APPENDIX B (Part 2 of 2)

CHAPTER 7

Network and Transport Security Protocols

Applications, systems, and networks can be made secure through the use of security protocols, *which provide a wide range of encryption and authentication services. Each protocol is placed within several layers of a computing infrastructure (that is, network, transport, and application layers). Figure 7-1 shows the various protocols and their locations within the* Transportation Control Protocol/Internet Protocol *(TCP/IP) stack. This chapter and Chapter 8 describe these protocols and explain how they operate within the TCP/IP stack. This chapter first covers the IPSec protocol, which provides security at the network layer. Then we take an in-depth look at the* Secure Sockets Layer *(SSL), which implements security at the transport layer.*

Internet Protocol Security

Internet Protocol Security (IPSec) is a framework of open standards for ensuring secure private communications over IP networks. Based on standards developed by the *Internet Engineering Task Force* (IETF), IPSec ensures confidentiality, integrity, and authenticity of data communications across a public IP network. IPSec is a necessary component of a

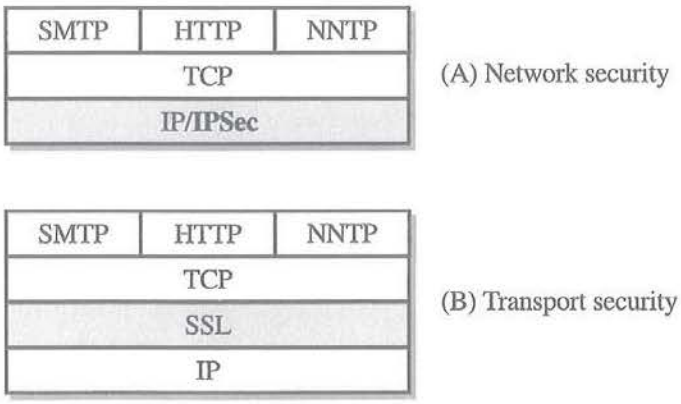


Figure 7-1
Protocol locations within TCP/IP: (a) network security and (b) transport security

standards-based, flexible solution for deploying a network-wide security policy.

IPSec implements network layer encryption and authentication, providing an end-to-end security solution in the network architecture. In this way, end systems and applications can enjoy the advantage of strong security without the need to make any changes. Because IPSec encrypted packets look like ordinary IP packets, they can be easily routed through any IP network, such as the Internet, without any changes to the intermediate networking equipment. The only devices that know about the encryption are the endpoints. This feature greatly reduces the cost of implementation and management.

IP Security Architecture

IPSec combines several security technologies to protect the confidentiality, integrity, and authenticity of IP packets. IPSec actually refers to several related protocols as defined in RFCs 2401-2411 and 2451. Two of these standards define IPSec and *Internet Key Exchange* (IKE). IPSec defines the information that is added to an IP packet to enable confidentiality, integrity, and authenticity controls; it also defines how to encrypt the packet data. IKE is used to negotiate the security association between two entities and to exchange keying material. The use of IKE is optional, but it relieves users of the difficult and labor-intensive task of manually configuring security associations. IKE should be used in most real-world applications to enable large-scale, secure communications.

IPSec Services

IPSec provides security services at the IP layer by enabling a system for selecting required security protocols, determining the algorithm(s) to use for the service(s), and implementing any cryptographic keys required to provide the following services:

- Access control
- Connectionless integrity (a detection method of the IP packet itself)
- Data origin authentication
- Rejection of replayed packets (a form of partial sequence integrity)
- Confidentiality (encryption)
- Limited traffic-flow confidentiality

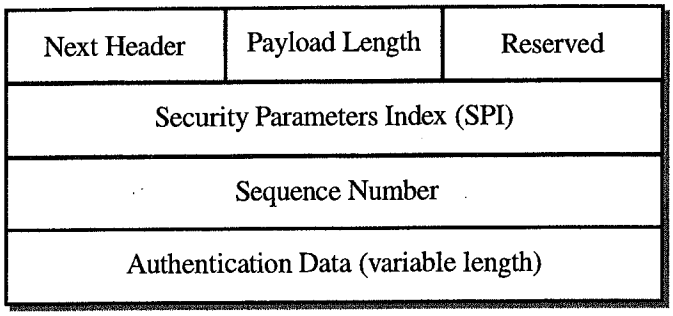
IPSec provides these services through the use of two protocols. The first one, the *authentication header* (AH) protocol, supports access control, data origin authentication, connectionless integrity, and the rejection of *replay* attacks, in which an attacker copies a packet and sends it out of sequence to confuse communicating nodes. The second protocol is the *encapsulating security payload* (ESP) protocol. ESP alone can support confidentiality, access control, limited traffic-flow confidentiality, and the rejection of replay attacks.

NOTE:
ESP and AH can be used in concert to provide all the services.

The Authentication Header Protocol

AH provides data integrity and authentication services for IP packets (see Figure 7-2). These services protect against attacks commonly mounted against open networks. AH uses a keyed-hash function rather than digital signatures because digital signature technology is too slow and would greatly reduce network throughput. Note, however, that AH does not provide confidentiality protection, so data can still be viewed as it travels across a network.

Figure 7-2
The authentication header protocol



AH contains the following fields:

- **Next Header** This field identifies the higher-level protocol following AH (for example, TCP, UDP, or ESP).
- **Payload Length** This field indicates the length of the AH contents.
- **Reserved** This field is reserved for future use. Currently, this field must always be set to zero.
- **Security Parameters Index** This field is a fixed-length, arbitrary value. When used in combination with the destination IP address, this value uniquely identifies a security association for this packet (that is, it indicates a set of security parameters for use in this connection).
- **Sequence Number** The field provides a monotonically increasing number for each packet sent with a given SPI. This value lets the recipient keep track of the order of the packets and ensures that the same set of parameters is not used for too many packets. The sequence number provides protection against replay attacks.
- **Authentication Data** This variable-length field contains the *integrity check value* (ICV) (see next section) for this packet. It may include padding to bring the length of the header to an integral multiple of 32 bits (in IPv4) or 64 bits (IPv6).

Integrity Check Value Calculation

The ICV, a truncated version of a *message authentication code* (MAC), is calculated by a MAC algorithm. IPSec requires that all implementations support at least HMAC-MD5 and HMAC-SHA1 (the HMAC symmetric authentication scheme supported by MD5 or SHA-1 hashes; see Chapter 6. To guarantee minimal interoperability, an IPSec implementation must support at least these schemes.

The ICV is computed using the following fields:

- The IP header fields that either do not change in transit or whose values are predictable upon arrival at the endpoint for the AH security association. Other fields are set to zero for the purpose of calculation.
- The entire contents of the AH header except for the Authentication Data field. The Authentication Data field is set to zero for the purpose of calculation.
- All upper-level protocol data, which is assumed to be immutable in transit.

NOTE:
The HMAC value is calculated completely, although it is truncated to 96 bytes (the default size for the Authentication Data field).

Transport and Tunnel Modes

AH services can be employed in two ways: in transport mode or in tunnel mode. The actual placement of the AH depends on which mode is used and on whether the AH is being applied to an IPv4 or an IPv6 packet. Figure 7-3 illustrates IPv4 and IPv6 packets before authentication services are applied.

In *transport* mode, the AH applies only to host implementations and provides protection for upper-layer protocols in addition to selected IP header fields. In this mode, AH is inserted after the IP header but before

Figure 7-3
IPv4 and IPv6 before AH is applied

Standard IPv4 packet

Original IP Header (any options)	TCP	Data

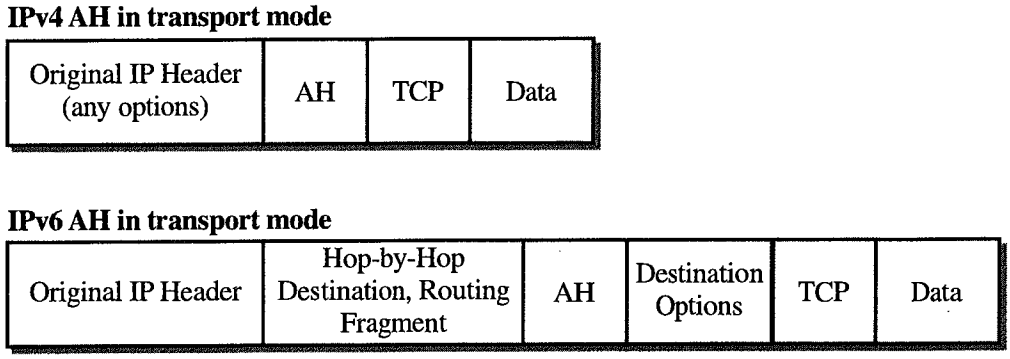
Standard IPv6 datagram

Original IP Header (any options)	Extension Headers (if present)	TCP	Data

any upper-layer protocol (such as, TCP, UDP) and before any other IPSec headers that have already been inserted. In IPv4, this calls for placing AH after the original IP header but before the upper-layer protocol. In IPv6, AH is viewed as an end-to-end payload; this means that intermediate routers should not process it. For this reason, the AH should appear after the original IP header, hop-by-hop, routing, and fragmentation extension headers. This mode is provided via the transport *security association* (SA). Figure 7-4 illustrates the AH transport mode positioning in typical IPv4 and IPv6 packets.

Figure 7-4

IPv4 and IPv6 header placement in transport mode



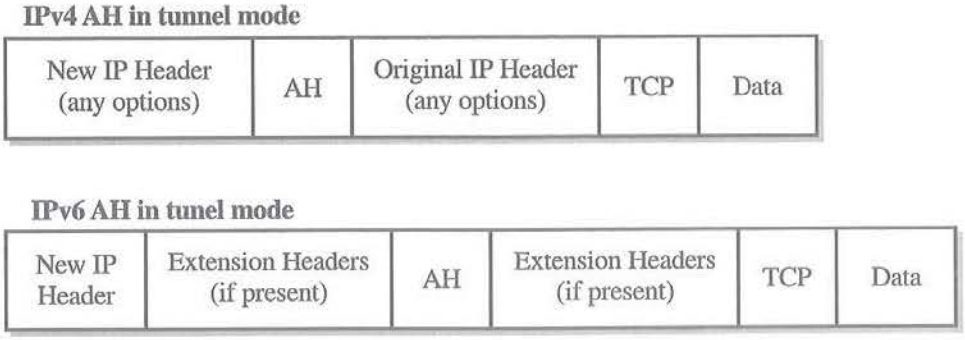
In *tunnel* mode, the AH can be employed in either host or security gateways. When AH is implemented in a security gateway (to protect transit traffic), tunnel mode must be used. In this mode, the AH is inserted between the original IP header and the new outer IP header. Whereas the inner IP header carries the ultimate source and destination addresses, the new outer IP header may contain distinct IP addresses (such as, addresses of firewalls or other security gateways). In tunnel mode, AH protects the entire inner IP packet, including the entire inner IP header. In tunnel mode, the position of AH relative to the outer IP header is the same as for AH in transport mode. This mode is provided via the tunnel SA. Figure 7-5 illustrates AH tunnel mode positioning for typical IPv4 and IPv6 packets.

NOTE:

ESP and AH headers can be combined in a variety of modes. The IPSec architecture document (RFC2401) describes the combinations of security associations that must be supported.

Figure 7-5

IPv4 and IPv6 header placement in tunnel mode

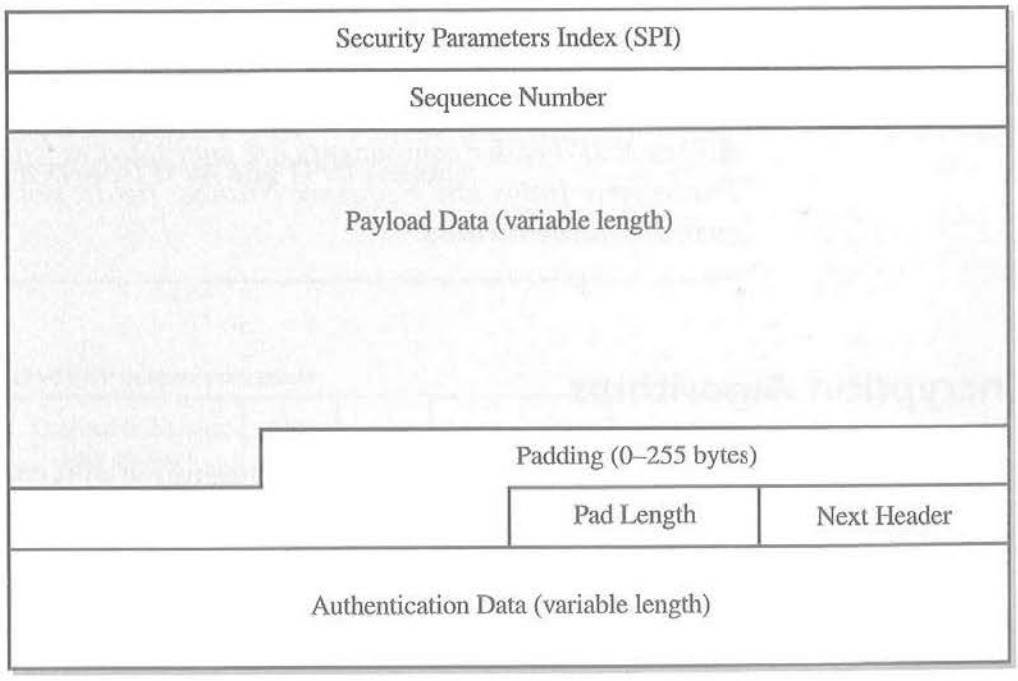


The Encapsulating Security Payload Protocol

The *encapsulating security payload* (ESP) protocol provides confidentiality services for IP data while in transit across untrusted networks. Optionally, ESP also provides authentication services. The format of ESP varies according to the type and mode of the encryption being used. In all cases the key associated with the encryption is selected using the SPI. Figure 7-6 illustrates the components of an ESP header.

Figure 7-6

Components of an ESP header



The ESP header contains the following fields:

- **Security Parameters Index** This field, as in the AH packet, is used to help uniquely identify a security association to be used.
- **Sequence Number** This field, again as in the AH packet, contains a counter that increases each time a packet is sent to the same address using the same SPI. It lets the recipient keep track of the packet order.
- **Payload Data** This variable-length field contains the actual encrypted data contents being carried by the IP packet.
- **Padding** This field provides space for adding bytes, as required by certain types of encryption algorithms (see Chapter 2). Data padding confuses *sniffers,* who try to access information about encrypted data in transit, in this case by trying to estimate how much data is being transmitted.
- **Pad Length** This field identifies how much of the encrypted payload is padding.
- **Next Header** This field identifies the type of data carried in the Payload Data field.
- **Authentication Data** This variable-length field contains a value that represents the ICV computed over the ESP packet minus the Authentication Data field. This field is optional and is included only if the authentication service is selected within the SA.

NOTE:

All the ESP header components are encrypted except for the Security Parameters Index and Sequence Number fields. Both of these fields, however, are authenticated.

Encryption Algorithms

The IPSec ESP standard currently requires that compliant systems have two cryptographic algorithms. Systems must have the DES algorithm using *cipher block chaining* (CBC) mode (see Chapter 2); compliant systems that require only authentication must have a NULL algorithm. However, other algorithms are defined for use by ESP services. Following are some of the defined algorithms:

- Triple DES
- RC5
- IDEA
- CAST
- BLOWFISH
- 3IDEA

ESP in Transport and Tunnel Modes

Like AH, ESP can be employed in two modes: transport mode and tunnel mode. These modes operate here in a similar way to their operation in AH, with one exception: with ESP, data, called *trailers,* are appended to the end of each packet.

In transport mode, ESP is used only to support host implementations and to provide protection for upper-layer protocols but not for the IP header itself. As with AH, in an IPv4 packet the ESP header is inserted after the original IP header and before any upper-layer protocols (for example, TCP, UDP) and before any other existing IPSec headers. In IPv6, ESP is viewed as an end-to-end payload; that is, intermediate routers should not process it. For this reason, the ESP header should appear after the original IP header, hop-by-hop header, routing header, and fragmentation extension header. In each case, the ESP trailer is also appended to the packet (encompassing the Padding, Pad Length, and Next Header fields). Optionally, the ESP authentication data field is appended if it has been selected. Figure 7-7 illustrates the ESP transport mode positioning in typical IPv4 and IPv6 packets.

Figure 7-7

IPv4 and IPv6 header placement in transport mode

IPv4 ESP in transport mode

Original IP Header (any options)	ESP Header	TCP	Data	ESP Trailer	ESP Authentication

IPv6 AH in transport mode

Original IP Header (any options)	Hop-by-Hop Destination, Routing Fragmentation	ESP Header	Destination Options	TCP	Data	ESP Trailer	ESP Authentication

Tunnel mode ESP can be employed by either hosts or security gateways. When ESP is implemented in a security gateway (to protect subscriber transit traffic), tunnel mode must be used. In this mode, the ESP header is inserted between the original IP header and the new outer IP header. Whereas the inner IP header carries the ultimate source and destination addresses, the new outer IP header may contain distinct IP addresses (such as, addresses of firewalls or other security gateways). In tunnel mode, ESP protects the entire inner IP packet, including the entire inner IP header. The position of ESP in tunnel mode, relative to the outer IP header, is the same as for ESP in transport mode. Figure 7-8 illustrates ESP tunnel mode positioning for typical IPv4 and IPv6 packets.

Figure 7-8

IPv4 and IPv6 header placement in tunnel mode

IPv4 AH in tunnel mode

New IP Header (any options)	ESP Header	Original IP Header (any options)	TCP	Data	ESP Trailer	ESP Authentication

IPv6 AH in tunnel mode

New IP Header	New Extension Headers	ESP Header	Original IP Header	Original Extension Header	TCP	Data	ESP Trailer	ESP Authentication

NOTE:

ESP and AH headers can be combined in a variety of modes. The IPSec architecture document describes the combinations of security associations that must be supported.

Security Associations

To communicate, each pair of hosts using IPSec must establish a *security association* (SA) between them. The SA groups together all the things that you need to know about how to communicate securely with someone else, such as the type of protection used, the keys to be used, and the valid duration of this SA. The SA establishes a one-way relationship between the sender and the receiver. For peer communications, a second SA is needed.

You can think of an SA as a secure channel through the public network to a certain person, group of people, or network resource. It's like a contract with whoever is at the other end. The SA also has the advantage in that it lets you construct classes of security channels. If you need to be a little more careful when talking to one party than another, the rules of your SA with that party can reflect extra caution—for example, specifying stronger encryption.

A security association is uniquely identified by three parameters:

- **Security Parameters Index** This bit string uniquely identifies a security association relative to a security protocol (for example, AH or ESP). The SPI is located within AH and ESP headers so that the receiving system can select the SA under which a received packet will be processed.
- **IP Destination Address** This parameter indicates the destination IP address for this SA. The endpoint may be that of an end user system or a network system such as a gateway or firewall. Although in concept this parameter could be any address type (multicast, broadcast, and so on), currently it can be only a unicast address.
- **Security Protocol Identifier** This parameter indicates whether the association is that of an AH or an ESP security association.

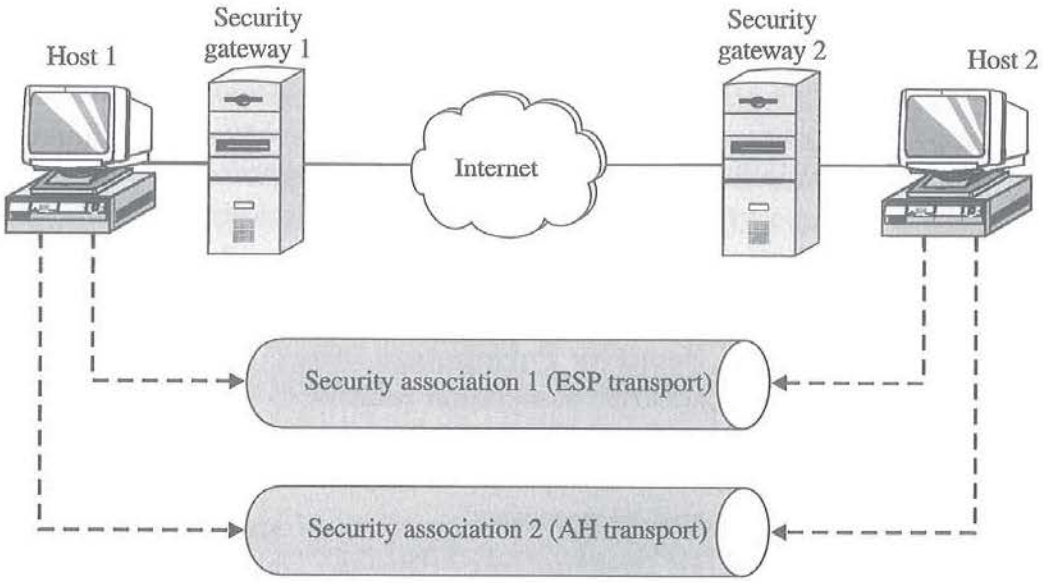
Combining Security Associations

Using a single SA, you can deploy either AH or ESP (but not both) to implement security for IP packets. However, there is no restriction on the use of multiple SAs, usually referred to as an SA *bundle*. The order in which the SAs are bundled is defined by your security policy. IPSec does define two ways of combining SAs: transport adjacency and iterated tunneling.

Transport adjacency refers to the process of applying multiple transport SAs to the same IP packet without using tunneling SAs. This level of combination lets you apply both AH and ESP IP packets but does not enable further nesting. The idea is that strong algorithms are used in both AH and ESP, so further nesting would yield no additional benefits. The IP packet is processed only once: at its final destination. Figure 7-9 illustrates the application of transport adjacency.

In *iterated tunneling*, you apply multiple (layered) security protocols by using IP tunneling. This approach allows multiple levels of nesting. Each

Figure 7-9
Transport adjacency



tunnel can originate or terminate at a different IPSec site along the path. Figure 7-10 shows three basic cases of iterated tunneling supported by the IPSec protocol.

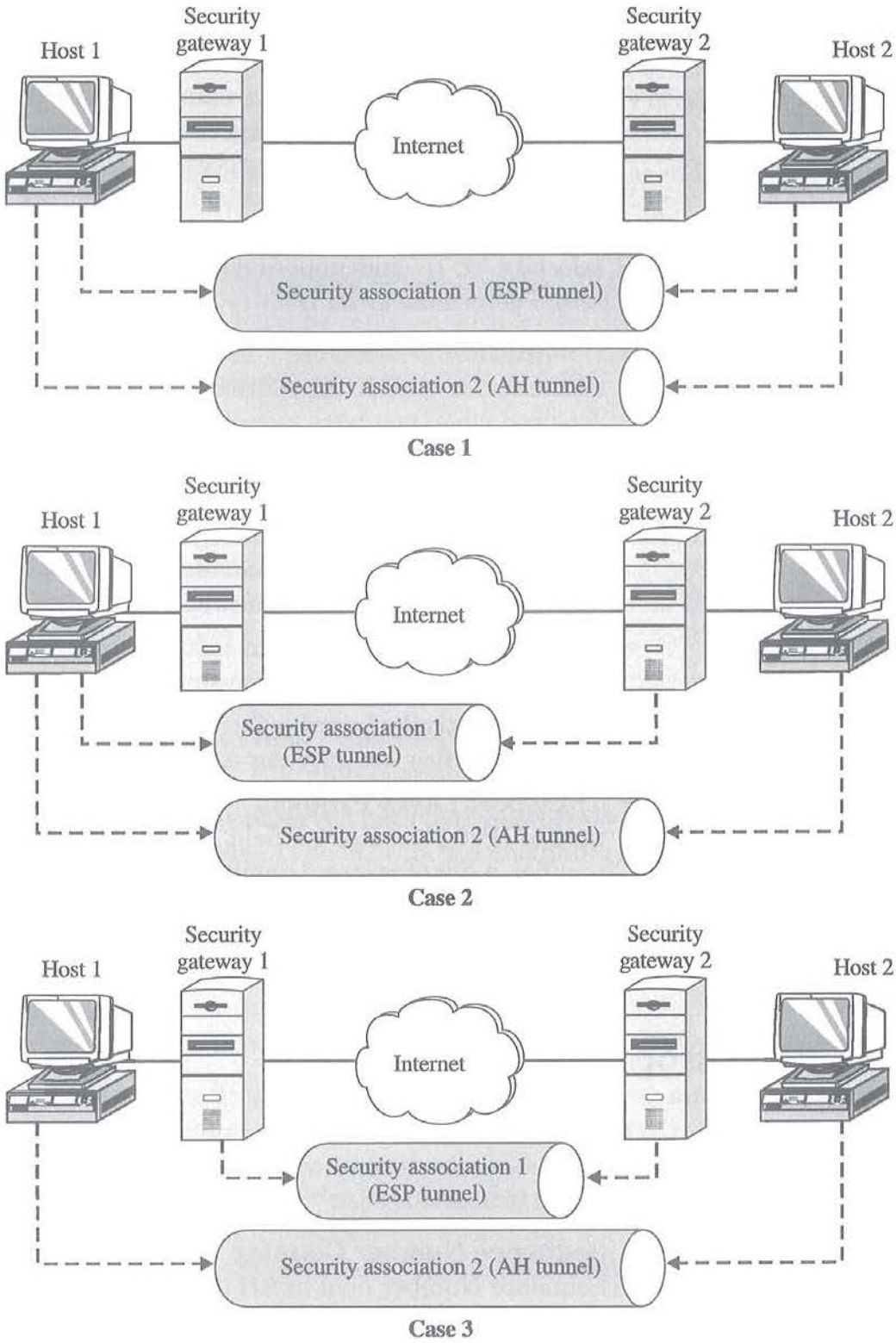
NOTE:
You can also combine transport adjacency and iterated tunneling. For example, you could construct an SA bundle from one tunnel SA and one or two transport SAs applied in sequence.

Security Databases

IPSec contains two nominal databases: the *Security Policy Database* (SPD) and the *Security Association Database* (SAD). SPD specifies the policies that determine the disposition of all IP traffic, inbound or outbound. SAD contains parameters that are associated with each currently active security association.

Figure 7-10

Three cases of iterated tunneling



Security Policy Database

An SA is nothing more than a management construct that is used to enforce a security policy. Because SPD is responsible for all IP traffic, it must be consulted during the processing of all traffic (inbound and outbound), including non-IPSec traffic. To support this, SPD requires distinct entries for inbound and outbound traffic; these entries are defined by a set of *selectors,* or IP and upper-layer protocol field values. The following selectors determine an SPD entry:

- **Destination IP Address** This can be a single IP address, a list of addresses, or a wildcard address. Multiple and wildcard addresses are used when you have more than one source system sharing the same SA (for example, behind a gateway).
- **Source IP Address** This can be a single IP address, a range of addresses, or a wildcard address. Multiple and wildcard addresses are used when you have more than one source system sharing the same SA (for example, behind a gateway).
- **Name** This can be either an X.500 distinguished name or a user identifier from the operating system.
- **Data Sensitivity Level** This is used for systems that provide information flow security (for example, unclassified or secret).
- **Transport Layer Protocol** This is obtained from the IPv4 Protocol field or IPv6 Next Header field. It can be an individual protocol number, a list of protocol numbers, or a range of protocol numbers.
- **Source and Destination Ports** These can be individual UDP or TCP port values, or a wildcard port.

Security Association Database

Each implementation of IPSec contains a nominal SAD, which is used to define the parameters associated with each SA. The following parameters are used to define an SA:

- **Sequence Number Counter** A 32-bit value used to generate the Sequence Number field in AH or ESP headers.
- **Sequence Counter Overflow** A flag indicating whether overflow of the sequence number counter should generate an auditable event and prevent transmission of additional packets on the SA.

- **Anti-Replay Window** A 32-bit counter that is used to determine whether an inbound AH or ESP packet is a replay.
- **AH Information** Parameters relating to the use of AH (such as authentication algorithms, keys, and key lifetimes).
- **ESP Information** Parameters relating to the use of ESP (such as encryption algorithms, keys, key lifetimes, and initialization values).
- **Lifetime of This Security Association** A time interval or byte count that specifies an SA's duration of use. When the duration is complete the SA must be replaced with a new SA (and new SPI) or terminated, and this parameter includes an indication of which of these actions should occur.

NOTE:

If a time interval is employed, and if IKE employs X.509 certificates for SA establishment, the SA lifetime must be constrained by the validity intervals of the certificates and by the "NextIssueDate" of the CRLs used in the IKE exchange for the SA. For more about CRLs, see Chapter 6.

- **IPSec Protocol Mode** Specifies the mode-tunnel, transport, or wildcard-of AH or ESP that is applied to traffic on this SA.
- **Path MTU** Any observed path *maximum transferable unit* (MTU) and aging variables. (The MTU is the maximum size of a packet without fragmentation.)

Key Management

As with any security protocol, when you use IPSec you must provide key management, such as supplying a means of negotiating with other people the protocols, encryption algorithms, and keys to be used in data exchange. In addition, IPSec requires that you keep track of all such agreements between the entities. IETF's IPSec working group has specified that compliant systems must support both manual and automated SA and cryptographic key management.

Following are brief descriptions of these techniques:

- **Manual** Manual key and SA management are the simplest forms of key management. A person (usually a systems administrator) manually configures each system, supplying the keying material and SA management data relevant to secure communication with other systems. Manual techniques can work effectively in small, static environments, but this approach is not very practical for larger networks.
- **Automated** By using automated key management protocols, you can create keys as needed for your SAs. Automated management also gives you a great deal of scalability for larger distributed systems that are still evolving. You can use various protocols for automated management, but IKE seems to have prevailed as the current industry standard.

Internet Key Exchange

IKE is not a single protocol; rather, it is a hybrid of two protocols. IKE integrates the *Internet Security Association and Key Management Protocol* (ISAKMP) with the Oakley key exchange protocol.

IKE performs its services in two phases. In the first phase, two IKE peers establish a secure, authenticated channel for communication by using a common IKE security association. IKE provides three modes of exchanging keying information and setting up SAs (see next section); in this first phase, only main or aggressive mode is employed.

In the second phase, SAs are negotiated on behalf of services such as IPSec or any other service that needs keying material or parameter negotiation. The second phase is accomplished via a quick mode exchange.

Main Mode

IKE's *main* mode provides a three-stage mechanism for establishing the first-phase IKE SA, which is used to negotiate future communications. In this mode, the parties agree on enough things (such as authentication and confidentiality algorithms, hashes, and keys) to be able to communicate securely long enough to set up an SA for future communication. In this mode, three two-way messages are exchanged between the SA initiator and the recipient.

As shown in Figure 7-11, in the first exchange, the two parties agree on basic algorithms and hashes. In the second, they exchange public keys for

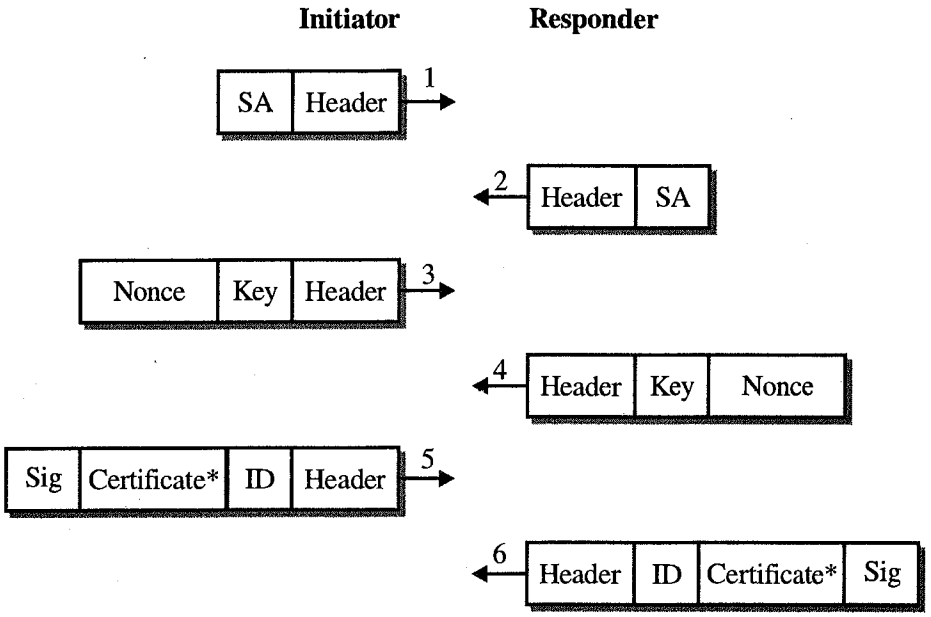


Figure 7-11

Transactions in IKE's main mode

* Indicates inclusion of optional certificate payload

a Diffie-Hellman exchange and pass each other *nonces* (random numbers signed and returned by the other party to prove its identity). In the third exchange, they verify those identities.

Aggressive Mode

Aggressive mode is similar to main mode in that aggressive mode is used to establish an initial IKE SA. However, aggressive mode differs in the way the messages are structured, thereby reducing the number of exchanges from three to two.

In aggressive mode, the proposing party generates a Diffie-Hellman pair at the beginning of the exchange and does as much as is practical with that first packet: proposing an SA, passing the Diffie-Hellman public value, sending a nonce for the other party to sign, and sending an ID packet that the responder can use to check the initiator's identity with a third party (see Figure 7-12). The responder then sends back everything needed to complete the exchange. All that's left for the initiator to do is to confirm the exchange.

The advantage of aggressive mode is its speed, although aggressive mode does not provide identity protection for the communicating parties. This means that the parties exchange identification information before

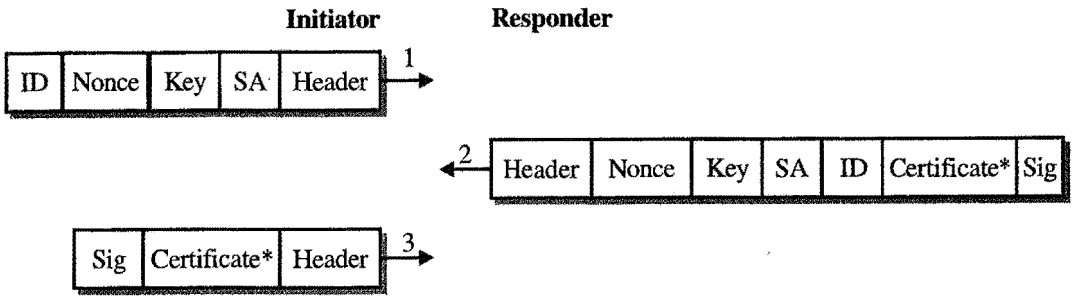


Figure 7-12
Aggressive mode transactions

* Indicates inclusion of optional certificate payload

establishing a secure SA in which to encrypt it. As a result, someone monitoring an aggressive mode exchange can identify the entity that has just formed a new SA.

Quick Mode

After two communicating entities have established an IKE SA using either main or aggressive mode, they can use *quick* mode. Quick mode, unlike the other two modes, is used solely to negotiate general IPSec security services and to generate fresh keying material.

Because the data is already inside a secure tunnel (every packet is encrypted), you can afford to be a little more flexible in quick mode. Quick mode packets are always encrypted and always start with a *hash payload,* which is composed using the agreed-upon pseudo-random function and the derived authentication key for the IKE SA. The hash payload is used to authenticate the rest of the packet. Quick mode defines which parts of the packet are included in the hash.

As shown in Figure 7-13, the initiator sends a packet with the quick mode hash; this packet contains proposals and a nonce. The responder

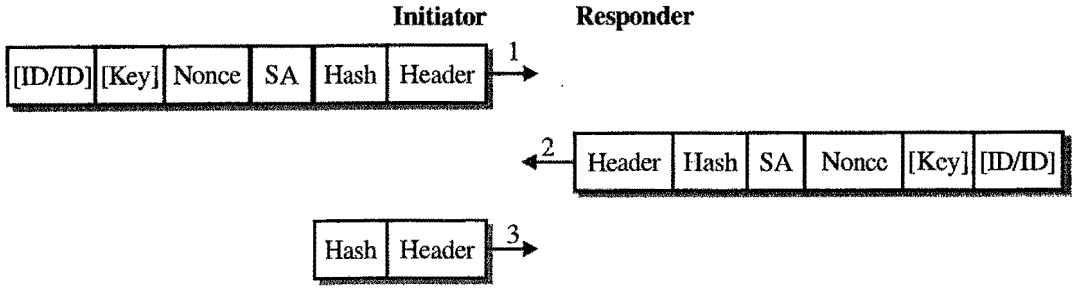


Figure 7-13
Quick mode transactions

then replies with a similar packet, this time generating its own nonce and including the initiator's nonce in the quick mode hash for confirmation. The initiator then sends back a confirming quick mode hash of both nonces, completing the exchange. Finally, using the derivation key as the key for the hash, both parties perform a hash of a concatenation of the following: the nonces, the SPI, and the protocol values from the ISAKMP header that initiated the exchange. The resulting hash becomes the new password for that SA.

Secure Sockets Layer

Secure Sockets Layer (SSL), the Internet protocol for session-based encryption and authentication, provides a secure pipe between two parties (the *client* and the *server*). SSL provides server authentication and optional client authentication to defeat eavesdropping, tampering, and message forgery in client-server applications. By establishing a shared secret between the two parties, SSL provides privacy.

SSL works at the transport layer (below the application layer) and is independent of the application protocol used. Therefore, application protocols (HTTP, FTP, TELNET, and so on) can transparently layer on top of SSL, as shown in Figure 7-14.

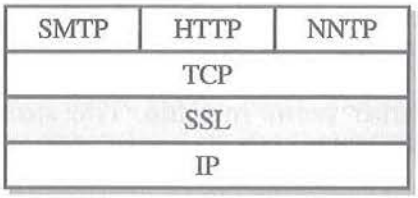


Figure 7-14
SSL in the TCP/IP stack

The History of SSL

Netscape originally developed SSL in 1994. Since then, SSL has become widely accepted and is now deployed and supported in all major Web browsers and servers as well as various other software and hardware products (see Figure 7-15). This protocol currently comes in three versions: SSLv2, SSLv3, and TLSv1 (also known as SSLv3.1). Although all three can be found in use around the world, SSLv3, released in 1995, is the predominant version.

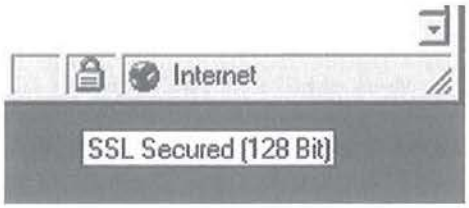


Figure 7-15
The padlock symbol in this browser denotes the use of SSL for Web security

SSLv3 solved many of the deficiencies in the SSLv2 release. SSLv3 enables either party (client or server) to request a new handshake (see next section) at any time to allow the keys and ciphers to be renegotiated. Other features of SSLv3 include data compression, a generalized mechanism for Diffie-Hellman and Fortezza key exchanges and non-RSA certificates, and the ability to send certificate chains.

In 1996, Netscape turned the SSL specification over to the IETF. Currently, the IETF is standardizing SSLv3 in its *Transport Layer Security* (TLS) working group. TLSv1 is very similar to SSLv3, with only minor protocol modifications. The first official version of TLS was released in 1999.

Session and Connection States

Any system of the type discussed in this chapter is composed of two parts: its state and the associated state transitions. The system's *state* describes the system at a particular point in time. The *state transitions* are the processes for changing from one state to another. The combination of all possible states and state transitions for a particular object is called a *state machine*. SSL has two state machines: one for the client side of the protocol and another for the server side. Each endpoint must implement the matching side of the protocol. The interaction between the state machines is called the *handshake*.

It is the responsibility of the SSL handshake protocol to coordinate the states of the client and server, thereby enabling each one's protocol state machine to operate consistently even though the state is not exactly parallel. Logically, the state is represented twice: once as the current operating state and (during the handshake protocol) a second time as the pending state. Additionally, separate read and write states are main-

tained. An SSL session can include multiple secure connections, and parties can have multiple simultaneous sessions.

The SSL specification defines the elements of a session state as follows:

- **Session Identifier** An arbitrary byte sequence chosen by the server to identify an active or resumable session state.
- **Peer Certificate** X.509v3 certificate of the peer. This element of the state can be null.
- **Compression Method** The algorithm used to compress data before encryption.
- **Cipher Spec** Specifies the bulk data encryption algorithm (null, DES, and so on) and a MAC algorithm (such as MD5 or SHA-1) used for message authentication. It also defines cryptographic attributes such as the hash size.
- **Master Secret** 48-byte secret shared between the client and the server.
- **Is Resumable** A flag indicating whether the session can be used to initiate new connections.

Furthermore, the SSL specification defines the following elements of a connection state:

- **Server and Client Random** Byte sequences that are independently chosen by the server and the client for each connection.
- **Server Write MAC Secret** The secret key that is used in MAC operations on data written by the server.
- **Client Write MAC Secret** The secret key that is used in MAC operations on data written by the client.
- **Server Write Key** The symmetric cipher key for data encrypted by the server and decrypted by the client.
- **Client Write Key** The symmetric cipher key for data encrypted by the client and decrypted by the server.
- **Initialization Vectors** The *initialization vector* (IV) required for each block cipher used in CBC mode. This field is first initialized by the SSL handshake protocol. Thereafter, the final ciphertext block from each record is preserved for use with the following record.
- **Sequence Numbers** Each party maintains separate sequence numbers for transmitted and received messages for each connection. When a party sends or receives a *change cipher spec* message (see

later section titled "The Change Cipher Spec Protocol"), the appropriate sequence number is set to zero. Sequence numbers are of type uint64 and may not exceed 2^{64}-1.

The Record Layer Protocol

As data is transmitted to and received from upper application layers, it is operated on in the SSL record layer (see Figure 7-16). It is here that data is encrypted, decrypted, and authenticated.

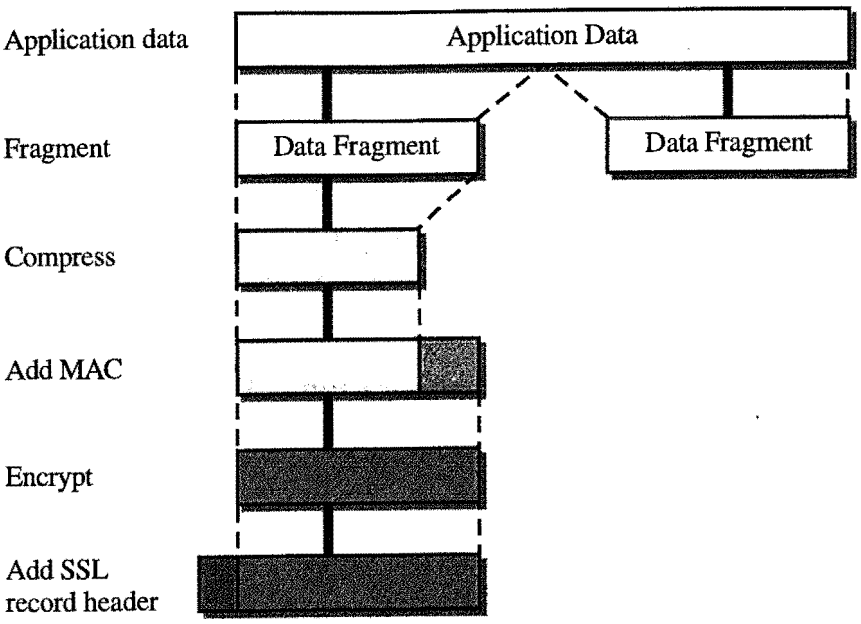


Figure 7-16
Overview of the SSL record layer

The following five steps take place in the record layer:

1. As the record layer receives an uninterrupted stream of data from the upper application layer, the data is *fragmented*, or broken into manageable plaintext blocks (or *records*). Each record is 16K or smaller.
2. Optionally, the plaintext records are compressed using the compression algorithm defined by the current session state.
3. A MAC is computed for each of the plaintext records. For this purpose, the shared secret key, previously established, is used.

4. The compressed (or plaintext) data and its associated MAC are encrypted using the symmetric cipher that has been previously agreed upon for this session. Encryption may not increase the overall length of the record beyond 1,024 bytes.
5. A header is added to each record as a prefix consisting of the following fields:

   Content Type This field indicates the protocol used to process the enclosed record in the next-higher level.
   Major Version This field indicates the major version of SSL in use. For example, TLS has the value 3.
   Minor Version This field indicates the minor version of SSL in use. For example, TLS has the value 1.
   Compressed Length This field indicates the total length in bytes of the plaintext record.

The party receiving this information reverses the process, that is, the decryption and authentication functions are simply performed in reverse.

NOTE:
Sequence numbers are also included with each transmission so that missing, altered, or extra messages are detectable.

The Change Cipher Spec Protocol

The change cipher spec protocol is the simplest of the SSL-specific protocols. It exists to signal a transition in the ciphering strategies. The protocol consists of a single message, which is encrypted and compressed by the record layer as specified by the current cipher specification. Before finishing the handshake protocol, both the client and the server send this message to notify each other that subsequent records will be protected under the just-negotiated cipher specification and associated keys. An unexpected change cipher spec message should generate an unexpected_message alert.

The Alert Protocol

One of the content types supported by the SSL record layer is the *alert* type. The alert protocol conveys alert messages and their severity to parties in an SSL session. Just as application data is processed by the record layer, alert messages are compressed and encrypted as specified by the current connection state.

When either party detects an error, the detecting party sends a message to the other. If the alert message has a fatal result, both parties immediately close the connection. Both parties are required to forget any session identifier, keys, and secrets associated with a failed connection. For all other nonfatal errors, both parties can cache information to resume the connection.

The following error alerts are always fatal:

- **Unexpected_message** This message is returned if an inappropriate message was received.
- **Bad_record_mac** This message is returned if a record is received without a correct message authentication code.
- **Decompression_failure** This message is returned if the decompression function received improper input (for example, the data could not be decompressed or it decompresses to an excessive length).
- **Handshake_failure** The return of this message indicates that the sender was unable to negotiate an acceptable set of security parameters given the available options.
- **Illegal_parameter** A field in the handshake was out of range or inconsistent with other fields.

The remaining alerts are as follows:

- **No_certificate** This message can be sent in response to a certification request if no appropriate certificate is available.
- **Bad_certificate** The return of this message indicates that a certificate was corrupted (that is, it contained a signature that did not verify).
- **Unsupported_certificate** The return of this message indicates that a certificate was of an unsupported type.
- **Certificate_revoked** The return of this message indicates that a certificate was revoked by its signer.

- **Certificate_expired** The return of this message indicates that a certificate has expired.
- **Certificate_unknown** The return of this message indicates that some other (unspecified) issue arose in processing the certificate, and it was rendered unacceptable.
- **Close_notify** This message notifies the recipient that the sender will not send any more messages on this connection. Each party is required to send this message before closing the write side of a connection.

The Handshake Protocol

The SSL handshake protocol is responsible for establishing the parameters of the current session state. As shown in Figure 7-17, both parties agree on a protocol version, select cryptographic algorithms, optionally

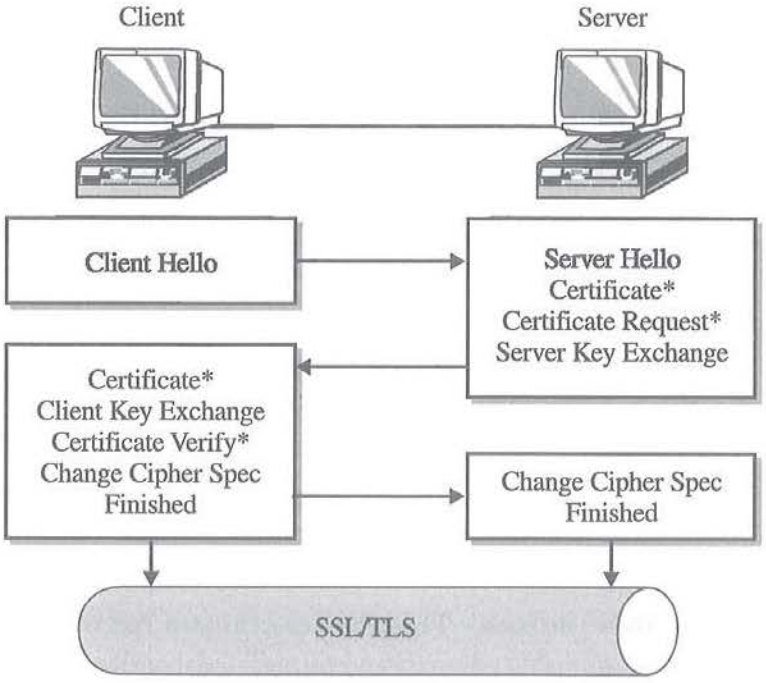


Figure 7-17
Overview of the handshake protocol

* Indicates optional or situation-dependent message

authenticate each other, and use public-key encryption techniques to generate shared secrets (described later under "Cryptographic Computations") through a series of messages exchanged between the client and the server. The following subsections explain in detail the steps of the handshake protocol.

The Client Hello Message

For communications to begin between a client and a server, the client must first initiate a *client hello* message. The contents of this message provide the server with data (such as version, random value, session ID, acceptable ciphers, and acceptable compression methods) about variables that are supported by the client. This message can come as a client response to a hello request (from the server), or on its own initiative the client can use it to renegotiate the security parameters in an existing connection.

A client hello message contains the following fields:

- **Client_version** This field provides the highest SSL version that is understood by the client.
- **Random** This field contains a client-generated random structure that will be used for later cryptographic computations in the SSL protocol. The 32-byte random structure is not entirely random. Rather, it is made up of a 4-byte date/time stamp, with the remaining 28 bytes of data being randomly generated. The date/time stamp assists in the prevention of replay attacks.
- **Session_id** This field contains a variable-length session identifier. This field should be empty if no session identifier is available or if the client wishes to generate new security parameters. If the session identifier does, however, contain a value, that value should identify a previous session between the same client and server whose security parameters the client wishes to reuse. (The reuse of session identifiers is discussed later in this chapter under "Resuming Sessions.")
- **Cipher_suites** This field contains a list of combinations for cryptographic algorithms supported by the client. This list is ordered according to the client's preference (that is, first choice first). This list is used to make the server aware of the cipher suites available to the client, but it is the server that ultimately decides which cipher will be

used. If the server cannot find an acceptable choice from the list, it returns a handshake failure alert and closes the connection.

- **Compression_methods** Similar to the cipher_suites field, this field lists all supported compression methods known to the client. Again, this list is ordered according to the client's preference. Although this field is not regularly used in SSLv3, future TLS versions will require support for it.

NOTE:

After the client hello message is sent, the client waits for a server hello message. If the server returns any handshake message other than a server hello, a fatal error results and communications are halted.

The Server Hello Message

After the server processes the client hello message, it can respond with either a handshake failure alert or a server hello message. The content of the *server hello* message is similar to that of the client hello. The difference is that whereas the client hello is used to list its capabilities, the server hello is used to make decisions that are then passed back to the client.

The server hello message contains the following fields:

- **Server_version** This field contains the version that was decided on by the server; this version will be used for further communications with the client. The server bases its decision on the highest version supported by both parties. For example, if the client states that it can support SSLv3 but the server supports up to TLS (or SSLv3.1), the server will decide on SSLv3.
- **Random** This field, similar in structure to that of the client's, is used for future cryptographic operations within SSL. It must, however, be independent of and different from that generated by the client.
- **Session_id** This field provides the identity of the session corresponding to the current connection. If the session identifier that was received by the client is nonempty, the server will look in the session cache for a match. If a match is found, the server can

establish a new connection, resuming the specified session state. In that case, the server returns the same value that was provided by the client, indicating a resumed session. Otherwise, this field contains a different value, identifying a new session.

- **Cipher_suite** This field indicates the single cipher suite selected by the server from the list provided by the client.
- **Compression_method** Similar to the cipher_suite message, this field indicates the single compression method selected by the server from the list provided by the client.

The Server Certificate Message

Immediately after the server hello message, the server can send its certificate or chain of certificates to be authenticated. Authentication is required in all cases of agreed-on key exchange (with the exception of anonymous Diffie-Hellman). The appropriate certificate type (generally an X.509v3 server certificate) must be used for the key exchange algorithm of the selected cipher suite.

This message also makes the public key available to the client. This public key is what the client uses to encrypt the actual session key.

NOTE:
A similar message type can be used for client-side authentication support.

The Server Key Exchange Message

The server sends a *server key exchange* message only when no certificate is present, when the certificate is used only for signing (as with *Digital Signature Standard* [DSS] certificates and signing-only RSA certificates), or when Fortezza key exchange is used. The message complements the cipher suite that was previously stated in the server hello message, providing the algorithm variables that the client needs in order to continue. These values depend on which algorithm has been selected. For example, with RSA key exchange (where RSA is used only for signatures), the mes-

sage would contain a temporary RSA public key exponent and modulus, and a signature of those values.

The Certificate Request Message

The optional *certificate request* message requests a certificate from the client for authentication purposes. It is made up of two parameters. The first parameter indicates the acceptable certificate types (RSA—signature-only, DSS—signature-only, and so on). The second parameter indicates the acceptable distinguished names of acceptable certificate authorities.

NOTE:
This message is to be used only by non-anonymous servers (servers not using anonymous Diffie-Hellman).

The Server Hello Done Message

As the name implies, the *server hello done* message is sent by the server to the client to indicate the end of the server hello and to signal that no further associated server hello messages are coming. After this message is sent, the server waits for the client to respond. On receipt of the server hello done message, the client should verify the certificate and any certificate chain sent by the server (if required) and should verify that all server hello parameters received are acceptable.

The Client Certificate Message

The *client certificate* message is the first message that a client can send after a server hello done message is received, and it is sent only if the server requests a certificate. If the client does not have a suitable certificate (for example, an X.509v3 client certificate) to send, it should send a no_certificate alert instead. Although this alert is only a warning, it is a matter of the server's discretion whether to continue or terminate communications.

The Client Key Exchange Message

Like the server key exchange message, the *client key exchange* message allows the client to send key information to the server. Unlike the server key exchange message, however, this key information pertains to a symmetric-key algorithm that both parties will use for the session.

NOTE:
Without the information contained in this message, communications cannot continue.

The content of this message depends on the type of key exchange, as follows:

- **RSA** The client generates a 48-byte *pre-master secret*, which it encrypts by using either the public key from the server's certificate or a temporary RSA key from a server key exchange message. This result is then sent to the server to compute the master secret key. (Computation of the master secret is discussed later in this chapter under "Cryptographic Computations.")
- **Ephemeral or Anonymous Diffie-Hellman** The client provides its own Diffie-Hellman public parameters to the server.
- **Fortezza** The client calculates public parameters using the public key in the server's certificate along with private parameters in the client's token. These parameters are then sent to the server.

The Certificate Verify Message

The *certificate verify* message is used to provide explicit verification of a client certificate. When using client authentication, the server authenticates the client using the private key. This message contains the pre-master secret key signed with the client's private key. The server validates the key against the client's certificate. The server is not required to authenticate itself to the client. Because the pre-master secret is sent to the server using the server's public key, only the legitimate server can decrypt it with the corresponding private key.

The Finished Message

Next, the client sends a change cipher spec message, followed immediately by the *finished* message. When the server receives the finished message, it too sends out a change cipher spec message and then sends its finished message. At this point the handshake protocol is complete and the parties can begin to transfer application data securely.

Be aware that the finished message is the first to be protected with the just-negotiated algorithms, keys, and secrets. As a result, the communicating parties can verify that the key exchange and authentication processes were successful. No acknowledgment of the finished message is required; parties can begin sending encrypted data immediately after sending the finished message. Recipients of finished messages must verify that the contents are correct.

NOTE:

The change cipher spec message is actually part of the change cipher spec protocol and not the handshake protocol.

Ending a Session and Connection

Before the end of communications, the client and the server must share knowledge that the connection is ending. This arrangement protects the session from a possible truncation attack, whereby an attacker tries to compromise security by prematurely ending communications. Either party can terminate the session by sending a close_notify alert before closing its own write session. When such an alert is received, the other party must send its own close_notify alert and close down immediately, discarding any pending writes.

NOTE:

If the SSL session is closed before either party sends a close_notify message, the session cannot be resumed.

Resuming Sessions

Public-key encryption algorithms are very slow. To improve performance, the parties can cache and reuse information exchanged during the handshake protocol. This process is called session ID reuse. If it is determined during the handshake protocol that the client and the server sides of the protocol share a session ID, the public-key and authentication operations are skipped, and the previously generated shared secret is reused during key generation.

Both parties (client and server) must agree to reuse a previous session ID. If either party suspects that the session may have been compromised or that certificates may have expired or been revoked, it should force a full handshake. Because an attacker who obtains a master secret key may be able to impersonate the compromised party until the corresponding session ID expires, the SSL specification suggests that the lifetime of cached session IDs expire after 24 hours. It also suggests that applications that run in relatively insecure environments should not write session IDs to stable storage.

Cryptographic Computations

We have used the term "shared secret" to explain how traffic is encrypted in SSL. Now let's look at the generation of the *master secret*, which is derived from the pre-master secret. In the case of RSA or Fortezza cryptography, the pre-master secret is a value generated by the client and sent to the server via the client key exchange message. In Diffie-Hellman cryptography, the pre-master secret is generated by each side (server and client) using the other party's Diffie-Hellman public values.

In each of these three cases, after a pre-master secret is generated and both sides are aware of it, the master secret can be computed. The master secret, which is used as the shared secret, is made up of several hash calculations of data previously exchanged in messages. Figure 7-18 shows the format of these calculations.

Encryption and Authentication Algorithms

SSLv3.0 supports a wide range of algorithms that provide various levels of security. These algorithms (encryption, key exchange, and authentication) support a total of 31 cipher suites, although some of them provide little or

Figure 7-18
Generating a master secret

```
Master_secret = MD5(pre_master_secret + SHA('A' + pre_master_secret +
                    ClientHello.random + ServerHello.random)) +
                MD5(pre_master_secret + SHA 'BB' + pre_master_secret +
                    ClientHello.random + ServerHello.random)) +
                MD5(pre_master_secret + SHA 'CCC' + pre_master_secret +
                    ClientHello.random + ServerHello.random)) +

*The characters A, BB, and CCC are actual
 ASCII values
```

no security in today's world. One such cipher suite is based on anonymous Diffie-Hellman, and the specification strongly discourages its use because it is vulnerable to man-in-the-middle attacks.

Summary

Security protocols make use of the various technologies described up to this point in the book to provide the necessary security. All cryptographic algorithms, whether they are symmetric, asymmetric, message digests, or message authentication, codes do very little on their own; instead, they are the basis for the security provided through set standard protocols such as IPSec or SSL.

Security protocols can be placed within the various layers of the TCP/IP networking stack. IPSec, for example, is located at the IP layer, while SSL is located between the TCP and application layers. The lower in the TCP/IP stack a protocol is placed, the more flexible and less user-intrusive the protocol is.

IPSec plays an important role in securing IP networks to provide private communications. It enables a wide range of security services, not only confidentiality but also authentication, access control, and protection against replay attacks. These services are available through the use of either or both the authentication header (AH) and encapsulating security payload (ESP).

SSL provides security at the transport (TCP) layer, which is below the application layer. The security provided by SSL can be thought of as a secure pipe placed between a client and server. Data is authenticated confidentially while in the pipe. It should be noted, however, that once either system (client or server) receives the data, the data is returned to its unprotected clear state.

Real-World Examples

Various products are available that provide security using the IPSec and SSL protocols. There are several toolkits available for developers who wish to build IPSec and SSL into their applications. RSA Security, Inc. provides two such commercial products, BSAFE Crypto-C/J and SSL-C/J, both of which are available for C programmers as well as Java programmers.

Other companies, however, have created end-user software and embedded hardware products using IPSec and SSL. In fact, most *virtual private networks* (VPNs) are based on the popular IPSec protocol. For example, Microsoft Windows 2000 makes use of IPSec to provide a VPN, something most end users aren't aware is available. Just as IPSec can be found in a various software and hardware products, so can SSL. SSL is by far the most widely distributed security protocol when it comes to e-commerce. One reason for SSL's widespread use is that it is incorporated in every copy of Netscape and Internet Explorer available today. SSL is also found within the operating system of various platforms. Many Linux vendors have included SSL in their systems; it provides security not only for HTTP communications, but also for other protocols as well, such as NNTP, SMTP, and FTP.

CHAPTER 8

Application-Layer Security Protocols

Like Chapter 7, this chapter looks at security protocols that are used in today's networks. But unlike the protocols described in Chapter 7, the protocols discussed in this chapter provide security services for specific applications (such as Simple Mail Transfer Protocol *(SMTP),* Hypertext Transfer Protocol *(HTTP), and so on). Figure 8-1 shows each of these application protocols along with its location in the* Transportation Control Protocol/Internet Protocol *(TCP/IP) stack. This chapter provides a detailed look at two well-known security protocols—S/MIME and SET—which operate above the application layer.*

S/MIME

Secure/Multipurpose Internet Mail Extensions (S/MIME) is a specification for securing electronic mail. S/MIME, which is based on the popular MIME standard, describes a protocol for adding cryptographic security services through MIME encapsulation of digitally signed and encrypted objects. These security services are authentication, nonrepudiation, message integrity, and message confidentiality.

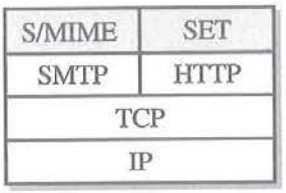


Figure 8-1
Application-layer protocols within TCP/IP

Although S/MIME is most widely known and used for securing e-mail, it can be used with any transport mechanism that transports MIME (such as HTTP). S/MIME can even be used in automated message transfer agents, which use cryptographic security services that do not require human intervention. The S/MIME specification even points out how to use its services to encrypt fax messages sent over the Internet.

The following section describes S/MIME along with the various MIME types and their uses. It explains how to create a MIME body that has been cryptographically enhanced according to *Cryptographic Message Syntax* (CMS), a formatting standard derived from PKCS #7. Finally, it defines and illustrates how cryptographic signature and encryption services are added to MIME data.

Overview

In the early 1980s, the *Internet Engineering Task Force* (IETF) developed Request for Comment (RFC) 822, which became the specification that defined the standard format of electronic mail messages. Along with RFC 821 (which defined the mail transfer protocol), RFC 822 was the foundation of the SMTP, which was designed to carry textual messages over the Internet.

MIME, also developed by the IETF, was designed to support nontextual data (such as graphics or video data) used in Internet messages. The MIME specification adds structured information to the message body that allows it to contain nontextual information. However, MIME does not provide any security services.

In 1995, RSA Data Security, Inc., led a consortium of industry vendors in the development of S/MIME. After work on the specification was under way, RSA passed it to the IETF for further development. S/MIMEv3 is the current version. Through continued development by the IETF S/MIME working group, the protocol has incorporated a number of enhancements.

S/MIME Functionality

S/MIMEv3 currently provides the following security enhancements to MIME content:

- **Enveloped data** This function supports confidentiality services by allowing any content type in a MIME message to be symmetrically encrypted. The symmetric key is then encrypted with one or more recipients' public keys. The encrypted data and corresponding encrypted symmetric key are then attached to the data structure, along with any necessary recipient identifiers and algorithm identifiers.
- **Signed data** This function provides data integrity services. A message digest is computed over the selected content (including any algorithm identifiers and optional public-key certificates), which is then encrypted using the signer's private key. The original content and its corresponding signature are then base-64 encoded (base-64 and other encoding methods are described later in "Transfer Encoding").
- **Clear-signed data** This function allows S/MIME to provide the same data integrity services as provided by the signed data function, while at the same time allowing readers that are not S/MIME-compliant to view the original data. Following the processes just described, a digital signature is computed over the selected content, but only this digital signature (and not the original data) is base-64 encoded.
- **Signed and enveloped data** This function supports both confidentiality and integrity services by allowing either the signing of encrypted data or the encrypting of signed data.

Cryptographic Algorithms

S/MIMEv3 implements support for several symmetric content-encryption algorithms. However, some S/MIME implementations still incorporate RC2 with a key size of 40 bits, and by today's standards, a 40-bit key is too weak. However, in most current S/MIMEv3 implementations, the user can choose from various content-encryption algorithms, such as DES, Triple DES, or RC2 with a key size greater than 40; see Chapter 2.

The specification does, however, spell out all algorithms to be used for security services within S/MIMEv3. Some of them are optional, and others are required. They are as follows:

- **Digest and hashing algorithms** These must support MD5 and SHA-1; however, SHA-1 should be used.
- **Digital signature algorithms** Both sending and receiving agents must support DSA and should also support RSA.
- **Key encryption algorithms** Sending and receiving agents must support Diffie-Hellman and should also support RSA encryption.
- **Data encryption (session key) algorithms** Sending agents should support RC2/40-bit key, RC2/128-bit key, and Triple DES. Receiving agents should support RC2/128 and Triple DES but must support RC2/40.

Which algorithm is best? It's a simple matter of looking at key length; the bigger the key, the greater the security. However, sending and receiving agents are not always at the same level. For instance, the sending agent may be attempting to encrypt something with RC2/128 for added security; however, the receiving agent may only have the ability to decrypt messages with RC2/40. For this reason, the S/MIME specification defines a process for deciding which algorithm is best when you're sending S/MIME messages.

The following are the specified rules that a sending agent should use in making its decision:

1. *Known capabilities.* If the sending agent has previously received a list of cryptographic capabilities of the recipient, the sender should choose the first (most preferred) capability listed to encrypt the outgoing message.
2. *Unknown capabilities but known use of encryption.* This rule applies when the sending agent has no idea of the encryption capabilities of the recipient but has received at least one previously encrypted message from that recipient. In this case, the sending agent should encrypt the outgoing message using that algorithm.
3. *Unknown capabilities and unknown version of S/MIME.* This rule applies when a sending agent has had no previous contact with the recipient and does not know its capabilities. The sending agent should use Triple DES because of its strength and because it is required by S/MIMEv3. However, if Triple DES is not used, the sending agent should use RC2/40.

S/MIME Messages

S/MIME messages are made up of the MIME bodies and CMS objects. The latter are derived from PKCS #7 data structures.

Before any cryptographic processing takes place, a MIME entity must be prepared. A MIME entity may be a subpart of a message or the whole message, including all its subparts. The latter type of MIME entity is made up only of the MIME headers and MIME body and does not include the RFC822 headers (To:, From:, and so on). This MIME entity is then converted to canonical form, and the appropriate transfer encoding is applied (both processes are discussed in the following sections).

After the MIME entity has been created and all proper encoding has taken place, the MIME entity is sent to security services, where the chosen security function is provided (enveloping, signing, or both). This process yields a CMS (or PKCS #7) object, which in turn is wrapped up in MIME and placed with the original message, according to the selected S/MIME content type.

Canonicalization

As stated in the preceding section, each MIME entity must be converted to a canonical form. This conversion allows the MIME entity to be uniquely and unambiguously represented in the environments where the signature is created and where the signature will be verified. This same process is performed for MIME entities that will be digitally enveloped as well as signed. Canonicalization provides a standard means by which data from various platforms can be exchanged.

Transfer Encoding

Whenever data is processed by digital equipment, it can be encoded and represented in a variety of ways, such as 7-bit, 8-bit, or binary. *Transfer encoding* ensures that data is represented properly for transfer across the Internet and ensures reliable delivery. One common method is base-64 encoding, which enables arbitrary binary data to be encoded so that it may pass through a variety of systems unchanged. For example, if 8-bit data is transferred and a 7-bit device (such as a mail gateway) receives it, there is a good chance that before it is forwarded to its final destination, it may be stripped of characters.

NOTE:

As you might expect, if a digitally signed message is altered or stripped of characters, it will be selected as invalid.

Enveloped-Only Data

The process of generating an encrypted MIME entity is called *digital enveloping* and is provided for by the *enveloped-data* content type. This content type consists of encrypted content of any type and encrypted content-encryption keys for one or more recipients. For each recipient, a digital envelope (made up of the encrypted content and associated encrypted content-encryption key) is created, ensuring confidentiality for the message while it is in transit. Figure 8-2 illustrates the S/MIME enveloped-data process.

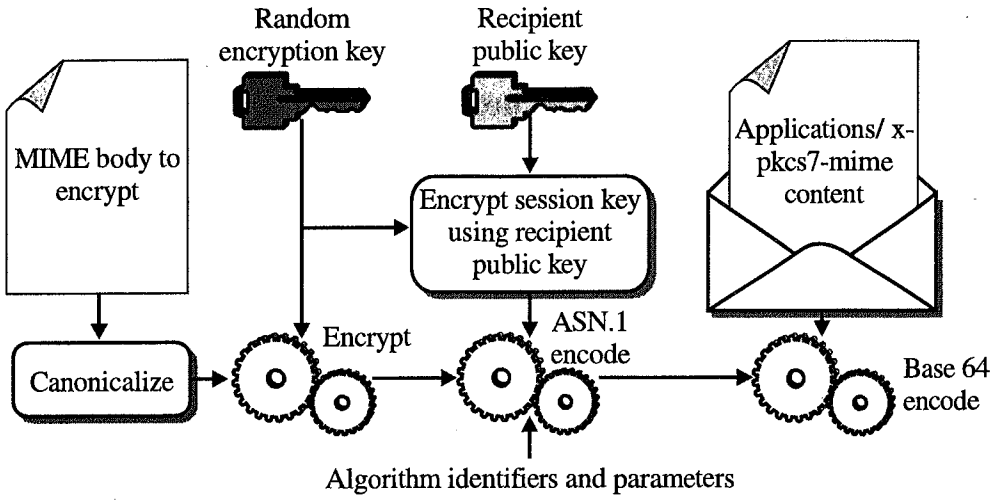


Figure 8-2
S/MIME enveloped-data process

To construct an enveloped-data content type, follow these steps:

1. For a chosen symmetric algorithm (that is, RC2, DES, and so on), generate a pseudo-random content-encryption key.
2. For each recipient, encrypt the content-encryption key. Which encryption to use depends on which key management system is used. The associated key management systems are as follows:

   RSA key transport The content-encryption key is encrypted in the recipient's public key.

Diffie-Hellman key agreement The recipient's public key and the sender's private key are used to generate a shared symmetric key, which is then used to encrypt the content-encryption key.
Known symmetric key The content-encryption key is encrypted using a previously distributed symmetric key.

3. For each recipient, create a block of data containing the recipient information. This information includes the encrypted content-encryption key and other recipient-specific information (such as version and algorithm identifiers).
4. Encrypt the message using the content-encryption key.
5. Prepend the recipient information to the encrypted content, and base 64-encode the result to produce the enveloped-data value.

When the digital envelope is received, the process is reversed to retrieve the original data. First, the enveloped data is stripped of its base-64 encoding. Then the appropriate content-encryption key is decrypted. Finally, the content-encryption key is used to decrypt the original content.

Signed-Only Data

The S/MIME specification defines two methods for signing messages:

- Application/pkcs7-mime with signed-data (usable only by S/MIME-compliant mailers)
- Multipart/signed, also known as clear signing (usable by all mailers)

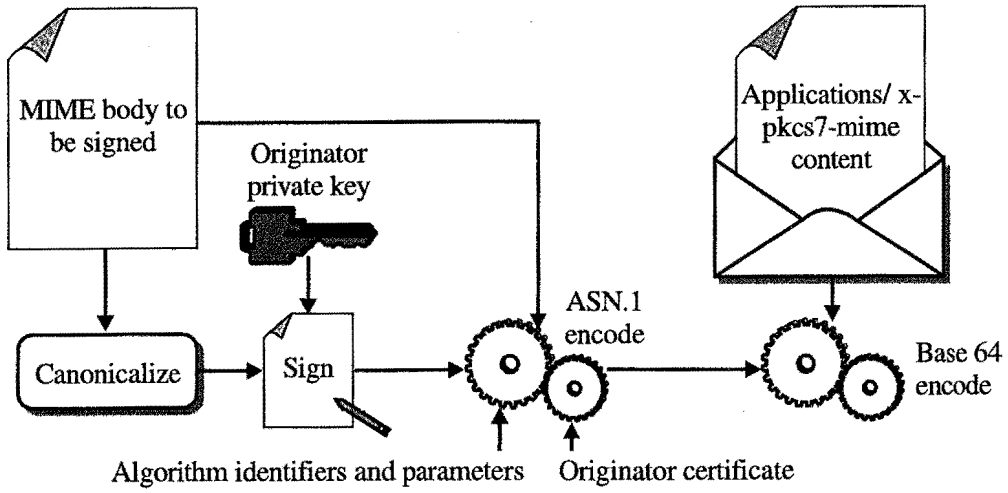
S/MIMEv3 doesn't mandate which method to use, but the specification mentions that the multipart/signed form is preferred for sent messages because of its readability by any mailer. The specification states that receiving agents should be able to handle both kinds.

Signed Data An S/MIME application/pkcs7-mime message with signed data may consist of any MIME content type, in which any number of signers in parallel can sign any type of content. Figure 8-3 illustrates S/MIME data signing.

The following steps apply to constructing a signed-data content type:

1. For each signer, select a message digest or hashing algorithm (MD5 or SHA-1).
2. Compute a message digest or hash value over all content to be signed.

Figure 8-3
S/MIME data signing



3. For each signer, digitally sign the message digest (that is, encrypt the digest using the signer's private key).
4. For each signer, create a signer information block containing the signature value and other signer-specific information (such as version and algorithm identifier).
5. Prepend the signed content with signer information (for all signers), and then base-64-encode it to produce the signed data value.

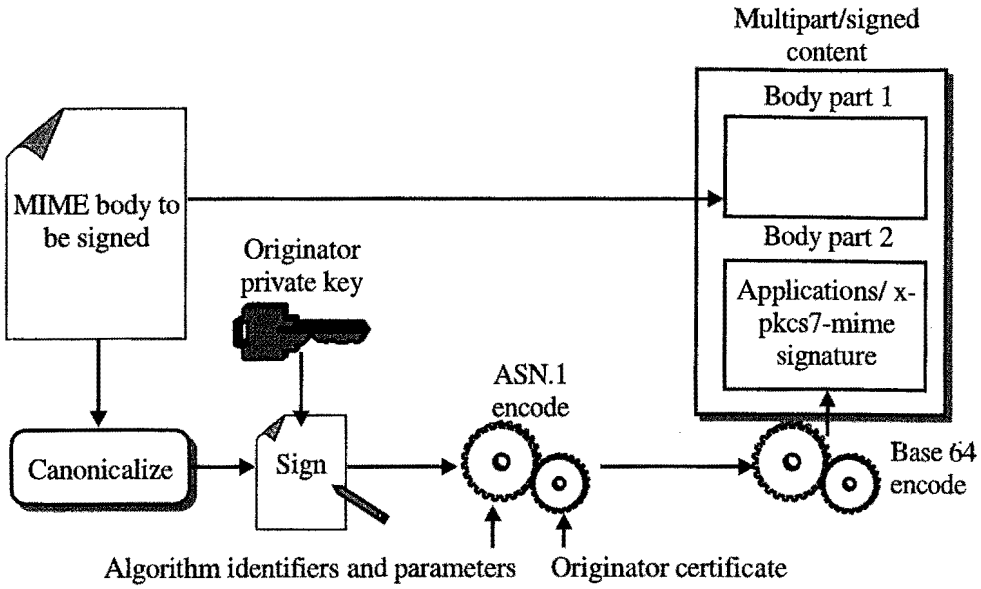
After it is received, the signed data content type is stripped of its base-64 encoding. Next, the signer's public key is used to decrypt and reveal the original message digest. Finally, the recipient independently computes the message digest and compares the result with that of the one that was just decrypted.

Clear-Signed Data It is possible that data you have digitally signed might be received by a recipient that is not S/MIME-compliant, rendering the original content unusable. To counter this problem, S/MIME uses an alternative structure, the multipart/signed MIME type.

The body of the multipart/signed MIME type is made up of two parts. The first part, which can be of any MIME content type, is left in the clear and placed in the final message. The contents of the second part are a special case of signed data, known as a *detached signature,* which omits the copy of the plaintext that may be contained within the signed data. Figure 8-4 illustrates the S/MIME clear-signed data process.

Figure 8-4

S/MIME clear-signed data process



Signing and Encrypting

S/MIME also supports both encryption and signing. To provide this service, you can nest enveloped-only and signed-only data. In other words, you either sign a message first or envelope the message first. The decision of which process to perform first is up to the implementer and the user.

NOTE:

The S/MIMEv3 specification (RFC2633) describes security risks involved with each technique (envelope first or signing first).

Registration Request

In addition to security functions, S/MIME defines a format for conveying a request to have a public-key certificate issued. A MIME content type, *application/x-pkcs10*, is used to request a certificate from a certification authority.

NOTE:
The specification does not mandate the use of any specific technique for requesting a certificate, whether it is through a certificate authority, a hardware token, or manual distribution. The specification does, however, mandate that every sending agent have a certificate.

Certificates-Only Messages

A *certificates-only* message is an application/pkcs7-mime and is prepared in much the same way as a signed-data message. This message, which is used to transport certificates to an S/MIME-compliant end entity, may be needed from time to time after a certification authority receives a certificate request. The certificates-only message can also be used for the transport of *certificate revocation lists* (CRLs).

Enhanced Security Services

Currently there are three optional enhanced security services that can be used to extend the current S/MIMEv3 security and certificate processing services.

- **Signed receipts** A signed receipt is an optional service that allows for proof of message delivery. The receipt provides the originator a means of demonstrating to a third party that the recipient not only received but also verified the signature of the original message (hopefully, this means that the recipient also read the message). Ultimately, the recipient signs the entire message and its corresponding signature for proof of receipt. Note that this service is used only for signed data.
- **Security labels** Security labels can be used in a couple of ways. The first and probably most easily recognizable approach is to describe the sensitivity of data. A ranked list of labels is used (confidential, secret, restricted, and so on). Another technique is to use the labels to control authorization and access, describing which kind of recipient should have access to the data (such as a patient's doctor, medical billing agents, and so on).
- **Secure mailing lists** When S/MIME provides its services, sending agents must create recipient-specific data structures for each

recipient. As the number of recipients grows for a given message, this processing can impair performance for messages sent out. Thus, *mail list agents* (MLAs) can take a single message and perform the recipient-specific encryption for every recipient.

Interoperability

Since the S/MIME standard first entered the public eye, a number of vendors have made efforts to incorporate it. However, a lack of interoperability is one pitfall that end users should take into account. For example, many vendors are still S/MIMEv2-compliant, whereas others have moved to S/MIMEv3 without supporting backward compatibility. Other problems include limits on the certificate processing available in various products.

To help promote product interoperability, the RSA Interoperability Test Center was established. This S/MIME test center allows vendors to perform interoperability testing on their products and to have the results published. The following Web address provides interoperability information as well as products that have been found to be S/MIME-compliant: http://www.rsasecurity.com/standards/smime/interop_center.html.

Secure Electronic Transaction (SET)

The Internet has made it easier than ever for consumers to shop, money to be transferred, and bills to be paid over the Internet at the press of a button. The price we pay for this ease of use, however, is increased opportunity for fraud. For example, Figure 8-5 illustrates how easy it is for those with very little character to fraudulently generate credit cards used for online payment, known in the industry as *payment cards*.

The *Secure Electronic Transaction* (SET) specification provides a framework for protecting payment cards used in Internet e-commerce transactions against fraud. SET protects payment cards by ensuring the confidentiality and integrity of the cardholder's data while at the same time providing a means of authentication of the card. The current version of the specification (SETv1) was initiated by MasterCard and Visa in February 1996 and was completed in May 1997.

SET is defined in three books. The first book, *Business Description,* describes the specification in business terms (that is, goals, participants,

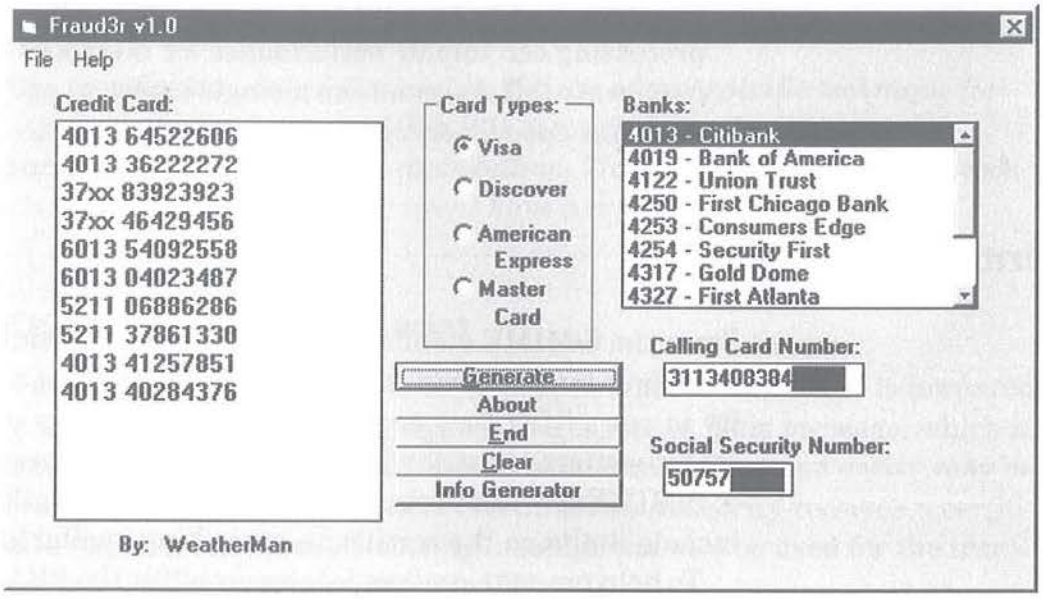


Figure 8-5

Generation of fraudulent credit cards

and overall architecture). The second book, *Programmer's Guide,* is a developer's guide, detailing the architecture, cryptography, and various messages used in SET. The third book, *Formal Protocol Definition,* provides a formal definition of the entire SET process. (All three books were published by Visa International and MasterCard on May 31, 1997.)

What follows is a high-level overview of the SET specification, outlining the business requirements, functions, and participants defined in the first book. We also cover SET certificates used and their management, describing the addition of SET-specific extensions. Finally, we look at the SET messages and transactions.

Business Requirements

The specification defines the business requirements of SET as follows:

- To provide confidentiality of payment information and enable confidentiality of the associated order information
- To ensure the integrity of all transmitted data
- To provide authentication that a cardholder is a legitimate user of a branded payment card account

- To provide authentication that a merchant can accept branded payment card transactions through its relationship with an acquiring financial institution
- To ensure the use of the best security practices and system design techniques to protect all legitimate parties in an electronic commerce transactions
- To create a protocol that neither depends on transport security mechanisms nor prevents their use
- To facilitate and encourage interoperability among software and network providers

SET Features

To meet its stated business requirements, SET defines the following necessary features:

- **Confidentiality of information** Confidentiality provides a secure channel for all payment and account information, preventing unauthorized disclosure. SET provides for confidentiality through the use of the DES algorithm.
- **Integrity of data** Data integrity ensures that the message content is not altered during transmission. This feature is provided through the use of digital signatures using the RSA algorithm.
- **Cardholder account authentication** Cardholder authentication provides merchants a means of verifying the cardholder as legitimate. Digital signatures and X.509v3 certificates are used to implement this function.
- **Merchant authentication** Merchant authentication gives cardholders a means of verifying that the merchant not only is legitimate but also has a relationship with a financial institution. Again, digital signatures and X.509v3 certificates are used to implement this service.
- **Interoperability** Interoperability allows the use of this specification in hardware and software from various manufacturers, allowing their use by cardholders or other participants.

SET Participants

Various participants use and interact with the SET specification. Figure 8-6 illustrates a simplified overview of the participants' interactions.

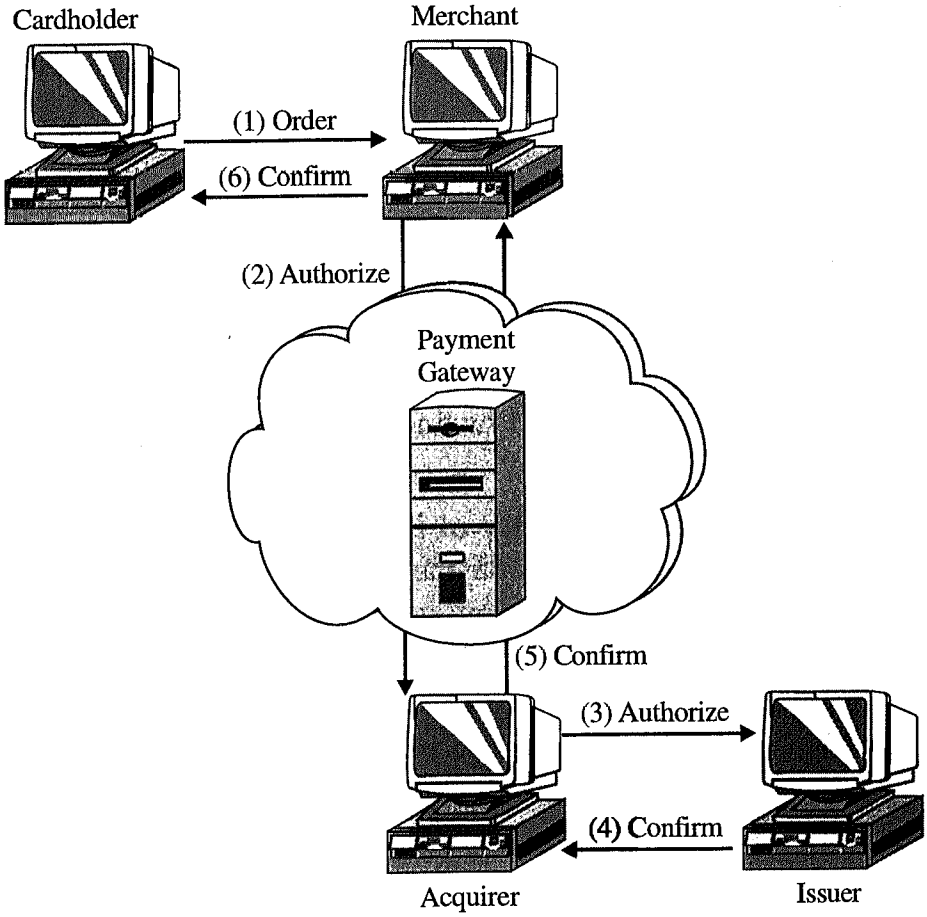


Figure 8-6
Interactions among SET participants

Following are these participants and their roles in the transactions governed by SET:

- **Issuer** The issuer is the bank or other financial institution that provides a branded payment card (such as a MasterCard or Visa credit card) to an individual. The card is provided after the individual establishes an account with the issuer. It is the issuer that is responsible for the repayment of debt, for all authorized transactions placed on the card.

- **Cardholder** The cardholder is the individual authorized to use the payment card. The SET protocol provides confidentiality services for the cardholder's transactions with merchants over the Internet.
- **Merchant** The merchant is any entity that provides goods and/or services to a cardholder for payment. Any merchant that accepts payment cards must have a relationship with an acquirer.
- **Acquirer** The acquirer is a financial institution that supports merchants by providing the service of processing payment cards. In other words, the acquirer pays the merchant, and the issuer repays the acquirer.
- **Payment gateway** The payment gateway is the entity that processes merchant payment messages (for example, payment instructions from cardholders). The acquirer or a designated third party can act as a payment gateway; however, the third party must interface with the acquirer at some point.

Dual Signatures

The SET protocol introduced dual signatures, a new concept in digital signatures. *Dual signatures* allow two pieces of data to be linked and sent to two different entities for processing. For example, within SET a cardholder is required to send an *order information* (OI) message to the merchant for processing; at the same time, a *payment instructions* (PI) message is required by the payment gateway. Figure 8-7 illustrates the dual signature generation process.

The dual signature process follows these steps:

1. A message digest is generated for both the OI and the PI.
2. The two message digests are concatenated (hashed) to produce a new block of data.
3. The new block of data is hashed again to provide a final message digest.
4. The final message digest is encrypted using the signer's private key, producing a digital signature.

A recipient of either message can check its authenticity by generating the message digest on its copy of the message, concatenating it with the message digest of the other message (as provided by the sender) and computing the message digest of the result. If the newly generated digest

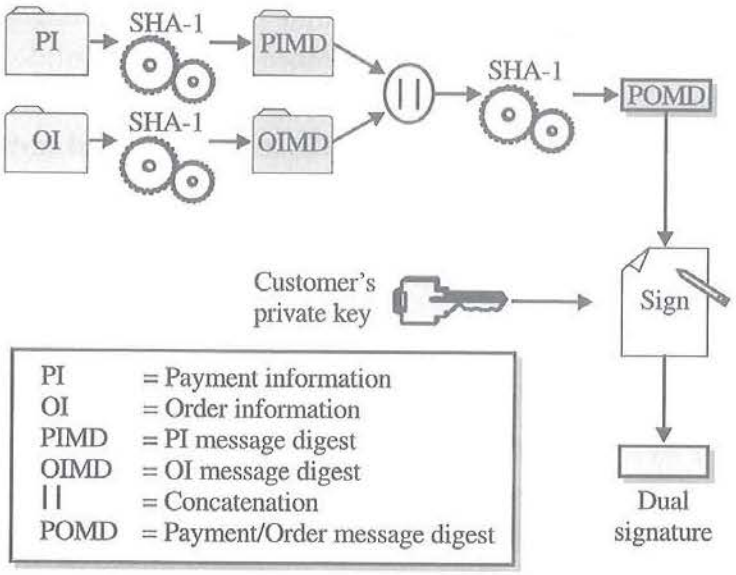


Figure 8-7
Generating dual signatures

matches the decrypted dual signature, the recipient can trust the authenticity of the message.

SET Certificates

The SET protocol provides authentication services for participants through the use of X.509v3, and has revocation provisions through the use of CRLv2 (both X.509v3 and CRLv2 are described in Chapter 6.). These certificates are application-specific; that is, SET has defined its own specific private extensions that are meaningful only to SET-compliant systems. SET contains the following predefined profiles for each type of certificate:

- **Cardholder certificates** function as electronic representations of payment cards. Because a financial institution digitally signs these certificates, they cannot be altered by a third party and can be generated only by the financial institution. A cardholder certificate does not contain the account number and expiration date. Instead, the account information and a secret value known only to the cardholder's software are encoded using a one-way hashing algorithm.
- **Merchant certificates** function as electronic substitutes for the payment card brand decal that appears in a store window; the decal

itself is a representation that the merchant has a relationship with a financial institution allowing it to accept the payment card brand. Because the merchant's financial institution digitally signs them, merchant certificates cannot be altered by a third party and can be generated only by a financial institution.

- **Payment gateway certificates** are obtained by acquirers or their processors for the systems that process authorization and capture messages. The gateway's encryption key, which the cardholder gets from this certificate, is used to protect the cardholder's account information. Payment gateway certificates are issued to the acquirer by the payment card brand organization.
- **Acquirer certificates** are required only in order to operate a certification authority that can accept and process certificate requests directly from merchants over public and private networks. Those acquirers that choose to have the payment card brand organization process certificate requests on their behalf do not require certificates because they are not processing SET messages. Acquirers receive their certificates from the payment card brand organization.
- **Issuer certificates** are required only in order to operate a certification authority that can accept and process certificate requests directly from cardholders over public and private networks. Those issuers that choose to have the payment card brand organization process certificate requests on their behalf do not require certificates because they are not processing SET messages. Issuers receive their certificates from the payment card brand organization.

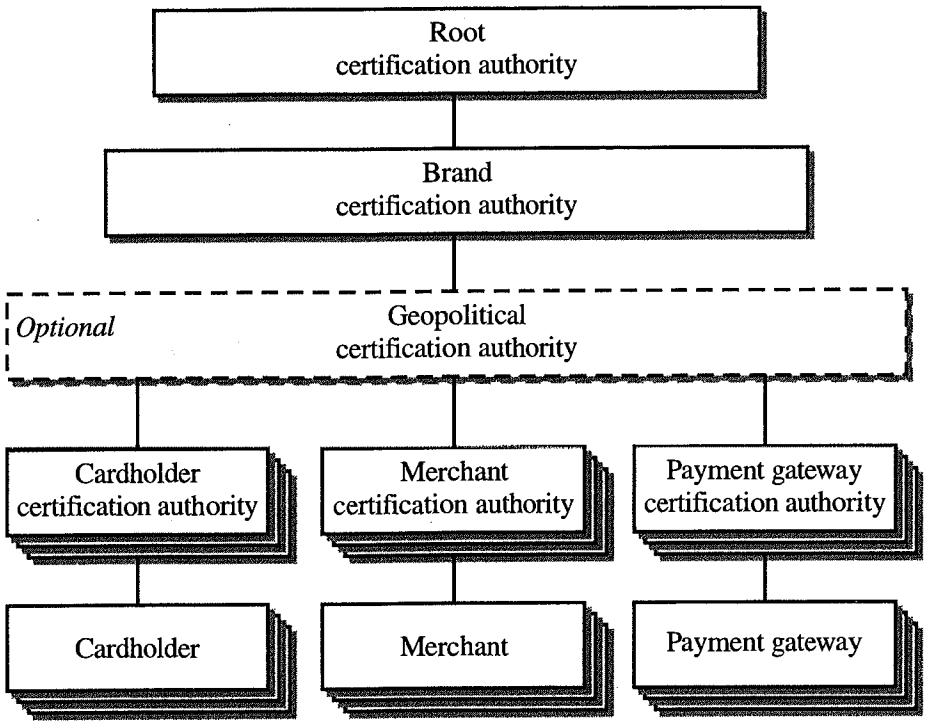
Certificate Management

The SET specification states that certificates must be managed through a strict certificate hierarchy, as shown in Figure 8-8 (certificate hierarchies are explained in Chapter 6).

In the case of SET, each certificate is linked to the signature certificate of the entity that digitally signed it. By following the trust tree to a known trusted party, a person can be assured that the certificate is valid. For example, a cardholder certificate is linked to the certificate of the issuer (or the brand organization on behalf of the issuer). The issuer's certificate is linked back to a root key through the brand organization's certificate. The public signature key of the root is known to all SET software and can be used to verify each of the certificates in turn.

Figure 8-8

SET certificate hierarchy



Payment Processing

To provide for secure payment processing over the Internet, the SET specification defines multiple transaction types, as shown in Table 8-1.

To illustrate how SET provides security of payment processing within e-commerce transactions, we next discuss each of the following transaction types in depth:

- Purchase request
- Payment authorization
- Payment capture

Purchase Request

The purchase request transaction is made up of four messages that are exchanged between the cardholder and the merchant:

1. *Initiate request.* When the cardholder has selected a purchase and decided which payment card to use, the cardholder is ready to initiate the request. To send SET messages to a merchant, the cardholder must have a copy of the merchant's and payment

Table 8-1

SET Transaction Types

Transaction	Description
Cardholder registration	Allows the cardholder to register with a CA.
Merchant registration	Allows a merchant to register with a CA.
Purchase request	Message from the cardholder containing order information (OI) and payment information (PI) and sent to the merchant and bank.
Payment authorization	Message between the merchant and payment gateway requesting payment authorization for a transaction.
Payment capture	Message from the merchant to the payment gateway requesting payment.
Certificate inquiry and status	A CA may send this message to either cardholders or merchants to state that more processing time is needed. *or* A cardholder or merchant may send this message to a CA to check the current status of a certificate request, or to receive the certificate if the request has been approved.
Purchase inquiry	Allows the cardholder to check the status of authorization, capture, or credit processing of an order after the purchase response has been received.
Authorization reversal	Allows a merchant to reverse an authorization entirely or in part.
Capture reversal	Allows a merchant to correct errors in previous capture requests, such as those caused by human error.
Credit	Allows a merchant to issue credit to a cardholder's account for various reasons (such as for returned or damaged goods).
Credit reversal	Allows a merchant to correct errors in a previous credit request.
Payment gateway certificate request	Allows a merchant to request a current copy of the payment gateway's certificates.
Batch administration	Message between merchant and payment gateway regarding merchant batches.
Error message	Indicates that a responder rejects a message because it fails tests of format or content verification.

gateway's key-exchange keys. The SET order process begins when the cardholder software (software that runs with your browser) requests a copy of the gateway's certificate. The message from the cardholder indicates which payment card brand will be used for the transaction.

2. *Initiate response.* When the merchant receives an initiate request message, a unique transaction identifier is assigned to the message. The merchant then generates an initiate response message containing its certificates and that of the payment gateway. This information is then digitally signed with the merchant's private key and transmitted to the cardholder.
3. *Purchase request.* Upon receipt of the initiate response message, the cardholder software verifies the certificates of both the merchant and gateway. Next, the cardholder software creates a dual signature using the OI and PI. Finally, the cardholder software generates a purchase request message containing a dual-signed OI and a dual-signed PI that is digitally enveloped to the payment gateway. The entire purchase request is then sent to the merchant.
4. *Purchase response.* When the merchant software receives the purchase request message, it verifies the cardholder's certificate contained within the message, as well as the dual-signed OI. The merchant software then begins processing the OI and attempts to gain authorization from the payment gateway by forwarding the PI. Finally, the merchant generates a purchase response message, which states that the merchant received the cardholder's request.

Upon receipt of the purchase response from the merchant, the cardholder software verifies the merchant certificate as well as the digital signature of the message contents. At this point, the cardholder software takes some action based on the message, such as displaying a message to the cardholder or updating a database with the status of the order.

Payment Authorization

During the processing of an order from a cardholder, the merchant attempts to authorize the transaction by initiating a two-way message exchange between the merchant and the payment gateway. First, an authorization request is sent from the merchant to the payment gateway; then an authorization response is received from the merchant by the payment gateway. The request and response are described as follows:

1. *Authorization request.* The merchant software generates and digitally signs an authorization request, which includes the amount to be authorized, the transaction identifier from the OI, and other information about the transaction. This information is then digitally enveloped using the payment gateway's public key. The authorization request and the cardholder PI (which is still digitally enveloped to the payment gateway) are transmitted to the payment gateway.
2. *Authorization response.* When the authorization request is received, the payment gateway decrypts and verifies the contents of the message (that is, certificates and PI). If everything is valid, the payment gateway generates an authorization response message, which is then digitally enveloped with the merchant's public key and transmitted back to the merchant.

Upon receipt of the authorization response message from the payment gateway, the merchant decrypts the digital envelope and verifies the data within. If the purchase is authorized, the merchant then completes processing of the cardholder's order by shipping the goods or performing the services indicated in the order.

Payment Capture

When the order-processing portion is completed with the cardholder, the merchant then requests payment from the payment gateway. Payment capture is accomplished by the exchange of two messages: the capture request and the capture response. This process is described as follows:

1. *Capture request.* The merchant software generates the capture request, which includes the final amount of the transaction, the transaction identifier, and other information about the transaction. This message is then digitally enveloped using the payment gateway's public key and transmitted to the payment gateway.
2. *Capture response.* The capture response is generated after the capture request is received and its contents verified. The capture response includes information pertaining to the payment for the transaction requested. This response is then digitally enveloped using the merchant's public key and is transmitted back to the merchant.

Upon receipt of the capture response from the payment gateway, the merchant software decrypts the digital envelope, verifying the signature and message data.

NOTE:

The merchant software stores the capture response and uses it for reconciliation with payment received from the acquirer.

Summary

Security protocols located at the application layer work slightly differently from those that operate on the IP (network) and TCP (transport) layers. Whereas IPSec (see Chapter 7) is used to provide security for all data being transferred across an IP network, S/MIME and SET are used solely to provide security for certain applications.

In 1995, a consortium of application and security vendors, led by RSA Data Security, Inc., designed the S/MIME protocol. Since then, the IETF S/MIME working group has taken control of S/MIME to continue its growth. S/MIME provides security not only for e-mail but also for any data that is transferred via the MIME protocol. Since its creation, S/MIME has continued to grow and improve its security services, adding support for mailing lists, signing receipts, and security labels.

SET is an open specification that provides a framework for protecting payment cards that are used in e-commerce transactions. Initiated by Visa and MasterCard in 1996, SET was completed in 1997, with the help of various other application developers and security vendors. The specification is described in three books totaling more than 900 pages.

Note that this chapter and Chapter 7 discuss only four selected protocols. Numerous others are available today, each of them supporting a specific security task.

Real-World Examples

Both S/MIME and SET have been incorporated in various applications. For secure e-mail, many companies and individuals have chosen to use S/MIME instead of a proprietary system such as PGP. In fact, many users have S/MIME-enabled mailers that they have not taken advantage of. S/MIME is incorporated in Microsoft's Outlook and Outlook Express e-mail applications as well as Netscape's Messenger software.

SET has also gained widespread use. Many of the vendors that visitors shop with daily across the Internet are SET-enabled. Currently, the merchants worldwide who use SET number in the hundreds. SET products are available not only for consumers but also for merchants, payment gateways, and SET certificate authorities. For a list of current SET-enabled products as well as the merchants that use them, visit http://www.setco.org/.

For both of these protocols, many security vendors also provide cryptographic APIs (application programming interfaces, or toolkits), which developers can use to produce secured applications. RSA Security, Inc., is one such company.

CHAPTER 9

Hardware Solutions: Overcoming Software Limitations

The performance of cryptosystems varies, and some of them come with a significant computational expense to computer systems. One way to address this problem is to apply cryptographic hardware. Cryptographic accelerators, for example, offer performance enhancements (as well as possible pitfalls). Cryptographic hardware, including various kinds of tokens, also plays a role in authentication, as does the old technology of biometrics, now being applied in new ways.

Cryptographic Accelerators

Cryptographic accelerators provide a means of performing the computationally expensive workload that usually accompanies various algorithms and protocols. Cryptographic accelerators work like math coprocessors: They implement in hardware a set of functions usually handled by software. Encoding these functions in silicon allows hardware to perform these tasks much faster.

Cryptographic accelerators provide usefulness on two fronts. First and most noticeable is increased speed, which is particularly important to

e-commerce companies that interact with a considerable number of customers daily. The second benefit is a spin-off of the first one: By reducing the workload on the system's CPU, accelerators allow the system to be used more efficiently for other tasks. Figure 9-1 shows a typical *Secure Socket Layer* (SSL) accelerator card.

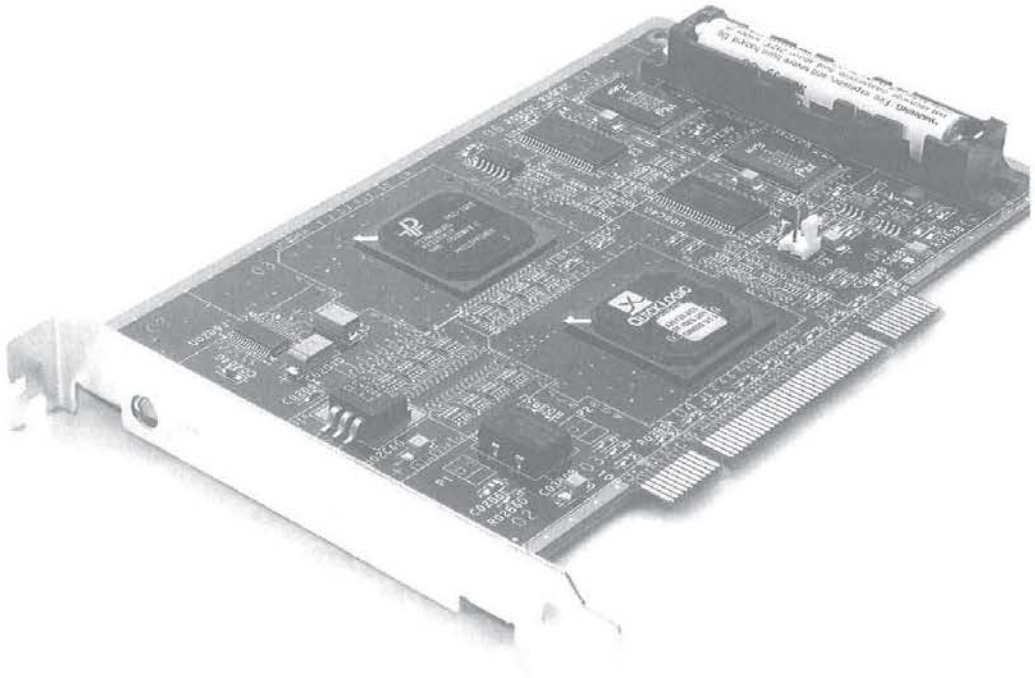


Figure 9-1
A typical SSL accelerator card

Another reason for the popularity of cryptographic accelerators is the certifications associated with them. NIST, for example, has certified many of them. The certification of each device depends on the safeguards that were implemented in it during manufacture.

NOTE:

Cryptographic accelerators often serve to slow down cryptographic operations because accelerators are I/O-bound. For example, a Web server that has farmed out private-key operations to a cryptographic accelerator often performs slower in SSL handshakes when the load is high. The reason for this is simple. I/O-bound operations are an order of magnitude slower than CPU-bound operations because getting the data to the hardware bus consumes an enormous amount of operating system and context-switching resources. An operating system with poor multitasking capabilities will likely be brought to its knees if it has to deal with a high number of SSL handshakes farmed out to an accelerator. Each thread must block and wait, and the CPU must manage all the blocked threads. This leads to a great deal of thread thrashing and, simply put, kills performance. For this reason, installing a cryptographic accelerator does not necessarily give you an across-the-board increase in speed. Where *and* how *the accelerator is applied are of prime importance.*

Authentication Tokens

In the realm of computer security, another important set of hardware devices is authentication tokens. Authentication tokens provide a means of authenticating and identifying an end user. Instead of memorizing passwords, end users protect their identity using a physical object that is unique to each user. An everyday analogy is the use of a driver's license to prove a person's identity.

Many tokens are designed for use with automated authentication systems. To verify the identity of the token's owner, the host system performs its authentication protocol using information encoded on the token. Because the uniqueness of the information is responsible for proving the identity of its bearer, the information must be protected against duplication or theft. Advanced tokens usually contain a microprocessor and semiconductor memory, and they support sophisticated authentication protocols that provide a high level of security.

In theory, authentication tokens enable the use of *single sign-on* (SSO) systems. As the name implies, SSO systems allow users to use an authentication token to sign on once to all applications they require access to. At the moment, true SSO is more or less a theoretical concept. In reality, even systems that use authentication tokens may have reduced sign-on capabilities.

Token Form Factors

Authentication tokens come in a variety of physical forms. The size, shape, and materials from which a token is manufactured are referred to collectively as the token's *form factor*. These parameters affect the durability, portability, security, and convenience of a given type of token. For example, some tokens have electrical contacts mounted on the outer surface of the token's casing. The electrical contacts are connected to an integrated circuit embedded in the token. When an electrostatic discharge of sufficient potential is applied to the contacts, the integrated circuit may be damaged. Because the human body can accumulate a significant static charge in dry weather, care must be taken in the design of such tokens to minimize the risk of damage from static discharges. To compensate for this, some types of tokens have contacts that are recessed in a conductive plastic casing. This type of token is less susceptible to damage from stray static discharges because the casing absorbs the charge before it reaches the contacts.

A token's form factor involves trade-offs that must be evaluated for a specific application. Tokens that have recessed contacts usually require a thicker casing than those that have surface-mounted contacts, and that can make it harder to carry the token in a pocket. Customers can sometimes select from a number of different form factors with the same functionality, making it possible to choose the form factor that is best suited to a particular application. Figure 9-2 shows three form factors.

Noncontact Tokens

Noncontact tokens, as their name implies, require no electrical or physical contact with a token reader device. Instead, noncontact tokens usually operate by transmitting data to and receiving data from a terminal, or they require that the user enter data that is then generated by the token.

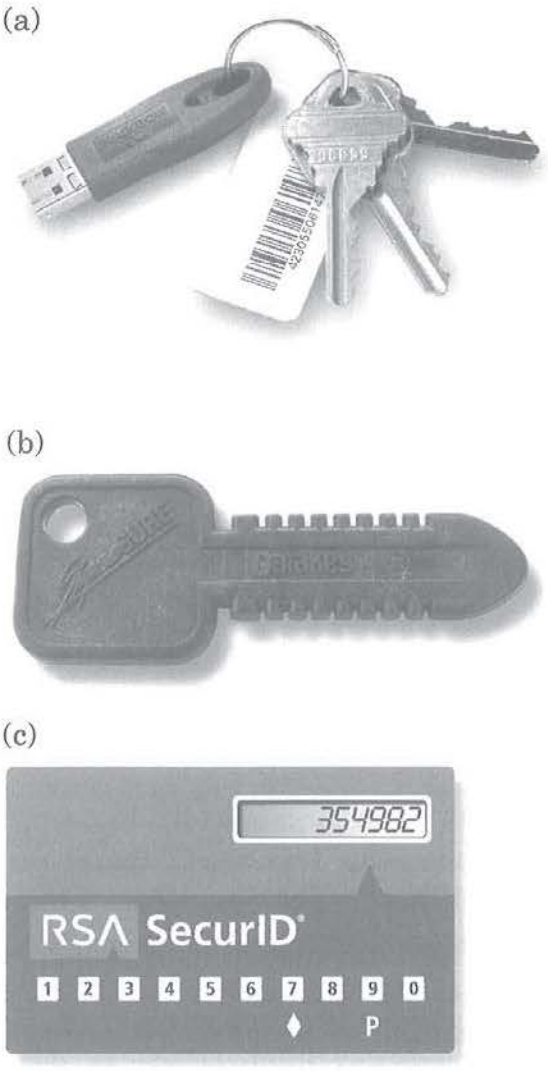


Figure 9-2
Cryptographic tokens from (a) Rainbow Technologies, (b) Datakey, and (c) RSA Security

Noncontact tokens include proximity cards, one-time password generators, and handheld challenge/response calculators.

Proximity Cards

Proximity cards are noncontact tokens that use radio frequency signals to authenticate users. Proximity cards contain micro-miniature electronic tuned circuits, a switching mechanism, and a power source. These cards transmit a coded signal either when they come within a certain range of a proximity reader or when someone activates them manually. Some proximity devices are also designed to transmit continually. A user merely

holds a uniquely coded proximity token or card within a given distance of a proximity reader, and the system reads the data within it. Figure 9-3 shows the XyLoc proximity card and reader from Ensure Technologies.

Figure 9-3

XyLoc proximity card and reader

NOTE:

Theoretically, authentication data (a coded signal in this case) is susceptible to replay attacks. That is, an outsider could conceivably record the signal being transmitted and replay it at a later time to gain access.

One-Time Password Generators

One-time password generators have proven to be one of the most successful types of authentication tokens to date. RSA Security, Inc., has proven this fact through its sales of the ACE/Server and SecurID products. The system has proven to be portable and to provide a very high level of security. Figure 9-4 shows a SecurID token in one of its (a) original form factors and (b) running on the Palm OS.

RSA's solution is made up of two components, which work in concert with each other. The ACE/Server is a back-end server application that houses a user's seed record. In turn, this seed is used by the ACE/Server application to produce a random six-digit numeric code on a configurable

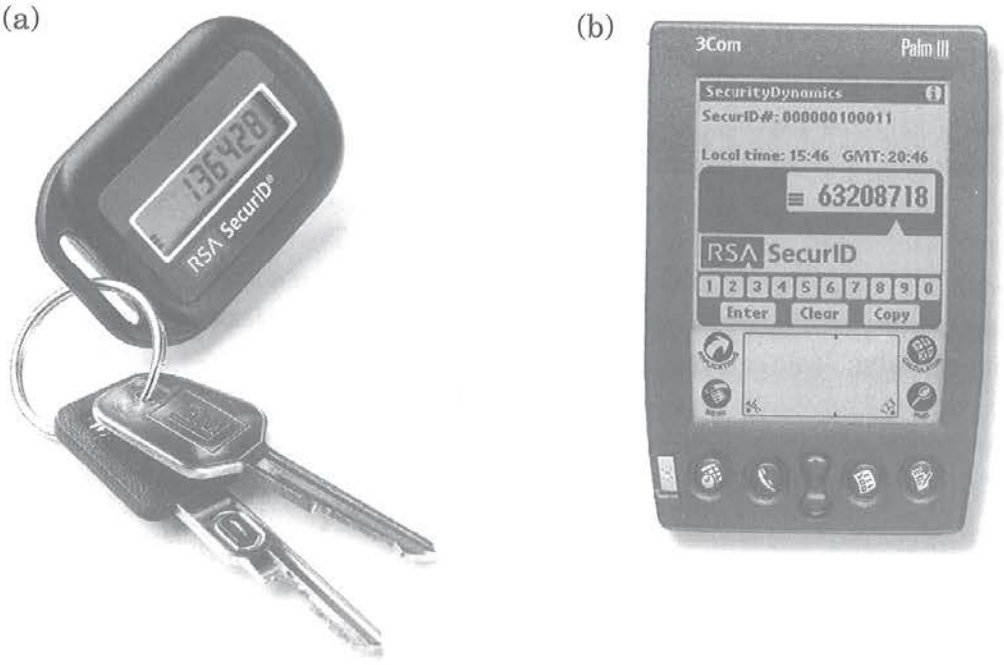


Figure 9-4
(a) SecurID token; (b) SecurID on Palm OS

time scale (for example, every 60 seconds a new six-digit numeric code is produced). The second component, the SecurID token, is also aware of the user's seed record. Like the ACE/Server, the SecurID produces a random numeric code. Figure 9-5 illustrates the user interaction with one-time passwords for authentication.

When users log in, they enter a four-digit PIN (known only to them) as well as the six-digit random code displayed by their token at that

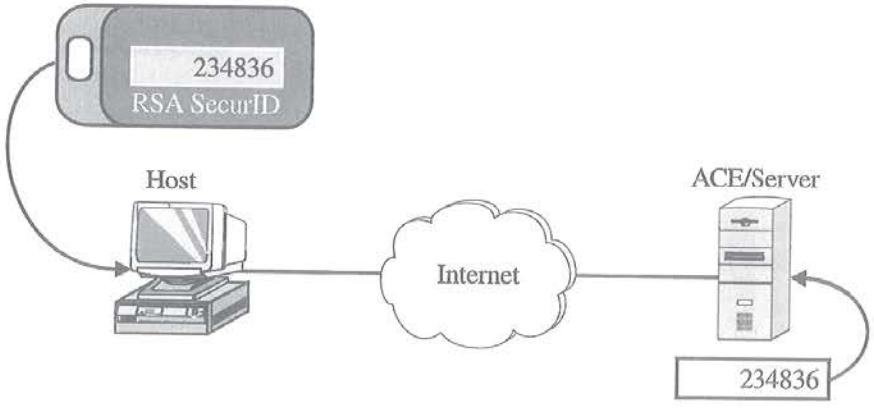


Figure 9-5
Authentication via a one-time password generator

moment. In this way, the system can authenticate the user's entry against the entry in the back-end server.

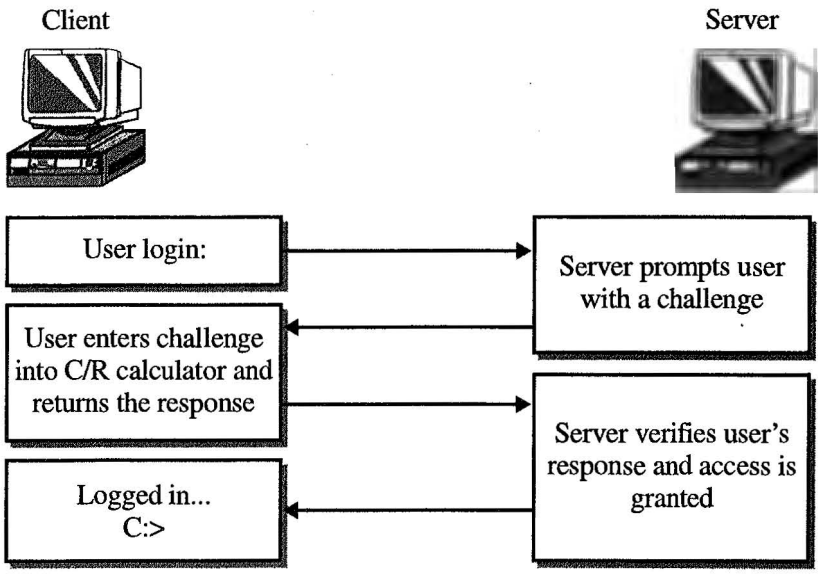
Challenge/Response Calculators

Challenge/response calculators work on a premise similar to that of one-time password generators. Through the use of a back-end server component and a handheld device, an initial seed record is synchronized. In the case of challenge/response calculators, however, there is slightly more user intervention.

As users log in, they are prompted with a random challenge from the host system. The users must then enter the displayed challenge into their calculator, which performs a cryptographic operation on the challenge password and displays the result. In turn, users enter this result (the response) into the host system to gain access. Figure 9-6 illustrates the common component setup and user intervention involved with challenge/response calculators.

Figure 9-6

User intervention in challenge/response calculators



NOTE:

Challenge/response calculators tend to be protected by a PIN that the user must enter before the challenge/response sequence.

Contact Tokens

To transfer data, most tokens must make physical contact with the reader device. For example, magnetic stripe tokens (the kind used in automated teller machines) are inserted into a reader so that the magnetic stripe makes contact with an electromagnetic sensing device. Most integrated circuit tokens require an interface in which electrical contacts located on the token physically touch matching contacts on the reader to supply such functions as power, ground, and data signals. The physical arrangement and functional definition of these contacts have an impact on the interoperability of tokens and reader devices because these devices cannot communicate unless the contacts are defined in the same way.

Smart Cards

A *smart card,* an intelligent token, is a credit card-sized plastic card that contains an embedded integrated circuit chip. It provides not only memory capacity but also computational capability. The self-containment of smart cards makes them resistant to attack because they do not depend on potentially vulnerable external resources. Because of this characteristic, smart cards are often used in applications that require strong security protection and authentication.

For example, a smart card can act as an identification card to prove the identity of the cardholder. It also can be used as a medical card that stores the cardholder's medical history. Furthermore, a smart card can be used as a credit or debit bankcard and used for offline transactions. In all these applications, the card stores sensitive data, such as biometrics information of the card owner, personal medical history, and cryptographic keys for authentication. Figure 9-7 shows a Datakey smart card and RSA smart card.

(a) (b)

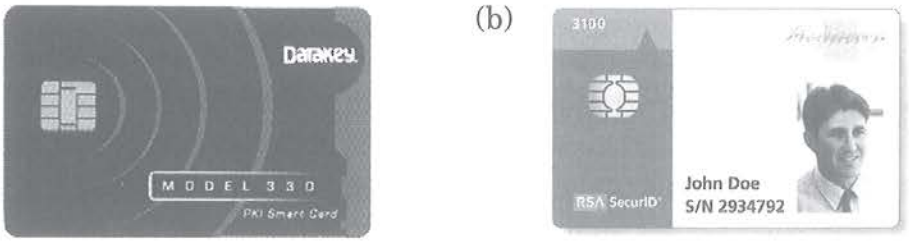


Figure 9-7
(a) Datakey smart card; (b) RSA smart card

Smart Card Standards

Smart card standards govern the physical properties and communication characteristics of the embedded chip. ISO 7816 is the international standard for smart cards. The standard itself is made up of six parts, each describing everything from electrical properties to card dimensions. The following is a description of each part of the ISO 7816 standard:

- **ISO 7816-1** Defines the physical dimensions of contact smart cards and the placement of chips, magnetic stripes, and any embossing on the cards. It also describes the required resistance to static electricity.
- **ISO 7816-2** Defines the location, purpose, and electrical characteristics of the smart card's contacts.
- **ISO 7816-3** Describes electronic signals and transmission protocols, defining the voltage and current requirements for the electrical contacts defined in ISO 7816-2.
- **ISO 7816-4** Across all industries, defines a set of commands to provide access, security, and transmission of card data (that is, the card reads and writes to its memory).
- **ISO 7816-5** Defines *Application Identifiers* (AIDs), which are used to identify a specific application.
- **ISO 7816-6** Describes encoding rules for data needed in many applications.

Currently *Europay International, MasterCard International*, and *Visa International* (EMV) are cooperatively developing specifications to facilitate the use of smart cards for payments worldwide. These specifications build upon the ISO 7816 standards that have been developed for smart cards that use electrical contacts.

Yet another standard, which has helped to ensure interoperability, is public-key cryptography standard PKCS #11. PKCS #11 provides functional specification for personal cryptographic tokens.

Types of Smart Cards

A variety of smart cards are available, each defined according to the type of chip it uses. These chips range in their processing power, flexibility,

memory, and cost. The two primary categories of smart cards—memory cards and microprocessor cards—are described in the following sections.

Memory Cards

Memory cards have no sophisticated processing power and cannot manage files dynamically. All memory cards communicate with readers through synchronous protocols. There are three primary types of memory cards:

- **Standard memory cards** These cards are used solely to store data and have no data processing capabilities. These cards are the least expensive per bit of user memory. They should be regarded as floppy disks of varying sizes without the lock mechanism. Memory cards cannot identify themselves to the reader, so the host system must recognize the type of card that is being inserted into a reader.
- **Protected/segmented memory cards** These cards have built-in logic to control access to memory. Sometimes referred to as *intelligent memory* cards, these devices can be set to write-protect some or all of the memory array. Some of these cards can be configured to restrict access to both reading and writing, usually through a password or system key. Segmented memory cards can be divided into logical sections for planned multifunctionality.
- **Stored value memory cards** These cards are designed to store values or tokens and are either disposable or rechargeable. Most cards of this type incorporate permanent security measures at the point of manufacture. These measures can include password keys and logic that are hard-coded into the chip. The memory arrays on these devices are set up as decrements, or counters, and little or no memory is left for any other function. When all the memory units are used, the card becomes useless and is thrown away or recharged.

CPU/MPU Microprocessor Multifunction Cards

These cards have on-card dynamic data processing capabilities. Multifunction smart cards allocate card memory into independent sections assigned to specific functions or applications. Embedded in the card is a microprocessor or microcontroller chip that manages this memory allocation and file access. This type of chip is similar to those found inside personal computers; when implanted in a smart card, the chip manages data in organized file structures via a *card operating system* (COS). Unlike

other operating systems, this software controls access to the on-card user memory. As a result, various functions and applications can reside on the card. This means that businesses can use these cards to distribute and maintain a range of products.

These cards have sufficient space to house digital credentials (that is, public and private key-pairs). Further, through the use of the use of the on-card microprocessor chip, many of the needed cryptographic functions can be provided. Some cards can even house multiple digital credential pairs.

Readers and Terminals

Smart cards can be plugged in to a wide variety of hardware devices. The industry defines the term *reader* as a unit that interfaces with a PC for the majority of its processing requirements. In contrast, a *terminal* is a self-contained processing device.

Terminals as well as readers can read and write to smart cards. Readers come in many form factors and offer a wide variety of capabilities. The easiest way to describe a reader is according to the method of its interface to a PC. Smart card readers are available that interface to RS232 serial ports, *Universal Serial Bus* (USB) ports, PCMCIA slots, floppy disk slots, parallel ports, IRDA (*infrared data*) ports and keyboards, and keyboard wedge readers. Another way to distinguish reader types is according to onboard intelligence and capabilities. Extensive price and performance differences exist between an industrial-strength intelligent reader that supports a wide variety of card protocols and a home-style Windows based-card reader that works only with microprocessor cards and performs all the data processing in the PC.

The options available in terminals are equally numerous. Most units have their own operating systems and development tools. They typically support other functions such as magnetic stripe reading, modem functions, and transaction printing.

The Pros and Cons of Smart Cards

There is sufficient evidence in the computer industry that smart cards greatly improve the convenience and security of any transaction. They provide tamperproof storage of user and account identity. They protect against a full range of security threats, from careless storage of user pass-

words to sophisticated system hacks. But smart cards, like other authentication systems, are vulnerable to various attacks.

Moreover, a major drawback of smart card technology is price. The cost is considerably higher than that of software-based access control (such as passwords), creating a barrier to widespread distribution of smart card technology. As more units are sold, however, we should begin to see prices fall, making smart cards and their associated hardware more affordable.

JavaCards

A JavaCard is a typical smart card: It conforms to all smart card standards and thus requires no change to existing smart card-aware applications. However, a JavaCard has a twist that makes it unique: A *Java Virtual Machine* (JVM) is implemented in its *read-only memory* (ROM) mask. The JVM controls access to all the smart card resources, such as memory and I/O, and thus essentially serves as the smart card's operating system. The JVM executes a Java bytecode subset on the smart card, ultimately allowing the functions normally performed off-card to be performed on-card in the form of trusted *loyalty applications.* For example, instead of using the card to simply store a private key, you can now use that private key to perform a digital signature.

The advantages of this approach are obvious. Instead of programming the card's code in hardware-specific assembly language code, you can develop new applications in portable Java. Moreover, applications can be securely loaded to the card post-issuance—after it's been issued to the customer. In this way, vendors can enhance JavaCards with new functions over time. For example, bankcards that initially give customers secure Internet access to their bank accounts might be upgraded to include e-cash, frequent flier miles, and e-mail certificates.

History and Standards

Schlumberger, a leading smart card manufacturer, provided one of the first working prototypes of a Java-based card in 1996. The original implementation was made up of a smart card that housed a lightweight Java bytecode interpreter. As work continued in this field, SUN Microsystems

issued the first JavaCard specification in October 1996. This specification was based on Schlumberger's experience.

It was not until February 1997 that the concept of a JavaCard finally took off, at which time Schlumberger and other smart card manufacturers formed the JavaCard Forum. By the end of 1997, the JavaCard Forum had released a new specification, JavaCard 2.0. This specification answered many of the shortcomings of the original specification and included many new concepts.

Another standard, which is of importance to JavaCards as well as to smart cards, is the *OpenCard Framework* (OCF). OCF, which was created by the OpenCard Consortium, is made up of many of the leading smart card and JavaCard manufacturers, as well as many application developers, such as Dallas Semiconductors, Gemplus, IBM Corp., Visa International, SUN Microsystems, and others.

OCF, similar to the JavaCard Forum, has been the driving force for the development Java-based systems. Unlike the JavaCard Forum, which provides development specifications for applications to be run on-card, OCF provides the development specifications for applications to be run in computers and terminals.

NOTE:

The application specifications provided by OCF are for use by systems that will communicate not only with JavaCards, but also with any smart card that follows the PKCS #11 standard.

JavaCard Operations

A JavaCard operates like a typical smart card. When the smart card reader sends a command, the JavaCard processes it and returns an answer. To maintain compatibility with existing applications for smart cards, a single JavaCard can process only one command at a time. Figure 9-8 illustrates the JavaCard components.

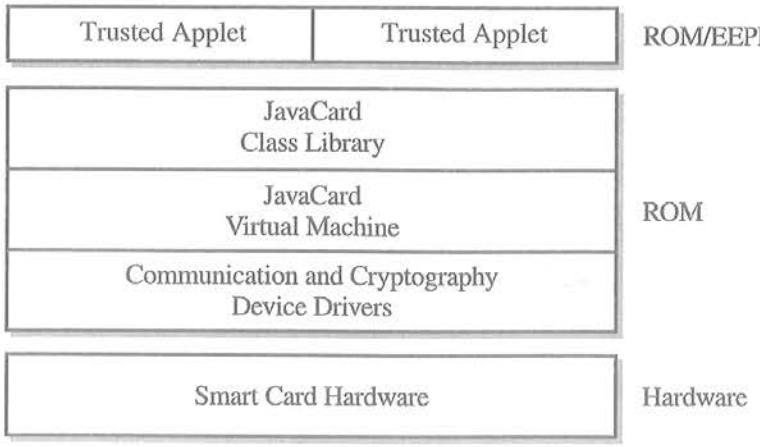


Figure 9-8
JavaCard components

Other Java Tokens

Another great advancement that has taken off because of JavaCard technology is the advent of other kinds of Java tokens, including Java rings. Java rings offer the most personal of tokens: jewelry. The ring is a steel casing that houses an 8-bit microprocessor called Crypto iButton. This microprocessor is similar to one you might find on smart card. It has its own real-time clock and a high-speed math accelerator to perform 1,024-bit public-key operations. Conceivably, it can hold additional information (such as a passport, driver's license, or medical data). The Crypto iButton microprocessor is not specific to Java rings and can be found in a number of other form factors, as shown in Figure 9-9.

Figure 9-9
Crypto iButton form factors: (a) wristwatch; (b) dogtag-type token; (c) Java ring

Biometrics

Biometrics is the science of measuring a characteristic of the human body; in its commercial application, such measurements are used to verify the claimed identity of an individual. Physical characteristics such as fingerprints, retinas and irises, palm prints, facial structure, and voice are some of the many methods being researched. Because these characteristics are relatively unique to each individual, biometrics provides an excellent means of authentication. As explained in the following sections, this technology is particularly useful for authentication when applied to commerce over the Internet.

Biometric systems are believed to provide a higher level of security than other forms of authentication, such as the use of passwords or PINs. One reason is that a biometric trait cannot be lost, stolen, or duplicated, at least not as easily as a password or PIN. Second, the use of biometrics provides nontransferable authentication. Simply stated, all other types of authentication, such as a key, are transferable. You can give someone your private key, but not your eyeball or finger (we hope).

Biometric Systems Overview

The various biometric recognition mechanisms typically operate in two modes: enrollment and verification. In the enrollment process, the user's biological feature (physical characteristic or personal trait) is acquired and stored for later use. This stored characteristic, commonly known as a *template,* is usually placed in a back-end database for later retrieval. The verification process is as you might expect. The user's characteristic is measured and compared against the stored template. The following sections describe these processes in greater detail.

Enrollment

For initial use of the biometric, each user must be enrolled by a system administrator, who verifies that each individual being enrolled is an authorized user. The biological feature is acquired by a hardware device, known as a *sensor,* which typically resides at the front end of the biometric authentication mechanism. When a physical feature is presented to the sensor, the sensor produces a signal that is modulated in response to variations in the physical quantity being measured. If, for example, the

sensor is a microphone used to capture a voice pattern, the microphone produces a signal whose amplitude (voltage or current) varies with time in response to the varying frequencies in a spoken phrase.

Because the signals produced by most biometric sensors are analog, they must be converted into digital form so that they can be processed by computer. An analog-to-digital converter is therefore the next stage in most systems. Analog-to-digital converters take an analog input signal and produce a digital output stream, a numeric representation of the original analog signal. Rather than use raw data from the sensor, biometric systems often process the data to extract only the information relevant to authentication. Further processing may be used to enhance differences and compress data. When the digital representation has been processed to the desired point, it is stored. Most biometric devices take multiple samples during enrollment to account for degrees of variance in the measurement. Figure 9-10 illustrates a typical enrollment process.

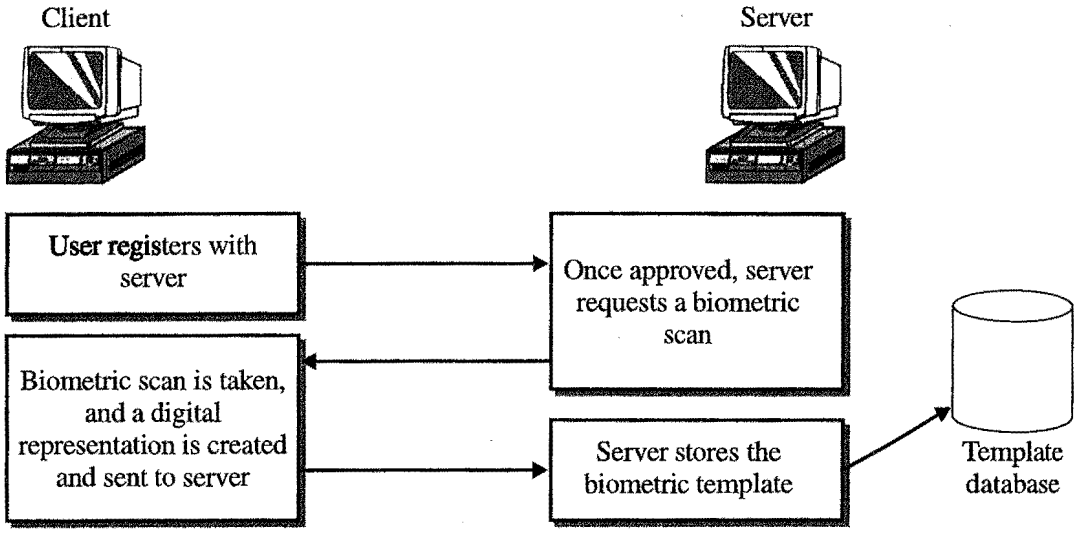


Figure 9-10
Enrollment process

Verification

After users are enrolled, their biometrics are used to verify their identity. To authenticate someone, his or her biological feature is acquired from the sensor and converted to a digital representation, called a *live scan.* Then the live scan is compared to the stored biometric template. Typically, the live scan does not exactly match the user's stored template. Because biometric measurements almost always contain variations, these systems

cannot require an exact match between the enrollment template and a current pattern. Instead, the current pattern is considered valid if it falls within a certain statistical range of values. A comparison algorithm is used to determine whether the user being verified is the same user that was enrolled.

The comparison algorithm yields a result that indicates how close the live scan is to the stored template. If the result falls into an "acceptable" range, an affirmative response is given; if the result falls into an "unacceptable" range, a negative response is given. The definition of "acceptable" differs for each biometric. For some biometrics, the system administrator may set the level of the acceptable range. If this level is set too low, however, the biometric fails to be a valid authentication mechanism. Similarly, if it is set too high, the authorized users may have trouble being authenticated. Pattern matching is fundamental to the operation of any biometric system and therefore should be considered a primary factor when you're evaluating a biometric product. Figure 9-11 illustrates a typical verification process.

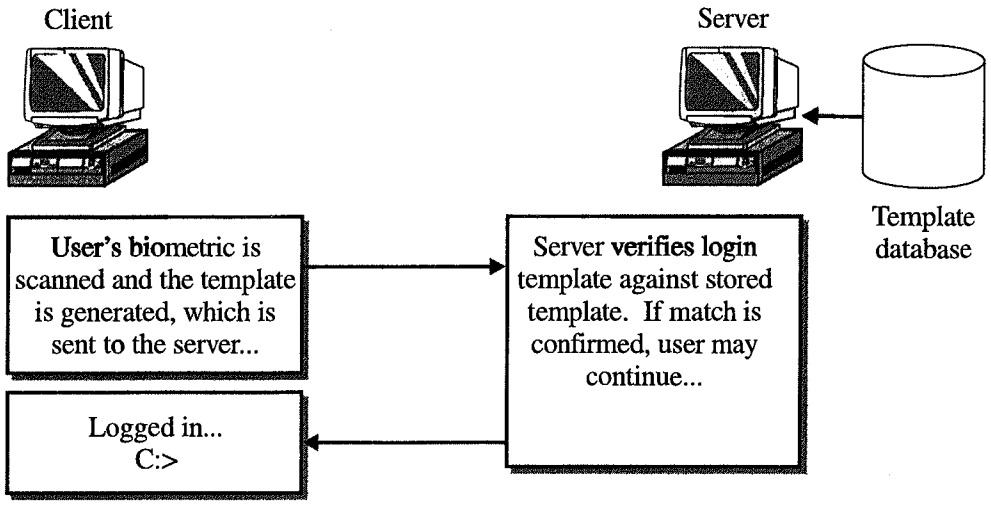


Figure 9-11
Verification process

Templates

In general, most available biometric authentication mechanisms function as explained in the preceding sections. One key feature of biometrics is the template. The accumulated templates of all users are referred to as the *template database.* These databases require the same protections as password databases. The size of the templates vary from system to sys-

tem. When you're testing these systems for accuracy, you should examine the templates to determine whether unique biometric features are adequately represented.

Another aspect of templates that affects biometric authentication is the approach taken by the comparison algorithm in using the template. Most devices use the template for verification, but some use it for identification. In the latter, the device takes a live scan and then compares it against the entire template database to determine whether any of the stored representations falls within the acceptable comparison algorithm range. In contrast, a biometric verification compares the live scan only against the single template of the person whom the user claims to be. For example, a user types a user name and then submits to a live scan for verification. The comparison algorithm compares the scan only to the template associated with that user name. Typically, verification biometrics are faster because they do not have to compare the live scan against the entire template database.

Recognition Methods

Just as every human body has countless unique characteristics, countless recognition methods can be used in biometrics. Let's look at some of the common biometric recognition methods in use.

Fingerprint Recognition

Fingerprint recognition is probably the most common form of biometrics available. This form of data encryption has evolved from the use of fingerprints for identification over the past several decades. By having an individual scan a fingerprint electronically to decode information, the transmitter of the data can be certain that the intended recipient is the receiver of the data. When scanned electronically, fingerprints provide a higher level of detail and accuracy than can be achieved with manual systems.

Another strength of fingerprint biometrics is that giving fingerprints is more widely accepted, convenient, and reliable than other forms of physical identification, especially when technology is used. In fact, studies have shown that fingerprint identification is currently thought to be the least intrusive of all biometric techniques. One concern of fingerprint biometrics is that latent prints left on a scanning medium will register a prior user; however, units exist that do not operate unless a "live" finger is on

the medium, and they register only the later imprint. The error rate experienced with this form of encryption is approximately 1 in 100,000 scans.

One of the most important features of fingerprint biometrics is its low cost. Scanners are fairly inexpensive, and as the technology becomes more common the cost should only decrease. In fact, in anticipation of widespread use of this technology in the future, some mouse manufacturers are developing products with built-in fingerprint scanner technology.

Optical Recognition

There are two common types of optical biometrics: retinal and iris. These devices are more accurate than fingerprint and hand devices because both the retina and the iris have more characteristics to identify and match than those found on the hand. Retinal and iris scanning devices have come a long way in recent years and now allow individuals to be scanned even through eyeglasses or contact lenses. The error rate for a typical retina or iris scanner is about 1 in 2,000,000 attempts, something that further demonstrates the reliability of this technology. Two drawbacks to these devices, however, are that they have difficulty reading images of those people who are blind or have cataracts and that they currently are cumbersome to use.

The cost of these systems averages $6,500, making them somewhat unattractive for network users. But as this technology becomes more standardized and accepted, the cost should fall and become less of a factor in decision making.

Facial Recognition

In this form of biometrics, an image is examined for overall facial structure. This approach is often less reliable than more common forms such as fingerprints and iris scans. Moreover, the interpretative functions performed by the computer are much more subjective using this technology. Although one benefit of facial biometrics is that it can be applied at either at close range or over greater distances, it loses accuracy progressively as the distance increases between the individual and the scanner. Changes in lighting can also increase the error rate.

An attractive feature of facial recognition products is their low cost. Units can typically be purchased for as little as $150. At this price, this technology might lend itself to electronic commerce, but the units can be cumbersome to use and still are not as reliable as other forms of biometrics for encryption purposes.

Voice Recognition

Voice recognition offers several advantages for use in encryption. Not only is voice biometrics perfect for telecommunications applications, but also most modern personal computers already have the necessary hardware to use the applications. Even if they don't, sound cards can be purchased for as little as $50, and condenser microphones start at about $10. This means that for less than $100, individuals can possess the technology needed to have fairly reliable biometric encryption technology for use over the Internet.

This type of biometric is not as accurate, however, as some other forms. The error rate for voice recognition ranges between two percent and five percent. However, it lends itself well to use in the public telephone system and is more secure than PINs.

Some drawbacks to this technology are that voiceprints can vary over the course of the day, and if a user has a health condition such as a cold or laryngitis, it can affect verification.

Signature Recognition

Most adults are familiar with the signing of documents. In our personal lives we sign everything from personal checks to birthday cards. In the business world we sign things such as expense accounts and other official documents. This widespread practice lends itself well to the use of signature recognition as a means of biometric verification in electronic commerce. This type of signature identification, however, is different from the normal two-dimensional signature that you find on a form or document. Biometric signature recognition operates in a three-dimensional environment that uses measurements not only of the height and width but also the amount of pressure applied in a pen stroke; the latter measurement gauges the depth of the stroke as if it were made in the air. This extra dimension helps to reduce the risk of forgery that can occur in two-dimensional signatures.

One drawback to signature recognition is that people do not always sign documents in exactly the same manner. The angle at which they sign may be different because of their seating position or their hand placement on the writing surface. Therefore, even though the three-dimensional approach adds to its ability to discern impostors, this method is not as accurate as other forms of biometric verification.

Signature recognition systems are not as expensive as some of the higher-end systems such as iris scanners; they are priced more in the

range of voice and fingerprint scanners, and that makes them affordable for network use.

Keystroke Recognition

This technology is not as mundane as it sounds. The concept is based on a password or PIN system but adds the extra dimension of keystroke dynamics. With this technology, not only must intruders know the correct password, but they must also be able to replicate the user's rate of typing and intervals between letters. Even if an unauthorized person is able to guess the correct password, it's unlikely that he will be able to type it with the proper rhythm unless he has the ability to hear and memorize the correct user's keystrokes.

Keystroke recognition is most likely one of the least secure of the new biometric technologies that have evolved in recent years, but it is also probably one of the least expensive and easiest to implement. It probably won't gain much attention for use in electronic commerce because similarly priced systems offer far more reliability.

Biometric Accuracy

When you're choosing a biometric authentication system, an important consideration is its accuracy. The accuracy of biometric authentication systems can be categorized by two measures: the *false acceptance rate* (FAR) and the *false rejection rate* (FRR). A system's FAR reflects the situation in which a biometric system wrongly verifies an identity by matching biometric features from individuals who are not identical. In the most common context, false acceptance represents a security hazard. Similarly, a system's FRR reflects the situation in which a biometric system is not able to verify the legitimate claimed identity of an enrolled person. In the most common context, the user of a biometric system will experience false rejection as inconvenience.

Suppliers of biometric systems often use FAR together with FRR to describe the capabilities of the system. Obviously, FRR and FAR are dependent on the threshold level. Increasing the threshold will reduce the probability of false acceptance and therefore enhance security. However, system availability will be reduced due to an increased FRR.

How these rates are determined is fundamental to the operation of any biometric system and therefore should be considered a primary factor

when a biometric system is evaluated. You should be aware that manufacturers' FAR and FRR numbers are extrapolated from small user sets, and the assumptions for the extrapolations are sometimes erroneous.

You should assess these performance factors with an eye toward the type of users who will use the system. For a proper live scan to be taken, users must become familiar with the device. You can expect it to take two weeks before the false rejection rate drops off. Another user consideration is that not all users may be able to use the biometric—for example, because of an impairment that prevents them from taking an acceptable scan. In that case, you may need to provide an alternative method to grant those users access, or you may have to select a biometric based on the needs of each set of users. When selecting a biometric, also consider user acceptance. Some biometrics have met with resistance from users because the technology is too invasive.

Combining Authentication Methods

Passwords, authentication tokens, and biometrics are subject to a variety of attacks. Passwords can be guessed, tokens can be stolen, and even biometrics have certain vulnerabilities; these threats can be reduced by applying sound design principles and system management techniques during the development and operation of your authentication system. One method that can substantially increase the system's security is to use a combination of authentication techniques.

For example, an authentication system might require users to present an authentication token and also enter a password. By stealing a user's token, an attacker would still not be able to gain access to the host system because the system would require the user's password in addition to the token. Although it might be possible to guess the user's password, the host system can make this extremely difficult by locking the user out after a specified number of invalid passwords have been presented in succession. After a user's account has been locked in this manner, only the appropriate system administrator or security officer should be able to unlock the account.

Tokens can also be used to store biometric templates for user authentication. After enrollment, the user's unique template could be stored on a token rather than in a file on the host system. When the user requests access to the system, a current template is generated and compared to the

enrollment template stored on the user's token. It would be preferable for this comparison to be carried out internally by the token because in that way the enrollment template would never need to leave the token. However, often this method is not possible because of the complexity of the algorithms used for the comparison. The microprocessors typically used in smart tokens cannot execute these algorithms in a reasonable time. If the template comparison is done by the host system, the host must provide adequate assurance that users' templates cannot be compromised. In addition, the token and host system should implement an authentication protocol that ensures two things: that the host system is obtaining the template from a valid token and that the token is submitting the template to a valid host. The ideal situation is to have both the biometric sensors and the comparison algorithm implemented on the token. In that case, the token can perform the entire biometric authentication process. Figure 9-12 shows one of the newer products available on the market, which combines authentication methods.

Figure 9-12

BioMouse Plus from American Biometric Company

Summary

A wide variety of cryptographic hardware is available on the market. Various tokens can be used for authentication, as can various microprocessor cards, biometrics, and accelerators. Each of these approaches has its place, for the right price.

Vendors

A great many vendors manufacture and sell cryptographic accelerators, tokens, smart cards, and biometric devices. Table 9-1 lists some of the manufacturers and the products they sell.

Table 9-1

Cryptographic Manufacturers

Companies/ Device	Accelerators	Tokens/ Smart cards	Biometrics
nCipher http://www.ncipher.com/	Nfast		
Compaq http://www.compaq.com/	AXL300		
Rainbow Technologies http://www.rainbow.com/	CryptoSwift, NetSwift	iKey, Sentinel	
RSA Security, Inc. http://www.rsasecurity.com/		SecurID	
DataKey http://www.datakey.com/		Datakey	
Ensure Technologies http://ensuretech.com/ cgi-bin/dp/framesethome.dt/		XyLoc Security Server	
Dallas Semiconductor http://www.dalsemi.com/index.html		iButton	
GemPlus http://www.gemplus.com/		GemClub-Micro, GemStart, GemWG10	
American Biometric Company http://www.abio.com/			BioMouse Plus
AuthenTec, Inc. http://www.authentec.com/			FingerLoc, EntrePad

CHAPTER 10

Digital Signatures: Beyond Security

Thanks to the Internet, e-commerce has dramatically changed our ways of conducting business. As each day passes, paper-based transactions—including agreements that have legal force—are becoming obsolete as the use of electronic agreements transmitted over the Internet increases in popularity. The main motivation for this change is convenience. Distance, for example, is no longer a barrier to getting an agreement signed. Within seconds, an electronic agreement can travel across the world, receive an electronic (or digital) signature, and be returned completed. But this new world of e-commerce requires close attention to legal and technical issues.

Users' experiences with digital signatures (see Chapter 5) have shown that this technology can save e-commerce parties time and money. Compared with paper signatures, digital signatures offer a number of benefits:

- **Message integrity** A digital signature is superior to a handwritten signature in that it attests to the contents of a message as well as to the identity of the signer. As long as a secure hash function is used, there is no way to take someone's signature from one document and attach it to another, or to alter the signed message in any way. The slightest change in a signed document will cause the digital signature verification process to fail. Thus, authentication allows people to check the integrity of signed documents. Of course, if

signature verification fails, it may be unclear whether there was an attempted forgery or simply a transmission error.

- **Savings** The use of open systems (such as the Internet) as transport media can provide considerable savings of time and money. Furthermore, adding automation means that data can be digitally signed and sent in a timely manner.
- **Storage** Business data (contracts and similar documents) can be stored much more easily in electronic form than in paper form. Furthermore, in theory an electronic document that has been digitally signed can be validated indefinitely. If all parties to the contract keep a copy of the time-stamped document, each of them can prove that the contract was signed with valid keys. In fact, the time stamp can prove the validity of a contract even if one signer's key becomes compromised at some point after the contract was signed.
- **Risk mitigation** If properly implemented, digital signatures reduce the risk of fraud and attempts by a party to repudiate (disavow) the contract.

Before companies and individuals adopt these new techniques, however, they must first address a few legal and technical concerns. In U.S. federal law, under the Statutes of Frauds, a party that claims that a contract was made must provide proof. The traditional method of proof is the document with a handwritten signature. The question is whether an electronic document containing a digital signature is secure and therefore reliable as proof. The Federal Rules of Evidence allow computer data to be admitted as business records if a foundation is established for their reliability. As this book is being written, new federal legislation has taken effect. This legislation provides that an electronic signature has the same legal status as a handwritten signature. It should be noted, however, that these new laws are still untested.

This chapter provides insight into the many aspects of digital and electronic signatures as they apply to e-commerce. We discuss concepts and requirements, legal and technical, that users must completely understand if they hope to apply these signatures. We also look at the various relevant laws, including the newly enacted federal *Electronic Signatures in Global and National Commerce* (E-SIGN) Act. Finally, we discuss the differences between electronic and digital signatures and how each falls short if the proper concepts and requirements aren't used.

Legislative Approaches

As we've discussed, digital signatures offer a range of benefits for businesses and consumers alike. For digital signatures to make their way into mainstream, however, two barriers must be overcome:

- How to give documents that exist only in electronic form the same legal status as paper documents
- How to provide a secure, reliable, and legally sanctioned method for "signing" electronic documents that will eliminate the need to generate and sign paper documents, thereby encouraging and facilitating electronic commerce

Both problems require legislative solutions.

Legal Guidelines from the American Bar Association

The *American Bar Association* (ABA), the organization that represents the legal profession in the United States, has done considerable work on the legal aspects of digital signatures. In 1996, the ABA's Information Security Committee, Section of Science and Technology, published a document titled "Digital Signature Guidelines." These guidelines were originally drafted to provide "general, abstract statements of principle, intended to serve as long-term, unifying foundations for digital signature law across varying legal settings." Many states have chosen to model their own digital signature legislation after these guidelines.

Many legal professionals, with the exception of the ABA special interest legal groups, are playing catch-up in the fast evolving and sometimes complicated digital world. As the number of e-commerce sites using digital signatures increases, so will the need for lawyers who can render sound legal advice. Clients will begin to look to attorneys and others for guidance about the appropriate level of security for a given line of electronic business and other transactions.

It will be of the utmost importance for attorneys to cooperate closely with business and technical specialists in the procurement and deployment of computer security systems generally, and specifically those systems that require electronic signatures. The legal consequences that flow

from the presence or absence of particular elements of data security will constitute risks, liabilities, and other potential costs that should be taken into account from the beginning.

Legal Concepts Related to Digital Signatures

Because electronic documents can be easily copied and modified without detection, they cannot automatically be assumed to be authentic. Moreover, unlike hand-written characters, digitally encoded characters are not unique. The signature on an electronic document is not physically connected to the document's content.

To withstand both legal and technical tests, the recipient of an electronic document containing a digital signature must be able to prove to an impartial third party (a court, a judge, or a referee before whom the parties have agreed to submit for resolution any issue or dispute) that the contents of the document are genuine and that it originated with the sender. In addition, the signature must be such that the sender cannot later disavow the contents of the document.

Before we go any further, let's review the concepts of nonrepudiation and authentication, which have been described earlier (see Chapters 5 and 6). These concepts play a key role in the legalities of digital signatures, and it is important to understand how they differ in the digital world compared with the paper world.

Nonrepudiation

Nonrepudiation, at its most basic, is the ability to prove to an impartial third party—after the fact—that a specific communication originated with and was submitted by a certain person or was delivered by a certain person. Nonrepudiation, then, defines the means that are used to prevent illegitimate breaches of contract on the same grounds. This means that evidence exists thst ties the identity of a party to the substance of a message or transaction at a certain point in time and that the evidence is sufficiently strong to prevent or rebut that party's subsequent denial of it.

The 1988 ISO Open Systems Interconnection Security Architecture standard provides a limited definition of nonrepudiation as a security ser-

vice that counters repudiation, where *repudiation* is defined as "denial by one of the entities involved in a communication of having participated in all or part of the communication." Signatures, seals, recording offices, certified mail, letters of credit, notaries, auditors, and collateralized bills of lading are examples of nonrepudiable business practices traditionally employed to support legally binding business transactions.

These elements of nonrepudiation must now be incorporated into the electronic environment—in real time, with full assurance, and without a paper trail.

In the absence of this kind of rigor, how can businesses operating at Internet speed avoid or resolve disputes? It is only with a full set of digital nonrepudiation elements that irrefutable evidence can be shown in a court of law. Otherwise, businesses aren't protected against breach of contract, fraud, currency fluctuations, insolvency, credit risks, incomplete funds delivery, and operational failure.

Nonrepudiation services provide trusted evidence that a specific action occurred. The concept of nonrepudiation, as it pertains to information security and digital signatures, can be broken into three types: nonrepudiation of origin, nonrepudiation of submission, and nonrepudiation of delivery.

- **Nonrepudiation of origin** This concept protects the recipient of a communication by guaranteeing the identity of the originator of a communication. It further confirms the time the message was sent and ensures that the message was not tampered with during transmission.
- **Nonrepudiation of delivery** This concept protects the sender of a communication by guaranteeing essentially the same elements as does nonrepudiation of origin. As with nonrepudiation of origin, it can be used to provide the time a message was sent and to indicate whether the data was tampered with during transmission.
- **Nonrepudiation of submission** This concept is similar to nonrepudiation of origin and delivery except that it is used to protect the sender against any claim by the recipient that the data wasn't sent or wasn't sent at a specific time.

Authentication

For the purposes of this chapter, and in relation to digital signatures, two types of authentication must be understood: signer authentication and data authentication.

For a document to have any legal force, the signer of the document must be authenticated; this concept is called *signer authentication.* If someone signs a loan certificate, for example, the bank can store the borrower's signature for use later in legal ways because the signature is believed to authenticate the borrower with a high probability. A signature should indicate who signed a document, message, or record, and it should be difficult for another person to produce the signature without authorization. If a public/private key pair is associated with an identified signer, the digital signature attributes the message to the signer. The digital signature cannot be forged unless the signer loses control of the private key (a "compromise" of the private key), such as by divulging it or losing the medium or device in which it is contained.

Data authentication is comparable to stamping a document in a way that disallows all future modifications to it. Data authentication is usually accompanied by *data origin authentication,* which binds a concrete person to a specific document (for example, by limiting the number of persons who use the stamp). A signature should identify what is signed, making it impracticable to falsify or alter either the signed matter or the signature without detection. The digital signature also identifies the signed message, typically with far greater certainty and precision than paper signatures. Verification reveals any tampering because the comparison of the hash results (one made at signing and the other made at verifying) shows whether the message is the same as when signed.

Signer authentication and data authentication are used to exclude impersonators and forgers, and they are essential ingredients in what is often called a *nonrepudiation service.* A nonrepudiation service provides assurance of the origin or delivery of data in order to protect the sender against false denial by the recipient that the data has been received, or to protect the recipient against false denial by the sender that the data has been sent. Thus, a nonrepudiation service provides evidence to prevent a person from unilaterally modifying or terminating legal obligations arising from a transaction effected by computer-based means.

Written Versus Digital Signatures

Although digital and written signatures can serve the same purposes, there are obvious physical differences. Let's look at the differences between the signatures applied to written and digital documents.

Written Documents

Traditionally, someone's signature on a literal document authenticates the origin of the data contained in it. Because people sign various documents during their lifetimes, their signatures become a part of their identity over time. By using a unique combination of pencil strokes that is very difficult for anyone else to forge, they can sign anything, almost without thinking. Additionally, loan certificates (and other documents that may have legal force) have been designed to guard against forging of a signed document. Examples include documents that use watermarks, embossing, and special ink treatment, all of which provide protection against photocopies and other forgeries.

Digital Documents

Electronic documents can easily be copied and modified without detection. To generalize this consideration, *digital information* is usually defined (loosely) as the kind of information not bounded to any concrete carrier, such as the ink on a piece of paper. Additionally, the digital information lacks *personality* (a file saved by someone can be easily updated by someone else having the appropriate permissions).

Clearly, the traditional methods of signing by appending the signature to an existing document do not work for electronic documents. Anyone can simply modify the document and append the same signature to it.

Requirements for the Use of Digital Signatures

For current digital signature legislation to withstand the test of litigation, a number of important issues must be resolved. The American Bar Association's "Guidelines for Digital Signatures" is an excellent foundation, but corporations and individuals might wish to focus on concerns not addressed in the guidelines. The following sections describe those requirements, which are essential if digital signatures are to stand up.

Public Key Infrastructures

To effectively incorporate digital signatures within an e-commerce framework, an organization should create and maintain a *public-key infrastructure* (PKI), as described in Chapter 6. To a point, having a PKI ensures that only valid keys are used in signing and verifying electronic documents.

The PKI must enforce policies whereby properly administered *certification authorities* (CAs) and *registration authorities* (RAs) are used, requiring end users to show reliable proof that authenticates them. Furthermore, public-key certificates can be housed in a central location that can be accessed by any relying party. Finally, a PKI serves to revoke or suspend certificates as needed.

Control of Key Revocation

Another important issue related to the use of digital signatures is the management of private signature keys. If an unauthorized person gains access to a private key, the thief will be able to forge the owner's signature on electronic documents. To prevent this, a user should be able to revoke a compromised signature key in the public directory. Here are some guidelines:

- All users should be able to revoke their public keys from the directory at any time. For this policy to work, CAs should save (in the public directory) information about all revoked keys.
- An authority should be able to revoke the signatures issued for its employees. A separate CA could certify digital signatures for employees of a given company.
- *Online Certificate Status Protocol* (OCSP), which was explained in Chapter 6, should be used to ensure that verifiers receive the most current revocation status.

Time-Stamping

Another issue is *time-stamping*. Digital signatures provided through the use of public-key technology can be called into question for a simple reason: If the signer of a particularly important document (for example, a

loan agreement) later wishes to repudiate her signature, she can dishonestly report the compromise of her private key and ask it to be revoked. A later verifier will not be able to certify whether the signing happened before or after the revocation.

Time-stamping is a set of techniques that enable you to ascertain whether an electronic document was created or signed at (or before) a certain time. In practice, most time-stamping systems use a trusted third party called a *time-stamping authority* (TSA). A *time stamp* is the TSA's digital attestation that an identified electronic document was presented to the TSA at a certain time.

A *time-stamping service* (TSS) is a collection of methods and techniques providing long-term authentication of digital documents. The object of a TSS is to authenticate not only the document but also the moment in time at which the document is submitted for authentication. Figure 10-1 illustrates the interaction between end-users and a trusted time-stamping server available from Datum.

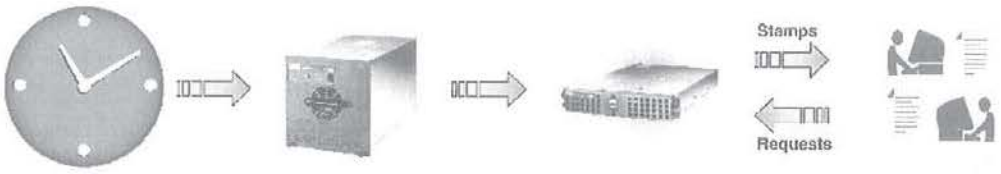


Figure 10-1
Time-stamping components

The importance of time-stamping becomes clear when there is a need for a legal use of electronic documents with a long lifetime. Without time-stamping, you cannot trust signed documents after the cryptographic primitives used for signing have become unreliable, nor can you resolve cases in which the signer repudiates the signing, claiming to have accidentally lost the signature key.

During recent years, especially in the context of legal regulation of the use of digital signatures, the organizational and legal aspects of time-stamping have become the subject of worldwide attention. Time-stamping helps to significantly lower the level of trust currently required of a PKI by making it possible to prove that a document was signed before the corresponding signature key was revoked. For that reason, organizations often depend on time-stamping to resolve the status of documents.

Current and Pending Legislation

Digital signature legislation has been an ongoing issue for some time. Worldwide, especially in Europe, digital signature laws have been in effect for about a decade. The *United Nations Commission on International Trade Law* (UNCITRAL), a model law on electronic commerce, took effect in 1996 and has had a major influence on signature laws worldwide. The UNCITRAL model law takes a high-level, enabling approach to electronic signatures and records, with no mention of digital signatures or cryptography.

Only during the past five years has the United States gained momentum in this legal arena. The first state law, enacted in Utah in 1995 and amended in March 1996, is widely recognized as an important and positive first step toward legal recognition of digital signature technology. The Utah act provides for the licensure of certification authorities by the Utah Department of Commerce. Utah's law also details the rights and liabilities of parties to a transaction using public-key cryptography and a licensed certification authority. In 1996, Washington state adopted legislation closely resembling the Utah law. Other states, most notably Georgia, began considering bills modeled after the Utah law, and, for a time, it seemed that a consensus was developing among the states.

Now, however, various policy issues have increasingly moved states toward approaches that are less regulatory, less technology-specific, and more incremental. For example, California and Arizona enacted legislation permitting the use of digital signatures for transactions involving state entities. This legislation authorized the two states' secretaries of state to promulgate regulations to achieve the purpose of the act. Still other states have passed laws permitting the use of electronic signatures for particular purposes, such as for medical records (Connecticut) or for budget and accounting purposes, such as electronic check signing by the treasurer (Delaware). Georgia, along with a number of states that had legislation resembling the Utah act, have allowed the bills to die and opted for further study.

The effort in Massachusetts exemplifies an attempt to craft laws that directly address the legal issues raised by electronic commerce but do not exclusively codify public-key cryptography in statute. This approach seeks generally to remove legal obstacles to electronic communications and transactions by giving legal effect to electronic signatures and electronic records. The law would also specifically provide for the admissibility of electronic signatures and records.

The problem with the state laws, however, is that no two sets of laws are the same. Building on the work in Massachusetts, the federal government is trying to provide a solution by working on new federal legislation. The U.S. House and Senate, after long negotiations, compromised on a new electronic signature bill, the Electronic Signatures in Global and National Commerce (E-SIGN) Act, on June 9, 2000. The E-SIGN Act makes electronic, or online, signatures as legally binding as ink-and-paper signatures and states that they can be used as evidence in legal proceedings.

The E-SIGN Act

President Bill Clinton signed the E-SIGN Act on June 30, 2000. E-SIGN gives legal recognition and effect to electronic signatures, contracts, and records, and it empowers the use of online contracts and provision of notices. The law became effective October 1, 2000, except for certain provisions affecting the use of electronic records to satisfy records retention requirements, which became effective March 1, 2001. E-SIGN requires a consumer to agree to electronically signed contracts and consent to receiving records over the Internet. Companies must verify that customers have an operating e-mail address and other technical means of receiving information. Some notices, such as evictions, health insurance lapses, or electricity lapses, must still come in paper form.

Under E-SIGN, federal agencies are given authority allowing them to unconditionally exempt specified types of records from the consumer consent provisions. Most notably, the legislation directs the *Securities and Exchange Commission* (SEC) to use this authority to issue a regulation that effectively allows mutual funds to provide prospective investors with an electronic fund prospectus at or before the time they access electronic sales literature, without first obtaining investor consent to the electronic format of the prospectus. In this way, funds can continue the practice, permitted under the SEC's interpretive releases, of using hyperlinks on their Web sites to give prospective investors simultaneous access to both sales literature and the fund's prospectus.

E-SIGN was originally designed to boost Internet e-commerce transactions, for both *business-to-business* (B2B) and *business-to-consumer* (B2C) markets, by eliminating paperwork arising from contracts. The effect of the E-SIGN Act is uniform nationwide legislation enabling the use of electronic records and signatures for interstate and international commerce.

Electronic Versus Digital Signatures: What's the Difference?

Simply put, an *electronic* signature is any symbol or method, accomplished by electronic means, that is executed or adopted by a party with present intention to be bound by or to authenticate a record. An electronic signature can be created by any electronic means. For example, the output of a sophisticated biometric device, such as a fingerprint computer recognition system, could qualify as an electronic signature, and so would the simple entry of a typed name at the end of an e-mail message. The principle is that the symbol or method was executed or adopted by the signer with a present intent to sign the record. This definition focuses on the traditional legal purposes of a signature and not on the particular medium or manner chosen to effect the signature.

In contrast, a *digital* signature refers to a particular implementation of public-key cryptography (such as the implementation described in Chapter 5). More formally, a digital signature can be defined as the transformation of a record using an asymmetric cryptosystem and a hash function such that a person having the initial record and the signer's public key can accurately determine (a) whether the transformation was created using the private key that corresponds to the signer's public key and (b) whether the initial record has been altered since the transformation was made.

In other words, a digital signature is created by use of a public-key system, whereas an electronic signature is produced by any computer method, including public-key systems. Digital signatures are technology-specific. Electronic signatures are technology-neutral.

The use of low-security electronic signatures, such as simply typing one's name on an e-mail, raises serious questions of proof regarding the authenticity of such a signature. However, there are times when little or no security is warranted. A given transaction or message may be informal, of little or no value, or otherwise not reasonably likely to form the basis of subsequent dispute. For example, it's common practice to conclude purely social e-mail messages with the typing of the sender's name. In this case, the name is a symbol intended to authenticate the document but not necessarily manifesting intent to be bound by the content—assuming there exists

any particular content at all. In this context, the word "authenticate" means merely the intention to represent that the signer was the sender. In common practice, e-mail among friends and close colleagues is often concluded with the initials of the sender alone.

For more formal, but low-risk, electronic transactions, a more robust signature system may be desirable. This does not necessarily mean that a full-fledged public-key solution is required. For example, some business and professional online services require entry of a user name and password to access their systems. After users are on the system, they may be entitled to additional information or services, such as online dialog with an expert or authorization to view value-added proprietary documents. Here, the electronic signature is created by use of a user name and password, probably relying on access control technology far less expensive and simpler to use than public-key cryptosystems. Depending on the understanding of the parties as evidenced by contracts, disclaimers, or other conditions of use, the use of this system may authenticate the user and also by implication, or perhaps expressly, express intent to be bound by billing rates or other terms.

Following is a description of various E-SIGN provisions:

- **Technology** E-SIGN requires that parties to a contract decide on the form of electronic signature technology. From a scanned handwritten signature to biometric-protected smart cards, E-SIGN allows the use of various forms of technology as long as both parties agree.
- **Notification** The E-SIGN Act provides the following with regard to notification:
  1. The consumer decides whether to use an e-signature or handwritten signature; the consumer must give consent before receiving bills and other documents only in electronic form.
  2. Cancellation and foreclosure notices must be sent on paper.
  3. The vendor must conduct test e-mailings before sending subsequent e-mail notifications.
  4. The law does not allow e-signatures on adoptions, wills, and product safety recalls.

- **Rights** Consumers must be made aware of any right or option to receive a disclosure in paper form and what they must do to obtain paper copies. Furthermore, consumers must be made aware of the right to withdraw consent to have records provided electronically, including any conditions, consequences, or fees associated with doing so. The organization must describe the procedures for withdrawing consent and for updating information needed to contact the consumer electronically.
- **Consent** Consumers must be presented with and must confirm the hardware and software requirements for access and retention of electronic records and must confirm consent to the contract. Both confirmations must be visibly and conspicuously separate from all other terms and agreements.
- **Consumer obligations** The consumer is obligated to inform electronic records providers of any change in e-mail address or other location to which the electronic records may be provided. Furthermore, the consumer is obligated to notify the electronic records provider before withdrawal of consent.
- **Enforcement** The E-SIGN Act provides for its enforcement by giving authority to government agencies as needed to protect the public interest.

Dealing with Legal Uncertainties

Because the E-SIGN Act does not prescribe the technology that must be used to sign and verify an electronic document, an electronic signature could simply be a person's typed name on e-mail. All that is required is for both parties to agree to the technology. To the best of our knowledge, such a signature in no way fosters nonrepudiation and authentication, which have always been the foundation for commerce as we know it.

Ultimately, we believe that a more solid foundation will be needed. The concepts of authentication and nonrepudiation are crucial to the operation of business transactions. To separate authorized users of information from unauthorized users, there must be a reliable way to ascertain the identity of the user. The Internet was not designed with adequate technical means to achieve this identification. In fact, without the existence of the requirements listed in "Requirements for the Use of Digital Signatures," it is easy impersonate someone else.

Finally, because the validity of documents with these new electronic signatures has never been challenged in court, their legal status is truly not yet defined. It's likely that through such challenges, we will see the courts issue rulings that will better define which methods, key sizes, and security precautions are acceptable for electronic signatures to be legally binding.

Summary

Digital signatures have the potential to possess greater legal authority than handwritten signatures. Why? Digital signatures may provide a higher degree of nonrepudiation and authenticity than their handwritten counterparts. For example, if a ten-page document is signed by hand on the tenth page, one cannot be sure that the first nine pages have not been altered. However, if an electronic document is signed with a digital signature, a third party can verify that not one byte of the contract has been altered. For this and other reasons, digital signatures also save the parties time and money.

However, if digital signatures are to replace handwritten signatures, serious issues—some of which revolve around current legislation—must be answered. For example, is the current E-SIGN Act enough? Do electronic signatures provide the same level of nonrepudiation and authenticity provided by handwritten signatures?

E-SIGN is a great leap forward for both interstate and international Internet commerce. However, E-SIGN should be seen more as a foundation on which to build with current and emerging technologies, such as the use of public-key technology, PKIs, and digital notaries.

Real-World Examples

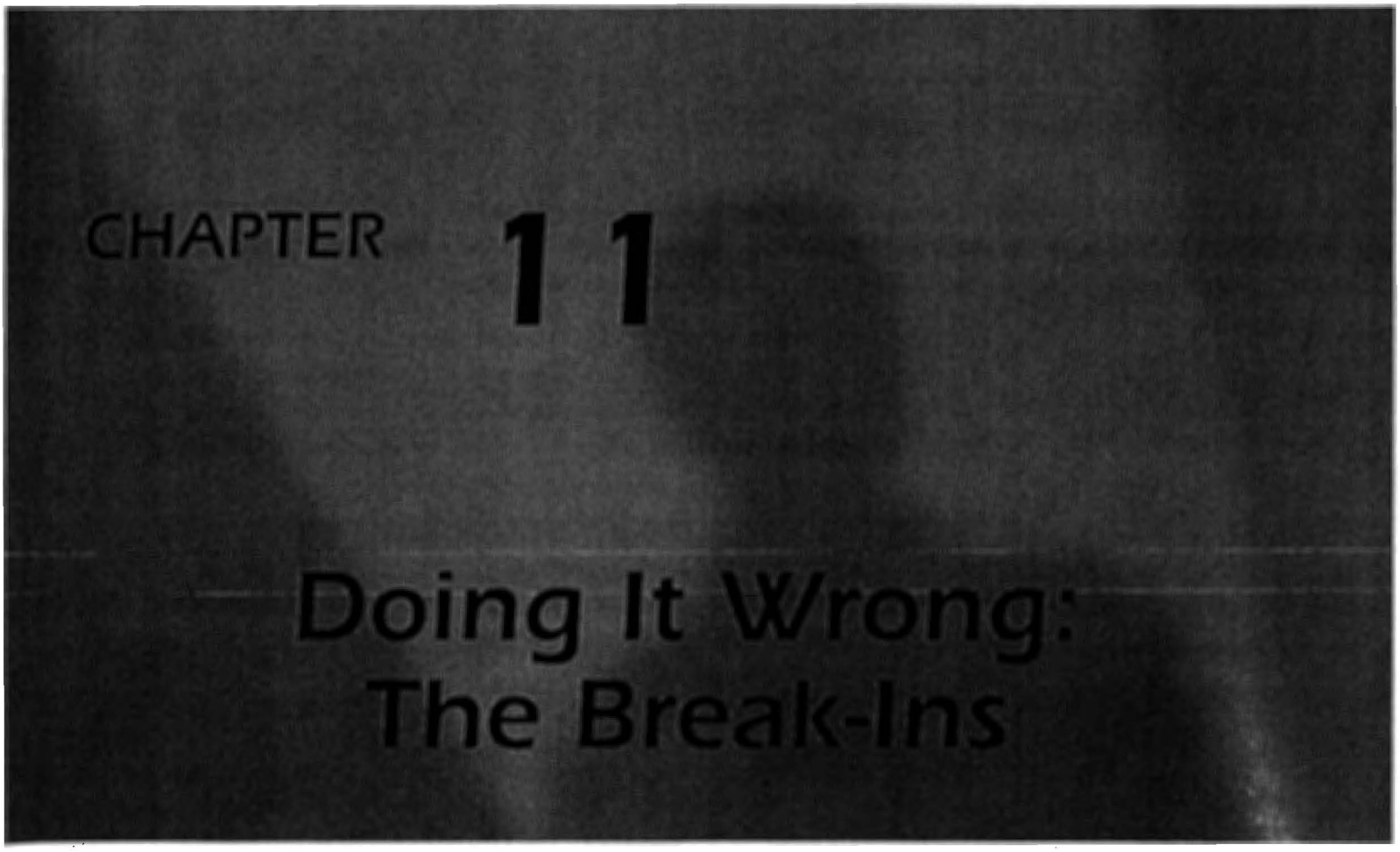
A number of relevant products can be purchased or downloaded free from the Internet. They range from enabling software to hardware that allows users to authenticate themselves to their private signing key. Following are only a couple of the available solutions.

- RSA Security, Inc., as well as a number of other security software vendors, offers developer *software development kits* (SDKs) and products. BSAFE Cert-C and Cert-J, for example, allow developers to use public-key certificates for a number of security services such as digital signatures.
- Datum carries an excellent time-stamp device to be used in conjunction with digital notaries or time-stamp authorities or to provide in-house time-stamp services.

In addition to security vendors that sell products designed for users and developers, we will likely see the advent of more businesses that will offer services to support digital signatures. Such services include certification authorities and time-stamp and digital notary services. Here are some examples:

- Digisign is one company that has already begun selling time-stamp and digital notary services.
- VeriSign is a certification authority that issues public-key certificates to end users.

Finally, we should not forget legal professionals. A great many legal professionals have taken the time to become technically savvy, and we expect to see this number increase as more related legal cases are seen in the future.



Over the past two decades, the computer industry has really taken off, and the number of security incidents has increased significantly. Corporations as well as individuals have learned the hard way that data can easily be accessed, disclosed, modified, or even deleted if proper security is not provided.

Over the years, companies have fallen short in their efforts to implement cryptographic solutions both in their own products and services and in attempts to protect their internal enterprise from intruders. This chapter summarizes the various types of losses that occur when a system is not properly secured. We also outline the kinds of threats and intruders that have come to be widely reported. Finally, we look at a number of case studies in which security was either overlooked or failed because of improper implementation. (We describe successful case studies in Chapter 12.)

Measuring Losses

The kinds of losses that organizations can experience because of lapses in computer security can be counted in a number of ways. Many people think

first of the direct forms of loss, such as the loss of data. However, when you look closely at what is at stake, loss of data is only the beginning. Following is a short list of the types of losses that occur:

- **Loss of data or secrets** When people hear the word "hacker," this is perhaps one of the first types of loss that they think of. This category includes the loss of user credit card numbers, compromise of financial reports, and unauthorized access to medical information.

NOTE:

The data itself need not have been stolen for a serious loss to result. Instead, an attacker may manipulate the data in such a way that it is rendered inaccurate or unusable.

- **Loss of reputation** After a successful breach of security, end users may abandon a service or product because they're afraid to use it. Yet another aspect of this type of loss is the effect it has on assessments of a corporation by financial analysts. Sometimes an analyst's negative evaluation can have as great an impact as the break-in itself. This may be one of the main reasons that corporations seldom report break-ins and theft of data.
- **Financial losses** In addition to direct financial thefts, loss of data and loss of reputation will result in financial losses. Financial losses can be one of the most difficult to quantify. One reason is that no one knows exactly how many current customers will not return following a break-in or, worse yet, how many potential new customers will never make the attempt.

Types of Security Threats

To implement security effectively, corporations as well as individuals need to be aware of a variety of potential threats. Let's take a look at each these threats.

NOTE:
Each of the following threats does not necessarily require direct human interaction. Through the use of computer viruses or Trojan Horse applications, data can easily be destroyed, manipulated, or sent to an intruder for viewing.

Unauthorized Disclosure of Data

Unauthorized disclosure of data results from an individual accessing or reading information and revealing it either accidentally or intentionally. Corporations and individuals are making greater use of networks, including private networks such as *local area networks* (LANs) and *wide area networks* (WANs) and public networks such as the Internet. As a result, some of the data stored or processed on the network may require some level of protection to ensure confidentiality. Network data or software may be compromised when it is accessed, read, and possibly released to an unauthorized individual.

A common cause of unauthorized access is the failure to encrypt sensitive information. Data can be compromised by exploiting the following types of vulnerabilities:

- Storing data in the clear (i.e., unencrypted) when it is considered sensitive enough to warrant encryption
- Failing to implement, monitor, and enforce appropriate authorization and access-control mechanisms where sensitive data is stored

Unauthorized Modification of Data

Information in digital form is often shared between many users and stored on numerous shared devices. The unauthorized modification of data includes the modification, deletion, or destruction of data or software in an unauthorized or accidental manner.

A particularly insidious event is data modification that goes undetected. When such modifications are present for long periods of time, the modified data may be spread throughout the network, possibly corrupting databases, spreadsheet calculations, and other forms of application data.

This kind of damage can compromise the integrity of application information. When undetected software changes are made, all system software can become suspect, warranting a thorough review (and perhaps reinstallation) of all related software and applications.

These kinds of unauthorized changes can be made in simple command programs (for example, in PC batch files), in utility programs used on multiuser systems, in major application programs, or in any other type of software. They can be made by unauthorized outsiders as well as those who are authorized to make software changes (although not, of course, the damaging changes we are speaking of here). These changes can divert information (or copies of the information) to other destinations, corrupt the data as it is processed, and impair the availability of system or network services.

The unauthorized modification of data and software can easily take place when data integrity services are not provided.

Unauthorized Access

Unauthorized access occurs when someone who is not authorized to use a system or network gains access, usually by posing as a legitimate user of the network. Three common methods are used to gain unauthorized access: password sharing, general password guessing, and password capture.

Password sharing allows an unauthorized user to assume the network access and privileges of a legitimate user with the latter's knowledge and acceptance. General password guessing is not a new means of unauthorized access. In password capture, a legitimate user is tricked into unknowingly revealing his or her login ID and password. Methods of password capture include the use of a Trojan Horse program. To a user, this program looks like a legitimate login program; however, it's designed solely to capture passwords.

Another method used to ultimately gain network access is to capture a login ID and password as they are transmitted across the network unencrypted. A number of methods for capturing cleartext network traffic, including passwords, are readily available.

Intruders can gain unauthorized network access by exploiting the following types of vulnerabilities:

- Lack of, or insufficient, identification and authentication schemes
- Password sharing
- The use of poor password management or easy-to-guess passwords
- Failure to patch known system holes and vulnerabilities
- The storage of network access passwords in batch files on PCs
- Lack of a time-out for login and log-off attempts

Disclosure of Network Traffic

Many users realize the importance of protecting confidential information when it is stored on their workstations or servers; however, it's also important to maintain confidentiality as the information travels across the network. The disclosure of network traffic occurs when someone who is unauthorized reads, or otherwise obtains, information as it traverses the network. Intruders can easily compromise network traffic by listening to and capturing traffic transmitted over the network transport media. Examples of attack methods include tapping into a network cable with the use of a hardware device that analyzes network traffic as it is transmitted.

Traffic analyzing software, or *sniffers*, allow intruders to access the network the traffic is traversing. One such application is Sun Microsystems' "snoop" utility, which was originally created to allow administrators to verify traffic flow across the network. But it also allows intruders running the Solaris operating system to watch the flow of network traffic.

Information that can be compromised in this way includes system and user names, passwords, electronic mail messages, application data, health records, and so on. For example, even if patient records are stored on a system in an encrypted form, they can be captured in plaintext as they are sent from a workstation or PC to a file server. Electronic mail message files, which usually have strict access rights when stored on a system, are often sent in plaintext across a wire, making them an easy target for capturing.

Disclosure of network traffic is usually the result of data sent in the clear, across both public and private networks.

Spoofing of Network Traffic

It's a basic principle of network security that data that is transmitted over a network should not be altered in an unauthorized manner—either by the network itself or by an intruder—as a result of that transmission. Network users should have a reasonable expectation that any messages they send will be received unmodified. A modification occurs when an intentional or unintentional change is made to any part of the message, including the contents and addressing information.

Spoofing of network traffic involves (1) the ability to receive a message by masquerading as the legitimate receiving destination or (2) masquerading as the sending machine and sending an unauthorized message to a destination. For an attacker to masquerade as a receiving machine, the network must be persuaded that the destination address is the legitimate address of the machine. (Network traffic can also be intercepted by listening to messages as they are broadcast to all nodes.) To masquerade as the sending machine and deceive a receiver into believing the message was legitimately sent, attackers can masquerade the address or mount a playback attack. A *playback* involves capturing a session between a sender and a receiver and then retransmitting the message (with either a new header, new message contents, or both).

Intruders can spoof or modify network traffic by exploiting the following types of vulnerabilities:

- Transmitting network traffic in plaintext
- Lack of a date/time stamp (showing sending time and receiving time)
- Failure to use message authentication codes or digital signatures
- Lack of a real-time verification mechanism (to use against playback)

Identifying Intruders

Every day, undesirable intruders make unauthorized entry into computer systems and networks. Who exactly are the intruders? These individuals range from recreational hackers to foreign intelligence agencies. Each of these groups has its own agenda and motivations. The following sections paraphrase the descriptions of various intruders that were noted in a recent Federal Bureau of Investigation Congressional statement titled "Cybercrime."

Insiders

Most corporations want to believe that their employees are the cream of the crop and would never violate corporate security. In reality, however, some employees are not what they seem. People who commit security crimes against their employers are motivated by a number of reasons; the disgruntled insider (a current or former employee) is a principal source of computer crimes for many companies. Insiders' knowledge of the target company's network often allows them to gain unrestricted access and damage the system or steal proprietary data. The 2000 survey by the Computer Security Institute and FBI reports that 71 percent of respondents detected unauthorized access to systems by insiders.

Hackers

Virtually every day we see news reports about recreational hackers, or "crackers," who crack into networks for the thrill of the challenge or to gain bragging rights in the hacker community. Remote cracking once required a fair amount of skill and computer knowledge, but recreational hackers can now download attack scripts and protocols from the World Wide Web and launch them against victim sites. Thus, even though attack tools have become more sophisticated, they have also become easier to use.

Terrorists

Increasingly, terrorist groups are using new information technology and the Internet to formulate plans, raise funds, spread propaganda, and communicate securely. Moreover, some terrorist groups, such as the Internet Black Tigers (who reportedly are affiliated with the Tamil Tigers), have been known to engage in attacks on foreign government Web sites and e-mail servers. "Cyber terrorism"—by which we mean the use of cyber tools to shut down critical national infrastructures (such as energy, transportation, or government operations) for the purpose of coercing or intimidating a government or civilian population—is thus a very real, although still largely potential, threat.

Foreign Intelligence Services

Not surprisingly, foreign intelligence services have adapted to using cyber tools as part of their espionage tradecraft. As far back as 1986, before the worldwide surge in Internet use, the KGB employed West German hackers to access U.S. Department of Defense systems in the well-known "Cuckoo's Egg" case. Foreign intelligence services increasingly view computer intrusions as a useful tool for acquiring sensitive U.S. government and private sector information.

Hactivists

Recently there has been a rise in what has been dubbed "hactivism"—politically motivated attacks on publicly accessible Web pages or e-mail servers. These groups and individuals overload e-mail servers and hack into Web sites to send a political message. Although these attacks generally have not altered operating systems or networks, they damage services and deny the public access to Web sites containing valuable information; and they infringe on others' right to communicate.

One such group, the Electronic Disturbance Theater, promotes civil disobedience online in support of its political agenda regarding the Zapatista movement in Mexico and other issues. In the spring of 2000, the group called for worldwide electronic civil disobedience, and it has taken what it terms "protest actions" against White House and Department of Defense servers. Supporters of Kevin Mitnick, recently convicted of numerous computer security offenses, hacked into the Senate Web page and defaced it in May and June 2000.

The Internet has enabled new forms of political gathering and information sharing for those who want to advance social causes; that is good for the promotion of democracy worldwide. But illegal activities that disrupt e-mail servers, deface Web sites, and prevent the public from accessing information on U.S. government and private sector Web sites should be regarded as criminal acts that deny others their human rights to communicate rather than as an acceptable form of protest.

Intruder Knowledge

How have intruders gained the knowledge that allows them to commit such serious break-ins? For the most part, few of the intruder types we've discussed have extensive knowledge of the inner workings of today's computer systems. Many of these intruders do nothing more than use the information and tools built by other intruders in the past. Many Web sites provide them with all the information and tools needed to break in or damage computer systems and networks.

This doesn't mean that the information and tools downloaded by intruders were created for the purpose of aiding such attacks. On the contrary, much of this knowledge is designed to help administrators and security officers recognize potential security holes within their systems and networks—such was the case with Sun Microsystems' snoop utility, described earlier in this chapter. It's through the use of these tools, however, that intruders are able to exploit the weaknesses inherent in many systems.

Case Studies

The following case studies illustrate various ways in which security can be improperly implemented. Each example is based on an actual account of a real corporation, although the names have not been used. In general, these real-word examples demonstrate that security breaches often focus on four areas: data at rest, data in transit, authentication, and improper implementation. It is staggering how often these four elements are involved in security lapses. By examining these cases in depth, we hope to prevent these types of incidents from reoccurring.

Data in Transit

Many Web sites are still providing communications in the clear (i.e., not encrypted). As a result, they make themselves vulnerable to attackers using sniffers, who monitor and intercept clear traffic for their own purposes. Worst yet, officials in many corporate enterprises feel that their data is safe as long as it remains within their firewalls. The problem is that with many employees within the same local area network, it is easy

for an employee with sinister intentions to view, destroy, or simply manipulate all the data traveling up and down the lines.

For example, one software vendor that recently joined the ranks of the "dot-com" world allowed for the unsecured transfer of data between its internal servers. This meant that account numbers and cardholder information flowed across their internal network completely visible to any employee. This corporation, like many other corporations, felt that as long as security was provided for information flowing across the Internet, there was no need to enable internal security (behind their corporate firewall).

This particular corporation found out the hard way that there was need for internal security. It turned out an employee had been saving customer credit card numbers as they zoomed across their internal network.

When asked why, the employee simply stated that he could. What if the employee had posted the credit card numbers on the Internet (for the world to see)? If the press had gotten hold of that story, the corporation would have most likely lost many customers. What if the employee had used the credit card numbers to make purchases for himself? The credit card corporations involved might have lost faith in the merchant and cancelled their contracts. Fortunately, the corporation discovered the employee's file of saved credit card numbers before any real harm had been done.

The corporation could have avoided this predicament by enabling SSL (described in Chapter 7) and making use of secure e-mail through a protocol such as S/MIME (described in Chapter 8).

The need for security in such situations is so obvious that we honestly don't know why it is sometimes difficult for others to grasp. No corporation would have unlocked doors. We'd be willing to bet that the CEO keeps his or her possessions under lock and key, as do the company's employees. The reason is obvious: People snoop, steal, or inadvertently look at things they shouldn't.

Data at Rest

A number of corporations that provide goods and services to Internet customers actually do a great job protecting customer data in transit by making use of the SSL protocol. However, they fail to realize (or maybe they choose to forget) that data requires further protection once it's at rest. SSL does not protect data after it leaves the security of the protocol. After data is received by either the client or the server, that data is decrypted.

Nevertheless, some companies fail to adequately protect such data. One corporation, an online music vendor, recently had the misfortune of having an unauthorized "guest" break in to its systems. The attack placed more than three million credit card numbers from the company's back-end databases at risk of disclosure on the Internet. Fortunately, in this incident, it has been reported that the credit card numbers were never obtained. Still, the potential for widespread credit card fraud was there.

In news reports, the corporation's upper management stated that they didn't quite understand how the attack occurred. The company had provided for security through the use of SSL to secure connections. However, this corporation could have and should have done more in the way of security. For example, it could have encrypted the credit card numbers before placing them in the database.

Authentication

Authentication is by far one of the easiest of the security services to implement, but many corporations limit their system and network security to user ID/password schemes. Many applications, whether they reside within an enterprise or at consumer sites, incorporate nothing more than a simple password or, worse yet, a four-digit PIN.

It's easy to experience firsthand the best example of the risk incurred by companies that use inadequate authentication safeguards. All you have to do is to sit down at someone's computer who uses a certain travel-services Web site. One of the authors of this book did just that. It was a simple matter to go to the site and select the button Lost/Forgotten Password. Within one minute, the password was e-mailed directly to the user's account (which the author could easily open as well). Within all of five minutes, he could have purchased two round-trip tickets to the Caribbean. Even if an individual had to guess a password or a PIN to access the site as another person, it would take a day at most.

In the digital age, with all the information provided in this book and others like it, no new technology is needed to greatly improve authentication security. The cost of an authentication token is nothing in comparison with the money that would be lost by a fraudulent purchase at such a site.

Another example occurred recently at a medical center, where the system was hacked by an intruder who entered by using a common tool used by network administrators known as VNC (*virtual network computing*). Through the use of VNC, the intruder was able to enter the file system

and gain access to various medical records. In all, the hacker accessed more than 4,000 cardiology patient records, 700 physical rehabilitation records, and every admission, discharge, and transfer record of the medical center within a five-month period.

Without regard to internal security and the sensitivity of medical records, all this data was stored in the clear. But let's focus on the more important issue: how the network was accessed in the first place. VNC in its current incarnation has very limited authentication mechanisms (i.e., user ID and password). This means that intruders need only try a number of passwords before they gain access.

In a case concerning medical records, you can easily see the losses add up. What if this sensitive data was released publicly across the Internet? There is the obvious loss of patient confidence, as well as the very real possibility of lawsuits. Furthermore, what if the medical records were modified? While it sounds like something from a movie, this could easily happen.

With that said, proper authentication could have been observed in this case. True user ID/password schemes do provide authentication to a point, but as the sensitivity of the data increases so should the degree of authentication required. At the medical center, authentication would have been best provided for by requiring the use of client-side certificates or a one-time password token.

Implementation

Improper implementation can be seen in many examples of security breaches. The fact is that it isn't easy to implement security services using cryptography. Considerable time and effort must be taken to ensure that the newly implemented system is secure.

One well-known bug, which was recently discovered, belongs to one widely used security application, which provides encryption and digital signatures to its users. In attempting to create a new key-escrow scheme (explained further in Chapter 6) that would be less intrusive to users, the application developers made a simple error.

This simple error allowed for the possible disclosure of all data that had been encrypted using its new functionality. Furthermore, the integrity of any information, which was digitally signed by the software, could be destroyed. In this case, the actual dollar losses may never be calculated, simply because we do not know exactly when this bug was first discovered

(we would like to believe that it was announced as soon as it was found). The corporation who originally developed the software must now spend even more money fixing the problem that they created.

The entire incident could have been prevented by following existing security protocols (in this case, sticking with none key-escrow schemes). While we can appreciate the fact that the company went to the trouble of implementing a new less-intrusive concept, we feel developers should first have their work verified by an objective third party. There are a number of security consultants and agencies that test and even certify the security of products.

Information Security: Law Enforcement

Just as legal professionals are beginning to look at the legal ramifications of information security (see Chapter 10), various law enforcement agencies are studying related enforcement issues. Within the past year alone, the FBI has begun increasing the number of field agents in its *National Infrastructure Protection Center* (NIPC). Over the next two years, the number of field offices nationwide is to be increased to 56.

Within the past year, the U.S. Department of Justice has also initiated a new section devoted to investigating computer crime. The *Computer Crime and Intellectual Property Section* (CCIPS) has a staff of attorneys who advise federal prosecutors and law enforcement agents about various issues raised by computer and intellectual property crime. Furthermore, the staff provides ongoing work in the areas of e-commerce security, electronic privacy laws, and hacker investigation.

Various other agencies provide a broad range of security services. One such agency is the *Computer Emergency Response Team / Coordination Center* (CERT/CC). CERT/CC was originally created in 1988 by DARPA (the *Defense Advanced Research Projects Agency*, part of the U.S. Department of Defense) after the Morris Worm incident, which crippled 10 percent of all computers on the Internet. CERT/CC works on a number of initiatives, such as research into security vulnerabilities, improvement of system security, and coordination of teams to respond to large-scale incidents.

Summary

Efforts made to improve the security of computer networks provide benefits beyond the reduction of risks for corporations. They also play an integral role in keeping fear at bay for the benefit of everyone who uses such systems. To really see the B2B and B2C e-commerce markets take off, we are going to have to see improvements in information security.

Various risks and vulnerabilities plague all the players in the new digital world. The number of intruders, ranging from internal employees to teenage hackers who threaten computer systems, continues to grow. These intruders are becoming more knowledgeable and finding better tools that enable them to attack unsuspecting systems. Still, as the case studies from this chapter have shown, corporations and developers alike often refuse to do everything in their power to provide for proper security. Although law enforcement agencies are quickly coming up to speed with today's technology, they are simply "fighting fires" when it comes to dealing with digital attacks at this point. However, by incorporating proper security from the onset, corporations, developers, and users can prevent cybercrime before it happens.

CHAPTER 12

Doing It Right: Following Standards

A growing number of techniques are available to help organizations ensure that they've incorporated adequate security in their products and services as well as provided security for their own enterprises. Every security professional should know certain important concepts. Various standards, guidelines, and regulations have been developed, and various external agencies and organizations can be called on, to help ensure that security is implemented properly. The experiences of successful organizations can be helpful in understanding how security can be properly incorporated into everything from back-end enterprises to end-user products and services.

As you learned in Chapter 11, it seems as if there is no way around it: Sooner or later your network will be broken into. It's an excellent idea to operate under this assumption. To a casual outsider or to those who are new to the field of information security, this practice may seem a bit overboard or even a little paranoid. But security experts think this way so that they can stay ahead of the bad guys. In this chapter, you'll learn the various ways that companies are properly implementing security in the digital age.

Security Services and Mechanisms

A security service is a collection of mechanisms, procedures, and other controls that are implemented to help reduce the risk associated with the threat of data loss or compromise. Some services provide protection from threats, and other services provide for detection of the occurrences of any breach. For example, an identification and authentication service helps reduce the risk posed by access to the system by an unauthorized user. An example of a service that detects a security breach is a logging or monitoring service.

The following security services are discussed in this section:

- **Authentication** is the security service that can be used to ensure that individuals accessing the network are authorized.
- **Confidentiality** is the security service that can be used to ensure that data, software, and messages are not disclosed to unauthorized parties.
- **Integrity** is the security service that can be used to ensure that unauthorized parties do not modify data, software, and messages.
- **Nonrepudiation** is the security service that can be used to ensure that the entities involved in a communication cannot deny having participated in it. Specifically, the sending entity cannot deny having sent a message (nonrepudiation with proof of origin), and the receiving entity cannot deny having received a message (nonrepudiation with proof of delivery).

NOTE:

Though not discussed in this chapter, access control is the security service that helps ensure that network resources are being used in an authorized manner.

Authentication

The first step in securing system resources is to implement a service to verify the identities of users, a process referred to as *authentication*. Authentication provides the foundation that determines the effectiveness

of other controls used on the network. For example, a logging mechanism provides usage information based on user ID, and an access-control mechanism permits access to network resources based on the user ID. Both controls are effective only under the assumption that the requester of a network service is the valid user assigned to that specific user ID.

Identification requires that the user be known by the system or network in some manner, usually based on an assigned user ID. However, unless the user is authenticated, the system or network cannot trust the validity of the user's claim of identity. The use is authenticated by supplying something possessed only by the user (such as a token), something only the user knows (such as a password), or something that makes the user unique (such as a fingerprint). The more of these kinds of authentication that the user must supply, the less risk there is that someone can masquerade as the legitimate user.

On most systems and networks, the identification and authentication mechanism is a scheme that combines a user ID with a password. Password systems can be effective if managed properly, but they seldom are managed properly. Authentication that relies solely on passwords often fails to provide adequate protection for systems for a number of reasons. First, users tend to create passwords that are easy to remember and hence easy to guess. On the other hand, passwords generated from random characters are difficult to guess but also difficult for users to remember. As a result, users may write down such passwords, and they are often found in areas that are easy accessible. It's not unusual, for example, to find passwords written on sticky notes mounted on computer monitors, where anyone can find them and use them to gain access to the network. The guessing of passwords is a science, and a great deal of research has been published that details the ease with which passwords can be guessed.

Proper password selection—striking a balance between the password being easy to remember for the user but difficult to guess for everyone else-has always been an issue. Password generators have been developed that produce passwords consisting of pronounceable syllables. Such passwords have greater potential of being remembered than those made of purely random characters. Some systems and network administrators require the use of an algorithm that produces random pronounceable passwords. Programs called *password checkers* enable a user to determine whether a new password is considered easy to guess and thus unacceptable.

Because of the vulnerabilities that still exist with the use of password-only mechanisms, more robust mechanisms can be used, such as token-based authentication or biometrics. A smart card–based or token-based

mechanism requires that a user be in possession of the token and additionally may require the user to know a PIN or password. These devices then perform a challenge/response authentication scheme using real-time parameters. The latter practice helps prevent an intruder from gaining unauthorized access through a login session playback. These devices may also encrypt the authentication session, preventing compromise of the authentication information through monitoring and capturing.

Locking mechanisms can be used for network devices, workstations, or PCs, requiring user authentication to unlock. These tools can be useful when users must leave their work areas frequently. These locks allow users to remain logged in to the network and leave their work areas (for an acceptably short period of time) without exposing an entry point into the network.

Confidentiality

Because access control through the use of proper authentication is not always possible (because of shared drives and open networks), data confidentiality services can be used when it's necessary to protect the secrecy of information. The use of encryption through symmetric or asymmetric ciphers (or both) can reduce the risk of unauthorized disclosure, both in the case of data at rest and data in transit, by making it unreadable to those who may capture it. Only the authorized user who has the correct key can decrypt the data.

Integrity

Data integrity services provide protection against intentional and accidental unauthorized modification of data. This service can be used for data while it is at rest on a back-end database or while it is in transit across a network. This service can be provided by the use of cryptographic checksums and highly granular access-control and privilege mechanisms. The more granular the access-control or privilege mechanism, the less likely it is that an unauthorized or accidental modification can occur.

Furthermore, data integrity services help to ensure that a message is not altered, deleted, or added to in any manner during transmission across a network. Most available security techniques cannot prevent the modification of a message, but they can detect that a message has been modified (unless the message is deleted altogether).

Nonrepudiation

Nonrepudiation helps to ensure that the entities in a communication cannot deny having participated in all or part of the communication. When a major function of the network is electronic mail, this service becomes crucial. Nonrepudiation services can be provided through the use of public-key cryptographic techniques using digital signatures.

Standards, Guidelines, and Regulations

Throughout this book, we've described a number of standards, guidelines, and regulations. For example, Chapter 6 explains how the X.509 standard can be used to provide for secure public-key operations, and Chapter 7 describes the SSL and IPSec protocols, which are used to provide various security services. The following sections outline the various organizations that have made the effort to ensure that each standard, guideline, and regulation provides for a proper security implementation.

The Internet Engineering Task Force

The *Internet Engineering Task Force* (IETF) is an international community of network designers, operators, vendors, and researchers. This group is concerned with the smooth operation of the Internet and the evolution of the Internet architecture.

The technical work of the IETF is done in its *working groups,* which are organized by topic into several areas (routing, transport, security, and so on). The working groups are managed by area directors (ADs), who are members of the *Internet Engineering Steering Group* (IESG). Providing architectural oversight is the *Internet Architecture Board* (IAB). The IAB also adjudicates appeals when someone complains about the policies adopted by the IESG. The IAB and IESG are chartered by the *Internet Society* (ISOC) for these purposes. The general area director also serves as the chair of the IESG and of the IETF and is an ex officio member of the IAB.

ANSI X9

X9 is a division of the *American National Standards Institute* (ANSI) that develops and publishes voluntary, consensus technical standards for the financial services industry. X9's voting membership includes more than 300 organizations representing investment managers, banks, software and equipment manufacturers, printers, credit unions, depositories, government regulators, associations, consultants, and others.

X9 develops standards for check processing, electronic check exchange, PIN management and security, the use of data encryption, and wholesale funds transfer, among others. Standards under development include electronic payments via the Internet, financial image interchange, home banking security requirements, institutional trade messages, and electronic benefits transfer.

X9's procedures ensure that interested parties have an opportunity to participate and comment on a developing standard before it is implemented. X9 standards are also reviewed by ANSI before publication to ensure that all requirements are met. ANSI conducts an audit of X9 operations every five years.

X9 is organized into seven subcommittees. At any given time the committee has 20 to 30 active working groups and more than 80 domestic and international standards projects. Organizations that vote on more than one subcommittee constitute a parent committee that sets policy and procedures.

National Institute of Standards and Technology

The *National Institute of Standards and Technology* (NIST) has published many guidelines and standards on the topic of information security. One of its key contributions to cryptography is the federal information-processing standard (FIPS 140-1), which describes a standard for secure cryptographic modules. FIPS 140-1 is discussed more fully in the following section.

NIST also administers a certification process for software and hardware cryptographic modules.

FIPS 140-1

FIPS 140-1 specifies the security requirements that are to be satisfied by a cryptographic module that is used in a security system protecting unclassified information in computer and telecommunication systems. Cryptographic modules conforming to this standard must meet the applicable security requirements described in the standard.

The FIPS 140-1 standard was developed by a working group composed of users and vendors and including government and industry participants. To provide for a wide spectrum of data sensitivity (such as low-value administrative data, large funds transfers, and data related to human life and safety) and a diversity of application environments (such as a guarded facility, an office, and a completely unprotected location), the working group identified requirements for four security levels for cryptographic modules. Each security level offers an increase in security over the preceding level. These four increasing levels of security are designed to support cost-effective solutions that are appropriate for different degrees of data sensitivity and different application environments.

Although the security requirements specified in this standard are intended to maintain the security of a cryptographic module, conformance to this standard does not guarantee that a particular module is secure. It is the responsibility of the manufacturer of a cryptographic module to build the module in a secure manner.

Similarly, the use of a cryptographic module that conforms to this standard in an overall system does not guarantee the security of the overall system. The security level of a cryptographic module should be chosen to provide a level of security that's appropriate to the security requirements of the application, the environment in which the module is to be used, and the security services that the module is to provide. The responsible authority in each agency or department must ensure that the agency or department's relevant computer or telecommunication systems provide an acceptable level of security for the given application and environment.

NIST emphasizes the importance of computer security awareness and of making information security a management priority that is communicated to all employees. Because computer security requirements vary among applications, organizations should identify their information resources and determine the sensitivity to and potential impact of losses. Controls should be based on the potential risks. Available controls include

administrative policies and procedures, physical and environmental controls, information and data controls, software development and acquisition controls, and backup and contingency planning.

NIST has developed many of the needed basic controls to protect computer information and has issued standards and guidelines covering both management and technical approaches to computer security. These include standards for cryptographic functions that are implemented in cryptographic modules as specified in the FIPS 140-1 standard. This standard is expected to be the foundation for NIST's current and future cryptographic standards.

Common Criteria

A standard known as *Common Criteria* (CC) was developed as the result of a series of international efforts to develop criteria for evaluation of information security. It began in the early 1980s, when the *Trusted Computer System Evaluation Criteria* (TCSEC) was developed in the United States. Ten years later, a European standard, the *Information Technology Security Evaluation Criteria* (ITSEC), was built on the concepts of the TCSEC. Then in 1990, the *International Organization for Standardization* (ISO) sought to develop a set of international evaluation criteria for general use. The CC project was started in 1993 in order to bring these (and other) efforts together into a single international standard for information security evaluation.

The CC aims to build consumer confidence by testing and certifying products and services. Typically, certifiers are commercial organizations operating testing laboratories accredited by ISO. Accreditors are sometimes closely involved in the determination of functional and assurance requirements for a system.

The Health Insurance Portability Act

Various regulations have been enacted at the local, state, and federal levels, each of them specifying unique requirements for the various market sectors. The *Health Insurance Portability Act* (HIPAA) is one such regulation. Handed down by the federal government and signed into law in 1996, HIPAA addresses both health insurance reform and administrative simplification. The latter section aims to standardize access to patient

records and the transmission of electronic health information between organizations. Important among these administrative issues is the proposed standard for security and electronic signatures, which mandates requirements for the following:

- **Confidentiality** This requirement is designed to keep all transfers of information private. Steps must be taken to ensure that information is not made available or disclosed to unauthorized individuals.
- **Integrity** This requirement ensures that data has not been changed or altered en route or in storage.
- **Authentication** This mandate means that organizations must make sure that the person sending the message is the person he or she claims to be.
- **Nonrepudiation** This principle ensures that after a transaction occurs, neither the originator nor the recipient can deny that it took place.
- **Authorization** This requirement limits access to network information and resources to users who have been authenticated based on defined privileges.

When it comes to people's most personal information—their medical history—people have always been concerned about confidentiality. As individuals, we are secretive about every aspect of our health, from weight to illness to prescriptions to payments. When it comes to medical information, it is essential that healthcare providers implement PKI security infrastructures so that they can use digital certificates to securely store, transmit, and access health records electronically. The thought of personal information on a public network can be intimidating, and patients need to be assured that their private medical histories continue to maintain unparalleled confidentiality.

Developer Assistance

Many software developers, whether they're implementing security for a retail product or for the newest e-commerce site or they're building applications for an enterprise, have chosen to outsource the security task. A good number of security consultants, architects, and managed security

services are available for hire. These individuals and companies provide a wide range of assistance in implementing security, allowing developers to spend more time working on the unique aspects of their products (where their talents are best used).

RSA Security, Inc., is one professional services organization that can assist developers in the design and implementation of security. In addition, many organizations, including RSA, can also provide certification when the system is completed.

Insurance

Many financial institutions and insurance companies have now begun insuring e-commerce Web sites. Many of these new insurers look for certification and require security audits. One way to implement insurance requirements is through the use of secure cryptographic modules (such as BSAFE, a software product family line available from RSA Security, Inc.).

American International Group, Inc. (AIG), one such insurance provider, has e-business divisions that offer insurance to companies with e-business initiatives. RSA Security, Inc., and AIG partnered in January 2000 to provide e-security to corporations. The partnership means that customers can take advantage of discounts offered by AIG for the use of RSA products.

Security Research

Sun Tzu stated it perfectly in *The Art of War*: "Know your enemy as you know yourself, and in a thousand battles you shall never perish." For organizations that are designing and implementing security systems, the most successful approach is to learn what intruders know so that you can come up with a way to stop them. The following list of Web sites is an excellent place to start to learn what the enemy knows.

http://www.cert.org/	CERT/CC is a center of Internet security expertise.
http://www.securityfocus.com/	Security Focus provides up-to-date information on current bugs. It also keeps an excellent collection of hacker-related articles.

http://www.slashdot.org/	This site provides information about current break-ins and describes the various known system vulnerabilities.
http://www.2600.org/	This site dubs itself the "Hacker Quarterly," providing its readers with current information on system hacks and cracks.

NOTE:
This list is by no means complete; it is merely a starting place where you can gain knowledge about security issues.

Case Studies

In contrast to the case studies presented in Chapter 11, the following case studies illustrate the ways in which corporations and developers took steps to properly implement security, focusing specifically on their techniques within the four commonly ignored areas described in Chapter 11: data at rest, data in transit, authentication, and implementation. Each case study shows how time and money can be saved by properly implementing security *before* an incident arises.

Implementation

One major hardware manufacturer recently looked into implementing a public-key infrastructure. After close analysis, company officials realized that not all the applications that the company needed secured were "PKI-ready;" that is, some of the applications did not support the use of public-key certificates. After working with various security architects and consultants, company officials learned that for this set of applications they need to provide a front-end server application that handled certificates. The employees already had one of the easiest-to-use PKI clients: a Web browser.

In this case, there is still a chance that sensitive data might be exposed at points where the new server application communicates with the legacy applications. However, the level of security at this company is now significantly higher than it was before the PKI was established.

Even if their systems and networks are never breached, this company has saved a significant amount of both time and money. Each of their legacy applications required a password to access, which meant they needed a fully staffed help desk to assist users with logging in and resetting forgotten passwords. Another cost advantage came through the use of digital signatures on electronic ordering forms, which are now legally binding (see Chapter 10).

Authentication

A California city government recently discovered that a typical city employee had to establish and memorize six to nine passwords to access various applications. With this number of passwords, city officials realized that time and money were being wasted on administering the effort to deal with lost and forgotten passwords. For those users who weren't having such problems, it was probably because they had written down the passwords next to their computer terminals, creating a serious security hazard.

The city quickly realized that they needed to reduce the average number of passwords required, while at the same time increasing security. After seeking assistance from various security groups and reviewing a variety of products, city officials decided on using biometrics. Through the use of biometric technologies, this city provided fingerprint scanners at each of its computer terminals, thereby eliminating the need for multiple passwords. As a result, trouble calls are down substantially, amounting in considerable savings. At the same time, concerns that sensitive data might be disclosed have decreased significantly.

Another example of a company making authentication more secure occurred in the banking industry. Think of how often you walk up to your ATM machine, insert your card, enter your PIN, and perform a transaction. And consider how many other PINs you may have, for example, for your brokerage account or for accounts with other banks. The more PINs you have the easier they are to forget, and having just one PIN for all your accounts puts you at risk. One major bank realized that they were spending a considerable amount of time and money on customers who were for-

getting their PIN number. Still more time and money were lost because of fraud (it is not uncommon for bank users to write their PIN code down in their wallet, or worse yet, give the code to a friend who performs a transaction for them.)

Realizing the need for an authentication alternative, the bank searched out security professionals and reviewed various software and hardware packages. After a thorough look at the costs and return on investment, the bank has decided implement a biometric system. Currently they are testing both finger and iris scanners.

The actual dollar amount saved in decreased fraud was not disclosed; however, you can only imagine how much the bank lost each year. Furthermore, the costs of keeping a customer help desk to reset PINs as needed has decreased significantly.

Data at Rest

Prior to last year; one federal law enforcement agency, which works closely with the relocation of witnesses, had not provided any true security for their field agent's laptops. Imagine for a moment what exactly might happen if one of these laptops were lost or stolen?

After realizing the possibility of confidential data possibly being exposed if one of these laptops were compromised, the agency added security software that encrypts and decrypts files as needed. Furthermore, the software requires strong, two-factor authentication through the use of a one-time password token.

By simply taking the time to add strong security to each of these laptops, this agency has more than likely saved not only time and money, but possibly even lives.

Another example, although not as dramatic as the first one, involves a major U.S. airline using encryption to provide confidentiality services to its back-end databases. What is unique about this case is that the encryption is applied not only to user information required for reservations, but also to all information pertaining to users' frequent flyer mileage accounts.

After the company began to recognize just how much these miles were worth once they started adding up, they decided it was time to provide security. Through the use of a symmetric-key algorithm, their customers' information as well as the miles they had earned were safely secured.

You might think that customer information or frequent flyer miles are not the most sensitive data in the world. However, this company realized

that if their customer accounts, especially corporate accounts, were ever disclosed publicly on the Internet it could mean financial disaster—not to mention the cost of paperwork if a customer's reservation was changed or removed from the database.

Data in Transit

One Canadian-based software corporation was providing a product to pharmacists and doctors that allowed for quicker prescription fulfillment. Their product simply transmitted the necessary patient data and prescription from the physician to the pharmacy. However, until recently, the data was not encrypted during transmission. The company felt that because the data was traveling across a dial-up phone line it was unnecessary to provide security.

The software corporation quickly realized that even with a dial-up phone connection, security was a necessity. However, the company was afraid that if they were to add security, that doctors and pharmacists might have difficulty using it (after all, doctors know medicine, not security). After speaking with outside security professionals, the corporation decided on the integration of the SSL protocol (discussed in Chapter 7) to provide security. The SSL protocol is virtually transparent to the end-users eliminating the difficulty factor.

The software company now knows they are selling a product that not only can protect the data as it is communicated, but also, through the use of SSL, can guarantee the data was not changed in transit. Furthermore, their market has now expanded to cover not only physicians and pharmacies using dial-up lines, but also those that use open networks (such as the Internet).

Summary

At first glance, it may appear that protecting the security of data in a network is a losing battle, but many developers, enterprises, and users have been successful in achieving this goal. This chapter tells only a few of many success stories in this arena.

As these case studies illustrate, security is a battle that you should not face alone. Instead, you should take advantage of the expertise of other

professionals. Here you've seen a number of ways in which security can be ensured through the use of existing standards, protocols, algorithms, and assistance from consultants who have been shown to be of great help. Through the certification process, many users can ensure that their security methods (especially cryptography) are as well designed and well implemented as possible. It is sometimes useful to consult with a trusted third party or external agency to gain a different point of view.

Security is an important issue for businesses and other organizations that are entrusted with personal data. In the long run, everyone benefits when consumers can have faith in the technology that enables not only the efficient storage of data, but also the potential for unprecedented communication and human growth.

APPENDIX A

Bits, Bytes, Hex, and ASCII

Throughout this book, we show data as hexadecimal (often shortened to "hex") numbers. Even if the data is a series of letters, it can be represented in hexadecimal numbers. This appendix describes bits, bytes, and hexadecimal numbers and explains how ASCII characters are formed.

Using Decimal, Binary, and Hexadecimal Numbers

A computer is a *binary* machine; everything is either "on" or "off," reflecting the fact that electric power is either flowing or not flowing through a given circuit. The machine can be programmed to interpret the state of being on or off as a 0 or a 1. If you string these 0s and 1s together, you can represent anything. For example, you can represent the decimal number 105 as the binary number 1101001.

To see how binary numbers work, it might be helpful to recall how decimal numbers work. A decimal number is composed of some number of "ones," "tens," "hundreds," and so on. As Figure A-1 shows, if you start

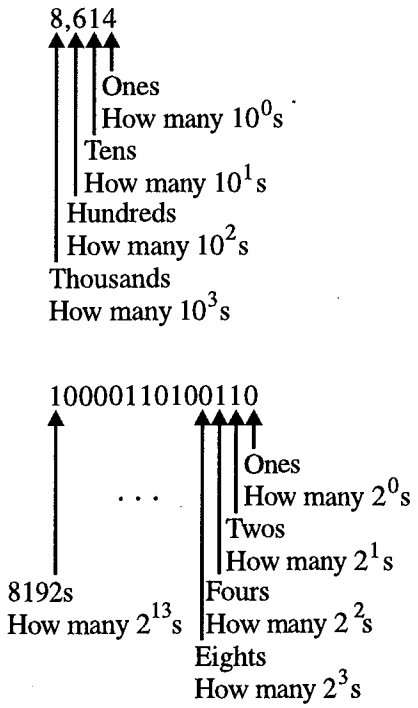


Figure A-1

The number 8,614 is decimal, comprising four "ones," one "ten," six "hundreds," and eight "thousands." The number 10000110100110 is the binary equivalent. The numerals in the positions indicate the number of powers of 1, 2, 4, 8, and so on

counting at zero and move from right to left, each place in a decimal number represents a number of 10^ns (10 to the nth power). A computer does the same thing except that it uses "two" as its base instead of "ten." For each place in a binary number, there can be only two possible values: a 0 or a 1. In Figure A-1, the binary number is computed if you start counting at zero and move from right to left to see how many "ones," how many "twos," how many "fours," and so on, each of which represents a power of 2.

Any value that can be represented as a decimal number can also be represented as a binary number. The binary number will take up more space, but any value can be expressed. Suppose you wanted to use the decimal number 2,535,294,694. A computer would "think" of it as binary, which would look like this:

```
1001 0111 0001 1101 1000 0110 1110 0110
```

Writing such numbers can be tedious, so programmers use hexadecimal as a convenience. The word "decimal" has to do with "ten" ("dec" means "ten," as in "decade" or "decathlon"), whereas "binary" refers to "two" ("bi" means "two," as in "bicycle" or "bifocals"), and the word "hexadecimal" refers to "sixteen." So binary numbers are "base two," decimal

numbers are "base ten," and hexadecimal numbers are "base sixteen." The digits used in binary numbers are

```
0 1
```

The digits used in decimal numbers are

```
0 1 2 3 4 5 6 7 8 9
```

And the digits used in hexadecimal numbers are

```
0 1 2 3 4 5 6 7 8 9 A B C D E F
```

Notice that each system has the same number of digits as what we've referred to as the "base." Base two uses two digits, base ten uses ten digits, and base sixteen uses sixteen digits. Table A-1 is a conversion table for numbers in the three bases.

Table A-1 Binary, Decimal, and Hexadecimal Equivalents

Base Two	Base Ten	Base Sixteen
0	0	0
1	1	1
10	2	2
11	3	3
100	4	4
101	5	5
110	6	6
111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Hexadecimal is convenient because any group of four binary digits (a binary digit is also known as a bit) can be represented in one hexadecimal digit. This means that you can take a big binary number, break it into groups of four digits, and rewrite it in hexadecimal. For example,

```
1001 0111 0001 1101 1000 0110 1110 0110
```

can be rewritten in hex as

```
0x97 1D 86 E6
```

The 1001 has been converted to 9 (see Table A-1), the 0111 has become 7, and so on. The 0x at the front is a notational convention indicating that the number is in hexadecimal.

It's possible to think of that number as "six 1s," "fourteen 16s," "six 256s," and so on. You start counting at zero and move from right to left, and each place represents the number of 16^ns.

Using Bits and Bytes

If you put eight bits together, you have one byte. The word "byte" is simply the technical jargon for a group of eight bits. For example, 1001 0111 is a byte. It can also be represented as 0x97, so two hex digits make up one byte. The number 0x97 1D 86 E6 is 32 bits, or four bytes. A byte is also a measure of space. If your computer has 1MB of memory, that means it has space enough to load one million bytes of data into memory. A byte can have 256 possible values, from 0 to 255 (0x00 to 0xFF).

NOTE:

Actually, a megabyte is 1,048,576 bytes, which is 2^{20}. Most quantities of things in the computer industry come in powers of 2, either for technical reasons or simply because. Computers are binary machines, so hardware constraints may dictate that you use a power of 2, and in software, working with numbers that are powers of 2 can often be more convenient than using other numbers. But sometimes the only reason to use a power of 2 is that a task is being computerized. For example, in cryptography, 104-bit symmetric keys are secure enough, but that number is not a power of 2. So people use 128-bit keys. There's no technical cryptographic reason to use 128 bits instead of 104, but 128 is a power of 2. People working in computer science sometimes choose numbers simply because they are powers of 2.

Using ASCII Characters

A computer chip can interpret only 1s and 0s and therefore does not have a native way to represent letters of the alphabet. So in the 1960s, at the beginning of the computer age, the American Standards Association, calling on the contributions of computer manufacturers, programmers, and others, came up with a standard way to represent letters as numbers. Because *A* is the first letter of the English alphabet, it could have been assigned the number 0x01; *B* could have been 0x02, and so on. It could have been, but that's not what the committee chose. The people involved were interested in representing more than just letters of the alphabet. They knew that computers would also need to interpret numerals, math symbols, and punctuation marks. In addition, they would need uppercase as well as lowercase letters.

Eventually, a standard was developed specifying that the bytes 0x20 through 0x7F would be used to represent the English alphabet, numerals, certain symbols, and certain punctuation marks. That's 96 standard characters. Table A-2 shows the values and their characters. The standard is called ASCII (pronounced ASK-ee), an acronym for *American Standard Code for Information Interchange*.

As it turned out, the original 96 ASCII characters were not sufficient because some languages had special marks on their letters (called *diacritical* marks), such as the umlaut (two dots) in *ü* or the cedilla (the squiggle on the bottom) in *ç*. Other languages had larger alphabets. In addition, values for more punctuation marks and control characters were needed. Over the years, standards committees have generated additional character sets. A byte can have only 256 possible values, and that is not enough space to hold all the possible characters. As a result, some of the new standards define characters in two bytes, allowing definition of as many as 65,536 characters. Other standards specify four bytes per character, giving space for more than four million characters. Most character sets include the original ASCII values along with the added values.

All this means that if a computer is operating on the expression 0x52 53 41 53, it could be the hex representation of the decimal number 1,381,187,923, or it could be the letters *RSAS*.

Table A-2

The Core 96-Member ASCII Character Set

20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

The character 0x20 is space, as in the space between two words. The character 0x7F is the DELETE key.

Using Computers in Cryptography

Keep in mind that to a computer, numbers can represent many kinds of meanings. For example, if a computer is looking at 0x42 A0 10 07, it might be looking at the decimal number 1,117,786,119, or those bits might mean something else. It could be an instruction, for example. A computer program is a series of instructions, and because a computer stores everything as binary numbers, instructions, too, can look like numbers. Moreover, each chip has its own instruction set, so a number on one computer may mean one instruction but on another computer may mean something else. For example, on one machine the bits 0x42 A0 10 07 might be the computer's way of saying, "Add the contents of register 16 to the contents of register 7 and store the result in register 7." Other values could represent memory addresses, other control characters, or some sort of data in another format.

In cryptography, though, these kinds of values are simply bytes and numbers. So when we talk about plaintext, we really mean bytes of data, no matter what meaning the owner of the data attributes to the bytes. A crypto algorithm looks at the data as bits to manipulate or numbers to crunch. Cryptography makes no distinction between bytes that represent letters of the alphabet and bytes that indicate instructions in a program. They are simply bytes.

APPENDIX B

A Layman's Guide to a Subset of ASN.1, BER, and DER

An RSA Laboratories Technical Note
Burton S. Kaliski Jr.
Revised November 1, 1993

NOTE:
This document supersedes June 3, 1991 version, which was also published as NIST/OSI Implementors' Workshop document SEC-SIG-91-17. PKCS documents are available by electronic mail to <**pkcs@rsa.com**>.

Abstract

This note gives a layman's introduction to a subset of OSI's Abstract Syntax Notation 1 (ASN.1), Basic Encoding Rules (BER), and Distinguished Encoding Rules (DER). The particular purpose of this note is to provide background material sufficient for understanding and implementing the PKCS family of standards.

Section 1: Introduction

It is a generally accepted design principle that abstraction is a key to managing software development. With abstraction, a designer can specify a part of a system without concern for how the part is actually implemented or represented. Such a practice leaves the implementation open; it simplifies the specification; and it makes it possible to state "axioms" about the part that can be proved when the part is implemented, and assumed when the part is employed in another, higher-level part. Abstraction is the hallmark of most modern software specifications.

One of the most complex systems today, and one that also involves a great deal of abstraction, is Open Systems Interconnection (OSI, described in X.200). OSI is an internationally standardized architecture that governs the interconnection of computers from the physical layer up to the user application layer. Objects at higher layers are defined abstractly and intended to be implemented with objects at lower layers. For instance, a service at one layer may require transfer of certain abstract objects between computers; a lower layer may provide transfer services for strings of 1's and 0's, using encoding rules to transform the abstract objects into such strings. OSI is called an open system because it supports many different implementations of the services at each layer.

OSI's method of specifying abstract objects is called ASN.1 (Abstract Syntax Notation 1, defined in X.208), and one set of rules for representing such objects as strings of 1's and 0's is called the BER (Basic Encoding Rules, defined in X.209). ASN.1 is a flexible notation that allows one to define a variety data types, from simple types such as integers and bit strings to structured types such as sets and sequences, as well as complex types defined in terms of others. BER describes how to represent or encode values of each ASN.1 type as a string of eight-bit octets. There is generally more than one way to BER-encode a given value. Another set of rules, called the Distinguished Encoding Rules (DER), which is a subset of BER, gives a unique encoding to each ASN.1 value.

The purpose of this note is to describe a subset of ASN.1, BER, and DER sufficient to understand and implement one OSI-based application, RSA Data Security, Inc.'s Public-Key Cryptography Standards. The features described include an overview of ASN.1, BER, and DER and an abridged list of ASN.1 types and their BER and DER encodings. Sections 2-4 give an overview of ASN.1, BER, and DER, in that order. Section 5 lists some ASN.1 types, giving their notation, specific encoding rules,

examples, and comments about their application to PKCS. Section 6 concludes with an example, X.500 distinguished names.

Advanced features of ASN.1, such as macros, are not described in this note, as they are not needed to implement PKCS. For information on the other features, and for more detail generally, the reader is referred to CCITT Recommendations X.208 and X.209, which define ASN.1 and BER.

Section 1.1: Terminology and Notation

In this note, an octet is an eight-bit unsigned integer. Bit 8 of the octet is the most significant, and bit 1 is the least significant.

The following meta-syntax is used in describing ASN.1 notation:

`BIT`	Monospace denotes literal characters in the type and value notation; in examples, it generally denotes an octet value in hexadecimal
n_1	Bold italics denotes a variable
[]	Bold square brackets indicate that a term is optional
{}	Bold braces group related terms
|	Bold vertical bar delimits alternatives within a group
...	Bold ellipsis indicates repeated occurrences
=	Bold equals sign expresses terms as subterms

Section 2: Abstract Syntax Notation 1

Abstract Syntax Notation 1, abbreviated ASN.1, is a notation for describing abstract types and values.

In ASN.1, a type is a set of values. For some types, there are a finite number of values, and for other types there are an infinite number. A value of a given ASN.1 type is an element of the type's set. ASN.1 has four kinds of types: simple types, which are "atomic" and have no components; structured types, which have components; tagged types, which are derived from other types; and other types, which include the `CHOICE` type and the `ANY` type. Types and values can be given names with the ASN.1 assignment operator (`::=`), and those names can be used in defining other types and values.

Every ASN.1 type other than CHOICE and ANY has a tag, which consists of a class and a nonnegative tag number. ASN.1 types are abstractly the same if and only if their tag numbers are the same. In other words, the name of an ASN.1 type does not affect its abstract meaning; only the tag does. There are four classes of tags:

1. *Universal,* for types whose meaning is the same in all applications; these types are only defined in X.208.
2. *Application,* for types whose meaning is specific to an application, such as X.500 directory services; types in two different applications may have the same application-specific tag and different meanings.
3. *Private,* for types whose meaning is specific to a given enterprise.
4. *Context-specific*, for types whose meaning is specific to a given structured type; context-specific tags are used to distinguish between component types with the same underlying tag within the context of a given structured type, and component types in two different structured types may have the same tag and different meanings.

The types with universal tags are defined in X.208, which also gives the types' universal tag numbers. Types with other tags are defined in many places, and are always obtained by implicit or explicit tagging (see Section 2.3). Table B-1 lists some ASN.1 types and their universal-class tags.

ASN.1 types and values are expressed in a flexible, programming-language-like notation, with the following special rules:

- Layout is not significant; multiple spaces and line breaks can be considered as a single space.
- Comments are delimited by pairs of hyphens (--), or a pair of hyphens and a line break.
- Identifiers (names of values and fields) and type references (names of types) consist of upper- and lowercase letters, digits, hyphens, and spaces; identifiers begin with lowercase letters; type references begin with uppercase letters.

The following four subsections give an overview of simple types, structured types, implicitly and explicitly tagged types, and other types. Section 5 describes specific types in more detail.

Table B-1 Some Types and Their Universal-Class Tags

Type	Tag Number (Decimal)	Tag Number (Hexadecimal)
INTEGER	2	02
BIT STRING	3	03
OCTET STRING	4	04
NULL	5	05
OBJECT IDENTIFIER	6	06
SEQUENCE and SEQUENCE OF	16	10
SET and SET OF	17	11
PrintableString	19	13
T61String	20	14
IA5String	22	16
UTCTime	23	17

Section 2.1: Simple Types

Simple types are those not consisting of components; they are the "atomic" types. ASN.1 defines several; the types that are relevant to the PKCS standards are the following:

- BIT STRING, an arbitrary string of bits (1's and 0's)
- IA5String, an arbitrary string of IA5 (ASCII) characters
- INTEGER, an arbitrary integer
- NULL, a null value
- OBJECT IDENTIFIER, an object identifier, which is a sequence of integer components that identify an object such as an algorithm or attribute type
- OCTET STRING, an arbitrary string of octets (eight-bit values)
- PrintableString, an arbitrary string of printable characters

- `T61String`, an arbitrary string of T.61 (eight-bit) characters
- `UTCTime`, a "coordinated universal time" or Greenwich Mean Time (GMT) value

Simple types fall into two categories: string types and nonstring types. `BIT STRING`, `IA5String`, `OCTET STRING`, `PrintableString`, `T61String`, and `UTCTime` are string types.

String types can be viewed, for the purposes of encoding, as consisting of components, where the components are substrings. This view allows one to encode a value whose length is not known in advance (e.g., an octet string value input from a file stream) with a constructed, indefinite-length encoding (see Section 3).

The string types can be given size constraints limiting the length of values.

Section 2.2: Structured Types

Structured types are those consisting of components. ASN.1 defines four, all of which are relevant to the PKCS standards:

1. `SEQUENCE`, an ordered collection of one or more types
2. `SEQUENCE OF`, an ordered collection of zero or more occurrences of a given type
3. `SET`, an unordered collection of one or more types
4. `SET OF`, an unordered collection of zero or more occurrences of a given type

The structured types can have optional components, possibly with default values.

Section 2.3: Implicitly and Explicitly Tagged Types

Tagging is useful to distinguish types within an application; it is also commonly used to distinguish component types within a structured type. For instance, optional components of a `SET` or `SEQUENCE` type are typically given distinct context-specific tags to avoid ambiguity.

There are two ways to tag a type: implicitly and explicitly.

- Implicitly tagged types are derived from other types by changing the tag of the underlying type. Implicit tagging is denoted by the ASN.1 keywords [***class number***] IMPLICIT (see Section 5.1).
- Explicitly tagged types are derived from other types by adding an outer tag to the underlying type. In effect, explicitly tagged types are structured types consisting of one component, the underlying type. Explicit tagging is denoted by the ASN.1 keywords [***class number***] EXPLICIT (see Section 5.2).

The keyword [***class number***] alone is the same as explicit tagging, except when the "module" in which the ASN.1 type is defined has implicit tagging by default. ("Modules" are among the advanced features not described in this note.)

For purposes of encoding, an implicitly tagged type is considered the same as the underlying type, except that the tag is different. An explicitly tagged type is considered like a structured type with one component, the underlying type. Implicit tags result in shorter encodings, but explicit tags may be necessary to avoid ambiguity if the tag of the underlying type is indeterminate (e.g., the underlying type is CHOICE or ANY).

Section 2.4: Other Types

Other types in ASN.1 include the CHOICE and ANY types. The CHOICE type denotes a union of one or more alternatives; the ANY type denotes an arbitrary value of an arbitrary type, where the arbitrary type is possibly defined in the registration of an object identifier or integer value.

Section 3: Basic Encoding Rules

The Basic Encoding Rules (BER) for ASN.1 give one or more ways to represent any ASN.1 value as an octet string. (There are certainly other ways to represent ASN.1 values, but BER is the standard for interchanging such values in OSI.)

There are three methods to encode an ASN.1 value under BER, the choice of which depends on the type of value and whether the length of the

value is known. The three methods are primitive, definite-length encoding; constructed, definite-length encoding; and constructed, indefinite-length encoding. Simple nonstring types employ the primitive, definite-length method; structured types employ either of the constructed methods; and simple string types employ any of the methods, depending on whether the length of the value is known. Types derived by implicit tagging employ the method of the underlying type, and types derived by explicit tagging employ the constructed methods.

In each method, the BER encoding has three or four parts:

1. **Identifier octets** These identify the class and tag number of the ASN.1 value, and indicate whether the method is primitive or constructed.
2. **Length octets** For the definite-length methods, these give the number of contents octets. For the constructed, indefinite-length method, these indicate that the length is indefinite.
3. **Contents octets** For the primitive, definite-length method, these give a concrete representation of the value. For the constructed methods, these give the concatenation of the BER encodings of the components of the value.
4. **End-of-contents octets** For the constructed, indefinite-length method, these denote the end of the contents. For the other methods, these are absent.

The three methods of encoding are described in the following sections.

Section: 3.1: Primitive, Definite-Length Method

This method applies to simple types and types derived from simple types by implicit tagging. It requires that the length of the value be known in advance. The parts of the BER encoding are as follows:

Identifier Octets

There are two forms: low tag number (for tag numbers between 0 and 30) and high tag number (for tag numbers 31 and greater).

Low-Tag-Number Form One octet. Bits 8 and 7 specify the class (see Table B-2), bit 6 has value "0," indicating that the encoding is primitive, and bits 5–1 give the tag number.

Table B-2 Class Encoding in Identifier Octets

Class	Bit 8	Bit 7
universal	0	0
application	0	1
context-specific	1	0
private	1	1

High-Tag-Number Form Two or more octets. First octet is as in low-tag-number form, except that bits 5–1 all have value "1." Second and following octets give the tag number, base 128, most significant digit first, with as few digits as possible, and with the bit 8 of each octet except the last set to "1."

Length Octets

There are two forms: short (for lengths between 0 and 127), and long definite (for lengths between 0 and $2^{1008} - 1$).

Short Form One octet. Bit 8 has value "0" and bits 7–1 give the length.

Long Form Two to 127 octets. Bit 8 of first octet has value "1" and bits 7–1 give the number of additional length octets. Second and following octets give the length, base 256, most significant digit first.

Contents Octets

These give a concrete representation of the value (or the value of the underlying type, if the type is derived by implicit tagging). Details for particular types are given in Section 5.

Section 3.2: Constructed, Definite-Length Method

This method applies to simple string types, structured types, types derived from simple string types and structured types by implicit tagging, and types derived from anything by explicit tagging. It requires that the length of the value be known in advance. The parts of the BER encoding are as follows.

Identifier Octets

As described in Section 3.1, except that bit 6 has value "1," indicating that the encoding is constructed.

Length Octets

As described in Section 3.1.

Contents Octets

The concatenation of the BER encodings of the components of the value:

- For simple string types and types derived from them by implicit tagging, the concatenation of the BER encodings of consecutive substrings of the value (underlying value for implicit tagging)
- For structured types and types derived from them by implicit tagging, the concatenation of the BER encodings of components of the value (underlying value for implicit tagging)
- For types derived from anything by explicit tagging, the BER encoding of the underlying value

Details for particular types are given in Section 5.

Section 3.3: Constructed, Indefinite-Length Method

This method applies to simple string types, structured types, types derived from simple string types and structured types by implicit tagging, and types derived from anything by explicit tagging. It does not require that the length of the value be known in advance. The parts of the BER encoding are as follows:

Identifier Octets

As described in Section 3.2.

Length Octets

One octet, 80.

Contents Octets

As described in Section 3.2.

End-of-Contents Octets

Two octets, `00 00`.

Since the end-of-contents octets appear where an ordinary BER encoding might be expected (e.g., in the contents octets of a sequence value), the `00` and `00` appear as identifier and length octets, respectively. Thus the end-of-contents octets are really the primitive, definite-length encoding of a value with universal class, tag number 0, and length 0.

Section 4: Distinguished Encoding Rules

The Distinguished Encoding Rules (DER) for ASN.1 are a subset of BER, and give exactly one way to represent any ASN.1 value as an octet string. DER is intended for applications in which a unique octet string encoding is needed, as is the case when a digital signature is computed on an ASN.1 value. DER is defined in Section 8.7 of X.509.

DER adds the following restrictions to the rules given in Section 3:

1. When the length is between 0 and 127, the short form of length must be used.
2. When the length is 128 or greater, the long form of length must be used, and the length must be encoded in the minimum number of octets.
3. For simple string types and implicitly tagged types derived from simple string types, the primitive, definite-length method must be employed.
4. For structured types, implicitly tagged types derived from structured types, and explicitly tagged types derived from anything, the constructed, definite-length method must be employed.

Other restrictions are defined for particular types (such as `BIT STRING`, `SEQUENCE`, `SET`, and `SET OF`), and can be found in Section 5.

Section 5: Notation and Encodings for Some Types

This section gives the notation for some ASN.1 types and describes how to encode values of those types under both BER and DER.

The types described are those presented in Section 2. They are listed alphabetically here.

Each description includes ASN.1 notation, BER encoding, and DER encoding. The focus of the encodings is primarily on the contents octets; the tag and length octets follow Sections 3 and 4. The descriptions also explain where each type is used in PKCS and related standards. ASN.1 notation is generally given only for types, although for the type `OBJECT IDENTIFIER`, value notation is given as well.

Section 5.1: Implicitly Tagged Types

An implicitly tagged type is a type derived from another type by changing the tag of the underlying type.

Implicit tagging is used for optional `SEQUENCE` components with underlying type other than `ANY` throughout PKCS, and for the `extendedCertificate` alternative of PKCS #7's `ExtendedCertificateOrCertificate` type.

```
ASN.1 notation:
[[class] number] IMPLICIT Type
class = UNIVERSAL  |  APPLICATION  |  PRIVATE
```

where ***Type*** is a type, ***class*** is an optional class name, and ***number*** is the tag number within the class, a nonnegative integer.

In ASN.1 "modules" whose default tagging method is implicit tagging, the notation [[***class***] ***number***] ***Type*** is also acceptable, and the keyword `IMPLICIT` is implied. (See Section 2.3.) For definitions stated outside a module, the explicit inclusion of the keyword `IMPLICIT` is preferable to prevent ambiguity.

If the class name is absent, then the tag is context-specific. Context-specific tags can only appear in a component of a structured or `CHOICE` type.

Example PKCS #8's `PrivateKeyInfo` type has an optional `attributes` component with an implicit, context-specific tag:

```
PrivateKeyInfo ::= SEQUENCE {
  version Version,
  privateKeyAlgorithm PrivateKeyAlgorithmIdentifier,
  privateKey PrivateKey,
  attributes [0] IMPLICIT Attributes OPTIONAL }
```

Here the underlying type is `Attributes`, the class is absent (i.e., context-specific), and the tag number within the class is 0.

BER Encoding

Primitive or constructed, depending on the underlying type. Contents octets are as for the BER encoding of the underlying value.

Example The BER encoding of the `attributes` component of a `PrivateKeyInfo` value is as follows:

- The identifier octets are `80` if the underlying `Attributes` value has a primitive BER encoding, and `a0` if the underlying `Attributes` value has a constructed BER encoding.
- The length and contents octets are the same as the length and contents octets of the BER encoding of the underlying `Attributes` value.

DER Encoding

Primitive or constructed, depending on the underlying type. Contents octets are as for the DER encoding of the underlying value.

Section 5.2: Explicitly Tagged Types

Explicit tagging denotes a type derived from another type by adding an outer tag to the underlying type.

Explicit tagging is used for optional `SEQUENCE` components with underlying type `ANY` throughout PKCS, and for the `version` component of X.509's `Certificate` type.

```
ASN.1 notation:
[[class] number] EXPLICIT Type
class = UNIVERSAL  |  APPLICATION  |  PRIVATE
```

where ***Type*** is a type, ***class*** is an optional class name, and ***number*** is the tag number within the class, a nonnegative integer.

If the class name is absent, then the tag is context-specific. Context-specific tags can only appear in a component of a SEQUENCE, SET, or CHOICE type.

In ASN.1 "modules" whose default tagging method is explicit tagging, the notation [[***class***] ***number***] ***Type*** is also acceptable, and the keyword EXPLICIT is implied. (See Section 2.3.) For definitions stated outside a module, the explicit inclusion of the keyword EXPLICIT is preferable to prevent ambiguity.

Example 1 PKCS #7's ContentInfo type has an optional content component with an explicit, context-specific tag:

```
ContentInfo ::= SEQUENCE {
  contentType ContentType,
  content
    [0] EXPLICIT ANY DEFINED BY contentType OPTIONAL }
```

Here the underlying type is ANY DEFINED BY contentType, the class is absent (i.e., context-specific), and the tag number within the class is 0.

Example 2 X.509's Certificate type has a version component with an explicit, context-specific tag, where the EXPLICIT keyword is omitted:

```
Certificate ::= ...
  version [0] Version DEFAULT v1988,
...
```

The tag is explicit because the default tagging method for the ASN.1 "module" in X.509 that defines the Certificate type is explicit tagging.

BER Encoding

Constructed. Contents octets are the BER encoding of the underlying value.

Example The BER encoding of the content component of a ContentInfo value is as follows:

- Identifier octets are a0.
- Length octets represent the length of the BER encoding of the underlying ANY DEFINED BY contentType value.

- Contents octets are the BER encoding of the underlying ANY DEFINED BY contentType value.

DER Encoding

Constructed. Contents octets are the DER encoding of the underlying value.

Section 5.3: ANY

The ANY type denotes an arbitrary value of an arbitrary type, where the arbitrary type is possibly defined in the registration of an object identifier or associated with an integer index.

The ANY type is used for content of a particular content type in PKCS #7's ContentInfo type, for parameters of a particular algorithm in X.509's AlgorithmIdentifier type, and for attribute values in X.501's Attribute and AttributeValueAssertion types. The Attribute type is used by PKCS #6, #7, #8, #9, and #10, and the AttributeValue-Assertion type is used in X.501 distinguished names.

ASN.1 Notation

ANY [DEFINED BY ***identifier***]where ***identifier*** is an optional identifier.

In the ANY form, the actual type is indeterminate.

The ANY DEFINED BY ***identifier*** form can appear only in a component of a SEQUENCE or SET type for which ***identifier*** identifies some other component and only if that other component has type INTEGER or OBJECT IDENTIFIER (or a type derived from either of those by tagging). In that form, the actual type is determined by the value of the other component, either in the registration of the object identifier value, or in a table of integer values.

Example X.509's AlgorithmIdentifier type has a component of type ANY:

```
AlgorithmIdentifier ::= SEQUENCE {
  algorithm OBJECT IDENTIFIER,
  parameters ANY DEFINED BY algorithm OPTIONAL }
```

Here the actual type of the `parameter` component depends on the value of the `algorithm` component. The actual type would be defined in the registration of object identifier values for the `algorithm` component.

BER Encoding

Same as the BER encoding of the actual value.

Example The BER encoding of the value of the `parameter` component is the BER encoding of the value of the actual type as defined in the registration of object identifier values for the `algorithm` component.

DER Encoding

Same as the DER encoding of the actual value.

Section 5.4: BIT STRING

The `BIT STRING` type denotes an arbitrary string of bits (1's and 0's). A `BIT STRING` value can have any length, including zero. This type is a string type.

The `BIT STRING` type is used for digital signatures on extended certificates in PKCS #6's `ExtendedCertificate` type, for digital signatures on certificates in X.509's `Certificate` type, and for public keys in certificates in X.509's `SubjectPublicKeyInfo` type.

ASN.1 Notation

```
BIT STRING
```

Example X.509's `SubjectPublicKeyInfo` type has a component of type `BIT STRING`:

```
SubjectPublicKeyInfo ::= SEQUENCE {
  algorithm AlgorithmIdentifier,
  publicKey BIT STRING }
```

BER Encoding

Primitive or constructed. In a primitive encoding, the first contents octet gives the number of bits by which the length of the bit string is less than the next multiple of 8 (this is called the "number of unused bits"). The sec-

ond and following contents octets give the value of the bit string, converted to an octet string. The conversion process is as follows:

1. The bit string is padded after the last bit with zero to seven bits of any value to make the length of the bit string a multiple of 8. If the length of the bit string is a multiple of 8 already, no padding is done.
2. The padded bit string is divided into octets. The first eight bits of the padded bit string become the first octet, bit 8 to bit 1, and so on through the last eight bits of the padded bit string.

In a constructed encoding, the contents octets give the concatenation of the BER encodings of consecutive substrings of the bit string, where each substring except the last has a length that is a multiple of eight bits.

Example The BER encoding of the `BIT STRING` value "01101110010111 0111" can be any of the following, among others, depending on the choice of padding bits, the form of the length octets, and whether the encoding is primitive or constructed:

```
03 04 06 6e 5d c0          DER encoding
03 04 06 6e 5d e0          Padded with "100000"
03 81 04 06 6e 5d c0       Long form of length octets
23 09                      Constructed encoding:
   03 03 00 6e 5d          "0110111001011101" + "11"
   03 02 06 c0
```

DER encoding

Primitive. The contents octets are as for a primitive BER encoding, except that the bit string is padded with zero-valued bits.

Example The DER encoding of the `BIT STRING` value "01101110010111 0111" is

```
03 04 06 6e 5d c0
```

Section 5.5: CHOICE

The `CHOICE` type denotes a union of one or more alternatives.

The `CHOICE` type is used to represent the union of an extended certificate and an X.509 certificate in PKCS #7's `ExtendedCertificate-OrCertificate` type.

ASN.1 notation

```
CHOICE {
  [identifier1] Type1,
  ...,
  [identifiern] Typen }
```

where ***identifier***$_1$, . . ., ***identifier***$_n$ are optional, distinct identifiers for the alternatives, and ***Type***$_1$, . . ., ***Type***$_n$ are the types of the alternatives. The identifiers are primarily for documentation; they do not affect values of the type or their encodings in any way.

The types must have distinct tags. This requirement is typically satisfied with explicit or implicit tagging on some of the alternatives.

Example PKCS #7's `ExtendedCertificateOrCertificate` type is a `CHOICE` type:

```
ExtendedCertificateOrCertificate ::= CHOICE {
  certificate Certificate, -- X.509
  extendedCertificate [0] IMPLICIT ExtendedCertificate
}
```

Here the identifiers for the alternatives are `certificate` and `extendedCertificate`, and the types of the alternatives are `Certificate` and `[0] IMPLICIT ExtendedCertificate`.

BER encoding

Same as the BER encoding of the chosen alternative. The fact that the alternatives have distinct tags makes it possible to distinguish between their BER encodings.

Example The identifier octets for the BER encoding are `30` if the chosen alternative is `certificate`, and `a0` if the chosen alternative is `extendedCertificate`.

DER encoding

Same as the DER encoding of the chosen alternative.

Section 5.6: IA5String

The `IA5String` type denotes an arbitrary string of IA5 characters. IA5 stands for International Alphabet 5, which is the same as ASCII. The

character set includes non-printing control characters. An `IA5String` value can have any length, including zero. This type is a string type.

The `IA5String` type is used in PKCS #9's electronic-mail address, unstructured-name, and unstructured-address attributes.

ASN.1 notation

```
IA5String
```

BER encoding

Primitive or constructed. In a primitive encoding, the contents octets give the characters in the IA5 string, encoded in ASCII. In a constructed encoding, the contents octets give the concatenation of the BER encodings of consecutive substrings of the IA5 string.

Example The BER encoding of the `IA5String` value "test1@rsa.com" can be any of the following, among others, depending on the form of length octets and whether the encoding is primitive or constructed:

```
16 0d 74 65 73                                  DER encoding
   74 31 40 72 73 61 2e 63 6f 6d

16 81 0d                                        Long form of length octets
   74 65 73 74 31 40 72 73 61 2e 63 6f 6d

36 13                                           Constructed encoding:
   16 05 74 65 73 74 31                         "test1" + "@" + "rsa.com"
   16 01 40
   16 07 72 73 61 2e 63 6f 6d
```

DER Encoding

Primitive. Contents octets are as for a primitive BER encoding.

Example The DER encoding of the `IA5String` value "test1@rsa.com" is

```
16 0d 74 65 73 74 31 40 72 73 61 2e 63 6f 6d
```

Section 5.7: INTEGER

The `INTEGER` type denotes an arbitrary integer. `INTEGER` values can be positive, negative, or zero, and can have any magnitude.

The `INTEGER` type is used for version numbers throughout PKCS, for

cryptographic values such as modulus, exponent, and primes in PKCS #1's `RSAPublicKey` and `RSAPrivateKey` types and PKCS #3's `DHParameter` type, for a message-digest iteration count in PKCS #5's `PBEParameter` type, and for version numbers and serial numbers in X.509's `Certificate` type.

ASN.1 Notation

```
INTEGER [{ identifier1(value1) ... identifiern(valuen) }]
```

where ***identifier***$_1$, . . ., ***identifier***$_n$ are optional distinct identifiers and ***value***$_1$, . . ., ***value***$_n$ are optional integer values. The identifiers, when present, are associated with values of the type.

Example X.509's `Version` type is an `INTEGER` type with identified values:

```
Version ::= INTEGER { v1988(0) }
```

The identifier `v1988` is associated with the value 0. X.509's `Certificate` type uses the identifier `v1988` to give a default value of 0 for the `version` component:

```
Certificate ::= ...
  version Version DEFAULT v1988,
...
```

BER Encoding

Primitive. Contents octets give the value of the integer, base 256, in two's complement form, most significant digit first, with the minimum number of octets. The value 0 is encoded as a single `00` octet.

Some example BER encodings (which also happen to be DER encodings) are given in Table B-3.

DER Encoding

Primitive. Contents octets are as for a primitive BER encoding.

Table B-3 Example BER Encodings of INTEGER Values

Integer Value	BER Encoding
0	02 01 00
127	02 01 7F
128	02 02 00 80
256	02 02 01 00
-128	02 01 80
-129	02 02 FF 7F

Section 5.8: NULL

The NULL type denotes a null value.

The NULL type is used for algorithm parameters in several places in PKCS.

ASN.1 Notation

```
NULL
```

BER Encoding

Primitive. Contents octets are empty.

Example The BER encoding of a NULL value can be either of the following, as well as others, depending on the form of the length octets:

```
05 00
05 81 00
```

DER Encoding

Primitive. Contents octets are empty; the DER encoding of a NULL value is always 05 00.

Section 5.9: OBJECT IDENTIFIER

The OBJECT IDENTIFIER type denotes an object identifier, a sequence of integer components that identifies an object such as an algorithm, an attribute type, or perhaps a registration authority that defines other object identifiers. An OBJECT IDENTIFIER value can have any number of components, and components can generally have any nonnegative value. This type is a nonstring type.

OBJECT IDENTIFIER values are given meanings by registration authorities. Each registration authority is responsible for all sequences of components beginning with a given sequence. A registration authority typically delegates responsibility for subsets of the sequences in its domain to other registration authorities, or for particular types of objects. There are always at least two components.

The OBJECT IDENTIFIER type is used to identify content in PKCS #7's ContentInfo type, to identify algorithms in X.509's AlgorithmIdentifier type, and to identify attributes in X.501's Attribute and AttributeValueAssertion types. The Attribute type is used by PKCS #6, #7, #8, #9, and #10, and the AttributeValueAssertion type is used in X.501 distinguished names. OBJECT IDENTIFIER values are defined throughout PKCS.

ASN.1 Notation

```
OBJECT IDENTIFIER
```

The ASN.1 notation for values of the OBJECT IDENTIFIER type is

```
{ [identifier] component1 ... componentn }
componenti = identifieri | identifieri (valuei) | valuei
```

where ***identifier***, ***identifier***$_1$, . . ., ***identifier***$_n$ are identifiers, and ***value***$_1$, . . ., ***value***$_n$ are optional integer values.

The form without ***identifier*** is the "complete" value with all its components; the form with ***identifier*** abbreviates the beginning components with another object identifier value. The identifiers ***identifier***$_1$, . . ., ***identifier***$_n$ are intended primarily for documentation, but they must correspond to the integer value when both are present. These identifiers can appear without integer values only if they are among a small set of identifiers defined in X.208.

Example Both of the following values refer to the object identifier assigned to RSA Data Security, Inc.:

```
{ iso(1) member-body(2) 840 113549 }
{ 1 2 840 113549 }
```

(In this example, which gives ASN.1 value notation, the object identifier values are decimal, not hexadecimal.) Table A-4 gives some other object identifier values and their meanings.

Table B-4 Some Object Identifier Values and Their Meanings

Object Identifier Value	Meaning
{ 1 2 }	ISO member bodies
{ 1 2 840 }	US (ANSI)
{ 1 2 840 113549 }	RSA Data Security, Inc.
{ 1 2 840 113549 1 }	RSA Data Security, Inc. PKCS
{ 2 5 }	Directory services (X.500)
{ 2 5 8 }	Directory services—algorithms

BER Encoding

Primitive. Contents octets are as follows, where $\boldsymbol{value}_1, \ldots, \boldsymbol{value}_n$ denote the integer values of the components in the complete object identifier:

1. The first octet has value $40 \times \boldsymbol{value}_1 + \boldsymbol{value}_2$. (This is unambiguous, since $\boldsymbol{value}_1$ is limited to values 0, 1, and 2; $\boldsymbol{value}_2$ is limited to the range 0 to 39 when $\boldsymbol{value}_1$ is 0 or 1; and, according to X.208, n is always at least 2.)
2. The following octets, if any, encode $\boldsymbol{value}_3, \ldots, \boldsymbol{value}_n$. Each value is encoded base 128, most significant digit first, with as few digits as possible, and the most significant bit of each octet except the last in the value's encoding set to "1."

Example The first octet of the BER encoding of RSA Data Security, Inc.'s object identifier is $40 \times 1 + 2 = 42 = 2a_{16}$. The encoding of $840 = 6 \times 128 + 48_{16}$ is `86 48`, and the encoding of $113549 = 6 \times 128^2 + 77_{16} \times 128 + d_{16}$ is `86 f7 0d`. This leads to the following BER encoding:

```
06 06 2a 86 48 86 f7 0d
```

DER Encoding

Primitive. Contents octets are as for a primitive BER encoding.

Section 5.10: OCTET STRING

The `OCTET STRING` type denotes an arbitrary string of octets (eight-bit values). An `OCTET STRING` value can have any length, including zero. This type is a string type.

The `OCTET STRING` type is used for salt values in PKCS #5's `PBE Parameter` type, for message digests, encrypted message digests, and encrypted content in PKCS #7, and for private keys and encrypted private keys in PKCS #8.

ASN.1 Notation

```
OCTET STRING [SIZE ({size | size1..size2})]
```

where ***size***, ***size***$_1$, and ***size***$_2$ are optional size constraints. In the `OCTET STRING SIZE` (***size***) form, the octet string must have ***size*** octets. In the `OCTET STRING SIZE` (***size***$_1$..***size***$_2$) form, the octet string must have between size1 and size2 octets. In the `OCTET STRING` form, the octet string can have any size.

Example PKCS #5's `PBEParameter` type has a component of type `OCTET STRING`:

```
PBEParameter ::= SEQUENCE {
  salt OCTET STRING SIZE(8),
  iterationCount INTEGER }
```

Here the size of the `salt` component is always eight octets.

BER Encoding

Primitive or constructed. In a primitive encoding, the contents octets give the value of the octet string, first octet to last octet. In a constructed encoding, the contents octets give the concatenation of the BER encodings of substrings of the `OCTET STRING` value.

Example The BER encoding of the `OCTET STRING` value `01 23 45 67 89 ab cd ef` can be any of the following, among others, depending on the form of length octets and whether the encoding is primitive or constructed:

```
04 08 01 23 45 67 89 ab cd ef        DER encoding
04 81 08 01 23 45 67 89 ab cd ef     Long form of length octets
24 0c                                Constructed encoding:
   04 04 01 23 45 67                 01 ... 67 + 89 ... ef
   04 04 89 ab cd ef
```

DER Encoding

Primitive. Contents octets are as for a primitive BER encoding.

Example The BER encoding of the `OCTET STRING` value `01 23 45 67 89 ab cd ef` is

```
04 08 01 23 45 67 89 ab cd ef
```

Section 5.11: PrintableString

The `PrintableString` type denotes an arbitrary string of printable characters from the following character set:

```
A, B, ..., Z
a, b, ..., z
0, 1, ..., 9
(space) ' ( ) + , - . / : = ?
```

This type is a string type.

The `PrintableString` type is used in PKCS #9's challenge-password and unstructured-address attributes, and in several X.521 distinguished names attributes.

ASN.1 Notation

```
PrintableString
```

BER Encoding

Primitive or constructed. In a primitive encoding, the contents octets give the characters in the printable string, encoded in ASCII. In a constructed encoding, the contents octets give the concatenation of the BER encodings of consecutive substrings of the string.

Example The BER encoding of the `PrintableString` value "Test User 1" can be any of the following, among others, depending on the form of length octets and whether the encoding is primitive or constructed:

```
13 0b 54 65 73 74 20 55 73 65 72 20 31        DER encoding

13 81 0b 54 65 73 74 20 55 73 65 72 20 31     Long form of length
                                              octets

33 0f                                         Constructed
   13 05 54 65 73 74 20                       encoding:
   13 06 55 73 65 72 20 31                    "Test " + "User 1"
```

DER Encoding

Primitive. Contents octets are as for a primitive BER encoding.

Example The DER encoding of the `PrintableString` value "Test User 1" is

```
13 0b 54 65 73 74 20 55 73 65 72 20 31
```

Section 5.12: SEQUENCE

The `SEQUENCE` type denotes an ordered collection of one or more types.

The `SEQUENCE` type is used throughout PKCS and related standards.

ASN.1 Notation

```
SEQUENCE {
  [identifier1] Type1 [{OPTIONAL | DEFAULT value1}],
  ...,
  [identifiern] Typen [{OPTIONAL | DEFAULT valuen}]}
```

where ***identifier***$_1$, . . ., ***identifier***$_n$ are optional, distinct identifiers for the components, ***Type***$_1$, . . ., ***Type***$_n$ are the types of the components, and ***value***$_1$, . . ., ***value***$_n$ are optional default values for the components. The identifiers are primarily for documentation; they do not affect values of the type or their encodings in any way.

The OPTIONAL qualifier indicates that the value of a component is optional and need not be present in the sequence. The DEFAULT qualifier also indicates that the value of a component is optional, and assigns a default value to the component when the component is absent.

The types of any consecutive series of components with the OPTIONAL or DEFAULT qualifier, as well as of any component immediately following that series, must have distinct tags. This requirement is typically satisfied with explicit or implicit tagging on some of the components.

Example X.509's Validity type is a SEQUENCE type with two components:

```
Validity ::= SEQUENCE {
  start UTCTime,
  end UTCTime }
```

Here the identifiers for the components are start and end, and the type of both components is UTCTime.

BER Encoding

Constructed. Contents octets are the concatenation of the BER encodings of the values of the components of the sequence, in order of definition, with the following rules for components with the OPTIONAL and DEFAULT qualifiers:

- If the value of a component with the OPTIONAL or DEFAULT qualifier is absent from the sequence, then the encoding of that component is not included in the contents octets.
- If the value of a component with the DEFAULT qualifier is the default value, then the encoding of that component may or may not be included in the contents octets.

DER Encoding

Constructed. Contents octets are the same as the BER encoding, except that if the value of a component with the DEFAULT qualifier is the default value, the encoding of that component is not included in the contents octets.

Sectopm 5.13: SEQUENCE OF

The SEQUENCE OF type denotes an ordered collection of zero or more occurrences of a given type.

The SEQUENCE OF type is used in X.501 distinguished names.

ASN.1 Notation

```
SEQUENCE OF Type
```

where ***Type*** is a type.

Example X.501's RDNSequence type consists of zero or more occurrences of the RelativeDistinguishedName type, most significant occurrence first:

```
RDNSequence ::= SEQUENCE OF RelativeDistinguishedName
```

BER Encoding

Constructed. Contents octets are the concatenation of the BER encodings of the values of the occurrences in the collection, in order of occurrence.

DER Encoding

Constructed. Contents octets are the concatenation of the DER encodings of the values of the occurrences in the collection, in order of occurrence.

Section 5.14: SET

The SET type denotes an unordered collection of one or more types.

The SET type is not used in PKCS.

ASN.1 Notation

```
SET {
  [identifier1] Type1 [{OPTIONAL | DEFAULT value1}],
  ...,
  [identifiern] Typen [{OPTIONAL | DEFAULT valuen}]}
```

where ***identifier***$_1$, . . ., ***identifier***$_n$ are optional, distinct identifiers for the components, ***Type***$_1$, . . ., ***Type***$_n$ are the types of the components, and ***value***$_1$, . . ., ***value***$_n$ are optional default values for the components. The identifiers are primarily for documentation; they do not affect values of the type or their encodings in any way.

The OPTIONAL qualifier indicates that the value of a component is optional and need not be present in the set. The DEFAULT qualifier also indicates that the value of a component is optional, and assigns a default value to the component when the component is absent.

The types must have distinct tags. This requirement is typically satisfied with explicit or implicit tagging on some of the components.

BER Encoding

Constructed. Contents octets are the concatenation of the BER encodings of the values of the components of the set, in any order, with the following rules for components with the OPTIONAL and DEFAULT qualifiers:

- If the value of a component with the OPTIONAL or DEFAULT qualifier is absent from the set, then the encoding of that component is not included in the contents octets.
- If the value of a component with the DEFAULT qualifier is the default value, then the encoding of that component may or may not be included in the contents octets.

DER Encoding

Constructed. Contents octets are the same as for the BER encoding, except that:

- If the value of a component with the DEFAULT qualifier is the default value, the encoding of that component is not included.
- There is an order to the components, namely ascending order by tag.

Section 5.15: SET OF

The SET OF type denotes an unordered collection of zero or more occurrences of a given type.

The SET OF type is used for sets of attributes in PKCS #6, #7, #8, #9, and #10, for sets of message-digest algorithm identifiers, signer information, and recipient information in PKCS #7, and in X.501 distinguished names.

ASN.1 Notation

```
SET OF Type
```

where ***Type*** is a type.

Example X.501's RelativeDistinguishedName type consists of zero or more occurrences of the AttributeValueAssertion type, where the order is unimportant:

```
RelativeDistinguishedName ::=
  SET OF AttributeValueAssertion
```

BER Encoding

Constructed. Contents octets are the concatenation of the BER encodings of the values of the occurrences in the collection, in any order.

DER Encoding

Constructed. Contents octets are the same as for the BER encoding, except that there is an order, namely ascending lexicographic order of BER encoding. Lexicographic comparison of two different BER encodings is done as follows: Logically pad the shorter BER encoding after the last octet with dummy octets that are smaller in value than any normal octet. Scan the BER encodings from left to right until a difference is found. The smaller-valued BER encoding is the one with the smaller-valued octet at the point of difference.

Section 5.16: T61String

The `T61String` type denotes an arbitrary string of T.61 characters. T.61 is an eight-bit extension to the ASCII character set. Special "escape" sequences specify the interpretation of subsequent character values as, for example, Japanese; the initial interpretation is Latin. The character set includes nonprinting control characters. The `T61String` type allows only the Latin and Japanese character interpretations, and implementors' agreements for directory names exclude control characters [NIST92]. A `T61String` value can have any length, including zero. This type is a string type.

The `T61String` type is used in PKCS #9's unstructured-address and challenge-password attributes, and in several X.521 attributes.

ASN.1 Notation

```
T61String
```

BER Encoding

Primitive or constructed. In a primitive encoding, the contents octets give the characters in the T.61 string, encoded in ASCII. In a constructed encoding, the contents octets give the concatenation of the BER encodings of consecutive substrings of the T.61 string.

Example The BER encoding of the `T61String` value "clés publiques" (French for "public keys") can be any of the following, among others, depending on the form of length octets and whether the encoding is primitive or constructed:

```
14 0f                                              DER encoding
   63 6c c2 65 73 20 70 75 62 6c 69 71 75 65 73

14 81 0f                                           Long form of length octets
   63 6c c2 65 73 20 70 75 62 6c 69 71 75 65 73

34 15                                              Constructed encoding:
   14 05 63 6c c2 65 73                            "clés" + " " + "publiques"
   14 01 20
   14 09 70 75 62 6c 69 71 75 65 73
```

The eight-bit character `c2` is a T.61 prefix that adds an acute accent (´) to the next character.

DER Encoding

Primitive. Contents octets are as for a primitive BER encoding.

Example The DER encoding of the `T61String` value "clés publiques" is

```
14 0f 63 6c c2 65 73 20 70 75 62 6c 69 71 75 65 73
```

Section 5.17: UTCTime

The `UTCTime` type denotes a "coordinated universal time" or Greenwich Mean Time (GMT) value. A `UTCTime` value includes the local time precise to either minutes or seconds, and an offset from GMT in hours and minutes. It takes any of the following forms:

```
YYMMDDhhmmZ
YYMMDDhhmm+hh'mm'
YYMMDDhhmm-hh'mm'
YYMMDDhhmmssZ
YYMMDDhhmmss+hh'mm'
YYMMDDhhmmss-hh'mm'
```

where:

YY is the least significant two digits of the year

MM is the month (01 to 12)

DD is the day (01 to 31)

hh is the hour (00 to 23)

mm are the minutes (00 to 59)

ss are the seconds (00 to 59)

`Z` indicates that local time is GMT, + indicates that local time is later than GMT, and - indicates that local time is earlier than GMT

hh' is the absolute value of the offset from GMT in hours

mm' is the absolute value of the offset from GMT in minutes

This type is a string type.

The `UTCTime` type is used for signing times in PKCS #9's signing-time attribute and for certificate validity periods in X.509's `Validity` type.

ASN.1 Notation

```
UTCTime
```

BER Encoding

Primitive or constructed. In a primitive encoding, the contents octets give the characters in the string, encoded in ASCII. In a constructed encoding, the contents octets give the concatenation of the BER encodings of consecutive substrings of the string. (The constructed encoding is not particularly interesting, since `UTCTime` values are so short, but the constructed encoding is permitted.)

Example The time this sentence was originally written was 4:45:40 P.M. Pacific Daylight Time on May 6, 1991, which can be represented with either of the following `UTCTime` values, among others:

```
"910506164540-0700"
"910506234540Z"
```

These values have the following BER encodings, among others:

```
17 0d 39 31 30 35 30 36 32 33 34 35 34 30 5a

17 11 39 31 30 35 30 36 31 36 34 35 34 30 2D 30 37 30
      30
```

DER Encoding

Primitive. Contents octets are as for a primitive BER encoding.

Section 6: An Example

This section gives an example of ASN.1 notation and DER encoding: the X.501 type `Name`.

Section 6.1: Abstract Notation

This section gives the ASN.1 notation for the X.501 type Name.

```
Name ::= CHOICE {
  RDNSequence }

RDNSequence ::= SEQUENCE OF RelativeDistinguishedName

RelativeDistinguishedName ::=
  SET OF AttributeValueAssertion

AttributeValueAssertion ::= SEQUENCE {
   AttributeType,
   AttributeValue }

AttributeType ::= OBJECT IDENTIFIER

AttributeValue ::= ANY
```

The Name type identifies an object in an X.500 directory. Name is a CHOICE type consisting of one alternative: RDNSequence. (Future revisions of X.500 may have other alternatives.)

The RDNSequence type gives a path through an X.500 directory tree starting at the root. RDNSequence is a SEQUENCE OF type consisting of zero or more occurrences of RelativeDistinguishedName.

The RelativeDistinguishedName type gives a unique name to an object relative to the object superior to it in the directory tree. RelativeDistinguishedName is a SET OF type consisting of zero or more occurrences of AttributeValueAssertion.

The AttributeValueAssertion type assigns a value to some attribute of a relative distinguished name, such as country name or common name. AttributeValueAssertion is a SEQUENCE type consisting of two components, an AttributeType type and an AttributeValue type.

The AttributeType type identifies an attribute by object identifier. The AttributeValue type gives an arbitrary attribute value. The actual type of the attribute value is determined by the attribute type.

Section 6.2: DER Encoding

This section gives an example of a DER encoding of a value of type Name, working from the bottom up.

The name is that of the Test User 1 from the PKCS examples [Kal93]. The name is represented by the following path:

```
                    (root)
                      |
          countryName = "US"
                      |
organizationName = "Example Organization"
                      |
        commonName = "Test User 1"
```

Each level corresponds to one `RelativeDistinguishedName` value, each of which happens for this name to consist of one `AttributeValueAssertion` value. The `AttributeType` value is before the equals sign, and the `AttributeValue` value (a printable string for the given attribute types) is after the equals sign.

The `countryName`, `organizationName`, and `commonUnitName` are attribute types defined in X.520 as:

```
attributeType OBJECT IDENTIFIER ::=
  { joint-iso-ccitt(2) ds(5) 4 }

countryName OBJECT IDENTIFIER ::= { attributeType 6 }
organizationName OBJECT IDENTIFIER ::=
  { attributeType 10 }
commonUnitName OBJECT IDENTIFIER ::=
  { attributeType 3 }
```

AttributeType

The three `AttributeType` values are OCTET STRING values, so their DER encoding follows the primitive, definite-length method:

```
06 03 55 04 06    countryName
06 03 55 04 0a    organizationName
06 03 55 04 03    commonName
```

The identifier octets follow the low-tag form, since the tag is 6 for `OBJECT IDENTIFIER`. Bits 8 and 7 have value "0," indicating universal class, and bit 6 has value "0," indicating that the encoding is primitive. The length octets follow the short form. The contents octets are the concatenation of three octet strings derived from subidentifiers (in decimal): $40 \times 2 + 5 = 85 = 55_{16}$; 4; and 6, 10, or 3.

AttributeValue

The three `AttributeValue` values are `PrintableString` values, so their encodings follow the primitive, definite-length method:

```
13 02 55 53                                              "US"

13 14                                                    "Example
   45 78 61 6d 70 6c 65 20 4f 72 67 61 6e 69 7a 61       Organization"
   74 69 6f 6e

13 0b                                                    "Test User 1"
   54 65 73 74 20 55 73 65 72 20 31
```

The identifier octets follow the low-tag-number form, since the tag for `PrintableString`, 19 (decimal), is between 0 and 30. Bits 8 and 7 have value "0" since `PrintableString` is in the universal class. Bit 6 has value "0" since the encoding is primitive. The length octets follow the short form, and the contents octets are the ASCII representation of the attribute value.

AttributeValueAssertion

The three `AttributeValueAssertion` values are `SEQUENCE` values, so their DER encodings follow the constructed, definite-length method:

```
30 09                          countryName = "US"
 06 03 55 04 06
 13 02 55 53

30 1b                          organizationName = "Example Organization"
 06 03 55 04 0a
 1 3 14  . . .  6f 6e

30 12                          commonName = "Test User 1"
 06 03 55 04 0b
 13 0b ... 20 31
```

The identifier octets follow the low-tag-number form, since the tag for `SEQUENCE`, 16 (decimal), is between 0 and 30. Bits 8 and 7 have value "0" since `SEQUENCE` is in the universal class. Bit 6 has value "1" since the encoding is constructed. The length octets follow the short form, and the contents octets are the concatenation of the DER encodings of the `attributeType` and `attributeValue` components.

RelativeDistinguishedName

The three `RelativeDistinguishedName` values are `SET OF` values, so their DER encodings follow the constructed, definite-length method:

```
31 0b
   30 09 ... 55 53

31 1d
   30 1b ... 6f 6e

31 14
   30 12  ...  20 31
```

The identifier octets follow the low-tag-number form, since the tag for `SET OF`, 17 (decimal), is between 0 and 30. Bits 8 and 7 have value "0" since `SET OF` is in the universal class. Bit 6 has value "1" since the encoding is constructed. The length octets follow the short form, and the contents octets are the DER encodings of the respective `AttributeValueAssertion` values, since there is only one value in each set.

RDNSequence

The `RDNSequence` value is a `SEQUENCE OF` value, so its DER encoding follows the constructed, definite-length method:

```
30 42
   31 0b ... 55 53
   31 1d ... 6f 6e
   31 14  ...  20 31
```

The identifier octets follow the low-tag-number form, since the tag for `SEQUENCE OF`, 16 (decimal), is between 0 and 30. Bits 8 and 7 have value "0" since `SEQUENCE OF` is in the universal class. Bit 6 has value "1" since the encoding is constructed. The length octets follow the short form, and the contents octets are the concatenation of the DER encodings of the three `RelativeDistinguishedName` values, in order of occurrence.

Name

The Name value is a CHOICE value, so its DER encoding is the same as that of the RDNSequence value:

```
30 42
   31 0b
      30 09
         06 03 55 04 06   attributeType = countryName
         13 02 55 53      attributeValue = "US"
   31 1d
      30 1b
         06 03 55 04 0a   attributeType = organizationName
         13 14             attributeValue = "Example Organization"
            45 78 61 6d 70 6c 65 20 4f 72 67 61 6e 69 7a 61
            74 69 6f 6e
   31 14
      30 12
         06 03 55 04 03   attributeType = commonName
         13 0b             attributeValue = "Test User 1"
            54 65 73 74 20 55 73 65 72 20 31
```

References

PKCS #1 RSA Laboratories. *PKCS #1: RSA Encryption Standard.* Version 1.5, November 1993.

PKCS #3 RSA Laboratories. *PKCS #3: Diffie-Hellman Key-Agreement Standard.* Version 1.4, November 1993.

PKCS #5 RSA Laboratories. *PKCS #5: Password-Based Encryption Standard.* Version 1.5, November 1993.

PKCS #6 RSA Laboratories. *PKCS #6: Extended-Certificate Syntax Standard.* Version 1.5, November 1993.

PKCS #7 RSA Laboratories. *PKCS #7: Cryptographic Message Syntax Standard.* Version 1.5, November 1993.

PKCS #8 RSA Laboratories. *PKCS #8: Private-Key Information Syntax Standard.* Version 1.2, November 1993.

PKCS #9 RSA Laboratories. *PKCS #9: Selected Attribute Types.* Version 1.1, November 1993.

PKCS #10 RSA Laboratories. *PKCS #10: Certification Request Syntax Standard.* Version 1.0, November 1993.

X.200 CCITT. *Recommendation X.200: Reference Model of Open Systems Interconnection for CCITT Applications.* 1984.

X.208 CCITT. *Recommendation X.208: Specification of Abstract Syntax Notation One (ASN.1).* 1988.

X.209 CCITT. *Recommendation X.209: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).* 1988.

X.500 CCITT. *Recommendation X.500: The Directory—Overview of Concepts, Models and Services.* 1988.

X.501 CCITT. *Recommendation X.501: The Directory—Models.* 1988.

X.509 CCITT. *Recommendation X.509: The Directory—Authentication Framework.* 1988.

X.520 CCITT. Recommendation X.520: The Directory—Selected Attribute Types. 1988.

[Kal93] Burton S. Kaliski Jr. *Some Examples of the PKCS Standards.* RSA Laboratories, November 1993.

[NIST92] NIST. Special *Publication 500-202: Stable Implementation Agreements for Open Systems Interconnection Protocols.* Part 11 (Directory Services Protocols). December 1992.

Revision History

June 3, 1991, Version

The June 3, 1991, version is part of the initial public release of PKCS. It was published as NIST/OSI Implementors' Workshop document SEC-SIG-91-17.

November 1, 1993, Version

The November 1, 1993, version incorporates several editorial changes, including the addition of a revision history. It is updated to be consistent with the following versions of the PKCS documents:

PKCS #1 *RSA Encryption Standard.* Version 1.5, November 1993.

PKCS #3 *Diffie-Hellman Key-Agreement Standard.* Version 1.4, November 1993.

PKCS #5	*Password-Based Encryption Standard.* Version 1.5, November 1993.
PKCS #6	*Extended-Certificate Syntax Standard.* Version 1.5, November 1993.
PKCS #7	*Cryptographic Message Syntax Standard.* Version 1.5, November 1993.
PKCS #8	*Private-Key Information Syntax Standard.* Version 1.2, November 1993.
PKCS #9	*Selected Attribute Types.* Version 1.1, November 1993.
PKCS #10	*Certification Request Syntax Standard.* Version 1.0, November 1993.

The following substantive changes were made:

Section 5	Description of `T61String` type is added.
Section 6	Names are changed, consistent with other PKCS examples.

APPENDIX C

Further Technical Details

In this appendix, you will find extra information not covered in the main body of the book. It is a deeper look at some of the topics described. This information is not necessary for a proper understanding of the main concepts, but should be interesting reading for those who want to explore cryptography a little further.

How Digest-Based PRNGs Work

As mentioned in Chapter 2, most PRNGs (*pseudo-random number generators*) are based on digest algorithms. The algorithm takes a seed and—just as sowing a botanical seed produces a plant—produces a virtually unlimited number of pseudo-random numbers. Here is a typical implementation using SHA-1 as the underlying digest algorithm.

Suppose the user wants two 128-bit session keys. The first step is give the seed to the PRNG, which digests it using SHA-1. The seed is the "message" of the message digest. This produces a 20-byte internal value, commonly called the *state*, which must be kept secret. Next, the user asks the PRNG for 16 bytes (the data of the first 128-bit session key). The PRNG uses SHA-1 to digest the state. Now the state, rather than the seed, is the

message of the message digest. The digest produces 20 bytes. The user needs only 16, so the PRNG outputs the first 16 bytes.

The user then requests 16 more bytes (the data of the second 128-bit session key). The PRNG has four left over from the last call; it could return them, but it also needs 12 more to fill the second request. To get the next 12 bytes, the PRNG changes the state somehow and digests the resulting new state. Because the PRNG has changed the state, this next block of 20 bytes will be different from the first block. The PRNG now has 20 new bytes. It returns the four left over from the first digest and the first 12 from the current digest.

Each time the PRNG produces output, it either returns leftovers or changes the state, digests the state, and returns as many bytes from that result as needed.

How does a PRNG change the state? It may simply add one to the current state. Recall that if you change a message, even if only by one bit, the resulting output will be significantly different. No matter what the input message is, the output will always pass tests of randomness. So if the PRNG takes a current state and adds one to it, digesting the new state will produce completely different, pseudo-random output. If the current state is

```
0xFF FF FF FF . . . FF
```

then adding one to it will change the state to

```
0x00 00 00 00 . . . 00
```

It's certainly possible to change the state by adding a different constant. Instead of adding 1, the PRNG could add a 20-byte number. In that way, all bytes of the state are manipulated in each operation.

A simpler PRNG would not bother with an internal state. Instead, it would digest the seed to create the first block of output and then would digest the first block of output to create the second block. Such a PRNG would be horrible.

Here's why. Suppose Ray (the attacker from Chapter 3) wants to read Pao-Chi's e-mail. The first thing Ray does is to get Pao-Chi to send him some encrypted e-mail—that is, to send him a few digital envelopes. With this e-mail, what Ray has is several 128-bit session keys (and possibly some initialization vectors if the encryption algorithm is a block cipher with a feedback mode). These keys are a series of pseudo-random bytes, each block produced by digesting the preceding block. With a little work, Ray can figure out a block boundary. Now Ray eavesdrops on Pao-Chi's future e-mails. What is the 128-bit session key used for the next e-mail?

It's simply the digest of the last block that Ray has. That's why good digest-based PRNGs use an internal state.

If the underlying digest algorithm is truly one-way (meaning that no one can determine the message from the digest), no one will be able to figure out what the state was that produced any particular pseudo-random output. If no one can figure out what the state is at any point in time, no one can compute what the next bytes will be.

Feedback Modes

In Chapter 2 you learned about block ciphers. A block cipher encrypts each block independently, so if the same block of plaintext appears more than once in a message, the resulting ciphertext block also will be repeated. This repetition could help an attacker. For example, suppose a company encrypts employee information in a database using the same key for each entry. If two entries contain the same block of ciphertext for "salary," anyone seeing that matching block would know that those two people earn the same salary.

Feedback modes make certain that each block of ciphertext is unique. (Except for that, they offer no additional security.) The most common feedback mode (described in Chapter 2) is *cipher block chaining* (CBC). When you encrypt data in CBC mode, each block of plaintext is XOR'd with the preceding block of ciphertext before the block is encrypted. There is no previous ciphertext for the first block, so it is XOR'd with an *initialization vector* (IV).

The term for no feedback is *electronic codebook* (ECB). Following are some other feedback modes.

Cipher Feedback Mode*

In *cipher feedback* (CFB) mode, you encrypt a block of data and XOR the plaintext with this encrypted block to produce the ciphertext. The block of data you encrypt is the preceding ciphertext. The first block has no preceding ciphertext, so it uses an IV. To create the first block of ciphertext, you encrypt the IV and XOR it with the plaintext. Now you save the

*(Source: RSA Labs)

resulting ciphertext for the next block. For the second block, you encrypt the preceding ciphertext (the result of the preceding XOR) and XOR the result of that with the plaintext. For example, suppose that the first plaintext block begins with the word "Goal." Here's the process.

1. Encrypt the IV. IV = 0xA722B551 . . . becomes 0x38F01321 . . . .
2. XOR the plaintext with the encrypted IV. Goal = 0x476F616C becomes 0x7F9F724D, which is the ciphertext.
3. Encrypt the preceding ciphertext. 0x7F9F724D . . . becomes 0xE1250B77 . . . .
4. XOR the plaintext with the encrypted preceding ciphertext, and repeat until the entire message is encrypted.

It's possible to define CFB mode so that it uses feedback that is less than one full data block. In fact, with CFB, it's possible to define a block size as one byte, effectively converting a block cipher into a stream cipher. Suppose you're using AES, a block cipher with a block size of 16 bytes. Here's what to do. First, use a 16-byte IV and encrypt it. You now have a block of 16 bytes that is the encrypted IV. Grab one byte of plaintext and XOR it with the most significant byte of the encrypted IV. Now that you've used that byte, throw it away by shifting the block of encrypted IV to the left. That leaves the least significant byte open. Fill it with the ciphertext (the result of the XOR). Now go on to the next byte of plaintext.

CFB mode is as secure as the underlying cipher, and using the XOR operation conceals plaintext patterns in the ciphertext. Plaintext cannot be manipulated directly except by the removal of blocks from the beginning or the end of the ciphertext.

Output Feedback Mode*

Output feedback (OFB) mode is similar to CFB mode except that the quantity XOR'd with each plaintext block is generated independently of both the plaintext and the ciphertext. Here's how to use this mode. For the first block of ciphertext, encrypt the IV and call this quantity the cipher block. Now XOR the cipher block with the plaintext. For the second block, encrypt the cipher block to create a new cipher block. Now XOR this new cipher block with the next block of plaintext.

*(Source: RSA Labs)

Assuming again that the first block of plaintext is "Goal," here's how OFB works.

1. Encrypt the IV to produce the cipher block. IV = 0xA722B551 . . . becomes 0x38F01321 . . . .
2. XOR the plaintext with the encrypted IV. Goal = 0x476F616C becomes 0x7F9F724D, which is the ciphertext.
3. Encrypt the preceding cipher block. 0x38F01321 . . . becomes 0x9D44BA16 . . . .
4. XOR the plaintext with the encrypted preceding cipher block, and continue in the same manner.

Feedback widths less than a full block are possible, but for security reasons they're not recommended. OFB mode has an advantage over CFB mode in that any bit errors that might occur during transmission are not propagated to affect the decryption of subsequent blocks. Furthermore, this mode can be programmed to take advantage of precomputations to speed the process. In CBC and CFB, you can't do the next step until completing the preceding step. Here, you can compute cipher blocks before computing the XOR or loading the next block of plaintext.

A problem with OFB mode is that the plaintext is easily manipulated. An attacker who knows a plaintext block m_i can replace it with a false plaintext block x by computing m_i XOR x to the corresponding ciphertext block c_i. Similar attacks can be used against CBC and CFB modes, but in those attacks some plaintext blocks will be modified in a manner that the attacker can't predict. Yet the very first ciphertext block (the initialization vector) in CBC mode and the very last ciphertext block in CFB mode are just as vulnerable to the attack as the blocks in OFB mode. Attacks of this kind can be prevented using, for example, a digital signature scheme or a MAC scheme.

Counter Mode*

Because of shortcomings in OFB mode, Whitfield Diffie has proposed an additional mode of operation termed *counter* mode. It differs from OFB mode in the way the successive data blocks are generated for subsequent

*(Source: RSA Labs)

encryptions. Instead of deriving one data block as the encryption of the preceding data block, Diffie proposes encrypting the quantity i + IV mod block bits for the ith data block.

Here's how it works. First, encrypt the IV and XOR it with the first block of plaintext. Now add 1 to the IV (if the IV were 0xFF FF ... FF, the addition would produce 0x00 00 ... 00; that's what the mod block bits means). Encrypt this new block, and XOR it with the next block of plaintext. If the cipher uses 8-byte blocks and the first block of data is "Goal #14," then counter mode would look like this:

1. Encrypt the IV to produce the cipher block. IV = 0xA722B551 041A3C96 becomes 0x38F01321 7922E09B.
2. XOR the plaintext with the encrypted IV. Goal #14 = 0x476F616C 20233134 becomes 0x7F9F724D 5901D1AF, which is the ciphertext.
3. Encrypt IV + 1. 0xA722B551 041A3C97 becomes 0x674B9B01 CFA38027.
4. XOR the plaintext with the encrypted IV + 1, and repeat the process.

Cryptanalysis of this method continues, but most cryptographers express confidence that counter mode will be a good alternative to CBC because this feedback mode can take advantage of precomputations and hence speed the process.

How to Plug Information Leaks from IV and Salts

One concern of cryptographers is the concept of leaking information. For example, the IV for a block cipher and the salt in PBE are not secret. Anyone eavesdropping on a digital conversation protected by a block cipher with a feedback mode will know what the IV is. Someone who finds a password-protected session key will know the salt. That's no problem because knowing the IV or salt does not help an attacker. But where did that IV and salt come from? A PRNG? The same PRNG that produced the session key? If it did, it means that the program has leaked information about the session key. The session key is related to the IV or salt (or both), and the attacker knows what those values are.

Certainly if the PRNG uses a good digest with an internal state, knowledge of the IV or salt will not lead the attacker to the session key. Even

though a relationship exists between the values, an attacker will not be able to exploit that relationship.

Nonetheless, information is leaking, and it's extremely simple to plug the hole. When you generate a session key, you use a PRNG with a good seed. For the IV or salt, you use a different PRNG seeded with the time of day.

You want every IV and salt you use to be unique. To do that, you use a different PRNG with a different salt each time. The time will be different for each instance, so there is a simple seed that will guarantee a different salt and IV each time. In Chapter 2, you saw that the time alone is a poor seed. That's because you don't want the attacker to do a brute force attack on the seed to reconstruct the PRNG and then reproduce the secrets generated by the PRNG. But if the thing generated by the PRNG is public, it doesn't matter whether the attacker can reproduce the seed. Knowledge of the seed in this case allows the attacker only to reconstruct a PRNG and reproduce values that have already been made public.

So to avoid leaking information, you use two PRNGs: one for generating secrets, and another one for generating salts and IVs. For performance reasons, you shouldn't waste time collecting a good seed for the PRNG that generates the salt and IV; simply use the time. It's fast, and it guarantees different output each time you use the PRNG.

Tamper-resistant Hardware*

In Chapter 3, you learned that some hardware devices are built to be tamper-resistant. Usually this means that they detect when someone is trying to access data through some means other than normal channels. How can these devices be made tamper-resistant?

Many techniques are used to make hardware tamper-resistant. Some of these techniques are intended to thwart direct attempts at opening a device and reading information from its memory; others offer protection against subtler attacks, such as timing attacks and induced hardware-fault attacks.

*(Source: RSA Labs)

At a very high level, here are a few general techniques used to make devices tamper-resistant:

- Employing sensors of various types (for example, light, temperature, and resistivity sensors) to detect malicious probing
- Packing device circuitry as densely as possible (dense circuitry makes it difficult for attackers to use a logic probe effectively)
- Using error-correcting memory
- Using nonvolatile memory so that the device can tell whether it has been reset (or how many times it has been reset)
- Using redundant processors to perform calculations and ensuring that all the calculated answers agree before outputting a result

RSA Padding

Chapter 4 describes the RSA digital envelope, in which a session key is encrypted with an RSA public key. The RSA public exponent is often 3, which means that to encrypt data, you treat the session key as a number and raise it to the 3rd power modulo the modulus. That's the same as finding $s \times s \times s \bmod n$, where s is the session key as a number, and n is the modulus. Most RSA keys are 1,024 bits, so the modulus is 1,024 bits long.

Modular multiplication means that the answer will always be less than the modulus. You compute the product, and if it is less than the modulus, there's no more work; if it's greater than the modulus, you reduce the result. To reduce a number means that you divide the intermediate result by the modulus and take the remainder as the answer. For example,

$$10 \times 10 \bmod 35 = 100 \bmod 35 = \text{rem}\,(100 \times 35) = 30$$

Because 10×35 is 2 with a remainder of 30 ($100 - 2 \times 35 = 100 - 70$), the answer is 30. The number 100 has been reduced to 30 modulo 35.

This reduction is essentially the reason RSA (and Diffie-Hellman and DSA) is secure. You compute an intermediate value and then reduce it. The attacker may know what the reduction is, but what was the intermediate value? If the attacker knows the intermediate value, it would not be hard to find the original number, but for each final number there are simply far too many possible intermediate values. That is, with big enough

keys, any one of trillions upon trillions of intermediate values produces the same final answer, and each intermediate value comes from a different starting point. A 1,024-bit modulus means that there are about 2^{1024} possible starting points, so there are about 2^{1024} possible intermediate values an attacker would have to examine to find the correct one. You saw in Chapter 2 how long it would take to examine 2^{128} keys, so you can imagine that using brute force to find the correct intermediate value among 2^{1024} possible numbers is not a viable option. That's why attackers would use other mathematical techniques, such as factoring or solving the discrete log problem, to break the algorithm, but those other techniques also take millions of years.

When you multiply two numbers, the size of the result is the sum of the sizes of the operands (or possibly one bit less). Assuming the session key you're encrypting is 128 bits long (the most common session key size), the size of $s \times s$ is 256. Because the RSA modulus is 1,024 bits long, the product $s \times s$ is smaller than the modulus, so no reduction is necessary. Finally, $s \times (s \times s)$ will be a 384-bit number (a 128-bit number times a 256-bit number). That, too, is less than the modulus, so no reduction is necessary.

So if you found $s \times s \times s \bmod n$, you would not do any reductions, and an attacker would know the intermediate value (it's the final result) and would be able to compute the original number s. The solution to this problem is to pad the data (see also Chapter 2). Here's how. Start with s, but make it a bigger number so that when finding $s \times s \times s$, you create intermediate answers larger than n and you have to reduce. Of course, you must pad in such a way that when decrypting, the recipient knows what the pad bytes are and can throw them away.

PKCS #1 Block 02 Padding

The most common padding scheme for RSA is defined in the Public Key Cryptography Standard #1 (PKCS #1; see the accompanying CD). It works like this. Start with a block that is the same size as the modulus. If the modulus is 1,024 bits, the block is 128 bytes long. The session key will fill 16 of those bytes (assuming that the session key is 128 bits), and this means that you'll need 112 bytes of padding. The padding comes first, followed by the key. So the first 112 bytes of the block are padding, and the last 16 bytes are the session key.

The first byte of padding is 00, the next byte is 02, and then the next 109 bytes are random or pseudo-random values, none of which can be 00. Finally, the 112th byte of padding is 00. If the session key were

```
FF FF FF FF   FF FF FF FF   FF FF FF FF   FF FF FF FF
```

then a padded block might look like this:

```
00 02 D0 CE 21 83 41 73 F6 84 32 06 A8 A6 AD 13
2B 65 27 86 28 EF 0E 8C CA 4F 20 C0 19 95 fE 6C
3E 69 1A 49 9C B7 CE 80 8A 9D C7 3D EC 6F 64 3A
A5 65 A0 A4 35 9A CA D4 CB CD 1D C8 60 6B E2 7F
2B BD 27 E1 47 F2 18 F0 65 41 9E 0D 1A CD B4 3D
24 14 C4 78 A6 A6 F3 1E 07 61 B6 C4 49 A0 77 18
BB 0E C7 72 E3 F1 79 1C 02 90 23 04 82 69 63 00
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

This block, then, becomes the value to encrypt. Cubing this number (cubing means to multiply a number by itself three times) will produce a number larger than the modulus, so reduction is necessary. The answer (ciphertext) might look something like this.

```
A0 2E 7D bE 8F 7A 3B DD 04 01 26 03 CC AF F5 7F
34 3F 49 22 C4 DC 48 09 E8 33 3B B0 59 DA D2 E7
B3 38 23 A7 D6 EB F1 B7 ED 3C 7B 45 81 4E 4F 3C
F4 BC 93 42 A8 8E 02 A9 05 1A fB 81 3E 8F 06 05
22 F3 90 9F 9B 35 13 A6 89 EC C3 5F 3F 6F 1D 9F
54 DE CB C0 0F F3 2F FF 1B 45 CA 80 B6 69 63 DF
54 C1 A7 B4 A2 D6 F5 53 E5 5D F1 D5 B9 F4 9E 5F
74 4C CD 72 C1 29 B7 FF D5 29 05 13 AD 04 BA 15
```

The recipient would decrypt this and recover the original padded value. That individual (more precisely, the recipient's software) would then need to *unpad,* or throw away the pad bytes, keeping only the session key. The first byte is 00, and the second byte is 02; those are simple—throw them away. Now throw away all the ensuing bytes that are not 00: the 109 random or pseudo-random numbers. The padding routine must never use 00 as one of those bytes, so during unpadding, the end of the padding is easy to spot; it's the first instance of 00. When the unpad routine reaches 00, it knows there's no more padding. It throws away the 00, and the rest of the bytes are the session key.

The Bleichenbacher Attack

In 1998, Daniel Bleichenbacher, a cryptographer at Bell Labs, came up with an attack against PKCS #1 Block 02 padding. This attack takes

advantage of the fact that the decryptor looks for specific bytes in specific locations. After decryption, the recipient will see whether the first byte is 00 and the second byte is 02 and whether there is a 00 after some random values.

Suppose that Ray, our attacker from Chapter 3, has an encrypted message from Pao-Chi to Gwen. If Ray can decrypt the RSA digital envelope portion of the communication, he will have the session key and can decrypt the message. Here's how the attack works. First, Ray computes a bogus RSA digital envelope that looks like Pao-Chi's envelope. To do that, Ray uses a special mathematical formula and uses as input Pao-Chi's correct envelope and a random or pseudo-random number (for details, see the RSA Labs Bulletin number 7, June 26, 1998, written by Daniel Bleichenbacher, Burt Kaliski, and Jessica Staddon). Ray then sends the substitute envelope to Gwen. If Gwen responds by saying that something went wrong, that the envelope didn't unwrap properly, Ray uses the same formula to create a new, different envelope using Pao-Chi's envelope and a different number (probably just the previous number plus 1) and sends the new envelope to Gwen. When an envelope unwraps improperly, it means that the first byte is not 00, or the second byte is not 02, or maybe there's no 00 to indicate the end of padding. Ray continues to send fake envelopes to Gwen until she responds by saying the envelope unwrapped properly.

When Ray has a fake envelope that works, he can figure out what Pao-Chi's original envelope is. The fake envelope and Pao-Chi's are related; Ray created the fake one based on the correct one and a number he chose. He uses this relationship to break the encryption. This technique does not break the private key; rather, it recovers only one envelope. Ray's fake envelope, when decrypted, does not produce the same thing Pao-Chi encrypted; rather, the result is something that simply looks like a digital envelope. It has the leading 00 02, and somewhere along the line there's another 00 to indicate the end of the padding. Gwen (or rather the software she uses to open the envelope) simply assumes that the numbers following this second 00, whatever those numbers happen to be, make up a session key. They don't—this is a bogus envelope—but to Gwen it looks like a legitimate envelope because all the marker bytes are there in the correct location.

Bleichenbacher's research indicates that Ray will probably need to send about 1,000,000 (one million) fake envelopes to recover one message. In some situations, he might even need to send 20,000,000 fake envelopes.

This attack is not likely to work using e-mail because Ray would have to wait for Gwen to open the one million e-mails and send a response to

each one, and eventually Gwen would stop trying to open any e-mail from Ray. But it might work if the recipient is using an automated responder. An example is an SSL server that simply responds to "hits," sending an error message when something goes wrong and opening a session when all goes right.

There are simple ways to thwart this attack (see the bulletin previously cited), and in fact, the SSL specification has a built-in countermeasure. It is probably safe, in the real world, to continue using PKCS #1 Block 02 padding when you're creating digital envelopes. However, if you want to avoid this attack, you can use a different padding scheme. The next section describes a padding scheme that's immune to the Bleichenbacher attack.

Optimal Asymmetric Encryption Padding

In 1995, two cryptographers-Mihir Bellare of the University of California at San Diego and Phillip Rogaway of the University of California at Davis-proposed a new way to pad RSA digital envelopes. They named this technique *Optimal Asymmetric Encryption Padding* (OAEP).

Suppose the RSA key is 1,024 bits and the session key is 128 bits. You create a buffer of 128 bytes (the same size as the RSA modulus) and place the session key at the end, just as in PKCS #1 Block 02. You now need 112 bytes of padding to precede the session key. The first byte is 00. The next 20 bytes, known as the seed, are random or pseudo-random. The next 20 bytes are the SHA-1 digest of some known data. (This known data is part of the algorithm identifier, the information you pass to the recipient indicating what you did to create the envelope.) The next 70 bytes are all 00, and then the last pad byte is 01. For example, assuming the session key is 16 bytes of 0xFF, at this point, you would have something that looks like this.

```
00 14 86 6A 90 11 B4 DE 48 66 25 03 9B E2 57 F5
2B BD 27 E1 47 F2 18 F0 65 41 9E 0D 1A CD B4 3D
24 C4 78 A6 A6 F3 1E 07 61 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

You're not finished yet. From the seed, you create a mask 107 bytes long. The mask-generating function is based on SHA-1. Then, you XOR

the mask with next 107 bytes, which are the digest, all the 00 bytes, the 01, and the session key. In the following, the underlined values have been masked.

```
00 14 86 6A 90 11 B4 DE 48 66 25 03 9B E2 57 F5
2B BD 27 E1 47 36 C2 B2 3A 61 C8 B8 86 20 30 1C
B9 C5 fD 13 E3 bE 37 9C F5 EF 79 1C 02 90 23 04
82 69 63 AA AF 37 30 64 66 D6 CD AF 35 58 99 BC
6F 3C 7E 14 AE 9D D1 FB F7 D1 F6 97 93 D9 B0 A6
8A D8 C4 44 87 F2 EC 77 EE 2D 9B F8 41 02 D8 50
23 00 57 49 EF D1 61 98 41 51 4D C8 A6 4D 74 A7
19 E4 4D 80 86 A9 6F AB 1E 57 98 B2 41 59 2F EA
```

Finally, you use this resulting value (the 107 bytes after the XOR operation) as a second seed to create a mask for the original seed. Then you XOR the original seed with this new mask.

```
00 8B 60 80 FE DE CA AF 72 77 4E 46 32 4D 6A F1
E3 82 37 5C DA 36 C2 B2 3A 61 C8 B8 86 20 30 1C
B9 C5 fD 13 E3 bE 37 9C F5 EF 79 1C 02 90 23 04
82 69 63 AA AF 37 30 64 66 D6 CD AF 35 58 99 BC
6F 3C 7E 14 AE 9D D1 FB F7 D1 F6 97 93 D9 B0 A6
8A D8 C4 44 87 F2 EC 77 EE 2D 9B F8 41 02 D8 50
23 00 57 49 EF D1 61 98 41 51 4D C8 A6 4D 74 A7
19 E4 4D 80 86 A9 6F AB 1E 57 98 B2 41 59 2F EA
```

The result looks random. To unpad, you skip the first 21 bytes (the decryptor knows the first byte is 00 and the next 20 bytes are the seed). You use the rest of the data as a seed (this is the second seed) to create a mask. You XOR this mask with the 20 bytes after the first byte (remember, the first byte is 00). Now you've reconstructed the original seed. You use the original seed to create a mask to XOR with the remaining 107 bytes. Then, you digest the known data and compare it to the 20 bytes after the seed. Also, you check to make sure that the next 70 bytes are all 00 and that the last byte before the session key is 01. If all these checks pass, the recipient has the session key.

OAEP has some variants. Different digest algorithms with different seed sizes are possible. The first byte might be a random byte instead of 00 (although there are mathematical limitations on what that byte can be).

Using this padding scheme means that a Bleichenbacher attack will almost certainly fail. The chances that someone could create a fake envelope that produces a valid OAEP block are so astronomically small that it will virtually never happen. The reason is the digesting. Remember that digest algorithms produce dramatically different output when the input is

changed, even slightly. A fake envelope would have to decrypt to something that by sheer chance created some digests that, after the XOR operation, created the 70 bytes of 00 followed by the one byte of 01 in the appropriate place.

Timing Attacks

How long does it take for your computer to perform a private-key operation (creating a digital signature or opening a digital envelope)? It turns out that there are slight variations in the amount of time needed to perform asymmetric algorithm operations. The actual time is dependent on the key itself and the input data.

Here's what we mean. Take two private keys, both of them the same algorithm (RSA, DH, DSA, or ECC) and the same size. Now perform the same operation (sign, encrypt, key agree) with the same data. How long *exactly* did each operation take? With one key it might take 0.2007415517 milliseconds, and with the second 0.2007415548 milliseconds. The difference is tiny, but there is a difference.

Or suppose that the time can be computed in cycles. One key makes the computation in 90,288,451 cycles, and another key uses 90,289,207 cycles. If you're not familiar with computer cycles, one cycle is one "tick" of the computer's internal clock. A hertz is one cycle per second, so a 450 megahertz (450MHz) computer can operate at 450 million cycles per second. Most processors can perform one integer addition in one cycle and one integer multiplication in two to six cycles (some processors might need 27 cycles to do one integer multiplication). So a 450MHz processor could do 450 million additions or 75 million to 225 million multiplications in one second. Actually, it gets complicated with pipelining and multiprocessing and integer units and floating-point units, but the point is that time can be measured in cycles as well as seconds.

However time is measured, the variations in time can aid an attacker. Knowing the input data (for example, what you're signing) and exactly how long it took you to perform the private-key operation (such as signing), an attacker can gain information about your private key. This is known as a *timing attack*. The attacker almost certainly needs timings from many private-key operations (each operation working on different input data) to figure out the entire key. The more exact the time, the fewer data points the attacker needs.

Highly controlled experiments on simple machines running simple software implementations have had some success in measuring the times of various operations. More success has been found timing tokens and other slow processors. But often the experiments have required hundreds if not thousands of timings to collect enough information on a particular key. Furthermore, using the *Chinese Remainder Theorem* (CRT; see Chapter 4) for RSA operations and Montgomery multiplication helps thwart the attack. (Peter Montgomery is a researcher who came up with a clever way to perform the internal operations of RSA, DH, DSA, and some ECC much faster.) Data and instruction caching may skew the measurements. Another way to defeat this attack is to prevent the attacker from knowing the input, an approach known as *blinding*. The attacker knows what the input is, but if you alter it before signing and then alter the resulting signature to compensate for the original alteration, the exact data operated on by the private key is unknown. Unfortunately, blinding is a drain on performance, adding another 40 percent to the total signature time.

In real-world applications, a timing attack may not be practical because virtually all current implementations of RSA employ CRT and the implementations of all public-key algorithms employ Montgomery math. Furthermore, attackers often have no way of knowing how long it took to perform the operation, or the measurements were not accurate enough. Possibly the target did not make enough private-key computations before changing keys.

In some situations, a timing attack may be more practical. One example is an SSL server performing private-key operations automatically. An attacker could request an SSL connection and time the response and then repeat the request, time the response again, and so on hundreds or thousands of times. For each SSL connection, the server creates a digital signature, each time signing something different. This is exactly what the attacker needs: knowledge of the data being signed, different data being signed each time, and many iterations.

No one has demonstrated a successful timing attack in real-world situations, including an attack on an SSL server. CRT and Montgomery math may be all that's needed to prevent a successful attack, or other operations may mask the signature time. But you may need to be aware of the possibility of a timing attack, especially if, in the future, you use a smart card in someone else's reader. In that case, blinding may become a prudent countermeasure.

Kerberos*

In Chapter 5, you learned about authentication using digital signatures. You may have heard about Kerberos, an alternative authenticating technique.

Kerberos is an authentication service developed by the Project Athena team at MIT, based on a 1978 paper by Roger Needham and Michael Schroeder. The first general-use version was version 4. Version 5, which addressed certain shortfalls in version 4, was released in 1994. Kerberos uses secret-key ciphers for encryption and authentication. Version 4 used only DES. Unlike a public-key authentication system, Kerberos does not produce digital signatures. Instead, Kerberos was designed to authenticate requests for network resources rather than to authenticate authorship of documents. Thus, Kerberos does not provide for future third-party verification of documents.

In a Kerberos system, a designated site on each network, called the Kerberos server, performs centralized key management and administration. The server maintains a database containing the secret keys of all users, authenticates the identities of users, and distributes session keys to users and servers that wish to authenticate one another. Kerberos requires trust in a third party (the Kerberos server). If the server is compromised, the integrity of the whole system is lost. Public-key cryptography was designed precisely to avoid the necessity to trust third parties with secrets.

Kerberos is generally considered adequate within an administrative domain; however, across domains, the more robust functions and properties of public-key systems are often preferred. Some developmental work has been done to incorporate public-key cryptography into Kerberos.

For detailed information on Kerberos, read "The Kerberos Network Authentication Service (V5)" (J. Kohl and C. Neuman, RFC 1510) at ftp://ftp.isi.edu/in-notes/rfc1510.txt.

*(Source: RSA Labs)

DH, ECDH, DSA, and ECDSA Certificates

In Chapter 6, you learned about certificates. The minimum contents of a certificate are the owner's name, public key, and the CAs signature. For RSA, the public key is the modulus and public exponent. For DH, ECDH, DSA, and ECDSA, the public key consists of the system parameters and a public value.

Often, messages contain certificates. For example, if Pao-Chi sends a signed message to Daniel, Daniel needs Pao-Chi's certificate to verify the signature. Pao-Chi can include his certificate as part of the message, saving Daniel the trouble of searching for it in public directories. If Satomi wants to pose as Pao-Chi, she could send a message with a certificate containing Pao-Chi's name but not his true public key. But she will have to get that certificate signed by a CA whose certificate was signed by the root that Daniel will use. Satomi will have to break the root's key to become the root (and hence create her own CA) or break a CA's key to create a valid certificate. That's not likely, so including the certificate in a message is no security problem.

Because messages contain certificates and because larger messages are sometimes expensive, it's often desirable to create smaller certificates. A protocol or company might demand that names be short (for example, that they carry no title, mailing address, fax number, or other such information) or that there be no extensions or attributes.

In the past, people have proposed shrinking public keys. DH, ECDH, DSA, or ECDSA keys can be compressed by excluding the system parameters. Everyone would have to get the system parameters in some other way.

But this is not a good idea. The purpose of a certificate is to guarantee that an attacker could not replace a true public key with a fake one. But if the parameters are not part of the certificate, an attacker could replace the parameters. Someone using a public key extracted from a certificate (creating a digital envelope or verifying a signature) would be certain of using the correct public value but not the correct parameters. If Satomi, for example, replaces the parameters on Pao-Chi's machine and if Pao-Chi tries to send a message to Daniel, he will create something that Daniel cannot read. Satomi almost certainly won't be able to read it either (she would still need to know Daniel's private value), so all she would be doing by changing the parameters would be creating a nuisance. This is a denial-of-service attack because her actions would deny Pao-Chi and Daniel the service of secure communication.

To prevent this problem, the company could enforce a policy in which everyone uses the same system parameters. It could create some system parameters and distribute them to all employees, each of whom would create individual public and private key pairs. This is generally not a security problem; sharing system parameters does not weaken the math. Some cryptographers warn that if "too many" people share system parameters (good luck getting one of those cryptographers to quantify "too many"), it might be possible to introduce weaknesses, but for the most part, sharing parameters does not aid an attacker. In such a situation, Satomi could not replace the system parameters on Pao-Chi's machine because he would know what the true parameters were; they're his. Presumably, Pao-Chi will have those parameters protected in such a way that Satomi could not alter them without his knowledge.

This parameters policy could work for communications among all the employees at Pao-Chi's company, but how could people outside the company guarantee a public key's authenticity? Everyone else would also need the system parameters. Suppose the company created a second certificate, this one for the parameters. That would mean two certificates would be required to verify one public key, defeating the purpose of saving space.

The best way to distribute DH, ECDH, DSA, and ECDSA keys is to include the system parameters in the certificate.

Problems with Using SSL to Protect Credit Cards

Chapter 7 describes the *Secure Socket Layer* (SSL), the latest version of which is known as *Transport Layer Security* (TLS). Many companies use SSL exclusively to protect credit card transactions. Unfortunately, that may not be a wise policy.

SSL encrypts data while in transit, so if someone runs a sniffer program on the Internet—checking traffic to see whether any credit card numbers are sent in the clear—SSL will protect the transaction. However, credit card numbers sent over the Internet usually are not stolen in transit; instead, they are stolen while in storage. Rather than eavesdropping on Internet messages, thieves break into servers storing sensitive material.

When you operate a Web site, you're essentially making your local files available for the world to see. Recall the discussion of permissions in Chapter 1. You can set the permissions on your files so that only certain

users have read or write access to them. A Web server has, in effect, set the read permission on many of its files to the entire world.

One mistake made by companies is to store the credit card numbers on the Web server. In fact, an MSNBC reporter discovered that on January 13, 2000, when he "was able to view nearly 2,500 credit card numbers stored by seven small e-commerce Web sites within a few minutes, using elementary instructions provided by a source. In all cases, a list of customers and all their personal information was connected to the Internet and either was not password-protected or the password was viewable directly from the Web site." (source: www.msnbc.com) The companies may set the permissions of the files containing the numbers to exclude the world, or they may not. It doesn't matter. As you saw in Chapter 1, simple OS permissions are no real deterrent to the majority of hackers and crackers.

The best policy is to store credit card numbers encrypted. Another possibility is to use a protocol, such as SET, in which credit cards numbers are transmitted using a digital envelope, the public key creating the envelope belonging to the issuing bank. Hence, a merchant never sees the credit card in the clear and can never store it unsecurely. Because SET has not been widely adopted, the credit card companies and banks may devise a new protocol.

One way consumers can protect themselves is to read the security policies of the e-commerce companies from which they might wish to make purchases. The following quotations indicate policies that are less than secure:

> " . . . uses SSL, an advanced encryption technology that protects your credit card information."
>
> "We use Secure Sockets Layer (SSL) technology to protect the security of your online order information."

The point is not that SSL has no value but rather that SSL does not address the storage issue. It is not a silver bullet that solves all security problems.

Here is a policy that is starting to get the right idea:

> "To ensure that your information is even more secure, once we receive your credit card information, we store it on a server that isn't accessible from the Internet."

Finally, here are quotes from a couple of security policies of Web sites that are truly interested in security.

"Every time you send us your credit card number and your billing and shipping information, we use the industry-standard Secure Sockets Layer (SSL) technology to prevent the information from being intercepted. We also encrypt your credit card number when we store your order and whenever we transfer that information to participating merchants."

"Within those systems, sensitive information is encrypted to protect your personal data, like credit card numbers."

Index

Symbols

A

B

C

D

E

F

G–H

I

J–K

L

M

N

O

P

Q-R

S

T

U

V

W

X

INTERNATIONAL CONTACT INFORMATION

AUSTRALIA
McGraw-Hill Book Company Australia Pty. Ltd.
TEL +61-2-9417-9899
FAX +61-2-9417-5687
http://www.mcgraw-hill.com.au
books-it_sydney@mcgraw-hill.com

CANADA
McGraw-Hill Ryerson Ltd.
TEL +905-430-5000
FAX +905-430-5020
http://www.mcgrawhill.ca

GREECE, MIDDLE EAST, NORTHERN AFRICA
McGraw-Hill Hellas
TEL +30-1-656-0990-3-4
FAX +30-1-654-5525

MEXICO (Also serving Latin America)
McGraw-Hill Interamericana Editores S.A. de C.V.
TEL +525-117-1583
FAX +525-117-1589
http://www.mcgraw-hill.com.mx
fernando_castellanos@mcgraw-hill.com

SINGAPORE (Serving Asia)
McGraw-Hill Book Company
TEL +65-863-1580
FAX +65-862-3354
http://www.mcgraw-hill.com.sg
mghasia@mcgraw-hill.com

SOUTH AFRICA
McGraw-Hill South Africa
TEL +27-11-622-7512
FAX +27-11-622-9045
robyn_swanepoel@mcgraw-hill.com

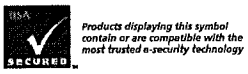
UNITED KINGDOM & EUROPE (Excluding Southern Europe)
McGraw-Hill Publishing Company
TEL +44-1-628-502500
FAX +44-1-628-770224
http://www.mcgraw-hill.co.uk
computing_neurope@mcgraw-hill.com

ALL OTHER INQUIRIES Contact:
Osborne/McGraw-Hill
TEL +1-510-549-6600
FAX +1-510-883-7600
http://www.osborne.com
omg_international@mcgraw-hill.com

BSAFE®

ENCRYPTION FROM THE MOST TRUSTED NAME IN E-SECURITY

Whether you need core cryptography routines, digital certificate management components, or fully implemented protocol for your application, the RSA BSAFE SDKs provide you with all of the components you need to make your applications absolutely safe and secure. By using RSA BSAFE products, your staff can save months of development time, enabling you to roll out mission-critical systems earlier and with more confidence. Contact RSA Security, your choice for authentication, encryption and PKI.



Products displaying this symbol contain or are compatible with the most trusted e-security technology

SECURITY™

The Most Trusted Name in e-Security®

www.rsasecurity.com/go/rsapress/CRYPTO

SECURITY™

The Most Trusted Name in e-Security®

The Company

RSA Security Inc. is the most trusted name in e-security, helping organizations build secure, trusted foundations for e-business through its two-factor authentication, encryption and public key management systems. RSA Security has the market reach, proven leadership and unrivaled technical and systems experience to address the changing security needs of e-business and bring trust to the new online economy.

A truly global company with more than 8,000 customers, RSA Security is renowned for providing technologies that help organizations conduct e-business with confidence. Headquartered in Bedford, Mass., and with offices around the world, RSA Security is a public company (NASDAQ: RSAS) with 2000 revenues of $280 million.

Our Markets and Products

With the proliferation of the Internet and revolutionary new e-business practices, there has never been a more critical need for sophisticated security technologies and solutions. Today, as public and private networks merge and organizations increasingly expand their businesses to the Internet, RSA Security's core offerings are continually evolving to address the critical need for e-security. As the inventor of leading security technologies, RSA Security is focused on three core disciplines of e-security.

Public Key Infrastructure

RSA Keon® public key infrastructure (PKI) solutions are a family of interoperable, standards-based PKI software modules for managing digital certificates and creating an environment for authenticated, private and legally binding electronic communications and transactions. RSA Keon software is designed to be easy to use and interoperable with other standards-based PKI solutions, and to feature enhanced security through its synergy with the RSA SecurID authentication and RSA BSAFE encryption product families.

Authentication

RSA SecurID® systems are a leading solution for two-factor user authentication. RSA SecurID software is designed to protect valuable network resources by helping to ensure that only authorized users are granted access to e-mail, Web servers, intranets, extranets, network operating systems and other resources. The RSA SecurID family offers a wide range of easy-to-use authenticators, from time-synchronous tokens to smart cards, that help to create a strong barrier against unauthorized access, helping to safeguard network resources from potentially devastating accidental or malicious intrusion.

Encryption

RSA BSAFE® software is embedded in today's most successful Internet applications, including Web browsers, wireless devices, commerce servers, e-mail systems and virtual private network products. Built to provide implementations of standards such as SSL, S/MIME, WTLS, IPSec and PKCS, RSA BSAFE products can save developers time and risk in their development schedules, and have the security that only comes from a decade of proven, robust performance.

Commitment to Interoperability

RSA Security's offerings represent a set of open, standards-based products and technologies that integrate easily into organizations' IT environments, with minimal modification to existing applications and network systems. These solutions and technologies are designed to help organizations deploy new applications securely, while maintaining corporate investments in existing infrastructure. In addition, the Company maintains active, strategic partnerships with other leading IT vendors to promote interoperability and enhanced functionality.

Strategic Partnerships

RSA Security has built its business through its commitment to interoperability. Today, through its various partnering programs, the Company has strategic relationships with hundreds of industry-leading companies—including 3COM, AOL/Netscape, Ascend, AT&T, Nortel Networks, Cisco Systems, Compaq, IBM, Oracle, Microsoft and Intel—who are delivering integrated, RSA Security technology in more than 1,000 products.

Customers

RSA Security customers span a wide range of industries, including an extensive presence in the e-commerce, banking, government, telecommunications, aerospace, university and healthcare arenas. Today, more that 8 million users across 7,000 organizations—including more than half of the Fortune 100—use RSA SecurID authentication products to protect corporate data. Additionally, more than 500 companies embed RSA BSAFE software in some 1,000 applications, with a combined distribution of approximately one billion units worldwide.

Worldwide Service and Support

RSA Security offers a full complement of world-class service and support offerings to ensure the success of each customer's project or deployment through a range of ongoing customer support and professional services including assessments, project consulting, implementation, education and training, and developer support. RSA Security's Technical Support organization is known for resolving requests in the shortest possible time, gaining customers' confidence and exceeding expectations.

Distribution

RSA Security has established a multi-channel distribution and sales network to serve the enterprise and data security markets. The Company sells and licenses its products directly to end users through its direct sales force and indirectly through an extensive network of OEMs, VARs and distributors. RSA Security supports its direct and indirect sales effort through strategic marketing relationships and programs.

Global Presence

RSA Security is a truly global e-security provider with major offices in the U.S., United Kingdom, Singapore and Tokyo, and representation in nearly 50 countries with additional international expansion underway. The RSA SecurWorld channel program brings RSA Security's products to value-added resellers and distributors worldwide, including locations in Europe, the Middle East, Africa, the Americas and Asia-Pacific.

For more information about RSA Security, please visit us at:
www. rsasecurity.com.

SOFTWARE AND INFORMATION LICENSE

The software and information on this CD-ROM (collectively referred to as the "Product") are the property of RSA Security Inc. ("RSA Security") and are protected by both United States copyright law and international copyright treaty provision. You must treat this Product just like a book, except that you may copy it into a computer to be used and you may make archival copies of the Products for the sole purpose of backing up our software and protecting your investment from loss.

By saying "just like a book," RSA Security means, for example, that the Product may be used by any number of people and may be freely moved from one computer location to another, so long as there is no possibility of the Product (or any part of the Product) being used at one location or on one computer while it is being used at another. Just as a book cannot be read by two different people in two different places at the same time, neither can the Product be used by two different people in two different places at the same time (unless, of course, RSA Security's rights are being violated).

RSA Security reserves the right to alter or modify the contents of the Product at any time.

This agreement is effective until terminated. The Agreement will terminate automatically without notice if you fail to comply with any provisions of this Agreement. In the event of termination by reason of your breach, you will destroy or erase all copies of the Product installed on any computer system or made for backup purposes and shall expunge the Product from your data storage facilities.

LIMITED WARRANTY

RSA Security warrants the CD-ROM(s) enclosed herein to be free of defects in materials and workmanship for a period of sixty days from the purchase date. If RSA Security receives written notification within the warranty period of defects in materials or workmanship, and such notification is determined by RSA Security to be correct, RSA Security will replace the defective diskette(s). Send request to:

RSA Press
RSA Security Inc.
2955 Campus Drive
Suite 400
San Mateo, CA 94403

The entire and exclusive liability and remedy for breach of this Limited Warranty shall be limited to replacement of defective CD-ROM(s) and shall not include or extend any claim for or right to cover any other damages, including but not limited to, loss of profit, data, or use of the software, or special, incidental, or consequential damages or other similar claims, even if RSA Security or The McGraw-Hill Companies, Inc. ("McGraw-Hill") has been specifically advised as to the possibility of such damages. In no event will RSA Security's or McGraw-Hill's liability for any damages to you or any other person ever exceed the lower of suggested list price or actual price paid for the license to use the Product, regardless of any form of the claim.

RSA SECURITY INC. AND THE McGRAW-HILL COMPANIES, INC. SPECIFICALLY DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Specifically, neither RSA Security nor McGraw-Hill makes any representation or warranty that the Product is fit for any particular purpose and any implied warranty of merchantability is limited to the sixty day duration of the Limited Warranty covering the physical CD-ROM(s) only (and not the software or information) and is otherwise expressly and specifically disclaimed.

This Limited Warranty gives you specific legal rights; you may have others which may vary from state to state. Some states do not allow the exclusion of incidental or consequential damages, or the limitation on how long an implied warranty lasts, so some of the above may not apply to you.

This Agreement constitutes the entire agreement between the parties relating to use of the Product. The terms of any purchase order shall have no effect on the terms of this Agreement. Failure of RSA Security to insist at any time on strict compliance with this Agreement shall not constitute a waiver of any rights under this Agreement. This Agreement shall be construed and governed in accordance with the laws of Massachusetts, irrespective of its choice of law principles. If any provision of this Agreement is held to be contrary to law, that provision will be enforced to the maximum extent permissible and the remaining provisions will remain in force and effect.

Learn how cryptography works—from the leading authority in e-security

Cryptography is one of the smartest ways to protect the information on your network and reduce the risk of security breaches and attacks from hackers. And because implementing cryptography is a complex process, you need the practical advice and proven techniques contained inside this official guide. Written by insiders at RSA Security, this expert resource explains the differences between symmetric-key and public-key cryptography, how PKI and X.509 affect security, how the RSA algorithm works within protocols, and much more. You'll also read actual case studies detailing different types of security vulnerabilities and what types of cryptography applications would prevent attacks.

This book will show you how to:

- Distinguish different types of symmetric-key encryption algorithms and know where each is best used
- Find out how password-based encryption works
- Communicate safely over unsecure channels using public-key technology
- Use public-key technology for authentication and non-repudiation
- Recognize how corporations use cryptography to improve security through real-world case studies
- Get details on current PKI standards and technology—including vendor information
- Understand X.509 certificates and directory structures
- Get an operational overview of widely-used protocols—including IPSec, SSL, and SET
- View cryptography from different perspectives—corporations, developers, and users
- Effectively use digital signatures and hardware solutions—smart cards, tokens, key storage devices, and more

Improve security and protect your company's information with the most authoritative guide to cryptography available.

ONLY AVAILABLE FROM RSA SECURITY

- RSA Laboratories' complete FAQ 4.1
- Public-Key Cryptography Standards
- Crypto Bytes Technical Newsletters

ABOUT THE AUTHORS:

STEVE BURNETT has degrees in math from Grinnell College in Iowa and The Claremont Graduate School in California. He has spent most of his career converting math into computer programs, first at Intergraph Corporation and now with RSA Security. A frequent speaker at industry conferences and college campuses, Steve is the lead engineer for RSA's BSAFE Crypto-C and Crypto-J products, which are general purpose cryptography software development kits in C and Java.

STEPHEN PAINE has worked in the security field throughout most of his career—formerly for the United States Marine Corps and for SUN Microsystems. He is currently a systems engineer for RSA Security, where he explains security concepts to corporations and developers worldwide and provides training to customers and RSA employees.

OSBORNE
REQUIRED READING for the Information Age
A Division of The McGraw-Hill Companies

NETWORKING/SECURITY

7 83254 03608 6

www.osborne.com

Book P/N 0-07-213138-1 of
ISBN 0-07-213139-X

90000
9 780072 131390

$59.99 USA | £43.99 UK