CHAPTER 10

The Theory of Cryptography

ONE of the essential ingredients of e-business and enterprise computing is cryptography. Cryptography plays a critical role in J2SE and J2EE security, as Part IV of this book demonstrates.

This chapter explains the theory of cryptography that will be used in Chapters 11, 12, and 13. First, this chapter describes secret-key cryptographic systems, as they are at the heart of most cryptographic services, including bulk-data encryption, owing to their inherent performance advantage. Next is an overview of public-key encryption, which is essential for conducting e-business, particularly across public networks, because of the relative ease of distributing cryptographic keys. In Chapter 11, secret- and public-key cryptography services are described in the context of the standard Java APIs: the Java Cryptography Architecture and the Java Cryptography Extension.

For readers who may feel intimidated by the mathematical jargon associated with cryptography, we have tried to explain the mathematics associated with cryptography in a clear and simple way. Our intent is to demystify the concepts and terms surrounding cryptography.

10.1 The Purpose of Cryptography

The purpose of cryptography is to protect data transmitted in the likely presence of an adversary. As shown in Figure 10.1, a *cryptographic transformation of data* is a procedure by which plaintext data is disguised, or *encrypted*, resulting in an altered text, called *ciphertext*, that does not reveal the original input. The ciphertext can be reverse-transformed by a designated recipient so that the original plaintext can be recovered.

Cryptography plays an essential role in

- **Authentication.** This process to prove the identity of an entity can be based on *something you know*, such as a password; *something you have*,

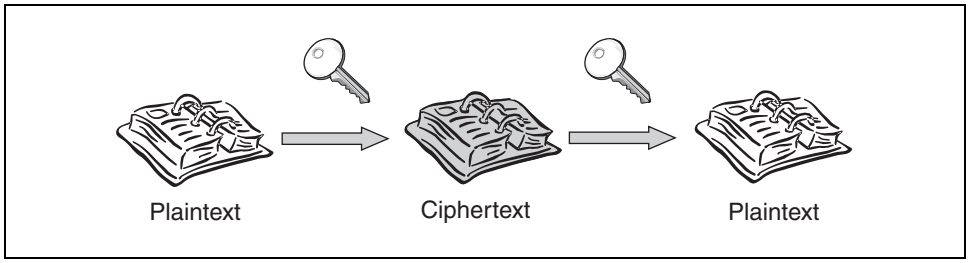

Figure 10.1. The Process of Encryption and Decryption

such as an encryption key or card; *something you are*, such as biometric measurements, including retinal scans or voice recognition; or any combination of these.

- **Data confidentiality.** With this property, information is not made available or disclosed to unauthorized individuals, entities, or processes. When two or more parties are involved in a communication, the purpose of confidentiality is to guarantee that only those parties can understand the data exchanged. Confidentiality is enforced by encryption.
- **Data integrity.** This property refers to data that has not been changed, destroyed, or lost in an unauthorized or accidental manner. The need for data integrity is especially evident if data is transmitted across a nonsecure network, such as the Internet, where a man-in-the-middle attack can easily be mounted. Integrity is enforced by mathematical functions applied to the message being transmitted.
- **Nonrepudiation.** *Repudiation* is the denial by one of the entities involved in a communication of having participated in all or part of the communication. *Nonrepudiation* is protection against repudiation and can be of two types.
  - *Nonrepudiation with proof of origin* provides the recipient of data with evidence that proves the origin of the data and thus protects the recipient against an attempt by the originator to falsely deny sending the data. Its purpose is to prove that a particular transaction took place, by establishing accountability of information about a particular event or action to its originating entity.
  - *Nonrepudiation with proof of receipt* provides the originator of data with evidence proving that data was received as addressed and thus protects the originator against an attempt by the recipient to falsely deny receiving the data.

In most cases, the term *nonrepudiation* is used as a synonym of *nonrepudiation with proof of origin*. Like integrity, nonrepudiation is based on mathematical functions applied to the data being generated during the transaction.

Keeping secrets is a long-standing tradition in politics, the military, and commerce. The invention of public-key cryptography in the 1970s has enabled electronic commerce to blossom in systems based on public networks, such as the Internet.

There are two primary approaches to cryptography (see Figure 10.2). In secret-key cryptography, the key used to decrypt the ciphertext is the same as the key that was used to encrypt the original plaintext. In public-key cryptography, the key used to decrypt the ciphertext is different from but related to the key that was used to encrypt the original plaintext.

Each approach has its strengths and weaknesses. Many of the cryptographic services enterprise applications need use both approaches. However, most application developers will not be aware of the underlying machinery that is deployed. For example, most users of SSL-enabled Web browsers are not aware that both public- and secret-key cryptography are essential parts of the SSL protocol.

Naively, we can think about cryptography primarily as a means for keeping and exchanging secrets. This is the confidentiality property that cryptography affords us. However, other essential cryptographic services are provided. When exchanging a message, whether encrypted or not, we often want to verify its integrity. Someone, particularly in public networks, may have modified the message. Data-integrity verification includes authenticating the origin of the message. Was the message from the source that we think sent the message? Once we accept that the message is from an authenticated entity and was not modified after being created, we also want to consider whether the sender can repudiate—deny sending—the message by claiming that someone stole the cryptographic key used to

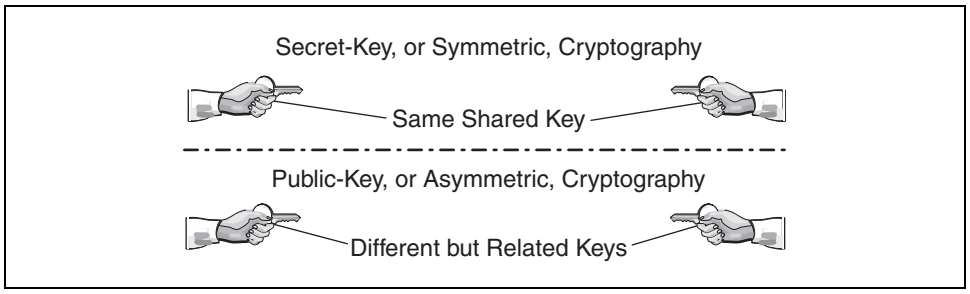

Figure 10.2. Secret-Key and Public-Key Encryption

authenticate the message. Therefore, nonrepudiation is an essential feature of cryptographic systems e-businesses use.

10.2 Secret-Key Cryptography

In *secret-key cryptography*, a sequence of bits, called the *secret key*, is used as an input to a mathematical function to encrypt a plaintext message; the same key is also used to decrypt the resulting ciphertext message and obtain the original plaintext (see Figure 10.3). As the same key is used to both encrypt and decrypt data, a secret key is also called a *symmetric key*.

10.2.1 Algorithms and Techniques

In this section, we examine the most common cryptographic algorithms that are based on the use of a secret key.

10.2.1.1 Substitutions and Transpositions

Some very early cryptographic algorithms manipulated the original plaintext, character by character, using the techniques of substitution and transposition.

- A *substitution*, or *permutation*, replaces a character of the input stream by a character from the alphabet set of the target ciphertext.
- A *transposition* replaces a character from the original plaintext by another character of that same plaintext. This results in shuffling yet still preserving the characters of the original plaintext.

An example of a substitution is the famous Caesar Cipher, which is said to have been used by Julius Caesar to communicate with his army. The Caesar

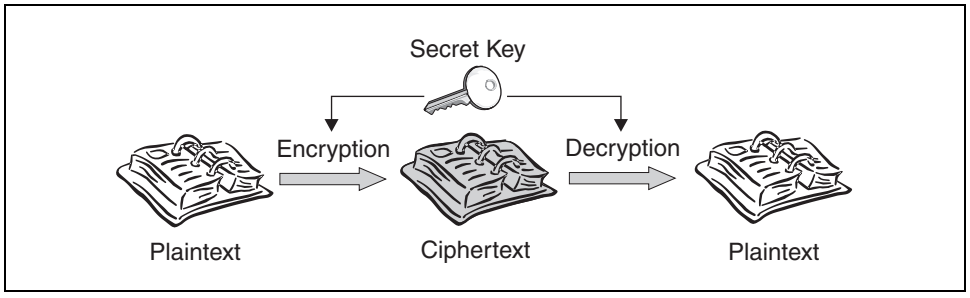

Figure 10.3. Secret-Key Encryption and Decryption

Cipher replaces each character of the input text by the third character to its right in the alphabet set. In Figure 10.4, the value 3 is added to the position of the input character; then modulo 26 is taken to yield the replacement character. If we assign numerical equivalents of 0–25 to the 26-letter alphabet A–Z, the transformation sends each plain character with position P onto the character with position $f(P) := P + 3 \pmod{26}$.

A *transposition cipher* consists of breaking the original plaintext into separate blocks first. A deterministic procedure is then applied to shuffle characters across different blocks. For example, a transposition can split the secret message `"PHONE HOME"` into the two separate blocks `"PHONE"` and `" HOME"`. Then, characters are cyclically shuffled across the two blocks to result in the ciphertext of `"POMHE HOEN"`. Another example of a simple transposition cipher consists of writing the plaintext along a two-dimensional matrix of fixed rows and columns and then simply transposing the matrix, as shown in Figure 10.5.

Original	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Substitution	D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

Figure 10.4. The Caesar Cipher

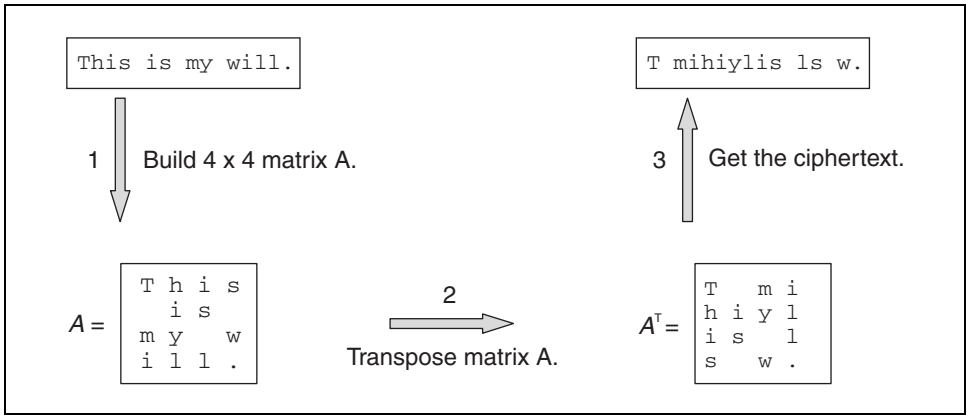

Figure 10.5. Transposition Matrix

Generally, transposition ciphers are easy to break. However, composing them by setting the result of one transposition as the input of another one greatly enhances the ciphering against attacks.

With the age of computers, early modern cryptography carried on these same concepts, using the various elementary transformations that we have listed. The primary difference is that these transformations now apply at the bit level of the binary representation of data instead of characters only.

10.2.1.2 The **XOR** *Operation*

A common transformation is the exclusive OR (XOR) operation, denoted by the symbols XOR, or $\oplus$. XOR is a bitwise function that maps an element of $\{0, 1\} \times \{0, 1\}$ onto the set $\{0, 1\}$, as shown in Figure 10.6. If we interpret the second operand as a key value, the XOR operation can be thought of as a bit-level substitution based on the bit values of the key. With such an assumption, XOR sends a 0 or 1 to itself when the corresponding key bit is 0 and inverts a 0 into a 1 and a 1 into a 0 when the corresponding key bit is 1.

The last property implies that when using a fixed-key value, the XOR operation can be applied to encipher a plaintext, which can then be recovered by simply applying the XOR operation to the ciphertext with the same key value. This property has led to the proliferation of many variants of weak encryption methods that rely solely on the simple XOR operation and thus are easily breakable.

Figure 10.7 shows how to XOR blocks of some plaintext P with a fixed-length key K, leading to ciphertext P'. The figure also shows that if P' is then XORed with K, the original plaintext P is produced.

Knowing a block of plaintext and its XOR transformation directly leads to K, by way of XORing the plaintext with the corresponding ciphertext, as shown in Figure 10.8. Similarly, by knowing two ciphertext blocks P' and Q' alone, one can XOR them together to yield the XOR of the corresponding plaintext blocks P and Q, as in Figure 10.9.

XOR:{0,1} × {0,1} → {0,1}

XOR	0	1
0	0	1
1	1	0

Figure 10.6. The XOR Operation Table

P = 1 0 1 0 1 1 0 0 0 1 0 1 1 1 1 0 0 1 0 1 0 0 0 1

K = 1 1 0 0 0 1 0 1

P	1 0 1 0 1 1 0 0	0 1 0 1 1 1 1 0	0 1 0 1 0 0 0 1
XOR K	1 1 0 0 0 1 0 1	1 1 0 0 0 1 0 1	1 1 0 0 0 1 0 1
= P'	0 1 1 0 1 0 0 1	1 0 0 1 1 0 1 1	1 0 0 1 0 1 0 0

P' = 0 1 1 0 1 0 0 1 1 0 0 1 1 0 1 1 1 0 0 1 0 1 0 0

P'	0 1 1 0 1 0 0 1	1 0 0 1 1 0 1 1	1 0 0 1 0 1 0 0
XOR K	1 1 0 0 0 1 0 1	1 1 0 0 0 1 0 1	1 1 0 0 0 1 0 1
= P	1 0 1 0 1 1 0 0	0 1 0 1 1 1 1 0	0 1 0 1 0 0 0 1

Figure 10.7. XORing Plaintext Blocks with a Fixed-Length Key

P block	1 0 1 0 1 1 0 0
XOR P' block	0 1 1 0 1 0 0 1
K	1 1 0 0 0 1 0 1

Figure 10.8. How to Get the Fixed-Length Key by XORing a Plaintext Block with Its Corresponding Ciphertext Block

P block	1 0 1 0 1 1 0 0
P' block	0 1 1 0 1 0 0 1

Q block	0 1 0 1 1 1 1 0
Q' block	1 0 0 1 1 0 1 1

P block XOR Q block	1 1 1 1 0 0 1 0
P' block XOR Q' block	1 1 1 1 0 0 1 0

Figure 10.9. Ciphertext-Block XOR and Plaintext-Block XOR Equality

Therefore, examining the bit patterns of $P \oplus Q$ can easily result in recovering one of the plaintexts by knowing some information about the other. The plaintext can then be XORed with its ciphertext to yield the keystream, where the *keystream* is the key used to encipher the plaintexts.

Despite the simplicity of the XOR operation and the weakness of encryption algorithms that use it with fixed keys, there is a way to make the sole use of such basic operation result in a perfect encryption scheme. A *one-time pad* is a key of randomly generated digits that is used only once. Use of such a key yields a perfect cipher. Such a cipher is provably secure against attacks in which a code breaker has knowledge of a set of ciphertexts.

The security of the one-time pad stems from the fact that the uncertainty in attempting to guess the keystream is equal to that of directly guessing the plaintext. Note, however, that the length of the keystream for the one-time pad is equal to that of the plaintext being encrypted. Such a property makes it difficult to maintain and distribute keys, which could be very long. This difficulty has led to the development of stream ciphers whereby the key is pseudorandomly generated from a fixed secret key.

10.2.1.3 Stream Ciphers

Stream ciphers are geared for use when memory buffering is limited or when characters are individually transformed as they become available for transmission. Because stream ciphers generally transform plaintext bits independently from one another, error propagation remains limited in the event of a transmission error. For example, the XOR operation lends itself to be used as a stream cipher.

10.2.1.4 Block Ciphers

Block ciphers divide a plaintext into identically sized blocks. Generally, the blocks are of length greater than or equal to 64 bits. The same transformations are applied to each block to perform the encryption.

All the widely known secret-key block-cipher algorithms exhibit the cryptographic properties desired in a block cipher. Foremost of these is the fact that each bit of the ciphertext should depend on all key bits. Changing any key bit should result in a 50 percent chance of changing any resulting ciphertext bit. Furthermore, no statistical relationships should be inferrable between a plaintext and its corresponding ciphertext. In the reminder of this section, we present the most common secret-key block-cipher algorithms.

Feistel Ciphers. A *Feistel cipher* uses a noninvertible function *f*, obtained as a sequence of substitutions and transpositions. A Feistel cipher consists of the following basic steps:

1. A plaintext message *m* is divided into two separate blocks of equal size: the *left block*, *L*, and the *right block*, *R*.

2. The original message, m, is transformed into an intermediate message, m', in which the left block, L', is the same as R, and the right block, R', is $L \oplus f(R)$, where the symbol $\oplus$, as usual, denotes the XOR operation.

These two steps are shown in Figure 10.10. Even though f is a noninvertible function, this design permits recovering m from m' by concatenating $R' \oplus f(L') = R' \oplus f(R) = L$ with $L' = R$.

Steps 1 and 2 must be iteratively repeated a number of times for a Feistel cipher to be secure. The number of iterations depends on the strength of the function f. It is possible to prove that, even with the strongest-possible function f, the iterations must be at least three in order for the Feistel cipher to be reliable.

DES. One of the most widely recognized secret-key block ciphers is the Data Encryption Standard (DES) algorithm. DES was developed by IBM cryptographers in the early 1970s and was adopted as a U.S. government standard in 1976. DES is intended for the protection of sensitive but unclassified electronic information. Because it uses the same key for both encryption and decryption, the algorithm is referred to as a *symmetric cipher*.

DES is a block cipher in which a 64-bit input plaintext block is transformed into a corresponding 64-bit ciphertext output. DES uses a 56-bit key expressed as a 64-bit quantity in which the least relevant bit in each of the 8 bytes is used for parity checking. DES is a Feistel algorithm that iterates over the data 16 times, using a combination of permutation and substitution transformations along with standard arithmetic and logical operations, such as XOR, based on the key value.

For many years, the DES algorithm withstood attacks. Recently, as the result of increased speed of computing systems, DES has succumbed to brute-force attack on several occasions, demonstrating its vulnerability to exhaustive searching of the key space.

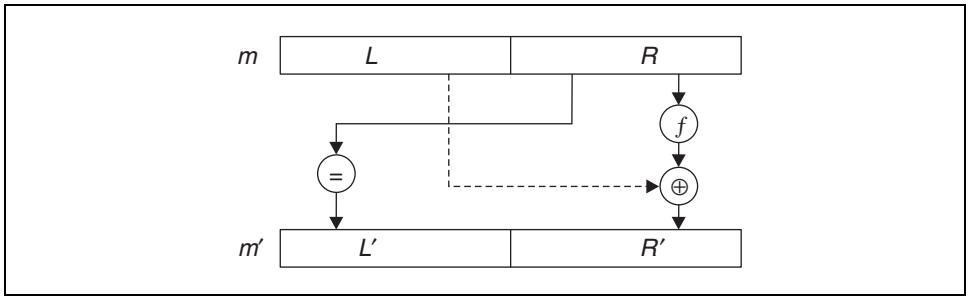

Figure 10.10. Basic Steps of a Feistel Cipher Algorithm

Triple-DES. *Triple-DES* is the DES algorithm applied three times, using either two or three keys.

- With two keys, Triple-DES proceeds by using the first key to encrypt a block of data. The second key is then used to decrypt the result of the previous encryption. Finally, the first key is once more used to encrypt the result from the second step. Formally, let us indicate the encrypting and decrypting functions based on a given key k with E_k and D_k, respectively. If k_1 and k_2 are the two Triple-DES keys and if m is the message to be encrypted, the encrypted message m' is obtained as

  $$E_{k_1}(D_{k_2}(E_{k_1}(m)))$$

  To decrypt m' and obtain the original plaintext m, it is necessary to compute

  $$D_{k_1}(E_{k_2}(D_{k_1}(m')))$$

- The three-key Triple-DES, stronger than the two-key Triple-DES, uses a separate key for each of the three steps described. With the notation that we have introduced, if k_1, k_2, and k_3 are three distinct keys, a plaintext message m is encrypted into its corresponding ciphertext message m' by

  $$E_{k_3}(D_{k_2}(E_{k_1}(m)))$$

  To decrypt m' and obtain the original plaintext m, it is then necessary to compute

  $$D_{k_1}(E_{k_2}(D_{k_3}(m')))$$

In Triple-DES, the second key is used for decryption rather than for encryption to allow Triple-DES to be compatible with DES. A system using Triple-DES can still initiate a communication with a system using DES by using only one key k. Formally, by choosing $k_1 = k_2 = k_3 = k$, the ciphertext m' corresponding to a plaintext message m is obtained from

$$E_k(D_k(E_k(m))) = E_k(m)$$

By contrast, m is obtained from m' by computing

$$D_k(E_k(D_k(m'))) = D_k(m')$$

This shows that Triple-DES with only one key reduces itself to DES.

IDEA. Although less visible than DES, the International Data Encryption Algorithm (IDEA) has been classified by some contemporary cryptographers as the most secure and reliable block algorithm. Like DES, IDEA encrypts plaintext data organized in 64-bit input blocks and for each, outputs a corresponding 64-bit ciphertext block. IDEA uses the same algorithm for encryption and decryption, with a change in the key schedule during encryption. Unlike DES, IDEA uses a 128-bit secret key and dominantly uses operations from three algebraic groups; XOR, addition modulo 2^{16}, and multiplication modulo $2^{16} + 1$. These operations are combined to make eight computationally identical rounds, followed by an output transformation resulting in the final ciphertext.

Rijndael. Recently chosen as the Advanced Encryption Standard (AES), a replacement of DES by the U.S. government, *Rijndael* is an iterated block cipher with a variable block length and a variable key length, both of which can independently be 128, 192, or 256 bits. The strong points of Rijndael are its simple and elegant design and its being efficient and fast on modern processors. Rijndael uses only simple whole-byte operations on single- and 4-byte words and requires a relatively small amount of memory for its implementation. It is suitable for implementations on a wide range of processors, including 8-bit hardware, and power- and space-restricted hardware, such as smart cards. It lends itself well to parallel processing and pipelined multiarithmetic logic unit processors.

A major feature of the Rijndael algorithm is that it presents a departure from the traditional Feistel ciphers. In such ciphers, some of the bits in the intermediate states of a cipher are transposed unchanged. The Rijndael algorithm does not adopt the Feistel structure. Instead, each round of transformations is composed of three distinct invertible subtransformations that treat each bit of the intermediate state of the cipher in a uniform and similar way.

10.2.1.5 Modes of Operation

Modes of operation are cryptographic techniques using block ciphers to encrypt messages that are longer than the size of the block. The most common modes of operation are electronic codebook (ECB) and cipher block chaining (CBC).

ECB. With the ECB mode of operation, a message is divided into blocks of equal size. Each block is then encrypted using a secret key. Figure 10.11 shows how ECB works, assuming the following.

1. The original message m is divided into n blocks $m_1, m_2, \ldots, m_n$.
2. For all $i = 1, 2, \ldots, n$, the plaintext block m_i is encrypted into a ciphertext block c_i with a secret key k. The encryption function associated with k is indicated with E_k. In ECB mode, the block-cipher algorithm typically used for encryption is DES.

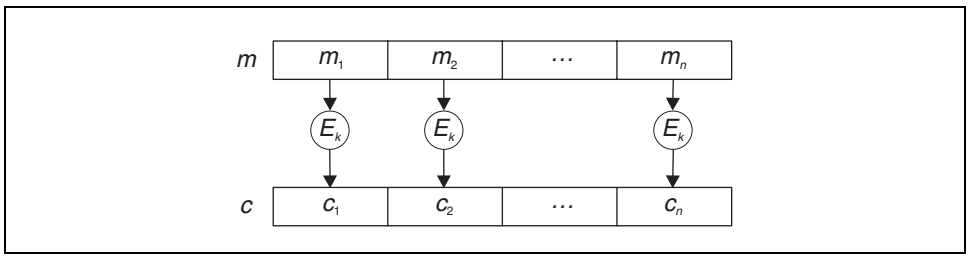

Figure 10.11. ECB Mode

3. The ciphertext blocks $c_1, c_2, \ldots, c_n$ are concatenated to form the ciphertext c corresponding to the message m.

ECB presents some limitations because each ciphertext block depends on one plaintext block only, not on the entire message.

CBC. Given a secret key k, the CBC mode of operation works as follows (see Figure 10.12).

1. The original message m is divided into n blocks $m_1, m_2, \ldots, m_n$.
2. A randomly chosen block of data is selected as the *initial vector* v. This initial vector must be known to the receiver as well. Therefore, a possibility is for both the sender and the receiver to be able to generate v independently as a function of the key k.
3. The first ciphertext block, c_1, is obtained by XORing v with m_1 and encrypting the result of the XOR operation with the secret key k. In other words,

$$c_1 = E_k(v \oplus m_1)$$

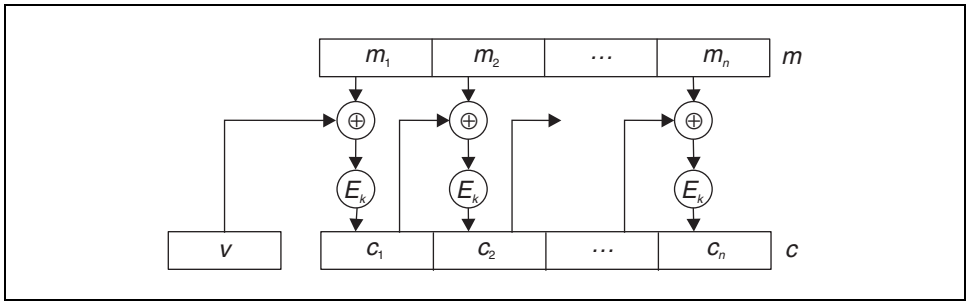

Figure 10.12. CBC Mode

where E_k is the encrypting function associated with the key k.

4. For all $i = 2, \ldots, n$, the ciphertext block c_i is obtained by XORing the plaintext block m_i with the ciphertext block c_{i-1} and encrypting the result of the XOR operation with the secret key k. In other words,

$$c_i = E_k(c_{i-1} \oplus m_i)$$

5. The ciphertext blocks $c_1, c_2, \ldots, c_n$ are concatenated to form the ciphertext c of the message m.

One of the key characteristics of CBC is that it uses a chaining mechanism that causes the decryption of a block of ciphertext to depend on all the preceding ciphertext blocks.

10.2.2 Secret-Key Security Attributes

This section examines the security implications of using secret-key cryptography.

10.2.2.1 Key Space

The strength of modern secret-key encryption methods no longer rests in the secrecy of the algorithm being used but rather in the secrecy of the encryption key. Breaking such cryptographic systems, therefore, can be achieved using a *brute-force attack*, the process of exhaustive searches over the *key space*. The latter is the set of all possible key values that a particular enciphering method can take.

For example, a generalization of the Caesar Cipher is an arbitrary permutation over the English alphabet. This results in 26! (factorial) possible keys corresponding to each of the permutations. Further constraining the permutation method to one that simply maps each letter in the alphabet to one at a fixed number of positions to its right (with a wraparound) and by enciphering each letter at a time (block length = 1), the key space narrows down to the much smaller set of the first 26 integers, $\{1, 2, \ldots, 26\}$. It should be noted, however, that the level of a secret-key encryption algorithm's security is not necessarily proportional to the size of the key space. For example, even though 26! is a very large number, it is possible to break the generalization of the Caesar Cipher by means of statistical analysis.

Most common secret-key cryptographic systems use unique, randomly generated, fixed-size keys. These systems can certainly be exposed to the exhaustive search of the key space. A necessary, although not sufficient, condition for any such cryptographic systems to be secure is that the key space be large enough to preclude exhaustive search attacks using computing power available today and for the foreseeable future. As ironic as it may sound, efficiency of enciphering methods will aid in the exhaustive brute-force search attacks.

10.2.2.2 Confidentiality

Using a secret-key algorithm to encipher the plaintext form of some data content allows only entities with the correct secret key to decrypt and hence retrieve the original form of the disguised data. Reliability of the confidentiality service in this case depends on the strength of the encryption algorithm and, perhaps more important, the length of the key used. The long lifetime of a secret key also might help diminish assurance in such a confidentiality service. Increasing the frequency with which a key is used increases the likelihood that an exhaustive key-search attack will succeed. Most modern systems make use of secret keys that remain valid for only the lifetime of a particular communication session.

10.2.2.3 Nonrepudiation

Secret-key cryptography alone is not sufficient to prevent the denial of an action that has taken place, such as the initiation of an electronic transaction. Although one can apply data privacy in such a scenario, the fundamental flaw of a nonrepudiation scheme based on secret-key cryptography alone is inherent in the fact that the secret key is dispensed to more than one party.

10.2.2.4 Data Integrity and Data-Origin Authentication

At a much lesser cost than encrypting the entirety of a plaintext, data integrity and data-origin authentication can be afforded by a secret cryptographic scheme using a message authentication code (MAC) function. The basic idea is to attach to each message m that is sent across a network the result $h(m)$ of a mathematical function h applied to the message m itself. If an error has occurred during the message transmission, such that the received message a is different from the message m that was originally sent, the message receiver will be able to detect the anomaly by independently computing $h(a)$ and comparing it to $h(m)$ (see Figure 10.13).

The main component of a MAC function is a hash digest function (see Figure 10.14). Hash digest functions are considered one of the fundamental primitives in modern cryptography. By definition, a *hash digest function* is a deterministic function that maps a message of arbitrary length to a string of fixed length n. Typically, n is 128 or 160 bits. The result is commonly known as a *message digest*. As the original data is often longer than its hash value, this result is sometimes also referred to as the original message's *fingerprint*.

Of course, a hash digest function is inherently *noninjective*. This simply means that multiple messages will be mapping to the same digest. In fact, the universe of the messages that can be digested is potentially unlimited, whereas the universe of all the message digests is limited by the set of the 2^n strings with n bits. However, the fundamental premise is that, depending on the strength of the hashing algorithm, the hash value becomes a more compact representation of the original data. This means that, although virtually possible, it should be computa-

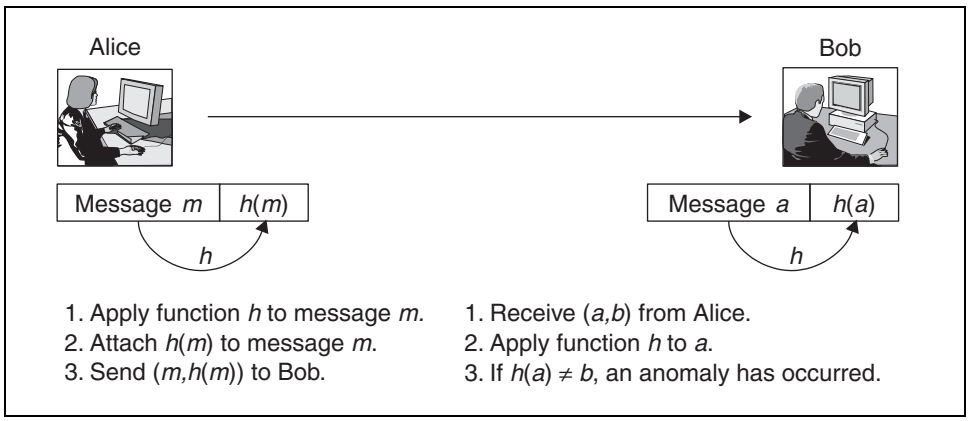

Figure 10.13. Data-Integrity Verification: Basic Scenario

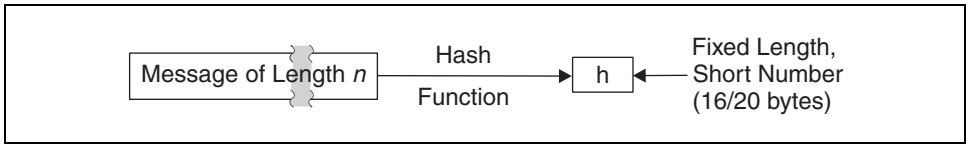

Figure 10.14. Producing a Message Digest with a Hash Function

tionally infeasible to produce two messages having the same message digest or to produce any message having a given, prespecified target message digest.

Message Digest V5 (MD5) and Secure Hash Algorithm V1 (SHA-1) are the most widely used cryptographic hash functions. MD5 yields a 128-bit (16-byte) hash value, whereas SHA-1 results in a 160-bit (20-byte) digest. SHA-1 appears to be a cryptographically stronger function. On the other hand, MD5 edges SHA-1 in computational performance and thus has become the de facto standard.

Hash functions alone cannot guarantee data integrity, because they fail in guaranteeing *data-origin authentication*, defined as the ability to authenticate the originator of a message (see Figure 10.15). The problem with digest functions is that they are publicly available. If a message m is intercepted by an adversary after being transmitted by Alice, the adversary can change m into a different message, m', compute $h(m')$, and send Bob the pair $(m', h(m'))$. By simply applying the function h to the received message m', Bob has no means of detecting that an adversary has replaced m with m'.

Data-origin authentication is inherently supported by secret-key cryptography, provided that the key is shared by two entities only. When three or more parties share the same key, however, origin authenticity can no longer be provided by

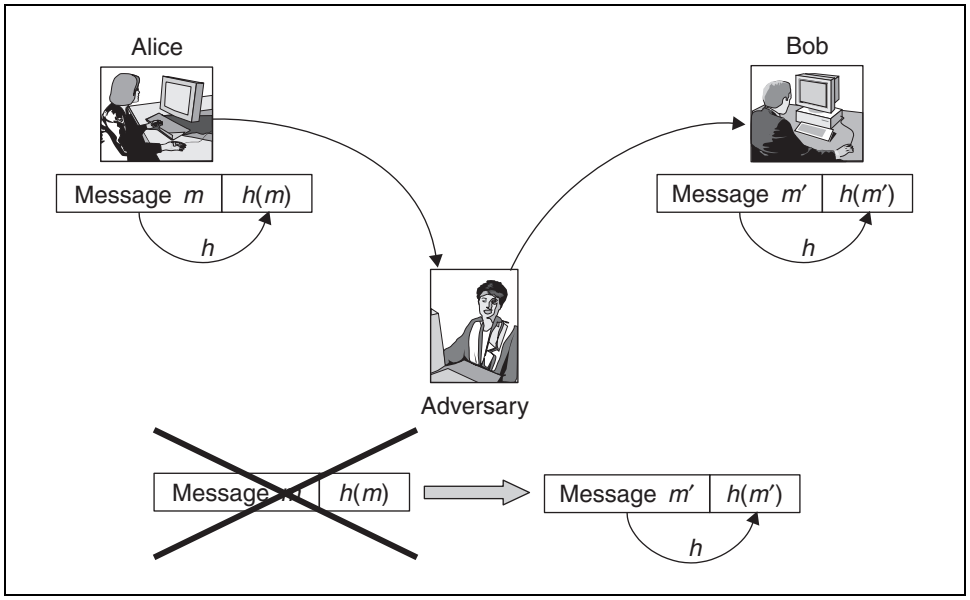

Figure 10.15. Data-Integrity Verification in the Presence of an Adversary

secret-key cryptography alone. Various secret-key-based authentication protocols have been developed to address this limitation. Public-key cryptography, described in Section 10.3 on page 359, provides a simpler and more elegant solution to this problem.

In contrast to using a pure and simple hash function to digest a message, a MAC function combines a hash digest fuction with secret-key encryption and yields a value that can be verified only by an entity having knowledge of the secret key. This way, a MAC function takes care of the problem described in Figure 10.15 and enables both data integrity and data-origin authentication.

Another simple solution to achieve data integrity and data-origin authentication is to apply a regular hash function h, such as SHA-1 or MD5, but rather than hashing the message m alone, the message is first concatenated with the key k, and then the result of the concatenation is hashed. In other words, the sender attaches to the message m the tag $h(k, m)$. This solution, however, exposes some theoretical weaknesses. A more reliable solution consists of attaching the tag $h(k, h(k, m))$.

A MAC can even be computed by using solely a secret-key block-cipher algorithm. For example, the last ciphertext block, encrypted in CBC mode, yields the final MAC value. This is a good choice for a MAC because one of the key characteristics of CBC is that it uses a chaining mechanism that causes the decryption of a block of ciphertext to depend on all the preceding ciphertext blocks. Therefore,

the MAC so defined is a *compact* representation of the *entire* message that can be computed *only* by an entity having knowledge of the secret key. Known instances of this procedure use DES and Triple-DES, resulting in DES-MAC and Triple-DES-MAC, respectively. A MAC mechanism that uses a cryptographic hash function is also referred to as HMAC. HMAC is specified in RFC 2104.[1]

10.3 Public-Key Cryptography

Public-key cryptography emerged in the mid-1970s with the work published by Whitfield Diffie and Martin Hellman.[2] The concept is simple and elegant yet has had a huge impact on the science of cryptography and its application. *Public-key cryptography* is based on the notion that encryption keys are related pairs, private and public. The *private key* remains concealed by the key owner; the *public key* is freely disseminated to various partners. Data encrypted using the public key can be decrypted only by using the associated private key and vice versa. Because the key used to encrypt plaintext is different from the key used to decrypt the corresponding ciphertext, public-key cryptography is also known as *asymmetric cryptography.*

The premise behind public-key cryptography is that it should be computationally infeasible to obtain the private key by simply knowing the public key. Toward achieving this premise, modern public-key cryptography derives from sophisticated mathematical foundations based on the one-way functions existing in the abstractions of number theory.

A *one-way function* is an invertible function that is easy to compute but computationally difficult to invert. A *one-way trapdoor function* is a one-way function that can be easily inverted only if one knows a secret piece of information, known as the *trapdoor.* Encryption is the easy one-way trapdoor function; its inverse, decryption, is the difficult direction. Only with knowledge of the trapdoor—the private key—is decryption as easy as encryption. Two of these currently known one-way functions, factoring large numbers and computing discrete logarithms, form the basis of modern public-key cryptography. Factoring large numbers is a one-way trapdoor function, whereas computing discrete logarithms is a one-way function with no trapdoors.

1. See `http://www.ietf.org/rfc/rfc2104.txt`.
2. W. Diffie and M. E. Hellman. "New Directions in Cryptography," *IEEE Transactions on Information Theory* 22, 6, (1976): 644–654.

10.3.1 Algorithms and Techniques

This section examines the most common cryptographic algorithms that are based on the use of a public- and private-key pair.

10.3.1.1 RSA

The most famous of the well-known trapdoor one-way functions is based on the ease of multiplying two large prime numbers; the reverse process, factoring a very large number, is far more complex. This consideration is at the basis of Rivest-Shamir-Adleman (RSA), certainly the most widely used public-key encryption algorithm.

Basic RSA Concepts. A *prime number*, by definition, is an integer that has no positive divisors other than 1 and itself. A nonprime integer is called *composite*. Two integers $a \geq 1$ and $b \geq 2$ are said to be *relatively prime* if their greatest common divisor GCD(a, b) is 1. The number of elements in the set

$$\{a \in \mathbf{Z} : 1 \leq a < b, \text{GCD}(a, b) = 1\}$$

where $\mathbf{Z}$ is the set of all integers, is often denoted by $\phi(b)$. The function ϕ is called the *Euler phi-function.*

Every integer $b \geq 2$ can be *factored* as a product of powers of primes in a unique way. For example, $60 = 2^2 \times 3 \times 5$. Factoring *large numbers*—numbers that expressed in binary format take 1,024 bits or more—is known to be computationally infeasible with current computing technology. Consequently, the one-way trapdoor problem is to make a very large number a public knowledge and secretly maintain its prime factors. With this in mind, we can now summarize the widely adopted RSA public-key algorithm.

How the RSA Algorithm Works. In simple terms, the *RSA algorithm* centers on three integer numbers: the *public exponent*, e; the *private exponent*, d; and the *modulus*, n. The modulus is obtained as the product of two distinct, randomly picked, very large primes, p and q. A well-known result from number theory implies that $\phi(n) = (p - 1)(q - 1)$. The two numbers e and d are characterized by the fact that they are greater than 1 and smaller than $\phi(n)$. In addition, e must be relatively prime with $\phi(n)$, and it must also be $de = 1 \pmod{\phi(n)}$, which means that d and e are the multiplicative inverse of the other modulo $\phi(n)$. The pair (e, n) is the RSA public key, whereas the pair (d, n) is the RSA private key.

A block of plaintext P whose numerical equivalent is less than the modulus is converted into a ciphertext block by the formula $P^e \pmod{\text{n}}$. Conversely, a ciphertext block C is converted back to its corresponding plaintext representation by the formula $C^d \pmod{\text{n}}$. These two formulas are the inverse of the other. Therefore,

whatever is encrypted with the public key can be decrypted only with the corresponding private key; conversely, whatever is encrypted with the private key can be decrypted only with the corresponding public key.

To better understand how RSA works, let us consider an example involving small numbers. We randomly pick two prime numbers, $p = 7$ and $q = 11$. This implies that $n = p \times q = 77$ and $\phi(n) = (p - 1)(q - 1) = 60$. A valid choice for the public exponent is $e = 13$. By solving the equation $13d = 1 \pmod{60}$, we get $d = 37$. Therefore, the RSA public key in this case is the pair (13, 77), and the corresponding RSA private key is the pair (37, 77). Let us now consider the plaintext message $P = 9$. By encrypting it with the RSA public key, we obtain the ciphertext message $C = 9^{13} \pmod{77} = 58$. To decrypt this message, we have to apply the RSA private key and compute $58^{37} \pmod{77} = 9$, which yields the original plaintext *P*.

To encrypt or decrypt a message, the RSA algorithm uniquely represents a block of data in either a plaintext or ciphertext form as a very large number, which is then raised to a large power. Note here that the length of the block is appropriately sized so that the number representing the block is less than the modulus. Computing such exponentiations would be very time consuming were it not for an eloquent property that the operation of exponentiation in modular arithmetic exhibits. This property is known as the *modular exponentiation by the repeated squaring method.*

Note that the one-way trapdoor function discussed in this section requires deciding on whether a randomly picked very large integer is prime. Primality testing, however, is a much easier task than factorization. Several methods have been devised to determine the primality of an odd number p, the most trivial of which is to run through the odd numbers starting with 3 and determine whether any of such numbers divides p. The process should terminate when the square root of p, $\sqrt{p}$, is reached, because if p is not a prime, the smallest of its nontrivial factors must be less than or equal to $\sqrt{p}$. Owing to the time complexity that it requires, in practice this procedure is stopped much earlier before reaching $\sqrt{p}$ and is used as a first step in a series of more complicated, but faster, primality test methods.

Security Considerations. Breaking the RSA algorithm is conjectured to be equivalent to factoring the product of two large prime numbers. The reason is that one has to extract the modulus n from the public-key value and proceed to factor it as the product of the two primes p and q. Knowing p and q, it would be easy to compute $\phi(n) = (p - 1)(q - 1)$, and the private key (d, n) could then be obtained by solving the equation $de = 1 \pmod{\phi(n)}$ for the unknown d. With the complexity of the fastest known factoring algorithm being in the order of $|n|$, where $|n|$ is the total number of the binary bits in the modulus n, this roughly means that, for example, every additional 10 bits make the modulus ten times more difficult to factor. Given

the state of factoring numbers, it is believed that keys with 2,048 bits are secure into the future. The fastest known factoring algorithm to date is the *number field sieve*.

10.3.1.2 Diffie-Hellman

The Diffie-Hellman (DH) key-agreement algorithm is an elegant procedure for use by two entities establishing a secret cryptographic key over a public network without the risk of exposing or physically exchanging it. Indeed, DH presents a critical solution to the secret-key distribution problem. The security of the algorithm relates to the one-way function found in the discrete logarithm problem.

Basic DH Concepts. Let q be a prime number. An integer α is called a *primitive root*, or *base generator* of q, if the numbers $\alpha \pmod q$, $\alpha^2 \pmod q$, ... , $\alpha^{q-1} \pmod q$ are distinct and consist of the integers from 1 to $q - 1$ in some permutations. For any integer y and a primitive root α of the prime number q, one can find a unique integer exponent x such that $y = \alpha^x \pmod q$. The exponent x is referred to as the *discrete logarithm* of y for the base α modulo q. This is a one-way function. In fact, computing y from x using this function is easy; for q about 1,000 bits long, this would take only a few thousand multiplications. However, the inverse function, $x = \log_\alpha y \pmod q$, which yields x from y, is computationally infeasible, as far as anyone knows; Diffie proved that with q still about 1,000 bits long and the best known algorithm, the discrete logarithm would take approximately 10^{30} operations.

How the DH Algorithm Works. The mathematics encompassed in the DH key-agreement algorithm is fairly simple. Let q and α be as explained previously. These two numbers are publicly available. Suppose that Alice and Bob want to agree on a secret key. Alice generates as her private key a secret random number x_A such that $1 \leq x_A < q$ and publishes the corresponding public key

$$y_A := \alpha^{x_A} \pmod q$$

Similarly, Bob generates as his private key a secret random number x_B such that $1 \leq x_B < q$ and publishes the corresponding public key

$$y_B := \alpha^{x_B} \pmod q$$

The secret key for Alice and Bob is

$$K_{AB} := \alpha^{x_A x_B} \pmod q$$

Alice can obtain this key by getting y_B from a public directory and then computing

$$y_B^{x_A} \pmod{q} = \alpha^{x_B x_A} \pmod{q} = \alpha^{x_A x_B} \pmod{q} = K_{AB}$$

Bob computes the same secret key in a similar way.

One problem in the algorithm that we have just described consists of finding a primitive root α of a given prime number q. The definition of primitive root does not help from a computational point of view, because it requires computing $q - 1$ powers in the worst case for every attempt to find a primitive root. However, a known algebraic theorem proves that an integer α is a primitive root of *9* if $\alpha^i \neq 1$ for any integer $i \in \{1, \ldots, q-1\}$ such that i is a divisor of $q - 1$. Therefore, the problem is reduced to factoring $q - 1$ and testing that $\alpha^i \neq 1$, where this time i varies only in the set of the divisors of $q - 1$. Unfortunately, as we discussed in Section 10.3.1.1 on page 360, factoring a large number is computationally infeasible too. In fact, this is exactly the one-way trapdoor function on which the security of the RSA algorithm relies. However, a solution to this problem for the DH algorithm consists of generating $q - 1$ before generating q itself. In other words, it is possible to generate $q - 1$ as the product of known primes—in which case, the factorization of $q - 1$ is known in advance—and subsequently test q for primality. As discussed in Section 10.3.1.1 on page 360, primality testing is a much easier task than factorization. An advantage of this algorithm is that its security does not depend on the secrecy of q and α. Once a pair of integers (q, α) has been found that satisfies the requirements described previously, the same pair can be published—in cryptography books, for example—and reused by algorithm implementors.

Security Considerations. With the algorithm described, Alice and Bob do not have to physically exchange keys over unsecure networks, because they can compute the same secret key independently of each other. An attacker would have to compute K_{AB} from the only public information available, y_A and y_B. No way to do this is known other than computing the discrete logarithm of y_A and y_B to find x_A and x_B, an operation that, as we said, is conjectured to be computationally infeasible even with the fastest known algorithm.

In order for Bob and Alice to be able to compute the same secret key independently of each other, they have to know each other's public keys. A general security problem that arises at this point is how to ascertain that the public key of an entity belongs to that entity. The DH algorithm does not offer a direct solution to this problem. However, we will see how to solve this problem in Section 10.3.4 on page 372.

10.3.1.3 Elliptic Curve

Recently, elliptic curves over finite fields have been proposed as another source of one-way trapdoor functions for use with existing public-key cryptographic systems.

Basic Elliptic-Curve Concepts. An *elliptic curve* in the plane x, y is the union of the singleton $\{\mathcal{O}\}$ with the set of the points (x, y) of the plane satisfying an equation of the form

$$y^2 + axy + by = x^3 + cx^2 + dx + e$$

where a, b, c, d, and e are real numbers, and x and y take on values in the real numbers. The element $\mathcal{O}$ is called *point at infinity.* For our purpose, it is sufficient to consider equations of the form

$$y^2 = x^3 + ax + b$$

Figure 10.16 shows the elliptic curve with equation $y^2 = x^3 - x$.

A form of *addition* can be defined over the set of points of an elliptic curve by imposing that if any three points on an elliptic curve lie on a straight line, their sum is $\mathcal{O}$. The operation of addition for an elliptic curve, indicated with the symbol +, is constructed on the following rules.

1. The point at infinity, $\mathcal{O}$, is the *additive identity.* This means that $\mathcal{O} = -\mathcal{O}$, and for any point P on the elliptic curve, $P + \mathcal{O} = \mathcal{O} + P = P$.
2. A vertical line meets the elliptic curve at two points with the same coordinates, say $P_1 = (x, y)$ and $P_2 = (x, -y)$. The vertical line also meets the curve at its infinity point, $\mathcal{O}$. This implies that $P_1 + P_2 + \mathcal{O} = \mathcal{O}$, and $P_1 = -P_2$. Therefore, the negative of a point is a point with the same x coordinate but negative y coordinate. This construction is illustrated in Figure 10.17.
3. If Q and R are two points with different x coordinates, draw a straight line between them and find the third point of intersection P_1. It is easily seen that P_1 exists and is unique, unless the line is tangent to the curve at either Q or R, in which case we take $P_1 = Q$ or $P_1 = R$, respectively. Because P_1, Q, and R lie on the same straight line, it must be $Q + R + P_1 = \mathcal{O}$, which implies $Q + R = -P_1$. This construction is illustrated in Figure 10.17.
4. To double a point Q, draw the tangent line in Q and find the other point of intersection S. Then $Q + Q = 2Q = -S$. This construction is illustrated in Figure 10.17.

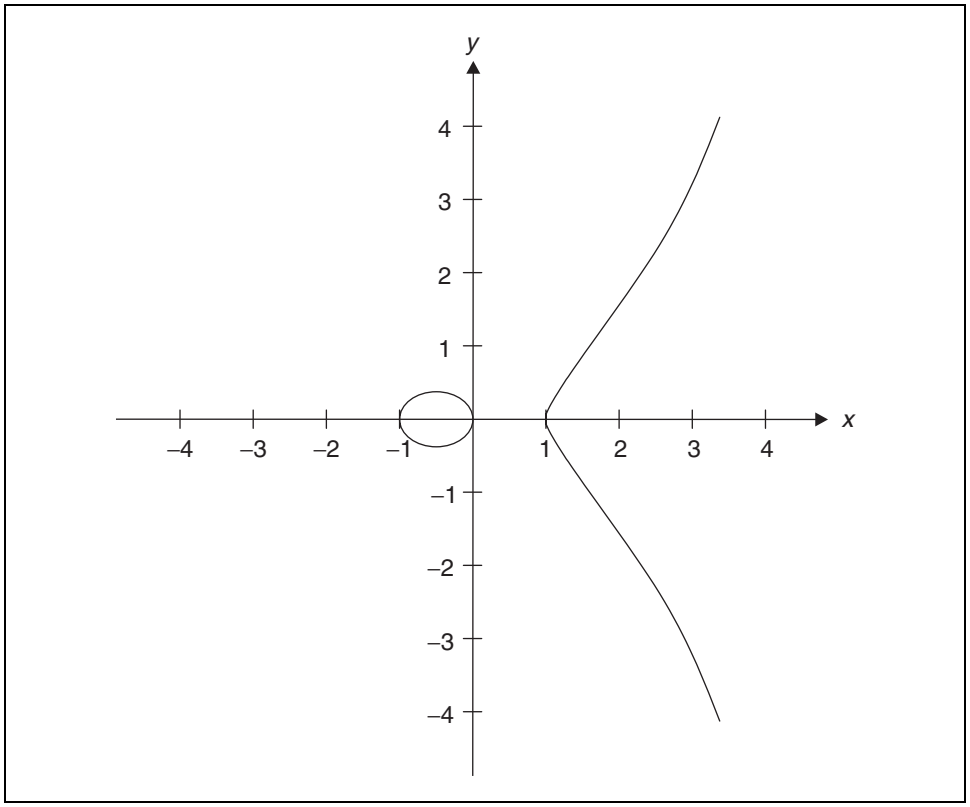

Figure 10.16. An Elliptic Curve

Figure 10.17 shows how to perform the addition operation on the elliptic curve $y^2 = x^3 - x$. It can be shown that if $4a^3 + 27b^2 \neq 0$, the operation of addition constructed on rules 1–4 has the following properties.

- **It is well defined.** Given any two points P and Q on an elliptic curve, their sum $P + Q$ is still a point on the same elliptic curve.
- **It is associative.** Given any three points P, Q, and R on an elliptic curve, $(P + Q) + R = P + (Q + R)$.
- **It is commutative.** Given any two points P and Q on an elliptic curve, $P + Q = Q + P$.
- **It possesses a unity element.** Rule 1 establishes that the unity element for the operation of addtion is the point at infinity, $\mathcal{O}$.
- **Every point on the elliptic curve has an inverse.** Given any point P on an elliptic curve, rules 1 and 2 show how to construct its inverse, $-P$.

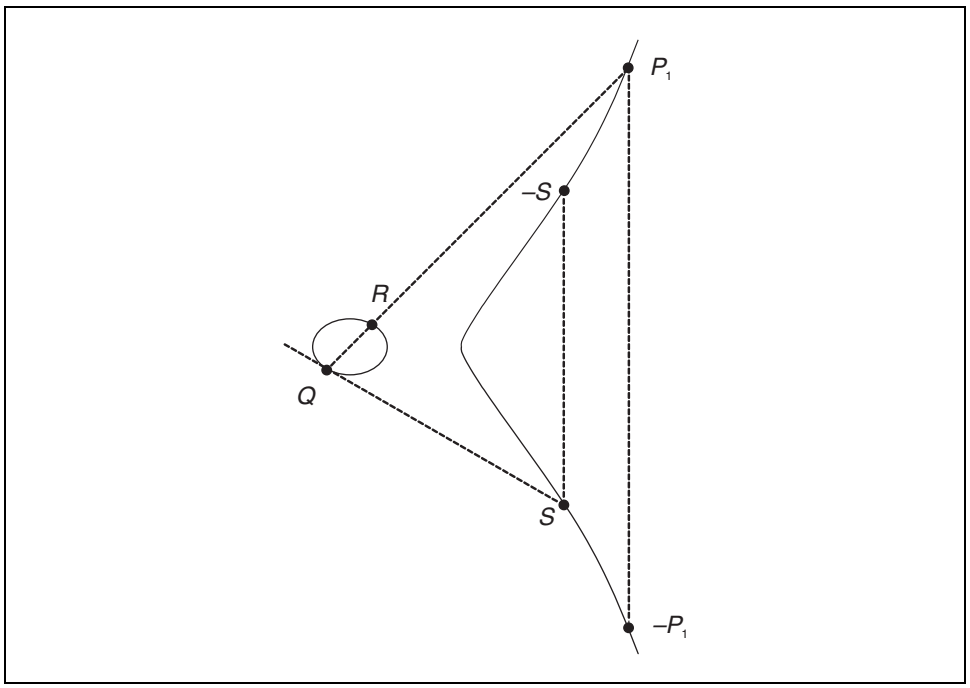

Figure 10.17. The Addition Operation on an Elliptic Curve

These properties can be summarized by saying that the set of the points of an elliptic curve, coupled with the operation of addition that we have just defined, is an *abelian group*. *Multiplication* of a point P on an elliptic curve by a positive integer k is defined as the sum of k copies of P. Thus $2P = P + P$, $3P = P + P + P$, and so on.

An elliptic curve can be defined on a finite field as well. Let $p > 3$ be a prime number. The *elliptic curve* $y^2 = x^3 + ax + b$ over $\mathbf{Z}_p$ is the set of solutions $(x, y) \in \mathbf{Z}_p \times \mathbf{Z}_p$ to the congruence $y^2 = x^3 + ax + b \pmod{p}$, where $a, b \in \mathbf{Z}_p$ are constants such that $4a^3 + 27b^2 \neq 0 \pmod{p}$, together with a special point $\mathcal{O}$, called the *point at infinity*. *Addition* of two points on an elliptic curve and *multiplication* of a point for an integer are defined in a way that is similar to elliptic curves over real numbers.

Note that the equation of an elliptic curve over the finite field $\mathbf{Z}_p$ is defined as for real numbers. The only difference is that an elliptic curve $\mathbf{Z}_p$ is not continuous. Rather, the points that belong in the curve are only the pairs of non-negative integers in the quadrant from (0, 0) to (p, p) that satisfy the equation modulo p.

Given an integer $k < p$ and the equation $Q = kP$, where P and Q are two points on an elliptic curve E over the finite field $\mathbf{Z}_p$, the one-way function here consists

of the easy operation of computing Q given k and P. The inverse problem of finding k given P and Q is similar to the discrete logarithm problem and is, in practice, intractable.

The Elliptic-Curve Algorithm. One straightforward application of the one-way function to DH is for two entities Alice and Bob to publicly agree on a point P on an elliptic curve E over a finite field $\mathbf{Z}_p$, where p is a very large prime number ($p \approx 2^{180}$). The criterion in selecting P is that the smallest integer value of n for which $nP = \mathcal{O}$ be a very large prime number. The point P is known as the *generator point*. The elliptic curve and the generator point are parameters of the cryptosystem known to all the participants.

To generate the key, the initiating entity, Alice, picks a random large integer $a < n$, computes aP over E, and sends it to the entity Bob. The integer a is Alice's private key, whereas the point aP is her public key. Bob performs a similar computation with a random large number b and sends entity Alice the result of bP. The integer b is Bob's private key, whereas the point bP is his public key. Both entities then compute the secret key $K = abP$, which is still a point over E.

Security Considerations. Given an elliptic curve E on a finite field $\mathbf{Z}_p$, where p is a very large prime number, the security of elliptic-curve cryptography depends on how difficult it is to determine the integer k given a point P on the curve and its multiple kP. The fastest known technique for taking the elliptic-curve logarithm is known as the *Pollard rho method*. With this algorithm, a considerably smaller key size can be used for elliptic-curve cryptography compared to RSA. Furthermore, it has been shown that for equal key size, the computational effort required for elliptic-curve cryptography and RSA is comparable. Therefore, there is a computational advantage to using elliptic-curve cryptography with a shorter key length than a comparably secure RSA.

10.3.2 Public-Key Security Attributes

This section examines the security implications of using public-key cryptography. Generally speaking, the strength of each algorithm is directly related to the type of the one-way function being used and the length of the cryptographic keys. Inverting the one-way functions we have discussed, namely, factoring a very large number and computing the discrete logarithm, is known to be practically infeasible within the computing means and the theoretic knowledge available today.

10.3.2.1 Confidentiality

The premise of the privacy service here is achieved by encrypting data, using the recipient's public key, and the fact that decryption can be done only by using the recipient's private key. For example, if Alice needs to send a confidential message

to Bob, she can encrypt it with Bob's public key, knowing that only Bob will be able to decrypt the ciphertext with his private key (see Figure 10.18).

Thus, only the recipient with knowledge of the private key is able to decrypt the enciphered data. It is worth noting that a privacy service strongly depends on the assurance that a public key is valid and legitimately belongs to the recipient.

One confidentiality problem that needs to be addressed by public-key encryption is the fact that in some cases, the plaintext corresponding to a given ciphertext can be easily understood. As an example, we consider the scenario in which Alice is a stock client and Bob a stockbroker, as shown in Figure 10.19.

Typically, Alice's messages are all likely to be of the type "Buy" or "Sell." Knowing this, an attacker could build a table mapping ciphertexts to plaintexts.

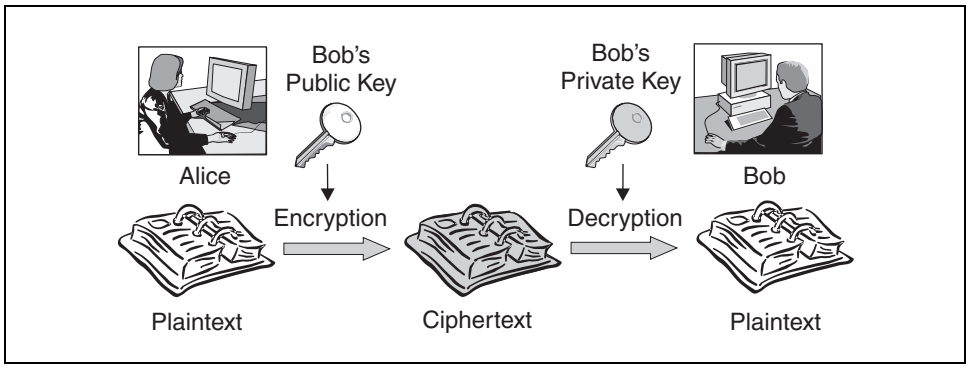

Figure 10.18. Public-Key Scenario

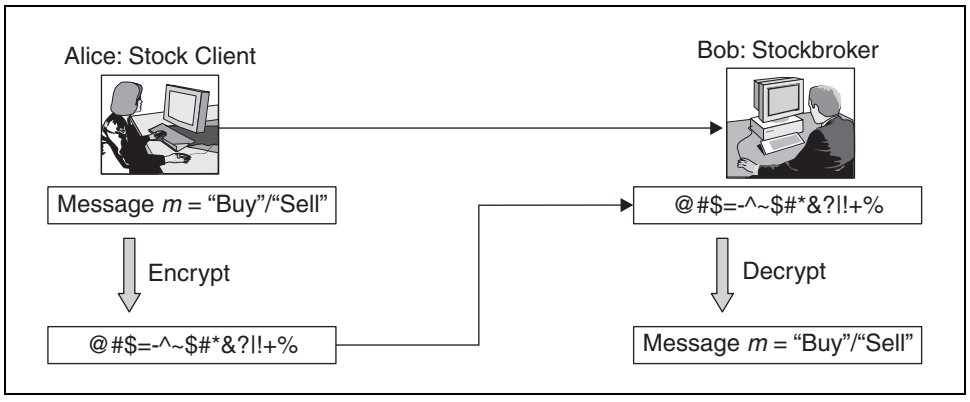

Figure 10.19. Scenario Requiring Message Randomization

This would break the confidentiality of the transmission. Even worse, the attacker could impersonate Alice and replace the ciphertext corresponding to "Buy" with the ciphertext corresponding to "Sell" and vice versa (see Figure 10.20).

This problem can be solved by *randomizing* the message. Before encrypting the plaintext message "Buy" or "Sell," the message-randomizing algorithm on Alice's side inserts a meaningless sequence of bits, which is randomly generated. As the ciphertext depends on the entire plaintext message, the ciphertexts produced by Alice are no longer recognizable. In addition, message randomization reduces the risks of message-prediction-and-replay-attacks (see footnote 6 on page 150).

10.3.2.2 Data Integrity, Data-Origin Authentication, and Nonrepudiation

As we said in Section 10.3.2.1 on page 367, privacy is provided by encrypting data, using a publicly available key, typically the recipient's public key. However, an eavesdropper may intercept the data, substitute new data, and encrypt it using the same public key. Simply applying a public-key algorithm to achieve privacy does not guarantee data integrity; nor does it guarantee data-origin authentication. In practice, digital signatures are the preferred method of achieving data integrity and data-origin authenticity. Another service that is inherently offered through digital signatures is nonrepudiation.

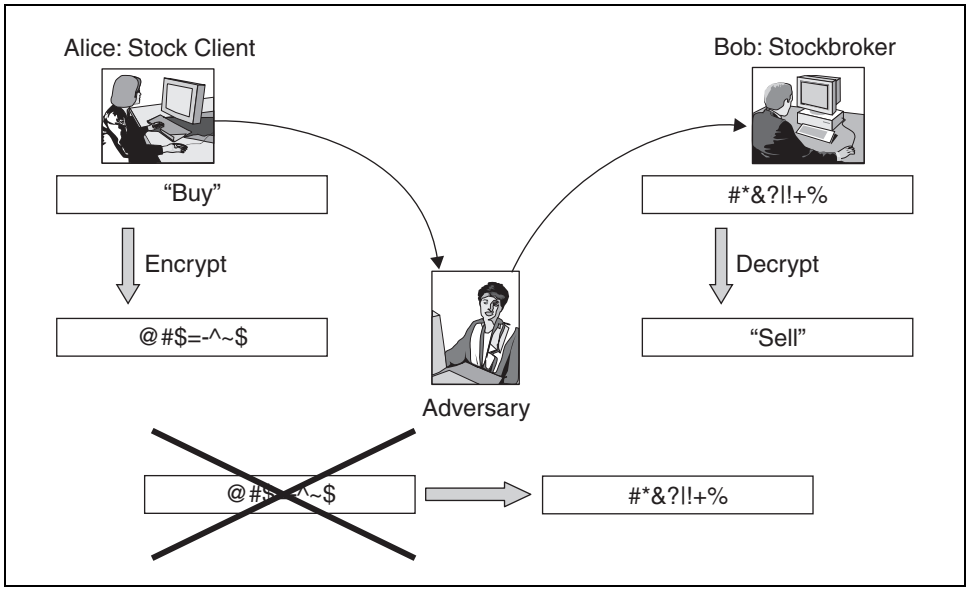

Figure 10.20. Message Randomization

10.3.3 Digital Signatures

The use of public-key cryptography combined with one-way hash functions enables the digital signing of documents. This process inherently enables data integrity and data-origin authentication and has the potential to withstand repudiation. In fact, using the private key of a public- and private-key pair to encrypt a data stream automatically binds the subject—a person or an entity—with the encrypted data.

The cost of encrypting an entire document in order to simply establish this binding can be prohibitive. Fortunately, digital signing of a document is a computationally affordable alternative, as it does not require encrypting the entire document.

If confidentiality is a requirement, the message originator, Alice, encrypts the message only once, with the public key of the receiver, Bob, thereby guaranteeing confidentiality, because the ciphertext can be decrypted only with the Bob's private key. However, data integrity, data-origin authentication, and nonrepudiation are not guaranteed, because anybody could have used Bob's public key to encrypt a different message, pretending that it was sent by Alice. To resolve this ambiguity, Alice attaches a digital signature to the encrypted message. The digital signature is obtained by applying a mathematical function to the plaintext message. This mathematical function depends on Alice's private key.

An eavesdropper who attempted to replace the transmitted data with new data could still encrypt the new data with Bob's public key but would not be able to use Alice's private key to generate Alice's digital signature. Once he receives the encrypted message and the digital signature, Bob decrypts the ciphertext with his own private key. Finally, he uses Alice's public key to verify Alice's digital signature. If the digital signature verifies, Bob knows that the original message has been sent by Alice and has not been compromised during transmission. Because Alice's private key has been used to compute the digital signature, this entire process guarantees data integrity, data-origin authentication, and nonrepudiation (see Figure 10.21).

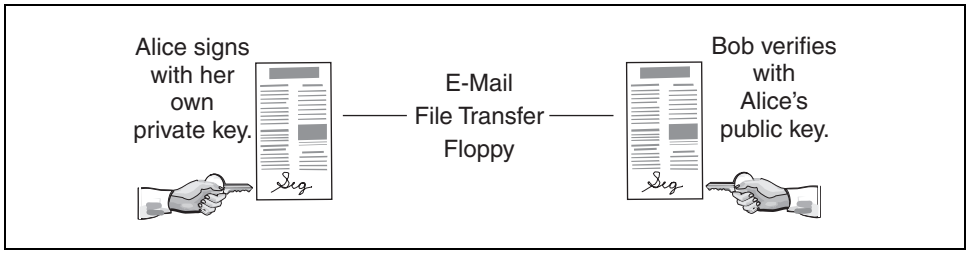

Figure 10.21. Digital-Signature Scenario

With the fundamental premise that the private key remains in the confines of its owner, verifying a digital signature using the associated public key certainly leaves no possibility for the originator to deny involvement. Denial, however, can always take place on the basis that a private key has been compromised. A strong nonrepudiation service never exposes the private keys it manages, even to the owner. Tamper-proof hardware modules for private keys become necessary for a legally binding nonrepudiation service.

If a confidentiality service is not needed, Alice can transmit the signed document to Bob in its cleartext form. The signature is provided to Bob for data-integrity verification, data-origin authentication, and nonrepudiation purposes.

The most well-known digital signature algorithms are RSA and Digital Signature Algorithm (DSA). These algorithms are discussed in the next two subsections.

10.3.3.1 RSA Signature

The RSA digital-signature algorithm proceeds along two main steps, as shown in Figure 10.22.

1. Using one of the common hashing algorithms, such as MD5 or SHA-1, a document is first digested into a much smaller representation: its hash value.
2. The hash value of the document, rather than the entire document itself, is then encrypted with the private key of the originator.

If confidentiality is needed, the document itself must be encrypted, as explained in Section 10.3.2.2 on page 369.

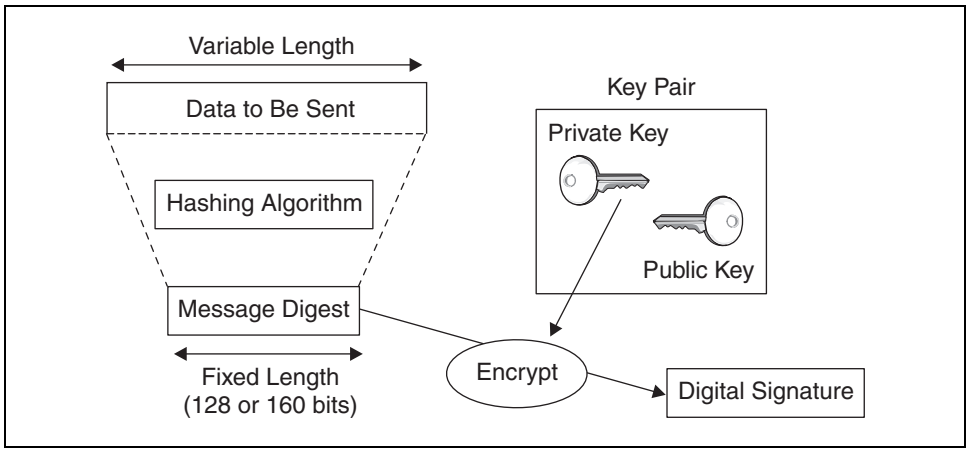

Figure 10.22. The Process of Computing a Message's RSA Digital Signature

10.3.3.2 DSA Signature

Other types of digital signatures rely on algorithms designed solely for signing but not encrypting. In other words, the digital signature is still obtained by encrypting the hash value of a document with the originator's private key, but the public and private key pair here can be used only for digital signing, not for encrypting arbitrary-size messages.

An example of this class of algorithms is the standard DSA, which computes a signature over an arbitrary-size input, using SHA-1 as a message digest, five public parameters, and a private key. DSA signatures have better performance characteristics than RSA does.

10.3.4 Digital Certificates

As we mention in Section 10.3.5 on page 375, authenticating the identity of a sending entity and protecting data to allow only authenticated and authorized receivers to view that data is an extremely important security requirement, especially for the exchange of security-sensitive data or when the nature of the transaction requires data-origin authentication and nonrepudiation. Encrypting a message with the receiver's public key guarantees confidentiality, whereas digitally signing a message by encrypting its hash value with the originator's public key guarantees data-origin authentication and nonrepudiation.

These scenarios are very attractive, but for them to work, it is necessary to have a means to bind a public- and private-key pair to its owner. To understand why, let us consider the following scenario. Alice wants to send Bob a confidential message in a secure manner over a public network. To do so, she needs to encrypt the message with Bob's public key. For sure, only Bob will be able to read the message once it is transmitted, because the message's ciphertext can be decrypted only with Bob's private key. However, how can Alice be sure that Bob is really Bob? Owning a public- and private-key pair does not give any assurance about the real identity of a person. Similarly, Bob may receive a signed message from Alice, and he can verify the digital signature's authenticity by decrypting it with Alice's public key, but how can he be sure that the entity that signed the message declaring to be Alice is really Alice?

A solution to this problem is to use digital certificates, which can be used to exchange public keys and to verify an entity's identity. An entity's *digital certificate* is a binary file that contains the entity's public key and Distinguished Name (DN), which uniquely identifies that entity, along with other pieces of information, such as the start and expiration dates of the certificate and the certificate's serial number (see Figure 10.23).

The international standard for public-key certificates is called X.509 (see Appendix B on page 553). This standard has evolved over time, and the latest version is V3. The most significant enhancement in X.509 V3 is the ability to add other,

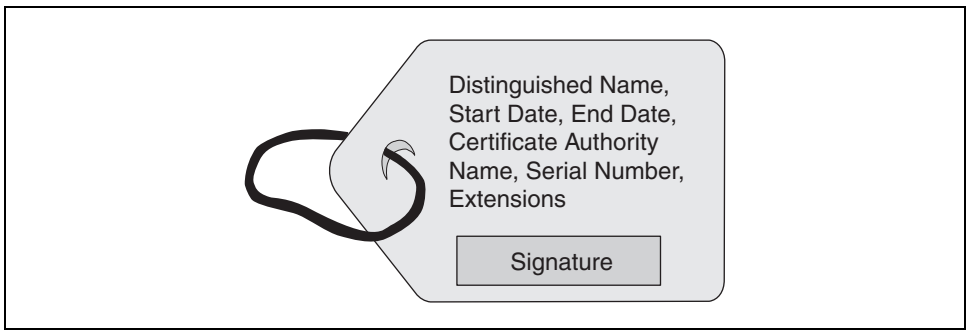

Figure 10.23. Information Contained in a Digital Certificate

arbitrary data in addition to the basic name, address, and organization identity fields of the DN. This is useful when constructing certificates for specific purposes. For example, a certificate could include a bank account number or credit card information.

Digital certificates are released by trusted third-party registry organizations called Certificate Authorities. These CAs are public organizations that are trusted by both the sender and the receiver participating in a secure communication. An entity, Alice, can receive her own certificate by generating a public- and private-key pair and by transmitting the public key along with a certificate request and proof of ownership of the public key to a CA. For serious applications, Alice can obtain a certificate only by applying in person and showing evidence of her identity. If Alice's request for a certificate is accepted, the CA wraps Alice's public key in a certificate and signs it with its own private key.

Alice can now convey her public key information to other entities by transmitting her certificate. A receiving entity, Bob, can verify the certificate's authenticity by verifying the CA's digital signature. This can be done without even contacting the CA, because CAs' public keys are available in all the most common client and server applications, such as Web browsers, Web servers, and other programs that require security. If the signature is verified, Bob is assured that the certificate really belongs to Alice. From this moment on, when he receives a message digitally signed by Alice, he knows that it is really Alice who signed it and transmitted it—data-origin authentication—and Alice will not be able to deny that the message originated from her—nonrepudiation. Similarly, by accessing Bob's certificate from a CA and by encrypting a message with Bob's public key, Alice is assured that only Bob, and no other person, will be able to decrypt the message—confidentiality.

As Figure 10.23 shows, certificates contain start and end dates. The validity of a certificate should not be too long, to minimize the risks associating with having inadvertently exposed the associated private key and to make sure that the current

key strength still makes it computationally infeasible to compute the private key from the public key. If the private key associated with the public key in a certificate gets inadvertently exposed, a certificate's owner should make an immediate request for suspending the certificate's validity. In this case, the CA will add an entry for that certificate in its certificate revocation list. A CRL also enumerates those certificates that have been revoked because their owners failed to comply with specific requirements. A CRL should always include data explaining why a certificate was suspended or revoked.

In the scenario that we have described in this section, there is only one CA that the sender and the receiver participating in a secure communication use to verify each other's public key's authenticity. In real-life situations, there are chains of CAs, whereby each successive CA verifies and vouches for the public key of the next identity in the chain. In this case, a public-key certificate embodies a chain of trust. Consider the situation shown in Figure 10.24.

A system has received a request containing a chain of certificates, each of which is signed by the next higher CA in the chain. The system has also a collection of *root certificates* from trusted CAs. The system can then match the top of the chain in the request with one of these root certificates, say, Ham's. If the chain of signatures is intact, the receiver can infer that Nimrod is trustworthy and has

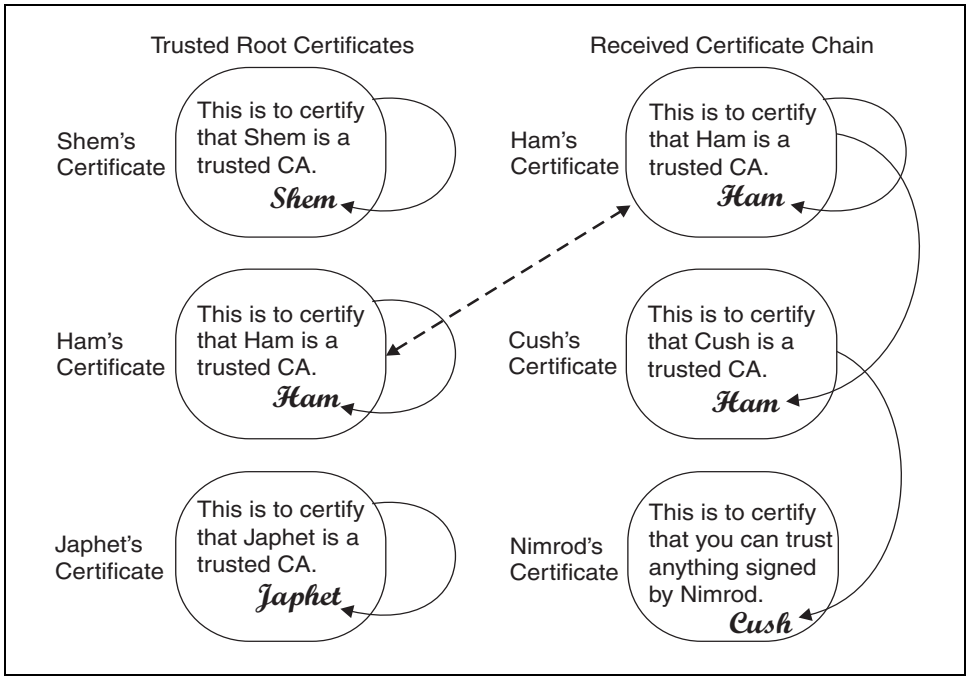

Figure 10.24. Certificate Hierarchy

inherited the trustworthiness from Ham. Note that one of the implications of a certificate chain is that the certificate at the top of the chain is *self-signed*.

10.3.5 Key Distribution

In public-key cryptography, an entity's private key never has to be exposed, whereas the corresponding public key is made publicly available. This makes it possible to obtain confidentiality, nonrepudiation, data-origin authentication, and data integrity without having to distribute the secret key. The main problem of public-key cryptography is that it is computationally expensive. Conversely, secret-key cryptography, described in Section 10.2 on page 346, offers better performance and scales well for Kerberos and distributed computing environment (DCE) security, even though its limitation lies in the fact that it becomes necessary to share the secret key across unsecure networks.

Combining public-key and secret-key cryptography yields the performance advantages of secret-key cryptography and the security enhancements of public-key cryptography, as shown in Figure 10.25. One algorithm that combines public-key and secret-key cryptography is DH (see Section 10.3.1.2 on page 362), which allows two parties to independently compute the same secret key. To do this, each

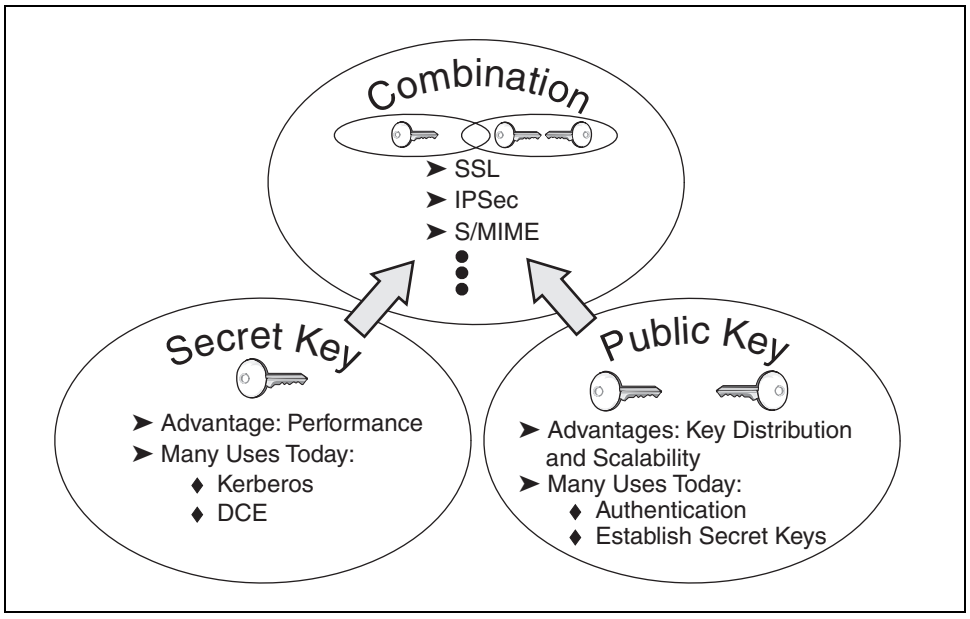

Figure 10.25. Combining Public-Key and Secret-Key Cryptography

entity uses its own private key and the other entity's public key. With Diffie-Helmann, the shared secret key is mathematically computed by the two parties, and there is no need to physically exchange it over the network.

Another way to use public-key cryptography for secure secret-key establishment over a public network is, essentially, to consider the secret key as the data that needs to be distributed with a privacy requirement. Thus, the secret key is encrypted using the public key of the target entity. The receiving entity uses its private key to decrypt the enciphered secret key and hence has established a common secret key with the sending entity. This is, for example, the approach used by the SSL and TLS protocols (see Section 13.1 on page 449). Other protocols that combine secret- and public-key cryptography are IPSec and S/MIME (see Section 12.2 on page 439).

Note that authenticating the identity of the sending entity is a strong security requirement. A breach in such a key-establishment mechanism risks exposing the entire cryptographic channel that follows key establishment.