link to the outside world, which may be unidirectional or bidirectional, depending on the system. This is usually a microwave link in the 5.795–5.805 GHz frequency band, as recommended by CEPT for this application. Alternatively, some systems used radio signals in the 400–500 MHz range or infrared light. The disadvantage of using infrared light is naturally that transmission is strongly affected by weather conditions, such as heavy snow or fog. In the systems that participated in the trial, the stated mass-production price of the OBU was between 50 and 150 euros, depending on the configuration.

The control stations were installed along the motorway on flyovers and sign bridges as necessary. No modifications to the roadway construction were necessary. The smart cards did not contain sophisticated electronic purses, but only very simple and fast debiting commands. This was in part due to the fact that no real money was used, and in part due to the short time available for transactions. In some systems, the optimization process went so far as to reduce the ATR to four bytes, in order to leave sufficient time for the actual debiting. This is understandable in light of the requirement for unrestricted traffic flow. At 250 km/h, a vehicle covers 70 m per second. The control station has a communications range of approximately 5 m in the 5.8-GHz frequency band. This yields a dwell time (the interval during which the vehicle is within range of the station) of around 70 ms. The card must perform the following processes within this interval:

- card reset and ATR transmission (≈10 ms)
- DES encryption to authenticate the card (≈12 ms)
- EEPROM write access to store the new balance (≈2 × 3.5 ms)
- data transmission from and to the card (≈30 ms)

Additional time is needed for data transmission between the OBU and the electronic toll station. You can calculate that the amount of time available is very tight. The smart cards were all operated at the maximum allowable clock frequency (usually 5 MHz). The data transmission rate between the OBU and the toll station, at 1 Mbit/s, was substantially higher than that between the OBU and the card. It thus has little influence on the total transmission time.

Here we describe three representative systems chosen from the 10 systems involved in the field trial. Since some of the systems tested were nearly identical except for technical details, none of the systems described here is identified by name. The described systems can thus be considered to be representative of the types of systems that were tested.

System 1

The first system has a classic infrastructure design. The necessary equipment is installed on two successive flyovers or sign bridges. The system is naturally designed so that toll collection is not affected if vehicles change lanes between the two stations. There are two different payment modes. In the postpaid mode, accumulated toll fees are collected from a bank account, as with a credit card. This makes it very difficult to preserve vehicle anonymity. In the prepaid mode, prepaid smart cards are used, and the appropriate amount is deducted from the balance in the card each time a toll is collected.

Technically, the system works as follows. At the first station, the OBU and smart card are activated. Following this, the toll is levied and a general assessment of the vehicle is made in order to classify it as a car or lorry. Since the amount of the toll depends on the vehicle class, it is necessary to verify the class recorded in the card. This is done by measuring the vehicle's height profile when it passes under the station, which is sufficient for reliable classification. When the vehicle is within the communications range of the subsequent second station, another link to the card is established via the OBU. The second electronic toll station checks whether the toll payment initiated by the first station has been successfully completed. If this is not the case, the vehicle is photographed. Nowadays, a vehicle can be uniquely identified from its number plate using a completely automatic process. The registered owner of the vehicle can then be sent an appropriate fine.

The system can be further extended as desired with various other functions. For example, the card can be checked against a blacklist when a payment is made, in order to eliminate the use of stolen cards. Another thing that was discussed was whether the cameras should be able to photograph all passing vehicles in certain special cases, such as after bank robbery with escape by car, in order to obtain information about escape routes.

The advantage of this system is that the OBUs are simple in construction and thus inexpensive. However, this means that the electronic toll stations must be correspondingly more complicated and expensive.

System 2

The second system is not based on a roadway infrastructure. Instead, it uses virtual toll stations, which exist only as databases in the vehicle OBUs. The OBUs contain the coordinates of all available toll stations, along with toll rates for the corresponding road segments. As soon as a vehicle enters a defined road segment, the amount of the toll is deducted from the smart card. Naturally, this process is much less time-critical than the process used in the first system.

The OBU knows the vehicle's coordinates at any point in time by means of an attached Global Positioning System (GPS) module. GPS is a worldwide system for determining positions that was originally developed in the USA for military purposes under Department of Defense contracts. This started in 1973, and the system was operationally complete in 1993. The GPS consists of 24 satellites that transmit encoded radio signals from an altitude of 20,000 km. Each signal contains the time of transmission, the satellite position and a satellite identifier. A suitable receiver, which is only as large as a pack of cigarettes, can receive the satellite signals in the 1.6-GHz band and use them to determine its geographic position. The accuracy for civilian applications is around 10–20 m. This can be reduced to a few meters with an improved technique using differential measurements (differential GPS), which relies on an additional signal broadcast by a terrestrial transmitter. Presently, this signal can be received only in Central Europe. For military use, positions can be located with an accuracy of around one meter throughout the world.

Of course, the second toll system also requires monitoring, but this is done by means of random samples, similar to current speed controls. The main advantages of this system are that it is not necessary to install any equipment along the roadway and that the tolls are levied at virtual locations. However, control is more difficult, and the smart cards cannot be used in the postpaid mode, since no information is exchange between the OBU and the outside world. On the other hand, this provides an advantage in terms of anonymity.

System 3

The third system represents the most ambitious technical solution to automatic vehicle toll collection. The OBU is equipped with a GPS receiver and a GSM mobile telephone. The position of the vehicle is determined using the GPS unit, and mobile telephone is used to obtain specific pricing data. As with the other two systems, the OBU contains a terminal with a smart card that is used for toll collection.

As in System 2, the toll stations are only virtual locations along the road that are defined by geographic coordinates. In this system, the road operator can also modify the rates at will using bidirectional communications with the OBU. The toll can thus be made to depend on the time of day, location, vehicle class and environmental considerations. The amount to be paid is computed by the OBU in the vehicle.

In order to ensure a certain level of anonymity, the tolls deducted from the smart card in the OBU are not immediately forwarded to the background system. Instead, they are accumulated in the card until a certain level is reached. The accumulated amount is then transmitted by mobile telephone for verification and billing. Thanks to the bidirectional data exchange, this system can naturally support both debit (prepaid) cards and credit (postpaid) cards. However, anonymity cannot be maintained if the tolls are paid after the fact. With prepaid smart cards, which are used like electronic purses, the fact that only cumulative tolls are paid means that the motorway operator cannot generate vehicle movement profiles. However, it is naturally possible for the GSM network operator to continuously determine the location of a moving vehicle by means of its permanently activated mobile telephone.

In this system, as in the others, continuous monitoring of moving traffic is necessary to check for drivers who try to avoid paying tolls. This is done using automatic cameras mounted on flyovers and mobile checkpoints. The associated monitoring computers can read vehicle registration numbers from OBUs or smart cards via the mobile telephones and compare them with vehicle number plates. If they do not match, or if no link to the OBU can be established, the vehicle is photographed and a penalty procedure is initiated.

The main advantage of this system is that no specific infrastructure has to be constructed, apart from that needed for monitoring. However, this means that the OBU is distinctly more expensive than in the other two systems. Two other advantages of this system are that it supports two different payment methods (prepaid and postpaid) and that data can be transmitted to the vehicle.

14.4 DIGITAL SIGNATURES

There are two fundamental prerequisites for the use of legally binding digital signatures. The first is a microcontroller smart card with a powerful numeric coprocessor, and the second is a clearly defined general legislative framework. The smart card is used to securely store the secret signature key and generate the digital signature. The legislation establishes the binding conditions that apply to all parties involved in this smart card application.

Of course, digital signatures can also be used in the form of a closed application that is completely independent of any legal context. However, in this case suitable contractual arrangements between the participating parties are necessary to make a digital signature binding on the person who generates the signature. In the business-to-business sector, for example, the use of digital signatures has been standard practice for several years. However, if the objective is an open system that can be used by parties that are not known to each other, suitable legal conditions are required. This legislative framework allows digital signatures to be made equivalent to normal signatures and makes it possible to bring suit in court with regard to a document so signed, just as with a handwritten signature.

The potential applications for legally recognized digital signatures, which are thus equivalent to handwritten signatures, are practically unlimited. In the simplest case, they can be used to sign electronic letters and orders, but they can also be used in an electronic payment system to sign direct debit authorizations or in a home banking system to sign remittance orders. The most important application for digital signatures will doubtless be signing all types of contractual agreements.

General conditions specified in standards

In order for digital signatures to be utilized interoperably in an open commercial environment, they must meet the relevant security requirements of various standards. As far as signature cards are concerned, the applicable standards are the ISO/IEC 7816-4 standard for general smart card commands and the 7816-8 standard for signature commands. With regard to card bodies, electrical properties and data transmission, the relevant standards of the ISO/IEC 7816 family must be taken into consideration. The authentication process between the smart card and the rest of the world is covered by ISO/IEC 9796-2. Basic mechanisms and methods for digital signatures are handled by ISO/IEC 14888. In addition, the structure and coding of the certificate normally comply with the X.509 standard.

A digital signature card can be used to generate a signature that is equivalent to a real signature. This means that large monetary or material values can be involved, depending on circumstances and the document bearing the signature. The system elements relating to security must therefore be evaluated in the context of a digital signature application. The proven means for this is an evaluation using the ITSEC or Common Criteria (CC) criteria catalog.

General statutory conditions in Germany

In order for digital signatures to be recognized as signatures that are generally valid, binding and subject to suit, there is no alternative to providing the necessary general statutory conditions. The state of Utah in the USA was the first to introduce legislation in this area, in 1995. This legislation has served as a model for legislation in many other countries, including German signature legislation.

The essential requirements of the German *Signaturgesetz* (Digital Signature Act) are the following:

- Operating a certification authority (a trust center) is subject to approval by the responsible public authority.
- Any person who requests a certificate must be reliably identified by the certification authority.
- The certificate for the signature key may bear a pseudonym of the key holder, in place of the actual name of the key holder.
- The actual name behind a pseudonym must be revealed in response to a formal request by certain public agencies.
- The certification authority must make the certificate for the public signature key available in a manner that allows it to be verified online.

- A certificate can be blocked under certain conditions, such as if this is requested by the holder of the signature key.
- The technical components for generating and verifying digital signatures must clearly and unambiguously represent the data referenced by the digital signature.
- Digital signatures from other countries can also be recognized in Germany by means of suitable agreements.

According to the German Digital Signature Act, a signature key certificate, which usually complies with the X.509 standard, must contain the following information:

- the name or pseudonym of the signature key holder
- the public signature key assigned to the signature key holder
- the designations of the hash algorithm and signature algorithm used for the certificate
- the serial number of the certificate
- the initial and final validity dates for the certificate
- the name of the certification agency
- statements as to whether the signature key can only be used for particular applications.

On June 13, 1997, the *Gesetz zur Regelung der Rahmenbedingungen für Informations- und Kommunikationsdienste* (IuKDG) was adopted by the German federal legislature (*Bundestag*). It came into force on August 1, 1997, and a revised version was published on 22 May, 2001. Article 3 of this act is the Digital Signature Act, which is called the *Signaturgesetz* (SigG) for short. This article is divided into 16 sections. The first paragraph of the first section states the objective of the Signature Act. It reads, 'The objective of the Act is to establish general conditions for digital signatures, under which they can be considered to be secure and which allow forgeries of digital signatures or falsifications of the signed data to be reliably detected.' This statement clearly indicates that no particular technical solution is specified, but only general conditions with regard to the use of digital signatures. As a natural consequence, the legal requirements are described at a relatively abstract level. The regulation and description of digital signature applications cannot be handled by legislation alone. A hierarchy of requirements is necessary for this, with the legal regulation – the Digital Signature Act – standing at the top of the hierarchy. This is followed by the associated implementation rules, which in Germany are called the Digital Signature Ordinance (*Signaturverordnung,* or SigV). On the next lower level, which already describes the requirements for digital signatures in technically very concrete terms, we find the measures catalogs. In Germany, these are issued by the regulatory agencies for the telecommunications and postal services. These three levels are obligatory for all applications of digital signatures in Germany, to the extent that the digital signatures are to be equivalent to regular signatures.

The level below the measures is the technical specifications. These define concrete solutions in unambiguous and non-interpretable form. The specifications depend on the application operator in question, and they must be aligned to the three higher levels of the hierarchy. However, if an application operator does not require the digital signature to have a legally binding character, he can naturally set up his system in whatever manner best meets his objectives. This is referred to as a legally non-compliant solution, as opposed to a legally compliant solution.

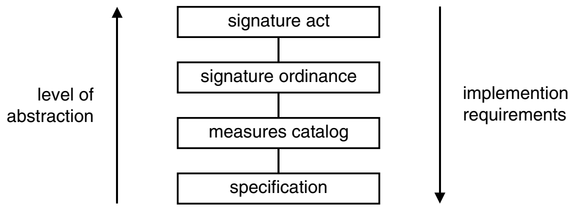


Figure 14.4 The hierarchic structure of the requirements for a digital signature application in Germany. The system of requirements is fashioned in a top-down manner, which means that the lower-level documents are based on those that are higher up in the list

The German Digital Signature Ordinance (SigV), which was adopted by the federal government on October 8, 1997 and came into force on November 1, 1997, contains the implementation regulations for the Digital Signature Act (SigG). For each of the 16 sections of the Signature Act, it contains detailed descriptions that are worded significantly less abstractly than the law. The following statements are contained in the SigV, among others:

- A person who requests a certificate must be identified by means of a personal identity document, passport or other suitable means.
- The data storage medium (usually a smart card with a signature key) must be held in personal safekeeping, and the associated identification data (e.g., a PIN) must be kept secret.
- When verifying a digital signature, it is necessary to also verify that the signature key certificate was valid when the signature was generated, and a check must be made to see if there are any restrictions on the use of the signature.
- A certificate cannot be valid for more than five years.
- Unblocking a blocked certificate is not allowed.
- A measures catalog with suitable security measures must be generated.
- The certification authority (trust center) must be audited by the responsible public authority.
- The signature keys must be generated in a manner such that the probability of two keys being the same verges on zero.
- Signature keys may not be duplicated.
- All security modifications to technical components must be recognizable to the user in question.
- The signature key may be used only after its holder has been identified by means of possession and knowledge.
- The actual signature key is not allowed to be revealed when it is used.

- A list of suitable algorithms for generating signature keys, computing hash values of signed data and generating and verifying digital signatures will be published in the *Bundesanzeiger* (Federal Gazette). The time interval during which they may be used will also be stated.
- If signed data are to remain valid for longer than the term of validity of the approved algorithm, the data must be signed anew, with a time stamp, before the expiry date of the algorithm.
- The essential components of a legally compliant signature application must be tested against the ITSEC criteria using the mechanism strength 'high'. The testing level depends on the individual component and how it is used. Table 14.2 lists the components and associated test levels.

Table 14.2 Comparison of the essential differences between the ITSEC E2 and E4 test levels with respect to signature cards

ITSEC test level E2 (high)	ITSEC test level E4 (high)
• components for acquiring and testing identification data, and for displaying the data to be signed	• components for generating keys
• components for displaying signed data and testing certificates	• components for storing and using signature keys (usually smart cards)
• components for generating time stamps	• components for generating and testing digital signatures that are commercially offered to third parties for their own use

The measures catalogs for the Signature Act and Signature Ordinance specify the technical requirements for this system. They are distinctly more concrete than the Signature Act. The technical specifications are in turn based on the measures catalogs. They specify the technical implementation in detail. In Germany, the most important technical specification for digital signatures is the DIN specification entitled *Spezifikation der Schnittstelle zu Chipkarten mit digitaler Signatur-Anwendung/Funktion nach SigG und SigV* ('Specification of the Interface to Smart Cards with Digital Signature Applications / Functions according to the SigG and the SigV'). The following subjects related to signature cards are dealt with in the DIN specification:

- answer to reset (ATR) and protocol parameter selection (PPS)
- data transmission protocols (T = 0 and T = 1)
- files, data objects and data formats
- authentication of the cardholder
- computation and verification of signatures
- logging the signature generation process
- procedures for generating and verifying signatures
- cooperation with terminals and the associated command sequences.

System architecture

The two essential components of a system for digital signatures are the trust center and the signature card. These two entities thus command most of our attention, since all of the processes related to security take place in them. In this regard, in a system for digital signatures we can make a fundamental distinction between issuing a card and the processes of signing and verifying signatures.

Issuing a digital signature card

The future user of a digital signature card must first register at a registration authority. The person to be registered must appear in person and present a recognized proof of identification, such as a personal identification card. At this point, the future user of the digital signature card can elect to have a pseudonym appear in place of his or her actual name. The registration authority transfers the information it has received and verified to the trust center, which initiates the generation of a key and the personalization of a signature card. These activities can take place in the trust center, but they can also be carried out by a third party or directly within the registration authority. In the process, the public key of the new smart card is signed by the certification service of the trust center, which makes it a recognized authentic public key. The signed public key is then entered into the public key directory of the trust center as a valid key, which makes it available for use by every system subscriber. After all this is finished, the new subscriber receives his or her digital signature card along with a PIN letter. This completes the issuing of the card, which can now be used.

Signing and verifying documents

When an electronic document is signed, it is compressed to a hash value and then signed using the digital signature card with the private key. This process can be initiated only after the cardholder (the signer) has unambiguously been identified, which currently means after the cardholder's PIN code has been entered. In the future, a test based on a biometric feature (such as a fingerprint) could be used for identification. The system components that are used for signing the data are the signature card, a terminal, and usually a PC.

The digitally signed document can now be sent in any desired manner. In order to verify the document, it is necessary to again compute the hash value. If necessary, the public key of the signer can be retrieved from the public key directory of the trust center. With this key, it is possible to verify whether the digital signature of the electronic document is genuine. In order to be sure that the private key of the signer has not been compromised, the revocation list (blacklist for keys) of the trust center must be consulted. The signature verifier can thereby be assured that the private key of the signer has not been blocked.

If the verifier needs to have confirmation of consulting the revocation list in a form that can be verified by a third party, he can obtain a suitable confirmation with reference to the verification data via the time stamp service of the trust center.

The system architecture and procedures for issuing digital signature cards and signing and verifying document with a digital signatures that are shown here are examples, which may differ from actual implementations. However, they provide a realistic if simplified overview of a typical digital signature system.

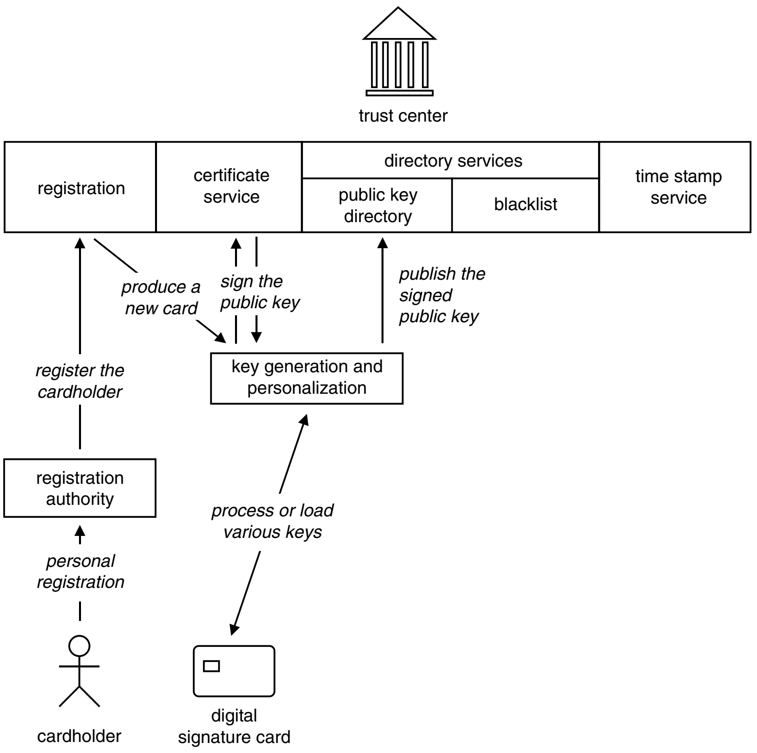


Figure 14.5 The basic procedure for issuing a digital signature card

The trust center (TC)

The trust center is the most important component of a digital signature system, aside from the signature cards. It supports six different functions: registering new users (registry); generating keys and personalizing new digital signature cards; a certification service; a directory service for public keys; a directory service for blacklists (revocation lists); and a time stamp service.

The registry service collects personal information and verifies the identity of a new user of a digital signature card. A registration agency normally acts as an intermediary between the trust center and a new user, so that the user does not have to personally appear at the trust center. It is only necessary to generate a key and personalize the signature card when a new card is issued. This does not necessarily have to take place within the trust center; it can for example be done by the card manufacturer. The key can be generated either internal to or external from the card. The certification service of a trust center signs public keys for digital signature cards using a private key belonging to the trust center, so that they can be recognized as being genuine. Naturally, a trust center can have more than one private key for generating certificates.

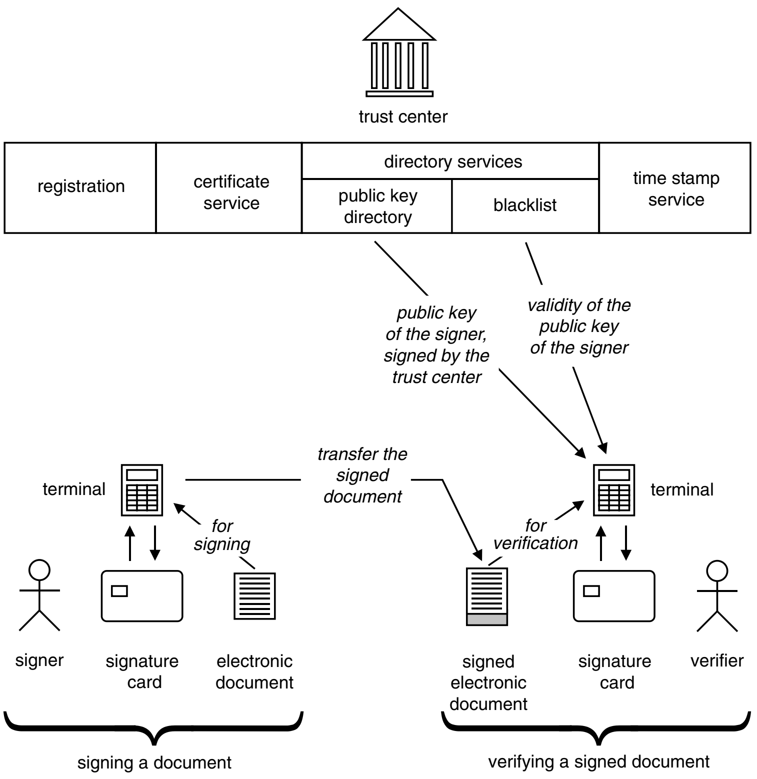


Figure 14.6 Basic procedures for signing and verifying a document using a digital signature

The directory service for public keys contains the public keys for the digital signature cards signed by the trust center. The directory service for the revocation list contains a list of all blocked keys, which are keys belonging to digital signature cards that may have been lost or compromised. According to the German Signature Act, these directory services must be accessible via generally available public networks.

The time stamp service need not necessarily be integrated into a trust center, but it is certainly available in most trust center implementations. This service is used to attach the current date and time to electronic information presented to the trust center. The information received by the trust center, together with the date and time, are signed using the private key of the time stamp service. This allows the supplier of the information to prove to a third party that the information signed by the time stamp service must have been available no later than the indicated time.

The digital signature card

A secure hardware environment is necessary for storing and using private signature keys. Smart cards represent an ideal solution to this requirement, since they are physically small, inexpensive and offer a high level of protection against externally reading or modifying the stored data. It can be asserted without reservation that digital signatures only became possible with the advent of smart cards.

The cryptographic algorithms used for this purpose require a microcontroller with a supplementary numeric processing unit (NPU) to allow signatures to be generated and tested in an acceptable length of time. In theory, a signed document could also be tested outside of the signature card, since all of the necessary information is also available there. Whether this will become significant in practice remains to be seen. In terms of security, however, it is better to perform the test inside the card.

Public and private keys for digital signature cards can be generated in either a centralized or a decentralized manner. In any case, the keys must be generated in a cryptographically secured environment, such as inside a security module or smart card. It is also important for the public key to then be signed using the private key of the trust center. This is necessary to ensure that the private and public keys of the signature card are recognized as authentic by the trust center. Without this recognition, it is not possible for a third party to verify that the digital signature of an electronic document actually comes from a genuine subscriber of a digital signature system.

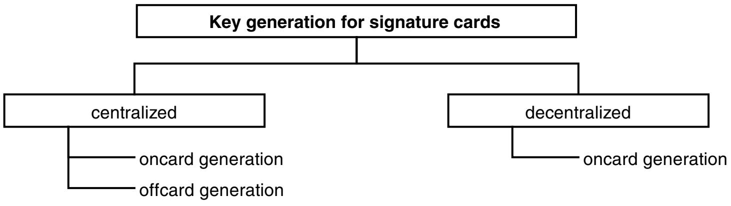


Figure 14.7 The possible options for generating keys for signature cards

With centralized key generation, all key pairs are generated in one place and signed using the private key of the trust center immediately after they are generated. The keys can be generated and then signed by the trust center either inside the card (*oncard*) or outside the card (*offcard*). Decentralized key generation can only be performed oncard, since the card is the only cryptographically secured environment in this situation. The advantages and disadvantages of the two methods are listed in Table 14.3. In practice, centralized key generation will probably become predominant, since it is very secure and also fits with the conventional manufacturing process for smart cards.

In Germany, the most important document for the signature card interface is the previously mentioned DIN specification. It extensively describes the necessary commands, files and processes for a digital signature card.

The signature application is located in a DF directly under the MF. The DIN specification provides nine EFs for this application, which contain all of the necessary data for the application. All data elements used are TLV encoded. The EFs and their contents are summarized in Table 14.4.

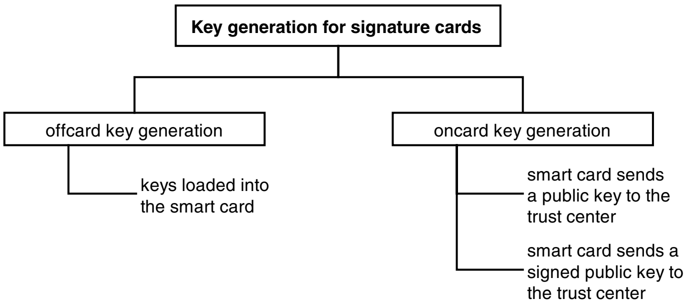

Key generation for signature cards
offcard key generation
oncard key generation
keys loaded into the smart card
smart card sends a public key to the trust center
smart card sends a signed public key to the trust center

Figure 14.8 The three basic options for generating keys and distributing keys to signature cards and trust centers

Table 14.3 Comparison of the three methods that can be used for generating keys. The method shown in the middle column is usually used in large real-life applications

Method	Offcard key generation	Oncard key generation, with the public key given to the trust center for signing	Oncard key generation, with the signed public key given to the trust center
Pro	• fast personalization • reproducing identical cards is possible	• the private signature key never leaves the smart card	• the private signature key never leaves the smart card
Neutral	• key escrow technically possible and easy	• key escrow difficult	• key escrow difficult
Con	• the private key is generated offcard	• the public key must be brought to the trust center via a secure route • reproducing identical cards is not possible	• the private key of the trust center is in the smart card • reproducing identical cards is not possible • personalization time-consuming

The following commands are used for signature cards:

APPEND RECORD	MANAGE SECURITY ENVIRONMENT
EXTERNAL AUTHENTICATE	PERFORM SECURITY OPERATION
GET CHALLENGE	READ BINARY
GET RESPONSE (optional for T = 0)	READ RECORD
INTERNAL AUTHENTICATE	SELECT FILE
MANAGE CHANNEL (optional)	UPDATE BINARY

The hash functions and signature algorithms that are approved for German signature cards are SHA-1, RIPEMD-160, RSA, DSA and ECC. Since it would take too long to compute the

Table 14.4 Simplified file tree of a signature card according to the German Signature Act, showing the most important files

File type	FID	Structure and size	Description
MF	'3F00'	—	Root directory
MF.EF$_{\mathbf{GDO}}$	'2F02'	transparent, *n* bytes	Serial number of the card and name of the card owner
DF$_{\mathbf{SigG}}$	not defined	—	Digital signature application DF (AID = 'D2 76 00 00 66 01')
DF$_{\mathbf{SigG}}$.EF$_{\mathbf{KEY}}$	—	undefined, *n* bytes	PINs; public and private keys
DF$_{\mathbf{SigG}}$.EF$_{\mathbf{SSD}}$	'1F00'	transparent, *n* bytes	Security service description (SSD) (information about the properties of the signature card)
DF$_{\mathbf{SigG}}$.EF$_{\mathbf{C.ICC.AUT}}$	'C000'	transparent, *n* bytes	Authentication certificate of the signature card
DF$_{\mathbf{SigG}}$.EF$_{\mathbf{C.CA.AUT}}$	'C008'	transparent, *n* bytes	Authentication certificate of the certification authority
DF$_{\mathbf{SigG}}$.EF$_{\mathbf{DM}}$	'D000'	transparent, 8 bytes	Display message (DM) for the terminal so the user can recognize it as genuine
DF$_{\mathbf{SigG}}$.EF$_{\mathbf{C.CH.DS}}$	'C100'	transparent, *n* bytes	Signature certificate of the card owner
DF$_{\mathbf{SigG}}$.EF$_{\mathbf{C.CA.DS}}$	'C108'	transparent, *n* bytes	Signature certificate of the certification authority
DF$_{\mathbf{SigG}}$.EF$_{\mathbf{PK.CA.DS}}$	'B000'	transparent, *n* bytes	Public key of the certification authority
DF$_{\mathbf{SigG}}$.EF$_{\mathbf{PROT}}$	'A000'	cyclic, *n* bytes	Signature generation log

hash value of a large electronic document inside the smart card, the hash value can be computed in a PC and then passed to the card for signing. Alternatively, the hash value can be computed in the PC up to the final block, following which the final block can be computed from the corresponding part of the document in the smart card and then signed. It is also possible to completely compute the hash value in the smart card, but this is frequently not practical, due to the well-known performance limitations of smart cards.

Summary and prospects

In Germany, the Digital Signature Act has created the legal framework necessary to allow digital signatures to be used reliably in everyday life. They will probably not be used extensively for things such as private house purchase contracts or wills, but instead for daily activities such as sending e-mail and ordering or paying for goods via public networks, which by definition are not secure.

It can safely be assumed that besides legally compliant solutions, there will also be many that are not legally compliant, since the implementation of a non-compliant solution is easier and much less costly. The requirements imposed on the security of the components used for digital signatures are set very high in Germany, which means that they do not favor quick and inexpensive solutions. However, this level of security is necessary if digital signatures are to be made equivalent to handwritten signatures. If it were possible to successfully manipulate digital signatures, the entire concept of digital signatures would lose its credibility, and in the worst case it would have to be abandoned.

Another important consideration in this regard is the behavior of multinational organizations, which are not inclined to pay significant attention to the legislation of any individual country. This means that it would certainly be possible for a solution that is not legally compliant to become established as an international quasi-standard.

Despite these risks, the German digital signature implementation has international influence, both on legislation and on the technical implementation of similar projects. If the potential of this system is utilized, Germany will find itself with a technically superior and exemplary solution to the implementation of digital signature systems.

14.5 THE PKCS #15 SIGNATURE APPLICATION

In 1998, RSA Inc. [RSA] presented a document titled ‘Cryptographic Token Information Format Standard’. This specification encompasses the description of files and associated data formats for a cryptographic token. Such a token is preferably a smart card with a cryptographic processor. ‘PKCS’ stands for ‘public key cryptography standards’, and the PKCS #15 specification draws on the Digital Certificates on Smart Cards (DC/SC) and Secured Electronic Information in Society (SEIS) specifications for digital signature applications using smart cards. Presently, the PKCS #15 specification is also covered in a compatible form by the ISO/IEC 7816-5 international smart card standard, but here we refer to the PKCS #15 specification, since it has become an internationally established reference.

Various requirements criteria were taken into consideration in generating the PKCS #15 specification. It was intended to be neutral with regard to specific platforms, manufacturers and applications, and to comply with the usual standards. Furthermore, it had to have a modular and extendable structure. Another requirement was that all data of the application must be adequately described, in order to allow them to be used as references for access.

The ISO/IEC 7816 family of standards was selected as the basis for representing files, with ASN.1 being used to describe formats. The commands necessary for processing the various data items do not form part of the PCKS #15 specification. For all of these reasons, PCKS #15 can be implemented relatively easily in a smart card having a typical multiapplication operating system. However, the concept allows any desired security token to be used for the PCKS #15 functions.

Up to now, PCKS #15 has been implemented in a number of large smart card applications. It is used for the Finnish FINEID[4] personal identification smart card, the Wireless Identification Module (WIM),[5] signature applications in SIMs and USIMs for the Wireless Application Protocol (WAP) and similar operational purposes in the telecommunications area. Within a relatively short time, the PCKS #15 specification has become established as broad-based international industry standard for data representation in smart-card digital signature applications.

The entire PCKS #15 application is located in a DF with a fixed AID. The RID has the value 'A0 00 00 00 63'. The associated PIX is the ASCII encoding of "PKCS-15", and thus consists

[4] See also Section 14.6, ‘The FINEID Personal Identification Card’
[5] See also Section 13.5, ‘The WIM’

of the byte sequence '50 4B 43 53 2D 31 35'. However, the PIX may also have a different value, depending on the implementation.

An entry in an optional EF_{DIR} file located directly below the MF, as specified in ISO/IEC 7816, may be used as a reference to the DF holding the PCKS #15 application. If several PCKS #15-compatible applications are present in the smart card, they can be uniquely identified and selected via the EF_{DIR} file. All other files for the application are contained in a directory named $DF_{PKCS\ \#15}$.

The $EF_{TokenInfo}$ file in the $DF_{PKCS\ \#15}$ directory contains general information about the smart card containing the PCKS #15 application. This includes a serial number, manufacturer identifier and supported cryptographic algorithms.

The central and most important EF of the PCKS #15 application is the object directory file (EF_{ODF}), which contains a directory of the directory files. These files can be referenced using two pointers, in order to allow them to be uniquely identified. This EF is effectively a sort of root directory, from which all other EFs of the application can be accessed using pointers.

The directory files EF_{PrKDF}, EF_{PuKDF}, EF_{SKDF}, EF_{CDF} and EF_{AODF} have a common internal structure consisting of a directory and pointers to the associated data (one pointer for each directory entry). These data can be stored in EFs inside the $DF_{PKCS\ \#15}$ directory, or in files located in another DF in the smart card. The use of pointer addressing also allows data in EFs with a transparent structure to be referenced using offset and length parameters. It would thus have been at least theoretically conceivable to store all of the data in a single large EF, although this would hardly be sensible with regard to data organization. The pointer linking system is even designed to allow data external to the card to be referenced using URLs. A typical use for this mechanism would be depositing a signature verification certificate in a trust center so that it does not have to be stored in the smart card. The smart card would then only reference the certificate by means of its URL.

At least one set of directory files must be present in the card, but it is allowed to have several files with the same functional use in a single card. In this case, the EF_{ODF} file acts as a dispatcher for the various files.

The private key directory file (EF_{PrKDF}) and public key directory file (EF_{PuKDF}) contain directories of private and public keys, respectively, that are present in the smart card. These two files thus effectively contain keys for asymmetric cryptographic algorithms, such as RSA, DAS and elliptic curves. Pointers to the associated keys are held in the directories. The secret key directory file (EF_{SKDF}), which has a similar structure, contains a directory with pointers to the secret keys for symmetric cryptographic algorithms present in the smart card. The certificate directory file (EF_{CDF}) and the authentication object directory file (EF_{AODF}) contain directories with pointers to certificates and authentication objects, respectively. X.509 certificates are typically used, and some examples of authentication objects are PINs and data for biometric user identification. Similarly, the data object directory file (EF_{DODF}) contains a directory with pointers to data objects.

The optional $EF_{UnusedSpace}$ file is intended to be used for managing unused memory in EFs that belong to the PCKS #15 application and are already present in the smart card.

In the original specification, the ASN.1 description of the possible data for a PKCS #15 application encompasses 14 pages. We do not consider it to be worthwhile to reproduce this description line by line here, since the original document can be obtained at no charge via the Internet [RSA], and we are not enchanted by multi-page listings in this or any other book. Nevertheless, as an example of such descriptions, Listing 14.1 shows a slightly compressed ASN.1 description of the attributes of PINs.

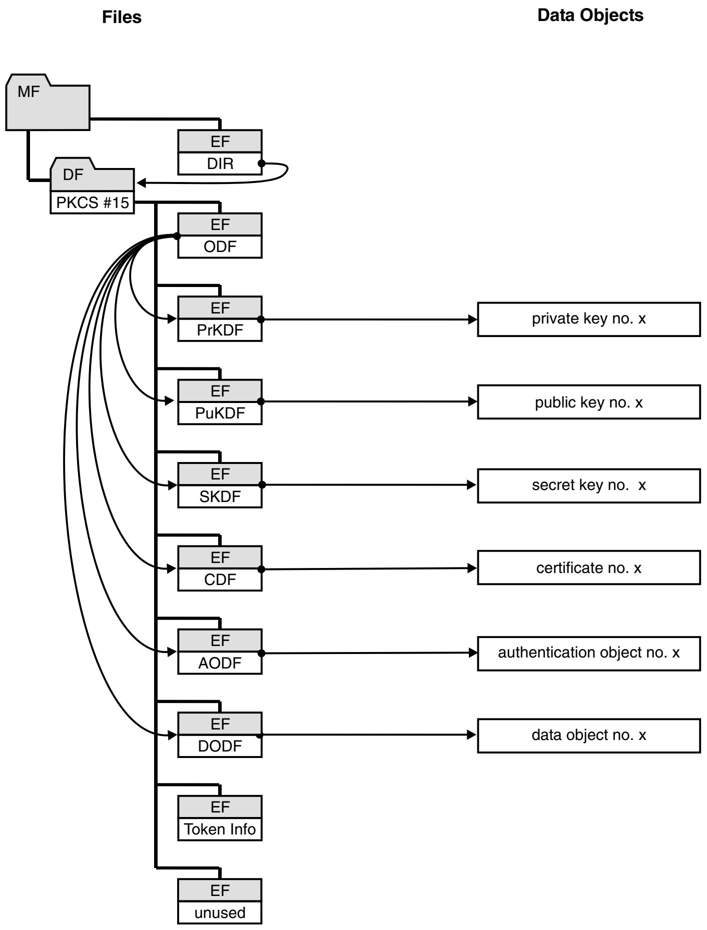


Figure 14.9 Overview of the files and file reference pointers used in a PKCS #15 signature application. The variable *x* represents a non-zero positive integer index that can assume different values for the various data objects

The PKCS #15 specification allows access to certain data objects, such as signature keys, private and secret keys and certificates, to be governed by rules. In accordance with the concept of this specification, this is also implemented using pointers to link the data objects to be protected to corresponding directory files containing information about the types of authentication to be used. One way this can be used is as a simple means to require the user to enter a PIN before using a signature key. In this case, a pointer would be used to link the information about the signature key in EF_{PrKDF} to an entry in EF_{AODF} containing information about the PIN to be used. Pointers referring to the corresponding EFs containing the actual data would also be stored in each of the relevant records in the EF_{PrKDF} and EF_{AODF} files. Finally, a pointer in EF_{PrKDF} would correspondingly identify an entry in EF_{CDF} containing information about the certificate belonging to the signature key, in order to allow the signature to be verified by third

Table 14.5 Typical files for a PKCS #15 application. There are two different types of access conditions: those for read-only (R/O) smart cards, which are used in high-security applications and cannot be modified after being issued, and those for read/write (R/W) smart cards used in normal security applications

$\mathrm{DF}_{\mathrm{PKCS\ \#15}}$		**PKCS #15 application**
	Description:	This directory holds all of the files belonging to the PKCS #15 application.
	File:	AID = 'A0 00 00 00 63 50 4B 43 53 2D 31 35'
$\mathrm{MF.DF}_{\mathrm{PKCS\ \#15}}.\mathrm{EF}_{\mathrm{ODF}}$		**Object directory file (ODF)**
	Description:	This file holds a directory of all files containing data objects that are present in the card, with corresponding pointers to these files.

	File:	FID = '5031'
		R/O accesses: READ: always; UPDATE: never;
		APPEND: EXTERNAL AUTHENTICATE / never
		R/W accesses: READ: always;
		UPDATE: EXTERNAL AUTHENTICATE / never;
		APPEND: EXTERNAL AUTHENTICATE / never

$\mathrm{MF.EF}_{\mathrm{PKCS\ \#15}}.\mathrm{EF}_{\mathrm{PrKDF}}$		**Private key directory file (PrKDF)**
	Description:	This file holds a directory of the available private keys and corresponding pointers to these keys.
	File:	FID = to be specified by the application provider R/O accesses: READ: always / PIN; UPDATE: never; APPEND: EXTERNAL AUTHENTICATE /never R/W accesses: READ: always / PIN; UPDATE: PIN; APPEND: PIN
$\mathrm{MF.DF}_{\mathrm{PKCS\ \#15}}.\mathrm{EF}_{\mathrm{PuKDF}}$		**Public key directory file (PuKDF)**
	Description:	This file holds a directory of the available public keys and corresponding pointers to these keys.
	File:	see $\mathrm{EF}_{\mathrm{PrKDF}}$
$\mathrm{MF.\ DF}_{\mathrm{PKCS\ \#15}}.\mathrm{EF}_{\mathrm{SKDF}}$		**Secret key directory file (SKDF)**
	Description:	This file holds a directory of available secret keys and corresponding pointers to these keys.
	File:	see $\mathrm{EF}_{\mathrm{PrKDF}}$

Table 14.5 (*Cont.*)

MF. $DF_{PKCS\ \#15}$.EF_{CDF}	**Certificate directory file (CDF)**
Description:	This file holds a directory of the available certificates and corresponding pointers to these certificates.
File:	see EF_{PrKDF}
MF. $DF_{PKCS\ \#15}$.EF_{AODF}	**Authentication object directory file (AODF)**
Description:	This file holds a directory of the available authentication objects and corresponding pointers to these authentication objects.
File:	FID = to be specified by the application provider

	R/O accesses: READ: always; UPDATE: never; APPEND: never
	R/W accesses: READ: always;
	UPDATE: PIN / EXTERNAL AUTHENTICATE / never;
	APPEND: PIN / EXTERNAL AUTHENTICATE / never

MF.$DF_{PKCS\ \#15}$.EF_{DODF}	**Data object directory file (DODF)**
Description:	This file holds a directory of the available data objects and corresponding pointers to these data objects.
File:	see EF_{PrKDF}
MF.$DF_{PKCS\ \#15}$.$EF_{TokenInfo}$	**General information file (TokenInfo)**
Description:	This file holds basic information about the application and supported functions.

File:	FID = '5032'
	R/O accesses: READ: always; UPDATE: never; APPEND: never
	R/W accesses: READ: always;
	UPDATE: PIN / AUTHENTICATE / never; APPEND: never

MF.$DF_{PKCS\ \#15}$.$EF_{UnusedSpace}$	**Free memory directory file**
Description:	This file holds a directory of unused memory space present in the EFs of the PKCS #15 application.
File:	FID = '5033' see EF_{PrKDF}

Listing 14.1 Sample ASN.1 description of the attributes of PINs as specified by PKCS #15, in a slightly simplified form

PinAttributes ::= `SEQUENCE {`	**Definition of the PIN attributes data object**
`pinFlags PinFlags,`	The possible binary PIN attributes are defined in the pinFlags data object.
`pinType PinType,`	The coding of the PIN is defined in the pinType data object.
`minLength INTEGER,`	The minimum allowed length of the PIN.
`storedLength INTEGER,`	Length specification for the stored PIN.
`maxLength INTEGER OPTIONAL,`	Optional specification of the maximum allowed length of the PIN.
`padChar OCTET STRING OPTIONAL,`	Optional specification of a fill character for padding the PIN.
`lastPinChange GeneralizedTime OPTIONAL,`	Optional time parameter used for logging the PIN.
`path Path OPTIONAL` `}`	Optional specification of a path to the DF for which the PIN is used.
PinFlags ::= `BIT STRING {`	**Definition of the binary attributes of the PIN**
`case-sensitive(0),`	Specifies whether it is necessary to distinguish between upper- and lower-case characters.
`local(1),`	Specifies whether this is a local (application-specific) or global (card-wide) PIN.
`change-disabled(2),`	Specifies whether the PIN can be changed.
`unblock-disabled(3),`	Specifies whether the retry counter for the PIN is allowed to be reset.
`initialized(4),`	Specifies whether the PIN has been initialized.
`needs-padding(5),`	Specifies whether the transferred PIN must be padded with a fill character.
`unblockingPin(6),`	Specifies whether the PIN is allowed to be used for resetting a PIN retry counter.

`soPin(7),`	Specifies whether this is a security officer PIN per PKCS#11.
`disable-allowed(8),`	Specifies whether PIN queries are allowed to be disabled for this PIN.
`integrity-protected(9),`	Specifies whether the PIN is only allowed to be made known to the smart card using secure messaging (secured by a MAC).
`confidentiality-protected(10),`	Specifies whether the PIN is only allowed to be made known to the smart card using secure messaging (secured by encryption).
`exchangeRefData(11)` `}`	Specifies whether both the old and new values must be given when the PIN is changed, or only the new value.
PinType ::= `ENUMERATED {`	**Definition of representation of the PIN value**
`bcd, ascii-numeric, utf8, ..., half-nibble-bcd, iso9561-1` `}`	The PIN can be coded as follows: BCD, ASCII, UTF8, half-nibble-BCD or ISO 9561-1.

parties. This example is illustrated in Figure 14.10. Although it takes a while to get used to this pointer-based concept, it is very powerful, flexible and above all open to future extensions.

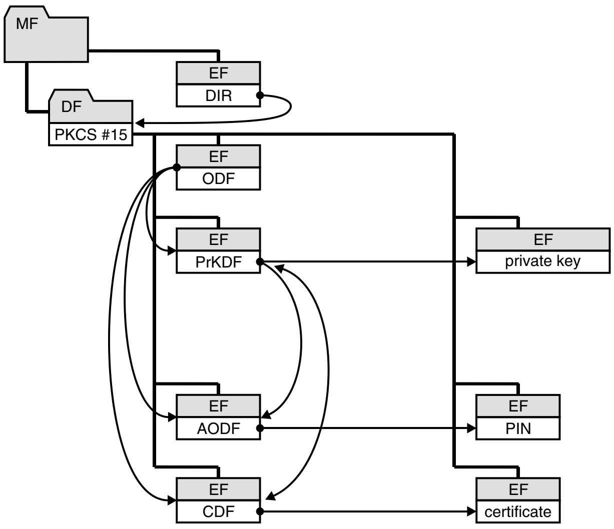


Figure 14.10 Example of a typical PKCS #15 pointer structure for protecting a signature key using prior PIN query and a link to a certificate that can be used to verify the data signed using the signature key. This diagram shows only the files necessary for a basic understanding of the structure

14.6 THE FINEID PERSONAL IDENTIFICATION CARD

In December 1999, Finland became the first country in the world to begin issuing electronic personal identification cards. The name of this system is Finnish Electronic Identification Card (FINEID). Each personal identification card is issued in ID-1 format with a photograph, signature and various other information. The card also has a smart card microcontroller in the standard position for contact-type data transmission.

The FINEID smart cards are part of a public-key infrastructure and are based on a PKCS #15-compliant signature application.[6] The related specifications are public and can be obtained free of charge via the Internet [FIENID]. Each FINEID smart card is valid for three years, and it can be used as a travel document in most European countries. Only the human-readable visual information on the card is necessary for this purpose, which does not require using a microcontroller.

However, the signature application in the smart card allows the FINEID card to also be used to generate legally valid digital signatures, which makes the entire spectrum of e-commerce and e-government applications available to the card user.

14.7 TACHOSMART

According to an ordinance of the Council of the European Community [EC 98], the control devices for road transport, which are called tachographs (or in common usage, 'trip recorders'), are to be replaced in the medium term by electronic tachographs using smart cards. These driver cards are referred to in the ordinance in question in highly simplified terms as 'memory cards', since they must be able to store data.

The operating principle of the Tachosmart system is as follows. Using suitable distance and speed sensors, the tachograph (which is protected against manipulation) measures the distance traveled by the vehicle as well as its speed. It also includes a real-time clock that is protected against external influences, as well as a smart card terminal. The driver of the vehicle receives his or her own personalized smart card, called a 'driver card', which can be used to identify the driver with respect to the tachograph unit.

The tachograph can thus monitor and log the amount of time each driver spends at the wheel and the speed of the vehicle. The EC ordinance provides that this information must be held in the smart card for at least 28 calendar days for each driver. Data older than this may be overwritten as necessary. However, the actual tachograph must store data for the previous 365 days in a manner that is secure against manipulation.

As is usual with EC ordinances, the details are governed by national legislation and specifications generated by tachograph manufacturers. Unfortunately, the technical documents are confidential, so it is not possible to publish any details. However, it presently appears that each smart card will use digital signatures that are compliant with the ISO/IEC 7816-4 and 7816-8 standards. Digital signatures are necessary to protect the data for the last 365 days stored in the tachograph against modification.

[6] See also Section 14.5, 'The PKCS #15 Signature Application'

There are also other types of cards in this system. They include test-station cards, which are issued by official agencies to bodies authorized to calibrate and program tachographs; control cards, which are used by official agencies; and company cards, which are used by vehicle owners to access the data stored in the tachographs and driver cards.

A relatively extensive public key infrastructure (PKI) is necessary to allow this rather elaborate infrastructure to be used securely, economically and throughout all of Europe.

15
Application Design

The first section of this chapter contains general information and characteristic data related to using smart cards. This information, which has been distilled from many applications, can be directly used for designing smart card applications. It thus represents a brief summary of the current state of the technology.

Much of the technical information in this chapter is strongly dependent on the actual hardware used, but it can nevertheless be used to generate estimates for a projected application that are adequate for practical purposes. In part, this is because physical and electrical characteristics of all commonly used smart card microcontrollers are largely the same within certain limits. They primarily differ only in their memory capacities.

In the second section of this chapter, we describe the operating principles of a number of tools that allow even complex applications to be created simply and quickly, without any need for programming. In the final section, several possible applications are illustrated by means of examples. These examples are constructed systematically, in order to make it easy to understand the reasons for using certain mechanisms and procedures.

The notes and examples presented in this chapter have been intentionally structured in a general and flexible manner, so that they can be used as design templates to fashion any desired application.[1]

15.1 GENERAL INFORMATION AND CHARACTERISTIC DATA

15.1.1 Microcontrollers

Production

If a new application requires a special smart card operating system to be developed, this will take a great deal of time. As a general rule, it takes around 12 months to design, program and

[1] Additional design information, particularly with regard to security, can be found in Section 8.2.4.2, 'Attacks at the logical level'

Smart Card Handbook, Third Edition. W. Rankl and W. Effing
© 2004 John Wiley & Sons, Ltd ISBN: 0-470-85668-8

test a completely new operating system. If library routines and existing modules can be used, a single programmer will still need around four to six months for this task. Performing these activities in parallel is possible only to a limited extent, due to the highly complex nature of programming in assembly language or C. As a result, generating a new smart card operating system takes a minimum of two to three months with current software technology, even if no expense is spared and as many people as possible are assigned to the task. Developing an operating system that supports executable program code is even more complex and laborious. As a guideline value, under favorable conditions a development time of around 18 months can be assumed if it is not possible to use existing designs and source code.

Once microcontroller fabrication is started by the semiconductor manufacturer, it takes from eight to 12 weeks until finished dice are available in moderate quantities (several tens of thousands). Placing the dice in the modules takes another one to two weeks. The lead time for producing microcontrollers is thus around 12 weeks.

Service life

The useful life of a smart card microcontroller primarily depends on the stiffness of the card body, the corrosion resistance of the module contacts and the number of possible EEPROM write/erase cycles.

The life expectancy of the card body is heavily dependent on the application area. With typical telecommunications cards (SIM and USIM), which are permanently located in a mobile telephone and never exchanged, there is practically no limit to the useful life of the card with respect to the card body. At the other extreme, such as an employee identification card that is used in the canteen as well as for access control, the card body may crack after two to three years. Keeping the card in a wallet carried in a hip pocket further increases the probability of failure of the card body.

The second limitation on the life expectancy of the card is the maximum number of insertion cycles. The card contacts have hard gold plating to increase their wear resistance, and they can survive approximately 20,000 contact cycles. After this, the contact surfaces will have become so scratched that dirt and grease will adhere to them and interfere with reliable electrical contact. In addition, the gold plating will be largely worn off, which leads to oxidation. This also adversely affects the electrical conductivity of the contact surfaces. Naturally, the life expectancy of the contacts (and thus the card) also depends very much on the type of contact unit used and the applied contact force. If the contact unit has an optimal design, the life expectancy of the card may be increased by a factor of two to four.

The primary limitation to the life expectancy of a smart card arises from the limited number of write/erase cycles in the EEPROM.[2] Most semiconductor manufacturers usually guarantee at least 100,000 write/erase cycles for each EEPROM page. However, in practice the EEPROM will not fail until after 500,000 to 1 million cycles if it is used at room temperature with a closely regulated supply voltage.

An EEPROM page fails gradually, rather than abruptly. Two signs of incipient failure are that the EEPROM cannot be set to the desired value on the first write attempt and that data written to the EEPROM are no longer present in the memory after a few hours. If a memory page continues to be used under these conditions, it will not be possible to store any data at

[2] See also Section 3.4.2, 'Memory types'

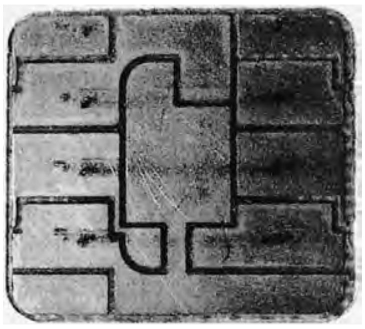


Figure 15.1 Photograph of a module after 50,000 insertion cycles using very high-quality contact elements in a testing machine constructed for this sort of testing. In practice, the contacts may be abraded considerably faster, depending on the type of contact unit used

all in the EEPROM page after a few thousand additional cycles, since the contents will be lost immediately. However, only one page at a time is affected (typical page sizes are 4 bytes and 32 bytes). Other memory pages remain unaffected by the failure of a particular page. This fact can be utilized to devise error recovery strategies when designing memory structures.

Data bits are stored in EEPROM as electrical charges held in tiny capacitors. Like all such devices, they have leakage currents that cause charge to be lost over time, which results in loss of the stored data. This effect is accelerated at high temperatures. The data retention time of EEPROM is thus not unlimited, and the value guaranteed by the manufacturer is only 10 years. The difficulty with specifying the data retention time is that it cannot be measured directly, since this would require waiting 10 years. Instead, the discharge time must be calculated by determining the value of the leakage current. Since it is almost impossible to measure this current, the maximum data retention time is determined indirectly by extrapolating the results of tests made with various values of certain ambient conditions, such as temperature and programming voltage. The data retention time also depends on the number of write/erase cycles. The specified value of 10 years applies to maximum-stress conditions, so a much longer lifetime can be expected under normal conditions. In terms of application design, it can be assumed that data will be retained for 10 years, although this limit should not actually be reached if at all possible.

A blanket statement about the life expectancy of a smart card can only be made if all of the general conditions are taken into consideration. A practical guideline is three to five years for a normal application without special requirements. In some areas, such as GSM, many organizations have decided to replace cards only as necessary (when they actually fail), which is economically advantageous.

Data transfer

The speed at which commands can be processed with a smart card depends primarily on the data transmission rate of the interface between the terminal and the smart card, in addition to internal processing speed.[3]

[3] See also Section 6.4, 'Data Transmission Protocols'

Since an asynchronous serial interface is used, each transmitted byte has to be extended to 12 bits, since a start bit, a parity bit and two stop bits must be transmitted in addition to the 8 bits of user data. This means that a working value of 1.25 ms per byte can be assumed at a data transmission rate of 9600 bit/s. Adding a margin of 20% for the obligatory transmission protocol overhead yields a value of 1.5 ms per byte. This is sufficiently accurate for making practical estimates of the durations of data transfers.

Table 15.1 Typical data transmission rates for smart cards

Function	Data rate with a 3.5-MHz clock	Data rate with a 4.9-MHz clock
Data transmission with divider = 372	9600 bit/s	13,212 bit/s
Data transmission with divider = 512	6975 bit/s	9600 bit/s
Data transmission with divider = 372, T = 0 or T = 1	≈ 7680 bit/s	10,570 bit/s
Data transmission with divider = 372, T = 1 and secure messaging (authentic mode)	≈ 3800 bit/s	≈ 5200 bit/s
Data transmission with divider = 372, T = 1 and secure messaging (combined mode)	≈ 1900 bit/s	≈ 2600 bit/s

Algorithm execution times

Since smart cards are often used as secure computers for executing algorithms, several typical values of processing times for cryptographic algorithms are provided separately in Section 4.7, ‘Cryptology’, arranged by algorithm type.

15.1.2 Applications

Besides the factors specifically related to the microcontroller, there are other aspects that should be considered in the conceptual design and development of a smart card application. With regard to conceptual design, these primarily relate to key distribution, managing application data and the basic principles of data exchange.

Key management

In all applications that use cryptographic algorithms, security is based on the secrecy of the associated keys. If a secret key becomes known for any reason, all of the security mechanisms based on it are rendered worthless. In principle, this eventuality cannot be ruled out with complete certainty, so suitable precautions must be taken.[4] The most trivial and most expensive option is to replace all of the cards, but this is economically out of the question in a large

[4] See also Section 4.8, ‘Key Management’

application. As a general rule, the direct cost of such replacement can be assumed to be 30 euros per card. In practice, therefore, the following measures are used to minimize the consequences of a key becoming known:

- Fundamentally, only card-specific keys should be located in smart cards. This means that each card has its own keys, which can only be used to clone a particular card if they become known. The associated master key must remain secret under all conditions.
- In order to provide protection in case a master key becomes compromised, several generations of card-specific keys can be stored in the cards. This makes it possible to switch over to a new key generation if necessary.
- Keys can be separated according to their intended uses. In practice, this means that (for example) an authentication key is not allowed to be used for data encryption. This helps minimize the consequences of a key becoming known.

These three principles should always be observed in every large application, as they guarantee the application operator a significantly more secure system. They may even save him a great deal of money that would otherwise be spent on replacing all of the cards. However, each of these measures increases the number of keys in the system. This can quickly lead to memory problems, and it requires a more-or-less elaborate key management system. In practice, therefore, it is necessary to very carefully consider whether all three of these measures should be used, or whether it is possible to accept certain compromises.

Data management

The principles that are standard in the informatics industry should be observed with regard to storing data in smart cards. For an application, this means that numerical classification schemes should be avoided as much as possible, since even small modifications or extensions often cause such schemes to collapse. Numerical identification schemes, on the other hand, are often excessively abstract, so in practice mixed schemes predominate.

Telephone-number systems provide a good example of a mixed numbering scheme. The first part of the number (the exchange number) is classificatory. If this number is known, the region where the telephone is located can be positively determined. The second part of the number (the subscriber number) is purely identificatory, since at least in small towns, it provides no information about the location of the subscriber. The two parts together form a typical mixed number, which can be extended in a fairly straightforward manner.

ASN.1-coded data objects[5] are very suitable for flexibly handling data objects having several different versions. Based on actual practice, the coding overhead using BER-TLV can be assumed to be approximately 25 % of the user data volume. This applies only to small data sets, but this is precisely the most common case in the smart card domain.

A basic remark relating to the memory capacities of smart cards can be made here: applications with more than 10 kB of user data are rare, since the available amount of memory is usually very limited.

[5] See also Section 4.1, 'Data Structures'

When a new application is being planned, the amount of memory that will have to be reserved in the smart card should be at least approximately estimated. This should include not only the user data, but also the necessary amount of administrative data. In modern operating systems, which have object-oriented file management systems and allow several applications to be present in a single smart card at the same time, the proportion of administrative data is relatively high. The size of the header for each file is usually 16 to 32 bytes. The amount of administrative data per file is fixed, which means that it does not depend on the amount of user data in the file. It makes no difference to the size of the file header whether there is only 1 byte or 200 bytes of user data in the file. Consequently, an effort is made to avoid creating a separate file for each data element, since otherwise too much memory would be taken up by administrative data.

Data exchange

Two primary considerations must be borne in mind with regard to data exchange between the terminal and the smart card. The first is that the card's serial interface is very slow compared with the performance of modern computer systems. Although the commonly used rate of 9600 bit/s makes data transmission very robust with respect to interference, it also means that data exchanges take a great deal of time. It is therefore a good idea to restrict the data exchanged between the terminal and the card to essential items. During a session, the terminal should not again request any data it has already received. Naturally, this does not apply to applications in which time has a very low priority. However, if people are involved in the process, minimizing the volume of data exchanged via the interface must always be given high priority.

With regard to data exchange, the following considerations are also relevant. If secure messaging[6] is used, in principle all of the data exchanged via the terminal/card interface can be very well protected, thus making an attack via this interface nearly impossible. However, secure messaging reduces the effective transmission rate. Consequently, it should only be used for data transmission when it is indispensable for reasons of system security. With the exception of secret keys, it is almost always sufficient to simply use a MAC to protect the user data (authentic mode). This does not reduce the transmission rate as severely as the additional encryption required by the combined mode. In addition, the authentic mode yields transparent data transmission at the interface, so the data can easily be checked externally. In some cases, this may be necessary in connection with data privacy legislation, since it allows the data passing across the interface to be monitored at any time without knowledge of the secret keys.

15.1.3 System considerations

Security

Many applications use triple DES as the cryptographic algorithm. There are many reasons for this, but in most cases the main reason is probably that DES and its cascaded version, triple DES,

[6] See also Section 6.6, 'Securing Data Transmission'

are very well known, so there is no need to venture into unknown territory. However, there are many other good data encryption and decryption algorithms besides DES. The fact that DES is well known also means that it is subject to the greatest number of attacks, one of which may sometimes be successful. In the case of a large smart card project, therefore, consideration should be given to using a cryptographic algorithm other than DES. This will minimize the consequences to the system if there is a successful attack on the DES algorithm. The current GSM cryptographic algorithms provide an excellent illustration of this philosophy. They were specially developed for this application, and to a certain extent they are fully independent of the DES algorithm.

User interface

User acceptance is critical to the success of a smart card project. Although this primarily relates to the man–machine interface of the terminal, the interface between the terminal and the card also has an effect. Experience shows that user acceptance suffers when transactions take longer than one second. In such cases, users often assume that there is a technical problem and attempt to pull the card out of the terminal, resulting in an uncontrolled termination of the session. In order to avoid this, all processes between the terminal and the smart card should be optimized to take less than one second to complete.

Above all, in this regard it is important to bear in mind that if the level of user acceptance is inadequate, the system will experience a significantly higher level of technical problems due to unforeseeable interventions, such as prematurely withdrawing the card from the terminal.

Design

Kerckhoff’s principle should always be applied in the process of designing a data processing and management system that uses smart cards. This principle states that the security of a system should rely solely on its secret keys. It is very difficult to test a system whose security is based on confidentiality of the data, because the tests can only be performed by a very limited number of people. Since this group of people has usually been involved in the design of the system, the quality of testing performed on this basis cannot be very high. The alternative of making all of the internal data available to the testers has its own drawbacks, since this could give a potential attacker free access to the data. In practice, it is generally a good idea to use a compromise approach, in which the fundamental design of the system is open but some special items are kept confidential.

If, completely counter to Kerckhoff’s principle, certain information in a smart card project must remain secret, it is a good idea to divide this knowledge among several people, rather than having the security of this information depend on a single person. In this way, each person is familiar with part of the system, but no single person knows the entire system.

With regard to designing a system based on smart cards, it is important to always keep the entire system in mind, rather than concentrating only on the cards and their immediate environment. This is the most commonly observed mistake in the course of many projects. Since smart cards are active devices, systems that employ them are always distributed systems, whose individual elements must act autonomously and in which pure master–slave relationships do not prevail. After the system has been integrated, it is essential that all of its components work harmoniously together, rather than just the cards and terminals. It is thus necessary to acquire and maintain an overall project perspective from the very beginning.

15.1.4 Compliance with standards

Smart cards are one of the very few components in the entire realm of informatics that are strongly based on international standards. This is in part due to the necessity for mutual compatibility among the products of various manufacturers of smart cards and terminals, and in part due to the requirement for interoperability among various smart card systems. Consequently, here we list a number of general standards and specifications that normally should be supported by current smart card operating systems, or with which such systems should be compatible.[7] Naturally, relevant national standards and specifications must also be observed for actual applications.

The most important family of standards is ISO/IEC 7816. It specifies the essential general physical and informatic constraints. With regard to file systems, commands and file access conditions, Parts 4, 8 and 9 of this family are of major significance. However, in this regard it should be borne in mind that this family of standards specifies a sort of ‘construction kit’ from which various items may be selected as desired. In other words, it is not necessary for every smart card to fully satisfy all aspects of every item specified in the standard.

In the realm of telecommunications, the TS 102.221 UICC specification is a key document. In addition, the GSM 11.11 specification for the SIM and the TS.31.102 specification for the USIM are fundamental documents for international telecommunications applications.

In the payment systems field, the EMV specification is a *de facto* standard that must be recognized by all other specifications. With regard to electronic purses, EN 1546 represents a basic standard, while CEPS is an exemplary model of an international purse system.

Presently, smart card operating systems with executable program code are practically limited to Java Card and its associated specification. The Open Platform specification is almost exclusively used as the reference for downloading applications.

Smart cards used in the digital signature environment are frequently based on the PKCS #15 standard. Certificates for use with digital signatures are frequently stored in X.509-compliant formats.

15.2 FORMULAS FOR ESTIMATING PROCESSING TIMES

In the process of developing the conceptual design of a smart card application or designing new smart card commands, it is frequently necessary to estimate processing times. Even when values acquired from experience are used, it is relatively difficult to obtain sufficiently accurate numbers using empirical estimates. In this section, we provide a collection of basic formulas for calculating processing times for smart card operating systems using contact-type interfaces for data transfer. If these formulas are used properly, they will give results that are acceptably

[7] A directory of smart card standards with comments can be found in Section 16.4, ‘Annotated Directory of Standards and Specifications’

accurate. However, they should by no means be regarded as foolproof or perfectly accurate under all circumstances. They are intended to be used to make numerical estimates with an error tolerance on the order of 10 %. We can thus highly recommend adding an appropriate safety margin in critical situations.

The names of the variables in the following formulas are intended to be self-explanatory, which makes relatively long names necessary. Unless otherwise explicitly stated, all fixed times are based on a standard clock frequency of 3.5712 MHz. However, the indicated times can be adjusted for other clock frequencies using the proportionality factor ***PF*** from Formulas (15.9) and (15.10). In all cases, it is assumed that the command in question is processed without any errors and that no errors occur during EEPROM operations or data transmission. With regard to data transmission, it is further assumed that command processing starts immediately after the last bit has been received, which means that the shortest possible character waiting time (CWT) has been chosen.

Command processing

The following three formulas form the basis for all timing calculations. They divide the total processing time into two parts: the data transfer time and the processing time inside the smart card:

$$t_{\text{total}} = t_{\text{data\_transter}} + t_{\text{ICC\_internal}} \tag{15.1}$$

$$t_{\text{data\_transfer}} = t_{\text{data\_transfer\_command}} + t_{\text{data\_transfer\_response}} \tag{15.2}$$

$$t_{\text{ICC\_internal}} = t_{\text{command\_interpreter}} + t_{\text{command\_execution}} \tag{15.3}$$

The time that the command interpreter needs for its activities depends only on the frequency of the applied clock signal, which is here taken to be 3.5712 MHz:

$$t_{\text{command\_interpreter}} \approx 1.5 \text{ ms} \tag{15.4}$$

$$t_{\text{command\_execution}} = t_{\text{EEPROM}} + t_{\text{cryptographic\_algorithm}} + t_{\text{command\_code}} \tag{15.5}$$

The exact processing time for a command can only be determined by a detailed analysis of program execution at the machine-code level. Given the large number of program branches and loops, as well as the various command options, this would lead to very complex formulas that would be unusable in practice. Consequently, we have simply divided the commands into three groups according to their complexity. Simple commands, such as SELECT FILE and READ BINARY, take the least time. Moderately complex commands, such as INTERNAL AUTHENTICATE, require somewhat more time for their internal processes. Highly complex commands, such as DEBIT IEP, take the longest time. Here it should be borne in mind that these global values do not include actions such as running cryptographic algorithms or executing EEPROM write accesses, but only the essential internal computations and queries.

$$t_{\text{command\_code}} \approx 5 \text{ ms (for simple commands)} \tag{15.6}$$

$$t_{\text{command\_code}} \approx 12 \text{ ms (for moderately complex commands)} \tag{15.7}$$

$$t_{\text{command\_code}} \approx 20 \text{ ms (for highly complex commands)} \tag{15.8}$$

Table 15.2 Definitions and descriptions of the variables for Formulas (15.1) through (15.8)

Variable	Unit	Description
$t_{\text{data\_transfer}}$	s	Data transfer time for a command and its associated response
$t_{\text{data\_transfer\_response}}$	s	Data transfer time for the response to a command
$t_{\text{data\_transfer\_command}}$	s	Data transfer time for a command
t_{EEPROM}	s	Time required to write data to the EEPROM
t_{total}	s	Time required to receive a command, process it and send the associated response
$t_{\text{ICC\_internal}}$	s	Time required to process a command inside the smart card
$t_{\text{command\_execution}}$	s	Time required to execute a command
$t_{\text{command\_code}}$	s	Time required to execute a specific program for a particular command (such as a cryptographic algorithm)
$t_{\text{command\_interpreter}}$	s	Time required to analyze a command and call the associated program code
$t_{\text{cryptographic\_algorithm}}$	s	Time required to execute a cryptographic algorithm

Proportionality factor for predefined functions

If the predefined time values given for certain functions assume that a particular clock frequency is used, the proportionality factor ***PF*** can be used as necessary to convert them to values corresponding to the clock frequency that is actually used.

$$PF = \frac{f_{\text{reference}}}{f_{\text{actual}}} \tag{15.9}$$

$$t_{\text{actual}} = t_{\text{reference}} \cdot PF \tag{15.10}$$

Table 15.3 Definitions and descriptions of the variables for Formulas (15.9) and (15.10)

Variable	Unit	Description
f_{actual}	MHz	Actual clock rate
$f_{\text{reference}}$	MHz	Reference clock rate for a given interval that depends on the clock rate
t_{actual}	s	Actual duration of an action
$t_{\text{reference}}$	s	Stated duration of a given interval that depends on the clock rate
PF	—	Proportionality factor for routines whose processing time depends on the clock rate

EEPROM operations

Before data can be written to the EEPROM, the affected part of the EEPROM may first have to be erased, depending on the content of the data to be written. With some smart card microcontrollers, the page size for erasing can be different from the page size for writing. This is taken into account in the following formulas.

To determine whether an EEPROM page must first be erased, the current content of the page and the content of the new data must be known. For conservative estimates, however, it should always be assumed that the page to be written must first be erased.

One small comment is appropriate with regard to the durations of EEPROM write and erase operations. Microcontrollers that are currently commonly used in smart cards do not have internal clocks, so the only timing reference for the operating system is the externally applied clock signal. If the microcontroller has a maximum specified clock frequency of 5 MHz, for example, all EEPROM write routines will be designed for this frequency. This means that if the actual clock frequency is less than the maximum value, the EEPROM write time will be proportionally longer. For precise calculations, this should be taken into account by using the proportionality factor. However, this all depends on the maximum clock frequency, which varies according to the microcontroller type and is also a parameter of the smart card operating system. Consequently, this effect is not taken into account here. In the future, it will not be such a significant factor, since the latest microcontrollers have internal clocks and can thus perform EEPROM operations with fixed timing, independent of the frequency of the applied clock.

$$\boldsymbol{t}_{\text{EEPROM}} = \boldsymbol{t}_{\text{EEPROM\_erase}} + \boldsymbol{t}_{\text{EEPROM\_write}} \tag{15.11}$$

$$\boldsymbol{t}_{\text{EEPROM\_erase}} = \frac{\boldsymbol{t}_{\text{page\_erase}}}{\boldsymbol{PS}_{\text{erase}}} \cdot \boldsymbol{n} \tag{15.12}$$

$$\boldsymbol{t}_{\text{EEPROM\_write}} = \frac{\boldsymbol{t}_{\text{page\_write}}}{\boldsymbol{PS}_{\text{write}}} \cdot \boldsymbol{n} \tag{15.13}$$

Table 15.4 Definitions and descriptions of the variables for Formulas (15.11) through (15.13) for EEPROM operations

Variable	Unit	Example	Description
$\boldsymbol{n}$	byte	20 bytes	Number of bytes to be written to the EEPROM; must be rounded up to the actual page size
$\boldsymbol{PS}_{\text{erase}}$	byte	32 bytes	Page size for erasing
$\boldsymbol{PS}_{\text{write}}$	byte	4 bytes	Page size for writing
$\boldsymbol{t}_{\text{EEPROM}}$	s	21 ms	Time required to write $\boldsymbol{n}$ bytes to the EEPROM, including prior erasing if necessary
$\boldsymbol{t}_{\text{EEPROM\_erase}}$	s	3.5 ms	Time required to erase $\boldsymbol{n}$ bytes in the EEPROM
$\boldsymbol{t}_{\text{EEPROM\_write}}$	s	17.5 ms	Time required to write $\boldsymbol{n}$ bytes in the EEPROM
$\boldsymbol{t}_{\text{page\_erase}}$	s/byte	3.5 ms (4 bytes)	Time required to erase one EEPROM page
$\boldsymbol{t}_{\text{page\_write}}$	s/byte	3.5 ms (4 bytes)	Time required to write one EEPROM page

Data transfer

The time required to transfer the command and the subsequent response depends primarily on the amount of data to be transmitted. The structures of the basic transmission protocol data units (TPDUs) and application protocol data units (APDUs) are described in detail in Sections 6.4.2 ('The T = 0 transmission protocol') and 6.4.3 ('The T = 1 transmission protocol').

$$t_{\text{data\_transfer\_command}} = t_{\text{byte\_transfer}} \cdot n_{\text{command\_data}} \tag{15.14}$$

$$t_{\text{data\_transfer\_response}} = t_{\text{byte\_transfer}} \cdot n_{\text{response\_data}} \tag{15.15}$$

$$n_{\text{command\_data}} = n_{\text{level\_2}} + n_{\text{header}} + n_{\text{body}} \tag{15.16}$$

$$n_{\text{response\_data}} = n_{\text{level\_2}} + n_{\text{trailer}} + n_{\text{body}} \tag{15.17}$$

Table 15.5 Definitions and descriptions of the variables for Formulas (15.14) through (15.17)

Variable	Unit	Typical value	Description
$n_{\text{command\_data}}$	byte	—	Amount of data transferred
$n_{\text{response\_data}}$	byte	—	Amount of data transferred
n_{header}	byte	4	Number of bytes in the command header. For the T = 1 protocol, these are the CLS, INS, P1 and P2 bytes
n_{body}	byte	—	Number of bytes in the command body or associated response body. If the command includes a data part, it includes a 1-byte or 2-byte length parameter for the command body or response body
$n_{\text{layer 2}}$	byte	4	Number of bytes for transport layer (layer 2) For the T = 1 protocol, these are the NAD, PCB, LEN and EDC bytes
$n_{\text{layer 2}}$	byte	4	Number of bytes for transport layer (layer 2) For the T = 1 protocol, these are the NAD, PCB, LEN and EDC bytes
$t_{\text{data\_transfer\_command}}$	s	—	Time required to transmit a command
$t_{\text{data\_transfer\_response}}$	s	—	Time required to transmit the response to a command
$t_{\text{byte\_transfer}}$	s	—	Time required to transmit one byte

The total time required to transmit a single byte depends on the clock rate, the bit-rate adjustment factor and the clock-rate conversion factor:

$$t_{\text{byte\_transfer}} = \frac{1}{D} \cdot \frac{F}{f} \cdot n \tag{15.18}$$

Table 15.6 Definitions and descriptions of the variables for Formula (18)

Variable	Unit	Typical value	Description
D	bit·byte/MHz·ms	1 bit·byte/MHz·ms	Bit-rate adjustment factor
f	MHz	3.5712 MHz	Clock frequency
F	1	372	Clock-rate conversion factor
n	bit	12 bits	Number of bits per byte (1 start bit, 8 data bits, 1 parity bit, 2 stop bits)
$t_{\text{byte\_transfer}}$	ms	1.25 ms	Transmission time for one byte

Calculated example: smart card READ BINARY command

Here we present a sample calculation of the estimated processing time for a smart card command with a simple structure. We have chosen the READ BINARY command for this example. As general conditions, we have selected the T = 1 transmission protocol with a divider value of 372 and a clock frequency of 5 MHz.

The time required for the data transfer can be calculated using Formula (18):

$$t_{\text{byte\_transfer}} = \frac{1}{D} \cdot \frac{F}{f} \cdot n = \frac{1}{1\,\frac{\text{bit} \cdot \text{byte}}{\text{MHz} \cdot \text{ms}}} \cdot \frac{372}{5\,\text{MHz}} \cdot 12 \text{ bits} \approx 0.89 \text{ ms/byte}$$

For data transmission using the T = 1 protocol, four bytes are needed for layer 2 in addition to the data for layer 7. The command header of READ BINARY has a length of four bytes (CLA, INS, P1, and P2), and the associated body contains one byte (Le). The response consists of the data that have been read, with a length of (Le), and a trailer consisting of the status bytes SW1 and SW2. With this information, we can use Formulas (15.16) and (15.17) to express the amount of data to be transmitted as a function of the value of ***Le***:

$$n_{\text{command\_data}} = n_{\text{layer\_2}} + n_{\text{header}} + n_{\text{body}}$$

$$n_{\text{command\_data}} = (4 + 4 + 1) \text{ bytes} = 9 \text{ bytes}$$

$$n_{\text{response\_data}} = n_{\text{layer\_2}} + n_{\text{trailer}} + n_{\text{body}}$$

$$n_{\text{response\_data}} = (4 + Le + 2) \text{ bytes} = (Le + 6) \text{ bytes}$$

From this, we can determine the times required to transmit the command and the response using Formulas (15.14) and (15.15), respectively:

$$t_{\text{data\_transfer\_command}} = t_{\text{byte\_transfer}} \cdot n_{\text{command\_data}} = (0.89\text{ms/byte}) \cdot (9 \text{ bytes}) = 8.01 \text{ ms}$$

$$t_{\text{data\_transfer\_response}} = t_{\text{byte\_transfer}} \cdot n_{\text{response\_data}}$$

$$t_{\text{data\_transfer\_response}} = (0.89 \text{ ms/byte}) \cdot (Le + 6 \text{ bytes}) = 0.89\,(Le + 6 \text{ bytes}) \text{ ms/byte}$$

This command is a simple command, so an execution time of 5 ms at a clock frequency of 3.5712 MHz can be assumed. We can modify this to correspond to the actual clock frequency of 5 MHz using Formulas (9) and (15.10):

$$PF = \frac{f_{\text{reference}}}{f_{\text{actual}}} = \frac{3.5712 \text{ MHz}}{5 \text{ MHz}} = 0.714$$

$$t_{\text{actual}} = t_{\text{reference}} \cdot PF = (5 \text{ ms}) \cdot 0.714 = 3.57 \text{ ms}$$

READ BINARY does not require any data to be written to the EEPROM, nor is it necessary to call a cryptographic algorithm. The time required for the command processing inside the smart card can thus be calculated as follows:

$$\boldsymbol{t}_{\text{command\_execution}} = \boldsymbol{t}_{\text{EEPROM}} + \boldsymbol{t}_{\text{cryptographic\_algorithm}} + \boldsymbol{t}_{\text{command\_code}}$$
$$\boldsymbol{t}_{\text{command\_execution}} = 0\text{ ms} + 0\text{ ms} + 3.57\text{ ms} = 3.57\text{ ms}$$

Under the additional assumption that the command interpreter needs around 1.5 ms to do its job with a 3.5712-MHz clock, we can now calculate the processing execution time for the command:

$$\boldsymbol{t}_{\text{actual}} = \boldsymbol{t}_{\text{reference}} \cdot \boldsymbol{PF} = (1.5\text{ ms}) \cdot 0.714 \approx 1\text{ ms}$$
$$\boldsymbol{t}_{\text{ICC\_internal}} = \boldsymbol{t}_{\text{command\_interpreter}} + \boldsymbol{t}_{\text{command\_execution}} = 1\text{ ms} + 3.5\text{ ms} = 4.5\text{ ms}$$

All of the values determined thus far can now be inserted into Formula (15.2), yielding an expression for the total processing time for the READ BINARY command as a function of the amount of data read:

$$\boldsymbol{t}_{\text{data\_transfer}} = \boldsymbol{t}_{\text{data\_transfer\_command}} + \boldsymbol{t}_{\text{data\_transfer\_response}}$$
$$\boldsymbol{t}_{\text{data\_transfer}} = 8.01\text{ ms} + (\boldsymbol{Le} + 6\text{ bytes})\text{ ms/byte}$$
$$\boldsymbol{t}_{\text{total}} = \boldsymbol{t}_{\text{data\_transfer}} + \boldsymbol{t}_{\text{ICC\_internal}} = 8.01\text{ ms} + (\boldsymbol{Le} + 6\text{ bytes})\text{ ms/byte} + 4.5\text{ ms}$$
$$\boldsymbol{t}_{\text{total}} = (12.51 + 0.89\,(\boldsymbol{Le} + 6\text{ bytes})\text{ byte}^{-1})\text{ ms} = (17.85 + 0.89\,\boldsymbol{Le}\text{ byte}^{-1})\text{ ms}$$

The result of these calculations is a reasonably good order-of-magnitude match to the empirically determined formula for READ BINARY at a clock rate of 3.5712 MHz (see Formula (15.21) in the following section). The difference between the two results arises from deviations in the data transfer times and the assumptions made above regarding times for processes inside the operating system.

Calculated example: smart card initialization

In the following numerical example, we calculate a rough estimate of the time required to initialize a smart card. Here we assume that 5 kB of data must be written to the EEPROM in order to initialize the card. The initialization data are transmitted using the T = 1 protocol with a divider value of 372 and a clock rate of 3.5712 MHz.
From Formula (15.18), the calculated time for transmitting a single byte is:

$$\boldsymbol{t}_{\text{byte\_transfer}} = \frac{1}{\boldsymbol{D}} \cdot \frac{\boldsymbol{F}}{\boldsymbol{f}} \cdot \boldsymbol{n} = \frac{1}{1\ \dfrac{\text{bit} \cdot \text{byte}}{\text{MHz} \cdot \text{ms}}} \cdot \frac{372}{3.5712\ \text{MHz}} \cdot 12\text{ bits} \approx 1.25\text{ ms/byte}$$

Assuming that the initialization command has a 4-byte header (CLA, INS, P1 and P2), the length parameter is 1 byte (***Lc***), 100 bytes of user data are transmitted per command and the response consists only of SW1 and SW2, we can calculate the number of bytes of data that are transmitted for the command and response:

$$\begin{gathered}
n_{\text{command\_data}} = n_{\text{level\_2}} + n_{\text{header}} + n_{\text{body}} \\
n_{\text{command\_data}} = (4 + 4 + 1 + 100) \text{ bytes} = 109 \text{ bytes} \\
n_{\text{response\_data}} = n_{\text{level\_2}} + n_{\text{trailer}} + n_{\text{body}} = (4 + 0 + 2) \text{ bytes} = 6 \text{ bytes}
\end{gathered}$$

From this, we can determine the duration of the transmission for the command and response:

$$\begin{aligned}
t_{\text{data\_transfer\_command}} &= t_{\text{byte\_transfer}} \cdot n_{\text{command\_data}} \\
&= (1.25 \text{ ms/byte}) \cdot (109 \text{ bytes}) = 136.25 \text{ ms} \\
t_{\text{data\_transfer\_response}} &= t_{\text{byte\_transfer}} \cdot n_{\text{response\_data}} \\
&= (1.25 \text{ ms/byte}) \cdot (6 \text{ bytes}) = 7.5 \text{ ms}
\end{aligned}$$

One hundred bytes of data must be written to the EEPROM. Under the additional assumption that one EEPROM page is 4 bytes and the write time is 3.5 ms per page, we can determine the time required to write the data to the EEPROM for each command:

$$t_{\text{EEPROM\_write}} = \frac{t_{\text{page\_write}}}{PS_{\text{write}}} \cdot n = \frac{3.5 \text{ ms}}{4 \text{ bytes}} \cdot 100 = 3.5 \text{ ms} \cdot 25 = 87.5 \text{ ms}$$

We also assume that the EEPROM does not have to be erased before the write operation, since this has already taken place during microcontroller testing:

$$t_{\text{EEPROM}} = t_{\text{EEPROM\_erase}} + t_{\text{EEPROM\_write}} = 0 \text{ ms} + 87.5 \text{ ms} = 87.5 \text{ ms}$$

It is not necessary to call a cryptographic algorithm for initialization, and the command that is used has a simple internal structure. An execution time of around 5 ms can therefore be assumed for the command code:

$$\begin{gathered}
t_{\text{command\_execution}} = t_{\text{EEPROM}} + t_{\text{cryptographic\_algorithm}} + t_{\text{command\_code}} \\
t_{\text{command\_execution}} = 87.5 \text{ ms} + 0 \text{ ms} + 5 \text{ ms} = 92.5 \text{ ms}
\end{gathered}$$

Now we have to insert the values calculated using Formulas (15.3) and (15.2), and this completes our calculation of the time required for an initialization command with 100 bytes of data. The command interpreter needs 1.5 ms on top of this:

$$\begin{gathered}
t_{\text{ICC\_internal}} = t_{\text{command\_interpreter}} + t_{\text{command\_execution}} = 92.5 \text{ ms} + 1.5 \text{ ms} = 94 \text{ ms} \\
t_{\text{data\_transfer}} = t_{\text{data\_transfer\_command}} + t_{\text{data\_transfer\_response}} \\
t_{\text{data\_transfer}} = 136.5 \text{ ms} + 7.5 \text{ ms} = 144 \text{ ms} \\
t_{\text{total}} = t_{\text{data\_transfer}} + t_{\text{ICC\_internal}} = 144 \text{ ms} + 94 \text{ ms} = 238 \text{ ms}
\end{gathered}$$

We have thus calculated that it takes 238 ms to transmit 100 bytes of data to the smart card, write the data to the EEPROM and return a response to the terminal in order to confirm successful processing of the command. However, according to our initial assumptions, a total of 5 kB of data (5120 bytes) must be written to the memory. To simplify the calculations, we assume that this takes approximately 52 times as long:

$$t_{\text{initialization}} = t_{\text{total}} \cdot 52 = 12.4 \text{ s} \approx 13 \text{ s}$$

According to our calculations, the initialization process should take 12.4 seconds. If we include a small safety margin, we can assume that the initialization will not take longer than 13 seconds. However, any transmission errors or EEPROM errors that may occur during initialization are not taken into account in our calculations. Such errors must be regarded as a sort of 'force majeure' that can only be dealt with statistically.

15.3 TIMING FORMULAS FOR TYPICAL SMART CARD COMMANDS

The formulas in this section are based on timing measurements made using actual smart card operating systems. Linear equations have been fitted to these measurements. All of these formulas are based on a smart card operating system with the T = 1 transmission protocol, a clock rate conversion factor of 372 and a bit-rate adjustment factor (***D***) of 1. The microcontroller

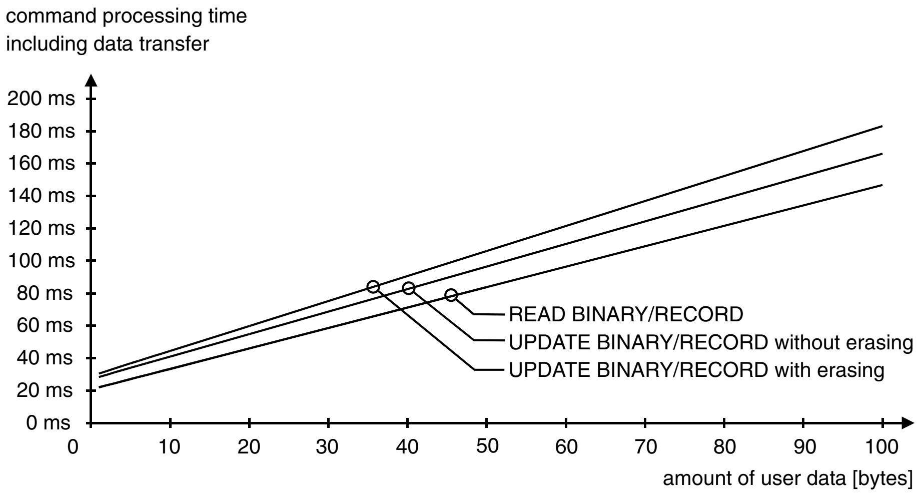


Figure 15.2 Processing times for smart card commands as a function of the amount of user data, including the data transfer times for the command and response. This chart is based on the T = 1 transmission protocol with a clock-rate conversion factor of 372, a clock frequency of 3.5712 MHz and an EEPROM write/erase cycle time of 3.5 ms for a 4-byte page

is clocked at 3.5712 MHz, and the write/erase cycle time of the EEPROM is 3.5 ms for a 4-byte page. It is also assumed that no error occurs during data transmissions or any necessary EEPROM write/erase operations. The formulas are valid for all values of ***n*** between 1 and 254.

Formulas (15.19) and (15.20) are intended to be used to calculate times for the SELECT FILE command, with the option of file selection using a 2-byte FID or an ***n***-byte DF name:

$$\boldsymbol{t}_{\text{total SELECT FILE with FID}} \approx 23 \text{ ms} \qquad (15.19)$$

$$\boldsymbol{t}_{\text{total SELECT FILE with DF name}} \approx (20.75 + 1.26 \cdot \boldsymbol{n}) \text{ ms} \qquad (15.20)$$

Formulas (15.21) through (15.24) can be used to estimate the times required by read commands for files with transparent and record-oriented structures. The variable ***n*** is the number of bytes to be read. These formulas can also be used if READ BINARY or READ RECORD is used with implicit file selection, since the time difference due to processing the implicit file selection is negligible:

$$\boldsymbol{t}_{\text{total READ BINARY}} \approx (20.77 + 1.26 \cdot \boldsymbol{n}) \text{ ms} \qquad (15.21)$$

$$\boldsymbol{t}_{\text{ICC\_internal READ BINARY}} \approx (2.02 + 0.01 \cdot \boldsymbol{n}) \text{ ms} \qquad (15.22)$$

$$\boldsymbol{t}_{\text{total READ RECORD}} \approx (20.70 + 1.26 \cdot \boldsymbol{n}) \text{ ms} \qquad (15.23)$$

$$\boldsymbol{t}_{\text{ICC\_internal READ RECORD}} \approx (1.95 + 0.01 \cdot \boldsymbol{n}) \text{ ms} \qquad (15.24)$$

Formulas (15.25) through (15.32) can be used to estimate the processing times for UPDATE BINARY and UPDATE RECORD commands, with and without implicit file selection. The command duration depends primarily on whether it is necessary to erase the associated EEPROM page prior to the write operation. The formulas are therefore divided into two sets according to this condition. With the formulas that include erasing prior to writing, it is always assumed that all pages must be erased. The number of bytes to be written is denoted by ***n***.

$$\boldsymbol{t}_{\text{total UPDATE BINARY without erasing}} \approx (25.55 + 1.39 \cdot \boldsymbol{n}) \text{ ms} \qquad (15.25)$$

$$\boldsymbol{t}_{\text{ICC\_internal UPDATE BINARY without erasing}} \approx (6.8 + 0.14 \cdot \boldsymbol{n}) \text{ ms} \qquad (15.26)$$

$$\boldsymbol{t}_{\text{total UPDATE BINARY with erasing}} \approx (27.26 + 1.54 \cdot \boldsymbol{n}) \text{ ms} \qquad (15.27)$$

$$\boldsymbol{t}_{\text{ICC\_internal UPDATE BINARY with erasing}} \approx (8.51 + 0.29 \cdot \boldsymbol{n}) \text{ ms} \qquad (15.28)$$

$$\boldsymbol{t}_{\text{total UPDATE RECORD without erasing}} \approx (25.35 + 1.38 \cdot \boldsymbol{n}) \text{ ms} \qquad (15.29)$$

$$\boldsymbol{t}_{\text{ICC\_internal UPDATE RECORD without erasing}} \approx (6.7 + 0.14 \cdot \boldsymbol{n}) \text{ ms} \qquad (15.30)$$

$$\boldsymbol{t}_{\text{total UPDATE RECORD with erasing}} \approx (27.13 + 1.54 \cdot \boldsymbol{n}) \text{ ms} \qquad (15.31)$$

$$\boldsymbol{t}_{\text{ICC\_internal UPDATE RECORD with erasing}} \approx (8.4 + 0.28 \cdot \boldsymbol{n}) \text{ ms} \qquad (15.32)$$

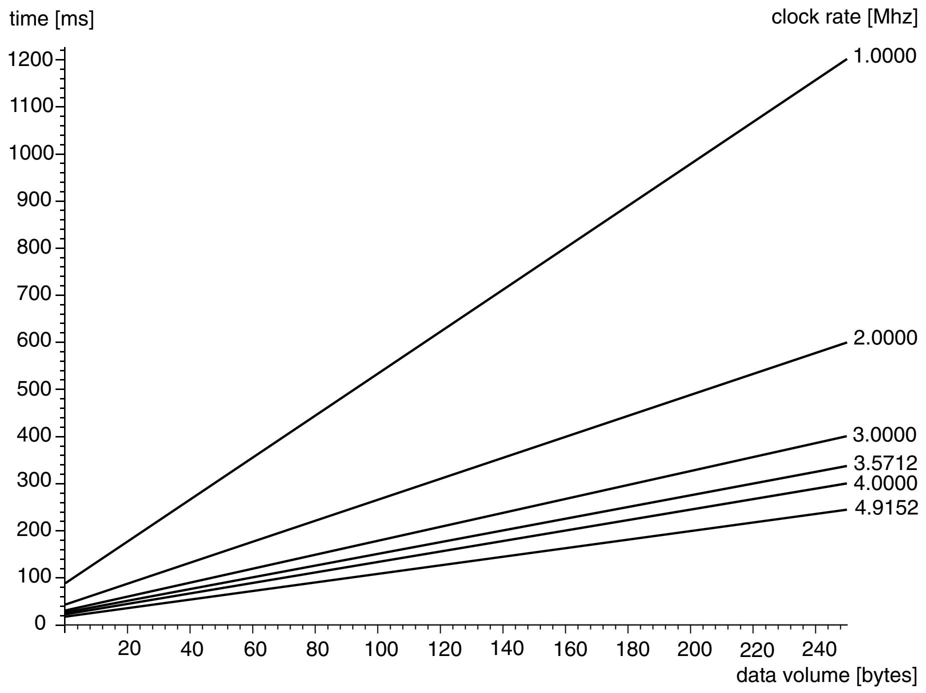


Figure 15.3 Data transfer time as a function of data volume and clock frequency. The values shown are based on the T = 1 data transmission protocol with a clock-rate conversion factor of 372, no extra guard time ($N = 12$), no chaining, XOR checksum procedure, no transmission errors and a case 2 or case 3 command (e.g., READ BINARY or UPDATE BINARY)

Several graphs of data transmission time versus data volume and clock frequency have been generated using the above formulas. They are shown in Figures 15.3 through 15.5. The EEPROM write time versus data volume and page size is shown in Figure 15.6.

15.4 TYPICAL COMMAND PROCESSING TIMES

The tables in this section are based on average times for the successful processing of smart card commands. Measurements were made on various types of smart cards with various operating systems. The listed values are averages. Actual values can differ significantly from the listed values in individual cases, depending on the operating system. All measurements were made with the T = 1 transmission protocol, a clock-rate conversion factor of 372, a clock frequency of 3.5712 MHz, an EEPROM write/erase cycle time of 3.5 ms for a 4-byte page and a software-based DES algorithm with a processing time of 17 ms for each 8 bytes.

Execution times for the commands marked with '⊗' exhibit a strong dependence on the manner in which the command is implemented and the scope of the supported functions. Consequently, processing times for these commands can vary widely.

Table 15.7 Average processing times for the READ BINARY command, as measured with several different smart card operating systems. The time behavior of this command is similar to that of READ RECORD. The values listed in parentheses are the amount of data read. The measurement conditions are described in the text

Command	Processing time excluding data transfer	Processing time including data transfer
READ BINARY (1 byte)	2.02 ms	22.02 ms
READ BINARY (2 bytes)	2.03 ms	23.28 ms
READ BINARY (3 bytes)	2.04 ms	24.54 ms
READ BINARY (4 bytes)	2.04 ms	25.79 ms
READ BINARY (5 bytes)	2.05 ms	27.05 ms
READ BINARY (10 bytes)	2.12 ms	33.37 ms
READ BINARY (20 bytes)	2.23 ms	45.98 ms
READ BINARY (50 bytes)	2.54 ms	83.79 ms
READ BINARY (100 bytes)	2.98 ms	146.73 ms

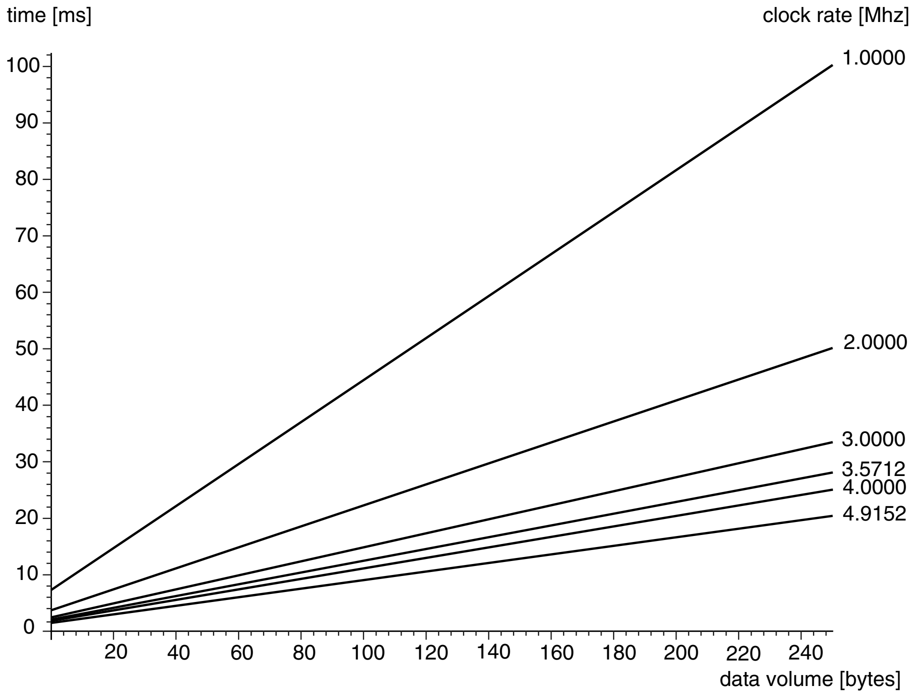


Figure 15.4 Data transfer time as a function of data volume and clock frequency. The values shown are based on the T = 1 data transmission protocol with a clock-rate conversion factor of 64, no extra guard time (N = 12), no chaining, XOR checksum procedure, no transmission errors and a case 2 or case 3 command (e.g., READ BINARY or UPDATE BINARY)

Table 15.8 Average processing times for the UPDATE BINARY command without prior erasing of the affected EEPROM pages, as measured with several different smart card operating systems. The time behavior of this command is similar to that of UPDATE RECORD. The values listed in parentheses are the amount of data read. The page size is 4 bytes and the read/erase time is 3.5 ms. All other measurement conditions are described in the text

Command	Processing time excluding data transfer	Processing time including data transfer
UPDATE BINARY, no erase (1 byte)	6.95 ms	26.95 ms
UPDATE BINARY, no erase (2 bytes)	7.01 ms	28.26 ms
UPDATE BINARY, no erase (3 bytes)	7.03 ms	29.53 ms
UPDATE BINARY, no erase (4 bytes)	7.11 ms	30.86 ms
UPDATE BINARY, no erase (5 bytes)	7.12 ms	32.12 ms
UPDATE BINARY, no erase (10 bytes)	7.33 ms	38.58 ms
UPDATE BINARY, no erase (20 bytes)	12.33 ms	56.08 ms
UPDATE BINARY, no erase (50 bytes)	18.16 ms	99.41 ms
UPDATE BINARY, no erase (100 bytes)	18.81 ms	162.56 ms

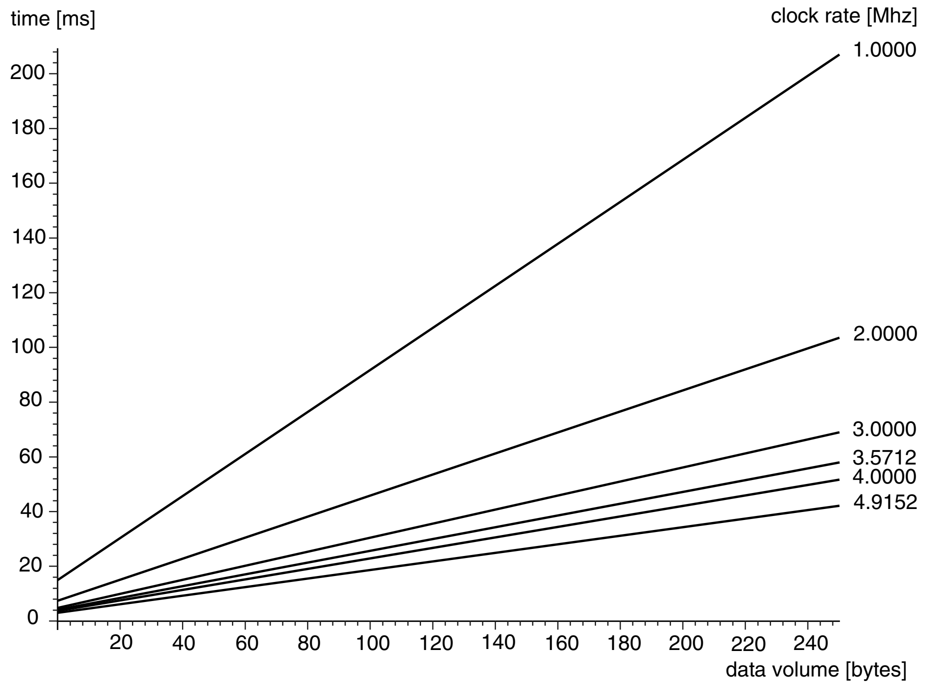


Figure 15.5 Data transfer time as a function of data volume and clock frequency. The values shown are based on the T = 1 data transmission protocol with a clock-rate conversion factor of 31, no extra guard time ($N = 12$), no chaining, XOR checksum procedure, no transmission errors and a case 2 or case 3 command (e.g., READ BINARY or UPDATE BINARY)

Table 15.9 Average processing times for the UPDATE BINARY command with prior erasing of the affected EEPROM pages, as measured with several different smart card operating systems. The time behavior of this command is similar to that of UPDATE RECORD. The values listed in parentheses are the amount of data read. The page size is 4 bytes and the read/erase time is 3.5 ms. All other measurement conditions are described in the text

Command	Processing time excluding data transfer	Processing time including data transfer
UPDATE BINARY, with erase (1 byte)	9.42 ms	29.42 ms
UPDATE BINARY, with erase (2 bytes)	9.51 ms	30,76 ms
UPDATE BINARY, with erase (3 bytes)	9.52 ms	32.02 ms
UPDATE BINARY, with erase (4 bytes)	9.48 ms	33.23 ms
UPDATE BINARY, with erase (5 bytes)	9.62 ms	34.62 ms
UPDATE BINARY, with erase (10 bytes)	9.85 ms	41.10 ms
UPDATE BINARY, with erase (20 bytes)	17.41 ms	61.16 ms
UPDATE BINARY, with erase (50 bytes)	25.87 ms	107.12 ms
UPDATE BINARY, with erase (100 bytes)	35.34 ms	179.09 ms

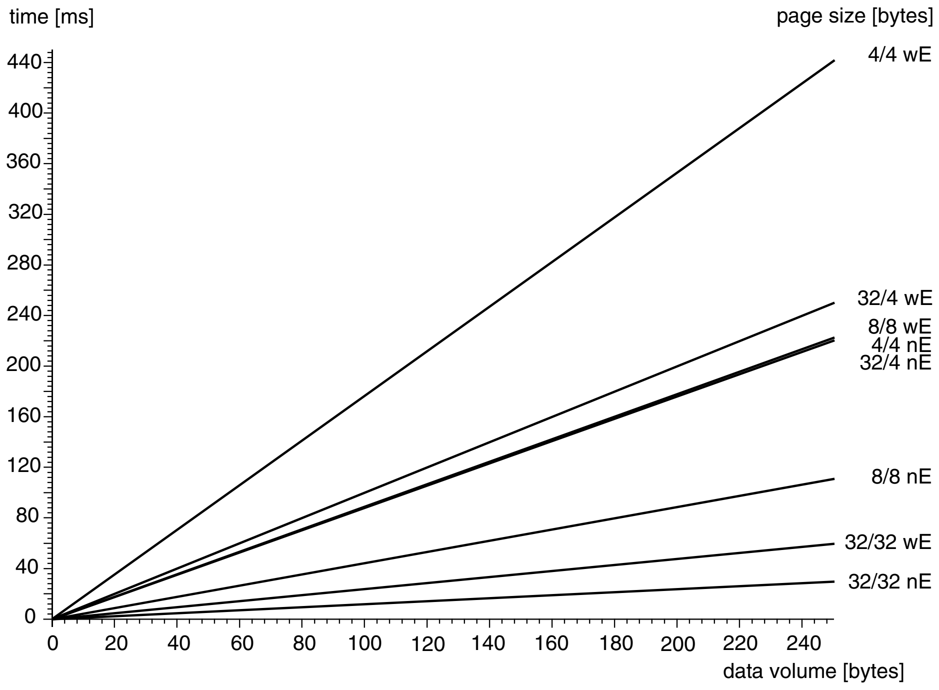


Figure 15.6 EEPROM write time as a function of data volume for various page sizes, independent of the clock frequency. Here it is assumed that no errors occur in the erase/write cycles. In the notation '*p1*/*p2* *x*E', the first numeric value (*p1*) indicates the page size for erasing, the second numeric value (*p2*) indicates the page size for writing, 'wE' indicates that all pages are erased before being written and 'nE' indicates that all pages are written without first being erased

Table 15.10 Average processing times of typical commands, as measured with several different smart card operating systems. The measurement conditions are described in the text

Command	Processing time excluding data transfer	Processing time including data transfer
ASK RANDOM (8-byte random number)	26 ms	55 ms
⊗ CREDIT	175 ms	222 ms
⊗ DEBIT	235 ms	270 ms
EXTERNAL AUTHENTICATE	22 ms	51 ms
GET CARD DATA (8 bytes)	4 ms	33 ms
⊗ INITIALIZE IEP for Load	89 ms	173 ms
⊗ INITIALIZE IEP for Purchase	135 ms	201 ms
INTERNAL AUTHENTICATE	26 ms	65 ms
INVALIDATE	15 ms	34 ms
MUTUAL AUTHENTICATE	95 ms	163 ms
REHABILITATE	15 ms	33 ms
SEEK	3 ms	22 ms
SELECT FILE (with an 8-byte AID)	3 ms	32 ms
SELECT FILE (with a 2-byte FID)	3 ms	24 ms
⊗ VERIFY (8-byte PIN)	27 ms	56 ms

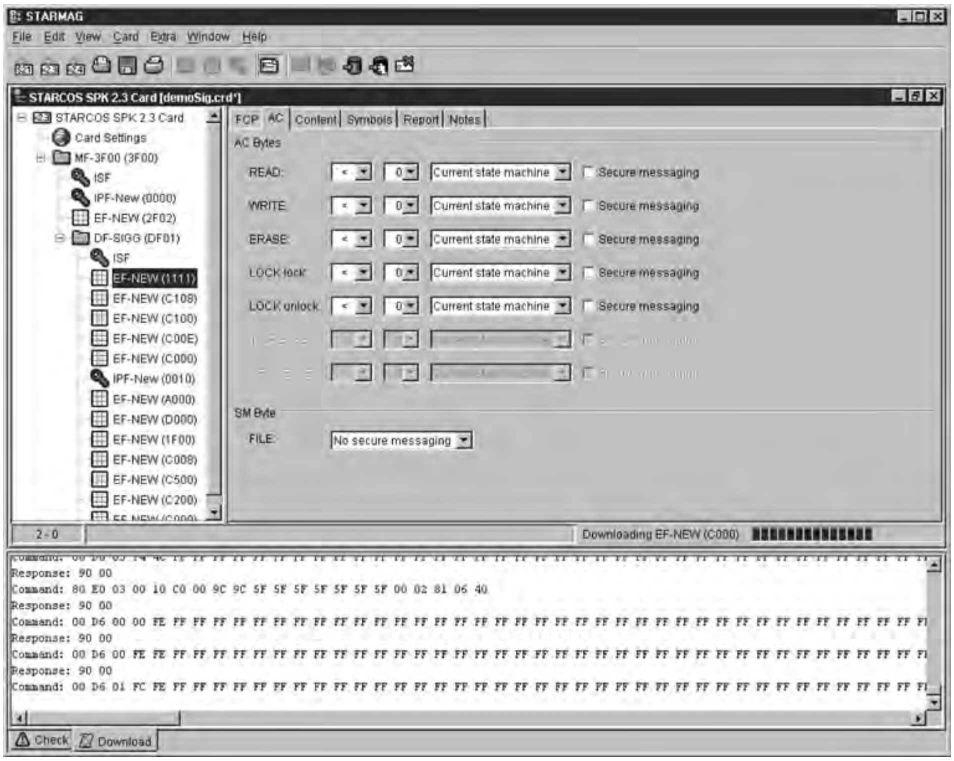
15.5 APPLICATION DEVELOPMENT TOOLS

Several PC-based programs are available for developing smart card applications. They allow users to quickly and easily develop complete applications without having any particular knowledge of the internals of the operating system used in the smart card.

With such tools, the first task is usually to construct a file tree to hold the various applications (i.e., DFs) and their associated EFs. Naturally, it is necessary to specify the file structures and relevant access conditions for the EFs. If the smart card operating system has a state machine for commands, its parameters can also be defined using the graphical user interface of the application generation program. Some application generator programs can also check the consistency of state machines. Since the application needs various data and keys in its EFs, a link to a database can be established after the files have been defined, in order to link the contents of the EFs in the individual cards to data sets in the database.

Once the complete application has been defined using the application generator, various general parameters of the smart card operating system can be configured, such as the transmission protocol and associated divider value. The application can then be experimentally loaded into one or more smart cards with appropriate memory capacity. After several test cards have been produced in this manner, they can be tested in a terminal. If this shows that modifications are necessary, it is possible to delete the application from the smart card and then load a revised version.

Besides these application development tools, smart card simulators are also available. A smart card simulator behaves exactly the same as a normal smart card, but it only consists



AC Bytes

READ: < 0 Current state machine ☐ Secure messaging
WRITE: < 0 Current state machine ☐ Secure messaging
ERASE: < 0 Current state machine ☐ Secure messaging
LOCK lock: < 0 Current state machine ☐ Secure messaging
LOCK unlock: < 0 Current state machine ☐ Secure messaging

SM Byte

FILE: No secure messaging

Downloading EF-NEW (C000)

```
Response: 90 00
Command: 80 E0 03 00 10 C0 00 9C 9C 5F 5F 5F 5F 5F 5F 5F 00 02 81 06 40
Response: 90 00
Command: 00 D6 00 00 FE FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
Response: 90 00
Command: 00 D6 00 FE FE FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
Response: 90 00
Command: 00 D6 01 FC FE FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

Figure 15.7 Screen display of a PC-based smart card application generator. The file tree in the smart card can be seen at the left, with a pane for entering the access conditions for a file at the right. A pane for displaying the data transmitted at the APDU level is located at the bottom (*Source:* Giesecke and Devrient)

of a dummy smart card connected to a PC interface by a cable. Suitable software in the PC simulates the card in real time. Naturally, applications can also be generated and tested in the PC, as described above. However, a PC is always needed to perform the simulation, which frequently creates difficulties due to the size of the equipment.

If all of the tests have been concluded satisfactorily and a larger quantity of cards is needed, the desired number of smart cards can be produced using a standard card production facility or a personalization machine. The application data generated by the PC program (files, commands, states, and so on) form the basis for the completion, initialization and personalization of the smart cards. The turnaround time for producing finished cards thus remains very low, despite the high degree of flexibility provided by this process.

If smart cards that support the downloading of executable program code (such as Java Card smart cards) are used, an extra step for generating the smart card application must be added to the process described above. However, since in practice the actual applications are indistinguishable from applications based on traditional smart card operating systems, even with current Java-Card based operating systems, the remaining steps stay the same.

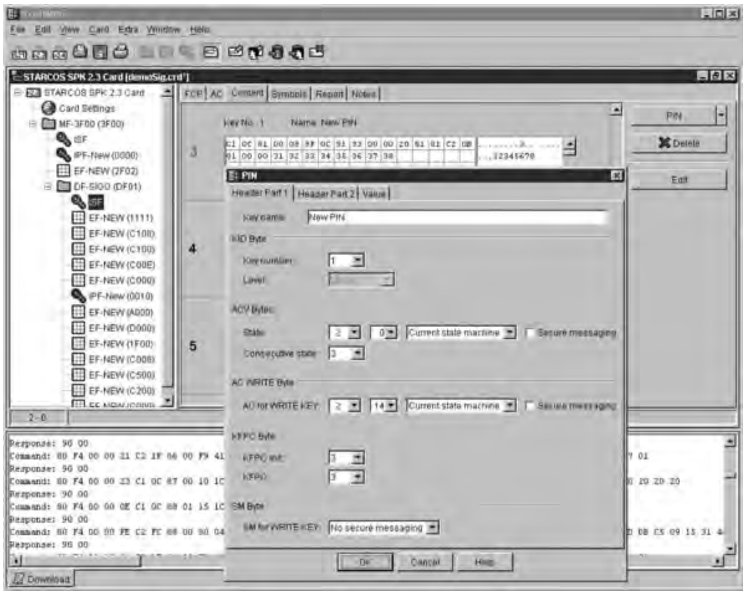


Figure 15.8 Screen display of a PC-based smart card application generator. The file tree of the smart card can be seen at the left. To the right of the file tree, the content of the currently selected file is shown in the background, and the access rules contained in an EF_{ARR} file can be edited in the superimposed window (*Source:* Giesecke and Devrient)

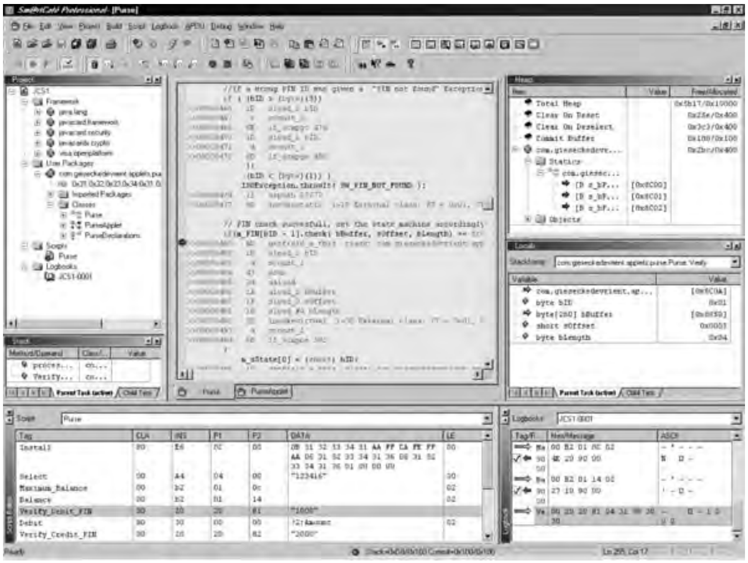
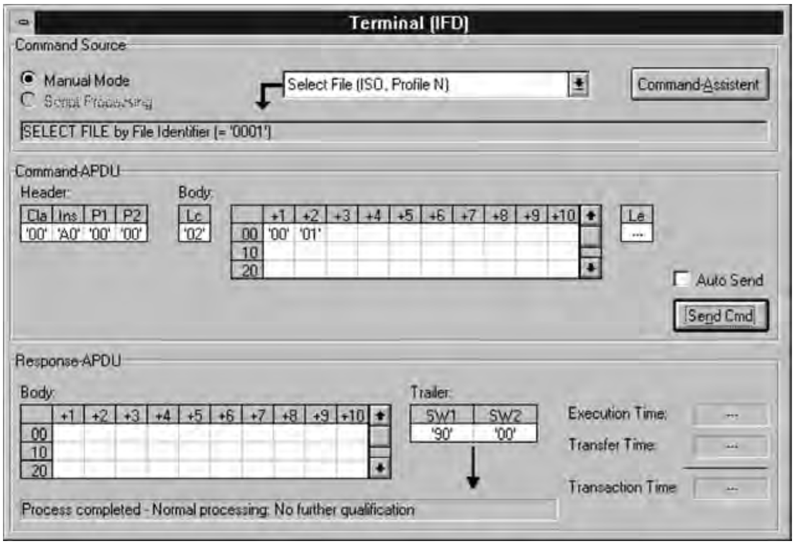


Figure 15.9 Example of a development environment with an integrated Java Card simulator for developing and testing Java programs for smart cards. The classes and methods are shown at the upper left in a tree structure. The Java source code and the bytecode translated from the source code are shown in an adjacent pane to the right. The panes to the far right show the stack, the heap and several variables. Panes showing the script processor and decoding are located at the bottom (*Source:* Giesecke and Devrient)



Terminal (IFD)

Command Source

● Manual Mode

○ Script Processing

Select File (ISO, Profile N)

Command-Assistent

SELECT FILE by File Identifier (= '0001')

Command-APDU

Header:

Cla	Ins	P1	P2
'00'	'A0'	'00'	'00'

Body:

Lc
'02'

	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10
00	'00'	'01'								
10										
20										

Le

☐ Auto Send

Send Cmd

Response-APDU

Body:

	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10
00										
10										
20										

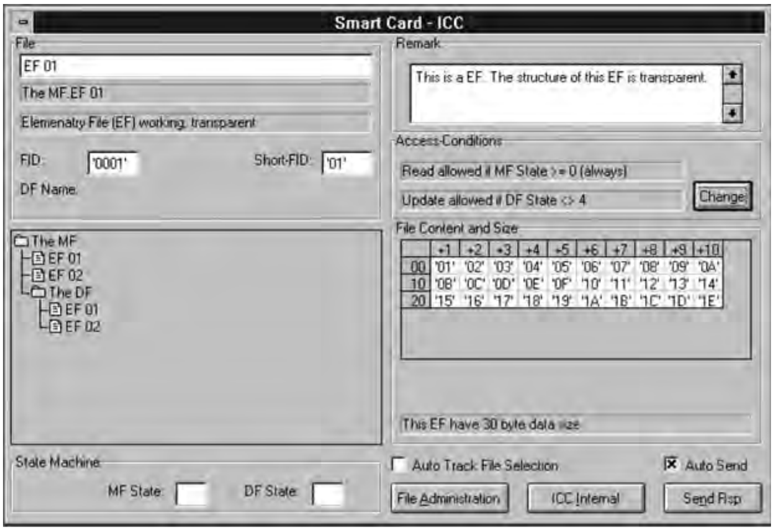
Trailer:

SW1	SW2
'90'	'00'

Execution Time:	---
Transfer Time:	---
Transaction Time:	---

Process completed - Normal processing: No further qualification

Figure 15.10 Terminal window of the Smart Card Simulator program [Rankl]



Smart Card - ICC

File

EF 01

The MF.EF 01

Elemenatry File (EF) working, transparent

FID: '0001' Short-FID: '01'

DF Name:

The MF
- EF 01
- EF 02
- The DF
  - EF 01
  - EF 02

Remark

This is a EF. The structure of this EF is transparent.

Access-Conditions

Read allowed if MF State >= 0 (always)

Update allowed if DF State <> 4

Change

File Content and Size

	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10
00	'01'	'02'	'03'	'04'	'05'	'06'	'07'	'08'	'09'	'0A'
10	'0B'	'0C'	'0D'	'0E'	'0F'	'10'	'11'	'12'	'13'	'14'
20	'15'	'16'	'17'	'18'	'19'	'1A'	'1B'	'1C'	'1D'	'1E'

This EF have 30 byte data size

State Machine

MF State: DF State:

☐ Auto Track File Selection

☒ Auto Send

File Administration ICC Internal Send Rsp

Figure 15.11 Smart card window of the Smart Card Simulator program [Rankl]

15.6 ANALYZING AN UNKNOWN SMART CARD

It is sometimes necessary to analyze an unknown smart card, for example to determine which smart card operating system it uses or which applications it contains. It is usually desirable to keep the necessary expense within appropriate limits, so the equipment available for performing the analysis often consists of nothing more than a computer and a terminal.

The method for analyzing an unknown smart card described here is only one of many possible methods. However, it has frequently proved itself in practice, although we should not fail to mention that considerable experience and extensive knowledge of smart card applications are necessary for successfully evaluating the results. Naturally, the method described here cannot be used to determine secret data or commands that are blocked by state machines. However, experience has shown that this method can at least allow a unknown smart card to be roughly classified.

The procedure is outlined in Figure 15.12 and Listing 15.1. The first step is to determine whether the smart card is actually operational. If this question can be answered in the

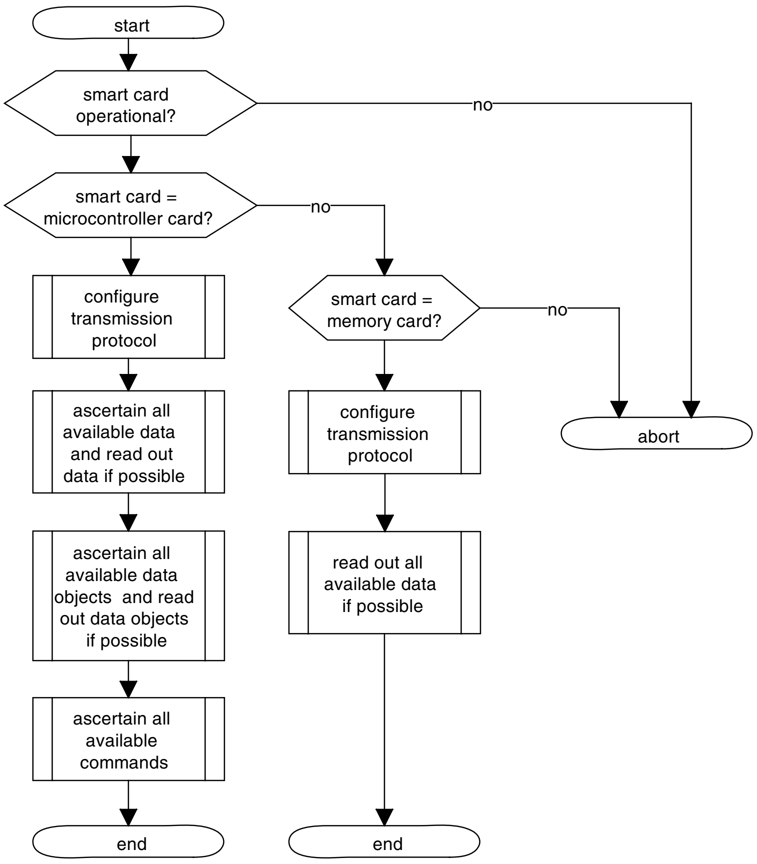

start
smart card operational?
no
smart card = microcontroller card?
no
configure transmission protocol
smart card = memory card?
no
ascertain all available data and read out data if possible
configure transmission protocol
abort
ascertain all available data objects and read out data objects if possible
read out all available data if possible
ascertain all available commands
end
end

Figure 15.12 Basic outline of a possible procedure for analyzing an unknown smart card

Listing 15.1 Pseudocode of a sample procedure for analyzing an unknown contact-type smart card

AnalysisUnknownCard:	**Program for analyzing an unknown smart card**
Perform an activation sequence for a microcontroller smart card, selecting the supply voltage according to ISO/IEC 7816–3 and ISO/IEC 7816–3 Amd.1	A typical activation sequence for a microcontroller smart card with appropriate selection of the supply voltage is a promising way to start the analysis. This sequence is described in Section 3.3.2, 'Supply voltage'.
IF (ATR received) THEN AnalysisSmartCard	If an ATR can be received, this is a microcontroller smart card.
Perform an activation sequence for a memory smart card	Not a microcontroller smart card, so try a memory card activation sequence.
IF (ATR received) THEN AnalysisMemoryCard	If an ATR can be received, this is a memory smart card.
STOP	The smart card could not be identified, since it did not send an ATR, so terminate the analysis.
AnalysisSmartCard:	**The unknown card is a microcontroller smart card**
OUTPUT: received ATR	From the ATR, it may be possible to draw conclusions about the smart card or the applications in the card.
Configure requested transmission protocol	Configure the transmission protocol (T = 0 or T = 1) requested in the ATR, using the appropriate parameters.
// scan all files Select EF_{DIR}	

IF (EF_{DIR} present) THEN (
   read the content of EF_{DIR}
   OUTPUT: content of EF_{DIR})
	If an EF_{DIR} file is present, it will contain data for the applications in the smart card.

Select typical AIDs	In this step, selection attempts are made using typical AIDs. The list of AIDs in Chapter 16 can be used for this purpose. If one or more selection attempts using an AID are successful, the corresponding application(s) has/have been positively identified.
OUTPUT: results of the selection attempts	Output the results of the selection attempts.
Using FIDs, select all EFs in the MF and attempt to read the contents of the selected EFs	It may be possible to draw conclusions about the smart card and its applications from the EF structures, access conditions and EF contents.
OUTPUT: results of the selection attempts, access conditions and read attempts, as well as any file contents read	Output the results of the selection and read attempts.
Using FIDs, select all EFs in each selectable DF and attempt to read the contents of the selected EFs	It may be possible to draw conclusions about the smart card and its applications from the EF structures, access conditions and EF contents. If the DF has a registered AID, the file contents can be interpreted using the corresponding specification (if available).

OUTPUT: results of the selection attempts, access conditions and read attempts, as well as any file contents read	Output the results of the selection and read attempts.
// scan all data objects	
Attempt to read out all data objects using GET DATA	
OUTPUT: results of the read attempts and the contents of any data objects read	It may be possible to draw conclusions about the smart card and its applications from the data objects. If the DF has a registered AID, the contents of the data objects can be interpreted using the corresponding specification (if available).
// scan all commands	
Test all combinations of CLA and INS	This method, which is described in detail in Section 8.2.4.2, ‘Attacks at the logical level’, can be used to determine the commands and secure messaging used by the smart card operating system.
OUTPUT: possible allowed combinations of CLA and INS	Output the results of the command analysis.
STOP	End of the analysis.
AnalysisMemoryCard:	**The unknown smart card is a memory card**
Configure transmission protocol	Try each memory card transmission protocol in turn, and as soon as correct data are received, use the currently configured transmission protocol.
OUTPUT: detected transmission protocol	Output the transmission protocol that has been found.
Read all data from the memory card	
OUTPUT: data content of the memory card	Under certain conditions, conclusions can be drawn about the memory card and its application from its data content.
STOP	End of the analysis.

affirmative, the next step is to determine whether it is a memory card or a microcontroller card. Following this, for both types of card an attempt is made to configure the appropriate transmission protocol and read data present in the memory, files or data objects. Based on this information, an attempt is then made to manually determine which applications are present in the smart card. In the case of a microcontroller card, it is often possible to also determine which commands are supported, and thereby draw conclusions about the smart card operating system present in the card.

15.7 LIFE-CYCLE MODELS AND PROCESS MATURITY

There are various methods that can be used to produce software. All of them can be generally described using life-cycle models, thus allowing them to be used for a variety of software development projects. Life-cycle models are also often referred to as ‘process models’. A life-cycle model describes, in general terms, the activities to be performed, the sequence of these activities, and the responsibilities and competences. Incidentally, most types of software development life-cycle model can also be used to direct the realization of nearly all types of creative activities and activities involving the development of something new.

In the 'trivial' life-cycle model for software development, the programmer sits down in front of his or her computer, after having received brief oral instruction regarding the task to be performed, and generates a program, which he or she then modifies until most of the objectionable errors have been eliminated and the customer for the software is more or less satisfied. Surprisingly enough, this simple method can be found not only in stories of the early days of software development, but also in many modern-day forms, in both small and large development projects. It is unquestionably possible to generate innovative and competitive programs with this 'garage company' mentality. However, with this life-cycle model the results with regard to compliance with deadlines, development costs and software quality can only be predicted within very broad limits. In the case of software development projects involving complicated tasks and several developers, the resulting complexity of the project can become so large that either the available budget and schedule are vastly exceeded or the entire project must be cancelled before being completed. Consequently, life-cycle models are used in professional software development to provide a development framework, in order to make the three classical factors (schedule, cost and quality) as accurately predictable as possible.

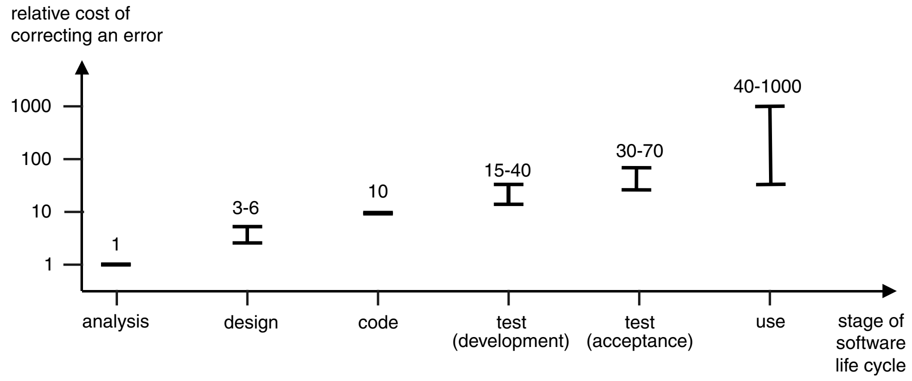


Figure 15.13 The cost of correcting an error as a function of the life-cycle stage of the software relative to the analysis stage. This diagram is based on a publication by Boehm [Boehm 81]

Industrial production processes – which naturally also include the development of software – have four characteristic features, as follows:

1. The development process is divided into stages.
2. Each development stage produces results that form the basis for the following stage.
3. The results of a particular stage are checked before the next stage is started.
4. The results of the individual stages are abstract representations of the product that become increasingly concrete from one stage to the next, with the actual product emerging from the final stage.

These characteristics actually originate from classical mechanical engineering, but in principle they are equally applicable to competent software engineering. They form the basis for the life-cycle models described here.

Developing software, and incidentally most other development activities as well, requires four general activities. The objective of the first activity is to answer the cardinal question, 'What has to be developed?' This means that the objective of the development must be defined as accurately and unambiguously as possible. Here 'unambiguously' means that the highly popular subjunctive terms 'should', 'could', and 'ought to' are not allowed to be used in the definition. Practical experience has shown that documents generated during this activity should contain as little prose as possible. Tabular listings and diagrams are ideal, since they leave little room for imprecision and ambiguity. The document resulting from this action is called a requirements specification document, or sometimes a user requirements specification. The activity of generating this document is called analysis or requirements analysis.

The requirements specification document forms the basis for all further development activities. Completeness and clarity are thus fundamentally important attributes of this document, which is not allowed to be altered after its final review. In some cases, changing even a single word in the requirements specification could have large consequences for all subsequent development steps.

Here we can use the requirements for the UMTS mobile telecommunication system as an example. The original requirements specified the exclusive use of asymmetric cryptographic algorithms. After extensive discussion, at a relatively late point in time the letter 'a' was deleted from the word 'asymmetric'. As a result, specifications based on these requirements, as well as a number of already existing implementations, had to be completely revised. This extreme example is intended to serve as a persuasive indication of the importance of the requirements specification for all subsequent development steps.

The second cardinal question in the development process is, 'How is the development to be done?' There are two aspects to this 'how'. The first aspect relates to organizational constraints, while the second aspect relates to the structure and architecture of the software to be developed. Answering this question involves fully and unambiguously describing all of the functions and data of the software to be produced. The principlal objectives here are to reduce the complexity of the entire development project to a manageable level, to ensure the modifiability and reusability of the developed components, and as necessary to make preparations for partitioning the implementation work. In professional software development, answering the 'how' question is the most costly component of the entire development process. This activity is called design, and the result of the design activity is called the design specification document or the software specification. In this book, this activity is consistently referred to as 'design', and the resulting document is referred to as the 'software specification'.

Like the requirements specification, the software specification must of course be unambiguous and not leave any room for interpretation. Ideally, it should be possible to give the finished and reviewed software specification to persons who have not been involved in the design process and have the software be correctly generated on the basis of this document alone, without any requests for clarification.

After the questions of what is to be developed, how it is to be developed and how it is to be constructed have been answered, implementation can begin. This is where software developers with a 'garage company' mentality actually start the whole development process. If the developer can work from a complete software specification that is not subject to interpretation, the amount of effort that must be expended on implementation is significantly less than the effort required for the two previous stages. With a proper software specification, the software can be simply programmed module by module, with debugging being performed by the person who does the programming. The result of this activity is the software modules, which generally should be free of trivial errors.

The developer documentation is generated in parallel with the programming. This consists of all of the documents produced by the programmers during their development activities. Here practical experience has convincingly shown that software should be documented directly in the source code, in part because this makes it significantly easier to find the documentation, but primarily because with this approach, the least amount of information is lost over time. Generating developer documentation is also supported by suitable tools, such as Javadoc, which can automatically generate adequate developer documentation from the information in the source code. In general, appropriate guidelines must also be followed with regard to the structure and documentation of the source code.

Following implementation, the module that has been generated, or several modules together, are tested together with the portion of the program that has already been produced up to that point. This is naturally called the test stage. Ideally, testers should be able to perform their tasks with the least possible amount of dependence on the programmers, and the programmers should receive only ‘OK’ or ‘not OK’ as a result. The advantage of this approach is that since the programmers do not know exactly what will be tested, they are compelled to debug their programs as completely as possible. This yields a relatively good ‘four eyes’ situation, with the result that significantly more errors are found than when programmers have a detailed knowledge of the tests their programs will be exposed to and thus ‘polish’ their code to successfully pass these tests.

The activities of designing and executing tests are governed by their own methodologies, which are extensively described in Chapter 9.

In large systems having a variety of components and multiple component suppliers, compatibility testing is performed after the software has successfully passed the development tests. After the compatibility tests have been successfully completed, there is usually a formal acceptance of the software by the customer. However, this acceptance may be based on previously performed tests and inspections and thus has a purely formal character. Following acceptance, the software is released and can be used.

After the software has been delivered and while it is being used, it may be necessary to expend effort on maintenance and updating, depending on the application. This consists of eliminating unacceptable errors and making relatively small functional modifications and upgrades to the existing software. An almost inevitable result of software maintenance is that the structure of the software tends to become increasingly vague with each new revision, even if the software was originally well structured. In many cases, the associated documentation or underlying software specification is not updated when maintenance is performed, leading to discrepancies between the abstract representation in the software specification and the actual product. There are two remedies to this situation. In the case of relatively small and simple programs, maintenance is performed without expending a lot of additional effort, but it is accompanied by planning for a new, completely revised version (‘refactoring’). However, this approach is only permissible for relatively small programs and prototypes. In the case of larger programs, extreme care must be taken when performing maintenance, which means that all software specifications and documents must be suitably updated. Modifications to source code must be performed equally carefully. It may even be necessary to rewrite relatively large portions of the source code in order to ensure that the software continues to have a clearly structured structure.

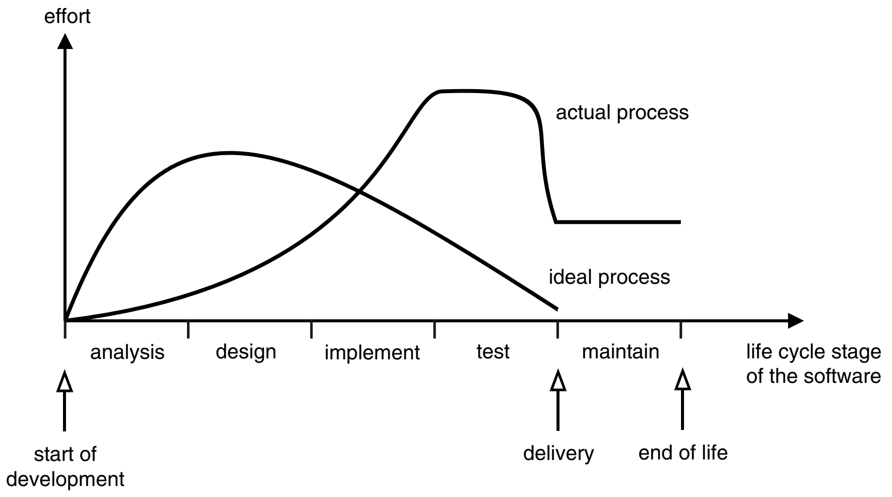


Figure 15.14 Comparison of development effort versus software life-cycle stage for an ideal development process and many actual development projects

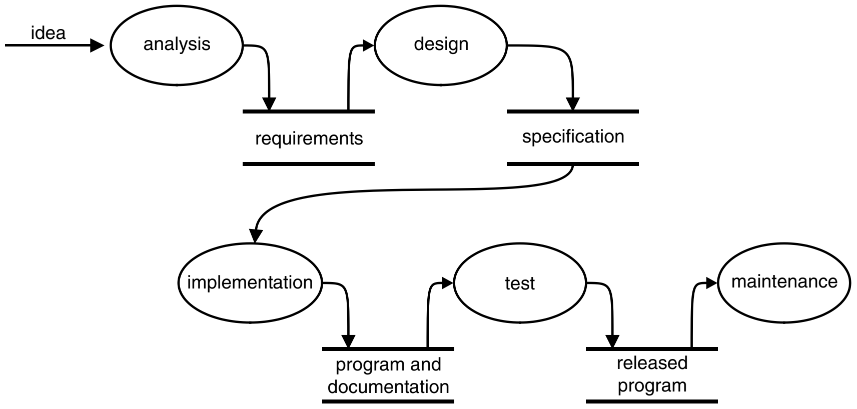


Figure 15.15 Data flow diagram showing the essential activities and documents of a typical software development project for smart cards

15.7.1 Life-cycle models

The life-cycle models described below show all of the activities related to the development process in a uniform, graphical manner. An IT-compliant notation similar to data flow diagrams has been used for this graphic representation, in order to make the life-cycle models readily understandable and allow them to be easily compared with each other. Only models that represent pure forms are shown, rather than those representing mixed forms. Pure forms have the advantage that both the positive and the negative features of the model can be clearly seen.

Furthermore, only those models that can reasonably be used in the development of software for smart cards have been included in the selection. The description of each life-cycle model contains enough information to allow its basic features to be understood, applied and possibly used in certain cases.

There are many other life-cycle models for software development besides the ones described here. However, they are often mixed forms or specialized versions of the described models. Some clients define their own life-cycle models, particularly in areas where security and reliability are particularly critical, such as military applications, nuclear engineering and aeronautical engineering. More extensive information on this subject can be found in [Blazert 98], among other sources.

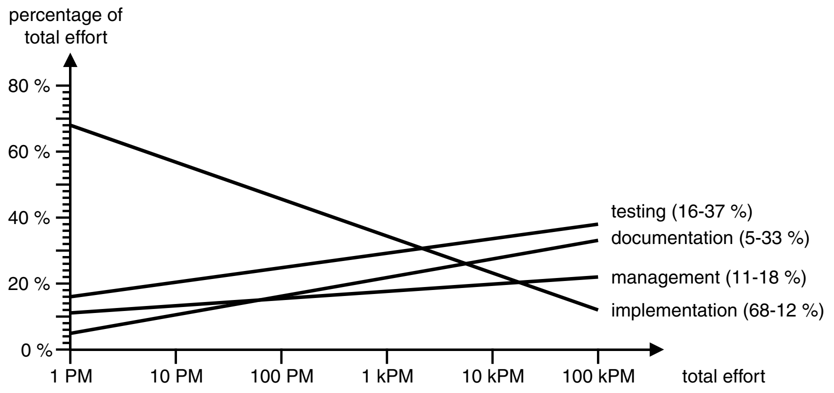


Figure 15.16 Comparison of the distribution of effort in a development project as a proportion of total effort. The sum of the individual efforts is always 100 %. The abbreviation 'PM' stands for 'person-month'. These data are based on a publication by C. Jones [Jones 91]

15.7.1.1 The waterfall model

The principle of the waterfall model was published in 1956 by Benington, and the addition of integrated retrograde jumps was published by Royce in 1970. The name of this model, from Boehm in 1981, arises from the stepwise arrangement of the activities, which resembles a waterfall. It was the first life-cycle model, and it represents a significant advance over the trivial 'brain to keyboard' model.

The essential features of the waterfall model are a sequential development process and a straight-line, top-down procedure. Each of the activities shown in Figure 15.17 is performed completely and in prescribed sequence, although it is also allowed regress to the previous activity. This life-cycle model is strongly document-driven, which means that specific documents are generated during each activity and are used in subsequent activities on completion of the activity in which they are generated. This completion is usually marked by a review of the documentation. The waterfall model allows customer involvement only at the beginning, during the definition stage. After this, only the developers are involved in the process until the ultimate release of the software.

The waterfall model is a simple life-cycle model that requires only a small amount of coordination effort. Using this model yields a clearly defined and readily understood development

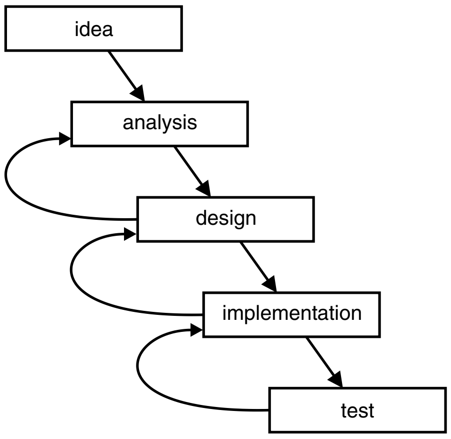


Figure 15.17 Schematic representation of the waterfall model in a form adapted to the development of software for smart cards

process. However, if the objectives are not fully defined or not fully known, the prescribed sequential process of the activities can easily lead to problems, since each activity in the sequence can only be started after the previous activity has been fully completed. The waterfall model thus has no provision for activities such as building simple prototypes or conducting simple tests in the definition or design stages, in order to explore implementation options. The fact that the customer is only involved at the beginning, during the definition phase, is a related drawback.

The waterfall model is well suited to development projects that do not explore unknown technical territory and have previously been carried out at least once in a similar form. Such development projects generally do not produce any surprises, since most of the general technical and organizational premises are well known at the start. An example from the realm of smart cards is porting a smart card operating system from one type of chip to another type. In this case, all that has to be done is to adapt the software to the technical specifications and features of the microcontroller in question, and to the extent that the code is programmed in assembler, recode the relevant machine instructions.

In summary, it can be said that the waterfall model is primarily suitable for non-creative development activities. As soon as unexplored technical territory is entered and innovative, creative developments are necessary, the waterfall model should not be used in any form, since it is not suitable for such projects. If it is nevertheless used, it can lead to massive problems in implementation, since it does not allow critical items to be adequately studied in advance during the analysis and design stages.

15.7.1.2 The V model

The V model is essentially a waterfall model with integrated quality assurance. It takes its name from the typical V-shaped diagram used to portray the required activities. Like the waterfall model, the V model has a sequential flow of activities. Each of the individual development activities (analysis, design and implementation) has a corresponding test activity. If necessary, it is possible to jump back to the previous activity. The V model was originally developed for embedded systems – which include smart card microcontrollers – and is a relatively sophisticated life-cycle model. The large amount of effort that it requires, particularly with regard to documentation, is balanced by the very high quality of the developed software. Consequently, the V model is primarily used in relatively large developments where high quality is required, such as smart card operating systems and military applications. A detailed presentation of the V model can be found in [Dröschel 99].

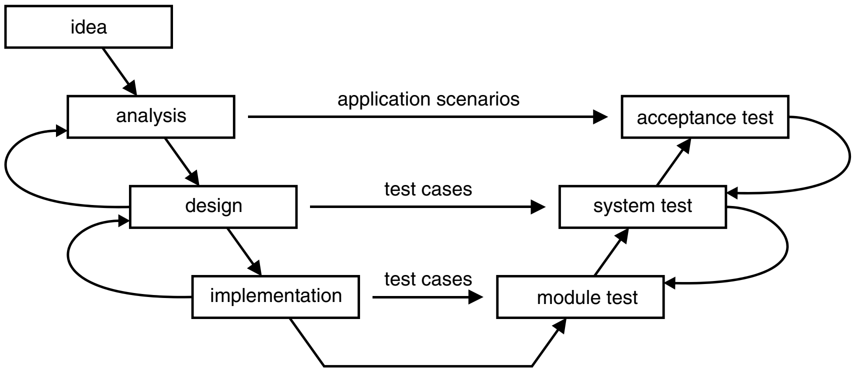


Figure 15.18 Schematic representation of the V model in a form adapted to the development of software for smart cards

The V model is very suitable for developing software for smart cards when it is important to translate prescribed specifications into program code at a high level of quality. One example would be implementing a GSM 11.11 application in an existing smart card operating system. In this case, the individual commands and the file system can be programmed according to the detailed GSM 11.11 interface specification without a large amount of creative effort. In this case, the most important consideration is complete compliance with the specifications in the GSM 11.11 document.

The V model should not be used for completely new developments, since like the waterfall model, it does not provide for iterative development steps or customer involvement after the analysis stage. In summary, the V model is ideal for developments that do not involve exploring unknown technical territory and in which a low level of errors and compliance with specifications have the highest priority.

15.7.1.3 The prototyping model

The waterfall model and the V model both envisage a clearly demarcated series of activities following each other in a defined sequence. However, this leads to problems in many software development projects, since it leaves little room for creative solutions resulting from experiments. The prototyping model introduces this additional degree of freedom into the life-cycle model.

The purpose of a software prototype is to demonstrate only certain parts of an entire software system. In the 'horizontal' version, only certain layers of a software system are implemented. In the case of a smart card operating system, an example of a horizontal prototype would be a complete implementation of all of the transmission protocols, but without processing the actual commands, which would only return dummy responses. This prototype could be used to fully test communication between the terminal and the card. Such a prototype might be used to experimentally demonstrate that a high data transmission rate can be achieved using software alone, without hardware support provided by a universal asynchronous receiver/transmitter (UART).

Logically enough, the opposite of a horizontal prototype is a 'vertical' prototype, in which only certain parts of the software are implemented across all layers. With reference to the previous example, such a prototype could be a complete implementation of a single command, such as INTERNAL AUTHENTICATE, together with a single transmission protocol, possibly only in rudimentary form. This prototype could be used to thoroughly measure the timing behavior of smart card authentication, including all data communications. Incidentally, this approach is the only possible way to perform such a task, since accurate and reliable timing data can only be obtained by experiment, not by analysis.

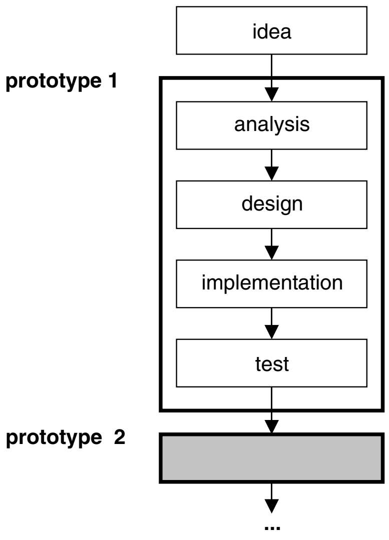


Figure 15.19 Schematic representation of the prototyping model in a form adapted to the development of software for smart cards

A prototype used in software development demonstrates specific features of the software in practical use, in order to allow alternative solution options to be experimentally tested in advance of the overall implementation. Based on the results of prototype testing, the design is then completed or the prototype is further refined, so that it can be used as part of the software to be developed.

Since critical requirements can be verified in advance using prototypes, the prototyping life-cycle model is well suited to development projects whose objectives are not precisely specified. Another positive feature of this life-cycle model is that the customer can review and approve the prototypes and then refine his – possibly scanty – requirements, which leads to a more harmonious relationship with the developers. The combination of prototypes that can be used for experiments and an improved relationship with the customer leads to a reduction of risks in the development process.

Besides these benefits, there are also several drawbacks that should not be underestimated. Progressing experimentally from prototype to prototype is costly in terms of both time and money, since it is generally necessary to pass through several development stages a number of times while searching for the proper solution. However, the greatest hazard in software development using the prototyping model is not so much technical as organizational. There is a risk that due to schedule or cost pressure, a prototype will be declared to be a finished product and thus delivered much too early. This leads to software that lacks the required functionality and is not fully tested, which means it probably still contains a large number of errors. With software that is generated in this manner, it also frequently happens that the documentation is incomplete or, in the worst case, non-existent.

As an aside, it can be noted that the prototyping model corresponds to the software development method used by many hobbyists. They start with a relatively small central program, usually without any formal definition or design, and extend it step by step while testing each extension, thus developing an increasingly larger program. Many programs, particularly those in the public domain and freeware realm, unfortunately quite clearly demonstrate the result of such a process: an ambitious core functionality has been completely programmed and tested, but the auxiliary functionality is only partially present, and in many cases the documentation is limited to the original core functions.

With respect to developing software for smart cards, the prototyping model is very suitable for studies and feasibility analyses in the areas of transmission protocols, cryptographic algorithms and file systems. However, the resulting prototype should not be incorporated 'as is' into the final product. Instead, it must be brought up to the same level of quality as the rest of the developed software. It is certainly possible for large blocks of prototype source code to be incorporated into the final product, as long as they are adapted to meet the requirements of the overall implementation.

Prototype development should not be limited to only the risky components of the program code in an effort to minimize development expenses and effort. In order to minimize the risk of having a prototype being declared to be a product, experienced developers generally implement only subsystems as prototypes, never complete systems.

15.7.1.4 The evolutionary and incremental models

The life-cycle models described up to this point require relatively exact requirements analyses and specifications. However, in some cases the requirements and specifications cannot be fully defined or can only be determined in very vague terms, such as when a totally new software concept is involved. In addition, actual use of the software gives rise to new requirements and desires on the part of users. The evolutionary and incremental life-cycle models allow these types of requirements to be incorporated into the software development process.

With both of these models, the full breadth and depth of the software is developed in a generational sequence, which means that both models are code-driven. After each generation is released, user experience is analyzed, and the results of this analysis enter into the development of the following generation as general requirements. The difference between the evolutionary and incremental models is that in the evolutionary model, a complete requirements analysis is performed for each new generation, while in the incremental model, a complete requirements analysis is only performed at the beginning of the overall development process. Both models are well suited to use in development projects where some of the requirements are unknown, since the functionality of the software can be adapted to the actual requirements step by step in the course of successive generations. Both models are program-code-driven, which means that the product is used in actual practice between successive generations, which differs from the prototyping model.

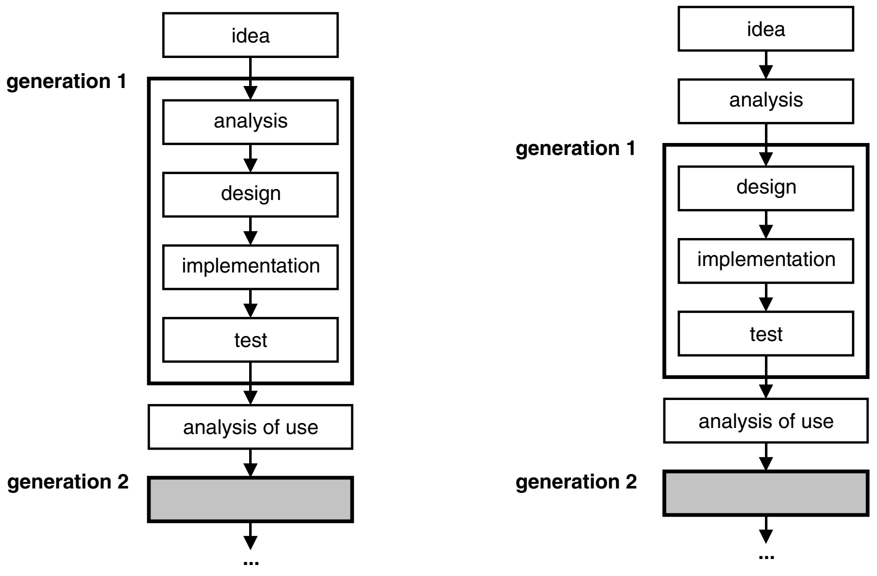


Figure 15.20 Left: schematic representation of the evolutionary process model in a form adapted to the development of software for smart cards. Right: comparative schematic representation of the incremental process model

Both the evolutionary model and the incremental model lead to a minimization of development risks. They also allow the direction of development to be steered within certain limits during the development process. However, these benefits come at the price of higher development costs and the risk that extensive modifications to the software architecture may be necessary in later generations, due to incomplete analysis at the beginning of the development process. This problem can become particularly severe with the evolutionary model, since a new analysis must be performed for each new generation. In the incremental model, the complete development is analyzed at the beginning, following which only differences with respect to the initial analysis are generated between successive generations. Consequently, changes from one generation to the next should not have any fundamental effect on the software architecture.

The evolutionary and incremental models can be used to advantage with completely new developments of a research nature. They allow a high degree of requirements coverage to be achieved in the course of several generations, which preferably should follow each other relatively quickly. Consequently, these two life-cycle models are usually used for implementing new smart card operating systems. Another application area is the development of new, loosely defined smart card applications that are interactively adapted to meet their ultimate requirements over the course of several generations.

15.7.1.5 The spiral model

The spiral model is a metamodel, which means that it is a model that can incorporate any of the previously described life-cycle models in order to use the most suitable model for each version of a software product. The spiral model takes its name from the spiral shape of the diagram used to represent this method, in which the area inside the spiral corresponds to the sum of the development costs.

The development process is divided into four stages in the spiral model. The objectives for the software to be developed are defined in the first stage, and in the next stage, possible options for attaining these objectives are determined using risk analysis. In the third stage, the software is developed using the most suitable life-cycle model. If the entire development is not thereby completed, the following cycle is planned in the fourth stage, based on the results of the previous stages. After this, the first stage is repeated again with the setting of new objectives.

Fundamentally, the spiral model is primarily suitable for large development projects, since it requires a relatively large amount of coordination effort due to its complex sequence. However, it has the advantage of being very flexibly adaptable to a variety of different tasks, and it is highly suitable for developing software over the course of several generations.

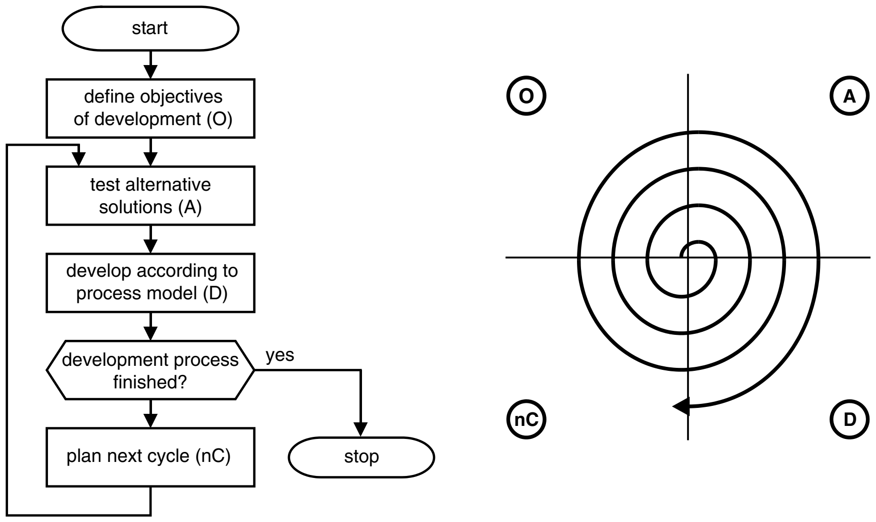

start
define objectives of development (O)
test alternative solutions (A)
develop according to process model (D)
development process finished?
yes
plan next cycle (nC)
stop
O
A
nC
D

Figure 15.21 Schematic representation of the spiral model in a form adapted to the development of software for smart cards. The area of the spiral approximately corresponds to the amount of development effort

For example, a card operating system can be developed using the spiral model as follows. First, versions 1, 2 and 3 of the operating system are developed using the evolutionary model. This produces three research versions that are used for studies and experimental purposes, during which the operating system is incrementally refined and adapted to meet the necessary requirements. The next version (version 4) is then developed as a deliverable customer version using the V model. The V model is chosen here because one of its characteristics is very high software quality. Customer versions of smart card software are sometimes formally evaluated (using ITSEC or CC), and most of the documents required for this purpose must be generated if the V model is used properly. In our example, the next software development task is porting the operating system to a different microcontroller. Since this involves only very small risks and has only a small creative element, it is preferably performed using the waterfall model. This example clearly shows how the spiral model can be used as a metamodel in which suitable life-cycle models can be used to best advantage for developing each version of a smart card operating system.

To take another example, we use the waterfall model to produce each new edition of the *Smart Card Handbook*. We first generate an analysis (the outline), which defines which topics must be revised or expanded and which completely new chapters are to be produced. Next, in collaboration with the publisher we decide how the changes and additions are to be implemented. This is followed by a period in which the text is written, the illustrations are generated and the book is laid out. After this, the publisher's proofreaders and production staff check whether everything has been done properly. The individual editions of the book are in turn embedded in an evolutionary process, since a complete analysis is performed for each new edition.

15.7.2 Process maturity

Several different methods are used within organizations to assess the quality of software development processes. Presently, the best-known method involves assessing process maturity using the Capability Maturity Model (CMM). The Software Engineering Institute (SEI), an American organization, began working on this model in late 1986 and published the first version in 1991. The currently valid version is Version 1.1, which dates from 1993 [CMM 93]. Besides CMM, there is also a relatively new method for assessing process maturity based on the ISO 15 504 standard, but up to now it has not been widely used in practice. Consequently, here we provide a general summary of this subject based on the CMM.

There are five levels of process maturity in the CMM scheme, with level 1 corresponding to the lowest process quality and level 5 corresponding to the highest process quality. At the first level, which is called the 'initial level', the person or organization doing the development is assigned a task and ultimately produces a product, using a process characterized by poorly predictable expenditures of time, uncertain costs and equally unpredictable quality. The individual steps in the development processes are not defined, and the developers behave more like artists than skilled or industrial workers. The product to be developed is sometimes produced just within the allowable limits of effort, schedule and quality, thanks to the heroic efforts of individual persons, using a development process that can unquestionably be described as chaotic.

At level 2, the 'repeatable' level, individual activities are defined within a specific framework, such as analysis, design, implementation and testing. The details of what happens within

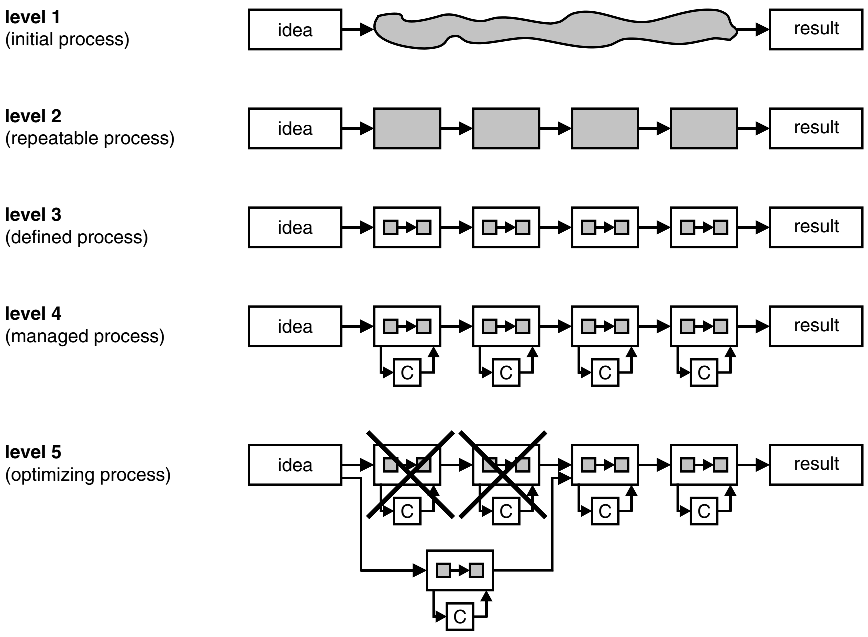


Figure 15.22 Schematic representation of the five CMM levels of process maturity. The blocks marked 'C' represent control processes

the contexts of these general activities are not specified. At the next level, the 'defined' level, the details of the individual activities are indeed defined. At this level, there is a sufficient degree of agreement on the content of the activities in the development process that they can be repeated or reconstructed at any time. As a result, the quality of the development process is significantly higher, and the range of variance in compliance with schedule and cost parameters during the development project is smaller than at the previous levels. At level 3, the process is also characterized by being independent of specific individuals, since all of the required activities are defined. By contrast, there is a high degree of dependence on individual persons at levels 1 and 2. At level 3, the quality of the developed software is still not predictable, since the process is rigidly defined in advance and cannot be flexibly adapted to the various requirements of the development process.

Level 4, which is called the 'managed' level, incorporates control loops within the individual activities, thus allowing the development process to be adapted to the varying requirements of development projects. The individual process stages are controlled using measured data acquired during the development process on the basis of software metrics.

The highest quality level is level 5, the 'optimizing' level. It involves applying controls not only to individual activities, but also across several activities. These activities can also be replaced as necessary by new, improved activities, allowing the overall development process to be constantly adapted to optimally satisfy varying requirements. Compliance with the three cardinal criteria – schedule, cost and quality – can be achieved within suitable limits by using process parameters recorded during the development process.

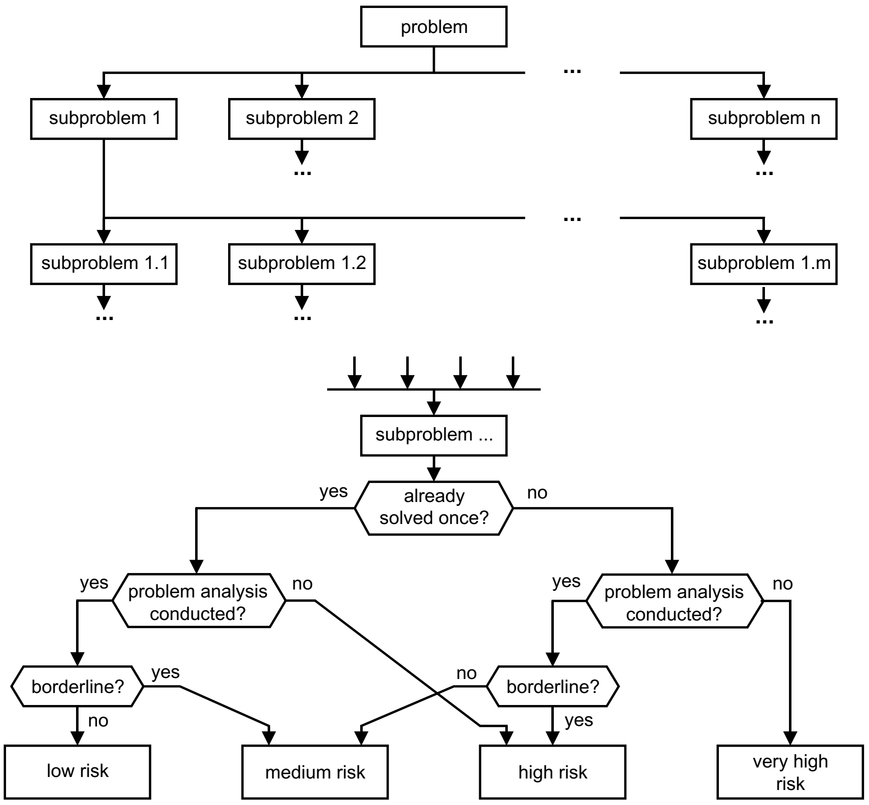


Figure 15.23 Flow chart for dividing and classifying a problem into risk classes. This flow chart forms a highly suitable basis for generating time and risk assessments of development projects. Ideally, subproblems should be broken down to the level where each one requires around one week to handle, with time and risk assessments being generated for each of these subproblems

With its five levels of maturity, the CMM is naturally a highly abstract representation of development processes. An existing process is assessed by having the persons involved in the process anonymously answer a list of approximately 100 yes/no questions. The objective of this is to obtain truthful responses, and thus an objective assessment of the development process within an organization.

In most companies, process quality is at level 1 or 2. Only a handful of companies throughout the world, most of which are active in typical high-end software areas such as aerospace, nuclear technology and military technology, can boast level 5 quality for all of their development activities.

Of course, the quality of software does not depend on process quality alone, and it is certainly possible for software produced using a poor-quality development process to be highly innovative and extremely successful commercially. However, the likelihood of meeting targets for cost, quality and completion deadlines decreases as the quality of the development process drops. In the worst case, it may be necessary to prematurely terminate a development project because the development budget has been exhausted, the completion date is no longer acceptable or the majority of the software will never function in a satisfactory manner, due to its high level of complexity. These risks decrease as the quality level in the development process increases.

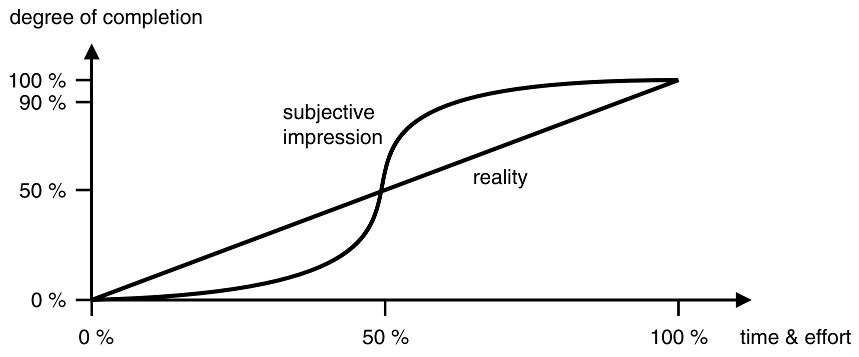


Figure 15.24 Comparison of the actual (measured) course of software development and the course of the development process as subjectively experienced by the software developer. This diagram assumes a constant total development effort

15.8 THE COURSE OF A SMART CARD PROJECT

The course of a smart card project is shown in Figure 15.25. Currently, card manufacturers also develop the associated microcontroller software. This means that the first phase (A) and last phase (E) of the project are performed by the same company. Phases B and C are performed by a semiconductor manufacturer. The dice can be built into the modules either directly by the semiconductor manufacturer or by the card manufacturer. Phase E, which includes initialization, personalization and related activities, is always fully performed by the card manufacturer. The card manufacturer also usually manages the smart card project, with the rest of companies more or less acting as subcontractors.

This brief description says very little about the time required for the individual production processes. However, this should not be underestimated, since several different companies work together to produce smart cards, and the elapsed time for some processes can be many weeks. Typical times for the completion time of the most important production steps are also shown in Figure 15.25. This example is based on the following assumptions:

- 50,000 cards are to be produced
- a new operating system must be generated, based largely on existing libraries
- all companies involved have mid-range production capacities
- each production process can start immediately after the necessary parts are received.

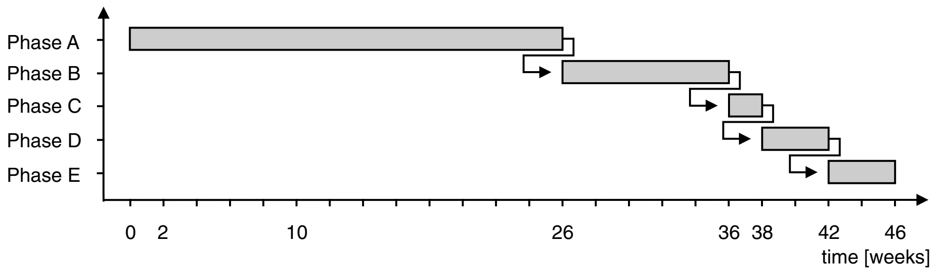


Figure 15.25 Gantt chart showing the phases of a typical smart card project

Phase A: mask generation	6 months
Phase B: semiconductor fabrication	10 weeks
Phase C: module production	2 weeks
Phase D: card body production	4 weeks
Phase E: initialization and personalization	4 weeks

15.9 DESIGN EXAMPLES FOR SMART CARD APPLICATIONS

There are basically two different ways to implement applications in smart cards. The first type of implementation is based on files with defined access conditions. With such an implementation, the necessary commands generally comply with the usual smart card standards, such as ISO/IEC 7816-4. The other type of implementation is based on program code executed in the smart card. There are several different variants of this type of implementation. The program code can be directly executed by the processor in the smart card (native code), or it can be interpreted. In the case of interpreted code, a distinction can be made between microbrowsers (such as XML derivatives) and virtual machines (such as Java). Detailed descriptions of the various options, including their advantages and disadvantages, can be found in Chapter 5, ‘Smart Card Operating Systems’.

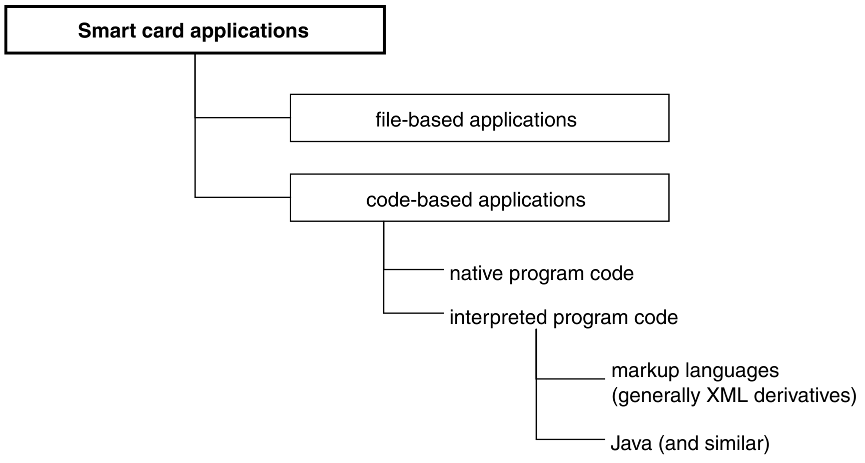


Figure 15.26 Classification of the basic options for implementing smart card applications

The following examples illustrate three typical smart card applications. These are mid-range electronic data-processing applications that do not require elaborate system designs. They might typically be used by medium-sized companies. The background system employed could be a PC located a secure environment, which means that the acquisition and operating costs would both be at the low end of the scale.

These simple examples clearly illustrate how a typical smart card application is constructed. The construction is explained step by step, gradually building up to the finished application in the smart card. In each case, we give only cursory attention to the terminals and the background system, but these aspects of the system can be deduced from the provided information.

All of these examples are based on the principle of distributing data among many individual, separate systems. This contrasts with commonly used centralized mainframe solutions, in which all of the information is stored in one place. If such an approach is translated into a smart card application, the result is that the card is only a sort of proof of identification, with all of the information being held in an omniscient background system. Whenever an application constructed this way must be extended, a regularly observed consequence is that the all-powerful background system must undergo an expensive and time-consuming upgrade.

Here we have attempted to take a different approach. The background system is only responsible for the management and consistency of the overall system. All other information is held locally in the cards. A global database is of course necessary for system administration, so that lost or faulty cards can be reproduced from data on hand. However, the information in this database should not be necessary for normal operation of the system.

A distributed smart card system may be thought of as a large tree with many branches, which like all trees draws its energy from photosynthesis in its many leaves. Energy production is distributed among the leaves, and it occurs in many places simultaneously. Figuratively speaking, an effective and well-designed smart card application behaves in the same way. The information is stored in decentralized cards and is thus protected against every form of attack. The large mass of information also does not burden the background system, which only has to deal with the centralized management tasks. The actual system processes are decentralized, just like the process of photosynthesis in the leaves of a tree, and they take place in parallel in localized terminals and smart cards. This makes it very easy to extend the overall system by adding more terminals and cards, without needing to worry about any major impact on the background computer system.

The opposite approach to the system just outlined is to concentrate all of the necessary activities in a central background system. In our analogy, this would amount to moving photosynthesis from the tree's leaves to its trunk, which would result in a huge trunk. The overall system would not only be very large and expensive, it would also be extremely vulnerable to disturbances in the background system. This should be avoided as much as possible.

In their ignorance of the characteristics of smart cards, many novice system operators make the mistake of designing the entire system from the top down. When it comes to the parts of the system that are most critical in terms of security, namely the terminals and smart cards, in many cases they simply stipulate that these components can be made secure in some more or less unknown manner.

By contrast, the designs described here represent a bottom-up approach, in which the system and its required features are defined by starting with the lowest-level object (the smart card) and working upwards. The risk of security gaps can be very effectively minimized using this approach, since the system is constructed securely from the smallest entity all the way to the top.

15.9.1 An electronic purse system for arcade games

Situation and objectives

This smart card application is intended to provide functions that allow small amounts of money to be paid into arcade game machines. Smart cards are to replace the coins conventionally used with such machines, in order to reduce operating costs.

There are two types of terminals. The first type is a loading terminal, which has a coin slot with a coin tester and can load electronic 'currency' into smart cards. The second type is a debiting terminal, which operates largely autonomously and debits the electronic currency from the smart card.

Requirements

The entire system should be anonymous, while still allowing all money flows to be monitored. If there is a suspicion of fraud, it must be possible to identify individual cards and selectively block them.

Since this is a payment system application, and considering that the equipment used is fully automatic and operates without human supervision, the design should aim for a medium level of security.

Proposed solution

The solution is based on a simple closed purse system that is specifically designed to meet the requirements of this application. Naturally, it could easily be used for similar applications by making small modifications to the files and procedures. We have avoided using an electronic purse system that is compliant with the CEN EN 1546 standard, since such a solution would be more expensive for the application provider than the proposed solution. In addition, our objective is to demonstrate the principles of a simple closed purse system.

In a central location, there is an automated machine that can accept both coins and banknotes and load the equivalent amount of electronic currency into a smart card. Neither a PIN nor any other user input is necessary, as the electronic purse is anonymous. The only accounting performed for the amounts loaded into the cards is based on individual card numbers, each of which is unique within the system.

A PC administers all of the data and the money flows. The PC also contains a database that holds general information for all issued cards. A daily or weekly balance calculation can be used to check that the money flows in the system remain closed.

At a payment (debit) terminal, the monetary units loaded into the smart card are debited from the electronic purse. A display on this terminal shows the user the amount that has been debited. To keep the cost of the terminals as low as possible, data transfers are protected by secure messaging instead of a shutter. Each terminal has a security module to store the secret key and keep track of the amounts paid, sorted by card number. At regular intervals, the data so obtained are transferred by cable or a special transfer card to the administration PC, which evaluates this information. The file tree of the proposed solution is shown in Table 15.11. It requires around 100 bytes in EEPROM, depending on the smart card operating system.

Table 15.11 File tree of the 'arcade games' sample application

File	FID	Structure	Description
MF	'3F00'	—	Smart card root directory
DF	—	—	Directory for the 'arcade games' application
DF.EF 1	'0001'	Transparent	Date of issue and card number
DF.EF 2	'0002'	Cyclic	Amount
DF.EF 3	'0003'	Linear fixed	Key 1
			Key 2
			Key 3
			Key 4

Regular data exchanges between the debiting terminals and the administration computer can be used to maintain a blacklist of blocked cards in the terminals. If a terminal determines that an inserted card is on the blacklist, it blocks the EF2 file containing the electronic money. After this, the card can no longer be used to make payments. The user must have the card unblocked at the administration terminal, and when this is done, a check can be made to see why the card was put on the blacklist.

The keys needed for this application are listed in Table 15.12. In the interest of having a simple overall system, we have not used derived keys or card-specific keys.

Table 15.12 Keys required for the 'arcade games' sample application

Key	Used for	Function	State transition
Key 1	MUTUAL AUTHENTICATE	Mutual authentication of the terminal and the smart card: – paying with the purse – blocking the purse	$x \to 1$
Key 2	MUTUAL AUTHENTICATE	Mutual authentication of the terminal and the smart card: – unblocking the purse – card management	$x \to 2$
Key 3	MUTUAL AUTHENTICATE	Mutual authentication of the terminal and the smart card: – loading the purse	$x \to 3$
Key 4	Secure messaging	Protecting data transmission – authentic mode	—

The proposed solution is very suitable for paying for services received from automatic equipment. Human supervision is unnecessary. However, it is not essential to use a special machine to automatically load 'money' into the cards. Cards could also be loaded manually at a service counter, in exchange for a cash payment equal to the amount to be loaded.

Table 15.13 Access conditions for the 'arcade games' application
($\geq$ 0: always, < 0: never, SM: secure messaging)

File	Read	Write	Block	Unblock	Increase amount	Decrease amount
EF 1	≥ 0	$= 2$	< 0	< 0	< 0	< 0
EF 2	$\geq 0 \wedge \text{SM}$	< 0	$= 1$	$= 2$	$= 3$	$= 1$
EF 3	< 0	< 0	$= 1$	$= 2$	< 0	< 0

With minor modifications, the system outlined here can also be used in a launderette or canteen.

Figure 15.27 illustrates the basic process for loading electronic currency into the electronic purse, while Figure 15.28 similarly illustrates the process for making a payment.

Smart card		**Terminal**	**User**
	↔	SELECT FILE (DF)	*Output*
	↔	SELECT FILE (EF 1)	"Insert money"
	↔	READ BINARY	
	↔	SELECT FILE (EF 2)	
	↔	READ BINARY	
	↔	ASK RANDOM	
	↔	MUTUAL AUTHENTICATE	
		enable secure messaging	
	←	INCREASE	
Response [. . . ||	→	IF (return code = OK)	*Output*
return code]		THEN command successfully executed	"xx euros loaded in
		ELSE abort	the smart card"

Figure 15.27 Basic command sequence for loading electronic monetary units into the purse in the 'arcade games' application

15.9.2 Access control system

Situation and objectives

The objective of this application is to create a smart card based, graduated access control system for a number of rooms and computer systems. This means that certain doors and computers will be fitted with terminals that will allow people to pass through the associated door or use the associated computer after communication with a smart card. It is important to be able to define various security levels, for which access can be limited to specific groups of users. Access will be granted after successful authentication and identification of the user. The necessary proof

Smart card		Terminal	User
	↔	SELECT FILE (DF)	
	↔	SELECT FILE (EF 1)	
	↔	READ BINARY	
	↔	SELECT FILE (EF 2)	
	↔	READ BINARY	
	↔	ASK RANDOM	
	↔	MUTUAL AUTHENTICATE	
		enable secure messaging	
	←	DECREASE	
Response [... ||	→	IF (return code = OK)	*Output*
return code]		THEN command successfully executed	"xx euros debited
		ELSE abort	from the smart card"

Figure 15.28 Basic command sequence for making a payment from the purse in the 'arcade games' application

will be possession of a genuine card and knowledge of its associated PIN. If both of these criteria are satisfied, access will be granted. The terminals must be able to maintain simple blacklists, so that 'lost' cards cannot be used for access, and such cards can be permanently blocked if necessary.

Requirements

In order to maximize user acceptance of the solution, the time required for any communication process between the terminal and the smart card, together with the subsequent granting of access, must not significantly exceed one second. A longer interval will sooner or later significantly impede user acceptance of the system and encourage users to use various tricks to circumvent security measures, such as propping open doors. Users must be able to select their own PIN codes so that they do not resort to writing their PIN codes on the cards.

The system should be designed for a moderately low level of security, as it is fairly unlikely to be subjected to elaborate attacks or analysis. The acquisition and operating costs of the system must not exceed those of a good conventional key–lock system, since the latter would otherwise represent a more economical alternative.

The system and smart card must be designed to allow timecard and canteen billing functions to be incorporated into the system at a later date.

Proposed solution

Simple terminals with 10-digit numeric keypads will be firmly attached to the appropriate doors and computers. These terminals can work autonomously, and they are fitted with economical, exchangeable security modules (such as smart cards in plug-in format). They can grant authorized persons access to the associated doors or computers. Any terminal at a critical or sensitive location can if necessary independently establish a link to the PC that serves as the central system computer, via a two-wire cable. This simple architecture satisfies the requirement for low operating costs.

The central computer has a simple multitasking operating system to allow it to execute several tasks in parallel. It is connected to a supplementary terminal that is responsible for administering the entire system.

The smart cards used in this system must have an operating system that can manage several applications and can create files (DFs and EFs) in the smart card file tree after the card has been issued, so that additional applications can be loaded as necessary after the cards have been issued. The amount of EEPROM required by the current application and projected future applications will not exceed 1 kB. The cards will be purchased from a card manufacturer complete with the operating system, and then configured appropriately using the administration computer.

All cards will initially have a standard and easily remembered PIN. The simplest option in this case is "0000". This eliminates the cost of generating PINs and preparing PIN letters. Upon receipt of the card, each user must change the standard PIN to some other number using the administration terminal, since all terminals will reject a PIN entry of "0000".

If the user forgets the PIN or the retry counter reaches its limit, the system terminal can be used to enter a new PIN and reset the retry counter after an authentication.

Since the system is required to have an intermediate level of security, and system management should not be too costly, a severely limited key management scheme is appropriate. Neither derived keys nor multiple generations of keys are used in this system. The keys only need to be separated by function, which leads to the arrangement shown in Table 15.14. The file tree that must be present in the smart card is described in Table 15.15, and the file access conditions are listed in Table 15.16.

File EF2, which contains the authorization data for access to rooms and computers, has a record-oriented structure. All records have the same length (linear fixed). Each record has an entry that indicates which rooms the cardholder is allowed to enter. It is also possible to define security levels, so that it is not necessary to explicitly list each room. Access can then be globally restricted to certain areas.

Table 15.14 The keys needed for the 'access control' application

Key	Used for	Function	State transition
C 1	MUTUAL AUTHENTICATE	Application management: – creating new files – writing to files – unblocking an application	$x \rightarrow 1$
V 2	MUTUAL AUTHENTICATE	Mutual authentication of the terminal and the smart card: – access authorization – blocking an application	$x \rightarrow 2$
PIN	VERIFY CHV	User identification	$2 \rightarrow 3$

Table 15.15 The file tree for the 'access control' application

File	FID	Structure	Description and contents
MF	'3F00'	—	Smart card root directory
DF	—	—	Directory for the 'access control' application
DE.EF 1	'0001'	Transparent	Last name, first name, department
DE.EF 2	'0002'	Linear fixed	Authorization level
DE.EF 3	'0003'	Linear fixed	Key 1; key 2; PIN

Table 15.16 File access conditions for the 'access control' application ($\geq$0: always, <0: never)

File	Read	Write	Block	Unblock	Create DFs and EFs
DF	—	—	—	—	= 1
EF 1	≥ 0	= 1	< 0	= 1	—
EF 2	= 3	= 1	= 2	= 1	—
EF 3	< 0	< 0	= 2	= 1	—

Since experience shows that it is frequently necessary to modify and restructure access control systems, the data content of each record should have a TLV structure. This allows extensions and modifications to be implemented in a technically elegant manner.

Only standard commands provided by commercially available ISO-compliant or ETSI-compliant operating systems are used for the smart cards. This means that nothing has to be programmed in the smart cards, which considerably reduces acquisition costs. The following commands are needed:

ASK RANDOM	READ BINARY
CHANGE CHV	REHABILITATE
CREATE	SELECT FILE
INVALIDATE	UNBLOCK CHV
MUTUAL AUTHENTICATE	VERIFY CHV
	WRITE BINARY

Figure 15.29 shows the typical command sequence for an access control session. If necessary, the terminal can use READ BINARY directly following the ATR to read the user's name from the file and check it against a blacklist. If the user's name is on the blacklist, further use of the card for access control can be prohibited by using the INVALIDATE command to block all EFs. If necessary, the application can be reactivated at the administration terminal using REACTIVATE, following mutual authentication.

If the system operator decides to also use the smart cards for canteen billing, a new application with its own DF and EFs must be generated. There are two ways to do this. Either all employees must bring their cards to the administration terminal, or the necessary files can be automatically downloaded during access checks. The second approach is certainly less expensive and more user-friendly, since it does not require any extra administrative effort.

Smart card		Terminal	User
	←	Reset	
ATR	→	IF ATR = OK	
		THEN continue	
		ELSE abort	
	↔	SELECT FILE (DF)	*Output*
			"Please enter PIN"
	↔	VERIFY CHV	
	↔	SELECT FILE (EF 2)	
	↔	READ RECORD	
		evaluate file contents	
		IF (permission = yes)	
		THEN actuate door opener	*Output*
			"Please enter"

Figure 15.29 Access control command sequence for the 'access control' application

The required control of access to computer systems is completely analogous to the process just described. The only difference is that instead of a door release mechanism being activated, a signal is sent to the computer to tell it to grant access to the user.

15.9.3 Testing the genuineness of a terminal

Situation and objectives

There are situations in which it should be possible for a card user to verify the authenticity of a terminal. One example is a terminal in a supermarket, in which the user must enter a PIN after inserting the card. A counterfeit terminal could be used to spy out secret PIN codes.[8] If the card is subsequently stolen by a person who already knows the PIN, the thief could use the card to make purchases or obtain money from a cash dispenser.

In the summer of 1997, a counterfeit cash dispenser at the Marienplatz in Munich was used in a comparable manner to illicitly collect magnetic-stripe data and associated PIN codes. If smart cards are used, good protection against this type of attack can be provided with a suitable application design.

Requirements

It is necessary to design a component of a smart card application that allows the card user to recognize a counterfeit smart card terminal (but not a manipulated terminal). The user must not need any additional technical aids or equipment to check the terminal.

[8] See also Section 8.1.1, 'Testing a secret number'

Proposed solution

The proposed solution involves storing a password that is known only to the card user in a file in the smart card. This file can be read by the terminal only after it has successfully authenticated itself with respect to the smart card via a secret key.

After this authentication process, the terminal is allowed to read the password from the file and show it on its display. As soon as the card user sees the password and verifies that it is correct, he or she can assume that the terminal is genuine, since only the user knows the password. Only after he or she has verified the password will the user enter the PIN that makes the rest of the transaction possible.

The procedure just described is recommended in the DIN specification for German signature cards, for example, in order to allow card users to determine whether public signature terminals are genuine.[9]

An important limitation of the solution must be mentioned. This is that it allows a counterfeit terminal to be recognized, but not a manipulated terminal. If the terminal software could be modified without losing the secret key in the process, it would be possible for a manipulated terminal to correctly authenticate itself with respect to the smart card and then display the password. This limitation should be taken into account in any application in which this technique is used. However, this is basically not a critical issue, since a terminal that can be manipulated to this extent will allow significantly more extensive forms of attack than just spying out PIN codes.

The proposed solution, which is presented in the form of specific files and access conditions in Tables 15.17 and 15.18, is not a complete smart card application. Instead, it is a sort of design template that can be merged into any desired application. Consequently, the FIDs and state transitions, as well as the procedure illustrated in Figure 15.30, can be modified as necessary to use different values or command sequences. This example is primarily intended to convey the basic idea of how the genuineness of a terminal can be tested, rather than to serve as a concrete application.

Table 15.17 Keys needed for testing the genuineness of a terminal

Key	Used for	Function	State transition
key 1	EXTERNAL AUTHENTICATE	Authentication of the terminal by the smart card	$x \rightarrow 1$
PIN	VERIFY CHV	Identification of the user	$1 \rightarrow 2$

Table 15.18 File tree and access conditions for testing the genuineness of a terminal ($\geq$0: always, <0: never)

File	FIC	Structure	Read	Write	File contents
EF 1	'0001'	transparent	$= 1$	$= 2$	password
EF 2	'0002'	linear fixed	< 0	< 0	key 1, PIN

[9] See also Section 14.4, ‘Digital Signatures’

Smart card		Terminal	User
	←	Reset	
ATR	→	IF ATR = OK	
		THEN continue	
		ELSE abort	
	↔	EXTERNAL AUTHENTICATE	
		(with key 1)	
	↔	IF (authentication is successful)	
		THEN continue	
		ELSE abort	
	↔	SELECT FILE (EF 1)	
	↔	READ BINARY	*Output the content of*
			the password file
			IF (password correct)
			THEN terminal is
			genuine
			ELSE abort
	↔	VERIFY CHV	*Output*
			"Please enter PIN"

Figure 15.30 Command sequence for verifying the genuineness of a terminal

16 Appendix

16.1 GLOSSARY

The following pages contain a list of terms typically used in the smart card world. Precise, comprehensive definitions of terms can also be found in the ISO/IEC 7816 family of standards. The equivalent standard in the area of electronic purses with regard to terminology is EN 1546, which comprehensively and concisely defines and explains all of the associated technical terms.

The keywords in this glossary are listed as abbreviations or in full according to customary usage. An arrow symbol (→) in front of a term refers to another entry in the glossary in which the term (set in *italics*) is explained.

Larger collections of general terms used in informatics can be found in the DIN 44 300 standard and numerous lexicons devoted to EDP terminology, such as [Pfaffenberger 97, Dictionary of Computing 91].

µP card

An alternate designation for → *microprocessor card*.

0-PIN

A common, known PIN used for all newly issued → *smart cards*, which does not allow access to the actual user functions. It is thus a type of → *trivial PIN*. The first time the card is used, the 0-PIN must be changed to a user-selected PIN using the usual mechanisms (usually CHANGE CHV), with the value of the 0-PIN not being an allowed value for the new PIN. The purpose of a 0-PIN is to allow the user to unambiguously determine whether the card is still in its original issued state when he or she receives it or has been illicitly used while underway. The term '0-PIN' comes from the fact that the value "0000" is often used for this type of PIN.

Smart Card Handbook, Third Edition. W. Rankl and W. Effing
© 2004 John Wiley & Sons, Ltd ISBN: 0-470-85668-8

1-μm/0.8-μm/... technology

In the fabrication of semiconductor chips, the performance of the technology used is traditionally expressed in terms of the dimension of the smallest possible transistor structure on the semiconductor material. This is usually the width of the gate oxide strip of a transistor. Currently, the smallest possible structure widths are approximately 0.25 μm and 0.13 μm. Naturally, it is always possible to make structures on the chip that are larger than the minimum dimension.

1K/2K/4K/.../*n*K-chip

The designation '*n*K-chip' (where *n* is a positive integer) is frequently used as a simplified type designation for a → *microcontroller* with a certain size of → *EEPROM* in kilobytes. A 32K-chip is thus a smart card microcontroller with 32 kB of EEPROM. Specifying the size of the EEPROM is sufficient for rough comparisons of commonly used smart card microcontrollers.

1G (first generation)

Refers to the first generation of mobile telecommunication networks, which have a cellular architecture and use analog technology. Typical examples of 1G systems are AMPS and the German C-Netz.

2G (second generation)

Refers to the second generation of mobile telecommunication networks, which have a cellular architecture and use digital technology. Typical examples of 2G systems are → *GSM* and → *CDMA*.

3DES

→ *triple DES*

3G (third generation)

Refers to the third generation of mobile telecommunication networks, which have a cellular architecture and use digital technology. A typical example of a 3G system is → *UMTS*, which in turn is a member of the → *IMT-2000* family.

3GPP (Third Generation Partnership Project) [3GPP]

The task of the Third Generation Partnership Project, which was founded by the five standards institutes ANSI T1 (USA), ARIB (Japan), ETSI (Europe), TTA (Korea) and TTC (Japan), is to generate internationally usable technical specifications for third-generation (→ *3G*) mobile telecommunications systems based on an enhanced GSM core system (→ *GSM*). The participating standards bodies will then translate these specifications into corresponding standards. 3GPP was founded in Copenhagen by the leading international standards organizations in the field of telecommunications. The Third Generation Partnership Project 2 (3GPPP2) has similar responsibilities, although the latter project focuses on further development of non-GSM systems (such as CDMA systems) in the direction of the third generation.

3GPP2

→ *3GPP*

4G (fourth generation)

Refers to the fourth generation of mobile telecommunication networks, which is currently only in the conceptual stage.

8-bit/16-bit/32-bit CPU

An important characteristic with regard to the processing power of a → *microprocessor* is the width of the register for data to be processed in the processing unit. It is expressed in terms of the number of bits.

A2C (administration to customer)

Public administration and end users.

A3 (algorithm 3)

Designation for a cryptographic algorithm used in → *GSM* for the authentication of the SIM by the background system using a challenge–response procedure. A3 is chosen by the network operator and is thus not the same for the entire GSM system.

A5 (algorithm 5)

Designation for a cryptographic algorithm used in → *GSM* for encrypting data on the air interface between the mobile station and the base station or background system. A5 is the same for the entire GSM system.

A8 (algorithm 8)

Designation for a cryptographic algorithm used in → *GSM* for generating session keys (Kc) used for encrypting speech data on the air interface. A8 is chosen by the network operator and is thus not the same for the entire GSM system.

Access conditions (AC)

In connection with the file system of a smart card, a finite number of conditions that must be satisfied prior to accessing the associated file using one of the various types of access supported by the operating system (e.g., read, write, delete). Access conditions are usually specified independently for each type of access.

Acquirer

An entity that establishes and manages data links and data exchanges between the operator of a payment system and individual service providers. An acquirer may consolidate individual transactions that it receives, so that the system operator receives only collective certificates.

Activation sequence

Specifies the order of events for activation of the electrical signals for a → *smart card microcontroller* when powering up a → *smart card*. It does not say anything about the sequence of events for mechanical contacting. The objective of the activation sequence is to protect the smart card microcontroller, which is sensitive to charges and voltages on its contacts. (→ *deactivation sequence*)

Administrative data

Data that are used only for managing → *user data* and no other particular significance with respect to an → *application*.

AES (Advanced Encryption Standard)

A symmetric → *cryptographic algorithm*, originally developed by Joan Daemen and Vincent Rijmen and published as the Rijndael algorithm. Following a public competition and evaluation process, the → *NIST* selected this algorithm as the successor to the DES in 2000 and published it as a US standard (FIPS 197) in 2001.[1]

AFNOR (*Association Française de Normalisation*)

A French standards organization based in Paris.

AID (application identifier)

An AID identifies an → *application* in a → *smart card*, as specified in ISO/IEC 7816-5. Part of the AID may be registered nationally or internationally, in which case it is reserved for the

[1] See also Section 4.7.1, 'Symmetric cryptographic algorithms'

registered application and is unique in the entire world. An AID consists of two data elements: a registered identifier (RID) and a proprietary identifier (PIX).[2]

AMPS (Advanced Mobile Phone System)

A cellular mobile telephone standard, predominantly used in the USA, Latin America, Australia and parts of Asia. It employs analog technology and operates in the 800-MHz band. AMPS mobile telephones do not have → *smart cards* and are often successfully attacked, in part for this reason. The upgraded version of AMPS is D-AMPS, a digital system that also operates in the 800-MHz band.

Analog

Refers to systems in which signals may assume an unlimited number of values.

Analysis

In the sense of software development, the process of determining the customer requirements for an informatics system and completely and unambiguously describing these requirements. In simplified terms, the result of analysis is a description of 'what' is to be produced. The subsequent step in a sequential software development project is → *design*.

Anonymization

Modifying person-specific data in such a manner that it is no longer possible to associate the modified data with the original person. (→ *pseudo-anonymization*)

ANSI (American National Standards Institute) [ANSI]

An American standards organization based in New York.

Anticollision method

A method that permits access to multiple contactless cards without interference.

APDU (application protocol data unit)

A software data container used to package data for an → *application* for exchange between a → *smart card* and a → *terminal*. The APDU is converted into a transmission protocol data unit (TPDU) by the transmission protocol and then sent by the smart card or terminal

[2] See also Section 5.6.1, 'File types'

via the serial interface. APDUs can be classified into → *command APDUs* and → *response APDUs*.[3]

API (application programming interface)

A software interface, specified in detail, that provides access to specific functions of a program.

Application

All of the data, files, → *commands*, processes, states, mechanisms, algorithms and programs in a → *smart card* that allow it to be used in a particular system. An application and its associated data are usually located in a dedicated DF directly below the MF. Such an application is often called an 'oncard application'. The opposite to this is an 'offcard application', which encompasses all of the programs and data not present in the smart card that are necessary for using the oncard application in the smart card.

Application operator

An entity that operates an → *application* using → *smart cards*. The application operator is usually the same as the application provider.

Applet

A program written in the Java programming language and executed by the virtual machine of a computer. For reasons of security, the functionality of an applet is restricted to a previously defined program environment. In the realm of → *smart cards*, applets are sometimes called 'cardlets'. An applet usually corresponds to a smart card → *application*.

Applet developer

A person or organization that develops an → *applet*.

ASK(amplitude-shift keying)

A modulation method in which the amplitude of the carrier wave is switched between two states.

ASN.1 (Abstract Syntax Notation 1)

A description language (syntax and grammar) for data that allows data and data types to be unambiguously defined and represented independent of the type of computer system used. The

[3] See also Section 6.5, 'Message Structure: APDUs'

corresponding data can then be coded in concrete terms using the → *BER (Basic Encoding Rules*) and the → *DER(Distinguished Encoding Rules).* ASN.1 is defined by ISO/IEC 8824 and ISO/IEC 8825.[4]

Assembler

A program that translates assembly-language programs into machine language, which can be executed by a processor. After the assembly process, it is usually necessary to link the resulting code using a linker program. 'Assembler' is also often used as a short form for 'assembly-language program code'.

Asymmetric cryptographic algorithm

→*cryptographic algorithm*

Asynchronous data transmission

Data transmission in which the data are transmitted independent of any prescribed timing reference. (→ *synchronous data transmission*)

Atomic operation

One or more operations in a program that are executed either entirely or not at all. In → *smart cards*, atomic operations are frequently used in connection with EEPROM write routines, in order to ensure that the data content is consistent at all times.[5]

ATR (answer to reset)

A sequence of bytes sent by a → *smart card* in response to a (hardware) reset. The ATR includes various parameters relating to the transmission protocol for the smart card.[6]

Attribute

In the sense of → *object-oriented programming*, a data container holding an → *object* (in the procedural sense, the variables). Attribute values can be read or modified using → *methods.*

[4] See also Section 4.1, 'Structuring Data'
[5] See also Section 5.10, 'Atomic Operations'
[6] See also Section 6.2, 'Answer to Reset (ATR)'

Authentication

The process of verifying the genuineness of an entity (such as a smart card) using a cryptographic procedure. Put simply, authentication amounts to using a prescribed procedure to determine whether someone is actually the person he or she claims to be.

Authenticity

A property possessed by an entity or message that is genuine and unaltered.

Authorization

Testing whether a particular action is allowed to be performed; equivalent to granting someone the authority to do something. For example, when a credit card transaction is authorized by the credit card issuer, the card data are checked to see if the data are correct, the amount of the purchase is less than the permitted limit and so on. The payment is then allowed if all checks are satisfactory. An authorization can be achieved by means of authentication of the party in question (such as a smart card). Put simply, authorization amounts to giving someone permission to perform a particular action.

Auto-eject reader

A terminal that can automatically eject an inserted card in response to an electrical or mechanical signal.

B2A (business to administration)

Designates the handling of → *e-commerce* business between enterprises and public administrations.

B2B (business to business)

Designates the handling of → *e-commerce* business between enterprises.

B2C (business to customer)

Designates the handling of → *e-commerce* business between enterprises and end users.

Background system

Any type of computer system above the level of the terminal that processes and manages data.

Bad case

The case in which a logical decision leads to an unfavorable or undesired result.

Bad-day scenario

Another expression for → *bad case*.

Baud

Designates the number of state changes per second during a data transmission. Depending on the transmission method used, one or more data bits can be transmitted for each change of state. For this reason, the baud rate is equivalent to the transmission rate in bits per second only in the special case that only one bit is transmitted for each change in state.

Bearer

Designates the bearer service used to transport data to a terminal device. For example, SMS is a possible bearer for WAP

Bellcore attack

→*differential fault analysis*

BER (Basic Encoding Rules)

The BER, which are defined in → *ASN.1*, allow data to be coded in the form of data objects. A BER-coded data object has a tag, a length and a value (the actual data component), and optionally an end marker, and is thus also referred to as TLV-coded data. The BER format also permits chained data objects. The Distinguished Encoding Rules (DER), which are a subset of the BER, indicate among other things how the length parameter of the data object is to be coded (1, 2 or 3 bytes).[7]

Big-endian

→*endianness*

Binary-compatible program code

A program that can be executed directly by a → *microprocessor* without using auxiliary programs or the like (→ *program code*).

[7] See also Section 4.1, 'Structuring Data'

Blacklist

A list in a database identifying all cards or devices that are no longer allowed to be used in a particular → *application.* (→ *hotlist, graylist, whitelist*)

Blackbox test

A test based on the assumption that the party performing the test has no knowledge of the internal processes, functions and mechanisms of the software being tested.

Bluetooth [Bluetooth]

A wireless network technology intended to be used for short-range communications (<100 m) in the 2.4 GHz band, with a maximum gross data transmission rate of around 1 Mbit/s. Ericsson, as the initiator of this technology, chose the name in memory of the Danish king Harald II, who lived approximately 1000 years ago and was nicknamed 'Bluetooth'. His major achievement was merging many separate regions into a unified kingdom.

Bond-out chip

A microcontroller mounted in a multi-pin ceramic package providing free access to all of the memory busses internal to the chip, thus allowing the commonly used mask-programmed →*ROM* to be replaced by memory external to the chip. A bond-out chip is used to allow software to be tested in the target hardware without using a → *ROM mask.*

Boot loader

A small, simple program whose only purpose is to load other, larger programs into memory, for example via a serial interface, and run them from memory (→ *loader*). A boot loader is typically used to load the actual program code into a new chip or a new piece of electronic equipment. In many cases, the boot loading process can be performed only once.

BPSK (binary phase shift keying)

180-degree phase shift keying, yielding two phase states.

Browser

A program for viewing hypertext documents, navigating among such documents and running → *program code* embedded in hypertext documents. Browsers with simple structures that require little memory and processing capacity are often called microbrowsers. Some microbrowsers run as → *applications* within a → *smart card operating system* (such as the SIM Alliance browser, also known as the S@T browser), while others are integrated into the software of the mobile telephone (e.g., WAP browsers). The functionality of browsers can be extended using downloadable software components called browser plug-ins.

Brute-force attack

An attack on a cryptographic system based on computing all possible values of a key.

BSI (*Bundesamt for Sicherheit in der Informationstechnik*) [BSI]

The German *Bundesamt for Sicherheit in der Informationstechnik* (BSI) was founded in 1991 as the successor to the *Zentralstelle für das Chiffrierwissen*. The functions of the BSI include investigating the security risks of IT applications, testing and evaluating the security of IT systems, formally approving IT systems for government agencies and assisting criminal investigation agencies and agencies charged with the protection of the German constitution. It also advises manufacturers, operators and users with regard to IT security, and in this regard it often specifies the general conditions for using cryptography in Germany.

Buffering

A typical type of attack on magnetic-stripe cards involving first reading and storing (buffering) the data on the magnetic stripe. After the data have been modified using a terminal (e.g., changing the state of the retry counter), the original data are written back to the magnetic stripe.

Bug fix

In software development, supplementary →*program code* used to remedy a known error (bug). In contrast to a →*work-around*, a bug fix eliminates the actual error.

Burst

→*signal burst*

Bytecode

This term has several different meanings and can only be correctly interpreted in the context in which it is used. One widely used meaning is related to the Java system, in which bytecode is the name given to the intermediate code produced (compiled) from the source code by a Java compiler. This bytecode is standardized by the Sun Corporation and is interpreted by the Java virtual machine. The term 'bytecode' is also used in the context of microbrowsers (→*browser*), where it is understood to mean the translated code produced from a hypertext document by the bytecode converter. The result of this translation is the bytecode, which is then interpreted by the microbrowser.

CAD (card acceptance device)

In the realm of electronic payment systems, the designation CAD is frequently used to refer to a smart card terminal, in place of the ISO abbreviation → *IFD* (interface device).

CAMEL (Customized Applications for Mobile Enhanced Logic)

Supplementary possible feature of →*GSM* for supporting the functionality of intelligent networks (IN). With CAMEL, for example, it is possible to modify a dialing number during call setup on the network. This allows applications such as international →*roaming* using prepaid cards and internationally available standard service numbers to be implemented in a simple manner.

CAP file (card application file)

A data format used to exchange data between the Java Offcard Virtual Machine and the Java Oncard Virtual Machine.

Card

General term used to refer to a thin rectangular piece of material with rounded corners whose physical dimensions comply with an international standard. A card can have various card components, including a semiconductor chip (→*chip card*, →*smart card*).

Card accepter

An entity with which cards can be used for a particular type of transaction (such as payment). A typical example is a merchant who accepts credit cards for making payments.

Card body

A plastic card forming an intermediate product in the production of smart cards. It is further processed in subsequent production steps and receives additional functional components, such as the embedded chip.

Card component

A supplementary functional unit of a →*card*, such as a →*signature panel*, →*embossing*, a →*magnetic stripe*, a chip (→*memory card*, →*microprocessor card*) or a keypad (→*system on card*).

Card issuer

An entity responsible for issuing cards. In the case of mono-application cards, the card issuer is usually also the application provider, but this is not necessarily the case.

Card manufacturer

An entity that produces card bodies in which it embeds modules.

Card Modeling Language (CML)

An abstract, operating-system independent description language for defining smart card →*applications*.

Card owner

A natural or legal person having legal control over a card who can do whatever he wishes with the card. In the case of a credit or debit card, the bank issuing the card is often the card owner, and the customer who uses the card is only the →*cardholder*.

Card reader

A device having a relatively simple electrical and mechanical construction used to accept → *smart cards* and make electrical contact with them. Unlike a terminal, a card reader does not have a display or a keypad. Despite the name, a card reader can usually also be used to write data to a card.

Card user

A person using a card, who is usually but not necessarily the → *cardholder*.

Cardholder

A person actually having a card in his possession and having the legal right to use the card. The cardholder need not necessarily be the same as the →*card owner*.

Cardholder verification method (CVM)

A method for the →*identification* of persons. This usually consists of PIN testing, but biometric user identification may be used in more sophisticated systems.

Cardlet

→*applet*

Cavity

The recess in the card body for the module to be implanted, usually produced by milling.

CCITT (*Comité Consultatif International Télégraphique et Téléphonique*)

Originally, an international committee for telephone and telegraph services, based in Geneva. With the assumption of additional responsibilities, it is now known as the → *ITU*.

CCS (cryptographic checksum)

A cryptographically generated checksum for data, which is used to allow manipulation of the data during storage to be recognized. A CCS used to protect data during transmission is called a message authentication code (→ *MAC*).

CDMA (code division multiple access)

A multiple-access method for the concurrent transmission of data from multiple transmitters to a single receiver within a frequency band. For this purpose, the narrow-band radio signal is mapped onto a wideband radio signal, or 'spread', using a transmitter-specific mapping rule. If this mapping rule is known, the receiver can recover the original narrowband signal from the received wideband signal. CDMA is used in UMTS for a the air interface between the mobile telephone and the base station.[8] With wideband code division multiple access (WCDMA), two separate frequency bands are used for → *uplink* and → *downlink*, for which reason this method is often referred to as frequency division / code division multiple access (FD/CDMA). With time division / frequency division multiple access (TC/CDMA), the uplink and downlink are separated using different time slots.

CDMA 2000 (Code Division Multiple Access 2000)

Third-generation (→ 3*G*) mobile telecommunication system using the 2000-MHz frequency band and having features similar to those of → *UMTS*. The smart card intended to be used in CMDA, which is called the → *R-UIM*, is optional.

Cell

In mobile telecommunications systems, the smallest subdivision of the geographic structure of the network.

[8] See also Section 13.1.1, 'Multiple-access methods'

Cellular technology

Refers to an analog or digital mobile telecommunication system organized in the form of cells. The transmitter and receiver stations of the network, which are commonly called base stations, are usually located at the approximate centers of the cells.[9]

CEN (*Comité Européen de Normalisation*)

A European standards organization based in Brussels. It is composed of all national European standards organizations and is the official institution of the European Union for generating European standards.

CEPS (Common European Electronic Purse Specifications) [CEPSCO]

A specification for → *electronic purses*, with emphasis on international interoperability (→ *interoperable*), including all components necessary for operating an electronic purse system. The first version of CEPS was published in 1999 by CEPSCO. It is based on many of the principles of EN 1546, the European standard for electronic purses.

CEPT (Conférence Européenne des Postes et Télécommunications)

A European standards organization for national telecommunications companies.

Certificate

A public key that has been signed by a trustworthy body and provided with associated administrative data, in order to allow it to be recognized as authentic by third parties (→*PKI*). The most widely used and best-known specification for the structure and coding of certificates is the X.509 standard.

Certification authority (CA)

A certification body in a public-key infrastructure (→ *PKI*) that certifies public keys for →*digital signatures*, which means that it guarantees their authenticity by signing the user's public key using its own private key. If necessary, the certification authority makes the signed public keys available in a directory (→*directory service*) in the form of →*certificates*. A CA can itself generate the necessary key pairs (private and public). For organizational reasons, certification authorities often have a hierarchical structure, with the highest-level certification authority being called the 'top-level CA' or 'root CA'.

[9] See also Section 13.1.2, 'Cellular technology'

Certificate revocation list (CRL)

A list, held by a → *directory service*, that identifies all certificates within a → *PKI* that are blocked and no longer accepted.

Challenge–response procedure

A commonly used authentication procedure in the smart card realm that is based on a secret key for a cryptographic algorithm, with the key being a shared secret of the communicating parties. One of the communicating parties sends the other party a random number (the challenge). The latter encrypts it using a cryptographic algorithm and sends the result (the response) back to the challenger. The challenger then applies the reverse function of the cryptographic algorithm to the encrypted version of the random number it has received and compares the result to the originally sent random number. If they match, the challenger knows that the other party also knows the secret key, and from this it concludes that the other party is authentic.[10]

Chinese remainder theorem

A technique used to accelerate the RSA algorithm. Since it requires knowing both of the prime numbers (p and q), it is only used for decryption or signing.

Chip card

A general term for a card, usually plastic, containing one or more semiconductor chips. A chip card can be either a → *memory card* or a → *microprocessor card*. In English-speaking countries, the term → *smart card* is generally used instead.

Chip module

A carrier and support for a die, with a set of contact elements arranged on its surface. The short form 'module' is frequently used to refer to the chip module.

Chip-on-tape (COT)

A packaging arrangement in which chip modules are placed in adjacent pairs on a thin, flexible tape that is typically 35 mm wide.

Chip size

The surface area of a chip, usually measured in square millimeters. The chip price is to a large degree directly proportional to the chip size. The maximum chip size for smart card microcontrollers is approximately 25 mm^2, due to the types of modules currently used.

[10] See also Section 4.11.2, 'Symmetric mutual authentication'

CHV

→*PIN*

CICC (contactless integrated chip card)

The official ISO name for a card for which data and power are transferred using electromagnetic fields without contact with the card. The chip may be a memory chip or a microcontroller chip.

Circuit-switched

Circuit-switched data transmission employs a direct connection (i.e., a physical line) between the two parties. In general, the charges for a circuit-switched connection are based on the duration of the connection, rather than the amount of data exchanged (→*packet-switched*). Some typical examples of circuit-switched data transmission are analog and ISDN telephone connections using the fixed telephone network.

Class

In the context of →*object-oriented programming*, a sort of abstract set of instructions for constructing an object, or in other words, for constructing the →*attributes* and →*methods* of an object and its relationships to other objects.

Class file

A class file stores a compiled Java program (one that has been translated into bytecode), along with supplementary information. After being loaded, the class file is executed by the Java virtual machine.

Cleanroom VM

→*Java Card virtual machine*

Clearing

In an electronic payment system, the process of settling accounts between a party that accepts electronic payments (usually a merchant) and the associated bank.

Clearing system

A computer-based background system that performs centralized account settlements in an electronic payment application.

CLIP

Europay brand name for several technologically different electronic purse systems using smart cards.

Clock-rate conversion factor

The clock-rate conversion factor (CRCF) defines the length of one bit (the bit interval) for data transmission, in terms of the number of clock cycles per bit interval. The short form 'divider' is commonly used as an equivalent term.

Clone

→*cloning*

Cloning

Attacking a smart card system by making a complete copy of the ROM and EEPROM of a microcontroller.

Closed application

A smart card → *application* that is only available to the application operator and cannot be used for general purposes.

Closed purse

An instance of a closed → *application* for an electronic purse. A closed purse can be used only within the limits defined by the application operator, and not for general payment transactions.

CMM (Capability Maturity Model)

An internationally used model for ascertaining the degree of maturity of software development. The degree of maturity is determined using a standardized list of questions and has five levels. The first maturity, Level 1 designates a more or less chaotic development process, while the highest possible maturity level, Level 5, designates a orderly and continually self-improving development process.[11]

[11] See also Section 15.7, 'Life Cycle Models'

CODEC (compressor/decompressor or coder/decoder)

A hardware chip or algorithm intended to be used for the compression and decompression or encryption and decryption of data.

Cold reset

→ *reset*

Collision

A collision occurs when two or more contactless cards located within the active range of a terminal concurrently transmit data to the terminal with the result that the received data cannot be decoded or unambiguously recognized.

Combicard

A registered trademark of ADE, which designates a → *dual-interface card.*

Command

In the realm of → *smart card operating systems*, an instruction to the smart card to perform a specific action. The result of a command is a response returned by the smart card, which at minimum contains status information and optionally may contain data related to the executed command. Commands are transferred to the smart card using → *command APDUs*, while responses are transferred using → *response APDUs*.

Command APDU

A → *command* sent from a terminal to a smart card, consisting of a command header and an optional command body. The command header in turn consists of a class byte, an instruction byte and two parameter bytes P1 and P2 (→ *APDU*).[12]

Common Criteria (CC) [CC]

A criteria catalog for the development and → *evaluation* of information technology systems, which is intended to replace national and international criteria catalogs such as → *TCSEC* and → *ITSEC*. The Common Criteria were first published in 1996 by the → *NIST* as Version 1.0, and since then they have been internationally standardized as ISO 15408. The currently valid revision is Version 2.0 of 1998.

[12] The command APDU is described in detail in Section 6.5.1, ‘Structure of the command APDU’

Compiler

A program that translates a program written in a language such as Basic or C into a machine language that can be directly executed by a processor. After a program has been compiled, it is normally necessary to link the code using a linker program.

Completion

The process of completing the operating system by loading the EEPROM portion. This allows the operating system to be modified and updated after the chips have been manufactured without requiring a new ROM mask to be generated. Identical data are written to each smart card during completion, so in principle it is a sort of initialization.

Contacts

The six or eight contact elements located on the front side of a → *smart card* form the electrical interface between the terminal and the microcontroller in the smart card. All electrical signal pass via these contacts.

Contactless card

Abbreviated designation for a type of → *smart card* for which energy and data are transferred using electromagnetic fields without any contact with the card (→ *CICC*).

Core foil

An alternate name for → *internal foil*.

Core voltage

The voltage used by a microprocessor or microcontroller directly within the chip. If the core voltage is lower than the external voltage applied to the chip, the external voltage must be suitably reduced by a voltage converter integrated into the chip. Low core voltages are necessary to compensate for reduced breakdown voltages resulting from increasingly smaller structure widths and to reduce the charge and discharge currents resulting from internal capacitances. A microcontroller built using 0.13-µm-technology, for instance, typically has a core voltage of 1.8 V.

COS (card operating system)

Common designation for a → *smart card operating system*. It often forms part of the product name of the operating system (e.g., STARCOS).

CP8

Brand name of a → *multiapplication smart card* operating system from Bull [Bull], available in several versions.

CPU (central processing unit)

→ *microprocessor*

CRC (cyclic redundancy check)

A simple, widely used type of error detection code (→ *EDC*) for protecting data. A CRC must be specified using an initial value and a divider polynomial before it can be used.

Credit card

A card, with or without a chip, that indicates that the cardholder has been extended credit within certain limits, and with which payment takes place some time after the goods or services have been received. This type of payment is often called 'buy now, pay later'. The widely used embossed credit cards are typical examples of this type of card.

Cryptoalgorithm

→ *cryptographic algorithm*

Cryptocard

→ *microprocessor card*

Cryptographic algorithm

A computational rule with at least one secret parameter, the → *key*, that can be used to encrypt or decrypt data. There are symmetric cryptographic algorithms (such as the DES algorithm) that use the same key for encryption and decryption, and asymmetric cryptographic algorithms (such as the RSA algorithm) that use a public key for encryption and a secret (private) key for decryption.

Cryptoprocessor

In the realm of smart cards, a supplementary numerical processing unit in a microcontroller that is optimized for the rapid computation of secret-key algorithms (such as DES) and/or public-key algorithms (such as RSA, DSA and ECC).

CT-API (Chipcard Terminal – Application Programming Interface)

An application-independent interface specification for connecting → *MKT* terminals to PCs; widely used in Germany. It is published by Teletrust Deutschland.

Customer card

In an electronic payment system, a → *smart card* used by customers to make payments at merchant terminals.

D-AMPS

→ *AMPS*

DEA (Data Encryption Algorithm)

Another name for → *DES*.

Deactivation sequence

Specifies the order of events for deactivation of the electrical signals for a → *smart card microcontroller* when powering down a → *smart card*. It does not say anything about the sequence of events for mechanical decontacting. The objective of the deactivation sequence is to protect the smart card microcontroller, which is sensitive to charges and voltages on its contacts. (→ *activation sequence*)

Debit card

A card, with or without a chip, that indicates that the cardholder has been granted certain powers of disposition, with which payment takes place when the goods or services are received. For this purpose, a debit card is linked to a bank account to allow the amount of the payment to be immediately transferred. This form of payment is often referred to as 'pay now'. A typical example of a debit card is the Eurocheque card.

Debugging

Searching for and eliminating errors, with the objective of detecting and correcting as many errors in a software program as possible. Debugging is normally performed by software developers during → *implementation* and is not the same as testing (→ *test*).

DECT (Digital Enhanced Cordless Telecommunications; previously 'Digital European Cordless Telecommunications')

A specification for cordless telephones operating the 1.9-GHz band using → *cellular technology* with digital data transmission; published by → *ETSI*. Although the DECT standard has provisions for using a smart card in the mobile part of the telephone, it is specified as being optional, with the result it is not used.

Defragmentation

The process of shifting data stored at different physical locations in memory until the data occupy a contiguous region of memory. The essential portions of a defragmentation process must operate in an atomic manner in order to prevent memory inconsistency in the event of premature termination of the process.

Delamination

The undesired separation of foils that have been attached to each other (laminated) using heat and pressure. Delamination of a card can for example be caused by using a non-thermoplastic ink to print overly large areas between the core foil and overlay foil. Such inks are commonly used in offset printing.

Depersonalization

Reversing the electrical → *personalization* of a smart card. If the → *smart card operating system* allows depersonalization, it may be performed using a special command following authentication. One use for depersonalization is to restore incorrectly personalized cards to their original condition, so that they can be reused.

DER (Distinguished Encoding Rules)

→ *BER*

DES (Data Encryption Standard)

The best known and mostly widely used symmetric → *cryptographic algorithm*, which was developed by IBM in combination with the NBS and published in 1977 as a US standard (FIPS 46) with the name 'Data Encryption Algorithm' (DEA).[13] The official successor to the DES is the → *AES*.

[13] See also Section 4.7.1, 'Symmetric cryptographic algorithms'

Design

In the context of software development, constructing a software architecture based on the requirements defined during → *analysis*. In simplified terms, the result of the design process is a description of 'how' the requirements are implemented in the software. In a sequential software development process, the subsequent stage is → *implementation*.

Deterministic

Designates a process or procedure that always produces the same result for a given set of initial conditions. It is the opposite of → *probabilistic*.

DF (dedicated file)

A directory in a smart card file system. The root directory (MF) is a special type of DF.

DF name

The DF name, like the file identifier (FID), is a DF attribute with a length of 1–16 bytes. It is used for selecting the DF, and it may contain a registered application identifier (AID), which has a length of 5–16 bytes and makes the DF internationally unique.[14]

Die, dice

A die (plural 'dice') is a small, flat piece of crystalline silicon on which a single semiconductor integrated circuit (such as a microcontroller) has been fabricated.

Differential fault analysis (DFA)

The principle of differential fault analysis was published in 1996 by Dan Boneh, Richard A. DeMillo and Richard J. Lipton, all of whom were employees of Bellcore [Boneh 96]. The method is based on intentionally introducing scattered errors into a cryptographic computation in order to determine the secret key. In the original method, only public-key algorithms were named, but within a few months this method of attack was rapidly extended [Anderson 96a], with the result that all cryptographic algorithms can in principle be attacked in this manner if they do not employ protective measures.

Differential cryptanalysis

A computational method for determining the value of a secret key using plaintext–ciphertext pairs having certain differences but the same key. The manner in which these differences

[14] See also Section 16.6, 'Registration Authorities for RIDs'

propagate with further DES cycles is analyzed to determine the key. This method was published by Eli Biham and Adi Shamir in 1990.

Digital

Designates a system in which signals can assume only a limited number of values.

Digital fingerprint

A commonly used designation for the hash value of a message (e.g., generated using the SHA-1 algorithm).

Digital signature

Digital signatures are used to establish the authenticity of electronic messages and documents. They are usually based on asymmetric cryptographic algorithms, such as the RSA algorithm. The legal validity of digital signatures is governed by legislation in many countries (such as the → *Signaturgesetz* in Germany). Digital signatures are sometimes referred to as 'electronic signatures'.

Digital watermark

A marking in an image or audio file, ideally invisible or inaudible, that cannot be removed and is used to protect proprietary rights. An analysis program can be used as necessary to check image or audio files for the presence of digital watermarks. Steganographic methods (→ *steganography*) are often used to generate digital watermarks.

Directory service

A service in a database that provides requestors with lists containing specific information. A typical example of such lists is a → *certificate revocation list*, which identifies all certificates that are no longer valid or accepted in a → *PKI*.

Divider

A short form for 'clock-rate conversion factor' (CRCF), which is commonly used in the smart card world. The CRCF specifies the duration of one bit interval during data transmission, in terms of the number of periods of the signal on the clock line.

Downlink

A connection from a higher-level system (such as a base station) to a lower-level system (such as a mobile telephone); the opposite of → *uplink*.

Download

Transferring data from a higher-level system (background or host system) to a lower-level system (e.g., a terminal); the opposite of → *upload*.

DPA (differential power analysis)

A method of attacking smart cards that represents an improvement on simple power analysis (→ *SPA*). It involves first making repeated measurements of the current consumption of a microcontroller for certain operations using known data with high time resolution and eliminating random noise by averaging. Following this, the current consumption is measured using unknown data, and conclusions regarding the unknown data are then drawn by analyzing the differences between the results for the known and unknown data. DPA was first made known in a publication by Paul Kocher, Joshua Jaffe and Benjamin Jun in June 1998 [Kocher 98].[15]

DRAM (dynamic random access memory)

A type of RAM having a dynamic structure that requires a continuous supply voltage and periodic refreshing to retain its content. DRAM cells are effectively capacitors. DRAM occupies less space on the chip than SRAM and is thus less expensive, but SRAM has shorter access times.

Dual-band mobile telephone

A mobile telephone that can operate in two different frequency bands (e.g., 900 MHz and 1800 MHz).

Dual-interface card

Designation for a → *smart card* having both contactless and contact-type interfaces for data transmission to and from the card.

Dual-mode mobile telephone

A mobile telephone that can operate in two different mobile telecommunication systems (e.g. GSM and AMPS).

Dual-slot mobile telephone

Designation for a mobile telephone having a second, externally accessible card contact unit, usually for an ID-1 → *smart card*, in addition to the contact unit for the user card (i.e., the

[15] See also Section 8.2.4.1, ‘Attacks at the physical level’

SIM). A dual-slot mobile telephone could for example be used with an existing smart card electronic purse to make payments via the mobile telecommunication network.

Dual-slot solution

A smart card application based on using the second card contact unit in a dual-slot mobile telephone.

Duplicating

Transferring genuine data to a second card with the objective of producing one or more identical (cloned) cards. Generally synonymous with → *cloning*.

Dynamic STK (dynamic SIM Application Toolkit)

An outmoded expression for microbrowser solutions (→ *browser*) that are compliant with the → *SIM Alliance* specification.

ECBS (European Committee for Banking Standards) [ECBS]

A European organization founded in 1992 to develop technical solutions and standards for the infrastructure of → *interoperable* trans-European financial transaction systems.

ECC (elliptic curve cryptosystem)

Designation for a cryptographic system (generally speaking, a cryptographic algorithm) based on elliptic curves.

ECC (error correction code)

A data checksum. An ECC can be used to allow errors in the data to be detected with a certain probability and in some cases fully corrected.

E-commerce (electronic commerce)

Refers to all forms of service, trade and associated financial transactions using public networks (primarily the Internet). The term → *m-commerce* is used when mobile terminals are used for e-commerce.

EDC (error detection code)

A data checksum. An EDC can be used to allow errors in the data to be detected with a certain probability. Typical examples of EDCs are the XOR and CRC checksums used in various data transmission protocols.

EDGE (Enhanced Data Rates for GSM and TDMA Evolution)

EDGE is intended to be the final evolutionary step for GSM networks. The EDGE specification allows a GSM mobile telephone to connect to a base station with a data rate of up to 384 kbit/s by using a different modulation scheme, without altering the existing network infrastructure.

EEPROM (electrically erasable programmable read-only memory)

A type of non-volatile memory, which is used in → *smart cards*. An EEPROM is divided into 'pages' of memory, with the page size being called its → *granularity*. The content of a memory page can only be altered or erased as an entity, and there is a physically determined upper limit to the number of write or erase cycles.[16] Data storage in an EEPROM cell is based on the Fowler–Nordheim effect, rather than hot electron injection as with → *Flash EEPROM*. The typical write time for EEPROM is 3 ms per memory page.

EF (elementary file)

The actual data storage element in a smart card file tree. An EF has either the attribute 'working' (for use by the terminal) or 'internal' (for use by the smart card operating system), and an internal structure (transparent, linear fixed, linear variable, cyclic, etc.).[17]

Electronic check

An → *electronic purse* variant using fixed, non-divisible monetary amounts. This type of payment is often referred to as 'pay before'.[18]

Electronic purse (e-purse)

A card with a chip that must be loaded with an amount of money before it can be used for making payments. This type of payment is often called 'pay before'. Some typical examples are the German Geldkarte, the Austrian Quick purse, Visa Cash, Proton and Mondex. Electronic purses may also support → *purse-to-purse transactions*.[19]

[16] See also Section 3.4.2, 'Memory types'
[17] See also Section 5.6.4, 'EF file structures'
[18] See also Section 12.1.2, 'Electronic money'
[19] See also Section 12.1.2, 'Electronic money'

Embossing

Part of the physical personalization of a card, consisting of raised characters stamped into the plastic card body.

Emulator

A device that imitates the operation of some other device or equipment (the target system). An emulator implemented in software is called a → *simulator*. Emulators are frequently used in developing software for not yet existing target systems. A smart card emulator is thus hardware circuitry that completely imitates the electrical and logical properties of a real smart card. Since the majority of the functionality is implemented in hardware, emulators are usually faster (closer to real-time) than simulators.

EMV (Europay, MasterCard, Visa) [EMV]

A joint specification for payment cards with chips and associated terminals belonging to Europay, MasterCard, Visa and American Express. These specifications have achieved the status of international industry standards for credit and debit cards and electronic purses. In the payment system sector, they thus represent the counterpart to the GSM 11.11 telecommunications standard.

EMV specification

→*EMV*

End-to-end link

Direct communication between two parties using the communication paths of one or more other entities that do not alter the information content of the actual data exchange. If the messages exchanged by the two originating parties are cryptographically secured, the term → *tunneling* is used. A typical example of an end-to-end link is direct communication between an application provider and a SIM that is compliant with GSM 03.48.

Endianness

The term 'endianness' refers to the order of the bytes within a byte string. 'Big-endian' means that the most significant byte stands at the beginning of the byte string, which consequently means that the least significant byte stands at the end of the string. 'Little-endian' refers to the opposite order, which means that the least significant byte comes first and the most significant bit comes last.

Enrollment

The process of originally acquiring the biometric data of a → *cardholder* and entering it into the corresponding smart card. The data stored in the smart card then form the basis for subsequent biometric user identification.

Envelope stuffing

Automatically folding letters and inserting them into envelopes.

EP SCP

→*SMG9*

EPROM (erasable programmable read-only memory)

A type of non-volatile memory, which was formerly used in smart cards but has been fully supplanted by → *EEPROM* technology. Since EPROM can only be erased by ultraviolet light, it can only be used for WORM storage (write once, read multiple) in smart cards.[20]

Error counter

A counter that accumulates negative results and determines whether a particular secret (PIN or key) may continue to be used. If the error counter reaches its maximum value, the secret is blocked and can no longer be used. The error counter is normally reset to zero when the operation is completed successfully (positive result). Also called a retry counter.

ETS (European Telecommunication Standard)

Designation for standards issued by → *ETSI*, which are primarily concerned with European telecommunications.

ETSI (European Telecommunications Standards Institute) [ETSI]

The standards institute of the European telecommunication companies, with headquarters in Sophia Antipolis, France. ETSI is responsible for defining standards in the field of European telecommunications. The most important ETSI standards are the family of standards for GSM (e.g., GSM 11.11 for the SIM) and UMTS (e.g., TS 31.102 for the USIM). Meetings of the expert groups of the ETSI are usually held in a wide variety of (touristically attractive) locations in Europe and throughout the world, for which reason some people are convinced that the abbreviation 'ETSI' stands for 'European travel and sightseeing institute'.

[20] See also Section 3.4.2, 'Memory types'

etu (elementary time unit)

The duration of one bit in smart card data transmission. The length of the etu is not defined in absolute terms, but instead in terms of the frequency of the clock signal applied to the card and the value of the clock-rate conversion factor (divider).

Eurosmart [Eurosmart]

An organization founded in 1994 to represent the interests of European manufacturers of smart cards, with offices in Brussels. The functions of Eurosmart are promoting and standardizing (→ *standard*) → *smart cards* and smart card systems, providing a forum for exchanging market data and technical data, and forging links to national and international standards committees.

Evaluation

The unbiased, objective, repeatable and reproducible assessment of an information technology system (hardware and/or software) by a reliable body according to the specifications of a criteria catalog. The IT system to be evaluated is called the → *target of evaluation*. Commonly used international criteria catalogs for the evaluation of → *smart cards* are the → *ITSEC* and → *Common Criteria*.

f1, f2, f3, f4, f5 (function 1 – function 5)

Designations for cryptographic functions used in → *UMTS* for authenticating the network and the → *USIM* and establishing cryptographically secured data transmission on the air interface. The central element of these security functions is a symmetric → *cryptographic algorithm* that can be parameterized using supplementary linked initial values. As an example algorithm for f1–f5, the USIM specification proposes the MILENAGE algorithm, which is essentially based on the → *AES*.

Fab

A semiconductor fabrication facility.

Face

The face of a semiconductor chip is the side holding the functional structures produced using semiconductor fabrication processes. Consequently, an expression such as 'face-to-face contacting' means that two chips with suitably configured functional structures are placed together such that they are electrically connected to each other.

FAT (file allocation table)

A table used in a file management system in which the storage area to be managed is divided into sections, called ‘clusters’. Data related to the occupancy and addresses of these sections are stored and managed using the file allocation table.

Fault tree analysis

A test method in which every program execution path in the program code is traversed in order to search for possible errors.

FD/CDMA (frequency division / code division multiple access)

→*CDMA*

FDMA (frequency division multiple access)

Ag →*multiple-access method* for concurrently transferring data from several transmitters to a single receiver using several different frequency bands. Each transmitter is allocated a particular frequency band within the total available frequency spectrum, within which it may exclusively transmit. Many mobile telephone systems (such as the German C-Netz) use FDMA for the air interface between the mobile telephone and the basis station.[21]

FIB (focused ion beam)

A device for generating a focused beam of ions for removing or depositing material on a semiconductor device.

FID (file identifier)

A two-byte attribute of a file. Each MF, DF and EF has a FID. The FID of the MF is always '3F00'.[22]

File body

→ *file header*

[21] See also Section 13.1.1, ‘Multiple-access methods’
[22] See also Section 5.6, ‘Smart Card Files’

File header

Files in smart cards are usually divided into two separate parts, consisting of the file header (which holds information about the → *file structure* and → *access conditions*) and the file body, which is linked to the file header by a pointer and holds the modifiable user data.

File structure

The externally visible structure of an → *EF*. File structures allow user data to be stored in a logically structured and compact manner. The standard file structures defined by ISO/IEC 7816-4 are transparent, linear fixed, linear variable and cyclic.[23]

File type

Identifies the sort of file for purposes of file management within a smart card, i.e., whether it is a directory file (MF or DF) or a file for storing user data (EF).

FIPS (Federal Information Processing Standard)

US American standards issued by the → *NIST*.

Firewall

An entity (hardware or software) that provides a security barrier between particular → *applications* or other entities. For example, a firewall can separate two applications in a smart card such that they cannot access each other's data across the firewall. The name comes from a type of wall used in building construction to contain possible fires.

Flash

Commonly used short form for → *Flash EEPROM*.

Flash EEPROM (Flash electrically erasable programmable read-only memory)

A type of non-volatile memory, which will be used in smart cards in the future. A Flash EEPROM resembles an → *EEPROM* in terms of its functionality and semiconductor structure, but data storage in Flash EEPROM cells is based on hot-electron injection, instead of the Fowler–Nordheim effect as in regular EEPROMs. In the hot-electron injection process, 'fast' electrons are generated by a high potential difference between the source and the drain, and some of these fast electrons penetrate the tunnel oxide layer and are stored in the floating

[23] See also Section 5.6.4, 'EF file structures'

gate. This effect reduces the write time to approximately 10 µs. Due to their large memory pages (typically 128 bytes at present), Flash EEPROMs are quite suitable for replacing mask-programmed → *ROM*.[24]

Floor limit

Defines the level at which a purchase must be authorized (→ *authorization*) by a third party. Authorization is not required below the floor limit, but it must always be obtained above the floor limit, since otherwise payment may not be possible or guaranteed.

Footprint

More precisely, 'memory footprint': refers to the allocation of memory for a particular purpose.

Foundry

A semiconductor fabrication facility operating on a contract basis to manufacture semiconductor devices developed by third parties.

FPLMTS (Future Public Land Mobile Telecommunications Service)

→ *IMT-2000*

FRAM (ferroelectric random-access memory)

A type of non-volatile memory, which is very rarely used in → *smart cards*. A FRAM is divided into memory pages (the page size is also called the → *granularity*). Data storage in this type of memory is based on the properties of a ferromagnetic substance placed between the control gate and the floating gate. FRAM cells typically have a write time of 100 ns per page and do not require a special erase voltage. However, the number of erase cycles is limited, and manufacturing FRAM involves processes that are difficult to master. Consequently, it has been used only rarely in smart card microcontrollers up to now.

Frame

A sequence of data bits and optional error detection bits bounded by frame delimiters. Frames for contactless data transmission with smart cards are defined in ISO/IEC 14 433.

[24] See also Section 3.4.2, 'Memory types'

Full duplex

Data transmission method in which each of the communicating parties can transmit and receive concurrently. (→ *half duplex*)

Garbage collection

A function that collects memory no longer used by an → *application* and makes it available as free memory. In the past, garbage collection was implemented by interrupting regular program execution. In modern computer systems, garbage collection is a low-priority thread that constantly searches the memory for regions that are no longer needed and returns them to the free memory pool.

Geldkarte

Brand name of an electronic purse introduced in Germany in 1996. 'Geldkarte' refers to both the application in a → *multiapplication smart card* and the smart card itself. The smart card operating system used for the Geldkarte or debit functionality is → *SECCOS*.

Glitch

A very short voltage dropout or voltage spike.

Global Platform [Global Platform]

An internationally active association founded in 1999 by various smart card companies to standardize technologies for → *multiapplication smart cards*. The most important specification published by Global Platform is the Open Platform specification (→ *OP*).

Good case

The case in which a logical decision yields a favorable or intended result.

GPRS (General Packet Radio System)

An extension of → *GSM*, standardized by → *ETSI*, for achieving higher data transmission rates with mobile telephones. GPRS provides a packet-switched connection with a data transmission rate of up to 115.2 kbit/s by bundling the eight available time slots, each of which has a capacity of 14,400 bit/s. A mobile telephone with GPRS technology is constantly connected to the network with respect to data transport and thus always available for data transmission. The data transmission rate is dynamically adapted to the currently required capacity, so only the capacity actually needed is used. For this reason, GPRS is very suitable for discontinuous data transfers.

Granularity

A frequently used alternative term for expressing the page size of an → *EEPROM*. For example, an EEPROM with a granularity of 32 has a page size of 32 bytes.

Graybox test

A mixed form combining elements of blackbox and whitebox tests, in which the party performing the test knows some but not all of the internal processes, functions and mechanisms of the software being tested.

Graylist

A list in a database identifying all → *smart cards* or devices that are under observation. (→ *blacklist, hotlist, white list*)

GSM (Global System for Mobile Communications)

A digital, cellular, interoperable, transnational and ground-based second-generation (→ 2*G*) mobile telecommunication system. The frequency bands allocated to this mobile telecommunications system are 900 MHz (GSM 900), 1800 MHz (GSM 1800) and 1900 MHz (GSM 1900). The GSM system is defined by a family of specifications published by → *ETSI*. The alliance of the major network operators and manufacturers is the → G*SM Association*. Originally, GSM was only planned to be used in certain central European countries as a successor to country-specific analog mobile telephone systems. However, it has developed into an international standard for mobile telecommunication systems. Due to the low data transmission rates of the GSM system (9600 bit/s and 14,400 bit/s), improvements to the system have become necessary. The evolutionary path of the GSM system with respect to data transmission capacity thus envisages circuit-switched → *HSCSD* and packet-switched → *GPRS* as the next steps in the further development of the system. Afterwards, the GSM data transmission rate can be further increased using → *EDGE* technology. The designated successor to GSM is → *UMTS*.[25]

GSM Association [GSM Association]

An internationally active body for coordinating mobile telecommunications systems, with offices in Dublin and London. It was founded in Copenhagen in 1987 and is responsible for the development and use of → *GSM* standards. The GSM Association represents more than 500 network operators, manufacturers and suppliers in the GSM industry.

[25] See also Section 13.3, 'The UMTS System'

Guilloches

Decorative patterns of interwoven lines, usually circular or oval, found on many banknotes and share certificates. Due to their fine structures, these patterns can only be reproduced at high quality using printing techniques, so they are difficult to copy.

HAL (hardware abstraction layer)

An intermediate layer in an operating system, which is used to conceal all hardware-specific features of the target platform from the rest of the operating system. The objective of this is to markedly simplify porting of the operating system, since changing the hardware platform only requires modifications within the HAL.[26]

Half-byte

→ *nibble*

Half duplex

Data transmission method in which each of the communicating parties cannot concurrently send and receive data. A → *full-duplex* connection is required for concurrent transmission and reception.

Handover

In a mobile telecommunication network, the interruption-free transfer of a mobile telephone from one cell to the next. In GSM, a handover is always initiated by the network.

Happy-day scenario

Another expression for → *good case*.

Hard mask

The term 'hard mask' means that the entire → *program code* is predominantly stored in ROM (→ *ROM mask*). This saves space compared with a soft mask, since ROM cells are significantly smaller than EEPROM cells. However, it has the disadvantage that the full duration of the process of producing a customer-specific semiconductor device is required to generate hard-masked microcontrollers. Consequently, the lead time for a hard mask is significantly longer than for a soft mask. Hard masks are normally used with large numbers of chips for smart cards having largely common functionality. The opposite of a hard mask is a → *soft mask*, which involves storing essential functions in EEPROM.

[26] See also Section 5.2, 'Basics'

Hash function

A hash function is a procedure for compressing data using a one-way function such that it is not possible to recompute the original data. A hash function produces a fixed-length result for an input with any arbitrary length, and it is designed so that any change to the input data has a very high probability of affecting the computed hash value (output data). SHA–1 is a typical representative of hash algorithms. The result of a hash function is a hash value, which is often also referred to as a digital fingerprint.[27]

HBCI (Home Banking Computer Interface)

A standard defined by the German banking industry for the implementation of home banking in Germany, with optional smart card support.

Hologram

A photographic exposure made using a holographic process. It produces a three-dimensional image of the photographed object. The object in the photograph can thus be seen from different angles, depending on the viewing angle of the observer. The holograms normally used with smart cards are embossed holograms, which produce reasonably satisfactory three-dimensional images under normal lighting conditions.

Home net

With respect to a customer of a mobile telecommunications system, the mobile telecommunication network operated by the company for which he or she is a customer.

Home zone

In a mobile telecommunications network, a → *location-based service* in which calls are charged at a significantly lower rate (normally the fixed-network rate) within a certain region (usually the immediate vicinity of the user's residence). As a result, the subscriber may not need to have a connection to the fixed telephone network.

Horizontal prototype

→ *prototype*

Hotlist

A database list of → *smart cards* and devices that probably have been manipulated and must not be accepted under any circumstances. (→ *blacklist, graylist, white list*)

[27] See also Section 5.2, 'Hash Functions'

HSCSD (High-Speed Circuit-Switched Data)

The circuit-switched HSCSD technology is an extension to the GSM standard for increasing the data transmission rate over the air interface to a theoretical value of 76,800 bit/s (8×9600 bit/s) for uplink or downlink by supplementary utilization of existing time slots. Existing GSM networks can be extended to support HSCSD at a relatively low cost by upgrading the base stations and using special mobile telephones. The drawback is that the demand for transmission channels can increase by as much as factor of eight.

HSM (hardware security module, host security module)

→ *security module*

HTML (hypertext markup language)

A logical markup language for hypertext documents in the WWW, which is conceptually based on XML. (→ *WML, WWW, XML, hypertext*)

Hybrid card

A card having two different card technologies. Cards having both magnetic stripes and chips, and smart cards with optical storage on the card surface, are typical examples.

Hypertext

Compared with normal text, hypertext has supplementary cross-references (hyperlinks) to other locations in the text or to other documents. These cross-references can be invoked by suitable user actions (usually by clicking on them). As opposed to normal linearly structured text, such as in books, hypertext allows any desired interlinking of texts to be achieved using cross-references. Typical examples of markup languages for hypertext documents are → *HTML* and → *WML.*

ICC (integrated chip card)

The official ISO name for a card with a chip, which may be a memory chip or a microcontroller chip.

ID-l card

Standard format for → *smart cards* as specified by ISO 7810 (length ≈85.6 mm, width ≈54 mm, thickness ≈0.76 mm).[28] However, the ID-000 (plug-in) format is predominately used in the mobile telecommunications area.

Identification

The process of verifying the authenticity of a device or a person by comparing a password provided by the device or person to a stored reference password. Identification can be considered to be a special case of → *authentication*, in which the identity of a person is authenticated. The method used for identification is sometimes referred to as the → *cardholder verification method*.

IEC (International Electrotechnical Commission) [IEC]

The IEC was founded in 1906 and is based in Geneva, Switzerland. Its function is to generate international standards for electrical and electronics technology.

IFD (interface device)

The official ISO name for a smart card terminal.

Implanter

A production machine for smart cards whose function is to insert modules in the cavities of smart cards, which is called 'implanting' in trade jargon.

Implementation

In the context of software development, producing a program on the basis of a software architecture defined in the → *design* stage. Implementation also includes debugging, but not testing (→ *test*), which occurs in the subsequent stage in a sequential software development project.

IMSI catcher

A device that taps GSM conversations by setting up its own cell. An IMSI catcher works by interposing itself between the mobile telephone and the base station; it represents itself as a base station with respect to the mobile telephone and as a mobile telephone with respect to the base station.

[28] See also Section 3.1.1, 'Card formats'

IMT-2000 (International Mobile Telecommunication 2000)

A concept of the → *ITU* for third-generation (→ *3G*) mobile telecommunications systems operating in the 2000-MHz frequency band. IMT-2000 arose in 1995 as a successor to the Future Public Land Mobile Telecommunication Service (FPLMTS), a mobile telecommunications concept initiated by the ITU in 1985 that failed to be translated into reality as an international standardized system in its original form. One possible realization of IMT-2000 is → *UMTS*.

Individualization

→ *personalization*

Initializer

An entity that performs →*initialization*.

Initialization

The process of loading the fixed, person-independent data of an → *application* into EEPROM. A synonym for initialization is 'pre-personalization'.

Instrumenting

Introducing special → *program code* into a program in order to allow the procedures and calls of the program to be analyzed for test purposes.[29]

Intelligent memory card

A memory card having additional logic circuitry for supplementary security functions that monitor memory accesses.

Internal foil

A foil located inside the stack of foils laminated together to make a card body; synonymous with 'core foil'. Normally, an internal foil is laminated between two outer (cover) foils, with these three foils together forming the card body. The internal foil often carries security features or electrical components, such as the coil for a contactless smart card.

[29] See also Section 9.3.3, 'Dynamic testing of operating systems and applications'

Interoperable

This adjective is used in the smart card world to designate solutions that are not tailored to a particular smart card → *application* or the equipment of a particular manufacturer. An → *open smart card operating system* is usually interoperable. The opposite of an interoperable solution is a → *proprietary* solution. An example of an interoperable smart card is the SIM, which can be used equally well in all types of GSM mobile telephones without compatibility problems.

Interpreter

A program that translates the instructions of a programming language such as Basic or Java into machine-language instructions that can be executed by a microprocessor, and immediately executes each instruction after it has been translated. Interpreted programs always run more slowly than compiled → *program code*, since the translation occurs at run time. However, a significantly higher level of hardware independence in programming is possible with interpreted code than with compiled code.

ISDN (Integrated Services Digital Network)

Designation for an internationally standardized digital telephone network that supports both telephone conversations and data transmission. An ISDN link consists of two base channels, each having a transmission rate of 64 kbit/s, and a control channel with a transmission rate of 16 kbit/s.

ISO (International Organization for Standardization) [ISO]

ISO was founded in 1947 and is based in Geneva, Switzerland. Its function is to support the generation of international standards in order to promote the free exchange of goods and services. The first ISO standard was published in 1951 and deals with temperatures with regard to length measurements.

ITSEC (Information Technique System Evaluation Criteria)

A catalog of criteria for the → *evaluation* and certification of the security of information technology systems in Europe, published in 1991. The → *Common Criteria* resulted from refining the ITSEC and combining the ITSEC with various national criteria.

ITU (International Telecommunications Union)

An international organization for the coordination, standardization and development of global telephone services, based in Geneva. It is the successor to the CCITT.

Java

A hardware-independent, object-oriented programming language (→ *object-oriented programming*) developed by the Sun Corporation, which is widely used on the Internet. Java source code is translated by a compiler into standardized bytecode, which is then usually interpreted by a virtual machine based on the target hardware (Intel, Motorola, etc.) and operating system (Windows, MacOS, Unix, etc.) platforms. There are also microprocessors (such as picoJava) that can directly execute Java bytecode.

Java card, Java Card

A Java card is a → *smart card* with a → *microcontroller* containing a → *Java Card virtual machine* and a → *Java Card runtime environment*. Java cards are → *multiapplication smart cards* incorporating the Java Card operating system, which can manage and run programs written in Java. Strictly speaking, Java Card is not a true → *operating system*, in part because the original specification does not include file management. However, in practice Java Card is considered to be the archetype of an → *open smart card operating system*.

Java Card Forum [JCF]

An internationally active organization founded by several smart card companies in 1997 to promote Java Card technology and develop related specifications (→ *Java Card*).

Java Card runtime environment (JCRE)

The Java Card runtime environment essentially consists of the → *Java Card virtual machine* (JCVM) and the Java Card API.

Java Card virtual machine (JCVM)

(→ *virtual machine*) A simulation of a microprocessor (usually implemented in software) whose function is to execute Java bytecode and manage Java classes and objects. The Java Card virtual machine also ensures application separation by means of firewalls and allows common utilization of data. In principle, it can be regarded as a type of interpreter. A Java VM implemented using publicly accessible information, i.e. without using additional information subject to a licensing agreement with Sun, is called a 'cleanroom VM'. Cleanroom implementations of the Java VM are generally considered to be free of any obligation to pay licensing fees to Sun.

Java development kit (JDK)

A collection of software tools supporting the development of Java software.

Kerckhoff's principle

A principle named after August Kerckhoff (1835–1903) that asserts that the entire security of a cryptographic algorithm should be based exclusively on the confidentiality of its key, rather than the confidentiality of the algorithm.

Kernel

The central part of a → *operating system*, which provides basic operating system functions to the overlying layers of the operating system.

Key

For a → *cryptographic algorithm*, the parameter that individualizes the encryption or decryption process. With a symmetrical cryptographic algorithm that is used to ensure security, the key must be secret, but the public key of an asymmetric cryptographic algorithm may be generally known.

Key fault presentation counter

→ *error counter*

Key management

Collectively, all administrative functions used for generating, distributing, storing, updating, destroying and addressing cryptographic keys.

Kinegram

A kinegram shows different images when viewed at different angles. It can show an apparently 'moving' image that changes in jerks, or it can show completely unrelated images at different viewing angles. Kinegrams are similar to holograms, which show three-dimensional images, but are not identical to them.

Lamination

The process of gluing together thin sheets of material using heat and pressure. Cards are generally laminated from several plastic foils.

Laser cutter

A device for drilling and cutting, preferably on a semiconductor chip, with a precision of a fraction of a micrometer using a high-energy laser beam.

Laser engraving

A process for blackening special plastic layers by ‘burning’ them with a laser beam. This is also colloquially referred to as ‘lasing’.

Lead-frame module

A type of low-cost module having contacts stamped from a copper alloy electroplated with a gold film and held together by a plastic mold body. A chip is placed on the lead-frame module by a pick-and-place robot and electrically bonded to the rear surfaces of the contacts using wire bonding. After this, the chip is covered by a blob of opaque epoxy resin for its protection.

Lead-frame process

Currently one of the least expensive ways to produce modules without incurring penalties with regard to mechanical stability.

Lead time

In semiconductor fabrication, the time between when the mask is provided (→ *ROM mask*) and the time when the first samples are ready.

Life cycle

The aggregate of the stages in the life of a → *smart card*, beginning with the production of the chip and the card, progressing through → *personalization* and use and ending with the logical or physical end of the card’s life. The individual stages in the smart card life cycle are used to define specific security measures and functionalities. An example of the partitioning of the life cycle of a card is the→ *Open Platform* specification.

Life cycle model

A model, sometimes referred to as a process model, that specifies, in abstract form, the organizational framework, work processes and activities of a development process, including the associated prerequisites and results. The objective is to achieve a uniform, general-purpose approach to software development. Some examples of life-cycle models are the waterfall model, the V model and cyclic development models.[30]

[30] See also Section 15.7, ‘Life-Cycle Models’

Linker

The function of a linker is to convert the symbolic memory addresses of compiled or assembled program code into absolute or relative memory addresses.

Little-endian

→ *endianness*

Load agent

An entity that loads electronic money into an electronic purse. In a manner of speaking, a load agent is the counterpart of a service provider.

Loader

A program that can be used to load other programs (→ *boot loader*), for example via a serial interface.

Location-based services

Value-added services for mobile telephone subscribers that are based on knowledge of the subscriber's current geographic position. Some examples are local weather forecasts, city maps that are dynamically adapted to the user's current location and integrated location data for service calls.

Logical channels

Logical channels allow data to be exchanged concurrently and independently with several → *applications* in a smart card. Although communication with the smart card still takes place via the single serial interface in the card, logical channels allow the applications in the smart card that receive the → *APDUs* to be individually addressed.[31]

M-commerce (mobile commerce)

Collective term for all types of services, trade and associated financial transactions using mobile terminals (such as mobile telephones and PDAs). If fixed terminals are used, the term → *e-commerce* is used.

[31] See also Section 6.7, 'Logical Channels'

M/Chip

The name given to an → *EMV*-compliant implementation of a chip-based debit/credit card from Europay and MasterCard. The M/Chip Select version uses both symmetric and asymmetric cryptographic algorithms and is a superset of M/Chip, which is a simplified version that uses only symmetric cryptographic algorithms.

MAC (message authentication code)

A cryptographic checksum for data that allows manipulation of the data during transmission to be detected. An equivalent checksum used to protect stored data is called a CCS (→ *cryptographic checksum*).

Magnetic card

A commonly used but technically incorrect short form of → *magnetic-stripe card*.

Magnetic-stripe card

A card with a magnetic stripe for recording and subsequently reading data. The magnetic stripe usually has three data tracks with different data recording densities. Tracks 1 and 2 are used only for reading after the card has been issued, but data may also be written to track 3 during normal use. The magnetic substance in the stripe may have either a high-coercivity characteristic or a low-coercivity characteristic.

Maosco

→ *Multos*

Mask

An abbreviated form of → *ROM mask*.

Memory card

A card with a chip that has a simple logic circuit along with memory that can be read and/or written. Memory cards can also have supplementary security logic units, which for example can allow the card to be authenticated.

Memory footprint

The structure of memory allocation in a computer system.

Merchant card

In an electronic payment system, a → *smart card* located in a merchant terminal and serving as a security module.

Method

In the context of → *object-oriented programming*, a function used to alter the values of the → *attributes* of an → *object*, which is generated by the → *class* of the object.

MexE (Mobile Station Execution Environment)

A Java framework for integrating a Java virtual machine (JVM) into a mobile telephone. It allows Java programs to be loaded into the mobile telephone and executed. This allows supplementary applications to be implemented directly in the mobile telephone, rather than in the SIM (as in current practice).

MF (master file)

The master file of a smart card file system is a special type of DF. It is the root directory of the file tree and is automatically selected after the smart card has been reset.

Microbrowser

→ *browser*

Microcontroller

A microcontroller consists of a → *microprocessor*, volatile memory (→ *RAM*), non-volatile memory (→ *ROM*, → *EEPROM*, → *Flash EEPROM*) and suitable interfaces for off-chip communications, all integrated into a single chip. It is thus a self-contained and fully functional computer on a single chip. Microcontrollers are primarily used in smart cards and control technology.

Microprocessor

The most important component of a → *microcontroller*. The microprocessor resolves the machine instructions specified by the program code into microinstructions and executes the microinstructions. A microprocessor contains the registers needed for instruction processing, a control mechanism and a processing unit. The actual processing unit of a microprocessor is sometimes simply called the 'processor'. The term 'central processing unit' (CPU) is often used as a synonym for 'microprocessor'.

Microprocessor card

A card containing a → *microcontroller* with a CPU, volatile memory (RAM) and non-volatile memory (→ *ROM*, → *EEPROM* etc.). A microprocessor card may also contain a numeric coprocessor (→ *cryptoprocessor*) to quickly execute public-key cryptographic algorithms. Such a card is sometimes called a cryptocard or cryptocontroller card.

MILENAGE algorithm

The symmetrical sample algorithm for the f1–f5 (→ *f 1*) functions of → *USIM*. The kernel of the MILENAGE algorithm is based on the → *AES*.

MKT (*Multifunktionales Kartenterminal*)

The German abbreviation (and name) of a specification for multifunctional smart card → *terminals* and the connections to such terminals using the → *CT-API* interface specification. It supports both → *memory cards* and → *microprocessor cards*. The specification is published by Teletrust Deutschland.

Module

→ *chip module*

Module manufacturer

An entity that attaches dice to blank modules and electrically connects each die to the module contacts.

Mondex [Mondex]

An → *electronic purse* system using smart cards that allows → *purse-to-purse transactions*.

Mono-application smart card

A smart card containing only one → *application*.

Monofunctional smart card

A microprocessor card whose operating system supports only one specific → *application*, and which may even be optimized for this application. Such cards provide little or no support for administrative functions, such as file creation and deletion.

Monolayer card

A card composed of only one layer of plastic (→ *multilayer card*).

MoU (Memorandum of Understanding)

The common legal basis for all GSM network operators. The organization behind the MoU is the → *GSM Association*.

Multiapplication smart card

A smart card containing several → *applications*, such as a bank card with a phone-card function.

Multifunctional card (MFC)

Usually, a microprocessor card that supports multiple → *applications* and has corresponding administrative functions for storing and deleting applications and files.

Multilayer card

A card made up of several layers of plastic foil, consisting of the outer or cover foils (overlay foils) and the inner foils (core foils). (→ *monolayer card*)

Multiple-access method

Any of several methods used in radio communications and information technology to make a limited frequency bandwidth concurrently or quasi-concurrently available to the largest possible number of users. The four commonly used multiple-access methods are frequency division multiple access (→ *FDMA*), time division multiple access (→ *TDMA*), code division multiple access (→ *CDMA*) and space division multiple access (→ *SDMA*).[32]

Multiple-copy sheet

In printing, a collection of small items (such as cards) printed on a single large sheet, which is divided into individual items after printing. This allows the printing process to be technically optimized, since many items can be printed in one pass on a large sheet instead of in several separate passes. For instance, a typical multiple-copy sheet for printing cards holds 42 cards on a large plastic sheet.

[32] See also Section 13.1.1, 'Multiple-access methods'

Multitasking

A computer system that supports multitasking allows several programs to be run quasi-concurrently. Each of the concurrently running programs is usually located in a separate address space that is protected against access by other programs, and it can exchange data with other programs only by means of special mechanisms. Multitasking is not the same as multithreading, in which a single program performs several different tasks quasi-concurrently. A computer system may support both multitasking and multithreading.

Multithreading

A computer system that supports multitasking allows a single program to perform several different tasks quasi-concurrently. The individual threads of a program normally use a common address space. Multithreading is not the same as multitasking, in which several different programs run concurrently, each with its own separate address space. A computer system may support both multitasking and multithreading.

Multos

Brand name of an open, multiapplication → *smart card operating system* (→ *open smart card operating system*).[33] The Maosco Consortium [Maosco] publishes the specification, licenses the software and operates the certification services for Multos.

Name space

A set of names in which all of the names are unique.

Native code

A program whose instructions are in the specific machine language of the microprocessor that executes the program.

NBS (National Bureau of Standards)

The name of the → *NIST* prior to 1988.

NCSC (National Computer Security Center) [NCSC]

The NCSC is a subagency of the US National Security Agency (NSA). It is responsible for testing security products and publishing criteria for secure computer systems, including the TCSEC.

[33] See also Section 5.14.2, 'Multos'

Negative file

→*blacklist*

Negative result

→*bad case*

Nibble

The four most significant or least significant bits of a byte; also called a half-byte.

NIST (National Institute of Standards and Technology) [NIST]

A section of the US Department of Commerce responsible for US national standards for information technology. The NIST, which was called the NBS until 1988, publishes the FIPS standards.

Noiseless

A property of a → *cryptographic algorithm* that always takes the same amount of time to encrypt or decrypt data, irrespective of the → *key*, plaintext and ciphertext involved. If a cryptographic algorithm is not noiseless, the size of the key space can be markedly reduced by analyzing the processing-time characteristics of the algorithm. This allows the key to be determined significantly faster than by using a brute-force attack.

Non-repudiation

A usually cryptographic method to ensure that the recipient of a message cannot refuse to acknowledge (repudiate) receipt of the message, thus enabling the sender of the message to prove that the intended recipient actually received the message. Non-repudiation is similar to a registered letter with return receipt in an ordinary postal system.

Non-volatile memory

A type of memory (such as ROM, EPROM or EEPROM) that retains its content even in the absence of power.

NPU (numeric processing unit)

→*cryptoprocessor*

NSA (National Security Agency) [NSA]

The official communications security agency of the US government. It reports directly to the Department of Defense, and one of its functions is to monitor and decode foreign communications. Developing new cryptographic algorithms and restricting the use of existing algorithms also fall under the authority of this agency.

Null PIN

→ *0−PIN*

Numbering

Embossing or printing a number on a smart card; typically used in the manufacturing of anonymous prepaid phone cards to give each card a visible, unique number so it can be unambiguously identified.

Object

In the context of → *object-oriented programming*, a software structure that is built according to the instructions defined by a → *class* and contains data, which means that it has → *attributes* that can be read and altered using the → *methods* defined in the class.

Object-oriented programming

Object-oriented programming is based on storing all of the data of a software application in → *objects*, which also provide → *methods* that can be used to read or modify the data. Objects are defined by → *classes*. A key aspect of object-oriented programming is that it focuses on the data to be processed, rather than on the processes as does → *procedural programming*. Some typical object-oriented programming languages are C++ and → *Java*.

OCF (Open Card Framework) [OCF]

The Open Card Framework specification describes a platform-independent, Java-based interface for integrating smart cards into any desired application on a PC. It presupposes the availability of a suitable driver for each type of terminal to be used with the PC in question, and that the smart cards are OCF-compatible.

Offcard application

→ *application*

Oncard application

→ *application*

Oncard matching

The ability of a → *smart card* to compare biometric data measured either oncard or offcard with reference stored in the smart card for the purpose of user → *identification*.

One-way function

A one-way function is a mathematical function that is easily computed but whose inverse function requires a large amount of computational effort.

OP (Open Platform)

Previously Visa Open Platform (VOP); an interface in a smart card operating system, originally specified by Visa International, that supports the management of smart card applications. The specification encompasses, among other things, downloading smart card applications, securing the application life cycle and linking a smart card application to the smart card operating system. The OP specification is effectively the international industry standard for multiapplication smart cards and application management. The current publisher of the OP standard is the Global Platform association.

Open application

An → *application* in a smart card that is available to a variety of service providers (such as merchants and vendors of services) without requiring a mutual legal relationship.

Open platform

→*open smart card operating system,* → *OP*

Open purse

An instance of an open → *application* for an → *electronic purse*. It can be used for general payment transactions with various service providers.

Open smart card operating system

A → *smart card operating system* is characterized as being open if it is possible for third parties to load applications and programs into the smart card and run them in a secure environment, all without the involvement of the → *operating system producer*. The three best-known open smart card operating systems are → *Multos*, → *Java Card* and → *Windows for Smart Cards*. Open smart card operating systems are usually → *interoperable*, rather than → *proprietary*. The term 'open platform' is also used to refer to an open smart card operating system, but it should not be confused with Open Platform (→ *OP*), which is an interface for managing applications in smart cards.

Operating system (OS)

An operating system encompasses all of the programs of a digital computer system that, in combination with the hardware features of the computer system, form the basis for its possible operational modes, in particular monitoring and controlling the execution of programs.[34]

Operating system producer

An entity that programs and tests an → *operating system*.

Optical memory card

A card in which information is 'burnt' into a reflective surface layer (similar to a CD).

OTA (over-the-air)

In → *GSM* and → *UMTS* systems, OTA refers to the possibility of establishing an → *end-to-end link* between the background system and the → *SIM* via the air interface between the base station and the mobile station. Such a link makes it possible to (for example) send a command directly and transparently from the background system to the SIM. OTA is also one of the foundations for all → *value-added services* in the SIM, since such services can also exchange data directly and transparently with higher-order systems via the air interface. The Short Message Service (→ *SMS*) is frequently used as the transport service (→ *bearer*) for OTA.

Package

Ag →*name space* in Java Card, and the smallest entity in the Java language. A package may have classes and interfaces.[35]

Packet-switched

With a packet-switched link, the sender partitions the data to be exchanged into packages, which are then transmitted individually to the recipient, possibly via separate paths. The recipient

[34] Based on the text of the German DIN 44 300 standard
[35] See also Section 5.14.1, 'Java Card'

then reassembles the packages to recover the original data. Charges for a packet-switched link usually depend on the amount of data exchanged, rather than the duration of the connection. Some typical examples of packet-switched links are X.25 and GPRS.

Padding

Extending a data string with filler data in order to bring it to a particular length. This is necessary if the length of the string must be an integral multiple of a certain block size (such as 8 bytes) to allow it to be further processed, for example by a cryptographic algorithm.

Page-oriented

A set of bytes in a memory that can only be written or erased as a group. In → *smart card microcontrollers*, only EEPROM and Flash EEPROM are page-oriented. Typical page sizes are 4, 32, 64 and 128 bytes. However, there are now microcontrollers with page sizes that are variable within a certain range, such as 1–128 bytes, instead of fixed.

Parallel data transmission

The concurrent transmission of several data bits (e.g. 8, 16 or 32) using a corresponding number of data lines. (→ *serial data transmission*)

Parity bit

Probably the best-known type of error detection code (EDC) is a parity bit appended to the byte to be protected. Before the parity bit can be calculated, it is necessary to specify whether even or odd parity is to be used. With even parity, the value of the parity bit is chosen such that the total number of bits with a value of 1 in the combined data byte and parity bit is an even number. With odd parity, the total number of bits with a value of 1 must be an odd number. With a single parity bit, one incorrect bit per byte can be reliably detected. However, it is not possible to correct a bit error, since the parity bit does not provide any information about the location of the altered bit.

Passivation

A protective layer on top of a semiconductor chip that screens it against oxidation and other chemical processes. The passivation layer must be partially or fully removed before the semiconductor can be physically manipulated.

Patch

In software development, a small program, sometimes written in machine code, that extends or alters the functionality of an existing program. Patches are commonly used to make quick, simple corrections to program errors. They can be implemented either as → *work-arounds* or as → *bug fixes*.

Patent

A document granting an inventor the right to the exclusive exploitation of an invention for a limited period in one or more countries. The maximum term of a patent is usually 20 years.

Pay before

This expression refers to money flow for cards used in payment systems. With pay-before, the real money flows out of the cardholder's account before the goods or services are actually purchased. A typical example of a pay-before card is an → *electronic purse*, which the user must load with electronic money before making purchases. In the telecommunications sector, this form of payment is called → *prepaid*.

Pay later

This expression refers to money flow for cards used in payment systems. With pay later, the real money flows out of the cardholder's account only some time after the goods or services are actually purchased. A typical example of a pay-later card is a credit card, for which it may take up to several weeks after a purchase before the money is transferred from the account of the purchaser to the account of the merchant.

Pay now

This expression refers to money flow for cards used in payment systems. With pay now, the real money flows out of the cardholder's account at the same time as the goods or services are purchased. A typical example of a pay-now card is a debit card, such as the Eurocheque card, which allows the money to be transferred from the account of the purchaser to the account of the merchant at the time that the purchase is made.

PC/SC (Personal Computer/Smart Card) [PC/SC]

The PC/SC specification describes an interface for integrating smart cards into any desired application, independent of the platform and programming language used. The prerequisites are that a suitable driver must be available for the terminal used with the PC, and the smart card must be PC/SC-compatible. Version 1.0 of the 'Interoperability Specification for ICCs and Personal Computer Systems' was published in December 1997.[36]

[36] See also Section 11.4.1, 'PC/SC'

PCD (proximity coupling device)

A card terminal for communicating with a contactless card. (→ *PICC*)

Persistent

Attribute of an object that continues to exist after its run time (as opposed to a → *transient* object). A persistent object thus continues to exist after the end of a session, as well as after a sudden loss of power, without any loss of data or data inconsistency.

Personalizer

An entity that performs → *personalization*.

Personalization

The process of associating a card with a person. This can be done using physical personalization (e.g. embossing or laser engraving) as well as by electronic personalization (loading personal data in the memory of the smart card). The term ‘individualization’ would be a more exact description of this process, since it is not always necessary to enter personal data into the chip when electronic personalization is performed, for instance in the production of anonymous → *prepaid SIMs*.

Phase 1, Phase 2, Phase 2+

These phases mark the successive evolutionary stages in the development of the GSM system. In Phase 1, the basic services were realized (including speech transmission, call forwarding, → *roaming* and → *SMS*). In Phase 2, which began in 1966, the Phase 1 services were augmented by additional services, including conference calls, call handoff, calling number conveyance and GSM in the 1800-MHz band. In Phase 2+, these services were extended with the functions of the → *SIM Application Toolkit*, HSCSD (High-Speed Circuit-Switched Data) and → *GPRS*, among other things.

PICC (proximity integrated-circuit card)

A contactless smart card with a range of approximately 10 cm.

PIN (personal identification number)

A secret number, usually consisting of four digits, used for the → *identification* of a person. In the telecommunications world, the designation ‘CHV’ (cardholder value) is usually used for the PIN.

PIN pad

Originally, a data-entry keypad with special mechanical and cryptographic protection for use in a terminal. In general usage, the entire terminal is often called a PIN pad.

PKCS #1/2/.../15 (Public Key Cryptographic Standard Number 1/2/.../15)

Public-key cryptography specifications published by RSA Inc. that focus on the use of asymmetric cryptographic algorithms, such as the RSA algorithm.[37]

PKI (public key infrastructure)

All of the facilities and systems needed to exchange and manage data using asymmetric cryptographic protection, including a → *certification authority*, a → *registration authority*, a → *directory service* for blacklists (→ *certificate revocation list*), a time-stamp service (→ *time stamp*) and → *signature cards*.

PLMN (public land-mobile network)

Technical term for a terrestrial mobile telecommunications system.

Plug-in card

A small-format smart card as specified in GSM 11.11 and TS 102.221, primarily used in the mobile telecommunications sector. The official ISO designation for this format is 'ID000', in contrast to the larger ID-1 format (→ *ID-1 card*) used for common smart cards. A plug-in card has a length of ≈25 mm, a width of ≈15 mm and a thickness of ≈0.76 mm.[38]

Polling

Periodic program-driven querying of an input channel in order to detect an incoming message. Depending on the repetition rate of the queries, polling can require significant processing capacity, for which reason it is usually avoided in favor of hardware-supported querying using interrupts. An example of polling is in mobile telephones, where it is used in connnection with the → *SIM Application Toolkit* to allow the → *SIM* to send proactive commands (→ *proactivity*) to the mobile telephone.

POS (point of sale)

The location where a particular item or service is sold.

[37] See also Section 4.7.2, 'Asymmetric cryptographic algorithms'
[38] See also Section 3.1.1, 'Card formats'

Positive result

→ *good case*

Postpaid

Refers to money flow for cards used in the telecommunications sector in which the real money of the cardholder flows only after the service (usually a telephone call or data transmission) has been received. With regard to their payment function, postpaid cards are comparable to credit cards. In payment systems, this form of payment is called → *pay later*.

Power-on reset

→ *reset*

Pre-personalization

Another name for → *initialization*.

Prepaid

Refers to money flow for cards used in the telecommunications sector in which the real money of the cardholder flows before the service (usually a telephone call or data transmission) is received. With regard to their payment function, postpaid cards are comparable to electronic purses. In payment systems, this form of payment is called → *pay before*.

Prepaid SIM

A prepaid and usually reloadable SIM. All billing and reloading functions are usually provided by the background system, so they have no effect on data objects or functions in the SIM. The opposite of this is a → *postpaid* SIM.

Proactivity

A transaction mechanism for smart cards that allows a smart card to independently initiate actions in the terminal. This circumvents the rigid master–slave relationship between the terminal and the smart card. Proactivity is realized by cyclic polling of the smart card by the terminal, with the polling interval being configurable in advance by the smart card. Proactivity originated with SIMs, and it is still predominantly used to allow SIMS to effectively assume control of certain functions of the mobile telephone in accordance with GSM 11.14.

Probabilistic

Designates a process or an algorithm that yields varying results from identical input conditions; the opposite of → *deterministic*.

Procedural programming

A programming method based on formulating a program as a series of instructions for a → *microprocessor*. For purposes of simplification, the program flow can be broken down into functions, with the necessary data being held in variables. A key aspect of procedural programming is that it focuses on the processes of the program, rather than on the data to be processed as does object-oriented programming. Some typical programming languages used for procedural programming are Basic and C.

Process model

Another term for → *life-cycle model*.

Processor

→ *microprocessor*

Processor card

A short form of → *microprocessor card*.

Program code

Designation for a program that can be directly executed by an → *interpreter* or → *microprocessor.* (→ *native code*)

Proprietary

An adjective used in the smart card world, often in a deprecatory sense, to refer to a company-specific solution whose specifications are not fully public or belong to a single company. The opposite of a proprietary solution is an open solution, which can also be used by third parties. However, these terms are used in a far from unambiguous manner. Seen objectively, many 'open' smart card operating systems are rather proprietary and dependent on a single company. An example of a proprietary smart card → *application* would be an electronic purse system for use in a specific area that does not comply with relevant specifications and that has been developed by a particular company as a special solution.

Protection profile (PP)

In the context of an → *evaluation*, an implementation-independent set of security requirements (→ *security target*) adapted to particular application areas for specific → *targets of evaluation*.

Proton [Proton]

Brand name of an internationally used electronic purse system with approximately 50 million issued cards (as of spring 2002). The specifications for Proton also define a multiapplication → *smart card operating system*.

Prototype

A (software) prototype is an executable model of the ultimate product with restricted functionality. It is used to experimentally investigate specific properties of the ultimate product. A horizontal prototype implements only one or more specific layers of the software, while a vertical prototype implements a specific portion of the software across all of the layers.

Pseudonymization

The process of modifying person-specific data using an assignment rule such that it is afterwards not possible to associate the data with the original persons without knowing the assignment rule. The term is based on the fact that in the simplest case, the original name of each person is replaced by a unique pseudonym. A separate assignment table (the assignment rule) can be used to restore the links between the pseudonyms and the original names. (→ *anonymization*)

PSTN (public switched telephone network)

Designates the regular public wire-bound telephone network.

Public-key algorithm

→ *cryptographic algorithm*

PUK (personal unblocking key)

A special → *PIN* for resetting a PIN error counter that has reached its maximum value. A PUK is usually longer than a PIN (e.g., 8 digits), since users do not need the PUK unless they have forgotten the PIN, at which time they can search for it in their documents. If the PUK is successfully used, a new PIN is established at the same time, since the old PIN is evidently no longer known to the user.

Pull technology

Information transfer resulting from fetching information from a higher-level system (such as a server) by a lower-level system (such as a mobile telephone). The opposite to pull technology is → *push technology*.

Purse holder

A person possessing a → *smart card* containing an electronic purse.

Purse provider

The entity responsible for the overall functionality and security of an electronic purse system. This is usually the issuer of the electronic money for the cards. The purse provider normally also guarantees the redemption of the electronic money.

Purse-to-purse transaction

Transfer of electronic monetary units from one electronic purse directly to another, without intervention by a third, higher-level system. Normally, such capability requires the purse system to operate anonymously and the electronic purses to use a single common key for this function.

Push technology

Information transfer resulting from sending information from a higher-level system (such as a server) to a lower-level system (such as a mobile telephone). The opposite to push technology is → *pull technology*.

Quick

Brand name of an electronic purse system introduced throughout Austria in 1995. The essential components of the Quick system are based on EN 1546, which is the European standard for interoperable electronic purse systems.[39]

Radicchio [Radicchio]

An international initiative of companies and organizations for developing mobile → *e-commerce* solutions using the → *PKI*.

[39] See also Section 12.3.1, 'The CEN EN 1546 standard'

RAM (random-access memory)

A type of volatile memory, which is used in smart cards as working memory. RAM loses its content in the absence of power. SRAM and DRAM are types of RAM with special technical properties.[40]

Record

A record (data set) is a specific quantity of data, similar to a string.

Redlist

→*hotlist*

Registration authority (RA)

An entity in the → *PKI* that receives requests for certification from requesting parties and forwards them to the → *certification authority* after verifying the authenticity of the requesting parties. A registration authority is thus the entity that generates a unique assignment of certificates to persons.

Remote applet management

The management (creation, deletion etc.) of → *applets* in a smart card by a background system. For example, in various → *GSM* systems applets can be loaded into a SIM or deleted from a SIM via the air interface.

Remote file management

The management (creation, deletion, writing, reading, modification of access conditions etc.) of files in a smart card by a background system. For example, various → *GSM* systems allow new files to be created in a SIM and data to be written to these files, all via the air interface.

Reset

Restoring a computer (in this case, a smart card) to a clearly defined initial state. A cold reset, or power-on reset, is initiated by switching the supply voltage off and then on again. A warm reset is initiated by a signal on the reset lead of the smart card without altering the supply voltage.

[40] See also Section 3.4.2, 'Memory types'

Response

→*command*

Response APDU

A reply sent by a → *smart card* in response to a → *command APDU* received from a terminal (→ *command*). It consists of optional response data and a mandatory 1-byte portion containing status words SW1 and SW2 (→ *APDU*).[41]

Reticule

→*ROM mask*

Retry counter

→*error counter*

Roaming

Accessibility of a mobile telephone in a network other than its → *home net*.

Roll back

Functionality of an operating system for maintaining data consistency in the event of an error or abnormal termination. With roll-back functionality, the data used in an improperly executed or aborted operation are replaced by the original data. This process can be initiated automatically or on demand, and in smart card operating systems it is often implemented using → *atomic operations*. Another strategy for maintaining data consistency is → *roll-forward* functionality.

Roll forward

Functionality of an operating system for maintaining data consistency in the event of an error or abnormal termination. With roll-forward functionality, in the event of an improperly executed or aborted operation the data that are available but inconsistent are fed back into the operation in such a way that on completion, they are again consistent. This process can be initiated automatically or on demand, but it is rarely implemented in smart card operating systems due to their high security requirements. Another strategy for maintaining data consistency is → *roll-back* functionality.

[41] See Section 6.5, 'Message Structure: APDUs'

ROM (read-only memory)

A type of non-volatile memory, which is used in smart cards. It is mainly used to store programs and static data, since the content of a ROM cannot be altered.[42]

ROM mask

In ordinary language, this term is used in a highly context-specific manner. The original meaning of 'ROM mask' is an exposure mask used in semiconductor fabrication to produce the ROM. However, the term 'mask' is only used when the mask is not reduced in scale when exposing the → *wafer*. If the structures are reduced in scale for imaging onto the wafer, the mask is referred to as a 'reticule'.

The expression 'mask' is also used in connection with → *smart card microcontrollers* to refer to the data content of the ROM, and in some cases it is even synonymous with the entire → *smart card operating system* (→ *soft mask*,→ *hard mask*).

ROMed application

A smart card application that is not located in the EEPROM, but instead is permanently located in the mask-programmed ROM of the → *smart card microcontroller*.

Round-trip engineering

A software development method in which the design and implementation activities are performed concurrently so that they influence each other. The software architecture and program code are automatically kept mutually consistent using software. This process is based on a formal modeling language (such as UML), from which at least the basic body of the program is generated using automatic program code generation. The insights and improvements obtained by refining and testing this program code flow back into the modeling of the program via a reverse engineering process. It is possible to produce software based on 'practical experience' in a relatively short time, with source code that is consistent with the software architecture, by cycling through this code generation / reverse engineering loop several times. Round-trip engineering is almost exclusively used with object-oriented languages (e.g. C++ and Java) in combination with UML.

RSA (Rivest, Shamir, Adleman)

The best known and most widely used asymmetric cryptographic algorithm. It was published by Ronald L. Rivest, Adi Shamir and Leonard Adleman in 1978, and its name comes from the initial letters of the last names of its authors. Its very simple operating principle is based on the arithmetic of large integers.[43]

[42] See also Section 3.4.2, 'Memory types'
[43] See also Section 4.7.2, 'Asymmetric cryptographic algorithms'

R-UIM (removable user identity module)

The usual designation for a CDMA-specific smart card. It is an optional security module that can be present in removable form in a mobile telephone of the CDMA 2000 mobile telecommunication system. The functionality of the R-UIM is similar to that of the → *SIM*, although the CAVE (cellular authentication, voice privacy and encryption) cryptographic algorithm is used for a large number of cryptographically secured functions in the R-UIM. A UIM Application Toolkit (UATK) based on the SIM Application Toolkit is also specified for the R-UIM.

Rule-based programming

A programming method based on formulating general rules to be applied to the problems to be solved. A computer can then independently solve these problems by using the rules. A key aspect of rule-based programming is that it does not focus on processes, as does → *procedural programming*, or the data to be processed, as does → *object-oriented programming*, but only on general rules. Some typical rule-based programming languages are Lisp and Prolog.

Salt

A random sequence used to extend a password in order to hinder dictionary attacks on stored passwords.

SAM (secure application module)

→*security module*

Sandbox

→*virtual machine*

SCOPE (Smart Card Open Platform Environment)

Specification for a type of → *HAL* (hardware abstraction layer) for → *Global Platform*.

Scrambling

A jumbled arrangement of the address, data and control busses on a microcontroller chip, such that it is not possible to recognize the functions of individual bus lines without inside information. With static scrambling, the busses of a given series of chips are all scrambled in the same way, while with dynamic scrambling, the busses are scrambled differently for each individual chip or even each individual session.[44]

[44] See also Section 8.2.4.1, 'Attacks at the physical level'

Scratch card

A card in the usual → *ID-1* format but thinner than usual, with a secret number printed underneath an opaque cover layer that can be scratched off. This cover layer acts as a seal that allows the integrity of the card to be visually checked before it is used. The functional purpose of a scratch card is similar to that of a PIN letter. Scratch cards are often used as vouchers for distributing one-time passwords for reloading → *prepaid SIMs*.

Script

Any interpreted program that is primarily used to implement a simple, short application or automate a frequently repeated procedure.

SDMA (space division multiple access)

A multiple-access method for concurrently transferring data from several transmitters to a receiver using a single frequency. For this purpose, the transmitters use directionally selective aerials aimed at the receiver in question. Due to the high cost of this method, in mobile telecommunication systems it can only be used with base stations, for instance using array aerials (adaptive aerials).[45]

SECCOS (Security Card Operating System)

A multiapplication → *smart card operating system* used for German Eurocheque cards with chips and → *Geldkarte*.

Secret-key algorithm

→ *cryptographic algorithm*

Secure messaging

All methods, protocols and cryptographic algorithms used to protect → *smart card* data transmissions against manipulation and tapping.[46]

Security environment (SE)

In a smart card, a designation for a logical container holding a set of fully defined security measures used by → *commands* related to security or used for → *secure messaging*. Security environments are very suitable for items such as technical measures used to ensure the security of

[45] See also Section 13.1.1, ‘Multiple-access methods’
[46] See also Section 6.6, ‘Securing Data Transmission’

the various stages of the → *life cycle* of a smart card. In the simplest case different security environments would be defined for the personalization and subsequent use of the card, so that different file → *access conditions* would be specified for the different stages of the smart card life cycle. Write access would be allowed to all files for personalization, but for normal use the access conditions would be specified according to the actual → *application*.

Security module

A component that is secured both mechanically and computationally and is used to store secret data and execute cryptographic algorithms. It is also known as a secure application module (SAM), hardware security module (HSM) or host security module (HSM).

Security target

In the context of an → *evaluation*, security targets describe the mechanisms to be tested for the → *target of evaluation*. They thus represent a sort of requirements catalog for the evaluation. The security targets for specific types of targets of evaluation and specific application areas for targets of evaluation can be described using → *protection profiles*.

Seed number (seed)

A random number used as the initial value for a pseudorandom number generator.

Sequence control

A method for specifying a compulsory sequence of activities. For example, the correct sequence of → *commands* for mutual authentication of a → *smart card* and a background system can be enforced using sequence control in the smart card. This is done by specifying the states and state transitions of a state machine in the → *smart card operating system* that defines the command sequence that must be followed.[47]

Serial data transmission

A type of data transmission in which individual data bits are sent sequentially along a data line. (→ *parallel data transmission*)

Service provider

In a smart card system, an entity offering services that are used and paid for by a user. In the case of an electronic purse system, a service provider is an entity that receives money from the electronic purse of a purse holder in exchange for goods or services.

[47] See also Section 5.8, 'Sequence Control'

Session

The time between the activation and deactivation sequences of a smart card, during which both the complete data exchange and the necessary computational mechanisms occur.

SET (Secure Electronic Transaction Standard)

A financial transaction protocol for secure payment via the Internet using credit cards, published by Visa and MasterCard in 1966. SET doe not compel the payer to have a smart card, since it can be implemented fully in software on a PC. An extension of SET called Chip-SET (C-SET) is presently only used inside France and not yet internationally standardized.

Shall, should & may

These auxiliary verbs are often found in international standards. Their meanings in this context are precisely defined and differ in part from their lax usage in common speech. ‘Shall’ means that the item in question must be implemented in accordance with the description. ‘Should’, although it may appear to be a recommendation, actually means that the described item is to be provided or complied with if at all possible. Only ‘may’ provides a true opportunity for choice with regard to implementation.

Shared secrets

A principle according to which no single person knows everything about a particular system. The intentional distribution of knowledge avoids making individual persons subject to attack, as well as preventing individual persons from acquiring excessive power over a system due to their knowledge. Distributing knowledge over several persons is a commonly used technique in the development of security components.

Short FID (SFI)

A 5-bit identifier for an EF that can have a value of 1 through 31. It is used with a write or read command (such as READ BINARY) to implicitly select an EF in a smart card.[48]

Shrink

Refers to reducing the surface area of a semiconductor chip by using a semiconductor technology with a smaller structure width. A smaller chip surface area allows a larger number of chips to be placed on an individual wafer. This in turn reduces the cost of the individual semiconductor chips, since the chip price is approximately proportional to the amount of space occupied by the chip on the wafer.

[48] See also Section 5.6.2, ‘File names’

Shutter

A mechanical device in a terminal that severs any wires leading out of the terminal from the card. This is intended to prevent manipulation of communications. If the wires cannot be cut, the inserted smart card will not be electrically activated.

Signal burst

A cohesive data packet transmitted between a base station and a mobile station via the air interface. Frequently simply referred to as a burst.

Signature Act

In general, a legislative act that governs the use of → *digital signatures*. In Germany, this is understood to mean the *Signaturgesetz (SigG)*, or in full the *Gesetz über Rahmenbedingungen für elektronische Signaturen* of 22 May 2001. This Act prescribes the general conditions for the use of digital signatures in Germany,[49] which are given more concrete form in the → *Signaturverordnung*.

Signature card

A →*smart card* with a →*microcontroller* whose principal function is to secure the storage and use of secret keys for →*digital signatures*.

Signaturgesetz (SigG)

The legislative act that governs the use of digital signatures in Germany (→*Signature Act*).

Signaturverordnung (*SigV*)

The German *Signaturverordnung* (Digital Signature Ordinance) of 1997,8 October, translates the general conditions prescribed by the → *Signaturgesetz* into concrete terms to the extent necessary to allow lists of specific measures to be generated as recommendations for the practical use of digital signatures. For example, the *Signaturverordnung* describes the necessary procedure for generating signature keys and identification data, as well as the necessary security concepts and the necessary testing stages for the signature components according to the ITSEC.[50]

[49] See also Section 14.4, 'Digital Signatures'
[50] See also Section 14.4, 'Digital Signatures'

SIM (subscriber identity module)

The usual designation for a GSM-specific smart card.[51] It is a mandatory security module that is present in mobile telephones in an exchangeable form. It may be the same size as a standard credit card (ID-1 format), or it may be a small plug-in card in the ID-000 format. The SIM bears the identity of the subscriber, and its primarily function is to secure the authenticity of the mobile station with respect to the network. Additional functions include executing programs with protection against manipulation (authentication), user identification (using a PIN) and storing data, such as telephone numbers. The equivalent of the SIM in the UMTS is the → *USIM*.[52]

SIM Alliance [SIM Alliance]

A consortium founded in 1999 by Gemplus, G + D, ORGA and Schlumberger in order to allow services developed for WAP to also be used with non-WAP-capable mobile telephones. For this purpose, the SIM must have a SIM-Alliance-capable browser and the mobile telephone must support GSM Phase 2+. This allows the → *SIM* to control the mobile telephone via the → *SIM Application Toolkit* to the extent that the majority of WAP contents and their functionality can be reproduced on the mobile telephone.[53]

SIM Application Toolkit (SAT; also STK (uncommon and outdated))

An extension to the GSM 11.11 specification, resembling a construction set and standardized in GSM 11.14, that allows the SIM to assume an active role in controlling the mobile telephone. For example, with the SIM Application Toolkit a SIM can output items to be shown on the display, request keypad entries and send and receive messages via the air interface. The SIM Application Toolkit forms the basis for most supplementary applications in mobile telephones. The equivalent of the SIM Application Toolkit for the UMTS is the → *USIM Application Toolkit* (USAT),[54] and for the → *R-UIM* the equivalent is the UIM Application Toolkit (UATK). The future generic foundation for all application toolkits for smart cards used in mobile telecommunications will be the Card Application Toolkit (CAT) defined by the → *EP SCP* expert group.

SIM Lock

A technique that firmly links a particular → *smart card* (a → *SIM*) to a particular mobile telephone. It involves either having the mobile telephone read certain data from the SIM and compare them with data stored in the mobile telephone, or having the SIM read unique data from the mobile telephone and compare them with stored data. If the data match, the mobile telephone can be used. It is generally possible to disable the SIM Lock function via the air

[51] See also Section 13.2, 'The GSM System'
[52] See also Section 13.3, 'The UMTS System'
[53] See also Section 13.5, 'The WIM'
[54] See also Section 13.3, 'The UMTS System'

interface or by entering a secret key using the keypad of the mobile telephone, in order to allow other smart cards to subsequently be used. The SIM Lock function is used to bind a mobile telephone subsidized by a network operator to a particular smart card and its payment mode (→ *prepaid*) for a certain length of time.[55]

SIM toolkit

Short for → *SIM Application Toolkit.*

SIMEG (Subscriber Identification Module Expert Group)

SIMEG was an expert group founded in 1988 that developed the specification for the interface between the smart card and the mobile telephone (GSM 11.11) under the authority of the → *ETSI*. In 1994, the name of the group was changed to → *SMG9*.

Simulator

Software that imitates the operation of a device (a target system). By contrast, an imitation using hardware is called an → *emulator*. Simulators are frequently used in developing software for target systems that do not yet exist. For instance, a smart card simulator consists of software that fully imitates a real smart card on the logical level. Simulators are generally slower than emulators, which means that they often cannot simulate the target system in real time.

Single sign-on (SSO)

A technique in which several different user identities for various applications are replaced by a single central user identity. This is realized using software that sends the corresponding identification names (→ *identification*) and passwords to the associated identification authorities on successful completion of central user identification. This avoids the need for the user to remember many different passwords.

Skimming

A typical type of attack on magnetic-stripe cards. It involves illicitly reading the magnetic-stripe data from a card not belonging to the attacker and copying it to the magnetic stripe of a blank card, which can then be used in the same way as the original card with respect to its magnetic stripe.

[55] See also Section 13.2, 'The GSM System'

Smart card

Strictly speaking, the term ‘smart card’ is an alternate name for a microprocessor card, in that it refers to a chip card that is ‘smart’. Memory cards thus do not properly fall into the category of smart cards. However, the expression ‘smart card’ is generally used in English-speaking countries to refer to all types of cards containing chips.

Smart card application

→ *application*

Smart card microcontroller

A → *microcontroller* specifically optimized for the needs of smart cards. These optimizations primarily relate to chip security aspects (e.g., protective layers and detectors), chip size and special functional units for requirements specific to smart cards (such as a UART for communications).

Smart card operating system

Often also referred to as ‘card operating system’ (COS). A specialized form of → *operating system* tailored to the needs of smart cards, encompassing all programs in a → *smart card microcontroller* that allow smart card → *applications* to be used and managed. For this purpose, the data, files, → *commands*, processes, states, mechanisms, algorithms and programs needed by one or more programs must be supported in a suitable manner. If a smart card operating system allows several applications to be run concurrently, it is called ‘multiapplication capable’. The trend in the development of smart card operating systems is toward → *open smart card operating systems*. Some typical examples of smart card operating systems are → *Multos*, → *Java Card*, → *Windows for Smart Cards* and → *STARCOS*.

Smart label

A data storage medium with a thin construction using contactless data transmission for communications. With the simplest versions, many of which do not contain chips, it is only possible to read data from the smart label. More sophisticated types of smart labels also allow data to be written to the label and/or processed in the label, similar to the functionality of a → *smart card.*

Smart object

A → *smart card microcontroller* packaged in a form other than the usual card. Some examples of smart objects are USB plugs and rings fitted with smart card microcontrollers.

Smartcard [Groupmark]

The term ‘Smartcard’ is a registered trademark of the Canadian company Groupmark.

SMG9 (Special Mobile Group 9)

SMG9 was an expert group operating under the authority of the → *ETSI* that developed specifications for the interface between the smart card and the mobile telephone (e.g. GSM 11.11, GSM 11.14 and so on). It was composed of representatives of card manufacturers, mobile telephone manufacturers and network operators. It was previously called → *SIMEG*. In 2000, SMG9 was dissolved and its tasks were divided between two new expert groups. The 3GPP T3 expert group is responsible for the application-specific interface between the mobile telephone and the → *SIM* or → *USIM*, while the ETSI Smart Card Platform (EP SCP) expert group deals with all generic topics in the area of smart cards used in the telecommunications sector.

SMS (short message service)

A GMS service for sending short messages having a maximum length of 160 alphanumeric characters. SMS messages are sent via the signaling channel instead of the data channel, which means that they can also be sent and received during an active telephone conversation. SMS is used not only for conveying short messages for subscribers, but also as a → *bearer* service for transmitting data to the mobile telephone (e.g., → *WAP*) or the SIM (→ *OTA*).

Soft mask

The term ‘soft mask’ means that part of the program code of the → *smart card operating system* is located in EEPROM and built on top of the code stored in ROM. Routines stored in EEPROM can be easily modified by overwriting, which means that they are ‘soft’. The term ‘mask’ in this case is actually not correct, since it is not necessary to produce a semiconductor exposure mask for program code stored in EEPROM. Soft masks are typically not used for large quantities of cards, rapid → *prototyping* or extensions, but instead for applications such as field trials. The opposite of a soft mask is a → *hard mask*, which means that all of the essential functions are stored in the ROM.

Software specification

An unambiguous, complete and non-redundant description of a software item. Its content must not include anything subject to interpretation, and it must be comprehensible to all reader groups having various functions (developers, testers, QA officers etc.) within an acceptable length of time.

SPA (simple power analysis)

A method for attacking smart cards that involves measuring the current consumption of a microcontroller with high time resolution. Conclusions about the internal processes and the data processed within the microcontroller can be drawn from its current consumption. SPA was made known in June 1998 in a publication by Paul Kocher, Joshua Jaffe und Benjamin Jun [Kocher 98] (→ *DPA*).[56]

SPA/DPA-resistant

Property of a cryptographic algorithm that does not allow the secret key being used to be determined using → *SPA* or → *DPA*.

Specification

In this book, and generally speaking, the term *specification* is used to refer to any document that resembles a standard but is generated or issued by (for example) a company or an industrial group, rather than by a national or international standards authority. (→ *standard*)

SRAM (static random-access memory)

A static RAM needs a constant supply of power to retain its contents, but it does not have to be periodically refreshed like DRAM. The access time of SRAM is less than that of DRAM, but SRAM occupies more space on the chip and is thus more expensive.[57]

Stack

A data structure in which the most recently entered data object is the first to be retrieved (last in, first out – LIFO). Probably the best-known stack is the program stack, which is used to hold return addresses when subroutines are called.

Standard

A document containing technical descriptions and/or precise criteria used as rules and/or definitions of characteristics and features in order to ensure that materials, products, processes and services can be used for their intended purposes. In this book, the term 'standard' is consistently used in connection with a national or international standards authority (such as ISO, CEN, ANSI or ETSI).

[56] See also Section 8.2.4.1, 'Attacks at the physical level'
[57] See also Section 3.4.2, 'Memory types'

STARCOS

Brand name of a multiapplication → *smart card operating system* from Gieseke & Devrient [GD], available in various versions since 1991.

State machine

A part of a program that specifies a sequence of events by means of a predefined state diagram, which consists of specific states and state transitions.

Steganography

The objective of steganography is to conceal messages within other messages such that they cannot be perceived by a naïve observer (man or machine). For example, a text could be encoded and hidden in an image file in such a way that it only marginally modifies the image, so the changes to the image are practically invisible (→ *digital watermark*). With a suitable analysis program, the text message (such as a copyright text) hidden in the image file can subsequently be reconstructed and again made visible.

Super smart card

A smart card with integrated complex card elements, such as a display or keypad.

Symmetric cryptographic algorithm

→ *cryptographic algorithm*

Synchronous data transmission

A form of data transmission in which data transmission depends on a predefined timing reference. This timing reference may for example be derived from the clock signal applied to the chip. (→ *asynchronous data transmission*)

System on card

In the smart card realm, a designation for a smart card containing supplementary card components in addition to the chip module. Commonly used supplementary components include displays, power sources (batteries and solar cells), keypads, aerials, sensors for biometric user identification (e.g., fingerprint readers) and loudspeakers. These card components can be driven from within the chip in the module, but this is not mandatory. Another designation for such cards is 'super smart card', although this is used less often.

T = 0

A transmission protocol governing data transmission between a terminal and smart card. The T = 0 protocol was the first internationally standardized transmission protocol for smart cards. It is a byte-oriented half-duplex protocol that operates asynchronously and is designed for minimum memory usage and maximum simplicity. It is used internationally for GSM cards, and is thus the most widely used of all smart card protocols. The T = 0 protocol is specified in the ISO/IEC 7816-3 standard. Compatible specifications are present in GSM 11.11, TS 102.221 and the EMV specification documents.

T = 1

A transmission protocol governing data transmission between a terminal and smart card. It is a block-oriented half-duplex protocol that operates asynchronously and provides separation between the data transport and application layers. The T = 1 protocol is specified in the ISO/IEC 7816-3 standard. Compatible specifications are present in TS 102.221 and the EMV specification documents.

T3

→*SMG9*

Tag

Identifier for a data object, primarily used in ASN.1 coding.[58]

Tape out

The time at which the chip design is completed and the resulting design data are passed to mask generation (→ *ROM mask*). This is an important milestone in chip production. The term comes from the fact that the mask data were formerly output to magnetic tape.

Target of evaluation (TOE)

The IT system to be evaluated (→ *evaluation*), or in other words, the object under test. For example, a TOE could be a microcontroller smart card (→ *smart card microcontroller*) with integrated software that must meet certain → *security targets*.

TCSEC (Trusted Computer System Evaluation Criteria)

A catalog of criteria for the development and → *evaluation* of the security of information technology systems in the US, published in 1983 by the National Computer Security Center

[58] See also Section 4.1, 'Structuring Data'

(→ *NCSC*). The successor to the national TCSEC is the internationally applicable → *Common Criteria*.

TD/CDMA (time division / code division multiple access)

→*CDMA*

TDES

→*triple DES*

TDMA (time division multiple access)

A multiple-access method for the quasi-concurrent transfer of data from multiple transmitters to a receiver using a single frequency. For this purpose, each transmitter is allocated a particular time slot for its exclusive use, which requires very precise synchronization. TDMA is used together with FDMA in GSM systems for the air interface between mobile telephones and base stations.[59]

Terminal

The counterpart to a smart card. A device, possibly having a keypad and display, that provides electrical power to the smart card and enables it to exchange data. The official ISO designation for a smart card terminal is 'interface device' (IFD), while in the financial transaction realm the usual designation for a terminal is 'card accepting device' (CAD).

Test

Development stage in which an already debugged program is methodically tested for proper functionality and compliance with the requirements established in the → *analysis* stage. The primary objective is not searching for errors in the program, but instead verifying the required functions. Testing is thus not the same as → *debugging*.

TETRA (Terrestrial Trunked Radio; previously Trans-European Trunked Radio)

Specification for a digital trunked radio system operating in the 380–420-MHz band using → *TDMA*, published by ETSI. Like → *GMS*, TETRA envisages the use of a → *SIM*, usually called a TETRA SIM, for subscriber identification. However, the TETRA SIM is optional and thus can be implemented in the form of software in the mobile station.

[59] See also Section 13.1.1, 'Multiple-access methods'

TETRA SIM

→*TETRA*

Thread

→*multithreading*

Time stamp

An attestation generated by an agency and bearing the digital signature of that agency, to the effect that certain digital data were present at this agency at a particular time.

TLV format

Commonly used expression for → *BER*-coded data objects conforming to → *ASN.1*, in which a prefixed label (tag) and length parameter (length) are used to uniquely describe a data item (value).

TPDU

→*APDU*

Transaction

A set of related → *commands* sent sequentially to a smart card in order to perform a specific task. A typical example of a transaction is the sequence of commands used to load an electronic purse.

Transaction number (TAN)

In contrast to a PIN, a TAN is valid for only one transaction, which means it can be used only once. The user typically receives several TANs printed on a slip of paper (as a list of five-digit numbers, for example), and these numbers must be used exactly in the prescribed order for individual transactions or sessions.

Transfer card

A → *smart card* used as a transport medium to exchange data between two entities. It has a large data memory and usually contains authentication keys for verifying whether the data to be transferred are allowed to be read or written by the entity in question.

Transient

Property of an object that exists only during the run time of a process; the opposite of → *persistent*.

Transmission protocol

In the smart card world, 'transmission protocol' refers to the mechanisms used for transmitting and receiving data between a terminal and a smart card. A transmission protocol describes in detail the OSI protocol layers used, data exchange in the good case, error detection mechanisms and response mechanisms in the event of errors.[60]

Transport protocol

An alternate and less commonly used name for → *transmission protocol.*

Trap door

A mechanism or algorithm intentionally included in software that can be used to bypass security functions or protective mechanisms.

Triple-band mobile telephone

A mobile telephone that can work in three frequency bands (e.g., GSM 900, GSM 1800 and GSM 1900).

Triple DES

Also known as TDES and 3 DES; a modified form of DES encryption consisting of invoking the DES algorithm three times in succession with alternating encryption and decryption. If the same key is used for all three DES invocations, triple-DES encryption corresponds to normal DES encryption. However, if two or three different keys are used, triple DES encryption is significantly stronger than single DES encryption.[61]

Trivial PIN

A → *PIN* that is easily guessed, such as "1234" (→ *0-PIN*).

[60] See also Chapter 6, 'Smart Card Data Transmission'
[61] See also Section 4.7.1, 'Symmetric cryptographic algorithms'

Trojan horse

Historically, the wooden horse that allowed Odysseus to gain entry to the strongly fortified city of Troy. In modern usage, a program that performs a specific ‘foreground’ task, but which can also perform other functions unknown to the user. A Trojan horse is introduced purposely into a computer system or host program. In contrast to a virus, it cannot reproduce itself.

Trust center (TC)

Besides the → *signature card*, the essential component of a → *PKI*. A trust center is the entity that generates, distributes and administers certificates. Depending on its constitution, it thus provides the functions of a → *certification authority*, a → *registration authority*, a → *verification service* and/or a → *time stamp* service.

Trusted third party (TTP)

A party recognized by two or more other parties as trustworthy, which may for example issue → *certificates*.

Tunneling

A technique for establishing a cryptographically secured end-to-end link between two parties using the communications paths of one or more other parties that do not modify the information content of the actual data exchange.

UART (universal asynchronous receiver/transmitter)

A general-purpose component operating independently of a → *microprocessor* to asynchronously transmit and receive data. When a UART is used, it is not necessary for the microprocessor to handle communications at the bit and byte level. This leads to a simplification of the communications protocol, and it can also result in higher data transmission rates than what can be realized by the microprocessor using a pure software solution.[62]

UCS (Universal Character Set)

An extension of the ASCII and Unicode encodings of character sets, specified in the ISO/IEC 10 646 international standard. UCS uses 32 bits for character encoding, although only half of the available address space is used ($2^{32}/2 = 2{,}147{,}483{,}648$). This address space is sufficient to represent all characters of all of the languages in the world. UCS is defined such that → *Unicode* forms a subset of UCS, and the encoding of the first 128 characters corresponds to the ASCII encoding.[63]

[62] See also Section 3.4.3, ‘Supplementary hardware’
[63] See also Chapter 4.2, ‘Coding Alphanumeric Data’

UICC (universal integrated chip card)

A smart card having a → *smart card operating system* in accordance with ISO/IEC 7816 that is optimized for telecommunications applications. The UICC is based on the TS 102.221 standard, which is published by → *ETSI*. The UICC forms the basis for the → *USIM*. It may have the usual credit-card dimensions or be made as a small plug-in card in ID000 format.

UIM (user identity module)

An outdated term for → *USIM*.

UML (Unified Modeling Language) [OMG]

A graphically oriented, method-independent modeling language for abstractly describing the static and dynamic aspects of object-oriented programs. The current version of UML is 1.3. The foundations of the semantics and notation of UML were created in the 1990s by Grady Booch, James Rumbaugh und Ivar Jacobson. UML is independent of any particular → *life-cycle model* for software development.[64] The Object Management Group (OMG) is responsible for the ongoing development of UML.

UMTS (Universal Mobile Telecommunication System)

The European successor to GSM and a member of the → *ITM-2000* family. UMTS is a third-generation (→ 3*G*) digital, cellular, interoperable, transnational land-based mobile telecommunication system. The frequency band allocated to this mobile telecommunication system lies at 2000 MHz. The UMTS system is defined by a number of specifications generated under the auspices of the → *3GPP* and published by → *ETSI*. UMTS represents the next major evolutionary step for → *GSM*. The essential changes with respect to GSM are a new air interface using → *CDMA* technology and a significantly higher data transmission rate of up to 2 Mbit/s.[65]

Unicode [Unicode]

An extension of the well-known ASCII character code. In contrast to the 7-bit ASCII code, Unicode employs 16 bits for coding. This allows the characters of the most widely used languages of the world to be supported. The first 256 Unicode characters are identical to the ISO 8859-1 ASCII characters.[66]

[64] See also Section 15.7, 'Life-Cycle Models'
[65] See also Section 13.3, 'The UMTS System'
[66] See also Section 4.2, 'Coding Alphanumeric Data'

Uplink

A connection from a lower-level system (such as a mobile telephone) to a higher-level system (such as a base station); the opposite of → *downlink*.

Upload

Transferring data from a lower-level system (such as a terminal) to a higher-level system (such as a background system or host system); the opposite of → *download*.

URL (uniform resource locator)

A unique alphanumeric address in the → *WWW*.

User

A person who uses a → *smart card*; not necessarily the same as the → *cardholder*.

User data

All data directly needed by an → *application*.

USIM (universal subscriber identity module)

The common name of the smart card → *application* for UMTS,[67] which resides in a → *UICC*. However, in practice the term 'USIM' is also used to refer to the UMTS smart card as well as the application, although this is not entirely correct. The USIM bears the identity of the subscriber, and its primary function is to secure the authenticity of the mobile station with respect to the network and vice versa. Additional functions include executing programs with protection against manipulation (authentication), user identification (using a PIN) and storing data, such as the telephone numbers. The USIM is based on the TS 31.102 standard published by → *ETSI*. The equivalent of the USIM in the → *GSM* system is the → *SIM*.[68]

USIM Application Toolkit (USAT)

A collection of functions standardized by TS 31.111 that allow a USIM card to assume an active role in controlling a mobile telephone. For example, a USIM can use the USIM Application Toolkit to output items to be shown on the display, request inputs from the keypad and transmit or receive messages via the air interface. The USIM Application Toolkit forms the basis for most

[67] See also Section 13.3, 'The UMTS System'
[68] See also Section 13.2.4, 'The SIM'

supplementary applications in mobile telephones. The equivalent of the USIM Application Toolkit in GSM is the → *SIM Application Toolkit* (SAT).[69]

Value-added service (VAS)

A supplementary smart card → *application* present in a smart card in addition to the main application. Such services usually presuppose a → *multiapplication smart card*.

Vertical prototype

→ *prototype*

Virgin card

A card that has not yet been implanted with a chip or visually or electronically personalized. A virgin card is essentially a printed, non-specific → *card body*, as used in the mass production of cards.

Virtual machine (VM)

A software simulation of a → *microprocessor*, usually having its own opcodes for machine instructions as well as a simulated address space. It allows software to be generated that is independent of the features of specific hardware. For instance, the virtual address space of a VM can be many times larger than the address space provided by the hardware. In the Java environment, the closed environment of the VM is often called the sandbox.[70]

Virtual merchant card

→ *virtual smart card*

Virtual smart card

A software simulation of a smart card in a different system, such as in a security module or a mobile telephone. A virtual merchant card, which is the simulation of a smart card in a merchant terminal, is a special case of a virtual smart card.

Visa Cash

Visa brand name for several technically different electronic purse systems using smart cards.

[69] See also Section 13.2.4, 'The SIM'
[70] See also Section 5.14.1, 'Java Card'

Visa Easy Entry (VEE)

A method for easy migration from magnetic-stripe credit cards to credit cards with microcontroller chips. This is accomplished by storing the name of the cardholder and all of the data on the magnetic stripe in an EF under a DF that is reserved for Visa. When a payment is made using the credit card, the terminal reads the data needed for the transaction from the chip instead of from the magnetic stripe. The advantage of this approach is that it is only necessary to upgrade the POS terminals to include smart card contact units, while the entire background system can be used as before without any modifications.

Volatile memory

A type of memory (e.g. RAM) that retains its contents only as long as power is applied.

VOP

→ *OP*

Wafer

A thin disc of silicon on which chips are built using semiconductor fabrication techniques. Wafers typically have a diameter of 150 mm (6 inches), 200 mm (8 inches) or 300 mm (12 inches).

WAP (wireless application protocol)

A term used to refer to a number of specifications for creating a link between a mobile terminal (mobile telephone, PDA etc.) and a server via a wireless network, for the purpose of directly exchanging data. The usual application for WAP is implementing Internet services in mobile telephones in a manner that is largely independent of the mobile telecommunications standard used. Incidentally, the designation ‘wireless application protocol’ refers not only to the technology, but also to the protocol used between the terminal and the background system. The WAP Forum, founded in June 1997 by Phone.com, Ericsson, Motorola and Nokia, is the internationally active standards committee for WAP. It is composed of representatives of more than 350 companies.[71]

WAP Forum

→ *WAP*

[71] See also Section 13.5, ‘The WIM’

Warm reset

→*reset*

WCDMA (wideband code division multiple access)

→*CDMA*

Whitelist

A list in a database indicating all smart cards and devices allowed to be used in a particular → *application.* (→ *blacklist,* → *graylist,* →*hotlist)*

White plastic

Refers to non-personalized blank cards used with fraudulent intent. The term originally comes from the typical blank cards made from white plastic that are used to produce test cards. However, it is now understood to also refer to cards that have been printed and have a wide variety of → *card components*, such as credit cards with magnetic stripes and holograms that have not yet been embossed.

Whitebox test

A test, also often also called a glassbox test, in which it is assumed that the party performing the test has complete knowledge of all of the internal processes and data of the software to be tested.

WIM (WAP identity module)

A security module for a → *WAP* terminal. The specification describes a PCKS #15-compatible smart-card → *application*. The principal functions of a WIM are generating and verifying digital signatures and encrypting data. A WIM may be either a separate, physical smart card or one of several applications in a multiapplication smart card. It is typically an application in a → *SIM* or → *USIM*.

Windows for Smart Cards [Microsoft]

An → *open smart card operating system* from Microsoft, also known as WfSC and WSC, that supports multiple → *applications* (→ *multiapplication smart card*) and downloadable programs. One of the special features of Windows for smart cards is that it uses a → *FAT*-based file system.[72]

[72] See also Section 5.7, 'File Management'

WML (wireless markup language)

A logical markup language based on XML used to generate applications for WAP. WML is very similar to HTML. WML applications stored in a WML site on a WAP server are translated on-the-fly into compact WML bytecode, which is transmitted via the wireless network to a the mobile terminal, where it is interpreted by a microbrowser (→ *browser*).

Work-around

In the context of software development, circumventing a known problem by 'programming around' it. A work-around avoids the negative effects of an error on the rest of the program, but it does not eliminate the actual error. For example, work-arounds in EEPROM are typically used to correct errors in ROM-based → *smart card operating systems* that are found after the chips have been produced, in order to prevent such errors from having negative effects on the operation of the operating system. However, it is entirely possible for the functionality of the operating system to be reduced relative to its original scope as a consequence of using work-arounds.

WWW, W3 (World-Wide Web)

A part of the international Internet, primarily characterized by its ability to link any desired documents using hyperlinks and the integration of multimedia objects into documents.

X.509

The X.509 standard published by the → *ITU* defines the structure and coding of → *certificates*. It is the most widely used standard for certificate structures (→ *PKI*) throughout the world.

XML (extended markup language)

A logical markup language that is both a successor to and an extension of HTML. XML can be used to define new language elements, which means that other markup languages, such as HTML and WML, can be defined using XML. XML is a subset of the powerful 'standard generalized markup language' (SGML), which is specified by an ISO standard.

ZKA (*Zentraler Kreditausschuss*)

The coordinating body for the electronic payment transactions of the German banks. The ZKA is composed of the following banking associations: the *Deutsche Sparkassen- und Giroverband* (DSGV), the *Bundesverband der Deutschen Volks- und Raiffeisenbanken* (BVR), the *Bundesverband deutscher Banken* (BdB) and the *Verbund öffentlicher Banken* (VÖB). The chairmanship of the ZKA is assumed by each of the four member associations in yearly rotation.

16.2 RELATED READING

The *Smart Card Handbook* focuses on smart cards and their applications. However, there are a large number of other disciplines that strongly affect smart cards and their further development, each of which has its own particular areas of interest and specialist literature. The authors of the *Smart Card Handbook* wish to maintain the focus of this book within its own field, rather than providing extensive descriptions of related disciplines, since that would vastly exceed the scope of this book. For readers who wish to increase their knowledge of these related subjects, we have prepared the following short list of related reading.

Subject	**Reference**
Operating systems	[Tanenbaum 02]
Smart card manufacturing	[Haghiri 02]
Java as a programming language	[Arnold 00]
Cryoptography	[Menezes 97], [Schneier 96]
RFID	[Finkenzeller 02]
Security of components and systems	[Anderson 01]
Software development	[Balzert 98]
Software development for Java Card	[Chen 00]

16.3 LITERATURE

The following publications are sorted first by the last name of the author and then in ascending order of publication date. 'Internet' is listed as the source of publications that appeared in newsgroups or discussion forums on the Internet.

[Anderson 01] Ross J. Anderson: *Security Engineering*, Wiley, Chichester 2001

[Anderson 92] Ross J. Anderson: *Automatic Teller Machines*, Internet, December 1992

[Anderson 96a] Ross J. Anderson, Markus G. Kuhn: *Improved Differential Fault Analysis*, Internet, November 1996

[Anderson 96b] Ross J. Anderson, Markus G. Kuhn: *Tamper Resistance – a Cautionary Note*, USENIX Workshop, November 1996

[Arnold 00] Ken Arnold, James Gosling, David Holmes: *The Java Programming Language*, 3rd edn, Addison Wesley, Boston 2000

[Balzert 98] Helmut Balzert: *Lehrbuch der Software-Technik*, Vol.2, 2nd edn, Spektrum Akademischer Verlag, Heidelberg 1998

[Bellare 95a] Mihir Bellare, Juan Garay, Ralf Hause, Amir Herzberg, Hugo Krawczyk, Michael Steiner, Gene Tsudik, Michael Waidner: *iKP – A Family of Secure Electronic Payment Protocols*, Internet, 1995

[Bellare 95b] Mihir Bellare, Philip Rogaway: *Optimal Asymmetric Encryption – How to Encrypt with RSA*, Internet, 1995

[Bellare 96] Mihir Bellare, Philip Rogaway: *The Exact Security of Digital Signatures – How to Sign with RSA and Rabin*, Internet, 1996

[Beutelsbacher 93] Albrecht Beutelsbacher: *Kryptologie*, 3rd edn, Vieweg Verlag, Braunschweig 1993

[Beutelsbacher 96] Albrecht Beutelsbacher, Jörg Schwenk, Klaus-Dieter Wolfenstetter: *Moderne Verfahren der Kryptografie*, Vieweg Verlag, Braunschweig 1996

[Biham 91] Eli Biham, Adi Shamir: *Differential Cryptoanalysis of DES-like Cryptosystems*, Journal of Cryptology, Vol. 4, No. 1, 1991

[Biham 93] Eli Biham, Adi Shamir: *Differential Cryptoanalysis of the Data Encryption Standard*, Springer-Verlag, New York 1993

[Biham 96] Eli Biham, Adi Shamir: *A New Cryptoanalytic Attack on DES*, Internet, 1996

[BIS 96] Bank for International Settlements: *Security of Electronic Money – Report by the Committee on Payment and Settlement Systems and the Group of Computer Experts of the Central Banks of the Group of Ten Countries*, Basel, August 1996

[Blumtritt 97] Oskar Blumtritt: *Nachrichtentechnik*, 2nd edn, Munich, Deutsches Museum, 1997

[Boehm 81] Barry W. Boehm: *Software Engineering Economics*, Prentice Hall, Upper Saddle River, New Jersey 1981

[Boneh 96] Dan Boneh, Richard A. DeMillo, Richard J. Lipton: *On the Importance of Checking Computations*, Math and Cryptography Research Group, Bellcore 1996

[Bronstein 96] I. N. Bronstein, K. A. Semendjajew: *Taschenbuch der Mathematik*, 7th edn, B. G. Teubner Verlagsgesellschaft, Leipzig 1997

[Buchmann 96] Johannes Buchmann: *Faktorisierung großer Zahlen*, Spektrum der Wissenschaft, September 1996

[Chen 00] Zhiqun Chen: *Java Card Technology for Smart Cards*, Addison Wesley, Boston 2000

[CMM 93] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, Charles V. Weber: *Capability Maturity Model for Software, Version 1.1*, Software Engineering Institute, Pittsburg 1993

[Dhem 96] J. F. Dhem, D. Veithen, J.-J. Quisquater: *SCALPS: Smart Card Applied to Limited Payment Systems*, UCL Crypto Group Technical Report Series, Université Catholique de Louvain, 1996

[Dictionary of Computing 91] *Dictionary of Computing*, Oxford University Press, Oxford 1991

[Diffie 76] Whitfield Diffie, Martin E. Hellman: *New Directions in Cryptography*, Internet, 1976

[Dröschel 99] Wolfgang Dröschel, Manuela Wiemers: *Das V- Modell 97*, Oldenbourg Verlag, Munich 1999

[Eberspächer 97] Jörg Eberspächer, Hans-Jörg Vögel: *GSM – Global System for Mobile Communication*, B. G. Teubner Verlag, Stuttgart 1997

[EFF 98] Electronic Frontier Foundation: *Frequently Asked Questions (FAQ) about the Electronic Frontier Foundation's "DES Cracker" Machine*, Internet, 1998

[EC 91] Commission of the European Communities: *Information Technology Security Evaluation Criteria (ITSEC)*, Version 1.2, June 1991

[EC 98] Council of the European Communities: *Council Regulation (EC) No 2135 of 24 September 1998 Amending Regulation (EEC) No 3821/85 on recording equipment in road transport and Directive 88/599/EEC concerning the application of Regulations (EEC) No 3820/85 and (EEC) No 3821/85*, Version 1.2, June 1991

[Fenton 96] Norman E. Fenton, Shari Lawrence Pfleeger: *Software Metrics*, Thomson Computer Press, London 1996

[Finkenzeller 02] Klaus Finkenzeller: *RFID-Handbuch*, 3rd edn, Carl Hanser Verlag, Munich/Vienna 2002

[Franz 98] Michael Franz: *Java – Anmerkungen eines Wirth-Schülers*, Informatik Spektrum, Springer-Verlag, Berlin 1998

[Freeman 97]	Adam Freemann, Darrel Ince: *Active Java – Object Oriented Programming for the World Wide Web*, Addison-Wesley, Reading, MA 1997
[Fumy 94]	Walter Fumy, Hans Peter Ries: *Kryptographie*, 2nd edn, R. Oldenbourg Verlag, Munich/Vienna 1994
[Gentz 97]	Wolfgang Gentz: *Die elektronische Geldbörse in Deutschland*, Diplomarbeit an der Fachhochschule München, Munich 1997
[Glade 95]	Albert Glade, Helmut Reimer, Bruno Struif: *Digitale Signatur*, Vieweg Verlag, Braunschweig 1995
[Gora 98]	Walter Gora: *ASN.1 – Abstract Syntax Notation One*, 3rd edn, Fossil Verlag, Köln 1998
[Gosling 95]	James Gosling, Henry McGilton: *The Java Language Environment – A White Paper*, Sun Microsystems, USA 1995
[Grün 96]	Herbert Grün: *Card Manufacturing Materials and Environmental Responsibility*, Presentation at CardTech/SecurTech, Atlanta, GA, May 1996
[GSM 95]	Proceedings of the Seminar for Latin America Decision Makers by GSM MoU Association and ECTEL: *Personal Communication Services based on the GSM Standard*, Buenos Aires 1995
[Guthery 02]	Scott B. Guthery, Mary J. Cronin: *Mobile Application Development with SMS and the SIM Toolkit*, McGraw-Hill, New York 2002
[Gutmann 96]	Peter Gutmann: *Secure Deletion of Data from Magnetic and Solid-State Memory*, USENIX Conferenz, San Jose, CA 1996
[Gutmann 98a]	Peter Gutmann: *Software Generation of Practically Strong Random Numbers*, Internet, 1998
[Gutmann 98b]	Peter Gutmann: *X.509 Style Guide*, Internet, 1998
[Haghiri 02]	Yahya Haghiri, Thomas Tarantino: *Smart Card Manufacturing: A Practical Guide*, Wiley, Chichester 2002
[Hassler 02]	Vesna Hassler, Martin Manninger, Mikhail Gordeev, Christoph Muller: *Java Card for E-Payment Applications*, Artech House, London 2002
[Hellmann 79]	Martin E. Hellmann: *The Mathematics of Public-Key Cryptography*, Scientific American, August 1979

[Hillebrand 2002]	Friedhelm Hillebrand (editor): *GSM and UMTS*, Wiley, Chichester 2002
[IC Protection 97]	*Common Criteria for IT Security Evaluation Protection Profile – Smartcard Integrated Circuit Protection Profile*, Internet, 1997
[Isselhorst 97]	Hartmut Isselhorst: *Betreiberorientierte Sichrheitsanforderungen für Chipkarten-Anwendungen*, Card-Forum, Lüneburg 1997
[Jones 91].	C. Jones: *Applied Software Measurement*, McGraw-Hill, New York 1991
[Jun 99]	Benjamin Jun, Paul Kocher: *The Intel Random Number Generator*, Internet, 1999
[Kaliski 93]	Burton S. Kaliski Jr.: *A Layman's Guide to a Subset of ASN.1, BER and DER*, RSA Laboratories Technical Note, Internet, 1993
[Kaliski 96]	Burton S. Kaliski Jr.: *Timing Attacks on Cryptosystems*, RSA Laboratories, Redwood City, CA 1996
[Karten 97]	Zeitschrift Karten: *Zur Sicherheit der ec-Karte PIN: Das Urteil des OLG Hamm*, Fritz Knapp Verlag, Frankfurt, August 1997
[Knuth 97]	Donald Ervin Knuth: *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 3rd edn, Addison-Wesley/Longman, Reading, MA 1997
[Kocher 95]	Paul C. Kocher: *Timing Attacks on Implementations of Diffie-Hellmann, RSA, DSS, and Other Systems*, Internet, 1995
[Kocher 98 a]	Paul C. Kocher, Joshua Jaffe, Benjamin Jun: *Introduction to Differential Power Analysis and Related Attacks*, Internet, 1998
[Kocher 98b]	Paul C. Kocher, Joshua Jaffe, Benjamin Jun: *Differential Power Analysis: Leaking Secrets*, Internet, 1998
[Kömmerling 99]	Oliver Kömmerling, Markus G. Kuhn, *Design Principles for Tamper-Resistant Smartcard Processors*, USENIX Workshop on Smartcard Technology, Chicago, USA, 10–11 May 1999
[Kuhn 97]	Markus G. Kuhn: *Probability Theory for Pickpockets – ec-PIN Guessing*, COAST Laboratory, Purdue University, West Lafayette, Indiana 1997

[Kuhn]	Markus G. Kuhn: *Attacks on Pay-TV Access Control Systems*, University of Cambridge, Internet, year unknown
[Lamla 00]	Michael Lamla: *Hardware Attacks on Smart Cards – Overview*, Eurosmart Security Conference, Marseille, 13–15 June 2000
[Leiberich 99]	Otto Leiberich: *Vom diplomatischen Code zur Falltürfunktion*, Spektrum der Wissenschaft, June 1999
[Lender 96]	Friedwart Lender: *Production, Personalisation and Mailing of Smart Cards – A Survey*, Smart Card Technologies and Applications Workshop, Berlin, November 1996
[Levy 99]	Steven Levy: *The Open Secret*, Wired, April 1999
[Lindholm 97]	Tim Lindholm, Frank Yellin: *The Java Virtual Machine Specification*, 2nd edn, Addison-Wesley, Reading, MA 1999
[Massey 88]	James L. Massey: *An Introduction to Contemporary Cryptology*, Proceedings of the IEEE, Vol. 76, No. 5, May 1988, pp 533–549
[Massey 97]	James L. Massey: *Cryptography, Fundamentals and Applications*, 1997
[Meister 95]	Giesela Meister, Eric Johnson: *Schlüsselmanagement und Sicherheitsprotokolle gemäß ISO/SC 27 – Standards in Smart Card-Umgebungen,* in: Albert Glade, Helmut Reimer, Bruno Struif: *Digitale Signatur*, Vieweg Verlag, Braunschweig 1995
[Menezes 93]	Alfred J. Menezes: *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishing, Boston, MA 1993
[Menezes 97]	Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone: *Handbook of Applied Cryptography*, CRC Press, Boca Raton, FL 1997
[Merkle 81]	Ralph C. Merkle, Martin E. Hellman: *On the Security of Multiple Encryption*, Internet, 1981
[Messerges 99]	Thomas S. Messerges, Ezzy A. Dabbish, Robert H. Sloan: *Investigations of Power Analysis Attacks on Smartcards*, USENIX Workshop on Smartcard Technology, Chicago, USA, 10–11 May 1999
[Meyer 82]	Carl H. Meyer, Stephen M. Matyas: *Cryptography*, Wiley, New York 1982

[Meyer 96]	Carsten Meyer: *Nur Peanuts – Der Risikofaktor Magnetkarte*, c't, July 1996
[Montenegro 99]	Sergio Montenegro: *Sichere und fehlertolerante Steuerungen*, Carl Hanser Verlag, Munich/Vienna 1999
[Moore 02]	Simon Moore, Ross Anderson, Paul Cunningham, Robert Mullins, George Tayler: *Improving Smart Card Security using Self-timed Circuits*, Internet, May 2002
[Müller-Maguhn 97a]	Andy Müller-Maguhn: *"Sicherheit" von EC-Karten,* Die Datenschleuder, Ausgabe 53, 1997
[Müller-Maguhn 97b]	Andy Müller-Maguhn: *EC-Karten Unsicherheit*, Die Datenschleuder, Ausgabe 59, 1997
[Myers 95]	Glenford J. Myers: *The Art of Software Testing*, 5th edn, Wiley, New York 1995
[Nebelung 96]	Brigitte Nebelung: *Das Geldbörsen-Konzept der ec-Karte mit Chip*, debis Systemhaus, Bonn 1996
[Nechvatal 00]	James Nechvatal, Elaine Barker, Lawrence Bassham, William Burr, Morris Dworkin, James Foti, Edward Roback, NIST: *Report on the Development of the Advanced Encryption Standard (AES)*, Internet, 2000
[Odlyzko 95]	Andrew. M. Odlyzko: *The Future of Integer Factorization*, AT&T Bell Laboratories, 1995
[Otto 82]	Siegfried Otto: *Echt oder falsch? Die maschinelle Echtheitserkennung*, Betriebswirtschaftliche Blätter, Heft 2, February 1982
[Peyret 97]	Patrice Peyret: *Which Smart Card Technologies will you need to Ride the Information Highway Safely?*, Gemplus, 1997
[Pfaffenberger 97]	Bryan Pfaffenberger: *Dictionary of Computer Terms*, Simon & Schuster/Macmillan, New York 1997
[Piller 96]	Ernst Piller: *Die "ideale" Geldbörse für Europa*, Card-Forum, Lüneburg 1996
[Pomerance 84]	C. Pomerance: *The Quadratic Sieve Factoring Algorithm*, Advances in Cryptology – Eurocrypt 84
[Press 92]	William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery: *Numerical Recipes in C – The Art of Scientific Computing*, 2nd edn, Cambridge University Press, Cambridge 1992
[Rivest 78]	Ronald L. Rivest, Adi Shamir, Leonard Adleman: *Method for Obtaining Digital Signatures and Public-Key Cryptosystems,* Internet, 1976

[Robertson 96] James Robertson, Suzanne Robertson: V*ollständige Systemanalyse*, Carl Hanser Verlag, Munich/Vienna 1996

[Rother 98a] Stefan Rother: *Prüfung von Chipkarten-Sicherheit,* Card-Forum, Lüneburg 1998

[Rother 98b] Stefan Rother: *Prüfung von Chipkarten-Sicherheit,* in *Tagungsband Chipkarten,* Vieweg Verlag, Braunschweig 1998

[RSA 97] RSA Data Security Inc.: *DES Crack Fact Sheet*, Internet, 1997

[Scherzer 00] Helmut Scherzer: *Chipkarten-Betriebssysteme – Gefahrenpotentiale und Sicherheitsmechanismen*, Forum IT-Sicherheit Smartcards, 14 March 2000

[Schief 87] Rudolf Schief: *Einführung in die Mikroprozessoren und Mikrocomputer*, 10th edn, Attempto Verlag, Tübingen 1987

[Schindler 97] Werner Schindler: *Wie sicher ist die PIN?*, speech presented at the ‘Kreditkartenkriminalität’conference, Heppenheim, October 1997

[Schlumberger 97] Schlumberger: *Cyberflex – Programmers Guide*, Version 6d, April 1997

[Schneier 96] Bruce Schneier: *Applied Cryptography*, 2nd edn, Wiley, New York 1996

[Schneier 99] Bruce Schneier: *Attack Trees – Modeling Security Threats*, Dr. Dobb’s Journal, December 1999

[Sedgewick 97] Robert Sedgewick: *Algorithmen*, 3rd edn, Addison-Wesley, Bonn/München/Reading, MA 1997

[SigG 01] Gesetz über Rahmenbedingungen für elektronische Signaturen, 22 May 2001

[Silverman 97] Robert D. Silverman: *Fast Generation of Random, Strong RSA Primes*, RSA Laboratories Crypto Byte, Internet, 1997

[Simmons 92] Gustavus J. Simmons (editor): *Contemporary Cryptology*, IEEE Press, New York 1992

[Simmons 93] Gustavus J. Simmons: *The Subliminal Channels in the U.S. Digital Signature Algorithm*, Proceedings of Symposium on the State and Progress of Research in Cryptography, Rome 1993

[Skorobogatov 02]	Sergei Skorobogatov, Ross Anderson: *Optical Fault Induction Attacks*, Internet, May 2002
[Sommerville 90]	Ian Sommerville: *Software Engineering*, Addison-Wesley, Wokingham 1990
[Steele 01]	Raymond Steele, Chin-Chun Lee, Peter Gould:*GSM, cdmaOne and 3G Systems*, Wiley, Chichester 2001
[Stix 96]	Gary Stix: *Herausforderung "Komma eins"*, Spektrum der Wissenschaft, February 1996
[Stocker 98]	Thomas Stocker: *Java for Smart Cards,* in: *Tagungsband Smart Cards,* Vieweg Verlag, Braunschweig 1998
[Tanenbaum 02]	Andrew S. Tanenbaum: *Moderne Betriebssysteme*, 3rd edn, Addison-Wesley Longman, Reading, MA 2002
[Thaller 93]	Georg Erwin Thaller: *Qualitätsoptimierung der Software-Entwicklung. Das Capability Maturity Model (CMM)*, Vieweg Verlag, Braunschweig 1993
[Tietze 93]	Ulrich Tietze, Christoph Schenk: *Halbleiter-Schaltungstechnik*, 10th edn, Springer-Verlag, Berlin 1993
[Vedder 97]	Klaus Vedder, Franz Weikmann: *Smart Cards – Requirements, Properties and Applications*, ESAT-COSIC course, Catholic University of Leuven, 1997
[Walke 00]	Bernhard Walke: *Mobilfunknetze und ihre Protokolle, Band 2: Bündelfunk, schnurlose Telefonsysteme, W-ATM, HIPERLAN, Satellitenfunk, UPT*, B. G. Teubner Verlag, Stuttgart 2000
[Weikmann 92]	Franz Weikmann: *SmartCard-Chips – Technik und weitere Perspektiven*, Der GMD-Spiegel 1'92, Gesellschaft for Mathematik und Datenverarbeitung, Sankt Augustin 1992
[Weikmann 98]	Franz Weikmann, Klaus Vedder: *Smart Cards Requirements, Properties and Applications*, in: *Tagungsband Smart Cards*, Vieweg Verlag, Braunschweig 1998
[Wiener 93]	Michael J. Wiener: *Efficient DES Key Search*, Crypto 93, Santa Barbara, CA 1993
[Yellin 96]	Frank Yellin: *Low Level Security in Java*, Internet, 1996
[Zieschang 98]	Thilo Zieschang: *Differentielle Fehleranalyse und Sicherheit von Chipkarten*, Internet, 1998

16.4 ANNOTATED DIRECTORY OF STANDARDS AND SPECIFICATIONS

This section contains an extensively commented directory of international standards, industry standards and specifications relevant to cards with and without chips. This directory primarily focuses on international standards, rather than local, country-specific standards. It lists standards produced by official standards organizations (such ANSI, CEN, ETSI and ISO), as well as quasi-standards that are relevant to smart cards, such as the EMV specification and Internet RFCs.

In addition to the annotated directory, Table 16.1 provides a summary of potentially helpful compilations, summaries and sources of standards and specifications related to specific subjects. Industry standards in particular are often available free of charge on the WWW. Unfortunately, this is not generally the case with official standards published by standards organizations.

Table 16.1 Summary of the most important Web servers for downloading standards and information related to smart cards

Standards or specification organization	Web server	Remarks
ANSI	[ANSI]	—
CEN	[CEN]	—
DIN	[DIN]	—
EMV	[EMVCO]	The specification can be downloaded from the Web server free of charge.
ETSI	[ETSI]	All ESTI standards (including those for GSM and UMTS) can be downloaded from the Web server free of charge.
FIPS	[NIST]	All FIPS standards can be downloaded from the Web server free of charge.
Global Platform	[Global Platform]	The specification can be downloaded from the Web server free of charge.
IEEE	[IEEE]	—
ISO/IEC	[ISO]	—
ITU	[ITU]	—
Java Card Forum	[JCF]	The specification can be downloaded from the Web server free of charge.
RFC	[RFC]	The specification can be downloaded from the Web server free of charge.
RSA Inc.	[RSA]	The specification can be downloaded from the Web server free of charge.
SEIS	[SEIS]	The specification can be downloaded from the Web server free of charge.

All standards and specifications are listed below in order of the name of the issuing organization and the numerical designation, ignoring prefixes (such as ‘pr’) and status indications (such as ‘DIS’). The date listed is the date at which the currently valid version first appeared. The most important standards for smart cards are marked with a ‘◆’.

A few brief remarks are in order regarding the naming of individual standards. First, extensions to ISO and ISO/IEC standards are usually contained in an amendment (Amd.). Each time a standard is revised, which normally takes place every five years, any amendments are incorporated into the main body of the standard as necessary. The title of a revised version of a standard thus differs from the title of its predecessor only by the year number and the sequential version number. New versions of CEN standards are identified in a similar manner. In the case of FIPS standards, the number of the revised edition forms part of the name of the standard (e.g., FIPS 140–2). Telecommunications standards from ETSI use a three-digit version number to distinguish different versions. In the case of industry standards, the revision level is indicated by a year number or a version number, depending on the publisher.

ANSI X9.8	Banking – Personal Identification Number Management and Security
– 1: 1995	Part 1: PIN Protection Principles and Techniques
– 2: 1995	Part 2: Approved Algorithms for PIN Encipherment
ANSI X 9.9: 1986	Financial Institution Message Authentication
ANSI X 9.17: 1985	Financial Institution Key Management
ANSI X 9.19: 1996	Financial Institution Retail Message Authentication
ANSI X 9.30	Public Key Cryptography Using Irreversible Algorithms for the Financial Services Industry
– 1: 1997	Part 1: The Digital Signature Algorithm (DSA)
– 2: 1997	Part 2: The Secure Hash Algorithm (SHA-1)
ANSI X 9.31: 1998	Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry
ANSI X9.55: 1997	Public Key Cryptography for the Financial Services Industry: Extensions to Public Key Certificates and Certificate Revocation Lists
ANSI X9.84: 2001	Biometric Information Management and Security *This very comprehensive standard specifies the basic architectural principles of a wide variety of biometric identification methods, as well as the requirements for the use, management and security of biometric data.*
ANSI X 3.92: 1981	Data Encryption Algorithm *Describes the DES algorithm.*

ANSI X 3.106: 1983	American National Standard for Information Systems – Data Encryption Algorithm – Modes of Operation
ANSI / IEEE 829: 1991	Standard for Software Test Documentation *Describes the methods and necessary documentation for testing software.*
ANSI / IEEE 1008: 1987	Standard for Software Unit Testing *Describes basic methods for testing software.*
ANSI / IEEE 1012: 1992	Software Verification and Validation Plans *Specifies the necessary test activities and test plans for software development. This standard is based on the waterfall model for software development.*
CCITT Z.100: 1993	CCITT Specification and Description Language (SDL)
CEPS, Version 2.1.3: 2001	Joint Specification for Common Electronic Purse Cards *CEPS is an important standard for electronic purses and is based on EN 1546. It provides the foundation for the majority of present and future European purse systems.*
Common Criteria, Version 2.1: 1999	*Identical to ISO/IEC 15 408* (*q.v.*)
DIN 9781-10: 1985	Büro- und Datentechnik; Identifikationskarten aus Kunststoff oder kunststofflaminiertem Werkstoff; Anforderungen an Echtheitsmerkmale *This very short standard defines the terms used in the context of authenticity features and lists general requirements for such features.*
DIN 44 300 – 1 . . . 9: 1988	Informationsverarbeitung – Begriffe *Defines many information technology concepts.*
EMV 2000	Integrated Circuit Card Specification for Payment Systems ◆ *This is the most important family of standards for smart cards used in payment systems. It is jointly published by EMVCo [EMV]. The family consists of four parts, called 'books', which deal with smart cards, associated debit and credit payment applications and related terminals.*[73]

[73] See also Section 12.4, 'The EMV Application'

Book 1 Version 4.0: 2000	Application Independent ICC to Terminal Interface Requirements *This part contains the specifications for the mechanical and electrical properties of the smart cards and terminals, including definitions of the activation and deactivation sequences, data transmission at the electrical level, the ATR and its associated parameters. In addition, it specifies the T = 0 and T = 1 transmission protocols, the APDU structure, logical channels and several fundamental card commands and application selection mechanisms.*
Book 2 Version 4.0: 2000	Security and Key Management *This part describes static and dynamic data authentication, PIN encryption and secure messaging. It also contains general conditions for managing the public keys of a payment system and requirements for terminal security, including associated key management.*
Book 3 Version 4.0: 2000	Application Specification *This part of the EMV specification defines a number of commands needed for smart cards and smart card applications for debit and credit cards and specifies transaction procedures. The appendix includes descriptions of all of the data objects, including their coding, specifications for the TLV coding of data and general approaches to integrating EMV smart cards into SET-based payment systems.*
Book 4 Version 4.0: 2000	Cardholder, Attendant and Acquirer Interface Requirements *Book 4 lists the mandatory and optional requirements for terminals that support EMV-compliant smart cards. This includes conceivable configurations, functional and security requirements for terminals, possible and permitted user messages including the character set used, and the interface to the acquirer. This standard also defines the basic features of the architecture of the terminal software and a model of a terminal-resident interpreter for executable program code. The appendix contains a listing of data objects relevant to the terminal and*

	recommendations for the technical design of the terminal, as well as examples of point-of-sale, cash dispenser and goods dispenser terminals.
EN 726	Identification Card Systems – Telecommunications Integrated Circuit(s) Card and Terminals
	Up until the mid-1990s, this family of standards occupied a leading position with regard to describing the functionality of smart card operating systems. However, it has now been completely supplanted by the ISO/IEC 7816 family of standards, the UICC standards and the EMV specifications, and is thus no longer significant.
– 1: 1994	Part 1: System Overview
– 2: 1995	Part 2: Security Framework
– 3: 1994	Part 3: Application Independent Card Requirements
	◆ *Defines file structures, commands, return codes and files for general-purpose applications, as well as basic mechanisms for smart cards for telecommunications applications. This standard is the ETSI counterpart of ISO/IEC 7816-4 and the corresponding framework for GSM 11.11.*
– 4: 1994	Part 4: Application Independent Card Related Terminal Requirements
– 5: 1999	Part 5: Payment Methods
	Defines various payment methods and associated file structures, data elements and processes for smart cards. The payment methods are intended to be used for telecommunication applications.
– 6: 1995	Part 6: Telecommunication Features
– 7: 1999	Part 7: Security Module
EN 753	Identification Card Systems – Intersector Thin Flexible Cards
– 1: 1997	Part 1: General Technical Specifications
– 2: 1997	Part 2: Magnetic Recording Technique
– 3: 1999	Part 3: Test Methods
EN 1038: 1995	Identification Card Systems – Telecommunication Applications – Integrated Circuit(s) Card Payphone
	Defines basic considerations for using smart cards with public card phones. This standard primarily contains references to previous standards, and it

	identifies the various places in the system where a security module can be effectively used to authenticate a phone card.
prEN 1105: 1995	Identification Card systems – General concepts applying to systems using IC cards in intersector environments – Rules for Inter-application Consistency *Defines the basic demands placed on a smart card in order to ensure interapplication use. It primarily contains references to prior standards, as well as various regulations for smart cards and terminals.*
prEN 1292: 1995	Additional Test Methods for IC Cards and Interface Devices *Defines tests for the general electrical parameters of smart cards and terminals and the basic data transfer between smart cards and terminals. This standard is an extension to ISO/IEC 10 373.*
EN 1332	Identification Card Systems – Man–Machine Interface
– 1: 1999	Part 1: Design Principles and Symbols for the User Interface
– 2: 1998	Part 2: Definition of a Tactile Identifier for ID-1 cards *Specifies a perceptible recess in ID-1 cards for detecting the orientation of the card.*
– 3: 1999	Part 3: Keypads
– 4: 1999	Part 4: Coding of User Requirements for People with Special Needs
EN 1362: 1997	Identification Card Systems – Device Interface Characteristics – Classes of Device Interfaces
EN 1387: 1996	Machine Readable Cards – Health Care Applications – Cards: General Characteristics
EN 1545-1: 1998	Identification Card Systems – Surface Transport Applications – Part 1: General
EN 1545-2: 1998	Identification Card Systems – Surface Transport Applications – Part 2: Transport Payment
prEN 1545-3: 1995	Identification Card Systems – Surface Transport Applications – Part 3: Tachograph

prEN 1545-4: 1995	Identification Card Systems – Surface Transport Applications – Part 4: Vehicle and Driver Licencing
EN 1546	Identification Card Systems – Inter-sector Electronic Purse
	◆ *The internationally most important standard for electronic purses, which forms the foundation for most purse systems. This family of standards has been kept relatively general, so it includes many options, but it is a very good and complete description of an electronic purse.*
– 1: 1999	Part 1: Definition, Concepts and Structures
	Defines terms used in the entire family of standards and describes the basic concepts and structures of intersector electronic purse systems.
– 2: 1999	Part 2: Security Architecture
	Describes the notation used for security mechanisms, the security architecture and associated procedures and mechanisms for intersector electronic purse systems.
– 3: 1999	Part 3: Data Elements and Interchanges
	Describes the data elements, files, commands and return codes used by all components of an intersector electronic purse system.
– 4: 1999	Part 4: Data Objects
	Describes the TLV mechanism for reading arbitrary data objects from files, and also provides a detailed presentation of the components and states of a state machine for a intersector electronic purse system. Also includes a list of tags for all data objects used.
EN 1867: 1997	Machine-readable Cards – Health Care Applications – Numbering System and Registration Procedure for Issuer Identifiers
EN 13 343	Identification Card Systems – Telecommunications IC Cards and Terminals – Test Methods and Conformance Testing for EN 726-3
– 1 prEN: 1998	Part 1: Implementation Conformance Statement (ICS) Pro-forma Specification
– 2 prEN: 1998	Part 2: Test Suite Structure and Test Purposes (TSS & TP)

– 3 prEN: 1998	Part 3: Abstract Test Suite (ATS) and Implementation Extra Information for Testing (IXIT) Pro-forma Specification
EN 13 344	Identification Card Systems – Telecommunications IC Cards and Terminals – Test Methods and Conformance Testing for EN 726-4
– 1 prEN: 1998	Part 1: Implementation Conformance Statement (ICS) Pro-forma Specification
– 2 prEN: 1998	Part 2: Test Suite Structure (TSS) and Test Purposes (TP)
– 3 prEN: 1998	Part 3: Abstract Test Suite (ATS) and Implementation Extra Information for Testing (IXIT) Pro-forma Specification
EN 13 345	Identification Card Systems – Telecommunications IC Cards and Terminals – Test Methods and Conformance Testing for EN 726-7
– 1 prEN: 1998	Part 1: Implementation Conformance Statement (ICS) pro-forma Specification
– 2 prEN: 1998	Part 2: Test Suite Structure and Test Purposes (TSS & TP)
– 3 prEN: 1998	Part 3: Abstract Test Suite (ATS) and Implementation extra Information for Testing (IXIT) pro-forma Specification
EN 1750: 1999	Identification Card Systems – Intersector Messages between Devices and Hosts – Acceptor to Acquirer Messages
EN 300812, Version 2.1.1: 2001	Terrestrial Trunked Radio (TETRA); Security Aspects; Subscriber Identity Module to Mobile Equipment (SIMME) Interface
ENV 1257	Identification Card Systems – Rules for Personal Identification Number Handling in Intersector Environments *Illustrates and explains security aspects related to using PINs, from transfering the PIN to the cardholder (PIN letter) to entering the PIN using a keypad (PIN pad).*
– 1 prENV: 1997	Part 1: PIN Presentation
– 2 prENV: 1997	Part 2: PIN Protection
– 3 prENV: 1997	Part 3: PIN Verification

ENV 13 729: 2000	Health Informatics – Secure User Identification – Strong Authentication using Microprocessor Cards
ETS 300 331: 1995	Digital Enhanced Cordless Telecommunications (DECT); DECT Authentication Module (DAM) *Describes the smart card (DAM) for the DECT system. Includes all associated commands, files, access conditions and authentication methods. Also defines the dimensions of the mini-ID and plug-in card formats. This standard is strongly based on the GSM 11.11 specification.*
FIPS 46-3: 1999	Data Encryption Standard (DES) ◆ *Describes the DES and triple-DES algorithms.*
FIPS 74: 1981	Guidelines for Implementing and Using the NBS Encryption Standard
FIPS 81: 1980	DES Modes of Operation
FIPS 140-2: 2001	Security Requirements for Cryptographic Modules ◆ *A fundamental, internationally used standard with regard to security requirements for security modules, which includes smart cards. It defines four different security levels for security modules and describes in detail seven security-related requirement areas. The content of this standard is very practically oriented and also addresses technical implementation details, such as criteria for the quality of random number generators.*
FIPS 180-1: 1995	Secure Hash Standard (SHA-1) ◆ *Describes the SHA-1 hash function.*
FIPS 186-2: 2000	Digital Signature Standard (DSS) ◆ *Describes the DSS algorithm.*
FIPS 197: 2001	Advanced Encryption Standard (AES) ◆ *Describes the AES algorithm.*
GSM 01.02, Version 6.0.1: 2001	Digital Cellular Telecommunications System (Phase 2+) (GSM); General Description of a GSM Public Land Mobile Network (PLMN) *Forms the basis for the architecture of all GSM mobile telecommunications networks.*
GSM 01.04, Version 8.0.0: 1999	Digital Cellular Telecommunications Systems (Phase 2) (GSM); Abbreviations and Acronyms

GSM 01.60, Version 6.0.0: 1998	Digital Cellular Telecommunications System (Phase 2+); General Packet Radio Service (GPRS) Requirements Specification of GPRS
GSM 02.09, Version 7.0.1: 1998	Digital Cellular Telecommunications Systems (Phase 2) (GSM); Security Aspects
GSM 02.17, Version 8.0.0: 1999	Digital Cellular Telecommunications Systems (Phase 2) (GSM); SIM Functional Characteristics *A short standard specifying the basic functionality required of a security module (SIM) for a GSM mobile telecommunications network. It is the GSM equivalent of the TS 21.111 standard for UMTS.*
GSM 02.19, Version 7.1.0: 1998	Digital Cellular Telecommunications System (Phase 2+) (GSM); Subscriber Identity Module Application Programming Interface (SIM API); Service Description; Stage 1 *A short standard listing all of the basic services of a language-independent API for executable program code (e.g., Java) in the SIM. Based on this standard, GSM 03.19 provides a detailed specification of a specific implementation to provide a Java Card API for SIMs.*
GSM 02.22, Version 7.0.0: 1999	Digital Cellular Telecommunications System (Phase 2+) (GSM); Personalization of GSM Mobile Equipment (ME); Mobile Functionality Specification *Describes mechanisms for personalizing and depersonalizing mobile equipment using specific data in the SIM (commonly known as SIM Lock).*
GSM 02.34, Version 6.0.0: 1997	Digital Cellular Telecommunications System (Phase 2+); High Speed Circuit Switched Data (HSCSD); Stage 1
GSM 02.48, Version 8.0.0: 2000	Digital Cellular Telecommunications System (Phase 2+) (GSM); Security Mechanisms for the SIM Application Toolkit; Stage 1 *A short standard describing the basic application-independent security mechanisms used with the SIM Application Toolkit as defined in GSM 11.14. Based on this standard, GSM 03.48 provides a detailed implementation specification.*
GSM 02.60, Version 6.3.0: 1997	Digital Cellular Telecommunications System (Phase 2+); General Packet Radio Service (GPRS); Service Description; Stage 1

GSM 03.19, Version 8.2.0: 2001	Digital Cellular Telecommunications System (Phase 2+); Subscriber Identity Module Application Programming Interface (SIM API); SIM API for Java Card; Stage 2 ◆ *Specifies a Java Card variant for use as a SIM with the SIM Application Toolkit, based on the Java Card 2.1 specifications. This standard is the key document for using Java Card in GSM. The basis for this is provided by GSM 02.19.*
GSM 03.20, Version 8.1.0: 1999	Global System for Mobile Communication (GSM) (Phase 2+); Security Related Network Functions
GSM 03.38, Version 7.2.0: 1999	Digital Cellular Telecommunications System (Phase 2+) (GSM); Alphabets and Language-specific Information *Specifies a GSM character set based on ASCII.*
GSM 03.40, Version 7.4.0: 2000	Digital Cellular Telecommunications System (Phase 2+) (GSM); Technical realization of the Short Message Service (SMS)
GSM 03.48, Version 8.7.0: 2001	Digital Cellular Telecommunications System (Phase 2+); Security Mechanisms for the SIM Application Toolkit; Stage 2 ◆ *Contains specifications for all security mechanisms needed for a connection between the background system and the SIM that is secure against eavesdropping and manipulation. Also describes the basic mechanism of a remote file management system using the SIM. The basis for this document is provided by GSM 02.48.*
GSM 09.91: 1995	European Digital Cellular Telecommunications System (Phase 2); Interworking Aspects of the Subscriber Identity Module – Mobile Equipment (SIM – ME) Interface between Phase 1 and Phase 2
GSM 11.10 Version 8.2.0: 2000	Digital Cellular Telecommunications System (Phase 2+) (GSM) – Mobile Station (MS) Conformance Specification *A very comprehensive test specification for GSM mobile stations.*
GSM 11.11 Version 8.5.0: 2001	Digital Cellular Telecommunications System (Phase 2+) – Specification of the Subscriber

	Identity Module – Mobile Equipment (SIM – ME) Interface ◆ *Specifies the physical and logical properties of the SIM by means of a description of the interface between the SIM and the GSM mobile telephone. Defines the dimensions of ID-1 and plug-in cards and the general mechanical parameters of the card and the contacts. Specifies general electrical parameters and the the structures and contents of the ATR and PPS. Also defines the possible data structures, security mechanisms, commands and return codes. Lists all data elements and files necessary for a SIM, along with typical command sequences. This standard is the GSM equivalent of the TS 31.101 and TS 31.102 UMTS standards.*
GSM 11.12 Version 4.3.1: 1998	Digital Cellular Telecommunications System (Phase 2); Specification of the 3 Volt Subscriber Identity Module – Mobile Equipment (SIM–ME) Interface *Specifies 3-V SIMs, including a compatibility list for SIMs programmed according to previous specifications. It only includes differences and extensions relative to GSM 11.11 with regard to 3V SIMs.*
GSM 11.13 Version 7.2.0: 2000	Digital Cellular Telecommunications System (Phase 2+); Test Specification for SIM API for Java Card *Specifies the test environment, test applications, test procedures, test coverage and individual test cases for the SIM API for Java Card as specified in GSM 03.19. The described tests exclusively address the IT aspects of a Java Card SIM for GSM. This standard provides an excellent and comprehensive illustration of how tests for a Java card can be described, constructed and executed.*
GSM 11.14 Version 8.8.0: 2001	Digital Cellular Telecommunications System (Phase 2+); Specification of the SIM Application Toolkit for the Subscriber Identity Module – Mobile Equipment (SIM – ME) Interface *Defines and extensively describes the SIM Application Toolkit (SAT) for SIMs. SAT describes an interface between the mobile telephone and the SIM for the partial control of the mobile*

	telephone by SIM-resident supplementary applications. This standard introduces proactive commands for the SIM and defines many new commands related to controlling the mobile telephone, such as display output, keypad polling and sending short messages. The UMTS equivalent of this standard is TS 31.111.
GSM 11.17 Version 7.0.2: 1998	Digital Cellular Telecommunications System (Phase 2+) (GSM); Subscriber Identity Module (SIM) Conformance Test Specification *Specifies the test environment, test equipment, test hierarchy and individual test cases for testing SIMs. The described tests exclusively address the electrical and IT aspects. Tests covering these aspects are specified in detail, including electrical power, data transmission, file management, commands and typical processes used in the GSM application. This specification is a very good and extensive illustration of how GSM tests can be described, constructed and executed. The UMTS equivalent of this standard is TS 31.122.*
GSM 11.18 Version 7.0.1: 1998	Digital Cellular Telecommunications System (Phase 2 +); Specification of the 1.8 Volt Subscriber Identity Module – Mobile Equipment (SIM – ME) Interface
GSM 11.19 Version 7.0.3: 1998	Digital Cellular Telecommunications System (Phase 2+) (GSM) – Specification of the Cordless Telephony System Subscriber Identity Module for both Fixed Part and Mobile
IEEE 828: 1990	Standard for Software Configuration Management Plans
IEEE 1363: 2000	Standard for RSA, Diffie-Hellman and Related Public-Key Cryptography ◆ *A very extensive and comprehensive standard, which addresses almost all aspects of asymmetric cryptographic algorithms, including generating keys, using digital signatures, key exchange and encryption.*
ISO 639	Codes for the Representation of Names of Languages
– 1: 2001	Part 1: Alpha-2 Code
– 2: 1998	Part 2: Alpha-3 Code

ISO/IEC 646: 1991	Information Technology – ISO 7-bit Coded Character Set for Information Interchange
ISO 3166	Codes for the Representation of Names of Countries and their Subdivisions
– 1: 1997	Part 1: Country Codes
– 2: 1998	Part 2: Country Subdivision Code
– 3: 1999	Part 3: Code for Formerly Used Names of Countries
ISO/IEC 4217: 1995	Codes for the Representation of Currencies and Funds
ISO 4909: 2000	Bank Cards – Magnetic Stripe Data Contents for Track 3
ISO/IEC 7501	Identification Cards – Machine Readable Travel Documents
– 1: 1997	Part 1: Machine Readable Passport
– 2: 1997	Part 2: Machine Readable Visas
– 3: 1997	Part 3: Official Travel Documents
ISO 7810: 1995	Identification Cards – Physical Characteristics *Describes the most important physical properties of cards without chips, and defines the ID-1, ID-2 and ID-3 card formats.*
ISO 7811	Identification Cards – Recording Technique *This family of standards is an important reference for the mechanical aspects of cards. It specifies the mechanical implementation of the essential card components.*
– 1: 1995	Part 1: Embossing *An exact definition of the 10 numeric characters and the basic method used to emboss cards.*
– 2: 2001	Part 2: Magnetic Stripe – Low Coercivity *Defines the size and position of the magnetic stripe on the card. Also specifies the physical properties of the magnetic material and the coding of the characters on the magnetic stripe.*
– 3: 1995	Part 3: Location of Embossed Characters on ID-1 Cards *Defines the possible locations for embossing on ID-1 cards.*

– 4: 1995	Part 4: Location of Read-only Magnetic Tracks – Tracks 1 and 2
	Defines the positions of the read-only tracks (tracks 1 and 2) on an ID-1card.
– 5: 1995	Part 5: Location of Read–Write Magnetic Track – Track 3
	Defines the position of the read/write track (track 3) on an ID-1card.
– 6: 2001	Part 6: Magnetic Stripe – High Coercivity
– 7 WD: 2001	Part 7: Magnetic Stripe – High Coercivity High Density
ISO 7812	Identification Cards
– 1: 2000	Part 1: Numbering System
	Specifies a numbering scheme for manufacturers of ID cards.
– 2: 2000	Part 2: Application and Registration Procedures
	Defines the registration authority and a form for registering applications. Also contains an algorithm for generating a Luhn checksum (modulo-10 checksum).
ISO 7813: 1995	Identification Cards – Financial Transaction Cards
	Defines the basic physical properties, dimensions and embossing of ISO 7810-compliant ID-1 cards for use in the financial transaction field. Also defines the data contents of tracks 1 and 2 of the magnetic stripe.
ISO/IEC 7816	Identification Cards – Integrated Circuit(s) Cards with Contacts
	◆ *The most important family of ISO standards for microcontroller smart cards. The first three parts primarily focus on the card and chip hardware. The remaining parts specify all mechanisms and properties of applications and operating systems for smart cards, as well as the associated informatics aspects.*
– 1: 1998	Part 1: Physical Characteristics
	Defines the physical characteristics of a card with a contact-type chip, as well as the tests to be used for such a card.
– 2: 1999	Part 2: Dimensions and Location of the Contacts

	Defines the sizes and positions of the contacts of a smart card, as well as the possible arrangements of the chip, magnetic stripe and embossing. Also describes the method to be used to measure the positions of the contacts on the smart card.
– 3: 1997	Part 3: Electronic Signals and Transmission Protocols
	◆ *The most important ISO standard for the general electrical parameters of a microcontroller smart card. It specifies all basic electrical characteristics, such as the supply voltage (3-V and 5-V), stopping the clock and reset behavior (cold and warm reset). It also defines the parameters, structure and possible sequences for the ATR and PPS. A large part of this standard deals with basic aspects of data transmission at the physical level (such as the divider) and the definition of the two transmission protocols (T = 0 and T = 1), and it includes extensive examples of communications sequences.*
– 4: 1995	Part 4: Inter-industry Commands for Interchange
	◆ *The most important application-level ISO standard for smart cards. It defines the file organization, file structures, security architecture, TPDUs, APDUs, secure messaging, return codes and logical channels. The majority of this standard is taken up by an extensive description of commands for smart cards. Fundamental smart card mechanisms for general industrial applications are also described.*
– 4 Amd. 1: 1997	Part 4 – Amendment 1: Use of Secure Messaging
– 5: 1994	Part 5: Numbering System and Registration Procedure for Application Identifiers
	Defines the numbering scheme for uniquely identifying national and international applications in smart cards. Also defines the exact data structure of the AID and describes the procedure for registering applications.
– 5 Amd. 1: 1996	Part 5 – Amendment 1: Registration of Identifiers
– 6 CD: 2001	Identification cards – Integrated Circuit(s) Cards with Contacts – Part 6: Inter-industry Data Elements

	Defines the data objects (DOs) and associated TLV tags for general industrial applications, and describes the associated TLV structures and procedures for reading data objects from smart cards.
– 7: 1999	Part 7: Inter-industry Commands for Structured Card Query Language (SCQL) *Defines supplementary smart card commands as an extension to ISO/IEC 7816-4. Defines the basic principles of a database system based on SQL, and specifies the commands for the associated SCQL accesses to smart cards.*
– 8: 1999	Part 8: Security Related Inter-industry Commands *This part of the family of standards is fully dedicated to functions and commands related to security. As an extension to ISO/IEC 7816-4, it defines additional mechanisms for secure messaging, as well as numerous commands for cryptographic functions, such as digital signatures, hash computation, MAC computation and the encryption and decryption of data.*
– 9: 2000	Part 9: Enhanced Inter-industry Commands *This standard is divided into three parts. The first part describes the life cycle of a smart card application at the file level in terms of states. The large second part describes access control objects (ACOs) that can be used to govern file accesses. The extensive third part defines search commands for file contents and administrative commands for creating and deleting files, which are necessary for managing applications.*
– 10: 1999	Part 10: Electronic Signals Answer to Reset for Synchronous Cards *For memory cards, this is the counterpart to Part 3 of this family of standards. It specifies the essential electrical characteristics of memory cards and defines the parameters and structure of the ATR and possible ATR procedures for synchronous cards.*
– 11 CD: 2000	Part 11: Card Structure and Enhanced Functions for Multiapplication Use *Defines commands for biometric user indentification and the associated data objects. In*

	addition, the appendix illustrates the basic features of methods for recording biometric data in the card (enrollment) and describes a scenario for verifying this biometric information.
– 15 CD: 2001	Part 15: Cryptographic Information Application *This part of the family, which is based on the PKCS #15 standard, defines all necessary data objects for an interoperable smart card for digital signatures. It includes descriptions of all data objects, directories and files needed for signature cards, as well as ASN.1 descriptions of all of the certificates, keys and other administrative data stored in the files.*
ISO 8372: 1987	Information Processing – Modes of Operation for a 64-Bit Block Cipher Algorithm ◆ *Defines the four operating modes for encryption algorithms using a 64-bit block size (e.g., DES): electronic codebook (ECB), cipher block chaining (CBC), output feedback (OFB) and cipher feedback (CFB). The block encryption modes described in ANSI X 3.106 and FIPS 81 form a subset of this standard.*
ISO 8583	Financial Transaction Card Originated Messages – Interchange Message Specifications *Standard for data transmission between a terminal and its host system. In Germany, communications between debit card terminals and the background system are based on this standard.*
– 1 CD: 1998	Part 1: Messages, Data Elements and Code Values
– 2: 1998	Part 2: Application and Registration Procedures for Institution Identification Codes (IIC)
– 3: 1988	Part 3: Maintenance Procedures for Messages, Data Elements and Code Values
ISO 8730: 1990	Banking – Requirements for Message Authentication *Fundamentals of securing data transmission and generating and testing MACs. The appendix contains extensive numerical examples, as well as a description of a DES pseudorandom number generator.*
ISO 8731	Banking – Approved Algorithms for Message Authentication

– 1: 1987	Part 1: DEA *A very short standard in which DEA is described as being suitable for MAC computation. Also contains a brief description of parity calculation for DES keys.*
– 2: 1992	Part 2: Message Authenticator Algorithm *Defines a fast algorithm for MAC computation in banking applications. The appendix contains numerical examples as well as an exact description of the algorithm.*
ISO 8732: 1988	Banking – Key Management *Extensive standard addressing principles and methods for key management among two or more participating parties using symmetric cryptographic algorithms.*
ISO/IEC 8824	Information technology – Open Systems Interconnection – Specification of Abstract Syntax Notation One (ASN.1) *Defines the basic ASN.1 coding rules.*
– 1: 1998	Part 1: Specification of Basic Notation
– 1: 1998 / Amd 1: 2000	Part 1 – Amendment 1: Relative Object Identifiers
– 1: 1998 / Amd 2: 2000	Part 1 – Amendment 2: ASN.1 Semantic Model
– 1: 1998 / Amd 3: 2000	Part 1 – Amendment 3: XML Value Notation
– 1: 1998 / Amd 4: 2000	Part 1 – Amendment 4: Version Number Support
– 2: 1998	Part 2: Information Object Specification
– 2: 1998 / Amd 1: 2000	Part 2 – Amendment 1: ASN.1 Semantic Model
– 2: 1998 / Amd 2	Part 2 – Amendment 2: XML Value Notation
– 3: 1998	Part 3: Constraint Specification
– 4: 1998	Part 4: Parameterization of ASN.1 Specifications
– 4: 1998 / Amd 1: 2000	Part 4 – Amendment 1: ASN.1 Semantic Model
ISO/IEC 8825	Information technology – Open Systems Interconnection – Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1) *Defines the ASN.1 data description language.*
– 1:1998	Part 1: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)
– 1:1998 / Amd 1:2000	Part 1 – Amendment 1: Relative Object Identifiers
– 2:1998	Part 2: Specification of Packed Encoding Rules (PER)

– 2:1998 / Amd 1:2000	Part 2 – Amendment 1: Relative Object Identifiers
– 3 FCD: 2001	Part 3: Specification of Encoding Control Notation (ECN)
– 3: FCD / Amd 1: 2001	Part 3 – Amendment 1: ASN.1 Extensibility Notation
– 4: WD 2000	Part 4: XML Encoding Rules (XER)
ISO/IEC 8859 - 1: 1998	Information Technology – 8-bit single-byte coded graphic character sets – Part 1: Latin Alphabet No. 1
ISO/IEC 9075	Information Technology – Database Languages – SQL2
	Defines the structured query language (SQL), database query language, which is a superset of the smart card database query language (SCQL).
– 1: 1999	Part 1: Framework (SQL/Framework)
– 1: 1999 / Amd 1: 2001	Part 1 – Amendment 1: On-Line Analytical Processing (SQL/OLAP)
– 2: 1999	Part 2: Foundation (SQL/Foundation)
– 2: 1999 / Amd 1: 2001	On-Line Analytical Processing (SQL/OLAP)
– 3: 1999	Part 3: Call-Level Interface (SQL/CLI)
– 4: 1999	Part 4: Persistent Stored Modules (SQL/PSM)
– 5: 1999	Part 5: Host Language Bindings (SQL/Bindings)
– 5: 1999 / Amd 1: 2001	Part 5 – Amendment 1: On-Line Analytical Processing (SQL/OLAP)
– 9: 2001	Part 9: Management of External Data (SQL/MED)
– 10: 2000	Part 10: Object Language Bindings (SQL/OLB)
– 11: CD 2001	Part 11: Information and Definition Schemas (SQL/schemata)
– 12: AWI 2000	Part 12: Replication
– 13: FCD 2001	Part 13: Java Routines and Types (SQL/JRT)
– 14: WD 2001	Part 14: XML-Related Specifications (SQL/XML) ISO/IEC 9126: 1991 Information Technology – Software product evaluation – Quality Characteristics and Guidelines for their Use
ISO/IEC 9126	Software Engineering – Product Quality
– 1: 2001	Part 1: Quality Model
– 2: CD 2001	Part 2: External Metrics
– 3: CD 2001	Part 3: Internal Metrics
– 4: CD 2001	Part 4: Quality in Use Metrics

ISO 9564	Banking – Personal Identification Number Management and Security
– 1: 1991	Part 1: PIN Protection Principles and Techniques
	Fundamentals of PIN selection, PIN management and PIN protection for general banking applications. The appendices define general requirements for PIN entry devices, among other things, as well as recommendations for the layout of suitable keypads and advice regarding erasing sensitive data on various media, such as magnetic tape, paper and semiconductor memories.
– 2: 1991	Part 2: Approved Algorithm(s) for PIN Encipherment
	A very short standard that defines DES as an algorithm for PIN encryption.
– 3: 2002	Part 3: PIN Protection Requirements for Offline PIN Handling in ATM and POS Systems
ISO/IEC 9646-3: 1998	Information Technology – Open Systems Interconnection – Conformance Testing Methodology and Framework – Part 3: The Tree and Tabular Combined Notation (TTCN)
	An extensive standard that describes a general high-level language for specifying tests. TTCN is used in a few isolated cases in the smart card environment.
ISO/IEC 9796	Information Technology – Security Techniques – Digital Signature Scheme giving Message Recovery
	Defines methods for generating and verifying digital signatures with message recovery. The appendix contains several numerical examples of key generation, signature generation and signature verification.
– 1: 1999	Part 1: Mechanisms using Redundancy
– 2: 1997	Part 2: Mechanisms using a Hash Function
– 3: 2000	Part 3: Discrete Logarithm Based Mechanisms
ISO/IEC 9797	Information Technology – Security techniques – Message Authentication Codes (MACs)
– 1: 1999	Part 1: Mechanisms using a Block Cipher
– 2: 1999	Part 2: Mechanisms using a Dedicated Hash Function

ISO/IEC 9798	Information Technology – Security techniques – Entity Authentication ◆ *This family of standards contains detailed descriptions of various cryptographic methods for authenticating one, two or three participating parties. It is the most important reference on the subject of authentication.*
– 1: 1997	Part 1: General *Defines the terms and notation used in the other parts of this family of standards.*
– 2: 1999	Part 2: Mechanisms using Symmetric Encipherment Algorithms *Specifies authentication methods based on symmetric cryptographic algorithms.*
– 3: 1998	Part 3: Mechanisms using Digital Signature Techniques *Specifies authentication methods based on asymmetric cryptographic algorithms.*
– 4: 1999	Part 4: Mechanisms using a Cryptographic Check Function *Specifies authentication methods based on cryptographic check functions.*
– 5: 1999	Part 5: Mechanisms using Zero Knowledge Techniques *Specifies authentication methods based on zero-knowledge techniques.*
ISO 9807: 1991	Banking and Related Financial Services – Requirements for Message Authentication (retail)
ISO/IEC 9979: 1999	Information Technology – Security techniques – Procedures for the Registration of Cryptographic Algorithms
ISO 9992	Financial Transaction Cards – Messages between the Integrated Circuit Card and the Card Accepting Device
– 1: 1990	Part 1: Concepts and Structures
– 2: 1998	Part 2: Functions, Messages (Commands and Responses), Data Elements and Structures *Defines commands, procedures, and data elements for smart cards used in financial transaction systems. Contains the definitions of*

	tags used in financial transaction systems and many cross-references to other standards in the ISO/IEC 7816 family.
– 4 DIS: 1993	Part 4: Common Data for Interchange
– 5 CD: 1991	Part 5: Organization of Data Elements
ISO/IEC 10 116: 1997	Information Technology – Security techniques – Modes of Operation for an *n*-bit Block Cipher Algorithm
	Describes the four standard opearting modes (ECB, CBC, CFB, OFD) for a block-oriented encryption algorithm. An appendix contains detailed comments regarding the use of each of the four modes, and another appendix contains corresponding numerical examples.
ISO/IEC 10 118	Information Technology – Security techniques – Hash Functions
	General principles of hash functions, as well as associated padding methods.
– 1: 2000	Part 1: General
– 2: 2000	Part 2: Hash Functions using an *n*-bit Block Cipher Algorithm
	Defines hash functions based on block-oriented encryption algorithms and describes algorithms with single-length and double-length keys. The appendix contains a numerical example for each type of key, based on the DES algorithm.
¯ 3: 1998	Part 3: Dedicated Hash Functions
¯ 4: 1998	Part 4: Hash Functions using Modular Arithmetic
ISO 10 202	Financial Transaction Cards – Security Architecture of Financial Transaction Systems using Integrated Circuit Cards
– 1: 1991	Part 1: Card Life Cycle
– 2: 1996	Part 2: Transaction Process
– 3: 1998	Part 3: Cryptographic Key Relationship
– 4: 1996	Part 4: Secure Application Modules
– 5: 1998	Part 5: Use of Algorithms
– 6: 1994	Part 6: Card holder Verification
– 7: 1998	Part 7: Key Management

	Defines general mechanisms for key management and key derivation. Both symmetrical and asymmetrical mechanisms are described.
– 8: 1998	Part 8: General Principles and Overview
ISO/IEC 10 373	Identification Cards – Test Methods
	◆ *Fundamental standard for card testing. Contains precise descriptions of test methods for card bodies and card bodies with implanted chips. The individual tests are described in detail, with many explanatory drawings.*
– 1: 1998	Part 1: General Characteristics Tests
– 2: 1998	Part 2: Cards with Magnetic Stripes
– 3: 2001	Part 3: Integrated Circuit(s) Cards with Contacts and Related Interface Devices
	Specifies the test environment, test methods and test procedures for electrical tests for contact-type smart cards. Also specifies detailed procedures for checking contact locations, electrical power, ATR and PPS data transmission and data transmission protocols.
– 4 CD: 1998	Part 4: Contactless Integrated Circuit Cards
– 5: 1998	Part 5: Optical Memory Cards
– 6: 2001	Part 6: Proximity Cards
– 7: 2001	Part 7: Vicinity Cards
ISO/IEC 10 536	Identification Cards – Contactless Integrated Circuit(s) Cards
	◆ *This standard descibes contactless smart cards whose application areas limit them to direct contact with the terminal.*
– 1: 2000	Part 1: Physical Characteristics
	Defines the physical characteristics of contactless smart cards and associated test methods.
– 2: 1995	Part 2: Dimension and Location of Coupling Areas
	Specifies the dimensions and locations of the coupling areas for contactless cards, and their use wih card terminals having card slots or surface interfaces.
– 3: 1996	Part 3: Electronic Signals and Reset Procedures
	Defines the electrical signals of the inductive and capacitive elements used to couple the smart card to the terminal.

– 4 CD: 1997	Part 4: Answer to Reset and Transmission Protocols *Specifies data transmission at the physical level, as well as the structure and parameters of the ATR and PPS for contactless smart cards. Defines the $T = 2$ data transmission protocol, with many sample scenarios for protocol procedures.*
ISO/IEC 10646	Information Technology – Universal Multiple-Octet Coded Character Set (UCS)
– 1: 2000	Part 1: Architecture and Basic Multilingual Plane
– 2: 2001	Part 2: Supplementary Planes
ISO 11 568	Banking – Key Management
– 1: 1994	Part 1: Introduction to Key Management
– 2: 1994	Part 2: Key Management Techniques for Symmetric Ciphers
– 3: 1994	Part 3: Key Life Cycle for Symmetric Ciphers
– 4: 1998	Part 4: Key Management Techniques for Public Key Cryptosystems
– 5: 1998	Part 5: Key Life for Public Key Cryptosystems
– 6: 1998	Part 6: Key Management Schemes
ISO/IEC 11 693: 2000	Identification Cards – Optical Memory Cards
ISO/IEC 11 694	Identification Cards – Optical Memory Cards and Devices – Linear Recording Method
– 1: 2000	Part 1: Physical Characteristics
– 2: 2000	Part 2: Dimensions and Location of the Accessible Optical Area
– 3: 2001	Part 3: Optical Properties and Characteristics
– 4: 1996	Part 4: Logical Data Structures
ISO/IEC 11 770	Information Technology – Security Techniques – Key Management
– 1: 1996	Part 1: Framework
– 2: 1996	Part 2: Mechanisms using Symmetric Techniques
– 3: 1999	Part 3: Mechanisms using Asymmetric Techniques
ISO/IEC 12 207: 1995	Information technology – Software Life Cycle Processes
ISO/IEC 13 239: 2000	Information Technology – Telecommunications and Information Exchange between Systems – High-level Data Link Control (HDLC) Procedures

ISO 13 491	Banking – Secure Cryptographic Devices
– 1: 1998	Part 1: Concepts, Requirements and Evaluation Methods
– 2: 2000	Part 2: Security Compliance Checklists for Devices used in Magnetic Stripe Card Systems
ISO/IEC 13 888	Information Technology – Security Techniques – Non-repudiation
– 1: 1997	Part 1: General
– 2: 1998	Part 2: Mechanisms using Symmetric Techniques
– 3: 1997	Part 3: Mechanisms using Asymmetric Techniques
ISO/IEC 14 443	Identification Cards – Contactless Integrated Circuit(s) Cards – Proximity Cards ◆ *This standard describes contactless smart cards that can be used at a distance of up to several tens of centimeters from a terminal.*
– 1: 2000	Part 1: Physical Characteristics
– 2: 2001	Part 2: Radio Frequency Power and Signal Interface
– 3: 2001	Part 3: Initialization and Anticollision
– 4: 2001	Part 4: Transmission Protocol
ISO/IEC 14 888	Information Technology – Security Techniques – Digital Signature with Appendix *This standard specifies basic mechanisms and methods for digital signatures with appendix. It is independent of any particular asymmetric cryptographic algorithm.*
– 1: 1998	Part 1: General
– 2: 1999	Part 2: Identity-based Mechanisms
– 3: 1998	Part 3: Certificate-based Mechanisms
ISO/IEC 15 292: 2001	Information Technology – Security Techniques – Protection Profile Registration Procedures
ISO/IEC 15 408	Information Technology – Security Techniques – Evaluation Criteria for IT Security
– 1: 1999	Part 1: Introduction and General Model
– 2: 1999	Part 2: Security Functional Requirements
– 3: 1999	Part 3: Security Assurance Requirements

ISO/IEC 15 693	Identification Cards – Contactless Integrated Circuit(s) Cards – Vicinity Cards *This standard describes contactless smart cards that can be used at a distance of up to one meter from a terminal.*
– 1 CD: 2000	Part 1: Physical Characteristics
– 2 WD: 2000	Part 2: Air Interface and Initialization
– 3 WD: 2001	Part 3: Anticollision and Transmission Protocol
– 4 WD: 1996	Part 4: Extended Command Set and Security Features
ISO 15 782	Banking – Certificate Management for Financial Services
– 1 DIS: 2000	Part 1: Public Key Certificates
– 2: 2001	Part 2: Certificate Extensions
ISO/IEC 15 946	Information Technology – Security Techniques – Cryptographic Techniques based on Elliptic Curves
– 1 FDIS: 2001	Part 1: General
– 2 FDIS: 2001	Part 2: Digital Signatures
– 3 FDIS: 2001	Part 3: Key Establishment
– 4 CD: 2000	Part 4: Digital Signatures giving Message Recovery
ISO 17 090	Public Key Infrastructure
– 1 CD: 2001	Part 1: Framework and Overview
– 2 CD: 2001	Part 2: Certificate Profile
– 3 CD: 2001	Part 3: Policy Management of Certification Authority
ITU X.509: 2000	Information Technology – Open Systems Interconnection – The Directory: Authentication Framework ◆ *Specifies the structure and coding of certificates. Internationally, it is the most commonly used basis for certificate structures, and it is identical to ISO/IEC 9594-8.*
Java Card 2.1: 2000	◆ *This industrial standard forms the basis for Java Card. It was generated by the Java Card Forum and published by the Sun Corporation. All of the standards in this family are mutually complementary and address various aspects of Java Card implementations.*

– Application Programming Interface	
	Specifies the complete interface (API) available to an applet in a Java Card environment. It esssentially consists of a comprehensive listing of all classes and interfaces of the Java Card API.
– Runtime Environment (JCRE) Specification	
	Specifies the Java Card runtime environment, which essentailly consists of the Java virtual machine and the Java Card API. It addresses the following topics in detail: the lifetime of the virtual machine, the lifetimes of applets, selecting applets, transient objects, sharing objects, transactions, the extent to which transactions are atomic and installing applets.
– Virtual Machine Specification	
	Specifies the Java Card virtual machine, including its detailed architecture, its instruction set and the format of CAP files
Multifunktionale Karten Terminals Spezification, Version 1.0: 1999	*The MKT specification, which is published by Teletrust Deutschland, is the quasi-standard in Germany for connecting terminals to PCs.*
– Part 1	MKT-Basiskonzept
– Part 2	CT-ICC-Interface – MKT-Schnittstelle für kontaktorientierte Chipkarten mit synchroner und asynchroner Übertragung
– Part 3	CT-API 1.1 – Anwendungsunabhängiges Card Terminal Applikation Programming Interface
– Part 4	CT-BCS – Anwendungsunabhängiges Card Terminal Basic Command Set
– Part 5	Chipkarten mit synchroner Übertragung – ATR und Datenbereiche
– Part 6	Chipkarten mit synchroner Übertragung – Übertragungsprotokolle
– Part 7	Chipkarten mit synchroner Übertragung – Anwendung von Interindustry Commands
OCF – API Docs V1.2: 2001	
OCF – Programmer’s Guide V 1.2: 2001	

Open Platform Card Specification 2.1: 2001	
	◆ *The most important specification with regard to managing applications in multiapplication smart cards. This very comprehensive specification contains a detailed presentation of the software and security architectures of multiapplication smart cards and a thorough description of the commands needed for this purpose. The appendix includes the specification of an API for application management with Java Card, which has become the* de facto *standard for this type of smart card.*[74]
PC/SC V1.0: December 1997	Interoperability Specification for ICCs and Personal Computer Systems
	This extensive, detailed specification forms the basis for linking smart cards and terminals to the resource management system of 16-bit and 32-bit Microsoft operating systems.
– 1	Part 1: Introduction and Architecture Overview
– 2	Part 2: Interface Requirements for Compatible IC Cards and Readers
– 3	Part 3: Requirements for PC-Connected Interface Devices
– 4	Part 4: IFD Design Considerations and Reference Design Information
– 5	Part 5: ICC Resource Manager Definition
– 6	Part 6: ICC Service Provider Interface Definition
– 7	Part 7: Application Domain and Developer Design Considerations
– 8	Part 8: Recommendations for ICC Security and Privacy Devices
PKCS	*The Public Key Cryptography Standards (PKCS) are industry standards published by RSA Inc. that focus on the use of asymmetric cryptographic algorithms.*
– PKCS #1 V 2.1: 2001	RSA Encryption Standard
	◆ *Describes mechanisms for encryption and decryption using the RSA algorithm.*
– PKCS #3 V 1.4: 1993	Diffie–Hellman Key-Agreement Standard
	Describes the mechanism of a key exchange procedure between two parties using the Diffie–Hellman procedure.

[74] See also Section 5.11, 'Open Platform'

– PKCS #5 V 2.0: 1999	Password-Based Cryptography Standard *Contains recommendations for the implementation of encryption, key derivation and MAC generation based on keys generated from passwords.*
– PKCS #11 V 2.11: 2001	Cryptographic Token Interface Standard ◆ *The* de facto *international standard for an API for invoking cryptographic functions. This API is called 'Cryptoki' (cryptographic token interface) and includes functions such as RC2, RC4, RC5, MD5, SHA-1, DES, triple-DES, IDEA, RSA, DSA, MAC computation and key generation for a wide variety of cryptographic algorithms.*
– PKCS #13 V 1.0: 1998	Elliptic Curve Cryptography Standard
– PKCS #14 V 1.0 (Proposal: 1998)	Pseudorandom Number Generation Standard *This unfinished standard with a relatively small scope contains suggestions for the conceptual design of random number generators.*
– PKCS #15 V 1.1: 2000	Cryptographic Token Information Format Standard ◆ *Internationally, this is the* de facto *standard for the data objects needed for an interoperable smart card for digital signatures. It includes descriptions of all directories and files needed for a signature card and ASN.1 descriptions of all certificates, keys and administrative data stored in the files.*
RFC 1319: 1992	The MD2 Message-Digest Algorithm
RFC 1320: 1992	The MD4 Message-Digest Algorithm
RFC 1321: 1992	The MD5 Message-Digest Algorithm
RFC 1750: 1994	Randomness Recommendations for Security *Describes the operating principles of various types of random number generators, and based on these principles, recommends methods for designing high-quality pseudorandom number generators for PCs.*
RFC 2706: 1999	ECML V1: Field Names for E-Commerce
SET Book 1, Version 1.0: 1997	Secure Electronic Transaction Specification, Book 1: Business Description

SET Book 2, Version 1.0: 1997	Secure Electronic Transaction Specification, Book 2: Programmer's Guide
SET Book 3, Version 1.0: 1997	Secure Electronic Transaction Specification, Book 3: Formal Protocol Definition
TIA/EIA/IS-820: 2000	Removable User Identity Module (R-UIM) for TIA/EIA Spread Spectrum Standards
TIA/EIA/IS-820-1: 2001	Removable User Identity Module (R-UIM) for TIA/EIA Spread Spectrum Standards, Addendum 1
TIA/EIA/IS-839: 2000	R-UIM Overview, Operation, and File Structure Support in TIA/EIA-136
TR 33.900, V 1.2.0: 2000	3rd Generation Partnership Project; Technical Specification Group SA WG3; A Guide to 3rd Generation Security Architecture *An overview of the security architecture, available security functions and possible attack scenarios for 3G mobile telecommunications networks. Details related to security are described in the relevant standards (TS 33.102, TS 33.120 and TS 21.133).*
TS 21.111, Version 4.0.0: 2001	3rd Generation Partnership Project; Technical Specification Group Terminals; USIM and IC card requirements *A short standard that specifies the basic functionality required for a security module (USIM) for a UMTS mobile communications network. This standard is the UMTS equivalent of the GSM 02.17 standard.*
TS 21.133, Version 4.0.0: 2001	Universal Mobile Telecommunications System (UMTS); 3G Security; Security Threats and Requirements
TS 22.038, Version 4.1.0: 2001	3rd Generation Partnership Project; Technical Specification Group Terminals; USIM/SIM Application Toolkit (USAT, SAT); Service description; Stage 1
TS 22.112, Version 5.0.0: 2001	Technical Specification; 3rd Generation Partnership Project; Technical Specification Group Terminals; USAT Interpreter – Stage 1
TS 23.038, V 4.3.0: 2001	3rd Generation Partnership Project; Technical Specification Group Terminals; Alphabets and language-specific information *Specifies the character coding used for SMS and USSD and the character sets used for UMTS.*

TS 23.040, V 4.3.0: 2001	3rd Generation Partnership Project; Technical Specification Group Terminals; Technical realization of the Short Message Service (SMS)
TS 23.048, Version 5.1.0: 2001	3rd Generation Partnership Project; Technical Specification Group Terminals; Security Mechanisms for the (U)SIM Application Toolkit; Stage 2
TS 31.102, Version 4.2.0: 2001	3rd Generation Partnership Project; Technical Specification Group Terminals; Characteristics of the USIM Application ◆ *Specifies the logical characteristics of the USIM smart card application by describing the interface between the USIM and the UMTS mobile telephone. Includes detailed descriptions of all files and their data objects, definitions of several somewhat less UMTS-specific commands and examples of command sequences for typical processes. Together with TS 31.101, it is the UMTS equivalent of the GSM GMS 11.11 standard.*
TS 31.110, Version 4.0.0: 2001	3rd Generation Partnership Project; Technical Specification Group Terminals; Numbering system for telecommunication IC card applications *Future versions of this standard will be published as TS 101.220.*
TS 31.111, Version 4.4.0: 2001	3rd Generation Partnership Project; Technical Specification Group Terminals; USIM Application Toolkit (USAT) *Defines and extensively describes the USIM Application Toolkit for USIMs. The USAT describes an interface between the mobile telephone and the USIM that allows supplementary applications in the USIM to assume partial control of the telephone. It introduces proactive commands for the USIM and defines many new commands related to control of the telephone for functions such as display output, keypad polling and sending short messages. The GSM equivalent of this standard is GSM 11.14.*
TS 31.112, Version 5.0.0: 2001	3rd Generation Partnership Project; Technical Specification Group Terminals; USAT Interpreter Architecture Description; Stage 2

TS 31.113, Version 5.0.0: 2001	3rd Generation Partnership Project; Technical Specification Group Terminals; USAT Interpreter Byte Codes
TS 31.114, Version 1.1.0: 2002	3rd Generation Partnership Project; Technical Specification Group Terminals; USAT Interpreter Protocol and Administration
TS 31.121, Version 4.0.0: 2001	3rd Generation Partnership Project; Technical Specification Group Terminals; UICC-Terminal Interface; USIM Application Test Specification
TS 31.122, Version 3.0.0: 2000	3rd Generation Partnership Project; Technical Specification Group Terminals; USIM Conformance Test Specification *Specifies the test environment, test equipment, test hierarchy and individual test cases for testing USIMs. The described tests exclusively address electrical and informatics aspects. Detailed specifications are provided for tests covering a wide range of subjects, such as electrical power, data transmission, file management, commands and typical processes in the UMTS application. This standard is a very good example of how USIM tests can be described, constructed and executed. It is the USIM equivalent of the GSM 11.17 standard for testing SIMs.*
TS 31.900, Version 3.1.0: 2001	3rd Generation Partnership Project; Technical Specification Group Terminals; SIM/USIM Internal and External Interworking Aspects
TS 33.102, Version 4.1.0: 2001	3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Security Architecture *A key standard for the entire security architecture of a UMTS mobile telecommunications network with regard to network access, authentication, confidentiality and data integrity. Includes complete descriptions, independent of any specific cryptographic algorithm, of network security functions, authentication protocols and encryption methods, as well as generating authentication vectors and the key derivations that are used.*
TS 33.103, Version 4.1.0: 2001	3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G security; Integration guidelines

TS 33.105, Version 4.1.0: 2001	3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Cryptographic Algorithm Requirements
TS 33.120, Version 4.0.0: 2001	Universal Mobile Telecommunications System (UMTS); 3G Security; Security Principles and Objectives
TS 35.205, Version 4.0.0: 2001	Universal Mobile Telecommunications System (UMTS); 3G Security; Specification of the MILENAGE algorithm set: An example algorithm set for the 3GPP Authentication and Key Generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 1: General
TS 35.206, Version 4.0.0: 2001	Universal Mobile Telecommunications System (UMTS); 3G Security; Specification of the MILENAGE algorithm set: An example algorithm set for the 3GPP Authentication and Key Generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 2: Algorithm Specification
TS 35.207, Version 4.0.0: 2001	Universal Mobile Telecommunications System (UMTS); 3G Security; Specification of the MILENAGE algorithm set: An example algorithm set for the 3GPP Authentication and Key Generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 3: Implementors' Test Data
TS 35.208, Version 4.0.0: 2001	Universal Mobile Telecommunications System (UMTS); 3G Security; Specification of the MILENAGE algorithm set: An example algorithm set for the 3GPP Authentication and Key Generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 4: Design Conformance
TS 35.909, Version 4.0.0: 2001	Universal Mobile Telecommunications System (UMTS); 3G security; Report on the design and evaluation of the MILENAGE algorithm set; Deliverable 5: An example algorithm for the 3GPP Authentication and Key Generation functions
TS 42.009, V4.0.0: 2001	3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Digital cellular telecommunications system (Phase 2+); Security aspects *Fundamental document containing an overview of the important security aspects of a PLMN.*

TS 51.011, Version 4.2.0: 2001	3rd Generation Partnership Project; Technical Specification Group Terminals; Specification of the Subscriber Identity Module – Mobile Equipment (SIM – ME) interface *Identical to GSM 11.11 in the new ETSI numbering system.*
TS 101.220, V 4.0.0: 2001	Integrated Circuits Cards (ICC); ETSI numbering system for telecommunication application providers *Specifies the AIDs, PIX and TAR for the SIM, TETRA-SIM and USIM. Also defines the code space of the PIX for the various types of supplemenatary applications of this type of telecommunications smart card.*
TS 102.221, Version 4.4.0: 2001	Smart cards; UICC–Terminal interface; Physical and logical characteristics ◆ *Specifies the logical characteristics of a USIM by means of a description of the interface between the USIM and the UMTS mobile telephone. Includes definitions of the ID-1 and plug-in card formats and specifies the general mechanical parameters of the card and its contacts, as well as all general electrical parameters. It also specifies the structure and data content of the ATR and PPS and defines transmission protocols, file structures, security mechanisms, commands and return codes. In addition, it lists all files and associated data objects that are independent of any particular telecommunications application. This standard forms the basis for smart card operating systems for the USIM. It is complemented by TS 31.103, which addresses all application-specific components of a USIM. These two standards form the UMTS equivalent of the GMS 11.11 standard.*
TS 102.222, Version 3.3.0: 2001	Integrated Circuit Cards (ICC); Administrative commands for telecommunications applications *Specifies the administrative commands for file management and associated security conditions for use with telecommunications smart cards.*

TS 102.223, Version 4.1.0: 2001	Smart Cards; Card Application Toolkit (CAT) *Defines and thoroughly describes a generic application toolkit for telecommunications smart cards. CAT describes an interface between the mobile telephone and the smart card that allows supplementary applications in the smart card to assume partial control of the telephone. This standard defines commands related to controlling the telephone for functions such as display output, keypad polling and sending short messages. It forms the basis for other standards, such as GSM 11.14 and TS 31.111.*
TS 102.224, Version 1.0.0: 2001	Smart Cards; Security mechanisms for the Card Application Toolkit; Functional requirements
TS 102.225, Version 1.0.0: 2001	Smart Cards; Secured packet structure for UICC applications
TS 102.226, Version 1.0.0: 2001	Smart Cards; Remote APDU Structure for UICC based Applications
TS 102.230, Version 4.0.0: 2001	Smart Cards; UICC-Terminal Interface; Physical, Electrical and Logical Test Specification *Specifies physical and electrical tests for UICCs, and describes basic tests for the communications link to the UICC and tests for the T = 0 and T = 1 transmission protocols.*
TS 102.240, Version 1.0.0: 2001	Smart Cards; UICC Application Programming Interface (UICC API); Service description
TS 102.241, Version 1.0.0: 2001	UICC Application Programming Interface (UICC API); UICC API for Java Card
TS 123.002, Version 4.4.0: 2002	Universal Mobile Telecommunications System (UMTS); Network architecture
Unicode Standard, Version 3.1.1: 2001	
Universal Serial Bus Specification, Revision 2.0, 2000	*This very comprehensive specification forms the basis for the USB interface.*
Wireless Application Protocol Identity Module Specification, Version 260: July 2001	*Specifies the physical and electrical properties of a WIM, which is the digital signature application for telecommunications smart cards. Lists all mechanisms, commands, data objects and files needed for a WIM application.*

16.5 CODING OF DATA OBJECTS

Additional tables of tags for data objects can be found in Chapter 5, which describes accesses to smart card resources in accordance with ISO/IEC 7816-9.

16.5.1 Data objects compliant with ISO/IEC 7816-4

Table 16.2 The coding of a number of important data objects as defined in ISO/IEC 7816-4

Tag	Data object	Template	Length (bytes)	Description
'62'	File control parameters (FCP) template	---	---	The FCP template contains file control parameters (FCP).
'64'	File management data (FMD) template	---	---	The FMD template contains file management data (FMD).
'6F'	File control information (FCI) template	---	---	The FCI template contains file control parameters (FCP) and file management data (FMD).
'80'	Number of data bytes in the file, excluding structure data	'62'	2	For EFs with transparent structure.
'81'	Number of data bytes in the file, including structure data	'62'	2	For all files.
'82'	File description	'62'	1–4	File access: °00-- ----°: file is not shareable (concurrent access via several logical channels not possible) °01-- ----°: file is shareable (concurrent access via several logical channels is possible) File type: °--00 0---°: working EF °--00 1---°: internal EF °--11 1---°: DF EF structure: °---- -000°: no data °---- -001°: transparent °---- -010°: linear fixed °---- -011°: linear fixed, content simple-TLV-coded °---- -100°: linear variable

Table 16.2 (*Cont.*)

				°---- -101°: linear variable, content simple-TLV-coded °---- -110°: cyclic °---- -111°: cyclic, content simple-TLV-coded
'83'	FID	'62'	2	For all files.
'84'	DF name	'62'	1–16	For DFs.
'86'	Security attribute	'62'	variable	
'88'	Short file identifier (SFI)	'62'	1	Definition of EFs from ISO/IEC 7816-9. SFI = b8 . . . b4 || °000°
'8A'	Life cycle status indicator (LCSI tag)	'62'	1	Bit coding specified in ISO/IEC 7816-9. °0000 0000°: no data °0000 0001°: creation state °0000 0011°: initialization state °0000 01x1°: operational state: activated °0000 01x0°: operational state: deactivated °0000 11xx°: termination state °yyyy xxxx°: y $\neq$ 0, proprietary

16.5.2 Data objects compliant with ISO/IEC 7816-6

Table 16.3 The coding of a number of important data objects as defined in ISO/IEC 7816-6

Tag	Data object	Template	Length (bytes)	Description
'4F'	AID	'61'/'6E'	5–16	—
'50'	Application name	'61'/'6E'	variable	
'59'	Card expiry date	'66'	3	Format: YYMMDD
'5B'	Name	'65'	39 max.	
'5F24'	Application expiry date	'6E'	3	Format: YYMMDD
'5F25'	Date of issue of the card	'6E'	3	Format: YYMMDD
'5F26'	Date of issue of the application	'66'	3	Format: YYMMDD
'5F28'	Country identifier	'66'	2	Format: 3 digits, coded per ISO 3166
'5F2A'	Currency identifier	'6E'	2	Format: 3 digits, coded per ISO 3166
'5F2B'	Date of birth	'65'	4	Format: YYYYMMDD
'5F42'	Address	'65'	variable	Address of a person
'5F4D'	Chip manufacturer	'66'	1	See Section 16.5.3.

16.5.3 Data objects for chip manufacturers as specified by ISO/IEC 7816-6

Table 16.4 Coding of data objects for chip manufacturers as defined in ISO/IEC 7816-6. This table provides a good overview of the manufacturers of smart card microcontrollers. The tag for chip manufacturers is ‘5F4D’.

Code	Company	Code	Company
'01'	Motorola	'0D'	Mitsubishi Electric
'02'	ST Microelectronics	'0E'	Samsung Electronics
'03'	Renesas	'0F'	Hyundai Electronics Industries
'04'	Philips Semiconductors	'10'	LG-Semiconductors
'05'	Infineon Semiconductors	'11'	Emosyn-EM Microelectronics
'06'	Cylinc	'12'	Inside Technologies
'07'	Texas Instrument	'13'	ORGA Kartensysteme
'08'	Fujitsu	'14'	Sharp Corporation
'09'	Matsushita Electronic	'15'	ATMEL
'0A'	NEC	'16'	EM Microelectronic-Marin
'0B'	Oki Electric	'17'	KSW Microtec
'0C'	Toshiba	'19'	Xicor

16.6 REGISTRATION AUTHORITIES FOR RIDs

The form for registering an RID is located in the appendix of the ISO/IEC 7816-5 standard. An application for an international RID is normally made via the relevant national authority, and there is a fee. The addresses of national registration authorities, as well as the registration procedures for RIDs, can usually be obtained from national standardization bodies.

Table 16.5 Registration authorities for RIDS compliant with ISO/IEC 7816-5

Region	Organization
International	TeleDanmark KTAS attn: ISO/IEC 7816-5 Registration Authority Teglholmsgade 1 1790 Copenhagen V Denmark
Germany	RID German National Registration Authority c/o GMD Bruno Struif Rheinstraße 75 D - 64 295 Darmstadt, Germany

16.7 SELECTED RIDS

Table 16.6 lists a number of examples of publicly known RIDs and AIDs. Unfortunately, RIDs are treated as confidential by registration authorities, so the list is not very long.

Table 16.6 Selected RIDs of typical smart card applications and organizations that use smart cards

Smart card applications	AID (application identifier) = RID || PIX
3GPP (UICC, USIM, USAT)	RID = 'A0 00 00 00 87' PIX = specific to the card issuer

Eurocheque card with chip in Germany	RID = 'D2 76 00 00 25'
	PIX = '45 50 01 00'

ETSI (SIM, SAT, Java Card SIM API, TETRA)	RID = 'A000000009'

FINEID
(Finnish personal ID card)
	RID = 'A0 00 00 00 63'
	PIX = '50 4B 43 53 2D 31 35'

Giesecke & Devrient	RID = 'D2 76 00 00 05'

PKCS #15	RID = 'A0 00 00 00 63'
	PIX = '50 4B 43 53 2D 31 35' = "PKCS-15"

WIM	RID = 'A0 00 00 00 63'
	PIX = '57 41 50 2D 57 49 4D' = "WAP-WIM"

Wolfgang Rankl	RID = 'D2 76 00 00 60'

16.8 TRADE FAIRS, CONFERENCES AND CONVENTIONS

Table 16.7 lists trade fairs, congresses and conventions that have smart cards or related subjects as at least one of their major themes. The listed places and dates are typical for the past several years, but they can change in the future, depending on the event organizer. As can be seen, a traveler with an interest in the subject can visit an event in a different country every month of the year.

Table 16.7 Selected annual events related to smart cards and cryptology

Event name	Place	Date
Asia Crypt	Asia	Fall
Card Tech / Secure Tech [CTST]	USA	September
Cards Africa	South Africa (Johannesburg)	November
Cards Asia	Singapore	February
Cards Australia	Australia (Melbourne)	August
Cards Latin America	Chile (Santiago de Chile)	July
Cartes	France (Paris)	October
CeBit	Germany (Hanover)	March
Crypto	USA (Santa Barbara, California)	Summer
Euro Crypt	Europe	Spring
GSM World Congress	France (Cannes)	February
Java One	San Francisco, USA	June
OmniCard	Germany (Berlin)	January
Smart Card	Great Britain (London)	February

16.9 WORLD WIDE WEB ADDRESSES

The following list of World Wide Web addresses does not claim to be complete. It should be seen as a cross-section of the various companies and institutions that are active in the field of smart cards. The listed addresses are thus entirely suitable for use as starting points for further research. Thanks to the hypertext structure of HTML documents, many of the listed sites contain links to other interesting documents and World Wide Web locations. Large collections of links are explicitly identified as 'link farms'.

When using this list, you should bear in mind that the Internet is very dynamic, so addresses can very quickly become outdated. This is also why we do not list specific documents, but have limited the listings to subdirectories. Even these are frequently changed when a Web server is reorganized, so in case of doubt we recommend that you use the address up to the organization or country code (**.com, *.de* and so on). After this, you can manually select currently valid directories on the Web server via the home page.

The classification of the Internet addresses and firms is based on their principal areas of activity. However, many of the listed firms are active in several of the indicated areas; this is normally shown explicitly. To the extent that it makes sense to do so, the country in which the firm or organization is located is also noted.

As a rule, you can find the postal address of a firm and the telephone number of a contact person on the home page of the firm. Consequently, postal addresses are not included in the list. If you have a specific need for particular information, we generally advise you to use appropriate search terms (keywords) and a powerful search engine to comb through the World Wide Web. This at least will ensure that you are working with a current cross-section of information.

Table 16.8 Summary of the descriptive categories used in the list of Web addresses

Category	Description
attacks	attacks on smart cards, smart card terminals, security modules etc.
card issuer	issuer of cards and/or smart cards
card manufacturer	manufacturer of cards with or without chips
card production machinery	machinery and equipment for producing cards
cryptography	cryptography related to smart cards
events	seminars, conferences and congresses related to smart cards
link farm	collection of links to other Internet sites
operating systems	operating systems for smart cards
patents	patents related to smart cards
publisher	journals and books related to smart cards
security technology	security technology related to smart cards
semiconductor manufacturer	manufacturer of semiconductors for smart cards, memory chips and/or microcontrollers
software	PC software for smart cards, smart card simulations
standards	standards related to smart cards and cryptography
terminal manufacturer	manufacturer of terminals for cards with or without chips
university	university or technical institute

[3GPP]	**3GPP** standards *http://www.3gpp.org/*
[3GPP2]	**3GPP2** standards *http://www.3gpp2.org/*
[AC]	**Austria Card, Austria** card manufacturer *http://www.austriacard.at/*
[ACG]	**ACG, Germany** chip merchant *http://www.acg.de/*
[ActivCard]	**ActivCard, USA** card manufacturer *http://www.activcard.com/*
[AltTech]	**alt.technology.smartcards FAQ** smart card FAQ site *http://www.scdk.com/atsfaq.htm*
[AM]	**American Magnetics, USA** terminal manufacturer *http://www.magstripe.com/*
[AmEx]	**American Express, USA** card issuer *http://www.americanexpress.com/*
[Anderson]	**Ross Anderson's Home Page, Great Britain** information about attacks on smart cards *http://www.cl.cam.ac.uk/users/rja14/*
[ANSI]	**ANSI, USA** standards *http://www.ansi.org/*
[ARM]	**ARM Ltd., Great Britain** processor core for smart card microcontrollers *http://www.arm.com/*
[ASM]	**ASM Lithography, The Netherlands** machinery for semiconductor manufacturing *http://www.asml.com/*
[Atmel]	**Atmel, USA** smart card microcontrollers *http://www.atmel.com/*

[Basiccard]	**Basic Card** smart card operating systems *http://www.basiccard.com/*
[BSI]	**Bundesamt for Sicherheit in der Informationstechnik (BSI), Germany** information about security *http://www.bsi.bund.de/*
[Card Forum]	**Card Forum, Germany** publisher in the smart card field *http://www.card-forum.com/*
[Cardshow]	**The Smart Card Cyber Show, France** http://www.cardshow.com/
[CC]	**Common Criteria** *http://www.commoncriteria.org/*
[CCC]	**Chaos Computer Club e.V., Germany** attacks on smart cards and cryptographic algorithms *http://www.ccc.de/*
[CDG]	**CDMA Development Group (CDG)** information about CDMA *http://www.cdg.org/*
[CEN]	**CEN** standards *http://www.cenorm.be/*
[CEPS]	**CEPSCO** information about CEPS *http://www.cepsco.com/*
[Certicom]	**Certicom Corp., Canada** cryptography, ECC *http://www.certicom.ca/*
[Counterpane]	**Counterpane, USA** cryptography *http://www.counterpane.com/*
[CR]	**Cryptography Research** attacks *http://www.cryptography.com/*
[CTST]	**CardTech/SecurTech Conference, USA** events relating to smart cards *http://www.ctst.com/*
[Dai Nippon]	**Dai Nippon Printing Co. Ltd., Japan** smart card manufacturer *http://www.dnp.co.jp/*

[Dallas Semi]	**Dallas Semiconductor, USA** semiconductor manufacturer; security processors *http://www.dalsemi.com/*
[Datacard]	**Datacard, USA** production machinery for smart cards *http://www.datacard.com/*
[De La Rue]	**De La Rue Card Systems, Great Britain** smart card manufacturer *http://www.delarue.com/*
[DIN]	**Deutsches Institut für Normung e.V. (DIN), Germany** standards *http://www.din.de/*
[DPA]	**Deutsches Patentamt, Germany** patents *http://www.deutsches-patentamt.de/*
[Drexler]	**Drexler Technology Corp., USA** cards with optically writeable and readable regions *http://www.lasercard.com/*
[ECBS]	**European Committee for Banking Standards** standards and specifications *http://www.ecbs.org/*
[ECC]	**The Error Correcting Codes (ECC) Home Page, Japan** link farm for error detection and correction codes *http://www.csl.sony.co.jp/person/morelos/ecc/codes.html*
[Emosyn]	**Emosyn** manufacturer of smart card microcontrollers *http://www.emosyn.com/*
[EMV]	**EMVCO** information about EMV *http://www.emvco.com/*
[Entrust]	**Entrust, Canada** cryptography *http://www.entrust.com/*
[ETSI]	**ETSI** standards *http://www.etsi.org/*
[Europay]	**Europay International, Belgium** card issuer *http://www.europay.com/*

[Eurosmart]	**Eurosmart** information about smart cards *http://www.eurosmart.com*
[FINEID]	**FINEID, Finland** information about FINEID *http://www.fineid.fi/*
[GD]	**Giesecke & Devrient GmbH, Germany** smart cards; operating systems; terminals *http://www.gieseckedevrient.com/* *http://www.gi-de.com/*
[Gemplus]	**Gemplus S.C.A., France** smart cards, operating systems, terminals *http://www.gemplus.com/*
[Global Platform]	**Global Platform** information about Global Platform *http://www.globalplatform.org/*
[Groupmark]	**Groupmark Ltd., Canada** smart card manufacturer *http://www.groupmark.com/*
[GSM]	**GSM MoU Association** link farm relating to GSM *http://www.gsmworld.com/*
[Gutmann]	**Peter Gutmann's Security and Encryption Links** *http://www.cs.auckland.ac.nz/~pgut001/*
[Hanser]	**Carl Hanser Verlag GmbH, Germany** pubisher (*Handbuch der Chipkarten*, *The Smart Card Simulator*) *http://www.hanser.de/*
[Hypercom]	**Hypercom Corp., USA** terminals *http://www.hypercom.com/*
[ICMA]	**ICMA – International Card Manufacturers Association** information about smart cards *http://www.icma.com/*
[IEC]	**IEC** standards *http://www.iec.ch/*

[IEEE]	**IEEE** standards *http://www.ieee.org/*
[Infineon]	**Infineon AG, Germany** semiconductor manufacturer *http://www.infineon.com*
[Ingenico]	**Ingenico, France** terminal manufacturer *http://www.ingenico.com/*
[Integri]	**Integri, Belgium** testing smart card operating systems *http://www.integri.com/*
[Iridium]	**Iridium, USA** information about the Iridium mobile telecommunications network *http://www.iridium.com/*
[ISO]	**ISO** standards *http://www.iso.ch/*
[ITU]	**ITU** standards *http://www.itu.ch/*
[JavaPOS]	**JavaPOS** Java for POS terminals *http://www.javapos.com/*
[Javasoft]	**Javasoft, USA** Java for smart cards *http://www.javasoft.com/*
[JCF]	**Java Card Forum, USA** Java, specifictions for Java in smart cards *http://www.javacardforum.org/*
[JTC1]	**ISO, Joint Technical Committee One** international standardization *http://www.jtc1.org/*
[Logika]	**Logika Comp Spa, Italy** personalization systems *http://www.logika.it/*
[MagTek]	**MagTek Inc., USA** terminals *http://www.magtek.com/*

[Maosco]	**Maosco Ltd., Great Britain** smart card operating system *http://www.multos.com/*
[MasterCard]	**MasterCard International, USA** card issuer *http://www.mastercard.com/*
[Microsoft]	**Microsoft, USA** Crypto-API, PC/SC *http://www.microsoft.com/*
[MIPS]	**MIPS** manufacturer of processors *http://www.mips.com/*
[MobM]	**Mobile Mind, USA** smart card company *http://www.mobile-mind.com*
[Mondex]	**Mondex International Ltd., Great Britain** electronic purse system *http://www.mondex.com/*
[Mühlbauer]	**Mühlbauer GmbH, Germany** card production machinery *http://www.muehlbauer.de/*
[MUSCLE]	**MUSCLE (Movement for the Use of Smart Cards in a Linux Environment)** MUSCLE project for linking smart cards to Linux systems *http:// www.linuxnet.com/*
[NIST]	**National Institute of Standards and Technology (NIST), USA** standards *http://www.nist.gov/*
[NSA]	**National Security Agency (NSA), USA** information about security and cryptography *http://www.nsa.gov/*
[Oberthur]	**Oberthur Smart Cards, USA** smart card manufacturer *http://www.oberthur.com/*
[OCF]	**OCF** OCF specification *http://www.opencard.org/*

[Oki]	**Oki, Japan** manufacturer of smart card microcontrollers and terminals *http://www.oki.com/* *http://www.oki.co.jp/*
[OMA]	**Open Mobile Alliance** successor to the WAP Forum *http://www.openmobilealliance.org*
[Omni]	**Omnikey** smart card terminals *http://www.omnikey.com*
[Orga]	**Orga GmbH, Germany** smart card manufacturer *http://www.orga.com/*
[PC/SC]	**PC/SC Working Group, USA** PC/SC specification *http://www.smartcardsys.com/*
[Philips]	**Philips, Germany** manufacturer of smart card microcontrollers *http://www.philips.com/* *http://www.semiconductors.philips.com/*
[Protechno]	**Protechno Card GmbH, Germany** manufacturer of desktop personalization machinery *http://www.protechno-card.com/*
[Proton]	**Proton** Proton electronic purse system *http://www.protonworld.com/*
[Radicchio]	**Radicchio** PKI *http:// www.radicchio.org/*
[Rankl]	**Home page of Wolfgang Rankl** errata lists for the *Handbuch der Chipkarten*, the *Smart Card Handbook* and the *Smart Card Simulator* (available as HTML documents) *http://www.wrankl.de*
[Renesas]	**Renesas Technology Corporation, Japan** smart card microcontrollers, terminals *http://www.renesas.com/*
[RFC]	**RFC Server** Internet standards; RFC http://www.rfc.net/
[RFID]	**RFID Handbook** information about RF ID *http://www.RFID-handbook.de*

[RSA]	**RSA Inc., USA** cryptography, PKCS specifications *http://www.rsa.com/*
[SCA]	**Smart Card Allliance** information about smart cards *http://www.smartcardallilance.org*
[SCARD]	**Smart Card Developer Association, USA** attacks, software *http://www.scard.org/*
[SCDK]	**Smart Card Developer's Kit** book *http://www.scdk.com/*
[Schlumberger]	**Schlumberger Ltd., France** smart card manufacturer *http://www.slb.com/*
[SET]	**Secure Electronic Transaction LLC, USA** SET home page *http://www.setco.org/*
[SETEC]	**SETEC, Finland** smart card manufacturer *http://www.setec.fi/*
[Siemens]	**Siemens, Germany** smart card operating system *http://www.siemens.com/*
[SIM Alliance]	**SIM Alliance** S@T browser *http://www.simalliance.org/*
[Smart Card Club]	**The Smart Card Club** *http://www.smartcardclub.co.uk/*
[Smarttrust]	**Smarttrust** microbrowser technology for smart cards *http://www.smarttrust.com/*
[SOSSE]	**Simple Operating System for Smartcard Education** smart card operating system *http://www.mbsks.franken.de/sosse*
[STM]	**ST Microelectronics, France** manufacturer of smart card microcontrollers *http://www.st.com/*

[Techno Data]	**Techno Data, Germany** magnetic-stripe cards, smart cards *http://www.technodata-ibk.com/*
[Teletrust]	**Teletrust, Germany** *http://www.teletrust.de/*
[TETRA]	**TETRA** information about TETRA *http://www.tetramou.com/*
[TI]	**Texas Instruments Inc., USA** semiconductor manufacturer *http://www.ti.com/*
[TIA]	**Telecommunications Industry Association** standards *http://www.tiaonline.org/*
[TNO]	**TNO (Netherlands Organization for Applied Research), The Netherlands** hardware testing of microcontrollers *http://www.tno.nl/*
[Topac]	**Topac GmbH** holograms *http://www.topac.de/*
[Ubiq]	**UbiQ Inc., USA** personalization *http://www.ubiqinc.com/*
[UCL]	**UCL Microelectronics Laboratory, Belgium** cryptography *http://www.dice.ucl.ac.be/*
[UCL-LL]	**UCL Microelectronics Laboratory – smart card link list** link farm *http://www.dice.ucl.ac.be/crypto/card.html*
[UMTS Forum]	**UMTS Forum** information about UMTS *http://www.umts-forum.org/*
[USB]	**USB** *http://www.usb.org/*
[Verifone]	**Verifone Inc., USA** terminal manufacturer *http://www.verifone.com/*

[Visa]	**Visa International, USA** card issuer *http://www.visa.com/* *http://www.visa.de/*
[WAP]	**WAP Forum** information about WAP *http://www.wapforum.org/*
[Wiley]	**John Wiley & Sons, Inc., Great Britain** publisher (*Smart Card Handbook)* *http://www.wiley.co.uk/*
[Zeitcontrol]	**Zeitcontrol Cardsystems GmbH, Germany** smart card manufacturer *http://www.zeitcontrol.de/*

16.10 CHARACTERISTIC DATA AND TABLES

16.10.1 ATR interval

Table 16.9 Time interval within which the ATR must be sent following a reset

Clock rate	Minimum time (400 clocks)	Maximum time (40,000 clocks)
1.0000 MHz	0.400 ms	40.000 ms
2.0000 MHz	0.200 ms	20.000 ms
3.0000 MHz	0.133 ms	13.333 ms
3.5712 MHz	0.112 ms	11.201 ms
4.0000 MHz	0.100 ms	10.000 ms
4.9152 MHz	0.081 ms	8.138 ms
5.0000 MHz	0.080 ms	8.000 ms
6.0000 MHz	0.067 ms	6.667 ms
7.0000 MHz	0.057 ms	5.714 ms
8.0000 MHz	0.050 ms	5.000 ms
9.0000 MHz	0.044 ms	4.444 ms
10.0000 MHz	0.040 ms	4.000 ms

16.10.2 ATR parameter conversion tables

The following tables are based on the definition of the ATR parameters CWT and BWT in the ISO/IEC 7816-3 standard. The indicated times are for a clock rate of 3.5712 MHz with various values of the clock rate conversion factor (divider) F.

Table 16.10 CWI/CWT conversion table (all times are based on a 3.5712-MHz clock with D = 1)

		F = 4	F = 8	F = 16	F = 31	F = 93
work etu:		1.120 µs	2.240 µs	4.480 µs	8.681 µs	26.042 µs
CWI	CWT (etu)	CWT (ms)				
0	12	0.013	0.027	0.054	0.104	0.313
1	13	0.015	0.029	0.058	0.113	0.339
2	15	0.017	0.034	0.067	0.130	0.391
3	19	0.021	0.043	0.085	0.165	0.495
4	27	0.030	0.060	0.121	0.234	0.703
5	43	0.048	0.096	0.193	0.373	1.120
6	75	0.084	0.168	0.336	0.651	1.953
7	139	0.156	0.311	0.623	1.207	3.620
8	267	0.299	0.598	1.196	2.318	6.953
9	523	0.586	1.172	2.343	4.540	13.620
10	1,035	1.159	2.319	4.637	8.984	26.953
11	2,059	2.306	4.612	9.225	17.873	53.620
12	4,107	4.600	9.200	18.401	35.651	106.953
13	8,203	9.188	18.376	36.752	71.207	213.620
14	16,395	18.364	36.727	73.454	142.318	426.953
15	32,779	36.715	73.430	146.859	284.540	853.620

Table 16.11 BWI/BWT conversion table (all values are based on a 3.5712-MHz clock with D = 1)

		F = 4	F = 8	F = 16	F = 31	F = 93
work etu:		1.120 µs	2.240 µs	4.480 µs	8.681 µs	26.042 µs
BWI	BWT (ms)	BWT (etu)				
0	100	89,291	44,651	22,331	11,531	3,851
1	200	178,645	89,323	44,661	23,051	7,684
2	400	357,131	178,571	89,291	46,091	15,371
3	800	714,251	357,131	178,571	92,171	30,731
4	1,600	1,428,491	714,251	357,131	184,331	61,451
5	3,200	2,856,971	1,428,491	714,251	368,651	122,891
6	6,400	5,714,005	2,857,003	1,428,501	737,291	245,764
7	12,800	11,427,925	5,713,963	2,856,981	1,474,571	491,524
8	25,600	22,855,765	11,427,883	5,713,941	2,949,131	983,044
9	51,200	45,711,445	22,855,723	11,427,861	5,898,251	1,966,084

16.10.3 Determining the data transmission rate

Table 16.12 Data transmission rate in bit/s for various clock frequencies in MHz with a clock rate conversion factor (F) of 372 and various values of the bit rate adjustment factor (D)

	D = 1	D = 2	D = 4	D = 8	D = 12	D = 16	D = 20	D = 32
F/D	372.00	186.00	93.00	46.50	31.00	23.25	18.60	11.63
Frequency								
1.0000	2,688	5,376	10,753	21,505	32,258	43,011	53,763	86,022
2.0000	5,376	10,753	21,505	43,011	64,516	86,022	107,527	172,043
3.0000	8,065	16,129	32,258	64,516	96,774	129,032	161,290	258,065
3.5712	9,600	19,200	38,400	76,800	115,200	153,600	192,000	307,200
4.0000	10,753	21,505	43,011	86,022	129,032	172,043	215,054	344,086
5.0000	13,441	26,882	53,763	107,527	161,290	215,054	268,817	430,108
6.0000	16,129	32,258	64,516	129,032	193,548	258,065	322,581	516,129
7.0000	18,817	37,634	75,269	150,538	225,806	301,075	376,344	602,151
8.0000	21,505	43,011	86,022	172,043	258,065	344,086	430,108	688,172
9.0000	24,194	48,387	96,774	193,548	290,323	387,097	483,871	774,194
10.0000	26,882	53,763	107,527	215,054	322,581	430,108	537,634	860,215

Table 16.13 Data transmission rate in bit/s for various clock frequencies in MHz with a clock rate conversion factor (F) of 512 and various values of the bit rate adjustment factor (D)

	D = 1	D = 2	D = 4	D = 8	D = 12	D = 16	D = 20	D = 32
F/D	512.00	256.00	128.00	64.00	42.67	32.00	25.60	16.00
Frequency								
1.0000	1,953	3,906	7,813	15,625	23,438	31,250	39,063	62,500
2.0000	3,906	7,813	15,625	31,250	46,875	62,500	78,125	125,000
3.0000	5,859	11,719	23,438	46,875	70,313	93,750	117,188	187,500
3.5712	6,975	13,950	27,900	55,800	83,700	111,600	139,500	223,200
4.0000	7,813	15,625	31,250	62,500	93,750	125,000	156,250	250,000
5.0000	9,766	19,531	39,063	78,125	117,188	156,250	195,313	312,500
6.0000	11,719	23,438	46,875	93,750	140,625	187,500	234,375	375,000
7.0000	13,672	27,344	54,688	109,375	164,063	218,750	273,438	437,500
8.0000	15,625	31,250	62,500	125,000	187,500	250,000	312,500	500,000
9.0000	17,578	35,156	70,313	140,625	210,938	281,250	351,563	562,500
10.0000	19,531	39,063	78,125	156,250	234,375	312,500	390,625	625,000

16.10.4 Sampling times for serial data

Table 16.14 is based on data transmission in compliance with the ISO/IEC 7816-3 standard. The indicated times have been calculated for a clock rate of 3.5712 MHz.

Table 16.14 Serial bit sampling times for data transmission with a divider value of 372

	Start	Lower limit	Midrange	Upper limit	End
Start bit	0 clocks 0.000 µs	112 clocks 31.250 µs	186 clocks 52.083 µs	260 clocks 72.917 µs	372 clocks 104.167 µs
Data bit 1/8	372 clocks 104.167 µs	484 clocks 135.417 µs	558 clocks 156.250 µs	632 clocks 177.083 µs	744 clocks 208.333 µs
Data bit 2/7	744 clocks 208.333 µs	856 clocks 239.583 µs	930 clocks 260.417 µs	1004 clocks 281.250 µs	1116 clocks 312.500 µs
Data bit 3/6	1116 clocks 312.500 µs	1228 clocks 343.750 µs	1302 clocks 364.583 µs	1376 clocks 385.417 µs	1488 clocks 416.667 µs
Data bit 4/5	1488 clocks 416.667 µs	1600 clocks 447.917 µs	1674 clocks 468.750 µs	1748 clocks 489.583 µs	1860 clocks 520.833 µs
Data bit 5/4	1860 clocks 520.833 µs	1972 clocks 552.083 µs	2046 clocks 572.917 µs	2120 clocks 593.750 µs	2232 clocks 625.000 µs
Data bit 6/3	2232 clocks 625.000 µs	2344 clocks 656.250 µs	2418 clocks 677.083 µs	2492 clocks 697.917 µs	2604 clocks 729.167 µs
Data bit 7/2	2604 clocks 729.167 µs	2716 clocks 760.417 µs	2790 clocks 781.250 µs	2864 clocks 802.083 µs	2976 clocks 833.333 µs
Data bit 8/1	2976 clocks 833.333 µs	3088 clocks 864.583 µs	3162 clocks 885.417 µs	3236 clocks 906.250 µs	3348 clocks 937.500 µs
Parity bit	3348 clocks 937.500 µs	3460 clocks 968.750 µs	3534 clocks 989.583 µs	3608 clocks 1010.417 µs	3720 clocks 1041.667 µs
Guard time/ Stop bit 1	3720 clocks 1041.667 µs	3832 clocks 1072.917 µs	3906 clocks 1093.750 µs	3980 clocks 1114.583 µs	4092 clocks 1145.833 µs
Guard time/ Stop bit 2	4092 clocks 1145.833 µs	4204 clocks 1177.083 µs	4278 clocks 1197.917 µs	4352 clocks 1218.750 µs	4464 clocks 1250.000 µs

16.10.5 The most important smart card commands

The following tables list the most important smart card commands with brief descriptions of their functions. These commands are taken from the following standards and specifications: ISO/IEC 7816-4, -7, -8, -9, EMV, GSM (GSM 11.11 & GSM 11.14), UICC (TS 31.111, TS 102.221, TS 102.222, TS 102.223), OP (Open Platform) and EN 1546.

Table 16.15 Summary of important standard smart card commands

Command	Function	INS	Standard
ACTIVATE FILE	Reversibly unblock a file.	'44'	ISO/IEC 7816-9
APPEND RECORD	Insert a new record in a file with a linear fixed structure.	'E2'	ISO/IEC 7816-4
APPLICATION BLOCK	Reversibly block an application.	'1E'	EMV
APPLICATION UNBLOCK	Unblock an application.	'18'	EMV

Table 16.16 (*Cont.*)

ASK RANDOM	Request a random number from the smart card.	'84'	EN 726-3
CHANGE CHV	Change the PIN.	'24'	GSM 11.11
CHANGE REFERENCE DATA	Change the data used for user identification (e.g., a PIN).	'24'	ISO/IEC 7816-8
CLOSE APPLICATION	Reset all attained access condition levels.	'AC'	EN 726-3
CONVERT IEP CURRENCY	Convert currency.	'56'	EN 1546-3
CREATE FILE	Create a new file.	'E0'	ISO/IEC 7816-9
CREATE RECORD	Create a new record in a record-oriented file.	'E2'	EN 726-3
CREDIT IEP	Load the purse (IEP).	'52'	EN 1546-3
CREDIT PSAM	Pay from IEP to the PSAM.	'72'	EN 1546-3
DEACTIVATE FILE	Reversibly block a file.	'04'	ISO/IEC 7816-9
DEBIT IEP	Pay from the purse.	'54'	EN 1546-3
DECREASE	Reduce the value of a counter in a file.	'30'	EN 726-3
DECREASE STAMPED	Reduce the value of a counter in a file that is protected using a cryptographic checksum.	'34'	EN 726-3
DELETE	Delete a uniquely identifiable object (such as a load file, application or key).	'E4'	OP
DELETE FILE	Delete a file.	'E4'	ISO/IEC 7816-9
DISABLE CHV	Disable PIN queries.	'26'	GSM 11.11 EN 726-3
DISABLE VERIFICATION REQUIREMENT	Disable user identification (e.g., PIN queries).	'26'	ISO/IEC 7816-8
ENABLE CHV	Enable PIN queries.	'28'	GSM 11.11 EN 726-3
ENABLE VERIFICATION REQUIREMENT	Enable user identification (e.g., PIN queries).	'28'	ISO/IEC 7816-8
ENVELOPE	Embed a second command in a smart card command.	'C2'	EN 726-3 ISO/IEC 7816-4
ERASE BINARY	Set the content of a file with a transparent structure to the erased state.	'0E'	ISO/IEC 7816-4
EXECUTE	Execute a file.	'AE'	EN 726-3
EXTEND	Extend a file.	'D4'	EN 726-3
EXTERNAL AUTHENTICATE	Authenticate the outside world with respect to the smart card.	'82'	ISO/IEC 7816-4
GENERATE AUTHORISATION CRYPTOGRAM	Generate a signature for a payment transaction.	'AE'	EMV-2
GENERATE PUBLIC KEY PAIR	Generate a key pair for an asymmetric cryptographic algorithm.	'46'	ISO/IEC 7816-8

Table 16.15 (*Cont.*)

GET CHALLENGE	Request a random number from the smart card.	'84'	ISO/IEC 7816-4
GET DATA	Read TLV-coded data objects.	'CA'	ISO/IEC 7816-4
GET PREVIOUS IEP SIGNATURE	Repeat the computation and output of the last signature received IEP.	'5A'	EN 1546-3
GET PREVIOUS PSAM SIGNATURE	Repeat the computation and output of the last signature received from the PSAM.	'86'	EN 1546-3
GET RESPONSE	Request data from the smart card (used with the T = 0 transmission protocol).	'C0'	GSM 11.11 ISO/IEC 7816-4
GET STATUS	Read the life-cycle state information of the card manager, application and load file.	'F2'	OP
GIVE RANDOM	Send a random number to the smart card.	'86'	EN 726-3
INCREASE	Increase the value of a counter in a file.	'32'	GSM 11.11 EN 726-3
INCREASE STAMPED	Increase the value of a counter in a file that is protected using a cryptographic checksum.	'36'	EN 726-3
INITIALIZE IEP	Initialize IEP for a subsequent purse command.	'50'	EN 1546-3
INITIALIZE PSAM	Initialize PSAM for a subsequent purse command.	'70'	EN 1546-3
INITIALIZE PSAM for Offline Collection	Initialize PSAM for offline booking of the amount.	'7C'	EN 1546-3
INITIALIZE PSAM for Online Collection	Initialize PSAM for online booking of the amount.	'76'	EN 1546-3
INITIALIZE PSAM for Update	Initialize PSAM for changing the parameters.	'80'	EN 1546-3
INSTALL	Install an application by invoking various oncard functions of the card manager and/or security domain.	'E6'	OP
INTERNAL AUTHENTICATE	Authenticate the smart card with respect to the outside world.	'88'	ISO/IEC 7816-4
INVALIDATE	Reversibly block a file.	'04'	GSM 11.11 EN 726-3
ISSUER AUTHENTICATE	Verify a signature of the card issuer.	'82'	EMV-2
LOAD	Load an application by transferring the load file.	'E8'	OP
LOAD KEY FILE	Load keys in files using cryptographic protection.	'D8'	EN 726-3
LOCK	Irreversibly block a file.	'76'	EN 726-3
MANAGE CHANNEL	Control the logical channels of a smart card.	'70'	ISO/IEC 7816-4
MANAGE SECURITY ENVIRONMENT	Change the parameters for using cryptographic algorithms in the smart card.	'22'	ISO/IEC 7816-8

Table 16.15 (*Cont.*)

MUTUAL AUTHENTICATE	Mutually authenticate the smart card and the terminal.	'82'	ISO/IEC 7816-8
PERFORM SCQL OPERATION	Execute an SCQL instruction.	'10'	ISO/IEC 7816-7
PERFORM SECURITY OPERATION	Execute a cryptographic algorithm in the smart card.	'2A'	ISO/IEC 7816-8
PERFORM TRANSACTION OPERATION	Execute an SCQL transaction instruction.	'12'	ISO/IEC 7816-7
PERFORM USER OPERATION	Manage users in the context of SCQL.	'14'	ISO/IEC 7816-7
PSAM COLLECT	Execute PSAM online booking of an amount.	'78'	EN 1546-3
PSAM COLLECT Acknowledgement	End PSAM online booking of an amount.	'7A'	EN 1546-3
PSAM COMPLETE	End paying the IEP against the PSAM.	'74'	EN 1546-3
PSAM VERIFY COLLECTION	End PSAM offline booking of an amount.	'7E'	EN 1546-3
PUT DATA	Write TLV-coded data objects.	'DA'	ISO/IEC 7816-4
PUT KEY	Write one or more new keys or replace existing keys.	'D8'	OP
REACTIVATE FILE	Unblock a file.	'44'	ISO/IEC 7816-9
READ BINARY	Read from a file with a transparent structure.	'B0'	GSM 11.11 ISO/IEC 7816-4
READ BINARY STAMPED	Read data from a file with a transparent structure that is secured with a cryptographic checksum.	'B4'	EN 726-3
READ RECORD / READ RECORD(S)	Read data from a file with a record-oriented structure.	'B2'	GSM 11.11 ISO/IEC 7816-4
READ RECORD STAMPED	Read data from a file with a record-oriented structure that is secured with a cryptographic checksum.	'B6'	EN 726-3
REHABILITATE	Unblock a file.	'44'	GSM 11.11 EN 726-3
RESET RETRY COUNTER	Reset an error counter.	'2C'	ISO/IEC 7816-8
RUN GSM ALGORITHM	Execute a GSM-specific cryptographic algorithm.	'88'	GSM 11.11
SEARCH BINARY	Search for a text string in a file with a transparent structure.	'A0'	ISO/IEC 7816-9
SEARCH RECORD	Search for a text string in a file with a record-oriented structure.	'A2'	ISO/IEC 7816-9
SEEK	Search for a text string in a file with a record-oriented structure.	'A2'	GSM 11.11 EN 726-3
SELECT/ SELECT (FILE)	Select a file.	'A4'	GSM 11.11 ISO/IEC 7816-4

Table 16.15 (*Cont.*)

SET STATUS	Write life-cycle state data for the card manager, application and load file.	'F0'	OP
SLEEP	Obsolete command for placing the smart card in a power-saving state.	'FA'	GSM 11.11
STATUS	Read various data from the currently selected file.	'F2'	GSM 11.11 EN 726-3
TERMINATE CARD USAGE	Irreversibly block a smart card.	'FE'	ISO/IEC 7816-9
TERMINATE DF	Irreversibly block a DF.	'E6'	ISO/IEC 7816-9
TERMINATE EF	Irreversibly block an EF.	'E8'	ISO/IEC 7816-9
UNBLOCK CHV	Reset a PIN retry counter that has reached its maximum value.	'2C'	GSM 11.11 EN 726-3
UPDATE BINARY	Write to a file with a transparent structure.	'D6'	GSM 11.11 EN 726-3 ISO/IEC 7816-4
UPDATE IEP PARAMETER	Change the general parameters of a purse.	'58'	EN 1546-3
UPDATE PSAM Parameter (offline)	Modify the parameters in the PSAM (offline).	'84'	EN 1546-3
UPDATE PSAM Parameter (online)	Modify the parameters in the PSAM (online).	'82'	EN 1546-3
UPDATE RECORD	Write to a file with a linear fixed, linear variable or cyclic structure.	'DC'	GSM 11.11 ISO/IEC 7816-4
VERIFY	Verify the transferred data (such as a PIN).	'20'	ISO/IEC 7816-4 EMV-2
VERIFY CHV	Verify the PIN.	'20'	GSM 11.11 EN 726-3
WRITE BINARY	Write to a file with a transparent structure using a logical AND/OR process.	'D0'	EN 726-3 ISO/IEC 7816-4
WRITE RECORD	Write to a file with a record-oriented structure using a logical AND/OR process.	'D2'	EN 726-3 ISO/IEC 7816-4

16.10.6 Summary of utilized instruction bytes

Tables 16.16 through 16.18 identify the INS codes that are used with various class bytes. The odd-numbered codes in the shaded columns cannot be used to encode commands, since the T = 0 transfer protocol uses these codes to control the programming voltage.[75]

[75] See also Section 6.4.2,'The T = 0 transmission protocol'. There is a proposal to revise ISO/IEC 7816-3 to eliminate the possibility of controlling an external programming voltage via the instruction byte in the future

Table 16.16 Summary of the INS byte codes used with a class byte (CLA) of '00' as specified in the ISO/IEC 7816-4, -7, -8 and -9 standards. The suffix number of the corresponding standard is shown for each code that is used

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0y					9										4	
1y	7		7		7											
2y	4		8		8		8		8		8		8			
3y																
4y	9				9		8									
5y																
6y																
7y	4															
8y			4.8		4				4							
9y																
Ay	9		9		4											
By	4		4													
Cy	4		4								4					
Dy	4		4				4				4		4			
Ey	9		4		9		9		9							
Fy															9	

Table 16.17 Summary of the INS byte codes used with a class byte (CLA) of '80' as specified in the EMV, EN 1546 and OP standards. The initial letters of the corresponding standard are shown for each code that is used

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0y																
1y							OP		OP						OP	
2y					OP											
3y																
4y																
5y	EN		EN		EN		EN				EN					
6y																
7y	EN		EN		EN		EN		EN		EN		EN		EN	
8y	EN		EN, EMV		EN		EN									
9y																
Ay									OP						OP, EMV	
By																
Cy											OP					
Dy									OP							
Ey					OP		OP									
Fy	OP		OP													

Table 16.18 Summary of the INS byte codes used with a class byte (CLA) of '80' as specified in the GMS 11.11 standard

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0y					X											
1y	X		X		X											
2y	X				X		X		X				X			
3y			X													
4y					X											
5y																
6y																
7y																
8y									X							
9y																
Ay			X		X											
By	X		X													
Cy	X		X													
Dy							X		X							
Ey																
Fy			X								X					

16.10.7 Smart card command coding

Tables 16.19 through 16.24 show the most important codes for some sample smart card commands. For the sake of clarity, it is assumed that neither secure messaging nor logical channel addressing is used. Refer to the ISO/IEC 7816-4 standard for the complete coding of these and other smart card commands.[76]

Table 16.19 Coding of the Case 4 command SELECT FILE, with the principal options

Data element	Code	Remark	
CLA	'00'	Class byte reserved for ISO/IEC 7816 commands without secure messaging.	
INS	'A4'	Instruction byte for SELECT FILE, which is the command for selecting a file (MF, DF or EF).	
P1	...	P1 = '00' $\wedge$ Lc = 0	Select the MF
		P1 = '00' $\wedge$ Lc $\neq$ 0	Select a file using its FID (FID in DATA)
		P1 = '04'	Select a file using its DF name (DF name in DATA)
		P1 = '08'	Select a file by specifying a FID-based path from the MF (path in DATA)
		P1 = '09'	Select a file by specifying a FID-based path from the currently selected DF (path in DATA)
P2	...	P2 = '00'	Return optional FCI
		P2 = '04'	Return optional FCP
		P2 = '08'	Return optional FMD
Lc	...	Coding described under 'P1'	
DATA	...	Coding described under 'P1'	
Le	Le = 0	Return all data belonging to the selected item.	

[76] See also Section 6.5.1, 'Structure of the command APDU'

Table 16.20 Coding of the Case 2 command READ BINARY as specified by ISO/IEC 7816-4, with the principal options

Data element	Code	Remark	
CLA	'00'	Class byte reserved for ISO/IEC 7816 commands without secure messaging.	
INS	'B0'	Instruction byte for READ BINARY, which is the command for reading data from a file with a transparent structure.	
P1	...	P1.b8 = 0	Read data from the currently selected file using an offset. Offset = (P1.b7 ... P1.b1 || P2)
		P1.b8 = 1	After implicit file selection using a short FID, read data using an offset. Short FID = (P1.b5 ... P1.b1), offset = P2
P2	...	Coding described under ‘P1’	
Le	...	Le = 0:	Read all data until the end of the file.
		Le > 0:	Le is the number of bytes to be read.

Table 16.21 Coding of the Case 3 command UPDATE BINARY as specified by ISO/IEC 7816-4, with the principal options

Data element	Code	Remark	
CLA	'00'	Class byte reserved for ISO/IEC 7816 commands without secure messaging.	
INS	'D6'	Instruction byte for UPDATE BINARY, which is the command for writing data to a file with a transparent structure.	
P1	...	P1.b8 = 0	Write data to the currently selected file using an offset. Offset = (P1.b7 ... P1.b1 || P2).
		P1.b8 = 1	After implicit file selection using a short FID, write data using an offset. Short FID = (P1.b5 ... P1.b1), offset = P2.
P2	...	Coding described under ‘P1’	
Lc	...	Lc is the number of bytes to be written.	
DATA	...	The bytes to be written, with a length of Lc.	

Table 16.22 Coding of the Case 2 command READ RECORD as specified by ISO/IEC 7816-4, with the principal options

Data element	Code	Remark	
CLA	'00'	Class byte reserved for ISO/IEC 7816 commands without secure messaging.	
INS	'B2'	Instruction byte for READ RECORD, which is the command for reading data from a file with a record-oriented structure.	
P1	...	P1 = 0	Read the current record.
		P1 ≠ 1	Read the record number having the record number or record identifier given in P1.

Table 16.22 (*Cont.*)

P2	...	P2.b8 ... P2.b4 = °00000°	Read data from the currently selected file.
		P2.b8 ... P2.b4 ≠ °00000°	Read data after implicit file selection using a short FID. Short FID = (P1.b8 ... P1.b4)
		P2.b3 ... P2.b1 = °000°	Read the first record, using the record identifier passed via P1.
		P2.b3 ... P2.b1 = °001°	Read the last record, using the record identifier passed via P1.
		P2.b3 ... P2.b1 = °010°	Read the next record, using the record identifier passed via P1.
		P2.b3 ... P2.b1 = °011°	Read the previous record, using the record identifier passed via P1.
		P2.b3 ... P2.b1 = °100° ∧ P1 = 0	Read the current record.
		P2.b3 ... P2.b1 = °100° ∧ P1 ≠ 0	Read the record having the record number given in P1.
		P2.b3 ... P2.b1 = °101°	Read all records from the record number given in P1 until the end of the file.
		P2.b3 ... P2.b1 = °110°	Read all records from the end of the file back to the record number given in P1.
Le	...	Le = 0:	Read all bytes until the end of the record(s).
		Le > 0:	Le is the length of the record(s).

Table 16.23 Coding of the Case 3 command UPDATE RECORD as specified by ISO/IEC 7816-4, with the principal options

Data element	Code	Remark	
CLA	'00'	Class byte reserved for ISO/IEC 7816 commands without secure messaging.	
INS	'DC'	Instruction byte for UPDATE RECORD, which is the command for writing data to a file with a record-oriented structure.	
P1	...	P1 = 0	Write the current record.
		P1 ≠ 0	Write the record having the record number given in P1.
P2	...	P2.b8 ... P2.b4 = °00000°	Write data to the currently selected file.
		P2.b8 ... P2.b4 ≠ °00000°	Write data following implicit file selection using a short FID. Short FID = (P1.b8 ... P1.b4).
		P2.b3 ... P2.b1 = °000°	Write the first record.
		P2.b3 ... P2.b1 = °001°	Write the last record.
		P2.b3 ... P2.b1 = °010°	Write the next record.
		P2.b3 ... P2.b1 = °011°	Write the previous record.
		P2.b3 ... P2.b1 = °100°	Write the record having the record number given in P1.
Lc	...	Lc is the length of the record to be written.	
DATA	...	The record to be written.	

Table 16.24 Coding of the Case 3 command VERIFY as specified by ISO/IEC 7816-4, with the principal options

Data element	Code	Remark	
CLA	'00'	Class byte reserved for ISO/IEC 7816 commands without secure messaging.	
INS	'20'	Instruction byte for VERIFY, which is the command for comparing transferred data to reference data (typically a PIN).	
P1	'00'	—	
P2	...	P2 = '00'	No explicit data is transferred.
		P2.b8 = °0°	Reference data valid for the entire smart card (global reference data) is used.
		P2.b8 = °1°	Reference data valid for one or more specific applications (local reference data) is used.
		P2.b7 || P2.b6 =°00°	RFU bits.
		P2.b5 ... P2.b1	Reference data identification number.
Lc	...	Lc is the length of the transferred comparison value.	
DATA	...	The transferred comparison value (usually a PIN).	

16.10.8 Smart card return codes

The return codes described in Table 16.25 are classified according to the scheme used in the ISO/IEC 7816-4 standard.[77] The following status codes are used:

NP: process completed, normal processing	EE: process aborted, execution error
WP: process completed, warning processing	CE: process aborted, checking error

Table 16.25 Selected standard smart card return codes as specified by ISO/IEC 7816-4

Return code	Status	Meaning	Standard
'61xx'	NP	Command successfully executed; 'xx' bytes of data are available and can be requested using GET RESPONSE.	ISO/IEC 7816-4
'6281'	WP	The returned data may be erroneous.	ISO/IEC 7816-4
'6282'	WP	Fewer bytes than specified by the Le parameter could be read, since the end of the file was encountered first.	ISO/IEC 7816-4
'6283'	WP	The selected file is reversibly blocked (invalidated).	ISO/IEC 7816-4
'6284'	WP	The file control information (FCI) is not structured in accordance with ISO/IEC 7816-4.	ISO/IEC 7816-4
'62xx'	WP	Warning; state of non-volatile memory not changed.	ISO/IEC 7816-4

[77] See also Section 6.5.2, 'Structure of the response APDU'

Table 16.25 (*Cont.*)

'63Cx'	WP	The counter has reached the value 'x' ($0 \leq x \leq 15$) (the exact significance depends on the command).	ISO/IEC 7816-4
'63xx'	WP	Warning; state of non-volatile memory changed.	ISO/IEC 7816-4
'64xx'	EE	Execution error; state of non-volatile memory not changed.	ISO/IEC 7816-4
'6581'	EE	Memory error (e.g. during a write operation).	ISO/IEC 7816-4
'65xx'	EE	Execution error; state of non-volatile memory changed.	ISO/IEC 7816-4
'6700'	CE	Length incorrect.	GSM 11.11 ISO/IEC 7816-4
'67xx'... '6Fxx'	CE	Check errors.	ISO/IEC 7816-4
'6800'	CE	Functions in the class byte not supported (general).	ISO/IEC 7816-4
'6881'	CE	Logical channels not supported.	ISO/IEC 7816-4
'6882'	CE	Secure messaging not supported.	ISO/IEC 7816-4
'6900'	CE	Command not allowed (general)	ISO/IEC 7816-4
'6981'	CE	Command incompatible with file structure.	ISO/IEC 7816-4
'6982'	CE	Security state not satisfied.	ISO/IEC 7816-4
'6983'	CE	Authentication method blocked.	ISO/IEC 7816-4
'6984'	CE	Referenced data reversibly blocked (invalidated).	ISO/IEC 7816-4
'6985'	CE	Usage conditions not satisfied.	ISO/IEC 7816-4
'6986'	CE	Command not allowed (no EF selected).	ISO/IEC 7816-4
'6987'	CE	Expected secure messaging data objects missing.	ISO/IEC 7816-4
'6988'	CE	Secure messaging data objects incorrect.	ISO/IEC 7816-4
'6A00'	CE	Incorrect P1 or P2 parameters (general).	ISO/IEC 7816-4
'6A80'	CE	Parameters in the data portion are incorrect.	ISO/IEC 7816-4
'6A81'	CE	Function not supported.	ISO/IEC 7816-4
'6A82'	CE	File not found.	ISO/IEC 7816-4
'6A83'	CE	Record not found.	ISO/IEC 7816-4
'6A84'	CE	Insufficient memory.	ISO/IEC 7816-4
'6A85'	CE	Lc inconsistent with TLV structure	ISO/IEC 7816-4
'6A86'	CE	Incorrect P1or P2 parameter.	ISO/IEC 7816-4
'6A87'	CE	Lc inconsistent with P1 or P2.	ISO/IEC 7816-4
'6A88'	CE	Referenced data not found.	ISO/IEC 7816-4
'6B00'	CE	Parameter 1 or 2 incorrect.	GSM 11.11 ISO/IEC 7816-4
'6Cxx'	CE	Bad length value in Le; 'xx' is the correct length.	ISO/IEC 7816-4
'6D00'	CE	Command (instruction) not supported.	GSM 11.11 ISO/IEC 7816-4
'6E00'	CE	Class not supported.	GSM 11.11 ISO/IEC 7816-4

(*Cont.*)

Table 16.25 (*Cont.*)

'6F00'	CE	Command aborted – more exact diagnosis not possible (e.g., operating system error).	GSM 11.11 ISO/IEC 7816-4
'9000'	NP	Command successfully executed.	GSM 11.11 ISO/IEC 7816-4
'920x'	NP	Writing to EEPROM successful after ‘x’ attempts.	GSM 11.11
'9210'	CE	Insufficient memory.	GSM 11.11
'9240'	EE	Writing to EEPROM not successful.	GSM 11.11
'9400'	CE	No EF selected.	GSM 11.11
'9402'	CE	Address range exceeded.	GSM 11.11
'9404'	CE	FID not found, record not found or comparison pattern not found.	GSM 11.11
'9408'	CE	Selected file type does not match command.	GSM 11.11
'9802'	CE	No PIN defined.	GSM 11.11
'9804'	CE	Access conditions not satisfied, authentication failed.	GSM 11.11
'9835'	CE	ASK RANDOM or GIVE RANDOM not executed.	GSM 11.11
'9840'	CE	PIN verification not successful.	GSM 11.11
'9850'	CE	INCREASE or DECREASE could not be executed because a limit has been reached.	GSM 11.11
'9Fxx'	NP	Command successfully executed; ‘xx’ bytes of data are available and can be requested using GET RESPONSE.	GSM 11.11

16.10.9 Selected chips for memory cards

Table 16.26 lists a selection of various types of typical memory chips for smart cards, which makes no claim to being complete or entirely correct. The primary purpose of this table is to give a general idea of the very wide selection of available memory chips. Here we would like to explicitly state that tables of this sort quickly become outdated, due to ongoing technical progress. Current general technical specifications are best obtained directly from the web servers of the various manufacturers, such as Infineon [Infineon], Philips [Philips] and ST Microelectronics [STM]. Similar components are also available from a variety of other manufacturers.

Table 16.26 Summary of selected memory chips for smart cards

Manufacturer/type	Memory capacity		Additional information	
Infineon SLE 4404	ROM: PROM: EEPROM:	16 bits 144 bits 256 bits	Vcc: Icc: W/E cycles: W/E time: HW:	5 V 3 mA 100,000 5 ms —

Table 16.26 (*Cont.*)

Infineon	ROM:	24 bits	Vcc:	5 V
SLE 4406S	PROM:	72 bits	Icc:	1 mA
	EEPROM:	32 bits	W/E cycles:	100,000
			W/E time:	5 ms
			HW:	counter for ≈20,000 units
Infineon	ROM:	—	Vcc:	5 V
SLE 44R35	PROM:	—	Icc:	3 mA
	EEPROM:	1 KB	W/E cycles:	100,000
			W/E time:	2 ms
			HW:	PIN logic for extra write protection, unilateral authentication; for contactless memory cards
Infineon	ROM:	—	Vcc:	5 V
SLE 4442	PROM:	32 bits	Icc:	10 mA
	EEPROM:	256 bytes	W/E cycles:	100,000
			W/E time:	2.5 ms
			HW:	—
Infineon	ROM:	24 bits	Vcc:	5 V
SLE 5536	PROM:	177 bits	Icc:	2.5 mA
	EEPROM:	36 bits	W/E cycles:	100,000
			W/E time:	3 ms
			HW:	counter for ≈ 20,000 units, unilateral authentication
Infineon	ROM:	24 bits	Vcc:	3–5 V
SLE 7736	PROM:	177 bits	Icc: 1 mA	
	EEPROM:	36 bits	W/E cycles:	100,000
			W/E time:	3 ms
			HW:	counter for ≈20,000 units
Philips	ROM:	—	Vcc:	☹
MF1 S70	PROM:	—	Icc:	☹
	EEPROM:	4 KB	W/E cycles:	100,000
			W/E time:	☹
			HW:	4-byte serial number, unilateral authentication, ISO/IEC 15 443A contactless I/F

(*Cont.*)

Table 16.26 (*Cont.*)

Philips	ROM:	—	Vcc:	☹
I-Code	PROM:	—	Icc:	☹
SLI ICPCB 7960	EEPROM:	896 bits	W/E cycles:	☹
			W/E time:	☹
			HW:	8-byte serial number, unilateral authentication, ISO/IEC 15 693 contactless I/F
ST Microelectronics	ROM:	—	W/E cycles:	100,000
LR 1512	PROM:	—	W/E time:	5 ms
	EEPROM:	512 bits	HW:	ISO/IEC 15 693
ST Microelectronics	ROM:	—	W/E cycles:	100,000
M35101	PROM:	—	W/E time:	5 ms
	EEPROM:	2048 bits	HW:	ISO/IEC 14 443B
ST Microelectronics	ROM:	16 bits	Vcc:	5 V
ST1335D	PROM:	—	Icc:	500 μA
	EEPROM:	272 bits	W/E cycles:	500,000
			W/E time:	3.5 ms
			HW:	counter for 32,767 units, unilateral authentication

16.10.10 Selected microcontrollers for smart cards

Table 16.27 lists a selection of various types of typical microcontrollers for smart cards, which makes no claim to being complete or entirely correct. The primary purpose of this table is to give a general idea of the wide selection of available smart card microcontrollers. Here we would like to explicitly state that tables of this sort quickly become outdated, due to ongoing technical progress. Current general technical specifications are best obtained directly from the web servers of the various manufacturers, such as Atmel [Atmel], Renesas [Renesas], Infineon [Infineon], Philips [Philips] and ST Microelectronics [STM]. Similar components are also available from a variety of other manufacturers.

The following abbreviations are used in the table:

Vcc:	Supply voltage range
Clock:	Clock frequency range
Icc:	Current consumption of the chip at the stated clock frequency (first value: operating; second value: low-power state with clock; third value: low-power state without clock)
Size (optional):	Die size
Structure:	Minimum structure width on the chip
Page:	EEPROM page size

W/E cyc.:	Guaranteed number of write/erase cycles per EEPROM page
W/E time:	Cycle time for writing or erasing one EEPROM page
HW:	Additional on-chip hardware
Timer:	Timer for counting clock cycles
UART:	Universal asynchronous receiver/transmitter (hardware-based data transmission)
CRC:	CRC processing unit
PLL:	Internal clock multiplier (phase-locked loop)
MMU:	Memory management unit
RNG:	Random number generator
DES:	DES accelerator (generally includes triple-DES accelerator)
AES:	AES accelerator
RSA:	RSA accelerator (generally includes EC accelerator)
☹	No publicly available information for this item

Table 16.27 Summary of selected microcontrollers for smart cards

Manufacturer and type	Memory capacity		Additional information	
Atmel AT90 SC6464C	Flash:	64 kB	CPU:	AVR
	EEPROM:	64 kB	Vcc:	2.7–3.3 V, 4.5–5.5 V
	RAM:	3 kB	Clock:	1–10 MHz
			Icc:	☹, ☹
			Size:	24 mm^2
			Structure:	0.35 µm
			EEPROM	
			Page:	1–128 bytes
			W/E cyc.:	500,000
			W/E time:	5 ms
			Flash EEPROM	
			Page:	128 bytes
			W/E cyc.:	500,000
			W/E time:	5 ms
			HW:	RISC CPU, two 16-bit timers, MMU, RNG, DES, RSA
Atmel AT90 SC19264RC	Flash:	192 kB	CPU:	AVR
	EEPROM:	64 kB	Vcc:	2.7–3.3 V, 4.5–5.5 V
	RAM:	6 kB	Clock:	1–16 MHz
			Icc:	☹, ☹
			Size:	24 mm^2
			Structure:	0.35 µm
			EEPROM	
			Page:	1–128 bytes
			W/E cyc.:	500,000
			W/E time:	5 ms
			HW:	RISC CPU, two 16-bit timers, UART, CRC, PLL, MMU, RNG, DES, RSA

(*Cont.*)

Table 16.27 (*Cont.*)

Infineon SLE 66C160P	ROM:	64 kB	CPU:	8051 derivative
	EEPROM:	16 kB	Vcc:	2.7–5.5 V
	RAM:	2.3 kB	Clock:	1–10 MHz
			Icc:	☹, ☹
			Structure:	0.25 µm
			EEPROM	
			Page:	1–64 bytes
			W/E cyc.:	500,000
			W/E time:	4.5 ms
			HW:	Two 16-bit timers, UART, CRC, PLL, MMU, RNG, DES
Infineon SLE 66C640P	ROM:	136 kB	CPU:	8051 derivative
	EEPROM:	64 kB	Vcc:	2.7–5.5 V
	RAM:	4.3 kB	Clock:	1–10 MHz
			Icc:	$\leq$ 10 mA, ☹
			Structure:	0.22 µm
			EEPROM	
			Page:	1–64 bytes
			W/E cyc.:	500,000
			W/E time:	4.5 ms
			HW:	Two 16-bit timers, UART, CRC, PLL, MMU, RNG
Infineon SLE 66CX642P	ROM:	200 kB	CPU:	8051 derivative
	EEPROM:	64 kB	Vcc:	1.62–5.5 V
	RAM:	4.3 kB	Clock:	1–15 MHz
			Icc:	$\leq$ 10 mA, ☹
			Structure:	0.22 µm
			EEPROM	
			Page:	1–64 bytes
			W/E cyc.:	500,000
			W/E time:	4.5 ms
			HW:	Two 16-bit timers, UART, CRC, PLL, MMU, RNG, DES, RSA
Infineon SLE 88CX720P	ROM:	240 kB	CPU:	32-bit Infineon 88
	EEPROM:	80 kB	Vcc:	1.62–5.5 V
	RAM:	8 kB	Clock:	1–15 MHz
			Icc:	$\leq$ 30 mA, ☹
			Structure:	0.22 µm
			EEPROM	
			Page:	1–64 bytes
			W/E cyc.:	500,000
			W/E time:	4.5 ms
			HW:	Two 16-bit timers, UART, CRC, PLL ($\leq$ 55 MHz), MMU, RNG, DES, RSA

Table 16.27 (*Cont.*)

Philips	ROM:	32 kB	CPU:	8051
P8WE6004	EEPROM:	4 kB	Vcc:	2.7–5.5 V
	RAM:	768 bytes	Clock:	1–8 MHz
			Icc:	☹, ☹
			Structure:	0.35 µm
			EEPROM	
			Page:	1–64 bytes
			W/E cyc.:	100,000
			W/E time:	2 ms/2 ms
			HW:	Two 16-bit timers, UART, RNG, DES
Philips	ROM:	32 kB	CPU:	8051
P8RF6004	EEPROM:	4 kB	Vcc:	2.7–5.5 V
	RAM:	1280 bytes	Clock:	1–8 MHz; 13.56 MHz (for RF)
			Icc:	☹, ☹
			Structure:	0.35 µm
			EEPROM	
			Page:	1–64 bytes
			W/E cyc.:	100,000
			W/E time:	2 ms/2 ms
			HW:	Two 16-bit timers, UART for ISO/IEC 7816 and ISO/IEC 14 433A, MMU, RNG, DES
Philips	ROM:	64 kB	CPU:	8051
P8RF5016	EEPROM:	16 kB	Vcc:	2.7–5.5 V
	RAM:	2300 bytes	Clock:	1–8 MHz; 13.56 MHz (for RF)
			Icc:	☹, ☹
			Structure:	0.35 µm
			EEPROM	
			Page:	1–64 bytes
			W/E cyc.:	100,000
			W/E time:	2 ms/2 ms
			HW:	Two 16-bit timers, UART for ISO/IEC 7816 and ISO/IEC 14 433A, MMU, RNG, DES, RSA
Philips	ROM:	208 kB	CPU:	XA2
P16WX064	EEPROM:	64 kB	Vcc:	2.7–5.5 V
	RAM:	7 kB	Clock:	1–6 MHz
	Flash (opt.):	32 kB	Icc:	☹, ☹
			Structure:	0.18 µm
			EEPROM	
			Page:	1–64 bytes
			W/E cyc.:	100,000
			W/E time:	2 ms/2 ms
			HW:	Two 16-bit timers, UART, CRC, MMU, RNG, DES, RSA

(*Cont.*)

Table 16.27 (*Cont.*)

Philips	ROM:	320 kB	CPU:	MIPS
P9CC160	EEPROM:	64 kB	Vcc:	1.62–5.5 V
	RAM:	7 kB	Clock:	1–6 MHz
	Flash:	96 kB	Icc:	☹, ☹
			Structure:	0.18 µm
			EEPROM	
			Page:	1–64 bytes
			W/E cyc.:	100,000
			W/E time:	2 ms/2 ms
			HW:	Two 16-bit timers, UART, MMU, RNG, DES, AES, RSA
Renesas	ROM:	32 kB	CPU:	H8
H8/3166	EEPROM:	2.2 kB	Vcc:	2.7–5.5 V
	RAM:	1 kB	Clock:	1–10 MHz
			Icc:	$\leq$ 10 mA, $\leq$ 100 µA
			Structure:	0.35 µm
			EEPROM	
			Page:	32 bytes
			W/E cyc.:	100,000
			W/E time:	3 ms/1.5 ms
			HW:	RNG
Renesas	ROM:	48 kB	CPU:	AE-3
AE350	EEPROM:	32.5 kB	Vcc:	2.7–5.5 V
	RAM:	1280 bytes	Clock:	1–10 MHz
			Icc:	$\leq$ 10 mA, $\leq$ 100 µA
			Structure:	0.35 µm
			EEPROM	
			Page:	64 bytes
			W/E cyc.:	100,000
			W/E time:	3 ms/1.5 ms
			HW:	RNG
Renesas	ROM:	128 kB	CPU:	AE-4
AE45X-B/C	EEPROM:	36 kB	Vcc:	2.7–5.5 V
	RAM:	4.5 kB	Clock:	1–10 MHz
			Icc:	$\leq$ 10 mA, $\leq$ 100 µA
			Structure:	0.35 µm
			EEPROM	
			Page:	64 bytes
			W/E cyc.:	500,000
			W/E time:	3 ms/1.5 ms
			HW:	Two 16-bit timers, UART for ISO/IEC 7816 and ISO/IEC 14 433, PLL, MMU, RNG, DES, RSA

Table 16.27 (*Cont.*)

Renesas AE460	ROM:	96 kB	CPU:	AE-4
	EEPROM:	64.5 kB	Vcc:	2.7–5.5 V
	RAM:	3 kB	Clock:	1–10 MHz
			Icc:	$\leq$ 10 mA,$\leq$ 100 µA
			Structure:	0.35 µm
			EEPROM	
			Page:	128 bytes
			W/E cyc.:	100,000
			W/E time:	3 ms/1.5 ms
			HW:	MMU, RNG, DES
Renesas AE46C	ROM:	196 kB	CPU:	AE-4
	EEPROM:	68 kB	Vcc:	2.7–5.5 V
	RAM:	6.5 kB	Clock:	1–10 MHz
			Icc:	$\leq$ 10 mA, $\leq$ 100 µA
			Structure:	0.35 µm
			EEPROM	
			Page:	128 bytes
			W/E cyc.:	500,000
			W/E time:	3 ms/1.5 ms
			HW:	Two 16-bit timers, UART, PLL, MMU, RNG, DES, RSA
ST Micro ST16SF4x	ROM:	16 kB	CPU:	6805
	EEPROM:		Vcc:	2.7–5.5 V
	1.25/2/4/8/16	kB	Clock:	1–5 MHz
	RAM:	384 bytes	Icc:	☹, ☹
			Structure:	0.7 µm
			EEPROM	
			Page:	1–32 bytes
			W/E cyc.:	300,000
			W/E time:	2.5 ms
			HW:	MMU, RNG
ST Micro ST19RF08	ROM:	32 kB	CPU:	ST7
	EEPROM:	8 kB	Vcc:	2.7–5.5 V
	RAM:	960 bytes	Clock:	1–10 MHz
			Icc:	☹, ☹
			Structure:	0.6 µm
			EEPROM	
			Page:	1–64 bytes
			W/E cyc.:	100,000
			W/E time:	1 ms
			HW:	One 16-bit timer, UART for ISO/IEC 7816 and ISO/IEC 14 433B, CRC, MMU, RNG, DES

(*Cont.*)

Table 16.27 (*Cont.*)

ST Micro	ROM:	176 kB	CPU:	ST7
ST19WG34	EEPROM:	34 kB	Vcc:	1.6–5.5 V
	RAM:	6 kB	Clock:	1–10 MHz
			Icc:	☹, ☹
			Structure:	0.18 µm
			EEPROM	
			Page:	1–64 bytes
			W/E cyc.:	500,000
			W/E time:	2 ms
			HW:	Three 16-bit timers, CRC, MMU, RNG, DES
ST Micro	ROM:	224 kB	CPU: ST22	
ST22WJ64	EEPROM:	64 kB	Vcc:	2.7–5.5 V
	RAM:	8 kB	Clock:	1–30 MHz
			Icc:	☹, ☹
			Structure:	0.18 µm
			EEPROM	
			Page:	1–128 bytes
			W/E cyc.:	500,000
			W/E time:	4.5 ms
			HW:	Two 16-bit timers, UART, PLL ($\leq$ 30 MHz), MMU, RNG, DES

Index