Web Server Technology

The Advanced Guide for World Wide Web Information Providers

Nancy J. Yeager
Robert E. McGrath

National Center for Supercomputing Applications

Morgan Kaufmann Publishers, Inc.
San Francisco, California

Sponsoring Editor	Michael B. Morgan
Production Manager	Yonie Overton
Production Editor	Elisabeth Beller
Text Design	Mark Ong, Side by Side Studios
Cover Design	Martin Heirakuji Graphic Design
Cover Photograph	Photonica/Joshua Sheldon
Copyeditor	Ken DellaPenta
Proofreader	Judith Abrahms
Composition	Nancy Logan
Illustration	Cherie Plumlee
Indexer	Valerie Robbins
Printer	Courier Corporation

Morgan Kaufmann Publishers, Inc.
Editorial and Sales Office
340 Pine Street, Sixth Floor
San Francisco, CA 94104-3205
USA

Telephone	415/392-2665
Facsimile	415/982-2665
E-mail	mkp@mkp.com
WWW	http://www.mkp.com

Order toll free 800/745-7323

Printed in the United States of America

00 99 98 97 96 5 4 3 2 1

Library of Congress Cataloging-in-Publication Data is available for this book.

ISBN 1-55860-376-X

CHAPTER EIGHT

Digital Commerce: Risks, Requirements, and Technologies

Whether you are a Web service provider or a casual Web user, it is important to understand your level of risk in using a secure Web service. This chapter will aid you in that task by defining requirements for secure services, describing security-enabling technologies, and presenting a method to evaluate a secure Web service. Lastly, we will critique several existing secure Web system models against the requirements and methods we have defined.

The Web was designed and is highly successful as an easy-to-use method for distributing public information. The Web is wide open. All information on the Web is public, and the Web is built upon a public network, the Internet. Many commercial Web services contain catalogs of products, services, and prices, and corporations use the Web to distribute information widely. Many users now want to take the technology one step further, to disseminate information in a controlled way. To do this, they need to incorporate confidentiality and access control to a subset of their Web documents. The same security mechanisms that restrict access to a set of documents to qualified individuals could be used to popularize commercial transactions over the Web. These mechanisms can also be used to address online privacy issues, such as protecting the confidentiality of medical records or credit ratings.

Computer security measures are intended to reduce the risks of using the system. The introduction of security mechanisms can never totally eliminate all risk; it can only diminish risks to an acceptable level. Any consideration of security must begin with risks: what are the risks and how can they be addressed? Besides reducing risks, security technologies aim to increase confidence and trust in the system. Customers will not use a system if they do not trust it to safeguard their assets and interests.

Who will be liable when the security services fail in a commercial Web service—the service provider or the customer? The answer for commercial Web services is still, as yet, undefined. However, as has been seen in similar electronic commerce systems, the service provider may be liable, in varying degrees, for the technological failures of the system (Anderson 1994; Gifford et al. 1995). Whether you are the customer or the service provider, liability is serious business, and your risks warrant a closer analysis.

In order to begin analyzing the risk incurred by digital commerce or secure Web systems, let's first look at the process of a real-world commercial transaction model.

8.1 A Familiar Model for Commercial Transactions: Credit Cards

When credit cards were first introduced, many doubted that the general public would trust such a mechanism for commerce. Today the Web is at the same point in its commercial evolution: no one knows whether it will be widely accepted as a vehicle for commerce.

In a credit card transaction, the ownership of a credit card *identifies* an individual for the purpose of the commercial transaction. The owner has possession of the card and his or her signature matches the one that is signed on the receipt of goods. As we look at commercial transactions of all kinds we will find that this identification process—that the owner of the card is truly the individual authorized to use the card—is the most crucial measure for assessing a secure transaction of any kind. The better the identification process, the less the risk assumed by the service provider or customer.

In the case of credit cards, some merchants and customers have abbreviated the identification process; they don't require a signature or don't review the signature. For example, when the customer presents only the number (the proof of possession) of the card over the telephone, there is little way to bind the authorized user's identity with ownership of the card. Abbreviating the identification process in this way decreases the effectiveness of the mechanism and increases the risk of an illegal transaction. If a signature is not required, it is much easier

for a disreputable third party to intercept the credit card number (and expiration date) and purchase goods and services; therefore, the risk of loss due to fraud is greatly increased. Today credit card companies accept the liability for stolen card numbers, provide help to customers with stolen card numbers, and charge more for their services to businesses that accept credit card numbers over the phone. The convenience of telephone transactions outweighs the risk of losses due to fraud. However, the business, and ultimately the customer, pays more for this added convenience.

8.2 Identifying Yourself

As we look at commercial transactions of all kinds, we will find that this identification process is the most important measure of effectiveness for secure transactions. First an individual asserts or claims an identity and then that assertion is verified. This assertion and verification together are called the *authentication* process.

In the human world, the identity of each person is founded on their physical existence. When there is a person standing in front of us, we use all our senses to identify them, by what they look like, how they sound, what they do and say, and so on. On a computer system these natural cues are entirely missing. To the computer, a person is just a collection of data; it has no way to tell which body the data is supposed to belong to, or even if the person ever really existed outside the computer. In one sense, the problem of authentication is to come up with some way that the computer can tell people apart and verify a person's honesty when he asserts an identity.

Most commercial transactions and secure applications employ some form of authentication. There are three factors that can be used in the authentication process (Miller 1994):

1. Knowledge—something a person knows
2. Possession—something a person owns
3. Characteristic—something a person is

All of these may serve to uniquely identify one individual person.

Many authentication systems operate on two-factor authentication, requiring two different sources of identification. The idea, of course, is that when the two means of identification match, it is likely that only the right person could supply that matching identification.

Credit card users use two-factor authentication for purchasing goods and services: you present a card (possession) and you sign for the purchase (an individual's characteristic signature). Automatic teller machines use two-factor authentication—you present a bank account card (possession) and input a personal identification number (PIN) (knowledge).

8.2.1 Biometrics

The last authentication factor, the characteristic, is so well developed that it has been recognized as a scientific field. Biometrics is the study of the measurement of physiological and behavioral traits. It is used to study human diversity and may also be used for identification based on physiological or behavioral traits.

Biometric methods analyze human physiological traits such as fingerprints, size and shape of hand and fingers, and retinal patterns. Biometric methods can also analyze behavioral characteristics such as an individual's signature, voice, or speech patterns. Even the speed and pattern of an individual's typing at a keyboard has been examined as a behavioral biometric identification method.

Physiological measurements are not always convenient or unobtrusive. The most positive identification possible is a DNA sample, but it is not reasonable to perform such an expensive procedure to verify a credit card purchase! Behavioral biometric techniques may be easier to measure but are subject to more variability than physiological biometric techniques. For example, the quality of your voice may change if you have a cold and your signature may change as you mature. No method is perfect, so it is important to consider the likely mistakes and their consequences. Biometric identification methods may fail in two ways: they may mistakenly confirm an identity when it is the wrong person (a *false acceptance*) and they may mistakenly reject the identity of the right person (a *false rejection*) (Miller 1994). Different biometric methods have different likelihoods of each type of error, which are expressed in two numbers, the false acceptance rate (FAR) and the false rejection rate (FRR). The selection of a biometric method must be tailored to each application's security requirements, based on the consequences of each type of mistake.

Consider, for example, the high security requirements of an entrance to a military laboratory. We would need a biometric method that could not easily be fooled into admitting unauthorized personnel. We would also require our biometric method to permit authorized applicants to enter the secured facility. Otherwise, qualified applicants would become frustrated and might eventually give up their entrance attempts. In this example, it would be of the utmost importance to deny access to unauthorized individuals—our biometric method must have a small FAR. If need be, we could tolerate a method that denied access to a few authorized applicants; we could always provide them with the phone number of a security officer to call. So we could tolerate a few false rejections—or a relatively high FRR. We could select fingerprints as a biometric system for authentication at our secure site. Fingerprints have a FAR of less than .0001 percent and a FRR of 2 to 3 percent. In other words, it is very unlikely that someone else's fingerprint will be taken to be yours, but the identification system sometimes might not recognize your fingerprint as your own.

The requirements of military security are completely different from the needs of commerce. What are the requirements for digital commerce? First, costs must

be kept reasonable. And second, customer satisfaction and convenience are paramount concerns. The service provider cannot afford to aggravate even a small number of customers or it will lose their business. Incorrectly rejecting the identity of a customer is very bad for business. So, unlike the military example, digital commerce demands a very low FRR. Furthermore, even a reliable method must be very quick and must be inexpensive enough to be widely used. It is better to lose a little money to thieves than to lose good customers because of an inconvenient or expensive identification method.

Most biometric methods cannot be used in digital commerce because they are either too expensive for the service provider or too inconvenient or too slow to satisfy the customer. Consider, for example, an automated signature verification system for use by banks for account and credit card verification. The system's cost is not prohibitive, only $1000. Banks have been slow to use this biometric method, though, because they are afraid their customers would not tolerate the system's FRR of their perfectly legitimate transactions (Miller 1994).

8.3 The Web, Security, and the Internet

We have seen in Chapter 7 that there are no privacy guarantees on the Internet today. With the right tools and a bit of technical expertise, someone can eavesdrop on conversations and capture data. For this reason, conducting digital commerce transactions on a public network like the Internet is risky business. But each day U.S. banks securely transfer a trillion dollars electronically (Adam 1992). How is this done securely? Typically, commercial transaction services, like electronic banking applications, operate over private networks, such as their own leased networks or the networks that make up the telephone system. The privacy of the standard telephone service today relies on the physical security of the network—the fact that no one can easily tap the telephone line. So these private networks are a more secure place to conduct transactions.

This section considers the requirements for digital commerce: what must be done to make the Internet a viable business environment. Imagine you are the purchasing agent for a pharmaceutical company that produces aspirin. You purchase a chemical called phenol as a raw material for your manufacturing process from your major supplier, Victor Chemical Company. VCC is an advanced company; it has a Web page that displays its entire product line, complete with catalog numbers, availability, and current prices.

Recently, Victor enhanced its online Web service with the capability to place chemical orders via its Web service. As a regular customer of Victor Chemical, you have a standing credit account number. The Web service displays a form, such as the example shown in Figure 8.1, for the customer to fill in with their credit card number or standing account number. This form is sent over the network to Victor's Web server, as explained in Chapter 3. Your order is received at

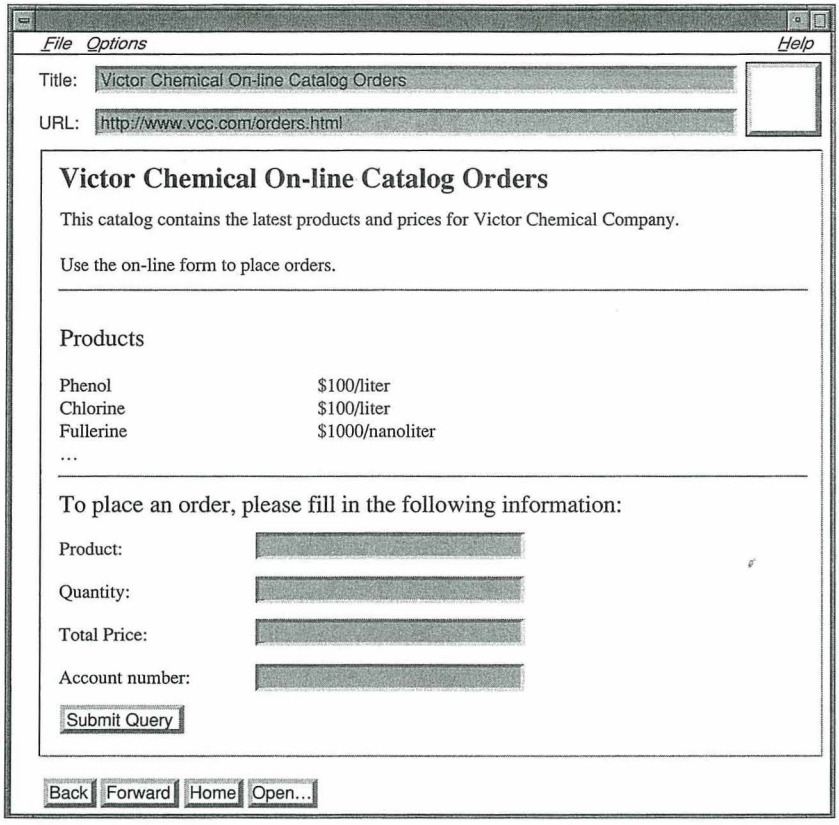

Figure 8.1 Placing a purchase order via the Web.

Victor and charged to the account number submitted. What's actually happening as you transmit your account number and what are the risks? Is it safe to use this new facility?

To answer this question, it is easiest to break the problem down into its components. We must look at how secure the message is upon creation, transmission, and receipt: the end-to-end security of the transaction. The Web forms service is a Web application running on top of an insecure public network, the Internet.

So your credit account number could be captured as it is transmitted to Victor's Web server. Additionally, someone may be able to masquerade as you on your computer and send bogus messages. The receiver would have no way to tell that the message was not legitimate.

Even if nothing unusual occurs, many transactions on the Internet are traceable. The sources and destinations of the messages can be discovered by snoopers. This in itself may deter the use of the Internet for some purposes. People often do not want their business transactions monitored.

8.4 Interim Digital Commerce Services for the Web

Despite the lack of security guarantees, many people want to use the Internet for commerce today, so several creative schemes have sprung up to circumvent the Internet security problems (Cain and McGrath 1995; Werner and DeAngelis 1995; Sefferud 1995). In many of these systems, an Internet commerce service acts as a go-between for the business and the customers, collecting the customers' credit card information over the telephone.

In one system (Werner and DeAnglelis 1995), the customer shopping on the Web fills out the Web form with their desired purchases, their phone number, and a time when they will be available at the given phone number to supply their credit card information. The business collects the order and then either calls the customer back manually or uses a programmed voice robot to call the customer at the specified time to collect the customer's credit card number. This scheme is clearly only as secure as any phone-order business and is not nearly as convenient for the customer.

In another scheme (Sefferud 1995), an internet commerce broker (ICB) acts as a go-between, collecting the credit card information of customers, collecting payment from the customers for purchases, and crediting the seller's account (see Figure 8.2). Here's how the scheme works:

1. The customer registers (over the phone) with the ICB and supplies credit card information. The ICB assigns the customer an ICB account number. Thereafter, the customer uses his ICB account number to make purchases.
2. As a customer places a transaction with a seller, he supplies his ICB account number.
3. The seller reports the transaction to the ICB.
4. The seller may validate the customer's ICB account number with the ICB before (or after) sending the purchased item to the customer.
5. The requested item, typically a document, is sent to the customer.
6. After the ICB receives the transaction report from the seller, it sends electronic mail to the customer asking for verification that they purchased the item.
7. The customer confirms via electronic mail that they did indeed purchase the item.
8. The ICB charges the customer's credit card account for the purchased item.

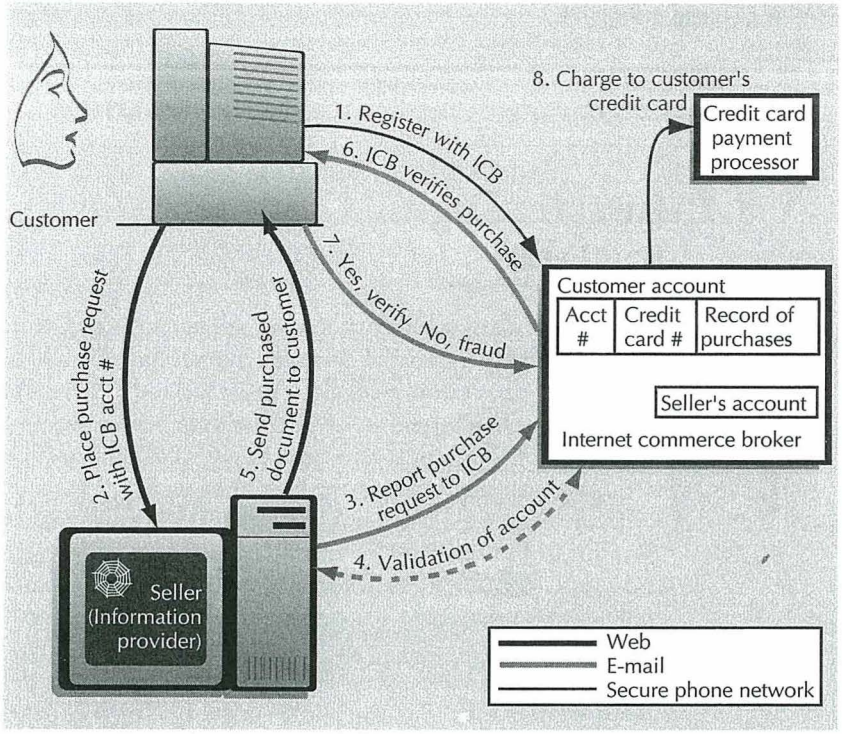

Figure 8.2 Interim digital commerce services on the Web.

This system is tailored to selling information, or "information commerce"—transactions not involving the purchase of physical goods or services. This would mean customers could purchase many documents on the Web, each for a small charge. The ICB collects all the transactions during a payment cycle. At the end of a payment cycle, the ICB bills the customer in the conventional way by calling a credit card processor over secure phone lines. The ICB then transfers remunerations into the seller's ICB account. The seller bears the risk of nonpayment by the customer. Deadbeat customers run the risk of having their accounts terminated by the ICB. This system assumes that the phone and electronic-mail services over the Internet are trustworthy. We have learned that Internet services like electronic mail are not necessarily secure.

These schemes have limited usefulness in the long term. They fall short in customer convenience and they do not provide what we suspect businesses really want to offer: spontaneous, secure commercial transactions via the Web. Let's take a closer look at the requirements for such a system.

8.5 Requirements for Digital Commerce

Let's return to the example of the Victor Chemical Company's Web service and summarize the security requirements we would want and expect from a Web service. The system must provide three important assurances: confidentiality, authenticity, and message integrity. It would also be desirable to have the option to execute spontaneous secure transactions and to use anonymous (cash) transactions.

1. *Confidentiality* The Web service must ensure that private transactions can't be captured and read by others. No one should be able to eavesdrop on our conversation and capture the account number we are transmitting in our order form. If the eavesdropper had the account number, they could connect to the Victor Chemical Web order form and purchase goods in our name.
2. *Authenticity* The Web service must ensure that "we are who we say we are." *Both parties* must be confident of each other's identity. Without this assurance, someone could make illegal purchases in our name or masquerade as the Victor Chemical Company's Web server and collect our credit account number as we submit our order form.
3. *Message integrity* The Web service must ensure that the message received is actually the message sent. It must not be possible to intercept and alter part or all of our order as it is transmitted over the network.
4. *Option for spontaneous secure transactions* We might prefer to conduct a transaction in a secure fashion with whomever we wanted, whenever we wanted. We might prefer not to have to register or have a login created on each Web server that conducted our transactions.
5. *Option for anonymous transactions* Wouldn't it be great if we had a cash equivalent in digital electronic form? This service would ensure that your purchase couldn't be logged and traced back to you. This is important because when each transaction is traceable, it is very easy for computers to accumulate large amounts of information about people based on what they buy.

8.6 Technology to Meet These Requirements

Today there is a whole range of security applications that meet these security service requirements to various degrees: phone cards, ATM cards, security cards or smartcards (for office and computer room access), and road toll payment card systems. In the future, Web applications for commerce, education, and health care will be called upon to meet these security requirements.

Let's look at the basic technologies that enable computer programs to meet these security requirements before we examine the special problems of integrating these security services into the World Wide Web.

8.6.1 Cryptography

Computer applications use *cryptographic algorithms* to provide a wide range of security services. An algorithm is a procedure or set of rules to solve a problem. A cryptographic algorithm is, in its simplest form, two sets of rules. One set of rules scrambles, or *encrypts,* the messages so that they cannot be understood. The other set of rules unscrambles, or *decrypts,* the messages so that they can once again be easily read. Cryptography can be used to protect information by restricting access to only a set of authorized individuals—those who have the ability to unscramble the message.

Conceptually, TV cable service providers use a form of cryptography to limit access to some of the TV channels they provide. The TV cable service providers begin with a pure TV channel signal. They then scramble, or encrypt, the signal. This message in an unintelligible form is called *ciphertext,* and the encryption process is a cryptographic algorithm, or a *cipher.* The scrambled signal is then broadcast through the cable company's network to all customers. If the customer has paid for the TV channel, the cable company provides to the customer a device called a *decoder.* This decoder decrypts the scrambled TV signal, converting it back to the regular signal, and displays it on the TV. Those customers who have not paid for the TV channel do not have the correct decoder, and they see only a jumbled mess on the pay channel. So pay-per-view cable TV service is restricted through cryptography, and what the customer buys is the ability to decode the signal.

Cryptography has been used in wartime since before the Roman Empire for communicating military strategies (Kahn 1967). In classical cryptography, all operations are performed on characters. Individual characters or groups of characters within a word or phrase are substituted or transposed. Everyone is probably familiar with newspaper cryptograms, in which a message is encoded by substituting letters. An A in the message is changed to a B in the cryptogram, B is

changed to C, and so on. Figure 8.3 shows a simple example of a cryptogram constructed by a simple substitution cipher—essentially the same type of code used by Julius Caesar two thousand years ago.

Cryptographic experts within a given army devised algorithms for transposing and substituting characters in their messages. Naturally, there also developed a set of skilled cipher analysts, or cryptanalysts, for capturing and decoding their enemies' ciphertext. The skills of these cryptographers and cryptanalysts have proven to be key to winning wars, perhaps most notably World War II (Kahn 1967; Bamford 1983).

Many character-based cryptographic algorithms, such as newspaper cryptograms, are easily cracked; that is, they can be deciphered by people other than the intended recipient. With the advent of computers, the complexity of cryptographic algorithms increased dramatically, as did the ability to crack codes. Cryptography is not based on words or letters of human language when executed by a computer. Instead, the cryptographic operations and operands are based on the computer's language: binary numbers, or bits. The bits that represent the contents of a message (text, pictures, audio) are subject to mathematical operations, such as addition, subtraction, multiplication, and division.

Computers use cryptography in the same way as the TV cable service. The data generated by a computer program begins as an unscrambled, or *plaintext,* message. An encryption algorithm on the sender's computer converts the message to ciphertext. The message is transmitted over the computer network as ciphertext. The receiver decrypts the ciphertext, converting the message back to its original plaintext form. This process is illustrated in Figure 8.4.

When used by computers today, cryptographic algorithms are carefully designed ordered sequences of mathematical operations that use one or more variables called *keys.* Enciphering and deciphering operations are based on binary-number keys. These keys are inputs to cryptographic algorithms just as the seed variable is input to a random number generator (Knuth 1981). The unique key chosen makes the result of encrypting data using the algorithm unique; selection of a different key causes the ciphertext produced to be different. The whole idea is that the ciphertext is different for each message and for each different key. The original message can be recovered from the ciphertext only by using the correct key and the same cryptographic algorithm used to encipher it.

Secret message:	`The lark is on the wing.`
Ciphertext:	`Uif mbsl jt po uif xjoh.`

Figure 8.3 A cryptogram constructed using a letter-by-letter substitution.

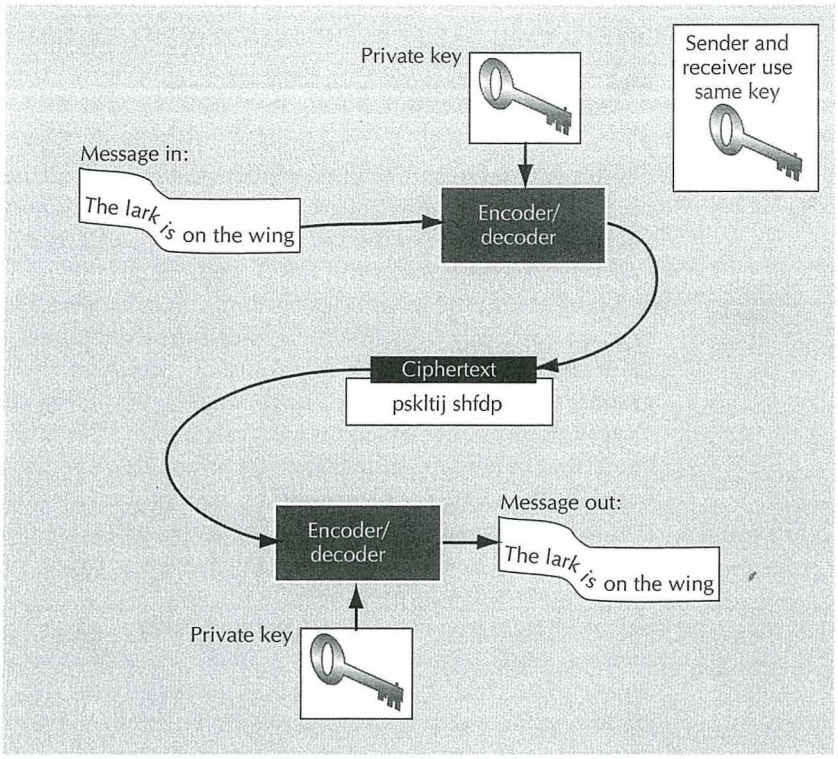

Figure 8.4 Private key cryptography. The encoder and the decoder are identical and use the same key.

Keys can be any size, but typically they are very large numbers. The Data Encryption Standard (DES), a widely used cryptographic algorithm, uses a 56-bit key. A 56-bit number (2^{56}) can represent a number as large as 7×10^{16}. To get an idea of the size of this number, 10^{18} (or 2^{61}) is an estimate of the total lifetime of the universe expressed in seconds. The entire number of people alive on the Earth is about 5×10^9 (5 billion people, or 2^{32}). Some implementations of another popular algorithm, the Rivest, Shamir, and Adleman (RSA) algorithm, use 2048-bit keys (Bidzos 1991; Garfinkel 1995; Schneier 1994). Imagine the magnitude of a number that 2048 bits would represent! If written out in zeros and ones, this key would be more than a page long!

In digital commerce systems, the possession of a key is associated with a user's identity—each user may own a unique key. The computers determine who is who by the cryptographic key that is presented. Since a person may maintain more than one role in their life, they may well need a set of keys to use. For example, I might use one key to represent my identity as bank president and conduct

a bank transaction on behalf of a bank patron. While at home, I might use another key to establish my identity as an individual customer of the bank and withdraw money from my personal account. And I would have an entirely different key to borrow books from the library.

The cryptographic algorithms that we have been discussing belong to a class of algorithms called *private key cryptography,* because the decoding key must be kept secret to protect the information. Private key cryptography is one of three classes of algorithms used by computer applications today to fulfill the requirements for digital commerce. The other two are public key cryptography and hashing algorithms. The next sections look at each of these three classes of algorithms.

8.6.2 Private Key Cryptography

Private key cryptography enables computer applications to fulfill the requirement of confidentiality: private transactions and data can't be captured and read by others. We have seen how this requirement is fulfilled for the cable TV service. The encrypted TV signal can be safely sent over public satellite and cable channels. If the data is captured on the network, it can't be read because it is ciphertext. One would need a decoder with the correct key to read the ciphertext.

Private key cryptography uses one key, known to both the sender and the receiver of the message. This key is used to encrypt and to decrypt the transmitted message. Data can be recovered from ciphertext only by using exactly the same key and the same algorithm used to encipher it (National Institute of Standards 1993a). Figure 8.5 shows how this works. Since the key must be kept secret, it is called a *private key.* The encryption and decryption operations are sometimes described as being symmetric, so private key cryptography is also called a *symmetric key* system.

8.6.3 Public Key Cryptography

Public key cryptography is used to fulfill the authenticity requirement: "I am who I say I am." Unlike private key cryptography, public key cryptography uses two keys, a matched keypair. The keypair consists of a private component and a public component. The private component of the keypair is known only to the owner of the keypair. The public component of the keypair is available to everyone.

These keypair components are related mathematically such that ciphertext created using one component of the keypair can only be decrypted using the other component of the keypair (MasterCard 1995). The special mathematical properties of the keypair rely on the fact that while some mathematical

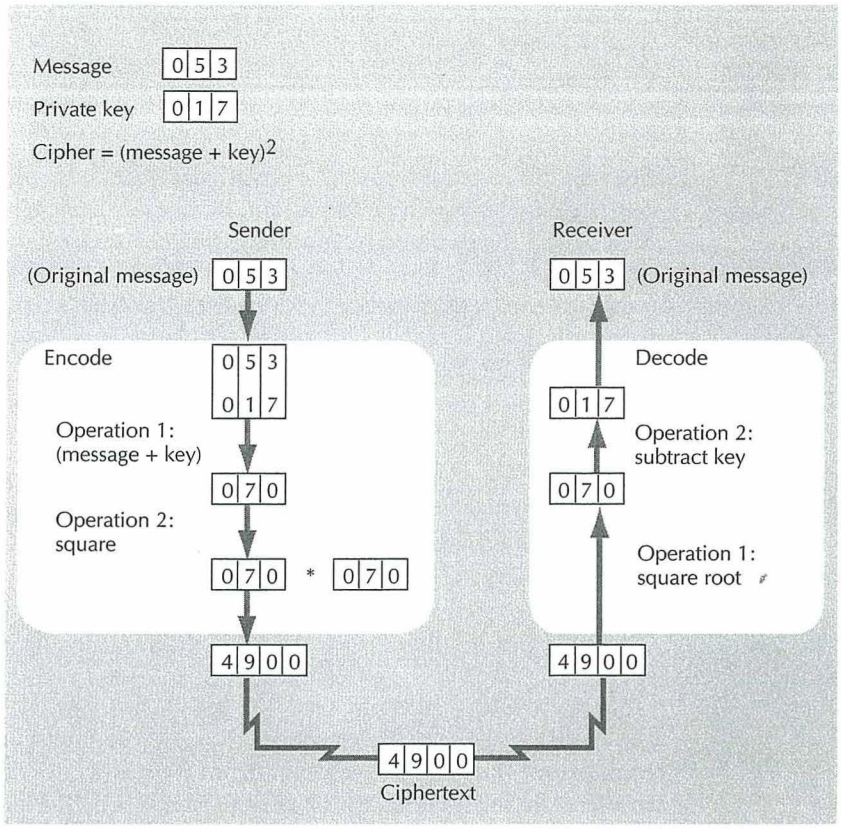

Figure 8.5 Step-by-step private key operation.

operations (addition and substraction, for example) can be performed easily in both directions, other mathematical operations (multiplication and factoring, for example) can be performed easily only in one direction. For example, the Rivest, Shamir, and Adleman algorithm is based on the fact that multiplying large prime numbers is much easier than factoring the product (result) of that multiplication. Other public key algorithms are based on different mathematical operations. (A detailed explanation of the mathematics behind public key cryptography can be found in Schneier 1994 and Garfinkel 1995.)

As seen in Figure 8.6, public key algorithms work in either direction. A sender may use her own keypair or the recipient's keypair for the encoding and decoding operations. So if a sender wants to send a confidential message, she can encrypt the message using her own private key component of her keypair (SKEYpriv), which is known only to her (see Figure 8.6a). The only way the receiver can decode the scrambled message is to use the public component, or

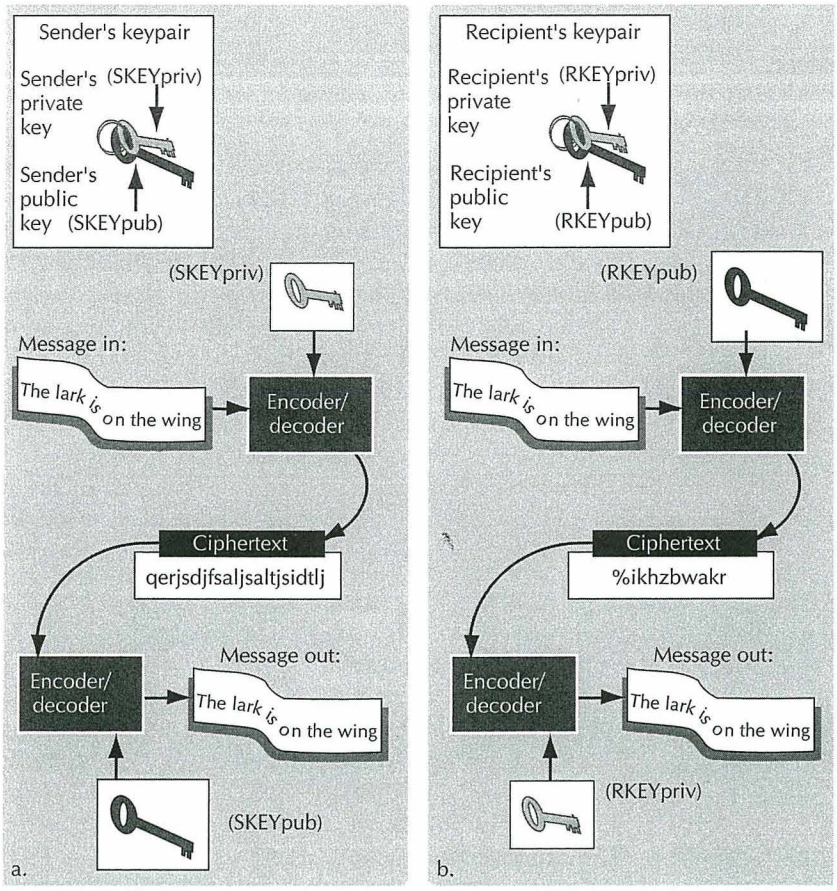

Figure 8.6 Public key encryption/decryption using (a) sender's keypair and (b) recipient's keypair.

public key, of the sender's keypair (SKEYpub). The recipient is assured that the message sender is authentic—"she is who she claims to be"—because, as sole owner of her private component of her keypair, only she could have generated the ciphertext. Likewise, as seen in Figure 8.6b, *any* sender can encrypt a message using the public key component of the intended recipient (RKEYpub). However, only the message recipient, the sole owner of the private component of the keypair (RKEYpriv), can decode and read the generated message.

There are not many widespread commercial applications that use public key cryptography as the *sole* cryptographic algorithm—it's usually combined with other cryptographic algorithms. For simplicity's sake, let's construct an application that uses only public key cryptography. Imagine that someone wants an

enhanced electronic mail (e-mail) service such that when you send mail to a business associate and identify yourself, the e-mail system assures the recipient that you are who you claim to be.

Each customer of this service would receive a keypair. The e-mail service provider would have to select and assign keypairs with special mathematical properties, as defined by the public key algorithm; the keypair could not simply be chosen at random. The service provider would maintain a directory service of public components (public keys) of each assigned keypair. If someone did not know your public key, they could call the directory assistance service, which would dispense your public key to anyone who asked. The private component of the keypair would be a number known only to you, similiar to a personal identification number (PIN). You might memorize it, store it on a computer, or store it on a magnetic ATM-style card.

How would this system be used to guarantee your identity? The guarantee is provided by the mathematical relationship between the private component of the keypair (which only you know) and the public key (which anyone can retrieve). Figure 8.7 demonstrates how this process might work. Note that as in Figure 8.6a, the sender's keypair (Jane's) is used for encrypting and decrypting.

1. Jane creates an e-mail message and the e-mail system at her end encrypts her plaintext message using the private component of Jane's keypair (which she and *only* she knows).
2. This ciphertext, along with a plaintext message header, which identifies the message as originating from Jane, is transmitted over the Internet.
3. The e-mail system at Joseph's end receives the ciphertext and the message header that claims that Jane is the author of the ciphertext. Joseph's e-mail system does not have a copy of Jane's public key, so it fetches it from the directory service.
4. The e-mail system at Joseph's end attempts to decrypt the ciphertext of Jane's words using Jane's public key. If the decryption is successful, then it is computationally infeasible—which is to say, effectively impossible—that anyone other than Jane could have created the sent message. If Joseph can read Jane's words, then the authenticity requirement is fulfilled: he knows "Jane is who she claims to be." This is the mathematical "magic" behind public key cryptography.

So why hasn't this simple service been put in place on the Internet? For one thing, while this system does guarantee authenticity, it does not provide a guarantee of privacy. Anyone could capture Jane's message, fetch her public key from the directory service, and decode her message. So this simple system could not be used over an insecure network like the Internet. In practice, the e-mail system must be augmented with other cryptographic algorithms in addition to a public key algorithm to be useful for transmitting messages over the Internet.

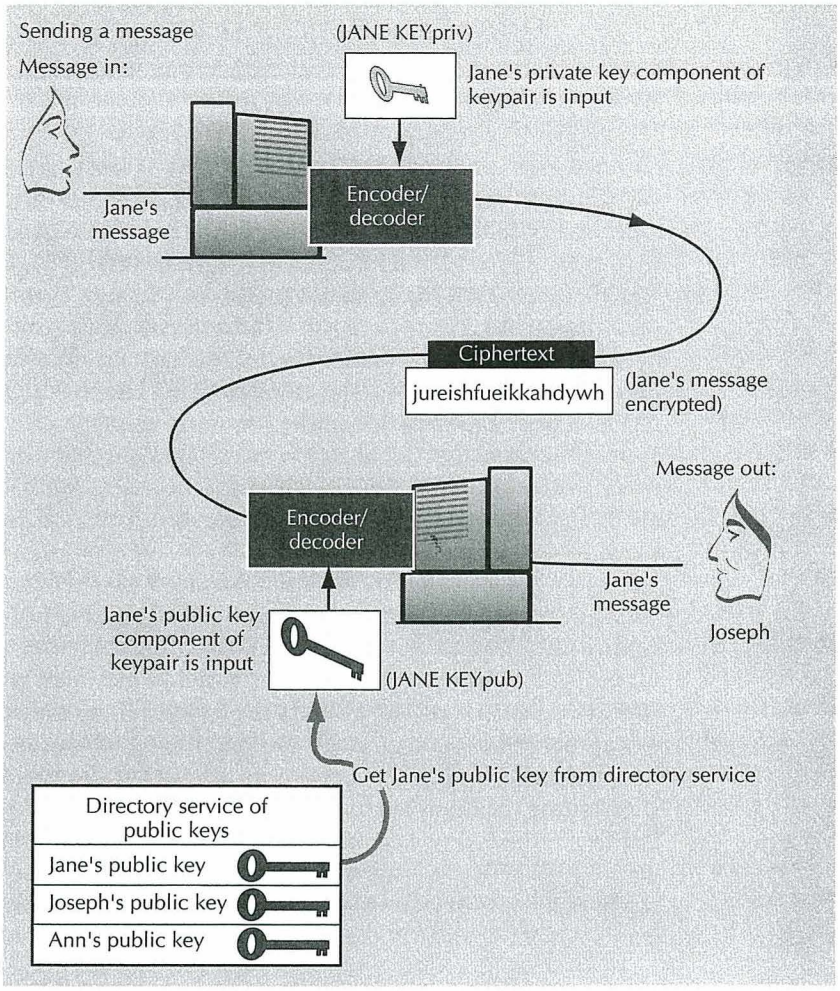

Figure 8.7 A public key-based electronic mail service.

8.6.4 Hashing Algorithms

Hashing algorithms are used to satisfy the message integrity requirement: "Messages received are in fact the messages sent." Unauthorized individuals cannot intercept and alter the transmitted message without the receiver's knowing.

A hashing algorithm is a set of rules that produces a unique number when applied to a message. This number represents the integrity of the message and is called the *message digest.* The mathematical properties of the hashing algorithm ensure that it is computationally infeasible, that is, nearly impossible, to find

two different messages that will produce the same message digest. Any change to a message will, with a very high probability, result in a different message digest. A message, for example, that is altered by one letter will produce a new, totally different, message digest.

Some commonly used hashing algorithms are Message Digest 4 (MD4) (Schneier 1994), Message Digest 5 (MD5) (Rivest 1992), and Secure Hash Algorithm (SHA) (National Institute of Standards 1993b). The size of the message digest produced varies with each algorithm: SHA generates a 160-bit message digest (about 20 characters), while MD4 and MD5 generate 128-bit message digests (about 16 characters). All these hashing algorithms can be used to preserve the integrity of data as it is transmitted over an insecure network. Here's how it works. First, the sender applies the hashing algorithm to his message to produce a message digest. This message digest and the original message are sent together over the network to the intended message recipient. The recipient applies the same hashing algorithm to the transmitted message to recompute a new message digest. If this newly recomputed message digest is the same value as the transmitted message digest, he knows that the message received is indeed the same as the message sent.

Hashing algorithms are seldom used alone. To effectively protect messages from alteration in transit, hashing algorithms must be used in conjunction with private or public key cryptography. Encrypting the message digest prevents an interloper from forging the message digest to match an altered message. Note that in the above scenario it would be possible for an attacker to forge a message by capturing both the message and the message digest in transit, altering the message, recomputing a new message digest for the modified message, and then substituting the modified message and message digest for the originals. To protect against this type of an attack, the message digest is encrypted, most commonly with public key cryptography. Systems that combine public key cryptography and hashing algorithms are so popular that they have earned their own name—*digital signatures.*

8.6.5 Digital Signatures

A digital signature on a document serves as an "official seal" or "signature" and verifies that the message was sent by you, the author, and has been unchanged in transit. Digital signature systems can guarantee authorship to a set of individuals. For example, two or more parties could "sign" a contract if they were using a secure Web service based on digital signatures.

Digital signature mechanisms have been combined with many computer applications, including Web clients and servers. Privacy Enhanced Mail (PEM) and Pretty Good Privacy (PGP) are two examples of digital signature mechanisms widely used for electronic mail (Kent 1993; Garfinkel 1995). A Web service that employs digital signatures would guarantee authorship and ensure message

integrity. Additionally, since digital signatures use public key cryptography, the secure Web service would be scalable to the Internet.

Figure 8.8 shows how a digital signature works. Imagine you are a stockbroker and your company has a secure Web service based on digital signatures. Your customers can buy or sell stocks through your Web service. As you post the going rates for stocks on your Web service, it is very important that the information received by the customers be authentic. The document they receive must be exactly what you intended to publish, not a fake that only looks like the real thing. Unfortunately, it is extremely easy to capture and modify other people's Web pages, so digital signatures may be needed to be sure that what you receive is the real thing. The properties of public key cryptography assure the recipients of the Web document that only you, the owner of the private component of the public keypair (SKEYpriv), can create your digital signature and affix it to the page containing the posted prices. Anyone who wants to browse the posted stock prices can get the public component of your keypair (SKEYpub) from a directory service. He can then verify your signature and thereby know that the stock prices on your Web service were posted by an authorized person. Note how digital signatures use the sender's public keypair for encrypting and decrypting (see Figure 8.6a) to guarantee the authenticity of the author.

In digital signature systems, public key cryptography is used to encrypt only the message digest, not the whole message. Public key cryptography is often too slow to encrypt large amounts of information. If encryption of the message is also required, the digital signature system is usually enhanced with private key cryptography algorithms. This type of document can be thought of as both signed and encrypted.

If the secure Web service used digital signatures, then the Web service would need an external directory service for storing and dispensing public keys. A public key is typically combined with other identifying information (such as name and address) and called a *certificate.* The external directory service that stores and dispenses certificates is called the *certificate authority* (CA).

8.6.6 Certificates, Certificate Authorities, and the Web

Let's return to the Web service for selling stocks. You, as the stockbroker, receive a digitally signed commitment from Ms. Wheaton to purchase a large number of shares of Victor Chemical Company stock. Are you sure this message is authentic? Well, it depends on who gave you Ms. Wheaton's public key. If you know Ms. Wheaton personally, at some point you may have met face-to-face to exchange public keys. This would be adequate assurance of the authenticity of the public key and, therefore, the message. What if you never met Ms. Wheaton and someone sent you a public key, identified as Ms. Wheaton's public key? How could you trust that this is really her key? Couldn't anyone create a keypair and

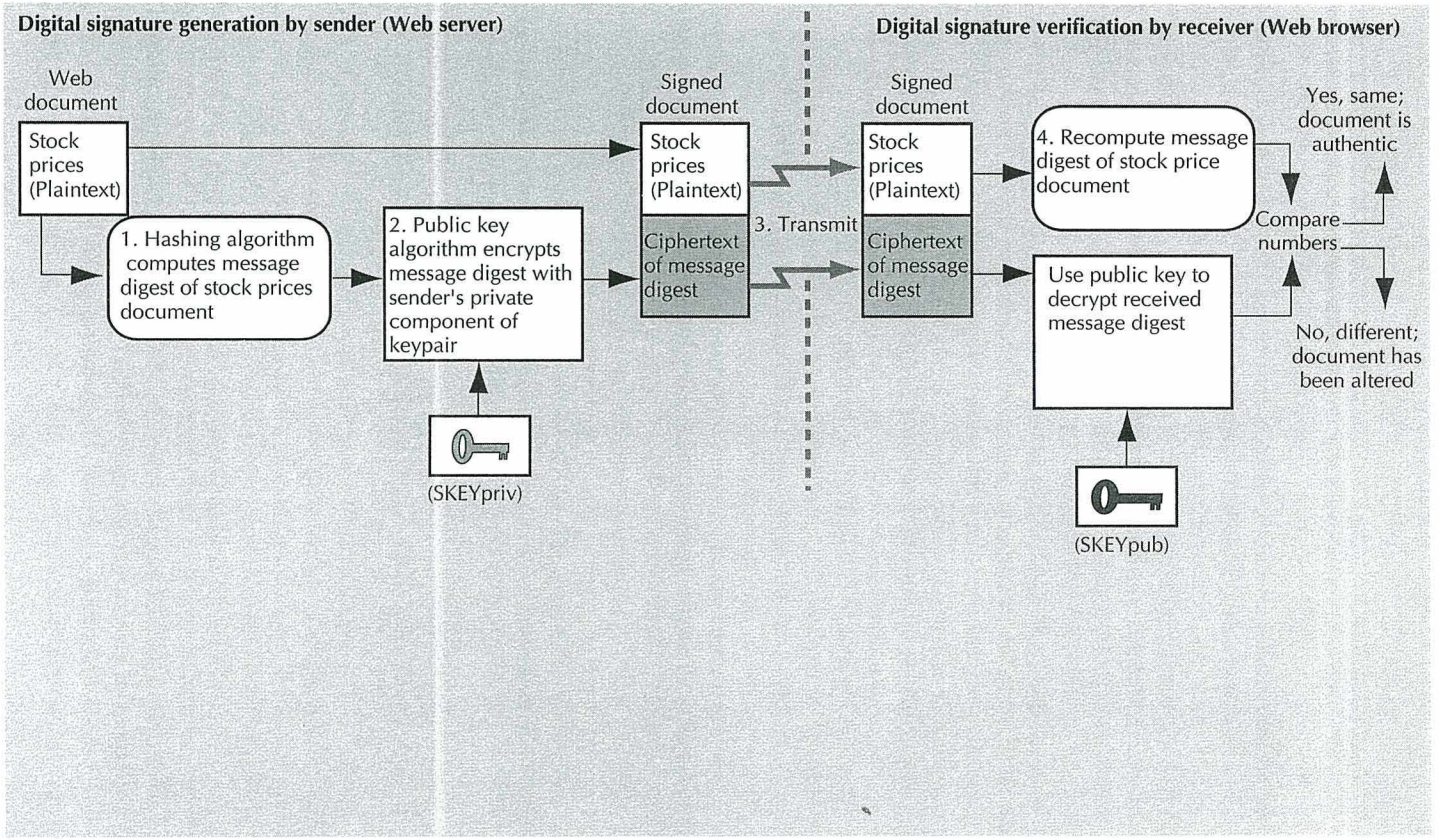

Figure 8.8 How digital signatures work.

send you the public component and tell you it's from Ms. Wheaton? You would believe any subsequent messages, like a large stock purchase, that were signed with "Ms. Wheaton's" digital signature.

What you really need to create spontaneous secure communications between two untrusted parties (Ms. Wheaton and the stockbroker) is a mutually trusted certificate authority (CA) to dispense public keys and vouch for their authenticity. This trusted CA would authenticate Ms. Wheaton once in a reliable fashion, perhaps requiring her birth certificate or fingerprint, and then create and register her certificate. The certificate would accurately bind the true user's identity (Ms. Wheaton) to her public key. The certificate would also contain information such as the name of the CA (which vouches for the validity of the public key), Ms. Wheaton's name, her e-mail address, the date, and the place where the certificate was created. When someone requests a public key for Ms. Wheaton, the CA dispenses the appropriate certificate. That way, when you receive Ms. Wheaton's purchase order, you can trust her certificate (public key) as much as you trust the CA that dispensed it.

There is no central, free CA for the Internet; this is a big problem for secure Web services. Why isn't this infrastructure in place? One reason is that secure key management is not free; in fact, it is costly. Also, if the CA is a trusted authority for digital commercial transactions, it is likely that it will be held liable in the event of a fraudulent transaction. Given that the Internet is shared by many, who would bear the cost of operating and managing this infrastructure? Who will set up the certificate authority for the Internet? Will it be a branch of the U.S. government, the Internet service providers, or large financial institutions? To be sure, there will be a set of Internet CAs, not simply one centralized authority. Just as a driver's license, library card, bank account, and passport are issued by different authorities, it is reasonable to expect that there will be many different CAs, public and private, regional and international, on the Internet. In all likelihood, people will need several types of digital credentials.

Proliferation of CAs is inevitable. Already the creation of a set of Internet CAs is under way: the U.S. Postal Service, VeriSign, Inc., MasterCard, and Visa have all announced certificate authority services for the Internet (VeriSign 1995a). CAs must be trusted entities, so selecting a CA is a sensitive issue. Some companies or individuals will trust only their own CA, not anyone else's.

We have now seen that private key cryptography fulfills the confidentiality requirement, public key cryptography fulfills the authenticity requirement, and hashing algorithms fulfill the message integrity requirement. Real systems need to meet some or all of these requirements, so these technologies are often combined to create a security framework for a given system. The next section examines some of the issues that arise when these technologies are integrated into a distributed network service like the Web.

8.7 Integrating Cryptographic Algorithms with the Web

8.7.1 Selecting a Cryptographic Algorithm

The most fundamental issues for integrating cryptographic algorithms with the Web are selecting the best cryptographic algorithm and correctly managing the keys. Selection of a cryptographic algorithm affects the interoperability and availability of the security-enhanced software. There are lots of algorithms to choose from: RSA, DSA, DES, IDEA, RC2, RC4, MD4, MD5, SHA (Garfinkel 1995; Bidzos 1991; Schneier 1994; FIPS 1994; National Institute of Standards 1993a, 1993b; Rivest 1992). For a Web client and a Web server to communicate securely, they must be enhanced with the same cryptographic algorithm or, more commonly, with the same suite of cryptographic algorithms. In practice, a secure Web service would most likely combine several cryptographic algorithms to take advantage of the strengths of each method.

Cryptographic software has a few unique features that distinguish it from just any other computer program. First of all, the distribution of cryptographic algorithms is regulated by the governments of many countries because these programs are considered munitions, weapons of war. One can certainly understand this, given the powerful role cryptography has historically played in wartime activities (Kahn 1967). Laws that regulate the distribution of cryptographic algorithms vary from country to country. In the U.S., the State Department regulates the export of certain types of cryptographic technology. The general idea is that the U.S. government does not want to export to other countries its strongest, most uncrackable cryptographic algorithms. These laws have an impact on all cryptographically enhanced software because they affect the availability and interoperability of these programs internationally.

These two properties of encryption—legal regulation and the requirement that only like sets of algorithms be interoperable—push crypto-enchanced versions of Web software toward a proliferation of interoperable versions of "secure" Web clients and Web servers. This is especially true for organizations or individuals that need secure communications on an international scale. Imagine, for example, that Victor Chemical Company has purchased a DES-enhanced version of secure Web software. It can easily use this version between its U.S.-based divisions. But DES may not be exported for use outside the U.S., so the VCC division offices in France may have difficulty finding the correct software to decode a DES-encrypted message from the U.S. office. Needless to say, this characteristic of secure World Wide Web software makes them seem much less "worldwide."

As we have seen, cryptographic systems fundamentally depend on keys, which are used to identify people and to restrict access to information. A practical cryptographic system must have the means to manage the necessary keys. Keys

must be generated, stored, distributed across an insecure network, used by the application, and ultimately destroyed. All the tasks may be done in a variety of ways—and some are more effective than others. This section examines the way keys would be handled for each of these cases in a secure Web service.

8.7.2 Key Generation and Destruction

The size of the cryptographic key is one of the primary gauges of an algorithm's crackability. Keys in cryptosystems must remain large, or thieves may be able to guess them. Thieves may systematically guess at keys, over and over again, using a computer program in a way very similar to the the way passwords are cracked (see Chapter 7). In an effort to design an easy-to-use cryptographic system some people may forget that keys must remain large and unguessable. For example, some banks, in an effort to make keys (PINs) easy to remember, have simplified the PIN until it was possible to guess someone's PIN in a relatively small number of attempts (Anderson 1994).

Technically speaking, all keys, of any length, are guessable given enough time and computation power to keep guessing and guessing. The larger the key size (keyspace), the greater the number of possible keys there are, though.

Keys 40 bits or less in length provide questionable security. For example, a version of Netscape's security-enhanced Web service that uses 40-bit keys, Secure Socket Layer (see Sections 8.11 through 8.13), was cracked in eight days with approximately one hundred server-class computers and one supercomputer (Trei 1995). The 128-bit version of SSL (which is currently available only within the U.S. due to export restrictions) would be much more difficult to crack through this "brute force" method.

Public keys 512 bits in length are generally thought to be "safe," or unguessable within a reasonable amount of time (Schneier 1995). With current factoring techniques, it would take one thousand 100-MHz Pentium computers a full year to crack a 512-bit public key.

Another way to reduce the possibility of someone's guessing a user's keys is to change the keys frequently. Historically, private keys were changed frequently to reduce the threat of an enemy's capturing the key and using it over and over again. During World War II, codebooks containing an army's current cryptographic keys were distributed to all units exchanging coded communications. Codebooks became less and less secure with use. So keys in the codebooks were changed frequently to avoid capture by the enemy. Computer systems employ this technique also. A new private key, *a session key*, may be produced for each secure communication. The use of session keys decreases the time an "enemy" has to decode a given key and limits the damage if a key is captured. Many secure Web services use session keys for this reason.

One implementation of SSL made a well-publicized error in session key generation, which made it too easy to guess the session key (Goldberg and Wagner 1995). Many cryptographic algorithms rely on a random number as input to the

key generation process. The SSL client-side software chose a "random" number, based on easily determined system parameters. Once the method for selecting a random number was published, users could apply the logic to obtain the session key in less than a minute. Given the session key and the encryption algorithm, users could potentially decode SSL-encrypted transactions.

In both private and public key systems, keys must be destroyed when they are no longer needed or when they are compromised. Session keys are automatically destroyed when the login session terminates. Destroying private keys is usually a simple matter: when the user is no longer part of the system, his private key is removed. However, destroying public keys can be problematic. Deleting the key from the directory service may not be sufficient, because users and computer applications may copy public keys into a local cache to save the time of fetching them with each use. When the key is invalidated, the cached copy becomes "stale," and the next time the user tries to use it the transaction will fail because they are not using the correct current key. If public keys are cached for convenience, they are subject to the same cache coherence problem as cached documents, as described in Chapter 5. In practice, changing public keys is, at best, inconvenient for both the sender and the recipient of encrypted communications.

With any key management system, deleted or expired keys may have to be archived indefinitely. The reason that old keys cannot simply be forgotten is that any data that was encrypted with the old key cannot be decrypted and read without using the old key. For example, if documents are archived to tape in encrypted form, they can only be restored using the correct key for the time they were encrypted. Similarly, a document containing an encrypted message digest or digital signature (see Section 8.6.5) can only be verified using the public key that was correct at the time the document was created and signed.

8.7.3 Key Storage and Use by the Application

Storing keys safely is an important practical problem for any cryptographically enhanced network service. When keys are stored on a computer, they are vulnerable to being copied or forged unless carefully protected. The greatest weakness of any cryptographically enhanced Web service is likely to be insecure storage of keys.

Secure storage of keys is critical to both Web servers and clients. A secure Web server system can be dedicated and specially configured to its mission. Usually only one authorized person, the system administrator, has access to the Web server. It should be possible, therefore, to protect cryptographic keys on a properly configured Web server.

But client systems are generally multipurpose workstations or personal computers and cannot be specially configured just to protect keys. Many users may have access to a Web client system. If a key is stored in a file, or even in memory

on the client system, there are ways for malicious users to access this private key if they can gain superuser privilege. Access to the key is even easier on personal computers, which do not have a login or privileged-access mechanism. With millions of potential Web clients, keys stored on the client machine are especially vulnerable. The issue of secure key storage on the client is a common problem that few cryptosystems attempt to solve.

Secure Socket Layer and Secure-HTTP, two popular cryptographically enchanced Web implementations (see Sections 8.10 through 8.12), both currently protect keys of the client by encrypting them with the user's password. In this case, the key is only as secure as the password, and therefore it is important to select a password that is not easily guessable. The best method for protecting keys is to put them in a microprocessor on a credit-card-sized *hardware token,* such as a *smartcard* (Sherman, Skibo, and Murray 1994). Hardware tokens are almost impossible to break into, and they provide the safest method for client-side key storage. However, using hardware tokens is somewhat expensive and inconvenient because every user must have a card containing the tokens.

8.7.4 Key Distribution across an Insecure Public Network

In practice, the successful use of a cryptographic algorithm in network services that operate over insecure networks depends on the safekeeping of the key. This is especially true for private key cryptography, where the key must remain private yet is shared by the sender and the receiver. Anyone who captures the encrypted message, knows the cryptographic algorithm, and has the private key can decode the message. Transmission and distribution of the shared key is crucial to maintaining the security of the system. Private keys must be either safely stored on the client or safely transmitted from a secure server over the network to the client.

One common threat to a secure Web service is the disclosure of keys to unauthorized personnel during the key distribution process. This situation is analagous to distributing keys (PINs) for automatic teller machines. In the ATM example, the PIN (key) and account cards are typically delivered to the bank patrons through the mail. This gives the postal workers, or anyone with access to the mail, an opportunity to intercept the cards and to read the keys (Anderson 1994).

It is inherently difficult to securely distribute private keys across an insecure public computer network. As we saw in Chapter 7, there is a danger that someone could be eavesdropping on the conversation and could steal the key as it is being transmitted. The basic solution to this problem is to devise a scheme whereby the key is not in a useful form if intercepted. One common technique used by secure distributed network services, like Secure RPC (Garfinkel and Spafford 1991) and Kerberos (Steiner, Neuman, and Schiller 1988), is to encrypt the key with the user's password before they are distributed across the network.

The user's password is not stored on the client workstation; it is stored, in some form, on the secure server.

Secure RPC stores the client's private component of the public keypair, encrypted with the user's password on the Secure RPC server. As the user logs in on the remote computer, the Secure RPC server sends the client system the encrypted private key over the network. If the user enters the correct password, then the client application can decode and use the private key for secure communications with the Secure RPC server.

Web clients and servers enhanced to use Kerberos also use this technique to distribute session keys for secure communications. With Kerberos, clients and servers can obtain session keys contained in Kerberos "tickets," distributed by a trusted Kerberos server. The tickets can be read only by the intended recipient (Web client or Web server) because they are encrypted with the private component of the recipient's keypair, which is known only to that recipient and to the Kerberos server (Cain 1995b; Shaffer and Simon 1994).

Any scheme that uses password authentication requires that all users be registered with the Web server or authentication server. That is, the user must establish his personal key, perhaps an encrypted password, before he can use the secure Web server. This can be a disadvantage for large-scale systems with a rapidly changing customer base: you really don't want to have to tell new customers to please come back next week because their password isn't ready yet! This scheme obviously does not satisfy the desirable option for "spontaneous" secure transactions with absolutely anyone.

Also, if your private keys are located on a remote server or set of servers, then all your security services must rely on management activities that are outside of your administrative control. If the host that maintains the keys is compromised, then all messages are exposed—through no fault of the Web server or users. This is not a comfortable position to be in. A second problem is that the secure key server(s) may become overloaded as the system grows. With millions of people using the Web, a single server cannot handle all the authentication requests any more than a single server could serve all the documents. These problems would limit secure Web servers based on private key cryptography from scaling to huge Internet-wide systems.

Public key cryptography solves some of these key distribution problems in that only the public key component is transmitted over the insecure network. This transmission occurs at no risk, since public keys are generally available anyway. The user's private key component is stored locally and, by design, there is no need to transmit this key over the insecure network. The user must still keep the private component of the keypair secret, so safe client-side storage is still an issue.

Public key cryptography is also superior in scaling to large systems like the Internet. Secure Web services based on public key schemes could grow well beyond a local area network to Internet-sized schemes. Public key components

could be stored in distributed databases and the public keys would not have to be guarded as in private key cryptography systems.

In practice, few secure Web services use only private key cryptography because of the problems with key distribution. A secure Web service is most likely to combine cryptographic algorithms to take advantage of the strengths of both methods. Let's look at one of the most common key distribution methods that do this.

8.7.5 A Common Key Distribution Scheme

One popular key distribution method uses public key cryptography to solve private key cryptography's key distribution problem. This combination of methods could be used to enhance a Web service with the ability to make spontaneous secure transactions. The public key cryptography authenticates the transaction and the private key cryptography speeds up the encryption of the message. Let's see how this works.

A customer, Jane, and a seller, Joseph, who have never interacted or conducted business before could set up a spontaneous secure communication using this method (Cain 1995c). Figure 8.9 illustrates this scheme:

1. Jane randomly picks a session key (for use in private key cryptography).
2. Jane retrieves Joseph's public key from a public key directory service.
3. Jane encrypts the session key using Joseph's public key.
4. Joseph uses his private component of his keypair to decrypt Jane's message and obtain the session key.
5. Jane and Joseph communicate securely by encrypting their conversation (private key cryptography) with the session key.

Note how this scheme uses the message recipient's keypair for encryption and decryption (see Figure 8.6b).

In this method, public key cryptography is used to authenticate Joseph and securely distribute the session (private) key. Once both Jane and Joseph have a shared session key, they can use that key to encrypt their entire conversation (the Web document). Private key cryptography algorithms (DES, for example) are very efficient at this "bulk" data encryption. This scheme takes advantage of public key cryptography's scalability and the efficiency of private key cryptography for data encryption. This method is used in existing secure distributed network services such as Secure RPC (Garfinkel and Spafford 1991) and secure Web implementations such as SSL and S-HTTP (Rescorla and Schiffman 1995; Hickman and Elgamal 1995). (See also Sections 8.9 through 8.12, below.)

This section has reviewed the basic tools and technologies to be used for enhancing the Web with security services. Before we examine the security-enhanced Web services available today, it is useful to consider operational hazards of deploying a digital commerce Web service.

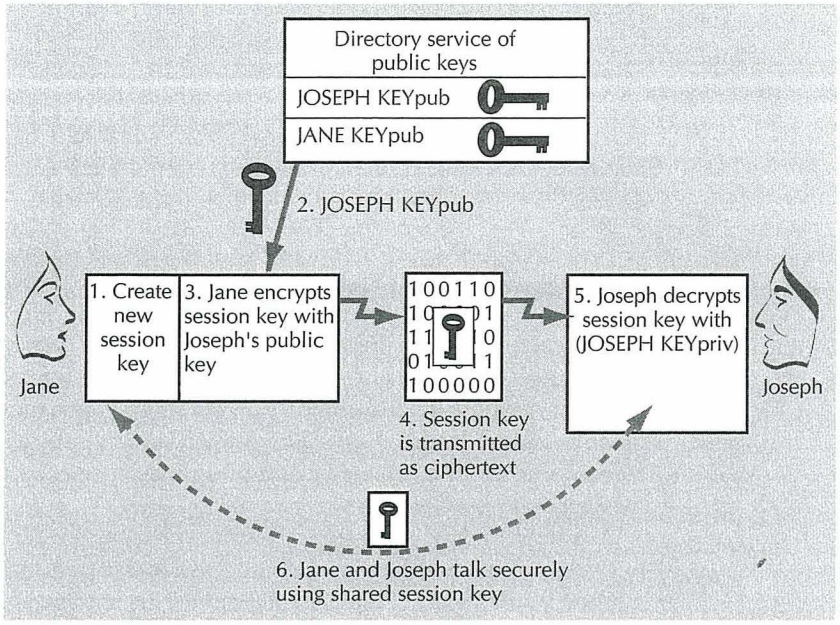

Figure 8.9 Setting up a spontaneous secure transaction.

8.8 Operational Considerations for Digital Commerce Systems

8.8.1 Low-Tech Threats

To determine the potential failures of a commercial Web service, it is useful to review the well-documented failure history of automatic teller machines (ATM systems). First deployed in the 1970s, ATM services were the first commercial application to use cryptography. Up to that time, cryptography had been used only for military applications. The major threat in a military environment is advanced cryptanalysis by the enemy. Cryptanalysts try to crack the cryptographic algorithm by looking for patterns in the ciphertext or by generating a ciphertext from a known message and then capturing and analyzing the ciphertext. Can we assume this same threat model for commercial services? Are thieves monitoring the network lines of ATM services and analyzing the ciphertext that represents banking transactions?

A survey of hundreds of documented ATM failures revealed that most fraudulent withdrawals were caused by flaws in the design and operation of the ATM

systems, not by skilled cryptanalyst attack (Anderson 1994). In fact, only two instances could be attributed to skilled attack. The three most common causes of fraudulent withdrawals in the ATM systems were

1. System software bugs
2. Stealing of PIN numbers from the mail
3. Thefts by bank staff

These failures are members of a larger group of failures that have nothing at all to do with the quality of a cryptographic algorithm. Let's consider some of the operational problems that can affect a secure Web service.

Lack of Education

User and system administrator education are of paramount importance in operating a secure Web service. Users must understand the importance of managing their keys and the significance of selecting a good password. System administrators must know how to configure a secure system correctly and understand the implications of disclosing critical key information. While these requirements may be intuitive, it is surprising how often ignorance is the source of failure in a secure system.

Corrupt Staff

The more people responsible for the management of keys and the maintenance of the cryptosystem, the higher the probability of failure. Banks have often discovered that their own staffs were responsible for many fraudulent withdrawals. Personnel with access to privileged key management operations were likely candidates for corruption. Implementing ATM cryptosystems in software, rather than in hardware, exposes the cryptosystem to theft by software developers and system administrators. For example, one ATM system provided a software capability to test the functioning of the unit. A test key was printed in a manual and available to the programming staff; it was intended for testing of the ATM units offline. It was later revealed that corrupt programmers were using the test key for years to steal from ATM machines in the field (Anderson 1994). So it would be wise to take realistic steps to monitor the trustworthiness of the people who are placed in positions of trust.

System Software Bugs

ATM and Web services are distributed network applications. In large, complex, heterogeneous ATM transaction processing systems, it is well acknowledged that errors, such as posting transactions to the wrong account, occur at least 0.01 percent of the time. This is a 1 in 10,000 chance of error. This may not sound so bad, but there may well be millions of transactions per day, which means there are hundreds of errors every day. Commercial transactions using the Web will, no doubt, have a similar error rate.

8.8.2 Denial of Service

If an unauthorized person can clog up, confuse, or knock out a system without even having a key, he can interfere with legitimate business. This is called a *denial-of-service attack*. A denial-of-service attack on a Web service could happen when someone, possibly a competitor, floods your Web service with so many requests that your Web server cannot answer other legitimate requests in a reasonable time, if at all. Legitimate customers wait too long for their requests to be satisfied, become frustrated, and give up. Failures due to denial are possible in all computer network-based services, such as the Web; however, they are often hard to distinguish from nonmalicious accidents, crashes, and just plain overload.

The HTTP protocol is particularly susceptible to denial-of-service attacks because it is so open—an HTTP server answers the door, without looking, to anyone that knocks. It is very easy for a malicious individual to flood a Web server with thousands of pointless requests. The concept of a user session would help the denial-of-service problem, although this would involve changing the HTTP protocol. At the present time, all a system administrator could do to prevent denial of service is to monitor the server, log who is bombing their system with invalid requests, and then ask them to stop it.

8.8.3 False Service Provider

Imagine setting up a false ATM in a shopping mall to collect account numbers and PINs from unsuspecting customers. The collected account numbers and PINs (customers' keys) could then be used to withdraw money from the customers' real accounts. Sound too unbelievable to be true? Well, false ATM machines have been set up (Anderson 1994). It takes at least a little skill to fake an ATM, but it takes even less skill to set up a false Web service to collect unsuspecting customers' credit card numbers. This type of attack is often called the "man in the middle" when it involves network-based services. The attacker intervenes between the client and the server. In this attack, the "man in the middle" pretends to be the real Web service provider and can thereby capture all confidential transaction information (Hickman and Elgamal 1995). This attack can be prevented if the Web browser requires the Web server to authenticate, using digital signatures, for example. This is exactly how the Web digital commerce implementations examined in the next section protect Web customers from the "man in the middle."

We have seen that the common threats in existing digital commerce are likely to be low tech. In security systems, one must examine the entire process to uncover the weakest link. This weakest link defines the security level for the entire system. Experience suggests that anyone providing a commercial Web service should address the listed administrative threats before they insist on

perfection from their cryptographic algorithm. An "unbreakable" cryptography service is useless if your keys aren't chosen, distributed, and managed correctly.

Now that we have examined operational considerations, let's examine two specifications of cryptographically enhanced Web servers. How well do such systems work, and how will they break down? And what set of criteria would we use to assess the capabilities of a secure Web service? These are important questions to answer before entrusting your assets to the Web.

8.9 Selecting a Secure Web Service

Given that you have addressed the operational concerns mentioned above, how do you select the best secure HTTP service?

There are currently two leading specifications for crypto-enhanced Web services: Secure HTTP (S-HTTP) (Wong 1995a, 1995b) and Secure Sockets Layer (SSL) (Hickman and Elgamal 1995). S-HTTP was developed for CommerceNet, a digital commerce testbed in Palo Alto, California (CommerceNet 1995a). SSL is a crypto-enhanced version of TCP/IP, a specification developed by Netscape Communications Corporation. Microsoft's Private Communication Technology (PCT) protocol is almost identical to the SSL protocol, with a few minor exceptions (Benaloh et al. 1995). SSL can be used with a variety of network applications, but it is most commonly used to enhance the security of HTTP. Both SSL and S-HTTP augment Web browsers and servers with cryptographic algorithms for authentication and privacy. The implementations of these protocol specifications are emerging technologies, as the specifications themselves are still evolving (Rescorla and Schiffman 1995; Hickman and Elgamal 1995). Both schemes define a new protocol for accessing secure HTML documents. According to the SSL scheme, a URL to be fetched using SSL, such as `example.html`, would be identified as

```
https://www.any.com/example.html
```

The same document would be identified as

```
shttp://www.any.com/example.html
```

in the S-HTTP scheme. The differences between the two expressions are the way the document is protected and the type of browser one would need to successfully select the hyperlink.

Using the `shttp:` or `https:` identifier ensures that a secure communication is used for transferring data between the client and the server. A browser that is not able to follow the proper protocol will not understand the URL containing `shttp:` or `https:` and, therefore, will not attempt to follow the link, so no message will be sent at all. This protects the browser from sending sensitive

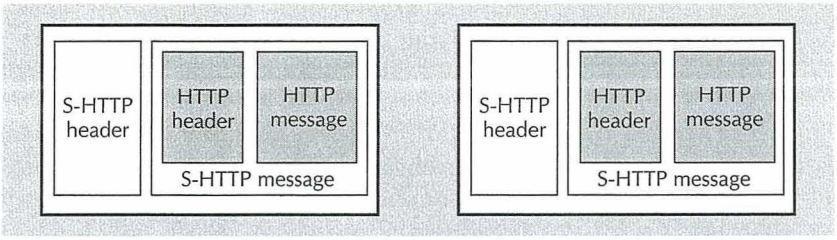

Figure 8.10 S-HTTP encapsulates HTTP messages.

information in cleartext over the Internet, perhaps a credit card number embedded in a form. The sensitive information will be sent only by a browser that uses the correct protocol to protect it.

SSL and S-HTTP are two new protocols. Just as HTTP headers accompany each HTTP document in the communication stream, S-HTTP headers encapsulate a message that is composed of the HTTP headers and HTTP documents. A typical communication stream illustrating the S-HTTP headers and message encapsulation is shown in Figure 8.10.

Both S-HTTP and SSL initiate a secure communication by negotiating the cryptographic algorithms to use in the communication. This means that the Web browser and Web server find a set of mutually supported cryptographic options, or *cryptopts,* to use in the communication. If the browser and the server cannot find a common crypto-language, then the negotiation fails and no HTTP messages are sent.

Figure 8.9 demonstrated that when public key cryptography is combined with the use of a session key, spontaneous secure communications are possible. S-HTTP and SSL use this mechanism. For the most common case, only the secure Web server needs to prove its identity to Web clients. Proof of the server's identity is needed to protect the Web users from the "man in the middle" attack (Section 8.7.3). Therefore, only the Web server is required to own a keypair. The private component of this keypair is known only to the secure Web server, while the public component is made available to all clients that request it.

Figure 8.11 shows how spontaneous secure communications and encapsulation work for the S-HTTP secure Web service. In Step 1, the client initiates the S-HTTP secure transaction by generating a session key. The HTTP message and headers are encrypted with the session key to form the S-HTTP message body. The client then acquires the server's public component of the keypair, uses that key to encrypt the newly generated session key, and inserts the ciphertext of the key in the S-HTTP header. Note how the regular HTTP headers (described in Chapter 2) and the HTTP message body are encapsulated within the new S-HTTP headers. The S-HTTP message and the S-HTTP headers are sent to the secure server.

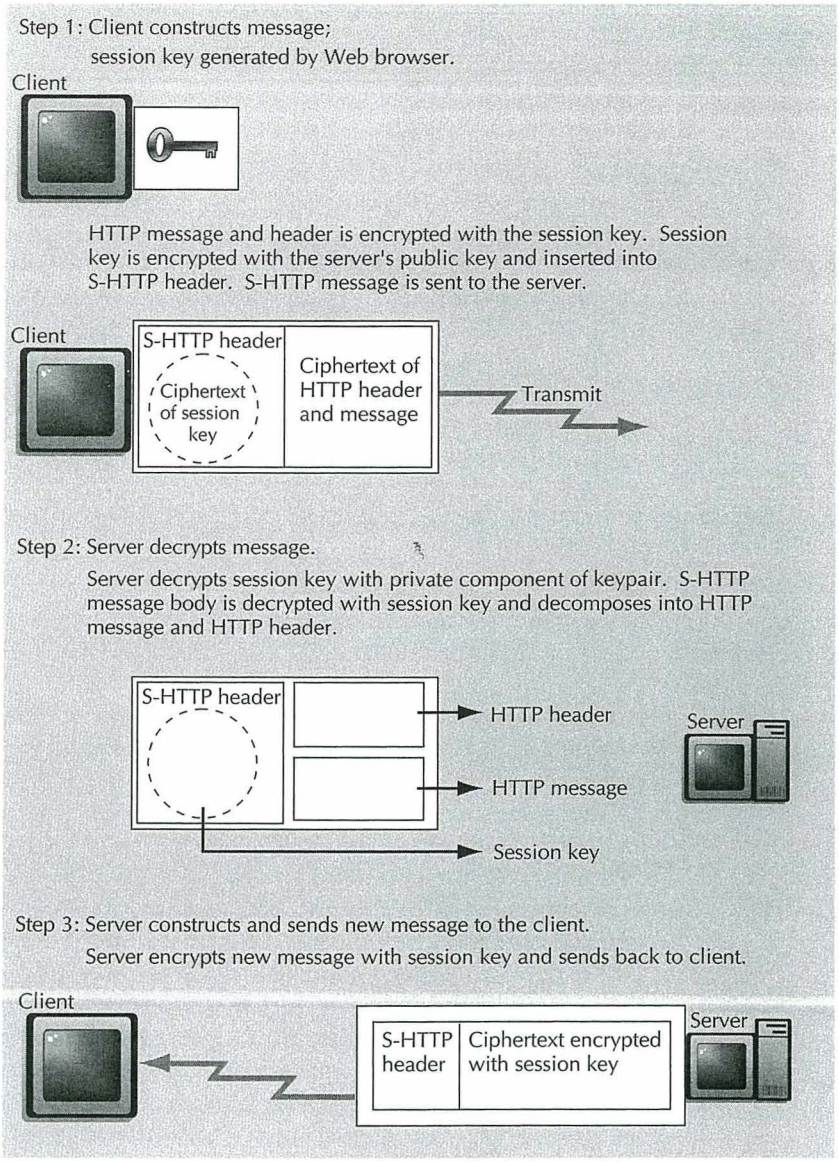

Figure 8.11 Spontaneous secure Web communications.

In Step 2, the server receives the S-HTTP headers and the encrypted message and uses the private component of the keypair, known only to the server, to decrypt the ciphertext in the S-HTTP header and derive the session key. This session key is then used to decrypt the S-HTTP message body. The message body has two parts: the HTTP header and the HTTP message.

In Step 3, the server uses the decoded session key, sent by the client, to encrypt a new message to be returned to the client.

It is important to note that the reverse scenario, a spontaneous communication initiated by the server, does not work. One might speculate that the server could initiate the communication by (1) generating the session key, (2) using the private component of its keypair to encrypt the session key, and (3) sending the message back to the client. What's wrong with this picture? The problem is that any client with access to the server's public key can decrypt the "secure" communication. Since all clients have access to every server's public key, anyone can decrypt the message! (The message is not confidential, as shown in Section 8.6.3, Figure 8.6a.)

The above example of secure S-HTTP communication is a simplified version. In practice, the public keys are stored in a certificate. The secure Web server registers with a trusted authority, the certificate authority, which assigns a valid certificate and keypair to the server. The CA then vouches for that certificate's authenticity to all Web clients who use the certificate.

The client uses the certificate of the server to get the server's public key and then to initiate a spontaneous secure communication. In theory, the Web browser would get the certificate over the Internet from a CA. As we have already noted, the infrastructure for Internet CAs is incomplete. Therefore, it is no surprise that both S-HTTP and SSL compensate for the lack of a widely available, low-cost Internet certificate authority. In both schemes, the Web user (browser) retrieves the server's certificate from the secure server itself. SSL stores the certificate in a file on the secure server. S-HTTP embeds the server's certificate in each secure HTML document, as explained in Section 8.10.

The secure storage of keys is a pervasive problem in all secure systems, including these two implementations. The private component of the keypair is stored encrypted with the administrator's (or user's) password in a file or database on the Web server (or Web client). To use the key, the user or Web administrator types in their password, the key is decrypted, and the key is stored in memory for the duration of the session. Some Web browsers and servers can optionally store keys in special hardware, such as a smartcard (hardware token), if the customer is willing to pay for the added expense (Hostetler 1995; V-ONE 1995).

In both schemes, the client may optionally authenticate to the server. If the client authenticates itself, the Web service is better equipped to send confidential information to the correct client. This option is useful when the server needs to be careful about the identities of clients who receive secure documents. But client authentication is not ideally suited for digital commerce where spontaneity is desired. Authenticating the client would require each client to obtain, securely store, and register a public keypair with a certificate authority before conducting the transaction. This could be inconvenient for the client and takes the spontaneity out of the digital transaction.

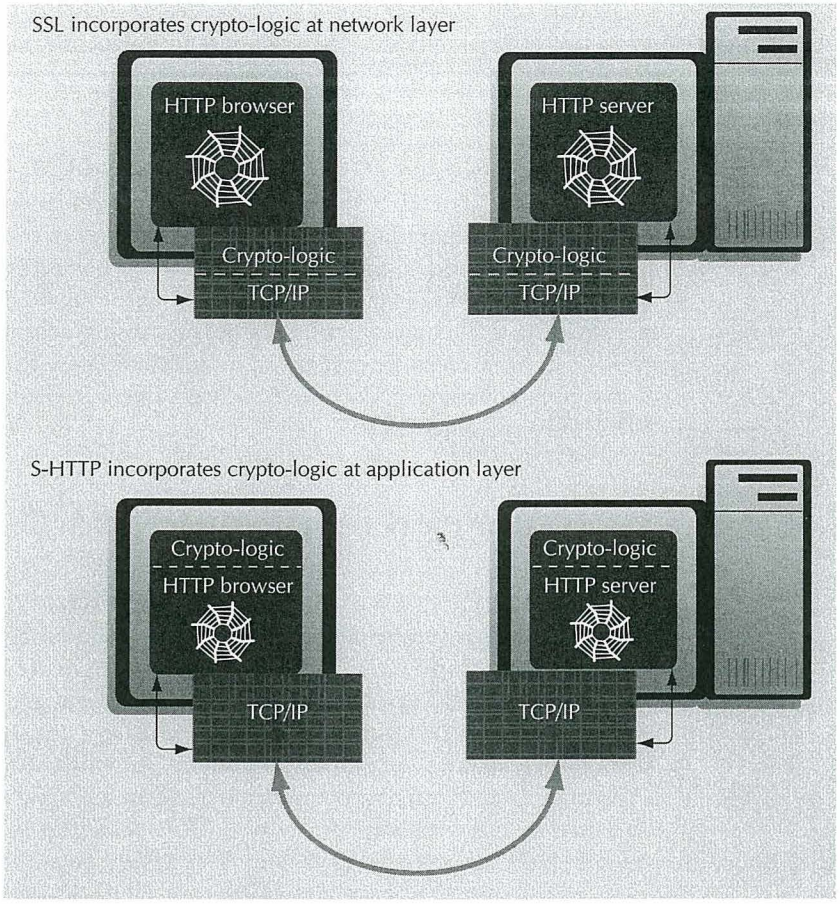

Figure 8.12 Where SSL and S-HTTP integrate cryptographic algorithms.

The major difference between SSL and S-HTTP is where the cryptographic logic resides. As Figure 8.12 illustrates, SSL incorporates the cryptographic algorithms at the network layer of the system software. The HTTP protocol can remain unchanged when it runs on top of SSL. The security of the application is gained when the application opens a Secure Socket Layer (TCP/IP) connection. Other existing applications, such as `telnet` and `ftp`, can run on top of SSL to make them secure also. The SSL Web server is a totally separate secure server, listening on its own port, 443. As will be shown in Figure 8.18, if a Web service provides both open and protected documents, it must run two Web servers: the regular Web server (port 80) and the SSL-enhanced Web server (port 443).

In contrast, a single S-HTTP-enhanced server can serve both open and protected Web documents. The cryptographic algorithms in S-HTTP are tightly integrated into the HTTP protocol in the browser and the server. In fact, these security enhancements extend the HTTP protocol so far as to constitute an entirely new protocol, the Secure HTTP, or S-HTTP, protocol. S-HTTP may be considered a new protocol because so many new headers are added to the HTTP protocol to specify such parameters as the encryption algorithm used, the public key certificate format, the data encoding format, and so on.

The S-HTTP and SSL schemes do not specify how the certificates and cryptopts are stored and managed by the client and the server. The protocols also do not define how the cryptographic features are displayed to the user or what control the user has over the cryptographic service. These vary from implementation to implementation. Let's see how two current implementations of SSL and S-HTTP handle these issues.

8.10 Secure HTTP (S-HTTP)

One of the most distinctive aspects of the CommerceNet S-HTTP implementation is that the cryptographic features are very visible and controllable. A document's security status, certificate, and cryptopts are represented by icons in the CommerceNet S-HTTP-enhanced Mosaic browser and are displayed to the user. The icons (shown in Figure 8.13) indicate what sort of cryptographic activity is taking place: signing, encrypting, and so on. The icons also indicate the status of secure documents: encrypted and signed documents, signed-only documents, and plaintext documents, for example. The icons help the user to be aware of the secure operations in progress and whether and how a document is protected.

The CommerceNet browser also gives the author control over the security features used to protect the document. Cryptopts permit the creator of the S-HTTP document to specify fully how the Web browser must behave, which cryptographic algorithms they must have, and which keys they must exchange in formulating a secure request.

To illustrate how S-HTTP manages certificates, cryptopts, and the secure communication setup, let's return to our original example of submitting a form containing a purchase order to the Victor Chemical Company server. In this example, Victor has enhanced its Web server with S-HTTP so that clients with S-HTTP-aware browsers can securely submit purchase orders. Before a secure S-HTTP service can be offered, HTML documents must be enhanced with embedded cryptopts.

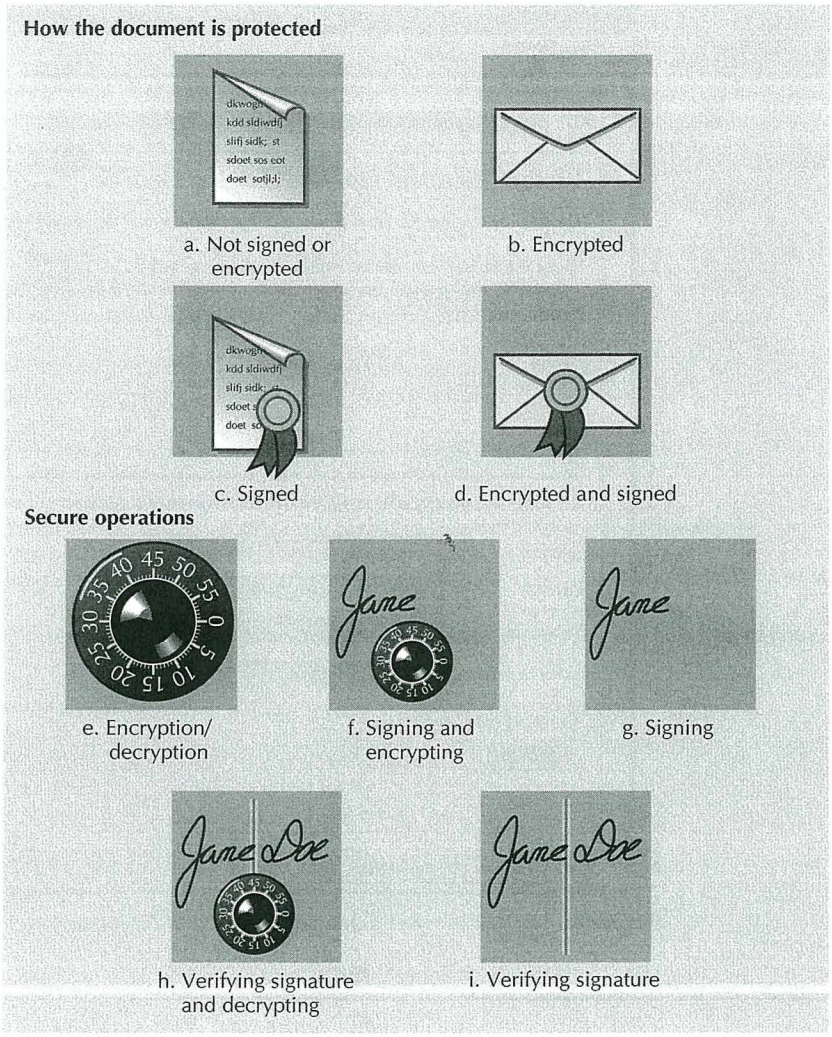

Figure 8.13 S-HTTP cryptographic icons.

8.10.1 Creating the Secure Document

The VCC document designer must first create an HTML form for customers to enter their orders. When displayed by a Web browser, the form would look something like Figure 8.14. The customer fills out the form and sends it, securely, to a

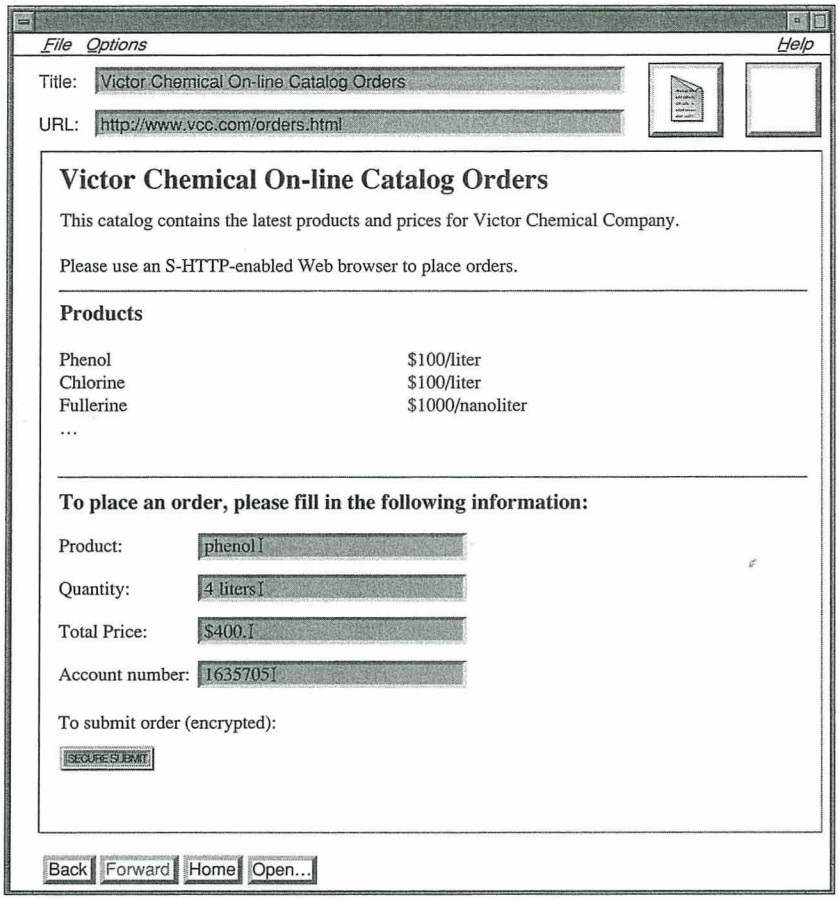

Figure 8.14 Secure browser displaying the VCC order form.

CGI script on the S-HTTP-enhanced Web server. The S-HTTP security-enhanced CGI script accepts the purchase order and sends the customer a purchase receipt.

The purchase order form in Figure 8.14, `http://www.vcc.com/orders.html`, must contain a hypertext link to the security-enhanced form, `shttp://www.vcc. com/scripts/take_orders.html`, and everything an S-HTTP-enhanced browser would need to initiate the secure communication (such as the certificate and the names of the cryptographic algorithms to be used in the secure transaction). S-HTTP defines new extensions to HTML to permit embedding the public key certificate and the cryptopts in the document. Figure 8.15 shows the HTML form, to illustrate how this cryptographic information is embedded in the HTML form.

```
<header>
<CERTS FMT=PKCS7>MIAGCSqGSIb3DQEHAqCAMIACAQExADCABgkghkiG9w0BBw
KZajPLrttFTY4oPbrUmHK7o72O9COqpTOIZ9i3LsskOMMJmVlQkwggHHMIIBUQI
DQYJKoZIhvcNAQECBQADYQA/+VKO8Q3aePaGZvRqDXD4WXFtXvp6Iy7x11gFPwg
QSBEYXRhIFN1Y3VyaXR5LCBJbmMuMS4wLAYDVQQLEyVMb3cgQXNzdXJhbmN1IEN
cnRpZmljYXRpb24gQXV0aG9yaXR5MB4XDTk0MDEwNzAwMDAwMFoXDTk2MDEwNzI
LCBJbmMuMRwwGgYDVQQLExNQZXJzb25hIENlcnRpZmljYXRlMGkwDWYJKoZIhvc
</CERTS>
</header>

<body>
<TITLE>Victor Chemical On-line Catalog Orders</TITLE>
<H1>Victor Chemical On-line Catalog Orders</H1>
<P>
This catalog contains the latest products and prices for Victor
Chemical Company. Use this on-line form to securely place orders.
<HR>
<form action="shttp://www.vcc.com/scripts/take_orders.html"
DN="CN=Victor Chemical Co.,OU=Persona Certificate,
O=&quot;RSA Data Security, Inc.&quot;,C=US"
CRYPTOPTS="SHTTP-Privacy-Domains: orig-required=PKCS-7;recv-required=PKCS-7;
SHTTP-Key-Exchange-Algorithms; orig-required=RSA;recv-required=RSA;
SHTTP-Signature-Algorithms; orig-required=RSA;recv-required=RSA;
SHTTP-Message-Digest-Algorithms; orig-required=MD5;recv-required=MD5;
SHTTP-Privacy-Enhancements; orig-required=signature,encrypt;recv-required=encrypt;>

<P>
<h2>Products</h2>
<pre>
Phenol                          $100/liter
Chlorine                        $100/liter
Fullerine                       $1000/nanoliter
<HR>
<H3>To place an order, please fill in the following information:</h3>
<P>
Product:        <INPUT TYPE="text" NAME="Qproduct" MAXLENGTH="256">
<P>
Quantity:       <INPUT TYPE="text" NAME="Qamount" MAXLENGTH="256">
<P>
Total Price:    <INPUT TYPE="text" NAME="Qdollars" MAXLENGTH="256">
<P>
Account number:<INPUT TYPE="text" NAME="Qacct" MAXLENGTH="256">
<P>
<P>
To submit order (encrypted):
<input type="submit" value="GO">
</FORM>
<P>
<HR>
```

a. b. c. d.

Figure 8.15 The S-HTTP enhanced document, `orders.html`

When the hypertext link (see Figure 8.15b) is selected by the user, the browser attempts to set up a secure communication and execute the CGI script, `shttp://www.vcc.com/scripts/take_orders.html`. The `shttp` protocol identifier denotes that an S-HTTP security-enhanced document is to be requested. Second, there is additional data within the hypertext link (see Figure 8.15c). These are attributes of, or other pertinent information about, the document's certificate. These attributes are described according to the standard X.509 certificate format (Kent 1993) :

- `DN` is the distinguished name: Victor Chemical Company. This is the server's name as registered with the certificate authority.
- `OU` is the type of certificate.
- `O` is the name of the certificate authority. In this case, RSA Data Security, Inc. is vouching for the authenticity of the certificate.
- `C` is the country.

The server's public key certificate (identified by CERTS in Figure 8.15a) is embedded at the top of the HTML form.

The HTML form also contains the cryptopts (see Figure 8.15d). In this case, the author of the CGI script has specified that the purchase order form must be encrypted by the Web browser as it is sent to the server (`recv-required=encrypt`). This `encrypt` specification guarantees the confidentiality of the document's contents as it is transmitted over the Internet to the Web server. By default, the server must authenticate to the client with a digital signature. The cryptopts specify that cryptographic algorithms RSA (public key) and MD5 (hashing) will be used for the digital signature. These choices of cryptographic algorithms assure the recipient of the HTML form that Victor Chemical Company is the server and that the form was unchanged in transit.

In practice, the cryptopts may be stored in a variety of ways. The cryptopts may be contained in the HTML document accompanying each secure hypertext link. The cryptopts can also be stored at the beginning of an HTML document to apply to all hyperlinks in that document. Or the cryptopts can be specified in a file to apply to every document on the secure Web server. There are several ways that a document may be protected with S-HTTP. Here are a few examples:

- Plaintext—no crypto options applied
- Signed by the server (authenticated via digital signatures)
- Signed by the client and the server (authenticated via digital signatures)
- Signed and encrypted by the server
- Signed and encrypted by both client and server
- Prearranged private keys

For each class of protection, there are several algorithms to choose from: RSA and DSA are the public key algorithms; MD5, MD2, and SHA are the hashing algorithms; the options for private key algorithms are DES, RC4, RC2, and IDEA.

For this example, the author of the CGI script has specified that the secure communication be signed and encrypted by the server using RSA, MD5, and DES.

8.10.2 Completing the Secure Transaction

Let's look, step by step, as the purchase form is securely sent to the VCC server.

1. First the customer selects a hypertext link and the VCC server sends the client the empty purchase order form (see Figure 8.14). Once the customer fills in the form with the purchase order, he can select the `Secure Submit` button to initiate the secure communication to the VCC server. If the customer wishes to know more about how the form is protected as it is sent, he can select an icon to examine the cryptographic algorithms that will be used in the transaction (see Figure 8.16). The `Secure Anchor Attributes` window shows that the form must be encrypted by the Web browser with `des-ecb` (DES being a specific type of private key cryptography). By default, the server must also always authenticate to the client with a digital signature (RSA public key cryptography and MD5 hashing algorithm).
2. When the customer selects the `Secure Submit` hyperlink, the negotiation of cryptographic algorithms is activated. The S-HTTP-enhanced Web browser must first check to make sure it supports the required cryptographic algorithms. If so, the transaction proceeds.
3. The S-HTTP-enhanced Web browser derives the public key component of the VCC server keypair from the certificate embedded in the purchase order form, `orders.html`, that was sent by the VCC Web server. The browser then generates a DES session key and encrypts the HTTP message

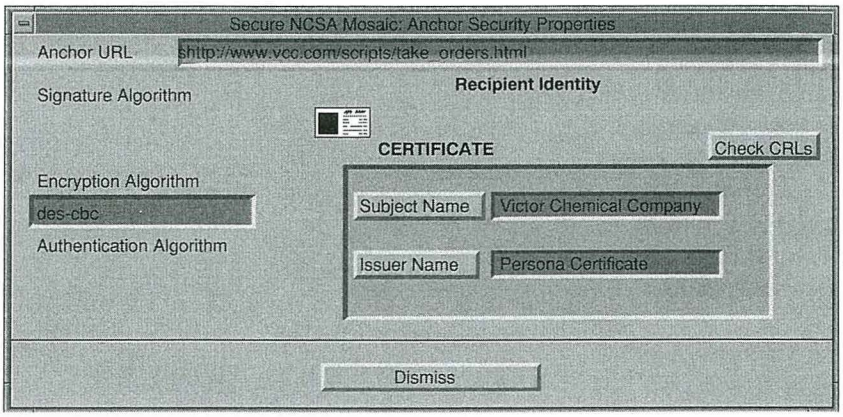

Figure 8.16 Display of secure anchor properties in VCC order form.

body (the customer's purchase order) with the DES session key. The security status icon of the browser's display indicates that encryption is taking place (see Figure 8.13e). The browser creates the encrypted message containing the customer's purchase order and encapsulates it inside the S-HTTP headers. The session key is encrypted with the server's public key and stored in the S-HTTP header. This message would look something like this:

```
Secure-HTTP/1.1 200 OK
Content-Transfer-Encoding: base64
Content-Type: application/http
Prearranged-Key-Info: des-ecb, 87tyhd94736kjkpu4,
inband:1
Content-Privacy-Domain: PKCS-7
—BEGIN PRIVACY ENHANCED MESSAGE—
&(erYWlksdie9023oi568tJ3rty-)VnKuETR
j%3dlfk21)98figusTRsiwldfo*jf%jdh#kfj
—END PRIVACY ENHANCED MESSAGE—
```

The body of the message is encrypted and digitally signed according to the information in the S-HTTP message header. The ciphertext, `87tyhd94736kjkpu4`, is the public key encrypted version of the session key (Rescorla and Schiffman 1995).

4. The encapsulated message, as shown in the preceding step, is received by the VCC server and decoded according to the instructions in the S-HTTP headers. The VCC server decrypts the session key with the private component of its keypair, and then uses that session key to decrypt the HTTP message body. The `take_orders.html` CGI script records the purchase order and then responds to the customer by sending a signed, encrypted purchase receipt (see Figure 8.17). The S-HTTP-enhanced Web server digitally signs the purchase encrypts the body of the document with the shared session key, and then returns the receipt to the Web browser (the customer).
5. The browser receives the signed, encrypted purchase receipt and uses the shared session key to decrypt the ciphertext. During this phase, the browser displays the icon in Figure 8.13e. The browser then verifies the digital signature on the purchase receipt. During this procedure, the browser displays the icon in Figure 8.13h.
6. The transaction is complete; an icon in the upper right of the browser's display indicates the security status of the purchase receipt HTML document (see Figure 8.17). The icon shows that the received document was encrypted and signed as it was transmitted (see Figure 8.13d).

Figure 8.17 VCC signed purchase receipt.

As this example has shown, completing a transaction using S-HTTP is much more complicated than a standard Web transaction. Let's examine how SSL would complete the same type of transaction.

8.11 The Secure Socket Layer (SSL)

SSL appears to the user to be a much simpler implementation than S-HTTP because a lot of the details of the cryptography and the cryptographic controls are hidden in the network layer of the system software. One difference between SSL and S-HTTP is in the handling, storage, and display of the certificates. Certificates and cryptopts are stored by the SSL layer, so the application does not have direct access to them. The secure Web server has one set of certificates and cryptopts that apply to all documents on the server.

SSL relies on the concept of a secure channel. This channel guarantees confidentiality in that all messages that pass over it are encrypted with DES, IDEA, RC2, or RC4. The server (and, optionally, the client) is authenticated with RSA, Diffie-Hellman (Garfinkel 1995; Schneier 1994), or Fortezza. The channel is made reliable by using a hashing algorithm (MD5) to ensure message integrity. Unlike S-HTTP, the choice of negotiating cryptographic algorithms in SSL is all or nothing. The client must negotiate one server authentication algorithm, one private key encryption algorithm, and one message integrity algorithm or else a secure channel is not granted and the request is not fulfilled.

The user has no knowledge of or control over the selection of the cryptographic algorithms. The negotiation is done entirely by the SSL software at the

outset of the communication, before any HTTP messages are sent through the secure channel.

Figure 8.18 illustrates the Web server architecture that Victor Chemical Company might use to offer an SSL-style secure Web service for accepting customers' purchase orders. Since VCC wants to serve plaintext advertising documents and confidential forms for submitting purchase orders, VCC must operate two separate Web servers. The regular (non-crypto) Web server delivers the plaintext documents from port 80, or from whatever port the VCC Web administrator may have selected. A separate SSL-enhanced Web server accepts the confidential purchase orders from VCC customers on port 443, the default port for SSL-protected HTTP communications.

Placing a VCC purchase order via SSL would follow these steps. First, the VCC customer selects a typical hypertext link indicating that they would like to submit a purchase order. The non-crypto Web server sends the Web browser a purchase order form, a plaintext HTML document that contains an `https`-style hypertext link. The user fills in the form and selects the `https` hypertext link. The SSL-enabled Web browser initiates a secure communication by sending a simple `Hello` message to port 443 of the SSL Web server.

At this point, the SSL modules on the client and server begin their handshake, or negotiation of cryptopts. This negotiation is completely transparent to the user. In this phase the client-side SSL module chooses, on behalf of the user, which suite of protocols to use. The client-side SSL module must choose a key

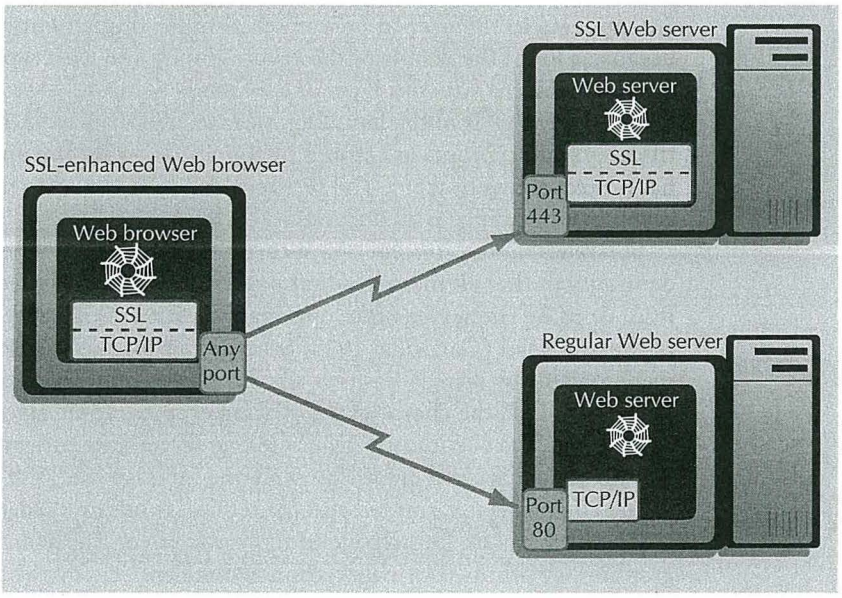

Figure 8.18 The Victor Chemical Company SSL secure Web service.

exchange (server authentication) algorithm, a private key cryptography algorithm, and a message integrity algorithm to use in the secure transaction. The server-side SSL module must inform the client that it can support the set of algorithms chosen by the client.

The SSL layers then begin to set up the spontaneous secure connection, as shown in both Figures 8.9 and 8.18. The server-side SSL software first asserts the server's identity by passing back its certificate to the Web SSL client. The client-side SSL module verifies that the signature of certificate authority on the server's certificate is valid. In current implementations this is done by comparing it to the public key of the certificate authority embedded in the client-side SSL software. The client-side SSL then generates a session key, encrypts that session key with the server's public key, and sends the ciphertext back to the server-side SSL module; if successful, the session key is derived. The server-side SSL module decrypts the message with its own private component of its keypair. Then the server-side SSL module completes the handshake by sending a message back to the client-side SSL module encrypted with the session key. When the client-side SSL module receives this message, it knows that the server-side SSL module was able to decrypt the message it sent. So the server's identity is confirmed. The SSL secure channel is established and is ready to securely transmit the customer's confidential purchase order to the VCC Web server.

Note that the SSL handshake protocol occurs before any HTTP messages are sent. After the secure channel has been established, the Web browser transforms the data in the purchase order form into an HTTP message (including the HTTP header), encrypts this HTTP message using the session key, and sends the message back to the server over the secure channel. The VCC server completes the secure communication by using the same session key derived in the handshake negotiation to encrypt a receipt acknowledging the customer's purchase and sending this back to the Web browser.

The VCC customer receives this secure document and knows that the document was sent securely because one of two icons is displayed, as shown in Figure 8.19. The tiny key in the bottom left of the Web browser and a thin blue line on the top of the document inform the user that the document was sent over a secure channel. The security status of the document received, such as its cryptopts and certificate, can also be displayed. These indicators are so subtle that a user may not even be aware that a secure transaction has occurred.

8.12 Which Is Better?

Having reviewed the mechanisms behind both S-HTTP and SLL, the natural question to ask is, Which is better? Since both implementations are not in their final form, a clear winner can't really be called. But one can make several qualitative observations.

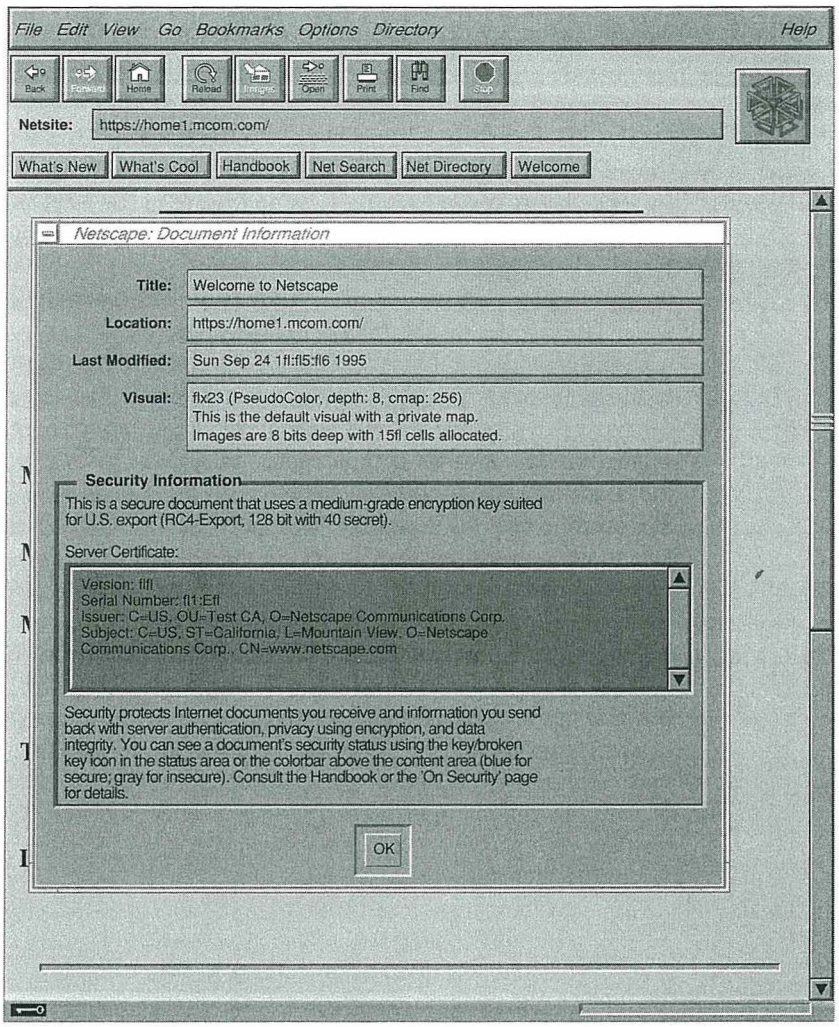

Figure 8.19 SSL document security status.

S-HTTP is a very complete implementation, with all the features one might want and expect from a secure HTTP implementation in the long term. One criticism of the current S-HTTP implementation is the lack of adequate cryptopts and certificate management tools. This is especially a problem for large sites with lots of documents. Imagine, for example, the worst case, where cryptopts are associated with each hyperlink in a secure document. Suppose the service undergoes a major change such that all documents must be moved to a server with a different DN (distinguished name), for example, as might happen in a

corporate merger. In this case, every cryptopt associated with every hyperlink must be changed. The server's ability to dynamically insert the cryptopts and certificates into each document with a server-side include helps but does not solve the problem (Smith 1995).

Encapsulating all security services at the network transport layer as SSL does has both strengths and weaknesses. SSL is very flexible—it can be used to supply security services to a variety of applications: FTP, `telnet`, and so on. SSL is simple because the details of the cryptography are hidden. This makes SSL easy to set up and administer. But this simplicity may be a disadvantage when administrators require finer control over the cryptographic features.

Since the security services are at the network layer, client machines and server machines authenticate, rather than the users and authors of documents. The cryptographic logic is hidden in the network, so the security services stop before they reach the Web application. Authentication needs to be at the application layer for users to be able to digitally sign individual messages and store digitally signed documents. Using SSL, it is not possible for each party in a digital transaction to receive a signed receipt. Also, if the user owns a set of certificates and needs to be able to select the correct one to use for a given transaction, SSL is not designed to provide this level of flexibility and control. Providing these options is not easy with the SSL scheme because the applications, and consequently the users, do not have fine-grained access to the cryptographic features. The point of SSL is to make applications oblivious to such matters. But in the long term the SSL scheme may be too limited for some applications.

Unfortunately, SSL and S-HTTP clients and servers do not interoperate with each other. Even though the browsers of both schemes can also retrieve unprotected documents, they cannot retrieve a secure document protected by any mechanism but their own. If the two implementations do not evolve and converge into a common single secure HTTP standard, the Web community will suffer. There will be a set of security-enhanced servers and clients, many of which do not speak to each other. Customers may need several security enhanced browsers, and service providers may need to offer different secure versions of the same service.

8.13 Electronic Payment Protocols and the Web

The SSL and S-HTTP examples in this chapter have described how a customer could securely send a credit card number to a merchant via the Web. To complete the transaction, though, the merchant must also verify that the customer's credit card number is authorized to make the purchase and, if so, must bill the customer's credit card account for the purchase. Typically, in current Web digital commerce schemes, the credit card company's transaction server is contacted

via the telephone network to complete these authorization and billing aspects of the transaction, as shown in Figure 8.20.

Credit card companies have announced plans to streamline the payment process between the customer, merchant, and credit card processor for purchases made on the Web. Their intent is to fully automate the electronic payment process and to use the existing infrastructure, the credit card transaction processor, for customer and merchant authentication, purchase authorization, and billing (Microsoft Corporation and Visa International 1995; MasterCard 1995).

The credit card companies will own and manage their own certificate authorities. They will issues certificates to both customers and merchants and assume the liability when things go wrong. Unlike SSL and S-HTTP, these electronic payment schemes eliminate the possibility for spontaneous secure transactions. Both the customer and the merchant must register with the financial institution and obtain the necessary certificates before they can perform any secure transactions. Customers must authenticate with personal keys for each secure transaction. In the secure Web schemes, only the merchant authenticates; the customers are anonymous. Strengthening the customer's authentication process decreases the possibility of fraud due to theft and makes electronic commerce less risky than dispensing a credit card number over the telephone. Requiring the customer to possess keys dispensed by the credit card company is a strong way to bind a customer's identity to their credit card account.

When a conventional transaction is completed, everyone involved receives a receipt, a record that the transaction has occurred and been approved by each party. To generate this type of record, electronic payment schemes must happen in a three-way authenticated communication between the customer, merchant, and credit card processor (Bellare et al. 1995). Secure Web services alone cannot set up this type of three-way authenticated communication; they can only set up a secure communication between a Web client and a Web server. Therefore, a new, cryptographically enchanced electronic payment protocol is needed to set up the connection and authenticate the three parties.

Electronic payment schemes complete the electronic equivalent of a conventional transaction. They allow customers to initiate and authorize purchase

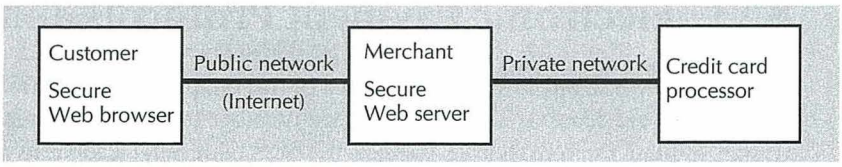

Figure 8.20 Payment with a secure Web service.

requests (the equivalent of signing a paper charge slip). The protocol also permits the merchant to forward the customer's purchase requests to the credit card processor for authorization (the typical charge submission process). And, finally, the protocol permits the credit card processor to authenticate the customer and the merchant, authorize or reject purchase requests, bill the customer's account, and issue receipts to all parties. These systems are still under development and it is not clear, as yet, how they will use secure Web services. Secure Web services may negotiate the cryptographic algorithms and securely deliver messages on behalf of the electronic payment process. However, it is likely that the Web browser and Web server will be used only for browsing and selecting the purchase (see Figure 8.21).

When the customer wishes to make an electronic payment, the request will be handed off to a separate entity, the client's payment tool module, which communicates with the payment processor and merchant via the electronic payment protocol, rather than the Web.

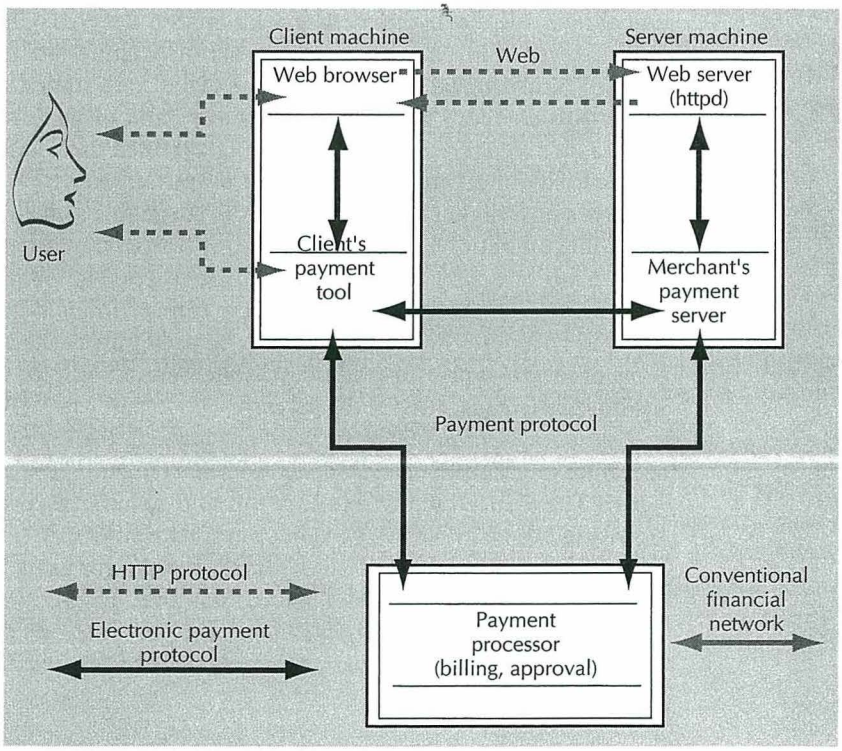

Figure 8.21 Emerging Web payment schemes.

8.14 Private Anonymous Transactions and Digital Cash

One of the disadvantages of digital payment schemes, such as S-HTTP, SSL, and online transactions in general, is that they are so completely traceable. Since the user and the server have to identify themselves to confirm the transaction, it is easy to keep a complete record of every transaction, and thereby collect a detailed financial history of every person and company. Worse, transactions of many sorts—financial, medical, educational, legal—may be correlated to create an extensive dossier on individuals. The essence of the problem is that authenticating, revealing your personal identity, for every valid transaction yields too much information, more than is really needed, to the service provider. All that is really needed is proof that the parties have met the financial or other required terms. Ideally, it should be possible to arrange transactions that are not traceable to the individuals involved.

Several schemes for private and anonymous transactions have been proposed, and at least one, Ecash, has been integrated into the Web (Chaum 1992; Digicash 1995). At this time, anonymous transaction schemes are experimental prototypes of future digital cash systems. The digital cash traded at this time does not convert to currency that is backed by any bank or government. Until digital "banks" are recognized by legally binding, certified fiscal authorities, digital cash can be used only as credit toward other online services.

Here's how digital cash systems work. Through a sequence of complex cryptographic operations, private anonymous transaction schemes limit the information any one party can obtain about the others. The transactions are untraceable: while each party is assured that the transaction is valid, none can identify the other parties. These protocols may be used for private anonymous transactions of several types: for credit-card-style transactions, for controlling access to health care records, and for digital cash.

Digital cash transactions are achieved through the use of an enhanced digital signature mechanism called a *blind signature* (Chaum 1992). A user's digital signature is enhanced with a random number-generating process, such that the signature can be known to be valid, but does not identify the particular person who signed it. The "money" in the Ecash system, for example, consists of individual tokens of various values, issued, stamped, and verified by the "bank." An Ecash token is basically a digital "banknote" or "cashier's check"; it is digitally signed by the "bank," and guarantees payment of the specified amount with no

need to identify the customer who spends it. The protocol is designed so that the "shop" cannot tell whose account the "money" came from and the "bank" cannot tell where the "money" is spent. Using Ecash is like using cash: the transactions are guaranteed to be valid, but the parties may remain anonymous. In addition, Ecash is a true virtual currency; there's no paper or billing required.

To use Ecash through the Web, a software module must be installed on the client system, often called the "wallet" or "purse." The user and the Web browser use the wallet to manage the user's digital cash and to arrange payments. Web servers must also install a similar module, which may run as a CGI program. In the Ecash scheme, the Web server is a "shop"; the information provider establishes a commercial account with one or more "banks," into which customers make payments. The payments are arranged through the auspices of the separate "bank service" using the Internet; the Web server itself does not need to run a "bank."

In a secure Web service, such as SSL or S-HTTP, only two parties, the client and the server, must agree on a common cryptographic protocol. Private anonymous transaction systems, however, are much more complex and require several parties to agree on a common set of cryptographic mechanisms. For example, in digital cash systems such as Ecash, the bank, business, and customer must all agree on a set of protocols. A private anonymous credit card payment system may require up to five parties to cooperate (Maher 1994). Establishing anonymous payment schemes for the Internet will require the creation of a lot of new infrastructure, in the form of certificate authorities and online banks, which will issue digital cash. This infrastructure is in its infancy.

There are many interesting problems that must be addressed if digital cash is deployed on a large scale. Will central banks, such as the United States Federal Reserve, issue digital cash? Or will digital currency be issued by private parties such as Digicash? How will digital currency be converted to and from conventional money? How will taxation be achieved? How do financial and property laws and regulations apply to digital commerce, which often has no geographic locus? Will electronic transactions be accepted as legally binding?

For private anonymous transaction systems to be viable, financial institutions will have to select a set of payment protocols and agree to set up and operate the required infrastructure on the Internet. In light of the uncertainties we have described, it is not clear how soon conventional financial institutions will embrace this technology. However, with the anticipated growth of digital commerce in the next few years, these issues will be hotly debated.

8.15 Conclusion

This chapter has surveyed the requirements for secure Web services and examined the technologies used to meet those requirements. The two secure Web services, SSL and S-HTTP, serve to illustrate the cost and complexity incurred by adding security to any network service. It is clear that the Internet requires new infrastructure, particularly certificate authorities, to become a reliable vehicle for digital commerce. And perhaps the largest-looming question is how Internet digital commerce services will be adopted by future financial institutions. Secure Web services are just a stepping stone to Internet-based online transaction systems.