PART 7

Security

Objectives

No one can deny the importance of security in data communications and networking. Security in networking is based on cryptography, the science and art of transforming messages to make them secure and immune to attack. Cryptography can provide several aspects of security related to the interchange of messages through networks. These aspects are confidentiality, integrity, authentication, and nonrepudiation.

Cryptography can provide confidentiality, integrity, authentication, and nonrepudiation of messages.

Cryptography can also be used to authenticate the sender and receiver of the message to each other. For example, a user who needs access to the resources of a system must first be authorized. We call this aspect entity authentication.

Cryptography can also provide entity authentication.

In this part of the book, we first introduce cryptography without delving into the mathematical foundations of this subject. We then briefly explore security aspects as applied to a network. Finally, we discuss some common protocols that implement security aspects at the upper three layers of the Internet model.

Part 7 of the book is devoted to different security aspects.

Chapters

This part consists of three chapters: Chapters 30, 31, and 32.

Chapter 30

Chapter 30 is a brief discussion of a broad topic called cryptography. Although cryptography, which is based on abstract algebra, can itself be a complete course, we introduce it here briefly and avoid references to abstract algebra as much as possible. We give just enough background information as a foundation for the material in the next two chapters.

Chapter 31

Chapter 31 is an introduction, and motivation, for the broad topic of network security. We discuss selected issues that are often encountered when dealing with communications and networking problems.

Chapter 32

Chapter 32 briefly discusses the applications of topics discussed in Chapters 30 and 31 to the Internet model. We show how network security and cryptography can be used in three upper layers of the Internet model.

CHAPTER 30

Cryptography

Network security is mostly achieved through the use of cryptography, a science based on abstract algebra. In this chapter, we briefly discuss the cryptography suitable for the scope of this book. We have tried to limit our discussion of abstract algebra as much as we could. Our goal is to give enough information about cryptography to make network security understandable. The chapter opens the door for studying network security in Chapter 31 and Internet security in Chapter 32.

30.1 INTRODUCTION

Let us introduce the issues involved in cryptography. First, we need to define some terms; then we give some taxonomies.

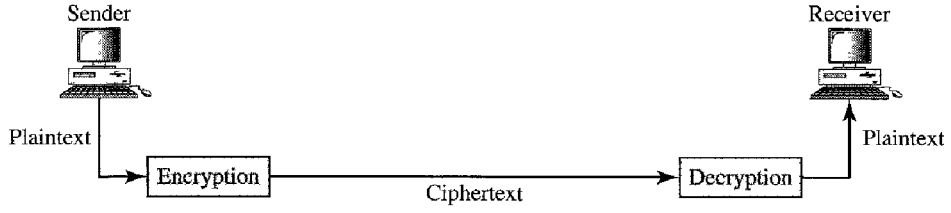
Definitions

We define some terms here that are used in the rest of the chapter.

Cryptography

Cryptography, a word with Greek origins, means "secret writing." However, we use the term to refer to the science and art of transforming messages to make them secure and immune to attacks. Figure 30.1 shows the components involved in cryptography.

Figure 30.1 *Cryptography components*



Plaintext and Ciphertext

The original message, before being transformed, is called **plaintext.** After the message is transformed, it is called **ciphertext.** An **encryption algorithm** transforms the plaintext into ciphertext; a **decryption algorithm** transforms the ciphertext back into plaintext. The sender uses an encryption algorithm, and the receiver uses a decryption algorithm.

Cipher

We refer to encryption and decryption algorithms as **ciphers.** The term *cipher* is also used to refer to different categories of algorithms in cryptography. This is not to say that every sender-receiver pair needs their very own unique cipher for a secure communication. On the contrary, one cipher can serve millions of communicating pairs.

Key

A **key** is a number (or a set of numbers) that the cipher, as an algorithm, operates on. To encrypt a message, we need an encryption algorithm, an encryption key, and the plaintext. These create the ciphertext. To decrypt a message, we need a decryption algorithm, a decryption key, and the ciphertext. These reveal the original plaintext.

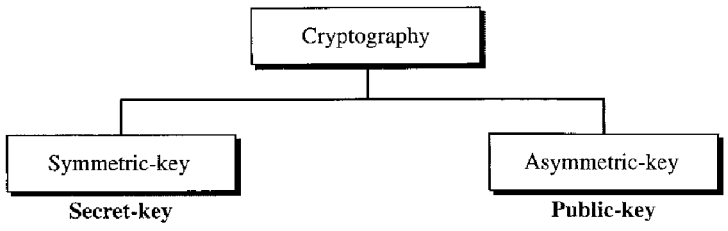
Alice, Bob, and Eve

In cryptography, it is customary to use three characters in an information exchange scenario; we use Alice, Bob, and Eve. Alice is the person who needs to send secure data. Bob is the recipient of the data. Eve is the person who somehow disturbs the communication between Alice and Bob by intercepting messages to uncover the data or by sending her own disguised messages. These three names represent computers or processes that actually send or receive data, or intercept or change data.

Two Categories

We can divide all the cryptography algorithms (ciphers) into two groups: symmetric-key (also called **secret-key**) cryptography algorithms and asymmetric (also called **public-key**) cryptography algorithms. Figure 30.2 shows the taxonomy.

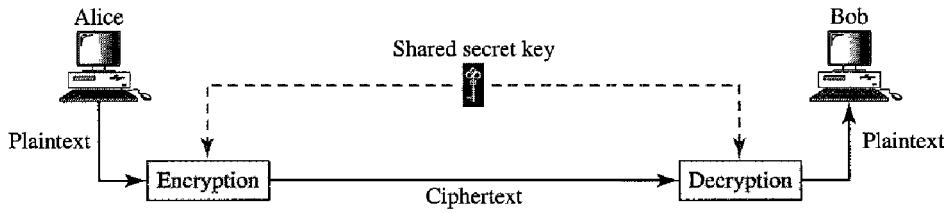
Figure 30.2 *Categories of cryptography*



Symmetric-Key Cryptography

In symmetric-key cryptography, the same key is used by both parties. The sender uses this key and an encryption algorithm to encrypt data; the receiver uses the same key and the corresponding decryption algorithm to decrypt the data (see Figure 30.3).

Figure 30.3 *Symmetric-key cryptography*

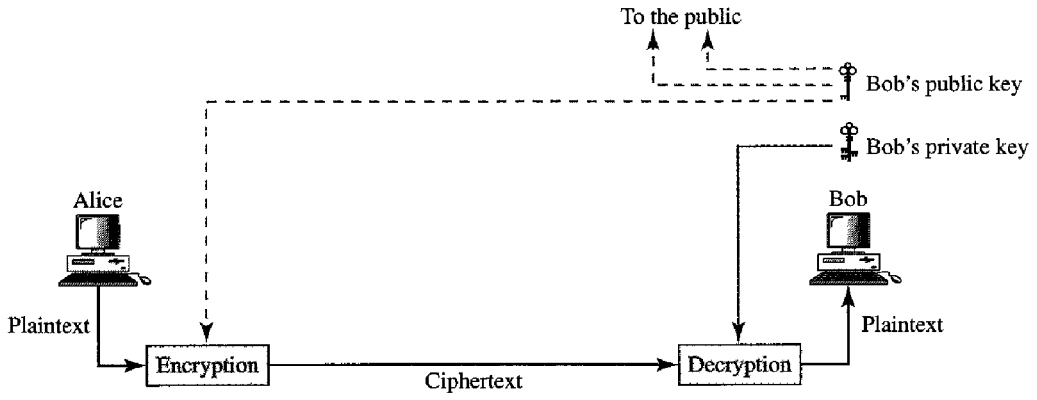


In symmetric-key cryptography, the same key is used by the sender (for encryption) and the receiver (for decryption). The key is shared.

Asymmetric-Key Cryptography

In asymmetric or public-key cryptography, there are two keys: a private key and a public key. The **private key** is kept by the receiver. The **public key** is announced to the public. In Figure 30.4, imagine Alice wants to send a message to Bob. Alice uses the public key to encrypt the message. When the message is received by Bob, the private key is used to decrypt the message.

Figure 30.4 *Asymmetric-key cryptography*

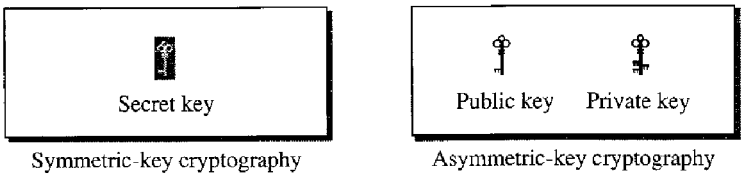


In public-key encryption/decryption, the public key that is used for encryption is different from the private key that is used for decryption. The public key is available to the public; the private key is available only to an individual.

Three Types of Keys

The reader may have noticed that we are dealing with three types of keys in cryptography: the secret key, the public key, and the private key. The first, the secret key, is the shared key used in symmetric-key cryptography. The second and the third are the public and private keys used in asymmetric-key cryptography. We will use three different icons for these keys throughout the book to distinguish one from the others, as shown in Figure 30.5.

Figure 30.5 *Keys used in cryptography*

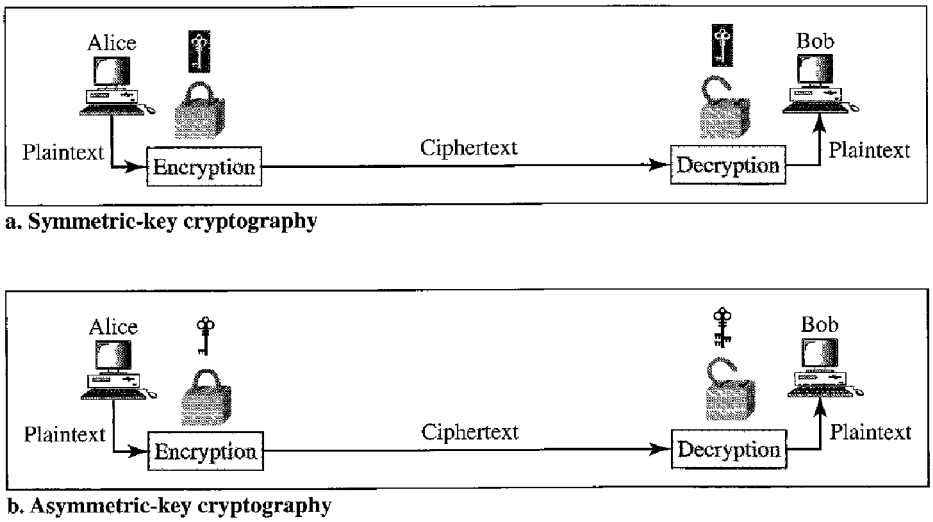


Comparison

Let us compare symmetric-key and asymmetric-key cryptography. Encryption can be thought of as electronic locking; decryption as electronic unlocking. The sender puts the message in a box and locks the box by using a key; the receiver unlocks the box with a key and takes out the message. The difference lies in the mechanism of the locking and unlocking and the type of keys used.

In symmetric-key cryptography, the same key locks and unlocks the box. In asymmetric-key cryptography, one key locks the box, but another key is needed to unlock it. Figure 30.6 shows the difference.

Figure 30.6 *Comparison between two categories of cryptography*



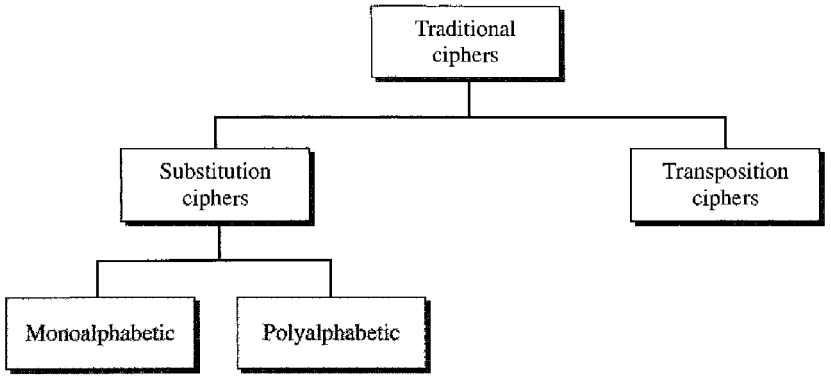
30.2 SYMMETRIC-KEY CRYPTOGRAPHY

Symmetric-key cryptography started thousands of years ago when people needed to exchange secrets (for example, in a war). We still mainly use symmetric-key cryptography in our network security. However, today's ciphers are much more complex. Let us first discuss traditional algorithms, which were character-oriented. Then we discuss the modern ones, which are bit-oriented.

Traditional Ciphers

We briefly introduce some traditional ciphers, which are character-oriented. Although these are now obsolete, the goal is to show how modern ciphers evolved from them. We can divide traditional symmetric-key ciphers into two broad categories: substitution ciphers and transposition ciphers, as shown in Figure 30.7.

Figure 30.7 *Traditional ciphers*



Substitution Cipher

A **substitution cipher** substitutes one symbol with another. If the symbols in the plaintext are alphabetic characters, we replace one character with another. For example, we can replace character A with D, and character T with Z. If the symbols are digits (0 to 9), we can replace 3 with 7, and 2 with 6. Substitution ciphers can be categorized as either monoalphabetic or polyalphabetic ciphers.

A substitution cipher replaces one symbol with another.

In a **monoalphabetic cipher,** a character (or a symbol) in the plaintext is always changed to the same character (or symbol) in the ciphertext regardless of its position in the text. For example, if the algorithm says that character A in the plaintext is changed to character D, every character A is changed to character D. In other words, the relationship between characters in the plaintext and the ciphertext is a one-to-one relationship.

In a **polyalphabetic cipher,** each occurrence of a character can have a different substitute. The relationship between a character in the plaintext to a character in the

ciphertext is a one-to-many relationship. For example, character A could be changed to D in the beginning of the text, but it could be changed to N at the middle. It is obvious that if the relationship between plaintext characters and ciphertext characters is one-to-many, the key must tell us which of the many possible characters can be chosen for encryption. To achieve this goal, we need to divide the text into groups of characters and use a set of keys. For example, we can divide the text "THISISANEASYTASK" into groups of 3 characters and then apply the encryption using a set of 3 keys. We then repeat the procedure for the next 3 characters.

Example 30.1

The following shows a plaintext and its corresponding ciphertext. Is the cipher monoalphabetic?

Plaintext: HELLO
Ciphertext: KHOOR

Solution

The cipher is probably monoalphabetic because both occurrences of L's are encrypted as O's.

Example 30.2

The following shows a plaintext and its corresponding ciphertext. Is the cipher monoalphabetic?

Plaintext: HELLO
Ciphertext: ABNZF

Solution

The cipher is not monoalphabetic because each occurrence of L is encrypted by a different character. The first L is encrypted as N; the second as Z.

Shift Cipher The simplest monoalphabetic cipher is probably the **shift cipher.** We assume that the plaintext and ciphertext consist of uppercase letters (A to Z) only. In this cipher, the encryption algorithm is "shift *key* characters down," with *key* equal to some number. The decryption algorithm is "shift *key* characters up." For example, if the key is 5, the encryption algorithm is "shift 5 characters down" (toward the end of the alphabet). The decryption algorithm is "shift 5 characters up" (toward the beginning of the alphabet). Of course, if we reach the end or beginning of the alphabet, we wrap around.

Julius Caesar used the shift cipher to communicate with his officers. For this reason, the shift cipher is sometimes referred to as the **Caesar cipher.** Caesar used a key of 3 for his communications.

The shift cipher is sometimes referred to as the Caesar cipher.

Example 30.3

Use the shift cipher with key = 15 to encrypt the message "HELLO."

Solution

We encrypt one character at a time. Each character is shifted 15 characters down. Letter H is encrypted to W. Letter E is encrypted to T. The first L is encrypted to A. The second L is also encrypted to A. And O is encrypted to D. The cipher text is WTAAD.

Example 30.4

Use the shift cipher with key = 15 to decrypt the message "WTAAD."

Solution

We decrypt one character at a time. Each character is shifted 15 characters up. Letter W is decrypted to H. Letter T is decrypted to E. The first A is decrypted to L. The second A is decrypted to L. And, finally, D is decrypted to O. The plaintext is HELLO.

Transposition Ciphers

In a **transposition cipher,** there is no substitution of characters; instead, their locations change. A character in the first position of the plaintext may appear in the tenth position of the ciphertext. A character in the eighth position may appear in the first position. In other words, a transposition cipher reorders the symbols in a block of symbols.

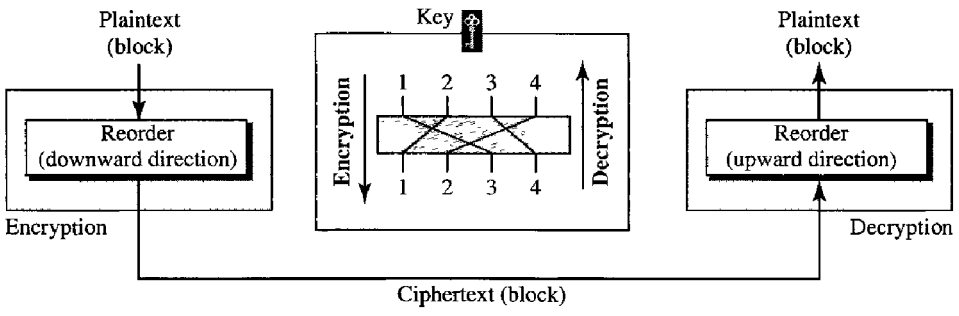
A transposition cipher reorders (permutes) symbols in a block of symbols.

Key In a transposition cipher, the key is a mapping between the position of the symbols in the plaintext and cipher text. For example, the following shows the key using a block of four characters:

Plaintext:	**2**	**4**	**1**	**3**
Ciphertext:	**1**	**2**	**3**	**4**

In encryption, we move the character at position 2 to position 1, the character at position 4 to position 2, and so on. In decryption, we do the reverse. Note that, to be more effective, the key should be long, which means encryption and decryption of long blocks of data. Figure 30.8 shows encryption and decryption for our four-character

Figure 30.8 *Transposition cipher*



block using the above key. The figure shows that the encryption and decryption use the same key. The encryption applies it from downward while decryption applies it upward.

Example 30.5

Encrypt the message "HELLO MY DEAR," using the above key.

Solution

We first remove the spaces in the message. We then divide the text into blocks of four characters. We add a bogus character Z at the end of the third block. The result is HELL OMYD EARZ. We create a three-block ciphertext ELHLMDOYAZER.

Example 30.6

Using Example 30.5, decrypt the message "ELHLMDOYAZER".

Solution

The result is HELL OMYD EARZ. After removing the bogus character and combining the characters, we get the original message "HELLO MY DEAR."

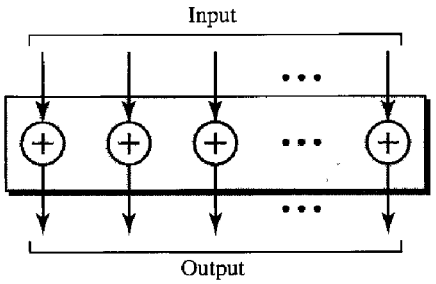
Simple Modern Ciphers

The traditional ciphers we have studied so far are character-oriented. With the advent of the computer, ciphers need to be bit-oriented. This is so because the information to be encrypted is not just text; it can also consist of numbers, graphics, audio, and video data. It is convenient to convert these types of data into a stream of bits, encrypt the stream, and then send the encrypted stream. In addition, when text is treated at the bit level, each character is replaced by 8 (or 16) bits, which means the number of symbols becomes 8 (or 16). Mingling and mangling bits provides more security than mingling and mangling characters. Modern ciphers use a different strategy than the traditional ones. A modern symmetric cipher is a combination of simple ciphers. In other words, a modern cipher uses several simple ciphers to achieve its goal. We first discuss these simple ciphers.

XOR Cipher

Modern ciphers today are normally made of a set of **simple ciphers,** which are simple predefined functions in mathematics or computer science. The first one discussed here is called the **XOR cipher** because it uses the exclusive-or operation as defined in computer science. Figure 30.9 shows an XOR cipher.

Figure 30.9 *XOR cipher*

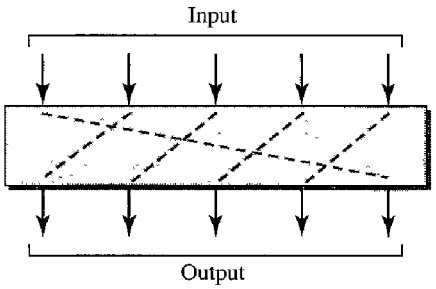


An XOR operation needs two data inputs plaintext, as the first and a key as the second. In other words, one of the inputs is the block to be the encrypted, the other input is a key; the result is the encrypted block. Note that in an XOR cipher, the size of the key, the plaintext, and the ciphertext are all the same. XOR ciphers have a very interesting property: the encryption and decryption are the same.

Rotation Cipher

Another common cipher is the **rotation cipher,** in which the input bits are rotated to the left or right. The rotation cipher can be keyed or keyless. In keyed rotation, the value of the key defines the number of rotations; in keyless rotation the number of rotations is fixed. Figure 30.10 shows an example of a rotation cipher. Note that the rotation cipher can be considered a special case of the transpositional cipher using bits instead of characters.

Figure 30.10 *Rotation cipher*



The rotation cipher has an interesting property. If the length of the original stream is N, after N rotations, we get the original input stream. This means that it is useless to apply more than $N - 1$ rotations. In other words, the number of rotations must be between 1 and $N - 1$.

The decryption algorithm for the rotation cipher uses the same key and the opposite rotation direction. If we use a right rotation in the encryption, we use a left rotation in decryption and vice versa.

Substitution Cipher: S-box

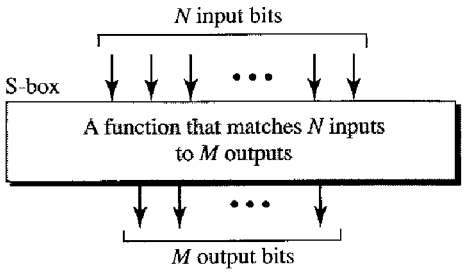
An **S-box** (substitution box) parallels the traditional substitution cipher for characters. The input to an S-box is a stream of bits with length N; the result is another stream of bits with length M. And N and M are not necessarily the same. Figure 30.11 shows an S-box.

The S-box is normally keyless and is used as an intermediate stage of encryption or decryption. The function that matches the input to the output may be defined mathematically or by a table.

Transposition Cipher: P-box

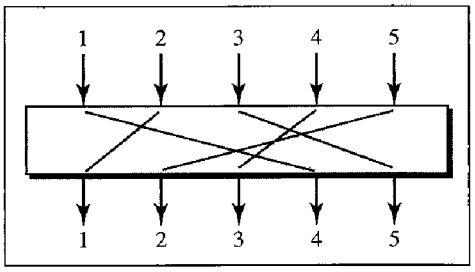
A **P-box** (permutation box) for bits parallels the traditional transposition cipher for characters. It performs a transposition at the bit level; it transposes bits. It can be implemented

Figure 30.11 *S-box*

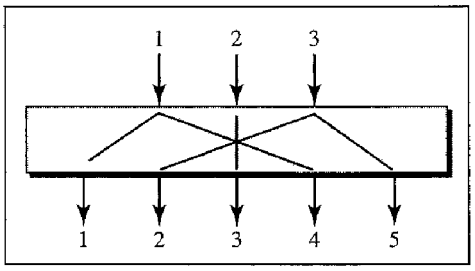


in software or hardware, but hardware is faster. P-boxes, like S-boxes, are normally keyless. We can have three types of permutations in P-boxes: the **straight permutation, expansion permutation,** and **compression permutation** as shown in Figure 30.12.

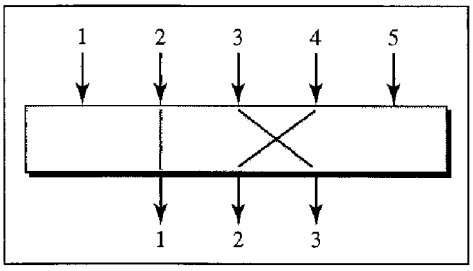
Figure 30.12 *P-boxes: straight, expansion, and compression*



a. Straight



b. Expansion



c. Compression

A straight permutation cipher or a straight P-box has the same number of inputs as outputs. In other words, if the number of inputs is N, the number of outputs is also N. In an expansion permutation cipher, the number of output ports is greater than the number of input ports. In a compression permutation cipher, the number of output ports is less than the number of input ports.

Modern Round Ciphers

The ciphers of today are called **round ciphers** because they involve multiple **rounds,** where each round is a complex cipher made up of the simple ciphers that we previously

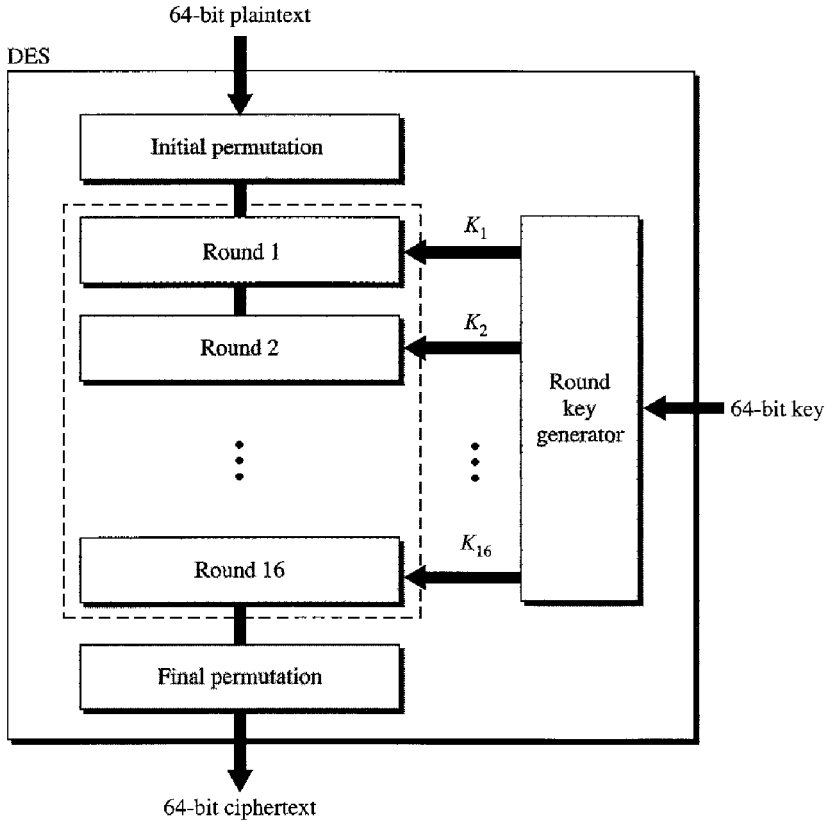
described. The key used in each round is a subset or variation of the general key called the round key. If the cipher has N rounds, a key generator produces N keys, $K_1, K_2, \ldots, K_N$, where K_1 is used in round 1, K_2 in round 2, and so on.

In this section, we introduce two modern symmetric-key ciphers: DES and AES. These ciphers are referred to as **block ciphers** because they divide the plaintext into blocks and use the same key to encrypt and decrypt the blocks. DES has been the de facto standard until recently. AES is the formal standard now.

Data Encryption Standard (DES)

One example of a complex block cipher is the **Data Encryption Standard (DES).** DES was designed by IBM and adopted by the U.S. government as the standard encryption method for nonmilitary and nonclassified use. The algorithm encrypts a 64-bit plaintext block using a 64-bit key, as shown in Figure 30.13.

Figure 30.13 *DES*

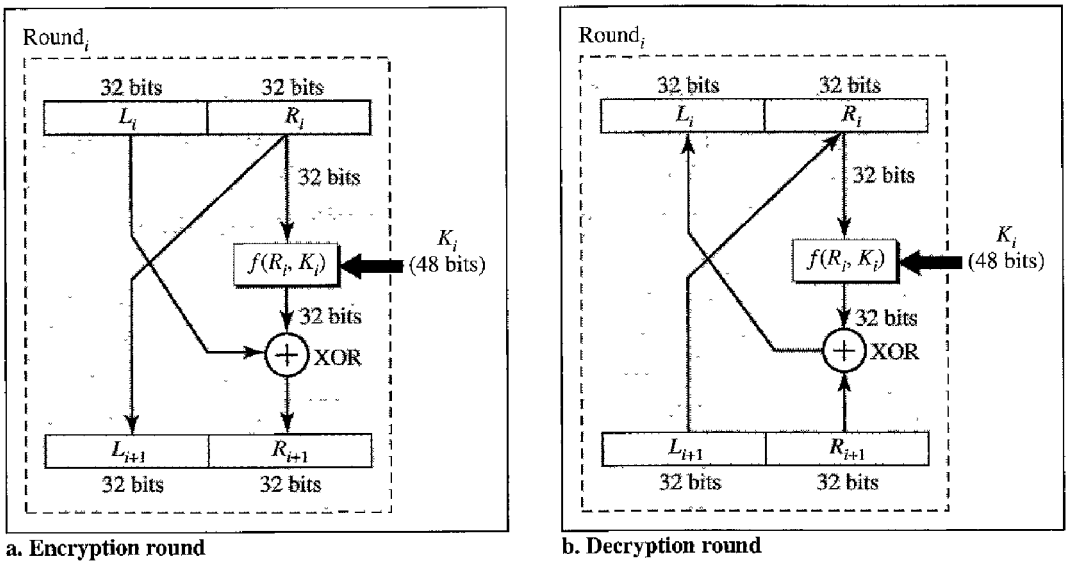


DES has two transposition blocks (P-boxes) and 16 complex round ciphers (they are repeated). Although the 16 iteration round ciphers are conceptually the same, each uses a different key derived from the original key.

The initial and final permutations are keyless straight permutations that are the inverse of each other. The permutation takes a 64-bit input and permutes them according to predefined values.

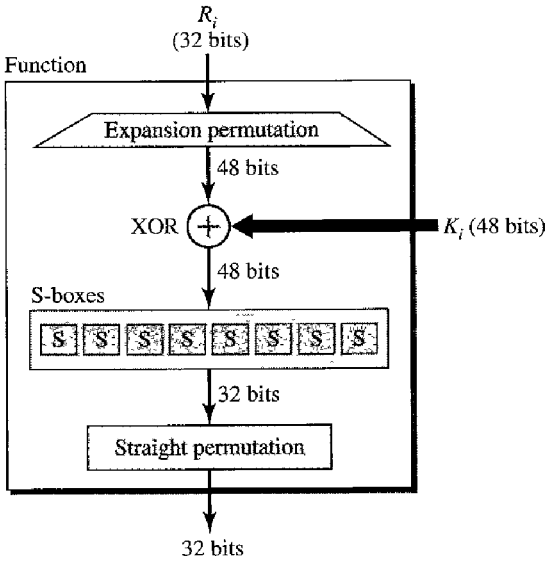
Each round of DES is a complex round cipher, as shown in Figure 30.14. Note that the structure of the encryption round ciphers is different from that of the decryption one.

Figure 30.14 *One round in DES ciphers*



DES Function The heart of DES is the **DES function.** The DES function applies a 48-bit key to the rightmost 32 bits R_i to produce a 32-bit output. This function is made up of four operations: an XOR, an expansion permutation, a group of S-boxes, and a straight permutation, as shown in Figure 30.15.

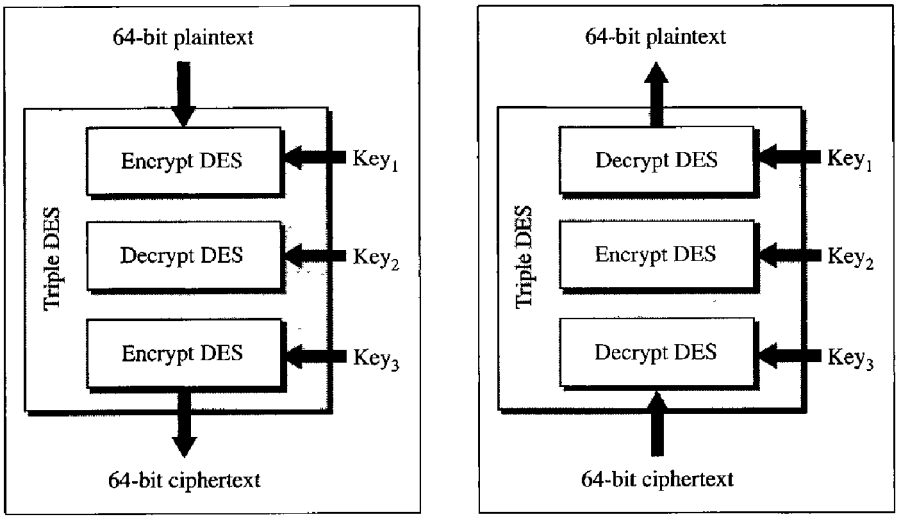
Figure 30.15 *DES function*



Triple DES

Critics of DES contend that the key is too short. To lengthen the key, **Triple DES** or 3DES has been proposed and implemented. This uses three DES blocks, as shown in Figure 30.16. Note that the encrypting block uses an encryption-decryption-encryption combination of DESs, while the decryption block uses a decryption-encryption-decryption combination. Two different versions of 3DES are in use: 3DES with two keys and 3DES with three keys. To make the key size 112 bits and at the same time protect DES from attacks such as the man-in-the-middle attack, 3DES with two keys was designed. In this version, the first and the third keys are the same ($Key_1 = Key_3$). This has the advantage in that a text encrypted by a single DES block can be decrypted by the new 3DES. We just set all keys equal to Key_1. Many algorithms use a 3DES cipher with three keys. This increases the size of the key to 168 bits.

Figure 30.16 *Triple DES*



a. Encryption Triple DES **b. Decryption Triple DES**

Advanced Encryption Standard (AES)

The **Advanced Encryption Standard (AES)** was designed because DES's key was too small. Although Triple DES (3DES) increased the key size, the process was too slow. The **National Institute of Standards and Technology (NIST)** chose the **Rijndael algorithm,** named after its two Belgian inventors, Vincent Rijmen and Joan Daemen, as the basis of AES. AES is a very complex round cipher. AES is designed with three key sizes: 128, 192, or 256 bits. Table 30.1 shows the relationship between the data block, number of rounds, and key size.

Table 30.1 *AES configuration*

Size of Data Block	*Number of Rounds*	*Key Size*
128 bits	10	128 bits
	12	192 bits
	14	256 bits

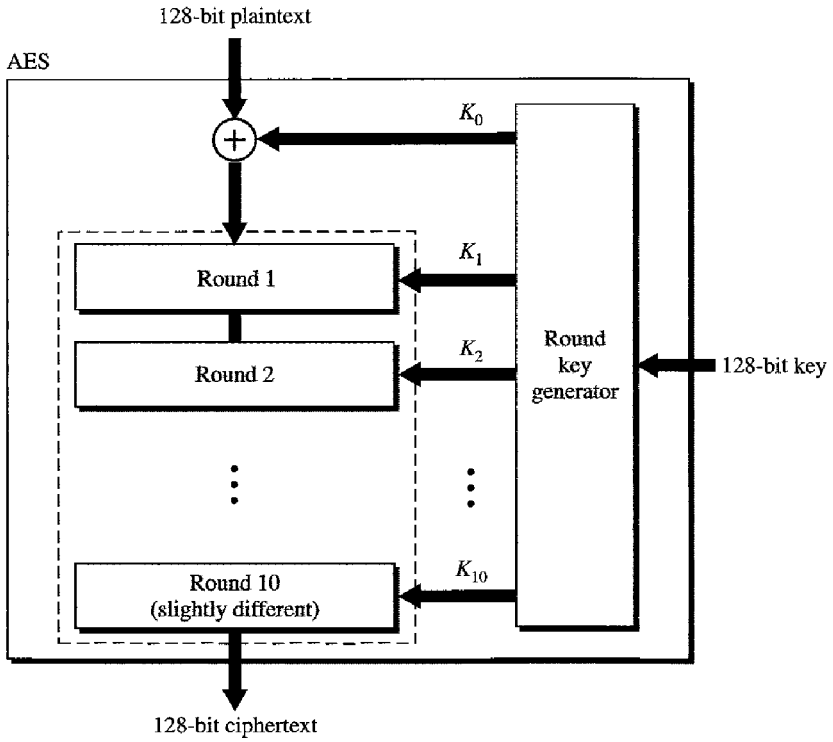
AES has three different configurations with respect to the number of rounds and key size.

In this text, we discuss just the 10-round, 128-bit key configuration. The structure and operation of the other configurations are similar. The difference lies in the key generation.

The general structure is shown in Figure 30.17. There is an initial XOR operation followed by 10 round ciphers. The last round is slightly different from the preceding rounds; it is missing one operation.

Although the 10 iteration blocks are almost identical, each uses a different key derived from the original key.

Figure 30.17 *AES*

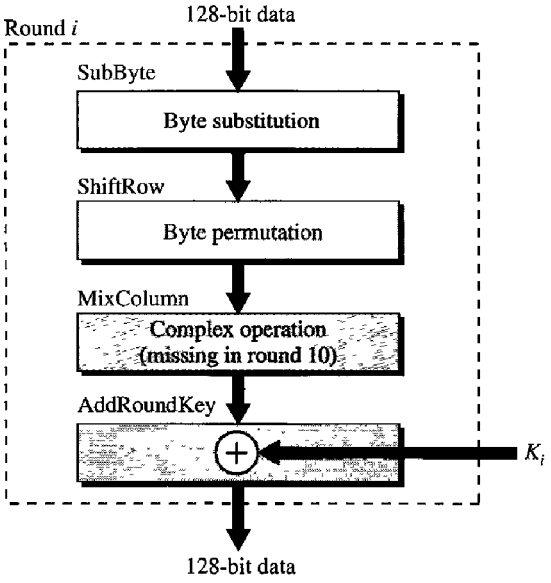


Structure of Each Round Each round of AES, except for the last, is a cipher with four operations that are invertible. The last round has only three operations. Figure 30.18 is a flowchart that shows the operations in each round. Each of the four operations used in each round uses a complex cipher; this topic is beyond the scope of this book.

Other Ciphers

During the last two decades, a few other symmetric block ciphers have been designed and used. Most of these ciphers have similar characteristics to the two ciphers we discussed in this chapter (DES and AES). The difference is usually in the size of the block or key, the number of rounds, and the functions used. The principles are the same. In order not to burden the user with the details of these ciphers, we give a brief description of each.

Figure 30.18 *Structure of each round*



IDEA The International Data Encryption Algorithm (IDEA) was developed by Xuejia Lai and James Massey. The block size is 64 and the key size is 128. It can be implemented in both hardware and software.

Blowfish Blowfish was developed by Bruce Schneier. The block size is 64 and the key size between 32 and 448.

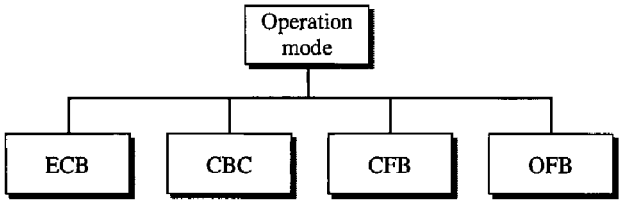
CAST-128 CAST-128 was developed by Carlisle Adams and Stafford Tavares. It is a Feistel cipher with 16 rounds and a block size of 64 bits; the key size is 128 bits.

RC5 RC5 was designed by Ron Rivest. It is a family of ciphers with different block sizes, key sizes, and numbers of rounds.

Mode of Operation

A **mode of operation** is a technique that employs the modern block ciphers such as DES and AES that we discussed earlier (see Figure 30.19).

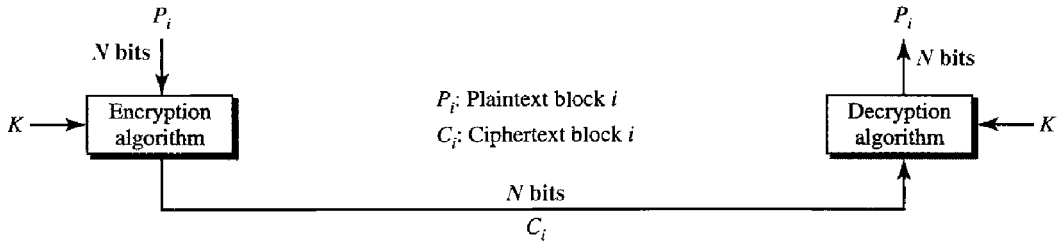
Figure 30.19 *Modes of operation for block ciphers*



Electronic Code Book

The **electronic code book (ECB) mode** is a purely block cipher technique. The plaintext is divided into blocks of N bits. The ciphertext is made of blocks of N bits. The value of N depends on the type of cipher used. Figure 30.20 shows the method.

Figure 30.20 *ECB mode*



We mention four characteristics of this mode:

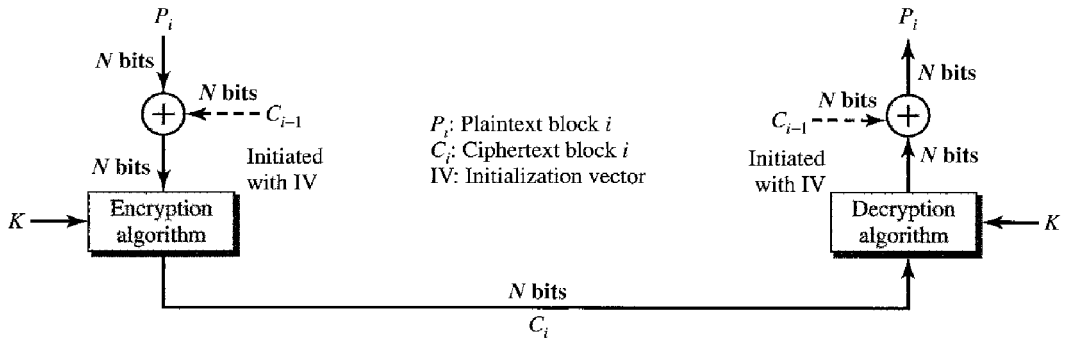
1. Because the key and the encryption/decryption algorithm are the same, equal blocks in the plaintext become equal blocks in the ciphertext. For example, if plaintext blocks 1, 5, and 9 are the same, ciphertext blocks 1, 5, and 9 are also the same. This can be a security problem; the adversary can guess that the plaintext blocks are the same if the corresponding ciphertext blocks are the same.
2. If we reorder the plaintext block, the ciphertext is also reordered.
3. Blocks are independent of each other. Each block is encrypted or decrypted independently. A problem in encryption or decryption of a block does not affect other blocks.
4. An error in one block is not propagated to other blocks. If one or more bits are corrupted during transmission, it only affects the bits in the corresponding plaintext after decryption. Other plaintext blocks are not affected. This is a real advantage if the channel is not noise-free.

Cipher Block Chaining

The **cipher block chaining (CBC) mode** tries to alleviate some of the problems in ECB by including the previous cipher block in the preparation of the current block. If the current block is i, the previous ciphertext block C_{i-1} is included in the encryption of block i. In other words, when a block is completely enciphered, the block is sent, but a copy of it is kept in a register (a place where data can be held) to be used in the encryption of the next block. The reader may wonder about the initial block. There is no ciphertext block before the first block. In this case, a phony block called the **initiation vector (IV)** is used. Both the sender and receiver agree upon a specific predetermined IV. In other words, the IV is used instead of the nonexistent C_0. Figure 30.21 shows the CBC mode.

The reader may wonder about the decryption. Does the configuration shown in the figure guarantee the correct decryption? It can be proven that it does, but we leave the proof to a textbook in network security.

Figure 30.21 *CBC mode*



The following are some characteristics of CBC.

1. Even though the key and the encryption/decryption algorithm are the same, equal blocks in the plaintext do not become equal blocks in the ciphertext. For example, if plaintext blocks 1, 5, and 9 are the same, ciphertext blocks 1, 5, and 9 will not be the same. An adversary will not be able to guess from the ciphertext that two blocks are the same.
2. Blocks are dependent on each other. Each block is encrypted or decrypted based on a previous block. A problem in encryption or decryption of a block affects other blocks.
3. The error in one block is propagated to the other blocks. If one or more bits are corrupted during the transmission, it affects the bits in the next blocks of the plaintext after decryption.

Cipher Feedback

The **cipher feedback (CFB) mode** was created for those situations in which we need to send or receive r bits of data, where r is a number different from the underlying block size of the encryption cipher used. The value of r can be 1, 4, 8, or any number of bits. Since all block ciphers work on a block of data at a time, the problem is how to encrypt just r bits. The solution is to let the cipher encrypt a block of bits and use only the first r bits as a new key (stream key) to encrypt the r bits of user data. Figure 30.22 shows the configuration.

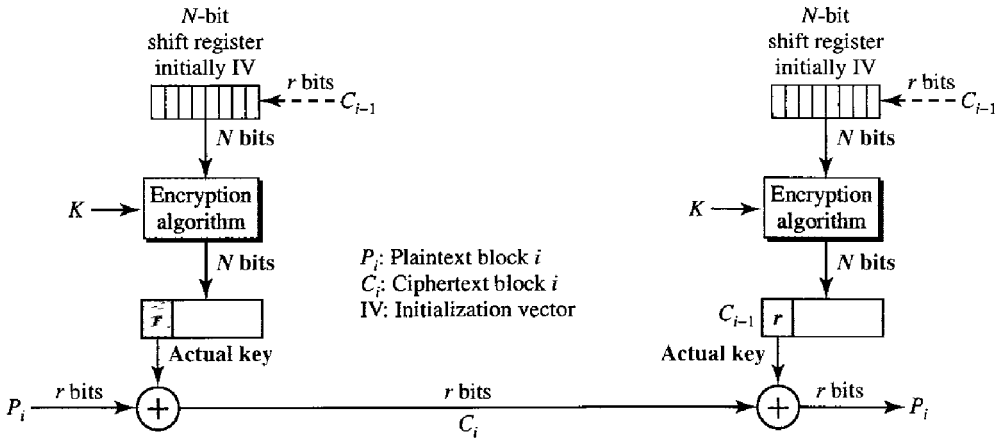
The following are some characteristics of the CFB mode:

1. If we change the IV from one encryption to another using the same plaintext, the ciphertext is different.
2. The ciphertext C_i depends on both P_i and the preceding ciphertext block.
3. Errors in one or more bits of the ciphertext block affect the next ciphertext blocks.

Output Feedback

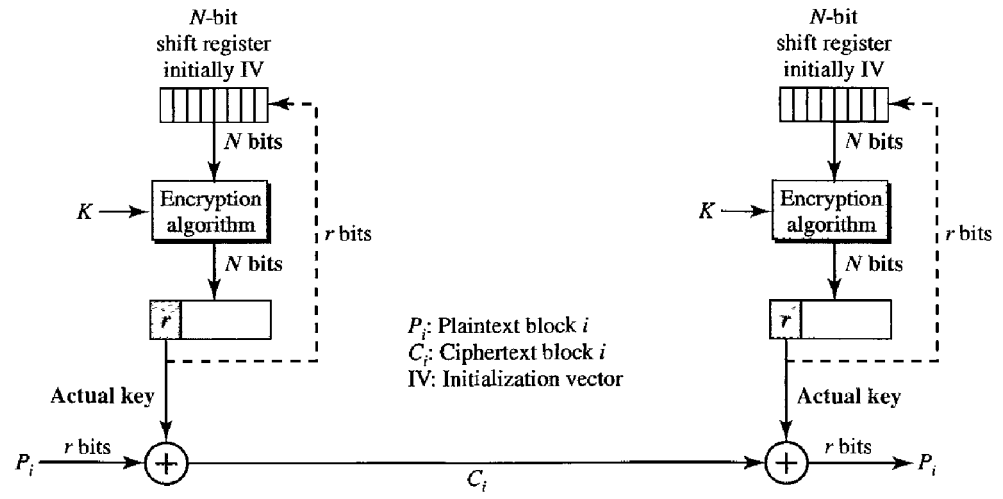
The **output feedback (OFB) mode** is very similar to the CFB mode with one difference. Each bit in the ciphertext is independent of the previous bit or bits. This avoids error

Figure 30.22 *CFB mode*



propagation. If an error occurs in transmission, it does not affect the future bits. Note that, as in CFB, both the sender and the receiver use the encryption algorithm. Note also that in OFB, block ciphers such as DES or AES can only be used to create the key stream. The feedback for creating the next bit stream comes from the previous bits of the key stream instead of the ciphertext. The ciphertext does not take part in creating the key stream. Figure 30.23 shows the OFB mode.

Figure 30.23 *OFB mode*



The following are some of the characteristics of the OFB mode.

1. If we change the IV from one encryption to another using the same plaintext, the ciphertext will be different.
2. The ciphertext C_i depends on the plaintext P_i.
3. Errors in one or more bits of the ciphertext do not affect future ciphertext blocks.

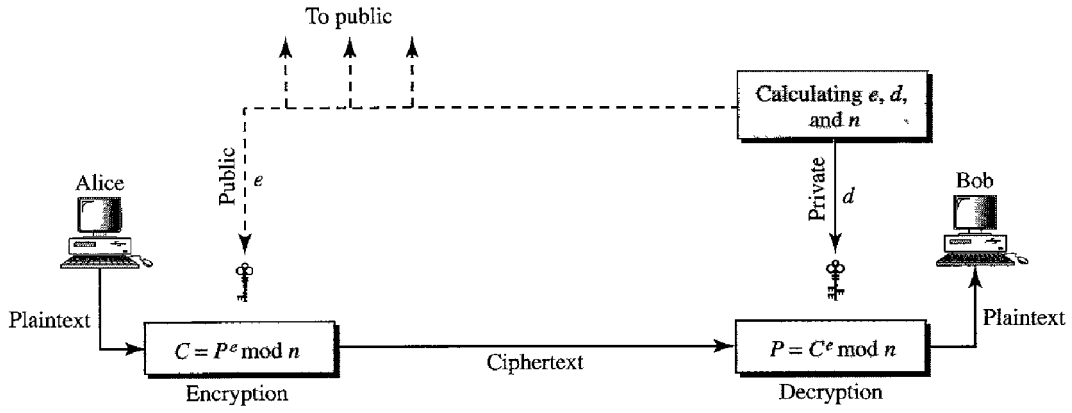
30.3 ASYMMETRIC-KEY CRYPTOGRAPHY

In the previous sections, we discussed symmetric-key cryptography. In this section we introduce asymmetric-key (public key cryptography). As we mentioned before, an asymmetric-key (or public-key) cipher uses two keys: one private and one public. We discuss two algorithms: RSA and Diffie-Hellman.

RSA

The most common public key algorithm is RSA, named for its inventors **Rivest, Shamir, and Adleman (RSA).** It uses two numbers, e and d, as the public and private keys, as shown in Figure 30.24.

Figure 30.24 *RSA*



The two keys, e and d, have a special relationship to each other, a discussion of this relationship is beyond the scope of this book. We just show how to calculate the keys without proof.

Selecting Keys

Bob use the following steps to select the private and public keys:

1. Bob chooses two very large prime numbers p and q. Remember that a prime number is one that can be divided evenly only by 1 and itself.
2. Bob multiplies the above two primes to find n, the modulus for encryption and decryption. In other words, $n = p \times q$.
3. Bob calculates another number $\phi = (p - 1) \times (q - 1)$.
4. Bob chooses a random integer e. He then calculates d so that $d \times e = 1 \bmod \phi$.
5. Bob announces e and n to the public; he keeps ϕ and d secret.

In RSA, e and n are announced to the public; d and ϕ are kept secret.

Encryption

Anyone who needs to send a message to Bob can use n and e. For example, if Alice needs to send a message to Bob, she can change the message, usually a short one, to an integer. This is the plaintext. She then calculates the ciphertext, using e and n.

$$C = P^e \pmod{n}$$

Alice sends C, the ciphertext, to Bob.

Decryption

Bob keeps ϕ and d private. When he receives the ciphertext, he uses his private key d to decrypt the message:

$$P = C^d \pmod{n}$$

Restriction

For RSA to work, the value of P must be less than the value of n. If P is a large number, the plaintext needs to be divided into blocks to make P less than n.

Example 30.7

Bob chooses 7 and 11 as p and q and calculates $n = 7 \cdot 11 = 77$. The value of $\phi = (7 - 1)(11 - 1)$ or 60. Now he chooses two keys, e and d. If he chooses e to be 13, then d is 37. Now imagine Alice sends the plaintext 5 to Bob. She uses the public key 13 to encrypt 5.

Plaintext: 5
$C = 5^{13} = 26 \bmod 77$
Ciphertext: 26

Bob receives the ciphertext 26 and uses the private key 37 to decipher the ciphertext:

Ciphertext: 26
$P = 26^{37} = 5 \bmod 77$
Plaintext: 5 **Intended message sent by Alice**

The plaintext 5 sent by Alice is received as plaintext 5 by Bob.

Example 30.8

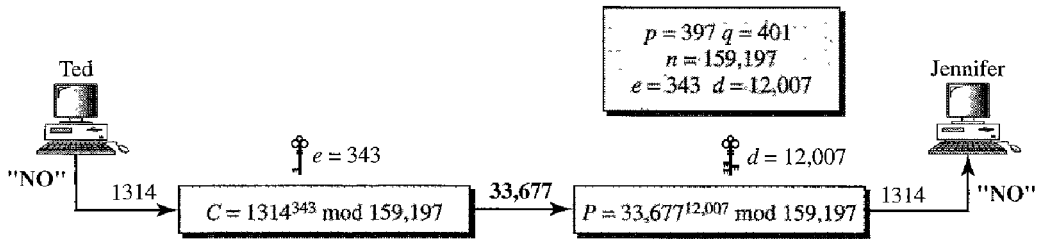
Jennifer creates a pair of keys for herself. She chooses $p = 397$ and $q = 401$. She calculates $n = 159{,}197$ and $\phi = 396 \cdot 400 = 158{,}400$. She then chooses $e = 343$ and $d = 12{,}007$. Show how Ted can send a message to Jennifer if he knows e and n.

Solution

Suppose Ted wants to send the message "NO" to Jennifer. He changes each character to a number (from 00 to 25) with each character coded as two digits. He then concatenates the two coded characters and gets a four-digit number. The plaintext is 1314. Ted then uses e and n to encrypt the message. The ciphertext is $1314^{343} = 33{,}677 \bmod 159{,}197$. Jennifer receives the message

33,677 and uses the decryption key d to decipher it as $33{,}677^{12{,}007} = 1314 \bmod 159{,}197$. Jennifer then decodes 1314 as the message "NO". Figure 30.25 shows the process.

Figure 30.25 *Example 30.8*



Example 30.9

Let us give a realistic example. We choose a 512-bit p and q. We calculate n and ϕ. We then choose e and test for relative primeness with $\phi(n)$. We calculate d. Finally, we show the results of encryption and decryption. We have written a program written in Java to do so; this type of calculation cannot be done by a calculator.

We randomly chose an integer of 512 bits. The integer p is a 159-digit number.

$\mathbf{p}$ = 9613034531358350457419158128061542790930984559499621582258315087964794045505647063849125716018034750312098666606492420191808780667421096063354219926661209

The integer q is a 160-digit number.

$\mathbf{q}$ = 1206019195723144691827679420445089600155592505463703393606179832173148214848376465921538945320917522527322683010712069560460251388714552496900035966004561 7

We calculate n. It has 309 digits.

$\mathbf{n}$ = 1159350417396761496889250986461588752377145737545414477548552613761478854083263508172768788159683251684688493006254857641112501624145523391829271625076567727274600970827141277304349605005563472745666280600999240371029914244722922157727985317270338393813346926841373276220009666766718318310883734208234443709 53

We calculate ϕ. It has 309 digits:

ϕ = 1159350417396761496889250986461588752377145737545414477548552613761478854083263508172768788159683251684688493006254857641112501624145523391829271625076567510542336084929167520344826279881175547876570139234444057169895817281960982263610754672118646121713591073586406140088851702653772772644673410662438576641 28

We choose $e = 35{,}535$. We then find d.

$$\mathbf{e} = 35535$$

$$\mathbf{d} = 5800830286003776393609366128967791759466906208965096218042286611138059385282235873170628691003002171085904433840217072986908760061153062025249598844480475682409662470814858171304632406440777048331340108509473852956450719367740611973265574242372176176746207763716420760033708533328853214470885955136670294831$$

Alice wants to send the message "THIS IS A TEST" which can be changed to a numeric value by using the 00–26 encoding scheme (26 is the *space* character).

$$\mathbf{P} = 19070818260818260026190418 19$$

The ciphertext calculated by Alice is $C = P^e$, which is

$$\mathbf{C} = 4753091236462268272063655506105451809423717960704917165232392430544529606131993285666178434183591141511974112520056829797945717360361012782188478927415660904800235071907152771859149751884658886321011483541033616578984679683867637337657774656250792805211481418440481418443081277305900469287424855916646210865 6$$

Bob can recover the plaintext from the ciphertext by using $P = C^d$, which is

$$\mathbf{P} = 19070818260818260026190418 19$$

The recovered plaintext is THIS IS A TEST after decoding.

Applications

Although RSA can be used to encrypt and decrypt actual messages, it is very slow if the message is long. RSA, therefore, is useful for short messages such as a small message digest (see Chapter 31) or a symmetric key to be used for a symmetric-key cryptosystem. In particular, we will see that RSA is used in digital signatures and other cryptosystems that often need to encrypt a small message without having access to a symmetric key. RSA is also used for authentication as we will see later.

Diffie-Hellman

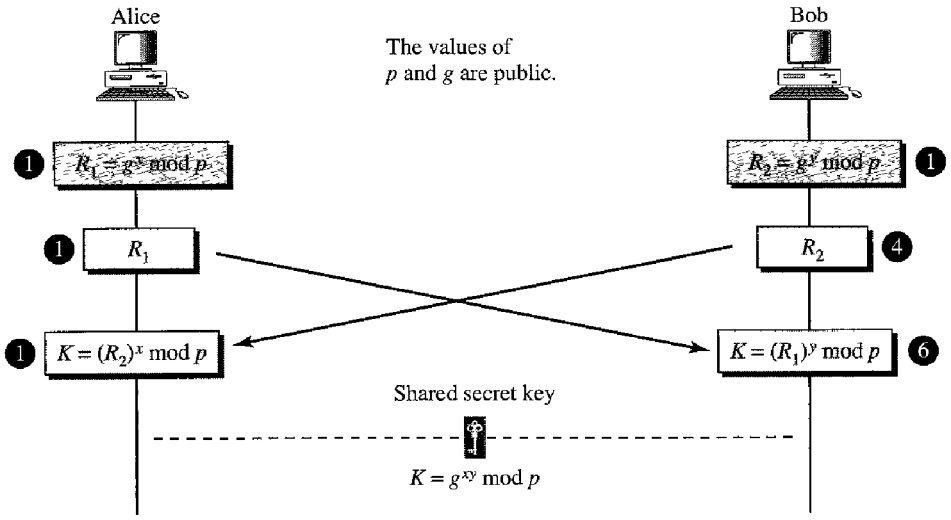
RSA is a public-key cryptosystem that is often used to encrypt and decrypt symmetric keys. Diffie-Hellman, on the other hand, was originally designed for key exchange. In the **Diffie-Hellman cryptosystem,** two parties create a symmetric **session key** to exchange data without having to remember or store the key for future use. They do not have to meet to agree on the key; it can be done through the Internet. Let us see how the protocol works when Alice and Bob need a symmetric key to communicate. Before establishing a symmetric key, the two parties need to choose two numbers p and g. The first number, p, is a

large prime number on the order of 300 decimal digits (1024 bits). The second number is a random number. These two numbers need not be confidential. They can be sent through the Internet; they can be public.

Procedure

Figure 30.26 shows the procedure. The steps are as follows:

Figure 30.26 *Diffie-Hellman method*



- **Step 1:** Alice chooses a large random number x and calculates $R_1 = g^x \bmod p$.
- **Step 2:** Bob chooses another large random number y and calculates $R_2 = g^y \bmod p$.
- **Step 3:** Alice sends R_1 to Bob. Note that Alice does not send the value of x; she sends only R_1.
- **Step 4:** Bob sends R_2 to Alice. Again, note that Bob does not send the value of y, he sends only R_2.
- **Step 5:** Alice calculates $K = (R_2)^x \bmod p$.
- **Step 6:** Bob also calculates $K = (R_1)^y \bmod p$.

The symmetric key for the session is K.

$$(g^x \bmod p)^y \bmod p = (g^y \bmod p)^x \bmod p = g^{xy} \bmod p$$

Bob has calculated $K = (R_1)^y \bmod p = (g^x \bmod p)^y \bmod p = g^{xy} \bmod p$. Alice has calculated $K = (R_2)^x \bmod p = (g^y \bmod p)^x \bmod = g^{xy} \bmod p$. Both have reached the same value without Bob knowing the value of x and without Alice knowing the value of y.

The symmetric (shared) key in the Diffie-Hellman protocol is $K = g^{xy} \bmod p$.

Example 30.10

Let us give a trivial example to make the procedure clear. Our example uses small numbers, but note that in a real situation, the numbers are very large. Assume $g = 7$ and $p = 23$. The steps are as follows:

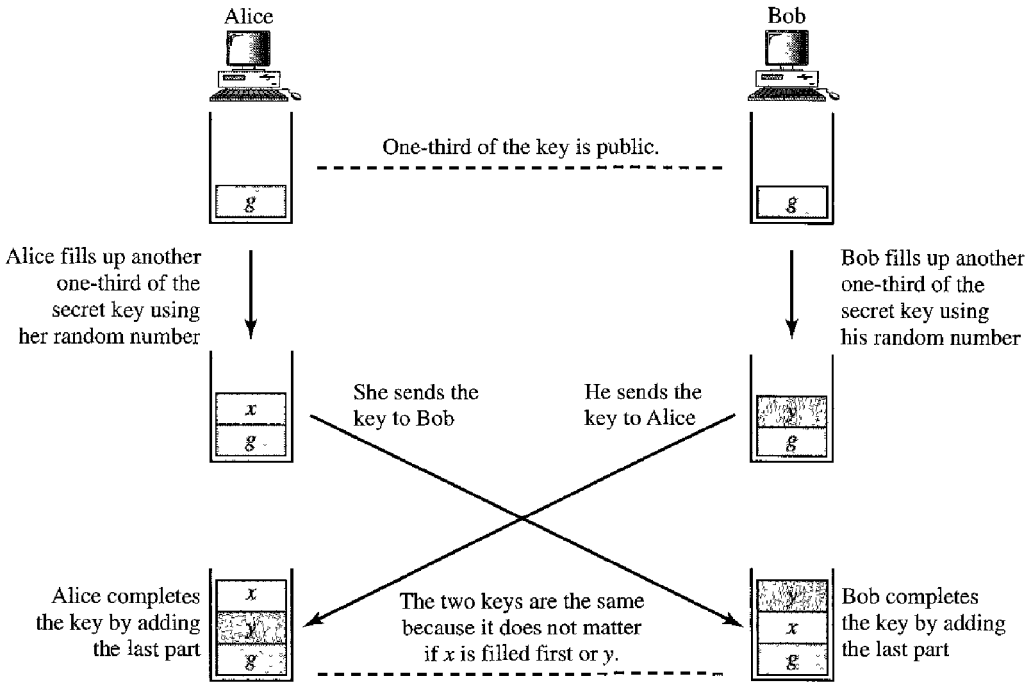
1. Alice chooses $x = 3$ and calculates $R_1 = 7^3 \bmod 23 = 21$.
2. Bob chooses $y = 6$ and calculates $R_2 = 7^6 \bmod 23 = 4$.
3. Alice sends the number 21 to Bob.
4. Bob sends the number 4 to Alice.
5. Alice calculates the symmetric key $K = 4^3 \bmod 23 = 18$.
6. Bob calculates the symmetric key $K = 21^6 \bmod 23 = 18$.

The value of K is the same for both Alice and Bob; $g^{xy} \bmod p = 7^{18} \bmod 23 = 18$.

Idea of Diffie-Hellman

The Diffie-Hellman concept, shown in Figure 30.27, is simple but elegant. We can think of the secret key between Alice and Bob as made of three parts: g, x, and y. The first part is public. Everyone knows one-third of the key; g is a public value. The other two parts must be added by Alice and Bob. Each adds one part. Alice adds x as the second part for Bob; Bob adds y as the second part for Alice. When Alice receives the two-thirds completed key from Bob, she adds the last part, her x, to complete the key. When Bob receives the two-thirds completed key from Alice, he adds the last part, his y, to complete the key. Note that although the key in Alice's hand consists of g-y-x and the key in Bob's hand is g-x-y, these two keys are the same because $g^{xy} = g^{yx}$.

Figure 30.27 *Diffie-Hellman idea*



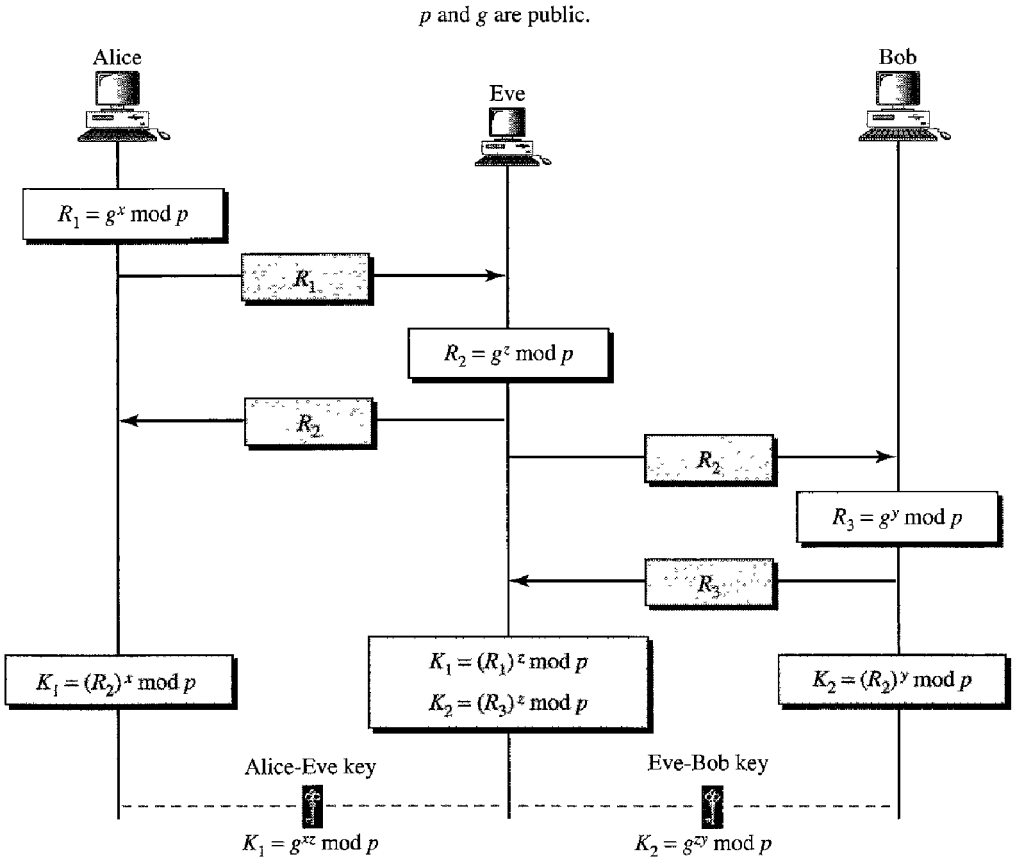
Note also that although the two keys are the same, Alice cannot find the value y used by Bob because the calculation is done in modulo p; Alice receives $g^y \bmod p$ from Bob, not g^y.

Man-in-the-Middle Attack

Diffie-Hellman is a very sophisticated symmetric-key creation algorithm. If x and y are very large numbers, it is extremely difficult for Eve to find the key, knowing only p and g. An intruder needs to determine x and y if R_1 and R_2 are intercepted. But finding x from R_1 and y from R_2 are two difficult tasks. Even a sophisticated computer would need perhaps years to find the key by trying different numbers. In addition, Alice and Bob will change the key the next time they need to communicate.

However, the protocol does have a weakness. Eve does not have to find the value of x and y to attack the protocol. She can fool Alice and Bob by creating two keys: one between herself and Alice and another between herself and Bob. Figure 30.28 shows the situation.

Figure 30.28 *Man-in-the-middle attack*



The following can happen:

1. Alice chooses x, calculates $R_1 = g^x \bmod p$, and sends R_1 to Bob.
2. Eve, the intruder, intercepts R_1. She chooses z, calculates $R_2 = g^z \bmod p$, and sends R_2 to both Alice and Bob.

3. Bob chooses y, calculates $R_3 = g^y \bmod p$, and sends R_3 to Alice; R_3 is intercepted by Eve and never reaches Alice.
4. Alice and Eve calculate $K_1 = g^{xz} \bmod p$, which becomes a shared key between Alice and Eve. Alice, however, thinks that it is a key shared between Bob and herself.
5. Eve and Bob calculate $K_2 = g^{zy} \bmod p$, which becomes a shared key between Eve and Bob. Bob, however, thinks that it is a key shared between Alice and himself.

In other words, two keys, instead of one, are created: one between Alice and Eve and one between Eve and Bob. When Alice sends data to Bob encrypted with K_1 (shared by Alice and Eve), it can be deciphered and read by Eve. Eve can send the message to Bob encrypted by K_2 (shared key between Eve and Bob); or she can even change the message or send a totally new message. Bob is fooled into believing that the message has come from Alice. A similar scenario can happen to Alice in the other direction.

This situation is called a **man-in-the-middle attack** because Eve comes in between and intercepts R_1, sent by Alice to Bob, and R_3, sent by Bob to Alice. It is also known as a **bucket brigade attack** because it resembles a short line of volunteers passing a bucket of water from person to person.

Authentication

The man-in-the-middle attack can be avoided if Bob and Alice first authenticate each other. In other words, the exchange key process can be combined with an authentication scheme to prevent a man-in-the-middle attack. We discuss authentication in Chapter 31.

30.4 RECOMMENDED READING

For more details about subjects discussed in this chapter, we recommend the following books. The items in brackets [. . .] refer to the reference list at the end of the text.

Books

Cryptography can be found in many books dedicated to the subject such as [Bar02], [Gar01], [Sti02], [Mao04], [MOV97], and [Sch96].

30.5 KEY TERMS

Advanced Encryption Standard (AES)
block cipher
bucket brigade attack
Caesar cipher
cipher block chaining (CBC) mode
cipher feedback (CFB) mode
ciphertext
compression permutation
cryptography
Data Encryption Standard (DES)
decryption
decryption algorithm
DES function
Diffie-Hellman cryptosystem
electronic code book (ECB) mode
encryption

encryption algorithm
expansion permutation
initiation vector (IV)
key
man-in-the-middle attack
mode of operation
monoalphabetic cipher
National Institute of Standards and Technology (NIST)
output feedback (OFB) mode
P-box
plaintext
polyalphabetic cipher
private key
public key
Rijndael algorithm
Rivest, Shamir, Adleman (RSA)
rotation cipher
round
round cipher
S-box
secret key
session key
shift cipher
simple cipher
straight permutation
substitution cipher
transposition cipher
Triple DES
XOR cipher

30.6 SUMMARY

- Cryptography is the science and art of transforming messages to make them secure and immune to attacks.
- The plaintext is the original message before transformation; the ciphertext is the message after transformation.
- An encryption algorithm transforms plaintext to ciphertext; a decryption algorithm transforms ciphertext to plaintext.
- A combination of an encryption algorithm and a decryption algorithm is called a cipher.
- The key is a number or a set of numbers on which the cipher operates.
- We can divide all ciphers into two broad categories: symmetric-key ciphers and asymmetric-key ciphers.
- In a symmetric-key cipher, the same key is used by both the sender and receiver. The key is called the secret key.
- In an asymmetric-key cipher, a pair of keys is used. The sender uses the public key; the receiver uses the private key.
- A substitution cipher replaces one character with another character.
- Substitution ciphers can be categorized into two broad categories: monoalphabetic and polyalphabetic.
- The shift cipher is the simplest monoalphabetic cipher. It uses modular arithmetic with a modulus of 26. The Caesar cipher is a shift cipher that has a key of 3.
- The transposition cipher reorders the plaintext characters to create a ciphertext.
- An XOR cipher is the simplest cipher which is self-invertible.
- A rotation cipher is an invertible cipher.

- An S-box is a keyless substitution cipher with N inputs and M outputs that uses a formula to define the relationship between the input stream and the output stream.
- A P-box is a keyless transposition cipher with N inputs and M outputs that uses a table to define the relationship between the input stream and the output stream. A P-box is invertible only if the numbers of inputs and outputs are the same. A P-box can use a straight permutation, a compression permutation, or an expansion permutation.
- A modern cipher is usually a round cipher; each round is a complex cipher made of a combination of different simple ciphers.
- DES is a symmetric-key method adopted by the U.S. government. DES has an initial and final permutation block and 16 rounds.
- The heart of DES is the DES function. The DES function has four components: an expansion permutation, an XOR operation, S-boxes, and a straight permutation.
- DES uses a key generator to generate sixteen 48-bit round keys.
- Triple DES was designed to increase the size of the DES key (effectively 56 bits) for better security.
- AES is a round cipher based on the Rijndael algorithm that uses a 128-bit block of data. AES has three different configurations: 10 rounds with a key size of 128 bits, 12 rounds with a key size of 192 bits, and 14 rounds with a key size of 256 bits.
- Mode of operation refers to techniques that deploy the ciphers such as DES or AES. Four common modes of operation are ECB, CBC, CBF, and OFB. ECB and CBC are block ciphers; CBF and OFB are stream ciphers.
- One commonly used public-key cryptography method is the RSA algorithm, invented by Rivest, Shamir, and Adleman.
- RSA chooses n to be the product of two primes p and q.
- The Diffie-Hellman method provides a one-time session key for two parties.
- The man-in-the-middle attack can endanger the security of the Diffie-Hellman method if two parties are not authenticated to each other.

30.7 PRACTICE SET

Review Questions

1. In symmetric-key cryptography, how many keys are needed if Alice and Bob want to communicate with each other?
2. In symmetric-key cryptography, can Alice use the same key to communicate with both Bob and John? Explain your answer.
3. In symmetric-key cryptography, if every person in a group of 10 people needs to communicate with every other person in another group of 10 people, how many secret keys are needed?
4. In symmetric-key cryptography, if every person in a group of 10 people needs to communicate with every other person in the group, how many secret keys are needed?
5. Repeat Question 1 for asymmetric-key cryptography.

6. Repeat Question 2 for asymmetric-key cryptography.
7. Repeat Question 3 for asymmetric-key cryptography.
8. Repeat Question 4 for asymmetric-key cryptography.

Exercises

9. In symmetric-key cryptography, how do you think two persons can establish a secret key between themselves?
10. In asymmetric-key cryptography, how do you think two persons can establish two pairs of keys between themselves?
11. Encrypt the message "THIS IS AN EXERCISE" using a shift cipher with a key of 20. Ignore the space between words. Decrypt the message to get the original plaintext.
12. Can we use monoalphabetic substitution if our symbols are just 0 and 1? Is it a good idea?
13. Can we use polyalphabetic substitution if our symbols are just 0 and 1? Is it a good idea?
14. Encrypt "INTERNET" using a transposition cipher with the following key:

3	**5**	**2**	**1**	**4**
1	**2**	**3**	**4**	**5**

15. Rotate 111001 three bits to the right.
16. Rotate 100111 three bits to the left.
17. A 6-by-2 S-box adds the bits at the odd-numbered positions (1, 3, 5, . . .) to get the right bit of the output and adds the bits at the even-numbered positions (2, 4, 6, . . .) to get the left bit of the output. If the input is 110010, what is the output? If the input is 101101, what is the output? Assume the rightmost bit is bit 1.
18. What are all the possible number combinations of inputs in a 6-by-2 S-box? What is the possible number of outputs?
19. The leftmost bit of a 4-by-3 S-box rotates the other 3 bits. If the leftmost bit is 0, the 3 other bits are rotated to the right 1 bit. If the leftmost bit is 1, the 3 other bits are rotated to the left 1 bit. If the input is 1011, what is the output? If the input is 0110, what is the output?
20. A P-box uses the following table for encryption. Show the box and connect the input to the output.

4	**2**	**3**	**1**
	1	**2**	

Is the P-box straight, compression, or expansion?

21. In RSA, given two prime numbers $p = 19$ and $q = 23$, find n and ϕ. Choose $e = 5$ and try to find d, such that e and d meet the criteria.
22. To understand the security of the RSA algorithm, find d if you know that $e = 17$ and $n = 187$. This exercise proves how easy is for Eve to break the secret if n is small.

23. For the RSA algorithm with a large n, explain why Bob can calculate d from n, but Eve cannot.
24. Using $e = 13$, $d = 37$, and $n = 77$ in the RSA algorithm, encrypt the message "FINE" using the values of 00 to 25 for letters A to Z. For simplicity, do the encryption and decryption character by character.
25. Why can't Bob choose 1 as the public key e in RSA?
26. What is the danger in choosing 2 as the public key e in RSA?
27. Eve uses RSA to send a message to Bob, using Bob's public key. Later, at a cocktail party, Eve sees Bob and asks him if the message has arrived and Bob confirms it. After a few drinks, Eve asks Bob, "What was the ciphertext?" Bob gives the value of the ciphertext to Eve. Can this endanger the security of Bob's private key? Explain your answer.
28. What is the value of the symmetric key in the Diffie-Hellman protocol if $g = 7$, $p = 23$, $x = 2$, and $y = 5$?
29. In the Diffie-Hellman protocol, what happens if x and y have the same value? That is, Alice and Bob have accidentally chosen the same number. Are the values of R_1 and R_2 the same? Are the values of the session keys calculated by Alice and Bob the same? Use an example to prove your claims.

Research Activities

30. Another asymmetric-key algorithm is called ElGamal. Do some research and find out some information about this algorithm. What is the difference between RSA and ElGamal?
31. Another asymmetric-key algorithm is based on elliptic curves. If you are familiar with elliptic curves, do some research and find the algorithms based on elliptic curves.
32. To make Diffie-Helman algorithm more robust, one uses cookies. Do some research and find out about the use of cookies in the Diffie-Helman algorithm.

CHAPTER 31

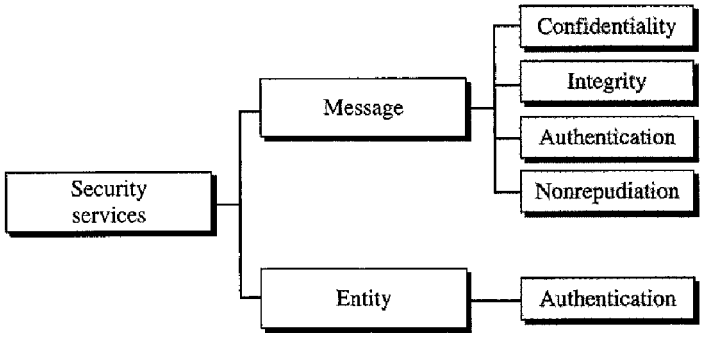
Network Security

In Chapter 30, we introduced the science of cryptography. Cryptography has several applications in network security. In this chapter, we first introduce the security services we typically expect in a network. We then show how these services can be provided using cryptography. At the end of the chapter, we also touch on the issue of distributing symmetric and asymmetric keys. The chapter provides the background necessary for Chapter 32, where we discuss security in the Internet.

31.1 SECURITY SERVICES

Network security can provide one of the five services as shown in Figure 31.1. Four of these services are related to the message exchanged using the network: message confidentiality, integrity, authentication, and nonrepudiation. The fifth service provides entity authentication or identification.

Figure 31.1 *Security services related to the message or entity*



Message Confidentiality

Message confidentiality or privacy means that the sender and the receiver expect confidentiality. The transmitted message must make sense to only the intended receiver. To all others, the message must be garbage. When a customer communicates with her bank, she expects that the communication is totally confidential.

Message Integrity

Message integrity means that the data must arrive at the receiver exactly as they were sent. There must be no changes during the transmission, neither accidentally nor maliciously. As more and more monetary exchanges occur over the Internet, integrity is crucial. For example, it would be disastrous if a request for transferring $100 changed to a request for $10,000 or $100,000. The integrity of the message must be preserved in a secure communication.

Message Authentication

Message authentication is a service beyond message integrity. In message authentication the receiver needs to be sure of the sender's identity and that an imposter has not sent the message.

Message Nonrepudiation

Message nonrepudiation means that a sender must not be able to deny sending a message that he or she, in fact, did send. The burden of proof falls on the receiver. For example, when a customer sends a message to transfer money from one account to another, the bank must have proof that the customer actually requested this transaction.

Entity Authentication

In entity authentication (or user identification) the entity or user is verified prior to access to the system resources (files, for example). For example, a student who needs to access her university resources needs to be authenticated during the logging process. This is to protect the interests of the university and the student.

31.2 MESSAGE CONFIDENTIALITY

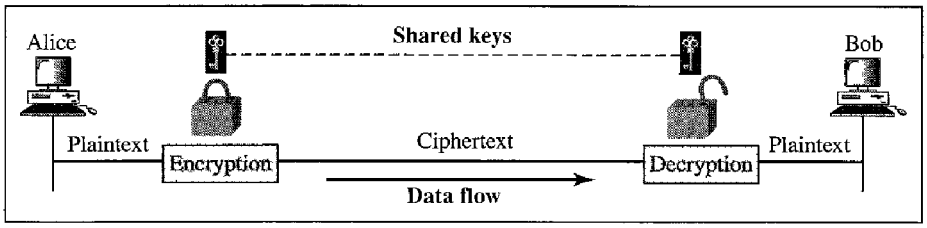
The concept of how to achieve message confidentiality or privacy has not changed for thousands of years. The message must be encrypted at the sender site and decrypted at the receiver site. That is, the message must be rendered unintelligible to unauthorized parties. A good privacy technique guarantees to some extent that a potential intruder (eavesdropper) cannot understand the contents of the message. As we discussed in Chapter 30, this can be done using either symmetric-key cryptography or asymmetric-key cryptography. We review both.

Confidentiality with Symmetric-Key Cryptography

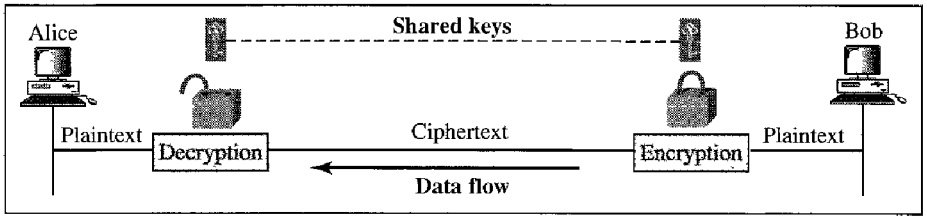
Although modern symmetric-key algorithms are more complex than the ones used through the long history of the secret writing, the principle is the same. To provide confidentiality with symmetric-key cryptography, a sender and a receiver need to share a secret key. In the past when data exchange was between two specific persons (for example, two friends or a ruler and her army chief), it was possible to personally exchange the secret keys. Today's communication does not often provide this opportunity. A person residing in the United States cannot meet and exchange a secret key with a person living in China. Furthermore, the communication is between millions of people, not just a few.

To be able to use symmetric-key cryptography, we need to find a solution to the key sharing. This can be done using a **session key.** A session key is one that is used only for the duration of one session. The session key itself is exchanged using asymmetric-key cryptography as we will see later. Figure 31.2 shows the use of a session symmetric key for sending confidential messages from Alice to Bob and vice versa. Note that the nature of the symmetric key allows the communication to be carried on in both directions although it is not recommended today. Using two different keys is more secure, because if one key is compromised, the communication is still confidential in the other direction.

Figure 31.2 *Message confidentiality using symmetric keys in two directions*



a. A shared secret key can be used in Alice-Bob communication



b. A different shared secret key is recommended in Bob-Alice communication

The reason symmetric-key cryptography is still the dominant method for confidentiality of the message is its efficiency. For a long message, symmetric-key cryptography is much more efficient than asymmetric-key cryptography.

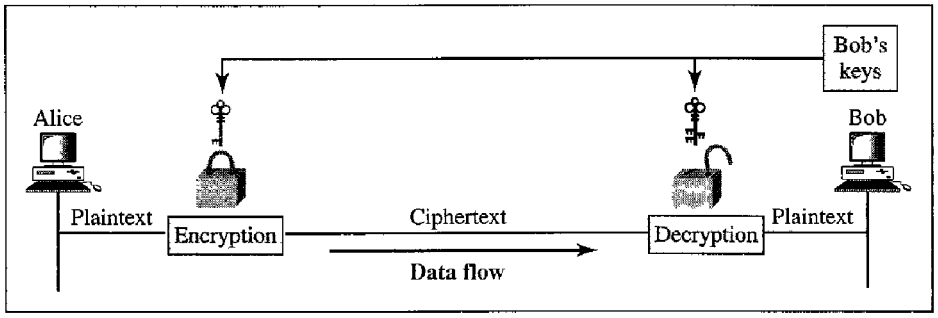
Confidentiality with Asymmetric-Key Cryptography

The problem we mentioned about key exchange in symmetric-key cryptography for privacy culminated in the creation of asymmetric-key cryptography. Here, there is no key sharing; there is a public announcement. Bob creates two keys: one private and one

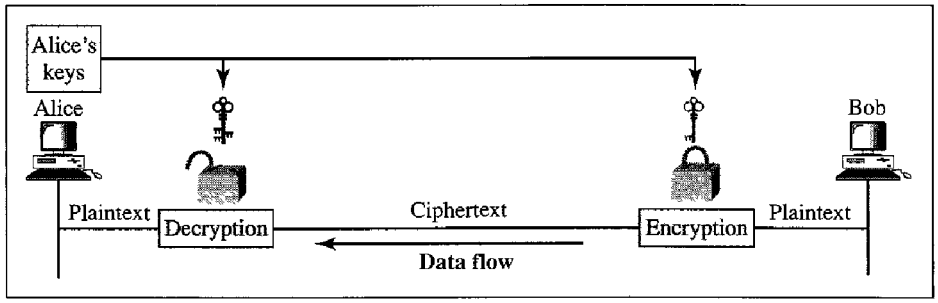
public. He keeps the private key for decryption; he publicly announces the public key to the world. The public key is used only for encryption; the private key is used only for decryption. The public key locks the message; the private key unlocks it.

For a two-way communication between Alice and Bob, two pairs of keys are needed. When Alice sends a message to Bob, she uses Bob's pair; when Bob sends a message to Alice, he uses Alice's pair as shown in Figure 31.3.

Figure 31.3 *Message confidentiality using asymmetric keys*



a. Bob's keys are used in Alice-Bob communication



b. Alice's keys are used in Bob-Alice communication

Confidentiality with asymmetric-key cryptosystem has its own problems. First, the method is based on long mathematical calculations using long keys. This means that this system is very inefficient for long messages; it should be applied only to short messages. Second, the sender of the message still needs to be certain about the public key of the receiver. For example, in Alice-Bob communication, Alice needs to be sure that Bob's public key is genuine; Eve may have announced her public key in the name of Bob. A system of trust is needed, as we will see later in the chapter.

31.3 MESSAGE INTEGRITY

Encryption and decryption provide secrecy, or confidentiality, but not **integrity.** However, on occasion we may not even need secrecy, but instead must have integrity. For example, Alice may write a will to distribute her estate upon her death. The will does not need to be encrypted. After her death, anyone can examine the will. The integrity of the will, however, needs to be preserved. Alice does not want the contents of the will to

be changed. As another example, suppose Alice sends a message instructing her banker, Bob, to pay Eve for consulting work. The message does not need to be hidden from Eve because she already knows she is to be paid. However, the message does need to be safe from any tampering, especially by Eve.

Document and Fingerprint

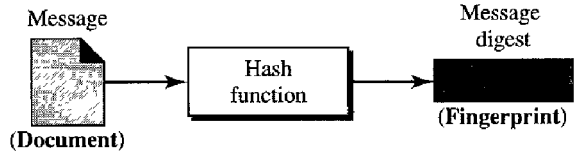
One way to preserve the integrity of a document is through the use of a **fingerprint.** If Alice needs to be sure that the contents of her document will not be illegally changed, she can put her fingerprint at the bottom of the document. Eve cannot modify the contents of this document or create a false document because she cannot forge Alice's fingerprint. To ensure that the document has not been changed, Alice's fingerprint on the document can be compared to Alice's fingerprint on file. If they are not the same, the document is not from Alice.

To preserve the integrity of a document, both the document and the fingerprint are needed.

Message and Message Digest

The electronic equivalent of the document and fingerprint pair is the **message** and **message digest** pair. To preserve the integrity of a message, the message is passed through an algorithm called a **hash function.** The hash function creates a compressed image of the message that can be used as a fingerprint. Figure 31.4 shows the message, hash function, and the message digest.

Figure 31.4 *Message and message digest*



Difference

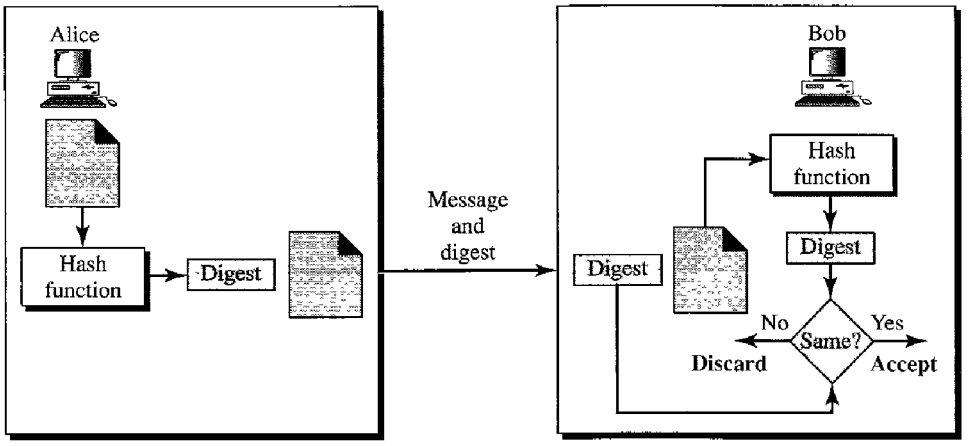
The two pairs document/fingerprint and message/message digest are similar, with some differences. The document and fingerprint are physically linked together; also, neither needs to be kept secret. The message and message digest can be unlinked (or sent) separately and, most importantly, the message digest needs to be kept secret. The message digest is either kept secret in a safe place or encrypted if we need to send it through a communications channel.

The message digest needs to be kept secret.

Creating and Checking the Digest

The message digest is created at the sender site and is sent with the message to the receiver. To check the integrity of a message, or document, the receiver creates the hash function again and compares the new message digest with the one received. If both are the same, the receiver is sure that the original message has not been changed. Of course, we are assuming that the digest has been sent secretly. Figure 31.5 shows the idea.

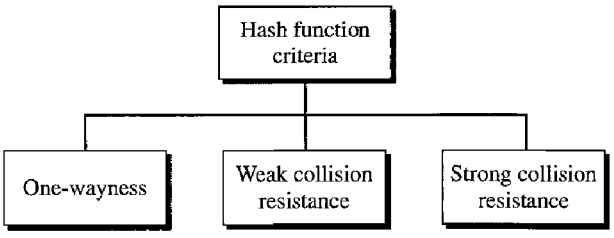
Figure 31.5 *Checking integrity*



Hash Function Criteria

To be eligible for a hash, a function needs to meet three criteria: one-wayness, resistance to weak collision, and resistance to strong collision as shown in Figure 31.6.

Figure 31.6 *Criteria of a hash function*



One-wayness

A hash function must have **one-wayness;** a message digest is created by a one-way hashing function. We must not be able to recreate the message from the digest. Sometimes it is difficult to make a hash function 100 percent one-way; the criteria state that it must be extremely difficult or impossible to create the message if the message digest is given. This is similar to the document/fingerprint case. No one can make a document from a fingerprint.

Example 31.1

Can we use a conventional lossless compression method as a hashing function?

Solution

We cannot. A lossless compression method creates a compressed message that is reversible. You can uncompress the compressed message to get the original one.

Example 31.2

Can we use a checksum method as a hashing function?

Solution

We can. A checksum function is not reversible; it meets the first criterion. However, it does not meet the other criteria.

Weak Collision Resistance

The second criterion, **weak collision** resistance, ensures that a message cannot easily be forged. If Alice creates a message and a digest and sends both to Bob, this criterion ensures that Eve cannot easily create another message that hashes exactly to the same digest. In other words, given a specific message and its digest, it is impossible (or at least very difficult) to create another message with the same digest.

When two messages create the same digest, we say there is a collision. In a week collision, given a message digest, it is very unlikely that someone can create a message with exactly the same digest. A hash function must have weak collision resistance.

Strong Collision Resistance

The third criterion, **strong collision** resistance, ensures that we cannot find two messages that hash to the same digest. This criterion is needed to ensure that Alice, the sender of the message, cannot cause problems by forging a message. If Alice can create two messages that hash to the same digest, she can deny sending the first to Bob and claim that she sent only the second.

This type of collision is called strong because the probability of collision is higher than in the previous case. An adversary can create two messages that hash to the same digest. For example, if the number of bits in the message digest is small, it is likely Alice can create two different messages with the same message digest. She can send the first to Bob and keep the second for herself. Alice can later say that the second was the original agreed-upon document and not the first.

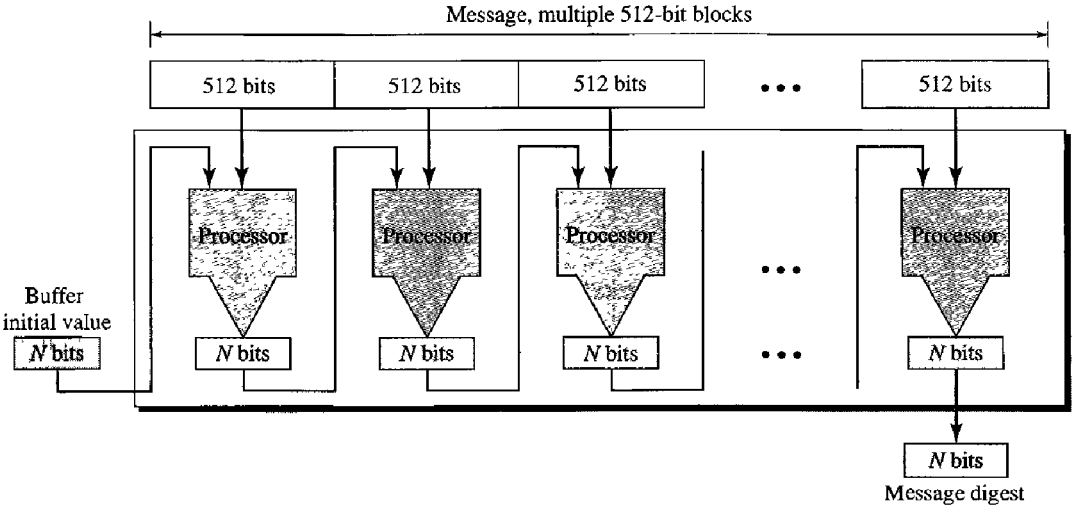
Suppose two different wills can be created that hash to the same digest. When the time comes for the execution of the will, the second will is presented to the heirs. Since the digest matches both wills, the substitution is successful.

Hash Algorithms: SHA-1

While many hash algorithms have been designed, the most common is SHA-1. **SHA-1 (Secure Hash Algorithm 1)** is a revised version of SHA designed by the National Institute of Standards and Technology (NIST). It was published as a Federal Information Processing Standard (FIPS).

A very interesting point about this algorithm and others is that they all follow the same concept. Each creates a digest of length N from a multiple-block message. Each block is 512 bits in length, as shown in Figure 31.7.

Figure 31.7 *Message digest creation*



A buffer of N bits is initialized to a predetermined value. The algorithm mangles this initial buffer with the first 512 bits of the message to create the first intermediate message digest of N bits. This digest is then mangled with the second 512-bit block to create the second intermediate digest. The $(n-1)$th digest is mangled with the nth block to create the nth digest. If a block is not 512 bits, padding (0s) is added to make it so. When the last block is processed, the resulting digest is the message digest for the entire message. SHA-1 has a message digest of 160 bits (5 words, each of 32 bits).

SHA-1 hash algorithms create an *N*-bit message digest out of a message of 512-bit blocks.

SHA-1 has a message digest of 160 bits (5 words of 32 bits).

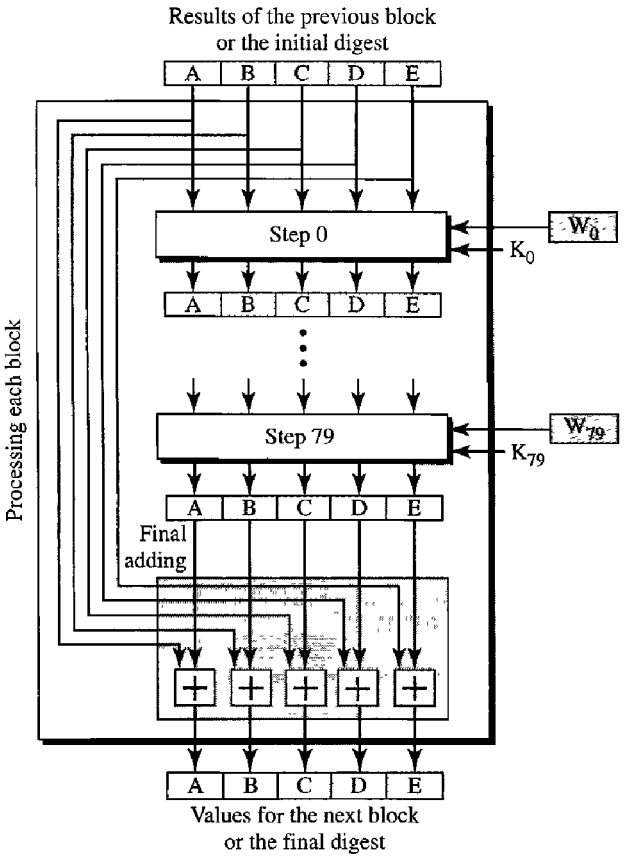
Word Expansion

Before processing, the block needs to be expanded. A block is made of 512 bits or 16 32-bit words, but we need 80 words in the processing phase. So the 16-word block needs to be expanded to 80 words, word0 to word79.

Processing Each Block

Figure 31.8 shows the general outline for the processing of one block. There are 80 steps in block processing. In each step, one word from the expanded block and one 32-bit constant are mangled together and then operated on to create a new digest. At the beginning of processing, the values of digest words (A, B, C, D, and E) are saved into five temporary variables. At the end of the processing (after step 79), these values are

Figure 31.8 *Processing of one block in SHA-1*



added to the values created from step 79. The detail of each step is complex and beyond the scope of this book. The only thing we need to know is that each step mangles a word of data and a constant to create a result that is fed to the next step.

31.4 MESSAGE AUTHENTICATION

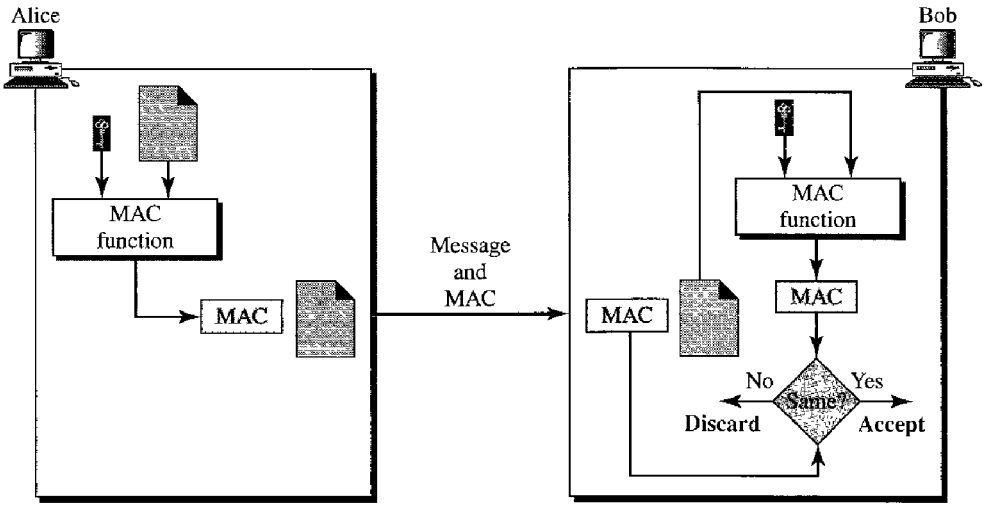
A hash function guarantees the integrity of a message. It guarantees that the message has not been changed. A hash function, however, does not authenticate the sender of the message. When Alice sends a message to Bob, Bob needs to know if the message is coming from Alice or Eve. To provide message authentication, Alice needs to provide proof that it is Alice sending the message and not an imposter. A hash function per se cannot provide such a proof. The digest created by a hash function is normally called a **modification detection code (MDC).** The code can detect any modification in the message.

MAC

To provide message authentication, we need to change a modification detection code to a **message authentication code (MAC).** An MDC uses a keyless hash function; a MAC uses a keyed hash function. A keyed hash function includes the symmetric key

between the sender and receiver when creating the digest. Figure 31.9 shows how Alice uses a keyed hash function to authenticate her message and how Bob can verify the authenticity of the message.

Figure 31.9 *MAC, created by Alice and checked by Bob*



Alice, using the symmetric key between herself and Bob (K_{AB}) and a keyed hash function, generates a MAC. She then concatenates the MAC with the original message and sends the two to Bob. Bob receives the message and the MAC. He separates the message from the MAC. He applies the same keyed hash function to the message using the symmetric key K_{AB} to get a fresh MAC. He then compares the MAC sent by Alice with the newly generated MAC. If the two MACs are identical, the message has not been modified and the sender of the message is definitely Alice.

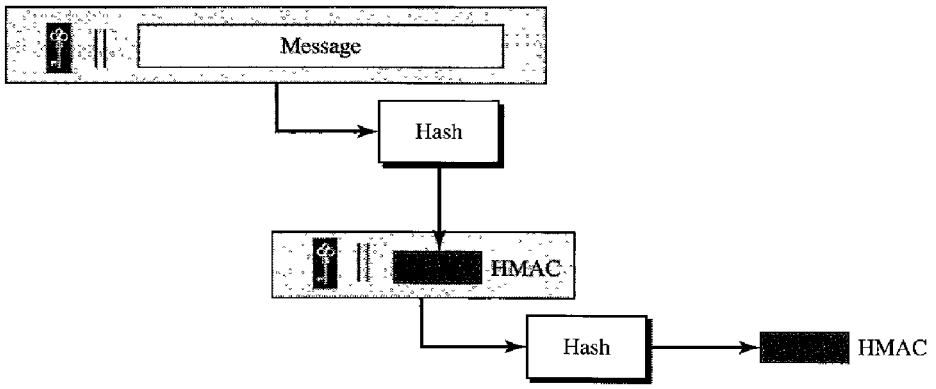
HMAC

There are several implementations of MAC in use today. However, in recent years, some MACs have been designed that are based on keyless hash functions such as SHA-1. This idea is a **hashed MAC,** called **HMAC,** that can use any standard keyless hash function such as SHA-1. HMAC creates a nested MAC by applying a keyless hash function to the concatenation of the message and a symmetric key. Figure 31.10 shows the general idea.

A copy of the symmetric key is prepended to the message. The combination is hashed using a keyless hash function, such as SHA-1. The result of this process is an intermediate HMAC which is again prepended with the key (the same key), and the result is again hashed using the same algorithm. The final result is an HMAC.

The receiver receives this final HMAC and the message. The receiver creates its own HMAC from the received message and compares the two HMACs to validate the integrity of the message and authenticate the data origin. Note that the details of an HMAC can be more complicated than what we have shown here.

Figure 31.10 *HMAC*



31.5 DIGITAL SIGNATURE

Although a MAC can provide message integrity and message authentication, it has a drawback. It needs a symmetric key that must be established between the sender and the receiver. A digital signature, on the other hand, can use a pair of asymmetric keys (a public one and a private one).

We are all familiar with the concept of a signature. We sign a document to show that it originated from us or was approved by us. The signature is proof to the recipient that the document comes from the correct entity. When a customer signs a check to himself, the bank needs to be sure that the check is issued by that customer and nobody else. In other words, a signature on a document, when verified, is a sign of authentication; the document is authentic. Consider a painting signed by an artist. The signature on the art, if authentic, means that the painting is probably authentic.

When Alice sends a message to Bob, Bob needs to check the authenticity of the sender; he needs to be sure that the message comes from Alice and not Eve. Bob can ask Alice to sign the message electronically. In other words, an electronic signature can prove the authenticity of Alice as the sender of the message. We refer to this type of signature as a **digital signature.**

Comparison

Before we continue any further, let us discuss the differences between two types of signatures: conventional and digital.

Inclusion

A conventional signature is included in the document; it is part of the document. When we write a check, the signature is on the check; it is not a separate document. On the other hand, when we sign a document digitally, we send the signature as a separate document. The sender sends two documents: the message and the signature. The recipient receives both documents and verifies that the signature belongs to the supposed sender. If this is proved, the message is kept; otherwise, it is rejected.

Verification Method

The second difference between the two types of documents is the method of verifying the signature. In conventional signature, when the recipient receives a document, she compares the signature on the document with the signature on file. If they are the same, the document is authentic. The recipient needs to have a copy of this signature on file for comparison. In digital signature, the recipient receives the message and the signature. A copy of the signature is not stored anywhere. The recipient needs to apply a verification technique to the combination of the message and the signature to verify the authenticity.

Relationship

In conventional signature, there is normally a one-to-many relationship between a signature and documents. A person, for example, has a signature that is used to sign many checks, many documents, etc. In digital signature, there is a one-to-one relationship between a signature and a message. Each message has its own signature. The signature of one message cannot be used in another message. If Bob receives two messages, one after another, from Alice, he cannot use the signature of the first message to verify the second. Each message needs a new signature.

Duplicity

Another difference between the two types of signatures is a quality called duplicity. In conventional signature, a copy of the signed document can be distinguished from the original one on file. In digital signature, there is no such distinction unless there is a factor of time (such as a timestamp) on the document. For example, suppose Alice sends a document instructing Bob to pay Eve. If Eve intercepts the document and the signature, she can resend it later to get money again from Bob.

Need for Keys

In conventional signature a signature is like a private "key" belonging to the signer of the document. The signer uses it to sign a document; no one else has this signature. The copy of the signature is on file like a public key; anyone can use it to verify a document, to compare it to the original signature.

In digital signature, the signer uses her private key, applied to a signing algorithm, to sign the document. The verifier, on the other hand, uses the public key of the signer, applied to the verifying algorithm, to verify the document.

Can we use a secret (symmetric) key to both sign and verify a signature? The answer is no for several reasons. First, a secret key is known only between two entities (Alice and Bob, for example). So if Alice needs to sign another document and send it to Ted, she needs to use another secret key. Second, as we will see, creating a secret key for a session involves authentication, which normally uses digital signature. We have a vicious cycle. Third, Bob could use the secret key between himself and Alice, sign a document, send it to Ted, and pretend that it came from Alice.

A digital signature needs a public-key system.

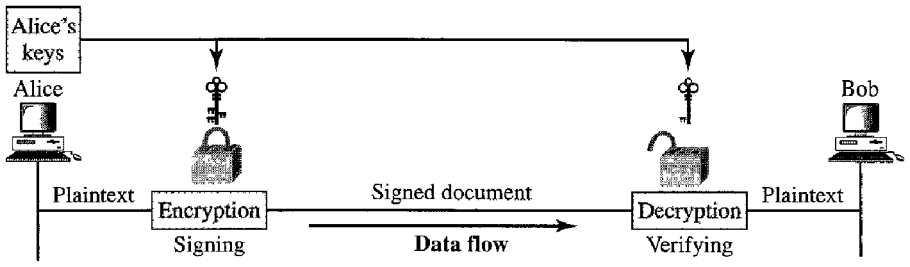
Process

Digital signature can be achieved in two ways: signing the document or signing a digest of the document.

Signing the Document

Probably, the easier, but less efficient way is to sign the document itself. Signing a document is encrypting it with the private key of the sender; verifying the document is decrypting it with the public key of the sender. Figure 31.11 shows how signing and verifying are done.

Figure 31.11 *Signing the message itself in digital signature*



We should make a distinction between private and public keys as used in digital signature and public and private keys as used for confidentiality. In the latter, the private and public keys of the receiver are used in the process. The sender uses the public key of the receiver to encrypt; the receiver uses his own private key to decrypt. In digital signature, the private and public keys of the sender are used. The sender uses her private key; the receiver uses the public key of the sender.

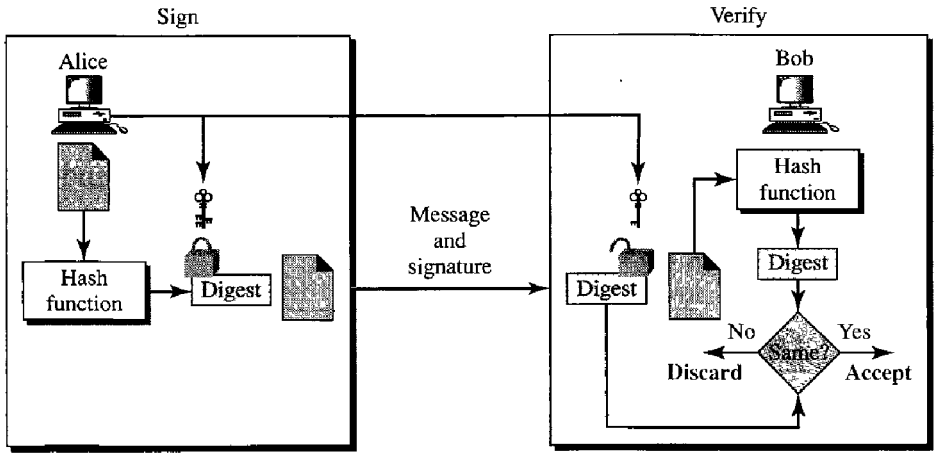
In a cryptosystem, we use the private and public keys of the receiver; in digital signature, we use the private and public key of the sender.

Signing the Digest

We mentioned that the public key is very inefficient in a cryptosystem if we are dealing with long messages. In a digital signature system, our messages are normally long, but we have to use public keys. The solution is not to sign the message itself; instead, we sign a digest of the message. As we learned, a carefully selected message digest has a one-to-one relationship with the message. The sender can sign the message digest, and the receiver can verify the message digest. The effect is the same. Figure 31.12 shows signing a digest in a digital signature system.

A digest is made out of the message at Alice's site. The digest then goes through the signing process using Alice's private key. Alice then sends the message and the signature to Bob. As we will see later in the chapter, there are variations in the process that are dependent on the system. For example, there might be additional calculations before the digest is made or other secret keys might be used. In some systems, the signature is a set of values.

Figure 31.12 *Signing the digest in a digital signature*



At Bob's site, using the same public hash function, a digest is first created out of the received message. Calculations are done on the signature and the digest. The verifying process also applies criteria on the result of the calculation to determine the authenticity of the signature. If authentic, the message is accepted; otherwise, it is rejected.

Services

A digital signature can provide three out of the five services we mentioned for a security system: message integrity, message authentication, and nonrepudiation. Note that a digital signature scheme does not provide confidential communication. If confidentiality is required, the message and the signature must be encrypted using either a secret-key or public-key cryptosystem.

Message Integrity

The integrity of the message is preserved even if we sign the whole message because we cannot get the same signature if the message is changed. The signature schemes today use a hash function in the signing and verifying algorithms that preserve the integrity of the message.

A digital signature today provides message integrity.

Message Authentication

A secure signature scheme, like a secure conventional signature (one that cannot be easily copied), can provide message authentication. Bob can verify that the message is sent by Alice because Alice's public key is used in verification. Alice's public key cannot create the same signature as Eve's private key.

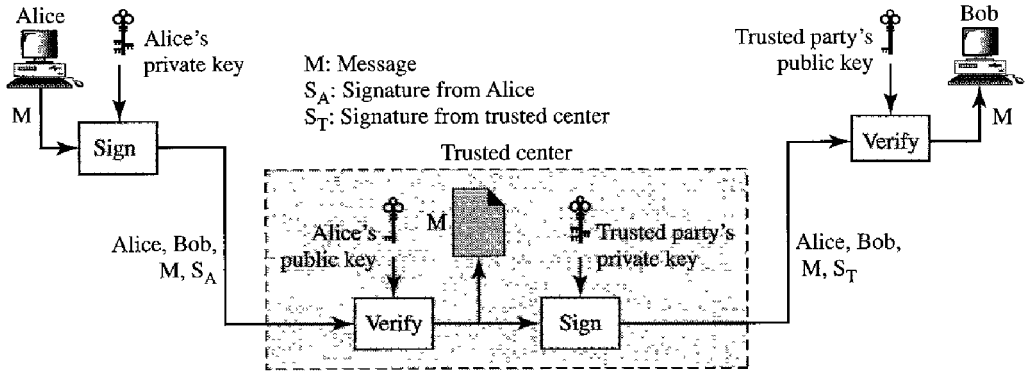
Digital signature provides message authentication.

Message Nonrepudiation

If Alice signs a message and then denies it, can Bob later prove that Alice actually signed it? For example, if Alice sends a message to a bank (Bob) and asks to transfer $10,000 from her account to Ted's account, can Alice later deny that she sent this message? With the scheme we have presented so far, Bob might have a problem. Bob must keep the signature on file and later use Alice's public key to create the original message to prove the message in the file and the newly created message are the same. This is not feasible because Alice may have changed her private/public key during this time; she may also claim that the file containing the signature is not authentic.

One solution is a trusted third party. People can create a trusted party among themselves. In Chapter 32, we will see that a trusted party can solve many other problems concerning security services and key exchange. Figure 31.13 shows how a trusted party can prevent Alice from denying that she sent the message.

Figure 31.13 *Using a trusted center for nonrepudiation*



Alice creates a signature from her message (S_A) and sends the message, her identity, Bob's identity, and the signature to the center. The center, after checking that Alice's public key is valid, verifies through Alice's public key that the message comes from Alice. The center then saves a copy of the message with the sender identity, recipient identity, and a timestamp in its archive. The center uses its private key to create another signature (S_T) from the message. The center then sends the message, the new signature, Alice's identity, and Bob's identity to Bob. Bob verifies the message using the public key of the trusted center.

If in the future Alice denies that she has sent the message, the center can show a copy of the saved message. If Bob's message is a duplicate of the message saved at the center, Alice will lose the dispute. To make everything confidential, a level of encryption/decryption can be added to the scheme as discussed in the next section.

Nonrepudiation can be provided using a trusted party.

Signature Schemes

Several signature schemes have evolved during the last few decades. Some of them have been implemented. Such as RSA and DSS (Digital Signature Standard) schemes. The latter will probably become the standard. However, the details of these schemes are beyond the scope of this book.

31.6 ENTITY AUTHENTICATION

Entity authentication is a technique designed to let one party prove the identity of another party. An *entity* can be a person, a process, a client, or a server. The entity whose identity needs to be proved is called the **claimant;** the party that tries to prove the identity of the claimant is called the **verifier.** When Bob tries to prove the identity of Alice, Alice is the claimant, and Bob is the verifier.

There are two differences between message authentication and **entity authentication.** First, message authentication may not happen in real time; entity authentication does. In the former, Alice sends a message to Bob. When Bob authenticates the message, Alice may or may not be present in the communication process. On the other hand, when Alice requests entity authentication, there is no real message communication involved until Alice is authenticated by Bob. Alice needs to be online and takes part in the process. Only after she is authenticated can messages be communicated between Alice and Bob. Message authentication is required when an e-mail is sent from Alice to Bob. Entity authentication is required when Alice gets cash from an automatic teller machine. Second, message authentication simply authenticates one message; the process needs to be repeated for each new message. Entity authentication authenticates the claimant for the entire duration of a session.

In entity authentication, the claimant must identify herself to the verifier. This can be done with one of three kinds of witnesses: *something known, something possessed,* or *something inherent.*

- ***Something known.*** This is a secret known only by the claimant that can be checked by the verifier. Examples are a password, a PIN number, a secret key, and a private key.
- ***Something possessed.*** This is something that can prove the claimant's identity. Examples are a passport, a driver's license, an identification card, a credit card, and a smart card.
- ***Something inherent.*** This is an inherent characteristic of the claimant. Examples are conventional signature, fingerprints, voice, facial characteristics, retinal pattern, and handwriting.

Passwords

The simplest and the oldest method of entity authentication is the **password,** something that the claimant *possesses*. A password is used when a user needs to access a system to use the system's resources (log-in). Each user has a user identification that is public and a password that is private. We can divide this authentication scheme into two separate groups: the **fixed password** and the **one-time password.**

Fixed Password

In this group, the password is fixed; the same password is used over and over for every access. This approach is subject to several attacks.

- **Eavesdropping.** Eve can watch Alice when she types her password. Most systems, as a security measure, do not show the characters a user types. Eavesdropping can take a more sophisticated form. Eve can listen to the line and then intercept the message, thereby capturing the password for her own use.
- **Stealing a Password.** The second type of attack occurs when Eve tries to physically steal Alice's password. This can be prevented if Alice does not write down the password; instead, she just commits it to memory. Therefore, a password should be very simple or else related to something familiar to Alice, which makes the password vulnerable to other types of attacks.
- **Accessing a file.** Eve can hack into the system and get access to the file where the passwords are stored. Eve can read the file and find Alice's password or even change it. To prevent this type of attack, the file can be read/write protected. However, most systems need this type of file to be readable by the public.
- **Guessing.** Eve can log into the system and try to guess Alice's password by trying different combinations of characters. The password is particularly vulnerable if the user is allowed to choose a short password (a few characters). It is also vulnerable if Alice has chosen something unimaginative, such as her birthday, her child's name, or the name of her favorite actor. To prevent guessing, a long random password is recommended, something that is not very obvious. However, the use of such a random password may also create a problem; Alice might store the password somewhere so as not to forget it. This makes the password subject to stealing.

A more secure approach is to store the hash of the password in the password file (instead of the plaintext password). Any user can read the contents of the file, but, because the hash function is a one-way function, it is almost impossible to guess the value of the password. The hash function prevents Eve from gaining access to the system even though she has the password file. However, there is a possibility of another type of attack called the **dictionary attack.** In this attack, Eve is interested in finding one password, regardless of the user ID. For example, if the password is 6 digits, Eve can create a list of 6-digit numbers (000000 to 999999), and then apply the hash function to every number; the result is a list of 1 million hashes. She can then get the password file and search the second-column entries to find a match. This could be programmed and run offline on Eve's private computer. After a match is found, Eve can go online and use the password to access the system. We will see how to make this attack more difficult in the third approach.

Another approach is called **salting** the password. When the password string is created, a random string, called the salt, is concatenated to the password. The salted password is then hashed. The ID, salt, and the hash are then stored in the file. Now, when a user asks for access, the system extracts the salt, concatenates it with the received password, makes a hash out of the result, and compares it with the hash stored in the file. If there is a match, access is granted; otherwise, it is denied. Salting makes the dictionary attack more difficult. If the original password is 6 digits and the salt is 4 digits, then hashing is

done over a 10-digit value. This means that Eve now needs to make a list of 10 million items and create a hash for each of them. The list of hashes has 10 million entries and the comparison takes much longer. Salting is very effective if the salt is a very long random number. The UNIX operating system uses a variation of this method.

In another approach, two identification techniques are combined. A good example of this type of authentication is the use of an ATM card with a PIN (personal identification number). The card belongs to the category "something possessed" and the PIN belongs to the category "something known." The PIN is actually a password that enhances the security of the card. If the card is stolen, it cannot be used unless the PIN is known. The PIN, however, is traditionally very short so it is easily remembered by the owner. This makes it vulnerable to the guessing type of attack.

One-Time Password

In this type of scheme, a password is used only once. It is called the **one-time password.** A one-time password makes eavesdropping and stealing useless. However, this approach is very complex, and we leave its discussion to some specialized books.

Challenge-Response

In password authentication, the claimant proves her identity by demonstrating that she knows a secret, the password. However, since the claimant reveals this secret, the secret is susceptible to interception by the adversary. In **challenge-response authentication,** the claimant proves that she *knows* a secret without revealing it. In other words, the claimant does not reveal the secret to the verifier; the verifier either has it or finds it.

In challenge-response authentication,
the claimant proves that she knows a secret without revealing it.

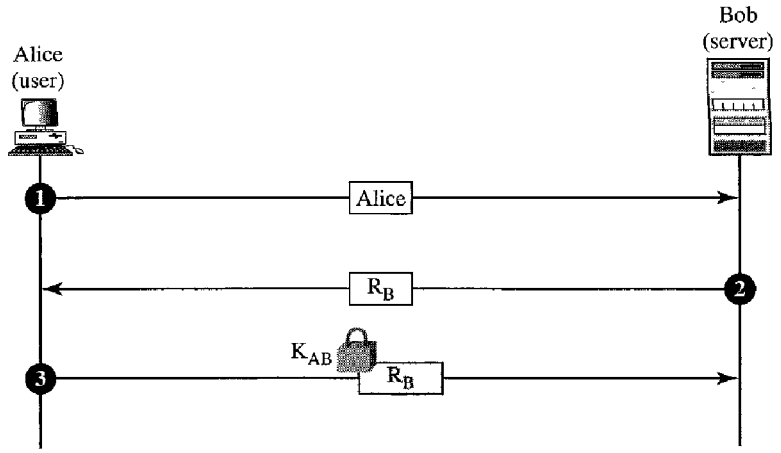
The challenge is a time-varying value such as a random number or a timestamp which is sent by the verifier. The claimant applies a function to the challenge and sends the result, called a *response,* to the verifier. The response shows that the claimant knows the secret.

The challenge is a time-varying value sent by the verifier;
the response is the result of a function applied on the challenge.

Using a Symmetric-Key Cipher

In the first category, the challenge-response authentication is achieved using symmetric-key encryption. The secret here is the shared secret key, known by both the claimant and the verifier. The function is the encrypting algorithm applied on the challenge. Figure 31.14 shows one approach. The first message is not part of challenge-response, it only informs the verifier that the claimant wants to be challenged. The second message is the challenge. And R_B is the nonce randomly chosen by the verifier to challenge the claimant. The claimant encrypts the **nonce** using the shared secret key known only to the claimant and the verifier and sends the result to the verifier. The verifier decrypts

Figure 31.14 *Challenge/response authentication using a nonce*



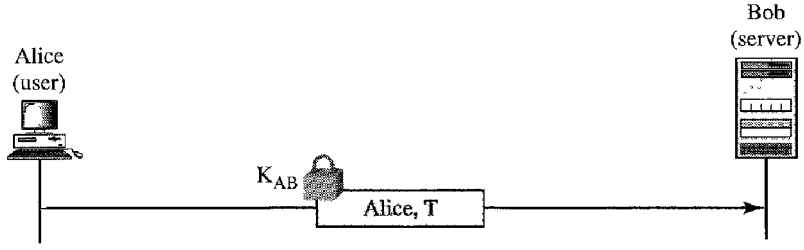
the message. If the nonce obtained from decryption is the same as the one sent by the verifier, Alice is granted access.

Note that in this process, the claimant and the verifier need to keep the symmetric key used in the process secret. The verifier must also keep the value of the nonce for claimant identification until the response is returned.

The reader may have noticed that use of a nonce prevents a replay of the third message by Eve. Eve cannot replay the third message and pretend that it is a new request for authentication by Alice because once Bob receives the response, the value of R_B is not valid any more. The next time a new value is used.

In the second approach, the time-varying value is a timestamp, which obviously changes with time. In this approach the challenge message is the current time sent from the verifier to the claimant. However, this supposes that the client and the server clocks are synchronized; the claimant knows the current time. This means that there is no need for the challenge message. The first and third messages can be combined. The result is that authentication can be done using one message, the response to an implicit challenge, the current time. Figure 31.15 shows the approach.

Figure 31.15 *Challenge-response authentication using a timestamp*



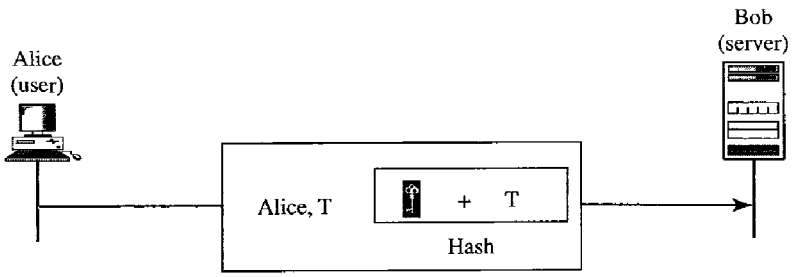
Using Keyed-Hash Functions

Instead of using encryption and decryption for entity authentication, we can use a keyed-hash function (MAC). There are two advantages to this scheme. First, the

encryption/decryption algorithm is not exportable to some countries. Second, in using a keyed-hash function, we can preserve the integrity of challenge and response messages and at the same time use a secret, the key.

Let us see how we can use a keyed-hash function to create a challenge response with a timestamp. Figure 31.16 shows the scheme.

Figure 31.16 *Challenge-response authentication using a keyed-hash function*

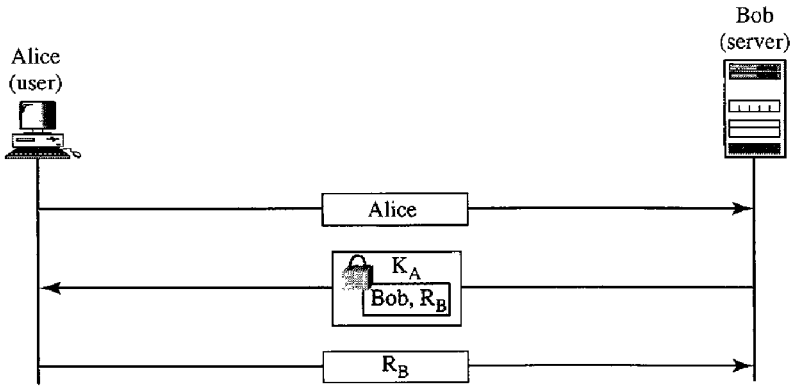


Note that in this case, the timestamp is sent both as plaintext and as text scrambled by the keyed-hash function. When Bob receives the message, he takes the plaintext T, applies the keyed-hash function, and then compares his calculation with what he received to determine the authenticity of Alice.

Using an Asymmetric-Key Cipher

Instead of a symmetric-key cipher, we can use an asymmetric-key cipher for entity authentication. Here the secret must be the private key of the claimant. The claimant must show that she owns the private key related to the public key that is available to everyone. This means that the verifier must encrypt the challenge using the public key of the claimant; the claimant then decrypts the message using her private key. The response to the challenge is the decrypted challenge. We show two approaches: one for unidirectional authentication and one for bidirectional authentication. In one approach, Bob encrypts the challenge using Alice's public key. Alice decrypts the message with her private key and sends the nonce to Bob. Figure 31.17 shows this approach.

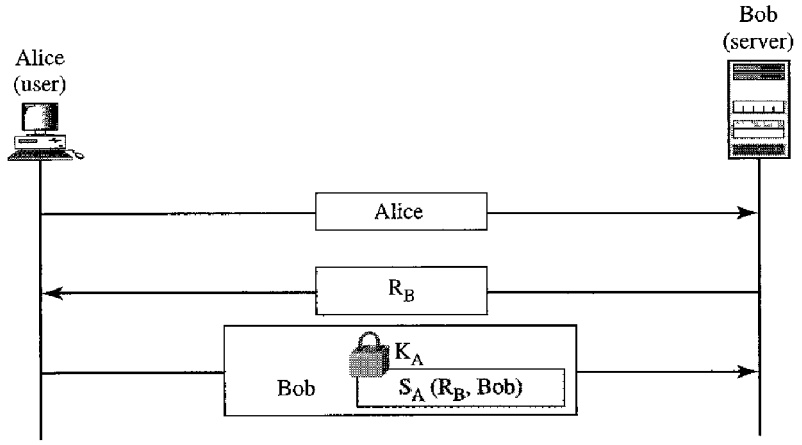
Figure 31.17 *Authentication, asymmetric-key*



Using Digital Signature

We can use digital signature for entity authentication. In this method, we let the claimant use her private key for signing instead of using it for decryption. In one approach shown in Figure 31.18, Bob uses a plaintext challenge. Alice signs the response.

Figure 31.18 *Authentication, using digital signature*



31.7 KEY MANAGEMENT

We have used symmetric-key and asymmetric-key cryptography in our discussion throughout the chapter. However, we never discussed how secret keys in symmetric-key cryptography and how public keys in asymmetric-key cryptography are distributed and maintained. In this section, we touch on these two issues. We first discuss the distribution of symmetric keys; we then discuss the distribution of asymmetric keys.

Symmetric-Key Distribution

We have learned that symmetric-key cryptography is more efficient than asymmetric-key cryptography when we need to encrypt and decrypt large messages. Symmetric-key cryptography, however, needs a shared secret key between two parties.

If Alice needs to exchange confidential messages with N people, she needs N different keys. What if N people need to communicate with one another? A total of $N(N-1)/2$ keys is needed. Each person needs to have $N-1$ keys to communicate with each of the other people, but because the keys are shared, we need only $N(N-1)/2$. This means that if 1 million people need to communicate with one another, each person has almost 0.5 million different keys; in total, almost 1 billion keys are needed. This is normally referred to as the N^2 problem because the number of required keys for N entities is close to N^2.

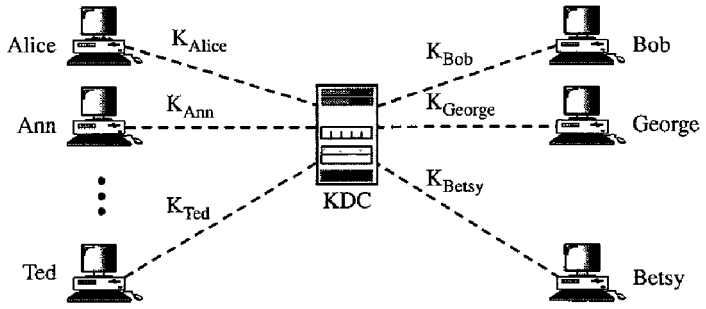
The number of keys is not the only problem; the distribution of keys is another. If Alice and Bob want to communicate, they need to somehow exchange a secret key; if Alice wants to communicate with 1 million people, how can she exchange 1 million keys with 1 million people? Using the Internet is definitely not a secure method.

It is obvious that we need an efficient way of maintaining and distributing secret keys.

Key Distribution Center: KDC

A practical solution is the use of a trusted party, referred to as a **key distribution center (KDC).** To reduce the number of keys, each person establishes a shared secret key with the KDC as shown in Figure 31.19.

Figure 31.19 *KDC*



A secret key is established between KDC and each member. Alice has a secret key with KDC, which we refer to as K_{Alice}; Bob has a secret key with KDC, which we refer to as K_{Bob}; and so on. Now the question is, How can Alice send a confidential message to Bob? The process is as follows:

1. Alice sends a request to KDC, stating that she needs a session (temporary) secret key between herself and Bob.
2. KDC informs Bob of Alice's request.
3. If Bob agrees, a session key is created between the two.

The secret key between Alice and Bob that is established with the KDC is used to authenticate Alice and Bob to the KDC and to prevent Eve from impersonating either of them. We discuss how a session key is established between Alice and Bob later in the chapter.

Session Keys

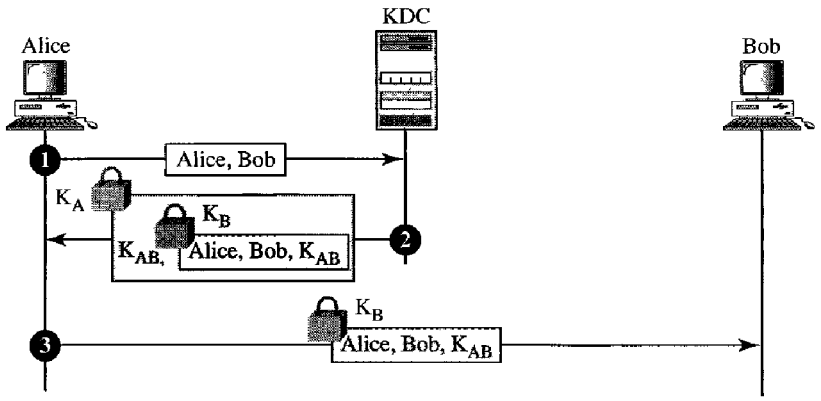
A KDC creates a secret key for each member. This secret key can be used only between the member and the KDC, not between two members. If Alice needs to communicate secretly with Bob, she needs a secret key between herself and Bob. A KDC can create a session (temporary) key between Alice and Bob using their keys with the center. The keys of Alice and Bob are used to authenticate Alice and Bob to the center and to each other before the session key is established. After communication is terminated, the session key is no longer valid.

A session symmetric key between two parties is used only once.

Several different approaches have been proposed to create the session key using ideas we previously discussed for entity authentication.

Let us discuss one approach, the simplest one, as shown in Figure 31.20. Although this system has some flaws, it shows the idea. More sophisticated approaches can be found in security books.

Figure 31.20 *Creating a session key between Alice and Bob using KDC*



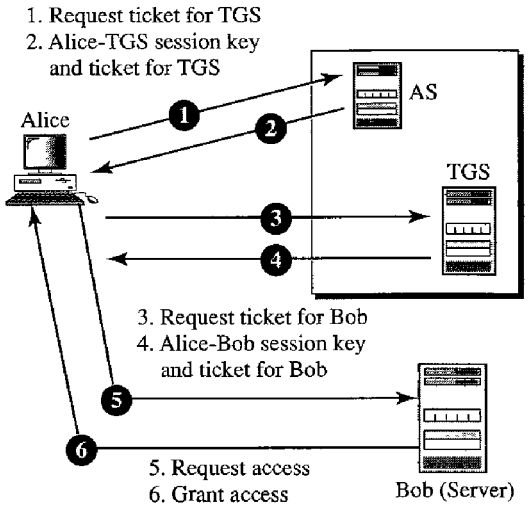
- **Step 1** Alice sends a plaintext message to the KDC to obtain a symmetric session key between Bob and herself. The message contains her registered identity (the word *Alice* in the figure) and the identity of Bob (the word *Bob* in the figure). This message is not encrypted, it is public. KDC does not care.
- **Step 2** KDC receives the message and creates what is called a **ticket.** The ticket is encrypted using Bob's key (K_B). The ticket contains the identities of Alice and Bob and the session key (K_{AB}). The ticket with a copy of the session key is sent to Alice. Alice receives the message, decrypts it, and extracts the session key. She cannot decrypt Bob's ticket; the ticket is for Bob, not for Alice. Note that we have a double encryption in this message; the ticket is encrypted and the entire message is also encrypted. In the second message, Alice is actually authenticated to the KDC, because only Alice can open the whole message using her secret key with KDC.
- **Step 3** Alice sends the ticket to Bob. Bob opens the ticket and knows that Alice needs to send messages to him using K_{AB} as the session key. Note that in this message, Bob is authenticated to the KDC because only Bob can open the ticket. Since Bob is authenticated to the KDC, he is also authenticated to Alice who trusts the KDC. In the same way, Alice is also authenticated to Bob, because Bob trusts the KDC and the KDC has sent the ticket to Bob which includes the identity of Alice.

Kerberos

Kerberos is an authentication protocol and at the same time a KDC that has become very popular. Several systems including Windows 2000 use Kerberos. It is named after the three-headed dog in Greek mythology that guards the Gates of Hades. Originally designed at M.I.T., it has gone through several versions. We discuss only version 4, the most popular.

Servers Three servers are involved in the Kerberos protocol: an authentication server (AS), a ticket-granting server (TGS), and a real (data) server that provides services to others. In our examples and figures *Bob* is the real server and *Alice* is the user requesting service. Figure 31.21 shows the relationship between these three servers.

Figure 31.21 *Kerberos servers*

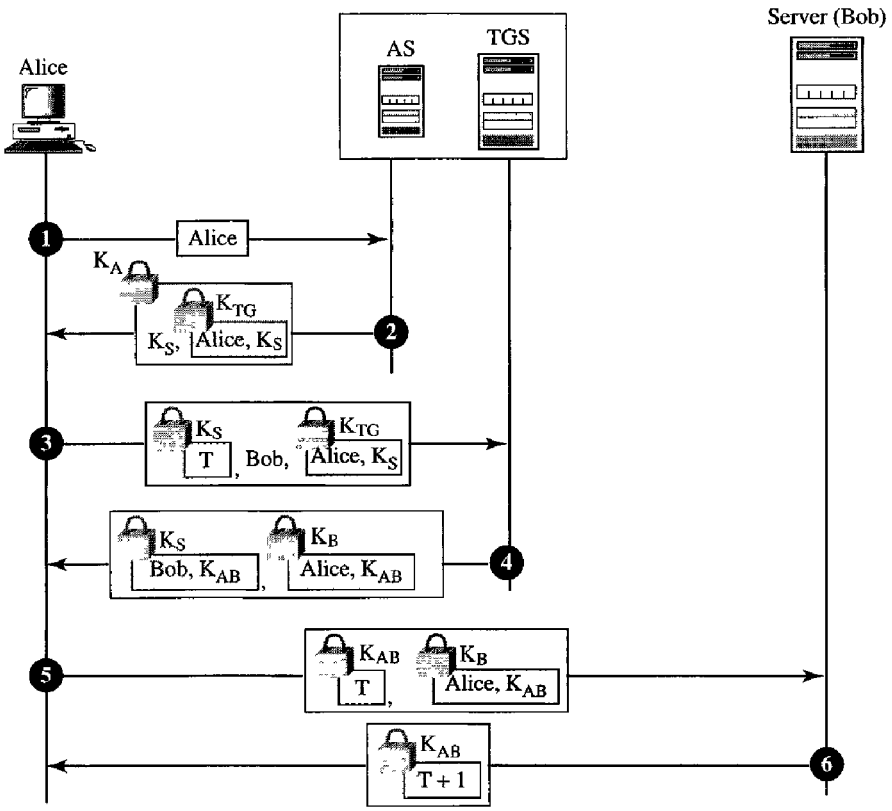


- **Authentication Server (AS).** AS is the KDC in Kerberos protocol. Each user registers with AS and is granted a user identity and a password. AS has a database with these identities and the corresponding passwords. AS verifies the user, issues a session key to be used between Alice and TGS, and sends a ticket for TGS.
- **Ticket-Granting Server (TGS).** TGS issues a ticket for the real server (Bob). It also provides the session key (K_{AB}) between Alice and Bob. Kerberos has separated the user verification from ticket issuing. In this way, although Alice verifies her ID just once with AS, she can contact TGS multiple times to obtain tickets for different real servers.
- **Real Server.** The real server (Bob) provides services for the user (Alice). Kerberos is designed for a client/server program such as FTP, in which a user uses the client process to access the server process. Kerberos is not used for person-to-person authentication.

Operation A client process (Alice) can access a process running on the real server (Bob) in six steps as shown in Figure 31.22.

- **Step 1.** Alice sends her request to AS in plaintext, using her registered identity.
- **Step 2.** AS sends a message encrypted with Alice's symmetric key K_A. The message contains two items: a session key K_S that is used by Alice to contact TGS and a ticket for TGS that is encrypted with the TGS symmetric key K_{TG}. Alice does not know K_A, but when the message arrives, she types her symmetric password. The password and the appropriate algorithm together create K_A if the password is correct. The password is then immediately destroyed; it is not sent to the network, and it does

Figure 31.22 *Kerberos example*



not stay in the terminal. It is only used for a moment to create K_A. The process now uses K_A to decrypt the message sent. Both K_S and the ticket are extracted.

- **Step 3.** Alice now sends three items to TGS. The first is the ticket received from AS. The second is the name of the real server (Bob), the third is a timestamp which is encrypted by K_S. The timestamp prevents a replay by Eve.
- **Step 4.** Now, TGS sends two tickets, each containing the session key between Alice and Bob K_{AB}. The ticket for Alice is encrypted with K_S; the ticket for Bob is encrypted with Bob's key K_B. Note that Eve cannot extract K_{AB} because she does not know K_S or K_B. She cannot replay step 3 because she cannot replace the timestamp with a new one (she does not know K_S). Even if she is very quick and sends the step 3 message before the timestamp has expired, she still receives the same two tickets that she cannot decipher.
- **Step 5.** Alice sends Bob's ticket with the timestamp encrypted by K_{AB}.
- **Step 6.** Bob confirms the receipt by adding 1 to the timestamp. The message is encrypted with K_{AB} and sent to Alice.

Using Different Servers Note that if Alice needs to receive services from different servers, she need repeat only steps 3 to 6. The first two steps have verified Alice's identity and need not be repeated. Alice can ask TGS to issue tickets for multiple servers by repeating steps 3 to 6.

Realms Kerberos allows the global distribution of ASs and TGSs, with each system called a realm. A user may get a ticket for a local server or a remote server. In the second case, for example, Alice may ask her local TGS to issue a ticket that is accepted by a remote TGS. The local TGS can issue this ticket if the remote TGS is registered with the local one. Then Alice can use the remote TGS to access the remote real server.

Public-Key Distribution

In asymmetric-key cryptography, people do not need to know a symmetric shared key. If Alice wants to send a message to Bob, she only needs to know Bob's public key, which is open to the public and available to everyone. If Bob needs to send a message to Alice, he only needs to know Alice's public key, which is also known to everyone. In public-key cryptography, everyone shields a private key and advertises a public key.

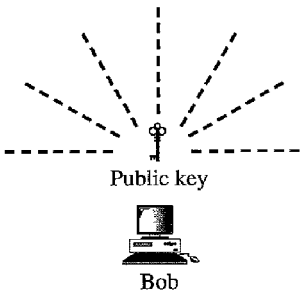
In public-key cryptography, everyone has access to everyone's public key; public keys are available to the public.

Public keys, like secret keys, need to be distributed to be useful. Let us briefly discuss the way public keys can be distributed.

Public Announcement

The naive approach is to announce public keys publicly. Bob can put his public key on his website or announce it in a local or national newspaper. When Alice needs to send a confidential message to Bob, she can obtain Bob's public key from his site or from the newspaper, or she can even send a message to ask for it. Figure 31.23 shows the situation.

Figure 31.23 *Announcing a public key*

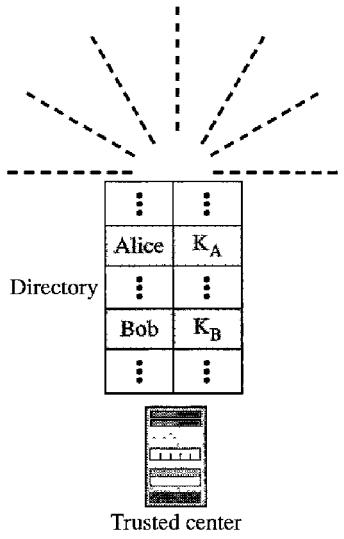


This approach, however, is not secure; it is subject to forgery. For example, Eve could make such a public announcement. Before Bob can react, damage could be done. Eve can fool Alice into sending her a message that is intended for Bob. Eve could also sign a document with a corresponding forged private key and make everyone believe it was signed by Bob. The approach is also vulnerable if Alice directly requests Bob's public key. Eve can intercept Bob's response and substitute her own forged public key for Bob's public key.

Trusted Center

A more secure approach is to have a trusted center retain a directory of public keys. The directory, like the one used in a telephone system, is dynamically updated. Each user can select a private/public key, keep the private key, and deliver the public key for insertion into the directory. The center requires that each user register in the center and prove his or her identity. The directory can be publicly advertised by the trusted center. The center can also respond to any inquiry about a public key. Figure 31.24 shows the concept.

Figure 31.24 *Trusted center*



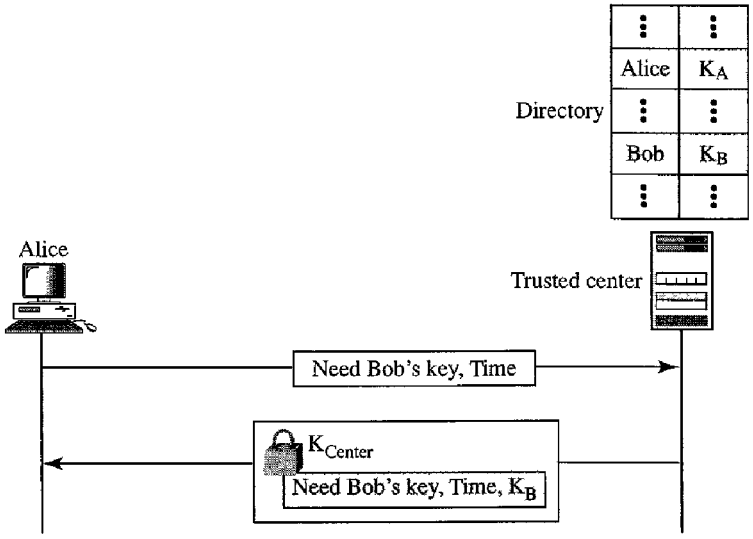
Controlled Trusted Center

A higher level of security can be achieved if there are added controls on the distribution of the public key. The public-key announcements can include a timestamp and be signed by an authority to prevent interception and modification of the response. If Alice needs to know Bob's public key, she can send a request to the center including Bob's name and a timestamp. The center responds with Bob's public key, the original request, and the timestamp signed with the private key of the center. Alice uses the public key of the center, known by all, to decrypt the message and extract Bob's public key. Figure 31.25 shows one scenario.

Certification Authority

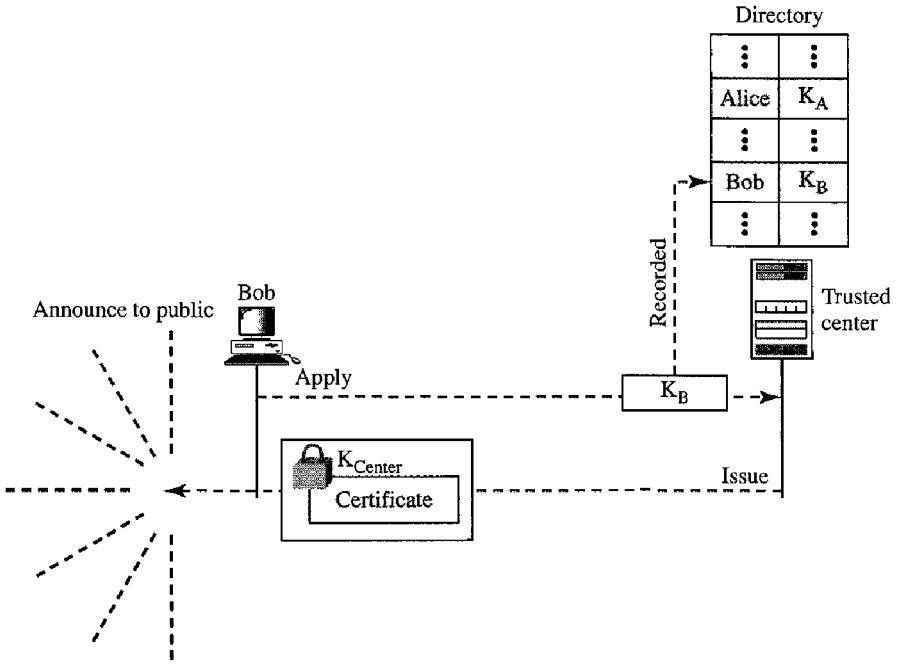
The previous approach can create a heavy load on the center if the number of requests is large. The alternative is to create public-key certificates. Bob wants two things: he wants people to know his public key, and he wants no one to accept a public key forged as his. Bob can go to a **certification authority (CA)**—a federal or state organization that binds a public key to an entity and issues a certificate. The CA has a well-known public key itself that cannot be forged. The CA checks Bob's **identification** (using a

Figure 31.25 *Controlled trusted center*



picture ID along with other proof). It then asks for Bob's public key and writes it on the certificate. To prevent the certificate itself from being forged, the CA signs the certificate with its private key. Now Bob can upload the signed certificate. Anyone who wants Bob's public key downloads the signed certificate and uses the public key of the center to extract Bob's public key. Figure 31.26 shows the concept.

Figure 31.26 *Certification authority*



X.509 Although the use of a CA has solved the problem of public-key fraud, it has created a side effect. Each certificate may have a different format. If Alice wants to use a program to automatically download different certificates and digests belonging to different people, the program may not be able to do so. One certificate may have the public key in one format and another in another format. The public key may be on the first line in one certificate and on the third line in another. Anything that needs to be used universally must have a universal format.

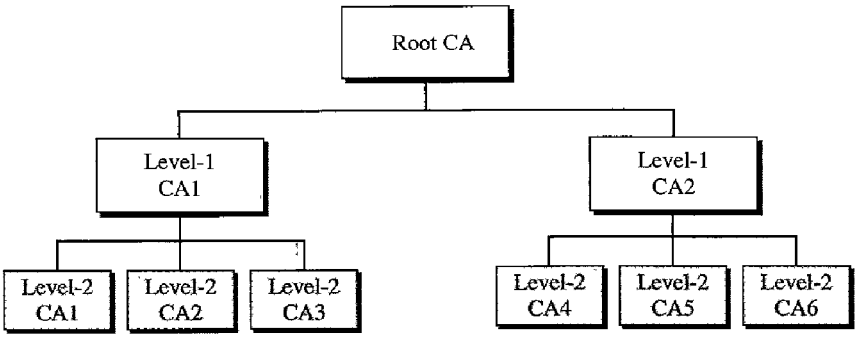
To remove this side effect, ITU has designed a protocol called **X.509**, which has been accepted by the Internet with some changes. X.509 is a way to describe the certificate in a structured way. It uses a well-known protocol called ASN.1 (Abstract Syntax Notation 1) that defines fields familiar to C programmers. The following lists the fields in a certificate.

- **Version** This field defines the version of X.509 of the certificate. The version number started at 0; the current version is 2 (the third version).
- **Serial number** This field defines a number assigned to each certificate. The value is unique for each certificate issued.
- **Signature** This field, for which the name is inappropriate, identifies the algorithm used to sign the certificate. Any parameter that is needed for the signature is also defined in this field.
- **Issuer** This field identifies the certification authority that issued the certificate. The name is normally a hierarchy of strings that defines a country, state, organization, department, and so on.
- **Period of validity** This field defines the earliest and the latest times the certificate is valid.
- **Subject** This field defines the entity to which the public key belongs. It is also a hierarchy of strings. Part of the field defines what is called the *common name,* which is the actual name of the beholder of the key.
- **Subject's public key** This field defines the subject's public key, the heart of the certificate. The field also defines the corresponding algorithm (RSA, for example) and its parameters.
- **Issuer unique identifier** This optional field allows two issuers to have the same *issuer* field value, if the *issuer unique identifiers* are different.
- **Subject unique identifier** This optional field allows two different subjects to have the same *subject* field value, if the *subject unique identifiers* are different.
- **Extension** This field allows issuers to add more private information to the certificate.
- **Encrypted** This field contains the algorithm identifier, a secure hash of the other fields, and a digital signature of that hash.

Public-Key Infrastructures (PKI)

When we want to use public keys universally, we have a problem similar to secret-key distribution. We found that we cannot have only one KDC to answer the queries. We need many servers. In addition, we found that the best solution is to put the servers in a hierarchical relationship with one another. Likewise, a solution to public-key queries is a hierarchical structure called a **public-key infrastructure (PKI)**. Figure 31.27 shows an example of this hierarchy.

Figure 31.27 *PKI hierarchy*



At the first level, we can have a root CA that can certify the performance of CAs in the second level; these level-1 CAs may operate in a large geographic or logical area. The level-2 CAs may operate in smaller geographic areas.

In this hierarchy, everybody trusts the root. But people may or may not trust intermediate CAs. If Alice needs to get Bob's certificate, she may find a CA somewhere to issue the certificate. But Alice may not trust that CA. In a hierarchy Alice can ask the next-higher CA to certify the original CA. The inquiry may go all the way to the root.

31.8 RECOMMENDED READING

For more details about the subjects discussed in this chapter, we recommend the following books and sites. The items in brackets [. . .] refer to the reference list at the end of the text.

Books

Several books are dedicated to network security, such as [PHS02], [Bis03], and [Sal03].

31.9 KEY TERMS

authentication server (AS)
certification authority (CA)
challenge-response authentication
claimant
dictionary attack
digital signature
eavesdropping
entity authentication
fingerprint
fixed password
hash function
hashed message authentication code (HMAC)
identification
integrity
Kerberos
key distribution center (KDC)
message authentication
message authentication code (MAC)
message confidentiality or privacy
message digest
message integrity

message nonrepudiation
modification detection code (MDC)
nonce
nonrepudiation
one-time password
one-wayness
password
privacy
public-key infrastructure (PKI)
salting
session key
SHA–1
signature scheme
signing algorithm
strong collision
ticket
ticket-granting server (TGS)
verifier
verifying algorithm
weak collision
X.509

31.10 SUMMARY

- Cryptography can provide five services. Four of these are related to the message exchange between Alice and Bob. The fifth is related to the entity trying to access a system for using its resources.
- Message confidentiality means that the sender and the receiver expect privacy.
- Message integrity means that the data must arrive at the receiver exactly as sent.
- Message authentication means that the receiver is ensured that the message is coming from the intended sender, not an imposter.
- Nonrepudiation means that a sender must not be able to deny sending a message that he sent.
- Entity authentication means to prove the identity of the entity that tries to access the system's resources.
- A message digest can be used to preserve the integrity of a document or a message. A hash function creates a message digest out of a message.
- A hash function must meet three criteria: one-wayness, resistance to weak collision, and resistance to strong collision.
- A keyless message digest is used as a modification detection code (MDC). It guarantees the integrity of the message. To authenticate the data origin, one needs a message authentication code (MAC).
- MACs are keyed hash functions that create a compressed digest from the message added with the key. The method has the same basis as encryption algorithms.
- A digital signature scheme can provide the same services provided by a conventional signature. A conventional signature is included in the document; a digital signature is a separate entity.
- Digital signature provides message integrity, authentication, and nonrepudiation. Digital signature cannot provide confidentiality for the message. If confidentiality is needed, a cryptosystem must be applied over the scheme.

- A digital signature needs an asymmetric-key system.
- In entity authentication, a claimant proves her identity to the verifier by using one of the three kinds of witnesses: something known, something possessed, or something inherent.
- In password-based authentication, the claimant uses a string of characters as something she knows.
- Password-based authentication can be divided into two broad categories: fixed and one-time.
- In challenge-response authentication, the claimant proves that she knows a secret without actually sending it.
- Challenge-response authentication can be divided into four categories: symmetric-key ciphers, keyed-hash functions, asymmetric-key ciphers, and digital signature.
- A key distribution center (KDC) is a trusted third party that assigns a symmetric key to two parties.
- KDC creates a secret key only between a member and the center. The secret key between members needs to be created as a session key when two members contact KDC.
- Kerberos is a popular session key creator protocol that requires an authentication server and a ticket-granting server.
- A certification authority (CA) is a federal or state organization that binds a public key to an entity and issues a certificate.
- A public-key infrastructure (PKI) is a hierarchical system to answer queries about key certification.

31.11 PRACTICE SET

Review Questions

1. What is a nonce?
2. What is the N^2 problem?
3. Name a protocol that uses a KDC for user authentication.
4. What is the purpose of the Kerberos authentication server?
5. What is the purpose of the Kerberos ticket-granting server?
6. What is the purpose of X.509?
7. What is a certification authority?
8. What are some advantages and disadvantages of using long passwords?
9. We discussed fixed and one-time passwords as two extremes. What about frequently changed passwords? How do you think this scheme can be implemented? What are the advantages and disadvantages?
10. How can a system prevent a guessing attack on a password? How can a bank prevent PIN guessing if someone has found or stolen a bank card and tried to use it?

Exercises

11. A message is made of 10 numbers between 00 and 99. A hash algorithm creates a digest out of this message by adding all numbers modulo 100. The resulting digest is a number between 00 and 99. Does this algorithm meet the first criterion of a hash algorithm? Does it meet the second criterion? Does it meet the third criterion?
12. A message is made of 100 characters. A hash algorithm creates a digest out of this message by choosing characters 1, 11, 21, . . ., and 91. The resulting digest has 10 characters. Does this algorithm meet the first criterion of a hash algorithm? Does it meet the second criterion? Does it meet the third criterion?
13. A hash algorithm creates a digest of N bits. How many different digests can be created from this algorithm?
14. At a party, which is more probable, a person with a birthday on a particular day or two (or more) persons having the same birthday?
15. How is the solution to Exercise 14 related to the second and third criteria of a hashing function?
16. Which one is more feasible, a fixed-size digest or a variable-size digest? Explain your answer.
17. A message is 20,000 characters. We are using a digest of this message using SHA-1. After creating the digest, we decided to change the last 10 characters. Can we say how many bits in the digest will be changed?
18. Are the processes of creating a MAC and of signing a hash the same? What are the differences?
19. When a person uses a money machine to get cash, is this a message authentication, an entity authentication, or both?
20. Change Figure 31.14 to provide two-way authentication (Alice for Bob and Bob for Alice).
21. Change Figure 31.16 to provide two-way authentication (Alice for Bob and Bob for Alice).
22. Change Figure 31.17 to provide two-way authentication (Alice for Bob and Bob for Alice).
23. Change Figure 31.18 to provide two-way authentication (Alice for Bob and Bob for Alice).
24. In a university, a student needs to encrypt her password (with a unique symmetric key) before sending it when she logs in. Does encryption protect the university or the student? Explain your answer.
25. In Exercise 24, does it help if the student appends a timestamp to the password before encryption? Explain your answer.
26. In Exercise 24, does it help if a student has a list of passwords and uses a different one each time?
27. In Figure 31.20, what happens if KDC is down?
28. In Figure 31.21, what happens if the AS is down? What happens if the TGS is down? What happens if the main server is down?
29. In Figure 31.26, what happens if the trusted center is down?

30. Add a symmetric-key encryption/decryption layer to Figure 31.11 to provide privacy.
31. Add an asymmetric-key encryption/decryption layer to Figure 31.11 to provide privacy.

Research Activities

32. There is a hashing algorithm called MD5. Find the difference between this algorithm and SHA-1.
33. There is a hashing algorithm called RIPEMD-160. Find the difference between this algorithm and SHA-1.
34. Compare MD5, SHA-1, and RIPEMD-160.
35. Find some information about RSA digital signature.
36. Find some information about DSS digital signature.

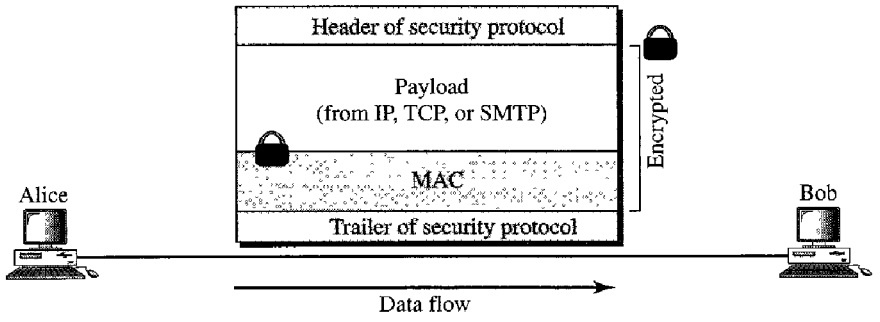
CHAPTER 32

Security in the Internet: IPSec, SSL/TLS, PGP, VPN, and Firewalls

In this chapter, we want to show how certain security aspects, particularly privacy and message authentication, can be applied to the network, transport, and application layers of the Internet model. We briefly show how the IPSec protocol can add authentication and confidentiality to the IP protocol, how SSL (or TLS) can do the same for the TCP protocol, and how PGP can do it for the SMTP protocol (e-mail).

In all these protocols, there are some common issues that we need to consider. First, we need to create a MAC. Then we need to encrypt the message and, probably, the MAC. This means, that with some minor variations, the three protocols discussed in this chapter take a packet from the appropriate layer and create a new packet which is authenticated and encrypted. Figure 32.1 shows this general idea.

Figure 32.1 *Common structure of three security protocols*



Note that the header or the trailer of the security protocol may or may not be included in the encryption process. Note also that some protocols may need more information in the secured packet; the figure shows only the general idea.

One common issue in all these protocols is *security parameters*. Even the simplified structure in Figure 32.1 suggests that Alice and Bob need to know several pieces of information, security parameters, before they can send secured data to each other. In particular, they need to know which algorithms to use for authentication and encryption/decryption.

Even if these algorithms can be predetermined for everyone in the world, which they are not as we will see, Bob and Alice still need at least two keys: one for the MAC and one for encryption/decryption. In other words, the complexity of these protocols lies not in the way the MAC data are calculated or the way encryption is performed; it lies in the fact that before calculating the MAC and performing encryption, we need to create a set of security parameters between Alice and Bob.

At first glance, it looks as if the use of any of these protocols must involve an infinite number of steps. To send secured data, we need a set of security parameters. The secure exchange of security parameters needs a second set of security parameters. The secure exchange of the second set of security parameters needs a third set of security parameters. And so on ad infinitum.

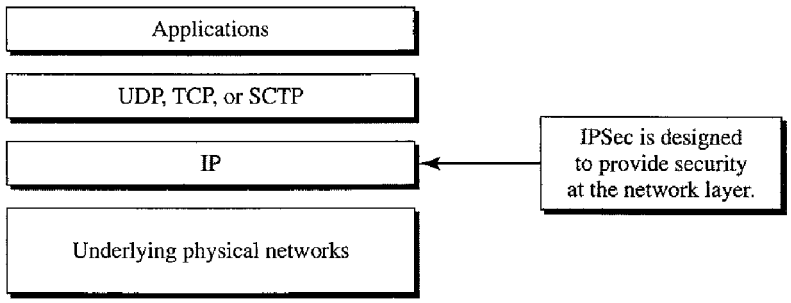
To limit the steps, we can use public-key cryptography if each person has a private and public key pair. The number of steps can be reduced to one or two. In the one-step version, we can use session keys to create the MAC and encrypt both data and MAC. The session keys and the list of algorithms can be sent with the packet but encrypted by using public-key ciphers. In the two-step version, we first establish the security parameters by using public-key ciphers. We then use the security parameters to securely send actual data. One of the three protocols, PGP, uses the first approach; the other two protocols, IPSec and SSL/TLS, use the second.

We also discuss a common protocol, the virtual private network (VPN), that uses the IPSec. At the end of the chapter, we discuss the firewall, a mechanism for preventing the attack on the network of the organization.

32.1 IPSecurity (IPSec)

IPSecurity (IPSec) is a collection of protocols designed by the Internet Engineering Task Force (IETF) to provide security for a packet at the network level. IPSec helps to create authenticated and confidential packets for the IP layer as shown in Figure 32.2.

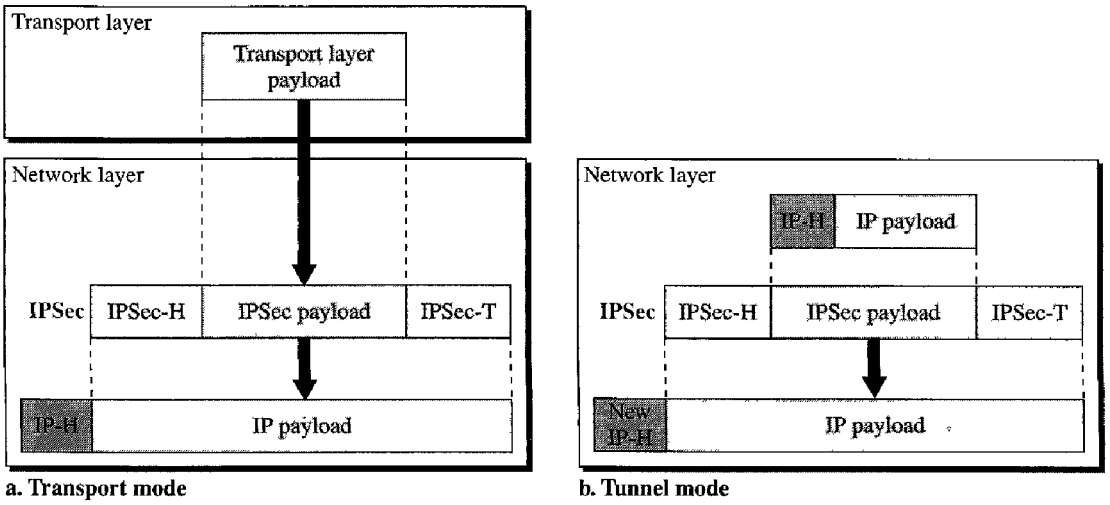
Figure 32.2 *TCP/IP protocol suite and IPSec*



Two Modes

IPSec operates in one of two different modes: the transport mode or the tunnel mode as shown in Figure 32.3.

Figure 32.3 *Transport mode and tunnel modes of IPSec protocol*



Transport Mode

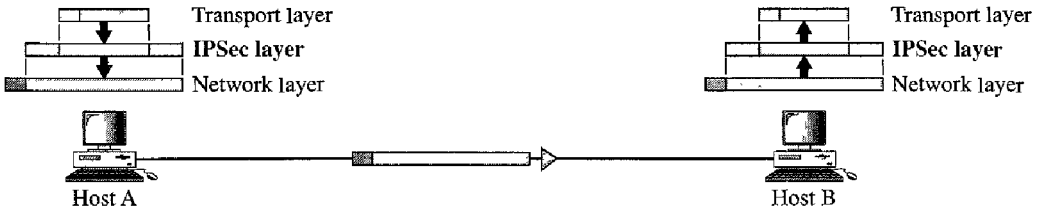
In the **transport mode,** IPSec protects what is delivered from the transport layer to the network layer. In other words, the transport mode protects the network layer payload, the payload to be encapsulated in the network layer.

Note that the transport mode does not protect the IP header. In other words, the transport mode does not protect the whole IP packet; it protects only the packet from the transport layer (the IP layer payload). In this mode, the IPSec header and trailer are added to the information coming from the transport layer. The IP header is added later.

IPSec in the transport mode does not protect the IP header; it only protects the information coming from the transport layer.

The transport mode is normally used when we need host-to-host (end-to-end) protection of data. The sending host uses IPSec to authenticate and/or encrypt the payload delivered from the transport layer. The receiving host uses IPSec to check the authentication and/or decrypt the IP packet and deliver it to the transport layer. Figure 32.4 shows this concept.

Figure 32.4 *Transport mode in action*

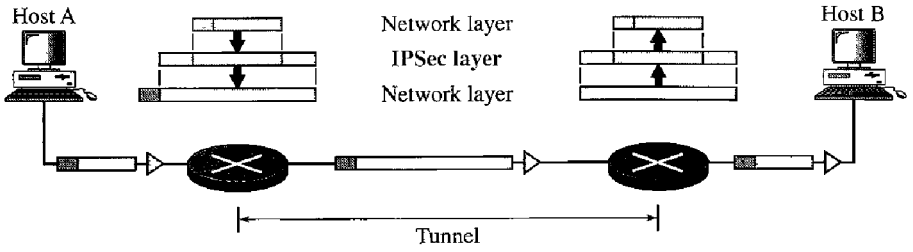


Tunnel Mode

In the **tunnel mode,** IPSec protects the entire IP packet. It takes an IP packet, including the header, applies IPSec security methods to the entire packet, and then adds a new IP header as shown in Figure 32.5.

The new IP header, as we will see shortly, has different information than the original IP header. The tunnel mode is normally used between two routers, between a host and a router, or between a router and a host as shown in Figure 32.5.

Figure 32.5 *Tunnel mode in action*



In other words, we use the tunnel mode when either the sender or the receiver is not a host. The entire original packet is protected from intrusion between the sender and the receiver. It's as if the whole packet goes through an imaginary tunnel.

IPSec in tunnel mode protects the original IP header.

Two Security Protocols

IPSec defines two protocols—the Authentication Header (AH) Protocol and the Encapsulating Security Payload (ESP) Protocol—to provide authentication and/or encryption for packets at the IP level.

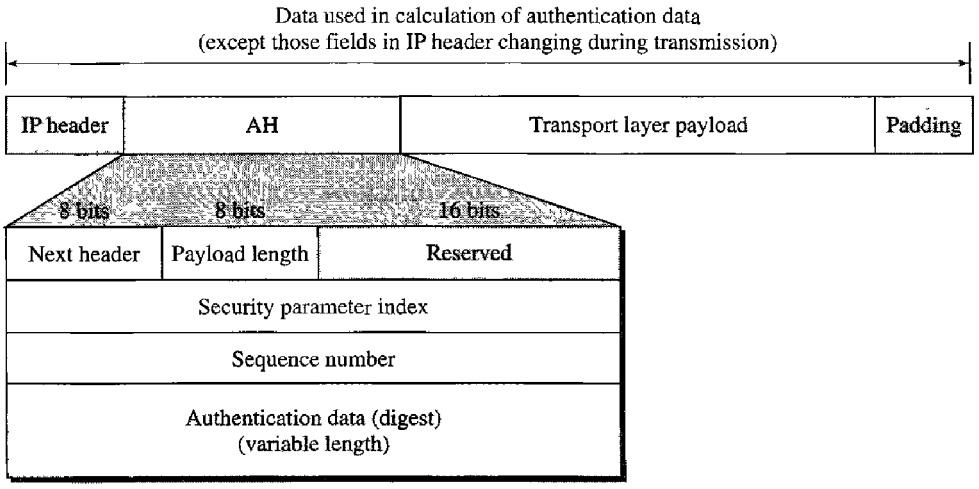
Authentication Header (AH)

The **Authentication Header (AH) Protocol** is designed to authenticate the source host and to ensure the integrity of the payload carried in the IP packet. The protocol uses a hash function and a symmetric key to create a message digest; the digest is inserted in the authentication header. The AH is then placed in the appropriate location based on the mode (transport or tunnel). Figure 32.6 shows the fields and the position of the authentication header in the transport mode.

When an IP datagram carries an authentication header, the original value in the protocol field of the IP header is replaced by the value 51. A field inside the authentication header (the next header field) holds the original value of the protocol field (the type of payload being carried by the IP datagram). The addition of an authentication header follows these steps:

1. An authentication header is added to the payload with the authentication data field set to zero.

Figure 32.6 *Authentication Header (AH) Protocol in transport mode*



2. Padding may be added to make the total length even for a particular hashing algorithm.
3. Hashing is based on the total packet. However, only those fields of the IP header that do not change during transmission are included in the calculation of the message digest (authentication data).
4. The authentication data are inserted in the authentication header.
5. The IP header is added after the value of the protocol field is changed to 51.

A brief description of each field follows:

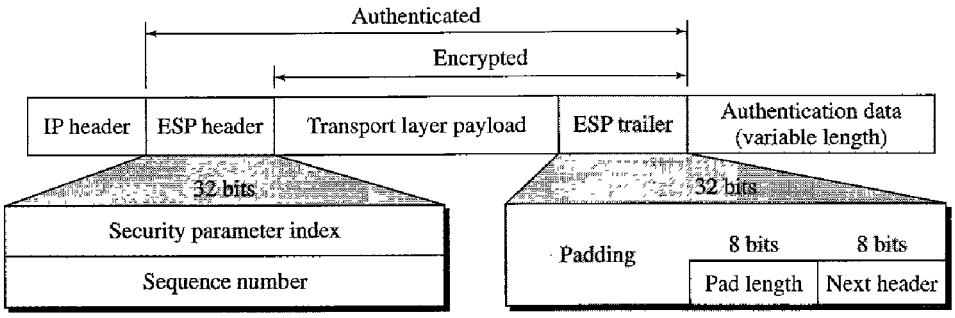
- **Next header.** The 8-bit next-header field defines the type of payload carried by the IP datagram (such as TCP, UDP, ICMP, or OSPF). It has the same function as the protocol field in the IP header before encapsulation. In other words, the process copies the value of the protocol field in the IP datagram to this field. The value of the protocol field in the new IP datagram is now set to 51 to show that the packet carries an authentication header.
- **Payload length.** The name of this 8-bit field is misleading. It does not define the length of the payload; it defines the length of the authentication header in 4-byte multiples, but it does not include the first 8 bytes.
- **Security parameter index.** The 32-bit security parameter index (SPI) field plays the role of a virtual-circuit identifier and is the same for all packets sent during a connection called a security association (discussed later).
- **Sequence number.** A 32-bit sequence number provides ordering information for a sequence of datagrams. The sequence numbers prevent a playback. Note that the sequence number is not repeated even if a packet is retransmitted. A sequence number does not wrap around after it reaches 2^{32}; a new connection must be established.
- **Authentication data.** Finally, the authentication data field is the result of applying a hash function to the entire IP datagram except for the fields that are changed during transit (e.g., time-to-live).

The AH Protocol provides source authentication and data integrity, but not privacy.

Encapsulating Security Payload (ESP)

The AH Protocol does not provide privacy, only source authentication and data integrity. IPSec later defined an alternative protocol that provides source authentication, integrity, and privacy called **Encapsulating Security Payload (ESP).** ESP adds a header and trailer. Note that ESP's authentication data are added at the end of the packet which makes its calculation easier. Figure 32.7 shows the location of the ESP header and trailer.

Figure 32.7 *Encapsulation Security Payload (ESP) Protocol in transport mode*



When an IP datagram carries an ESP header and trailer, the value of the protocol field in the IP header is 50. A field inside the ESP trailer (the next-header field) holds the original value of the protocol field (the type of payload being carried by the IP datagram, such as TCP or UDP). The ESP procedure follows these steps:

1. An ESP trailer is added to the payload.
2. The payload and the trailer are encrypted.
3. The ESP header is added.
4. The ESP header, payload, and ESP trailer are used to create the authentication data.
5. The authentication data are added to the end of the ESP trailer.
6. The IP header is added after the protocol value is changed to 50.

The fields for the header and trailer are as follows:

- **Security parameter index.** The 32-bit security parameter index field is similar to that defined for the AH Protocol.
- **Sequence number.** The 32-bit sequence number field is similar to that defined for the AH Protocol.
- **Padding.** This variable-length field (0 to 255 bytes) of 0s serves as padding.
- **Pad length.** The 8-bit pad length field defines the number of padding bytes. The value is between 0 and 255; the maximum value is rare.
- **Next header.** The 8-bit next-header field is similar to that defined in the AH Protocol. It serves the same purpose as the protocol field in the IP header before encapsulation.

- **Authentication data.** Finally, the authentication data field is the result of applying an authentication scheme to parts of the datagram. Note the difference between the authentication data in AH and ESP. In AH, part of the IP header is included in the calculation of the authentication data; in ESP, it is not.

ESP provides source authentication, data integrity, and privacy.

IPv4 and IPv6

IPSec supports both IPv4 and IPv6. In IPv6, however, AH and ESP are part of the extension header.

AH Versus ESP

The ESP Protocol was designed after the AH Protocol was already in use. ESP does whatever AH does with additional functionality (privacy). The question is, Why do we need AH? The answer is, We don't. However, the implementation of AH is already included in some commercial products, which means that AH will remain part of the Internet until the products are phased out.

Services Provided by IPSec

The two protocols, AH and ESP, can provide several security services for packets at the network layer. Table 32.1 shows the list of services available for each protocol.

Table 32.1 *IPSec services*

Services	*AH*	*ESP*
Access control	Yes	Yes
Message authentication (message integrity)	Yes	Yes
Entity authentication (data source authentication)	Yes	Yes
Confidentiality	No	Yes
Replay attack protection	Yes	Yes

Access Control IPSec provides access control indirectly by using a Security Association Database (SADB) as we will see in the next section. When a packet arrives at a destination, and there is no security association already established for this packet, the packet is discarded.

Message Authentication The integrity of the message is preserved in both AH and ESP by using authentication data. A digest of data is created and sent by the sender to be checked by the receiver.

Entity Authentication The security association and the keyed-hashed digest of the data sent by the sender authenticate the sender of the data in both AH and ESP.

Confidentiality The encryption of the message in ESP provides confidentiality. AH, however, does not provide confidentiality. If confidentiality is needed, one should use ESP instead of AH.

Replay Attack Protection In both protocols, the **replay attack** is prevented by using sequence numbers and a sliding receiver window. Each IPSec header contains a unique sequence number when the security association is established. The number starts from 0 and increases until the value reaches $2^{32} - 1$ (the size of the sequence number field is 32 bits). When the sequence number reaches the maximum, it is reset to zero and, at the same time, the old security association (see the next section) is deleted and a new one is established. To prevent processing of duplicate packets, IPSec mandates the use of a fixed-size window at the receiver. The size of the window is determined by the receiver with a default value of 64.

Security Association

As we mentioned in the introduction to the chapter, each of three protocols we discuss in this chapter (IPSec, SSL/TLS, and PGP) needs a set of security parameters before it can be operative. In IPSec, the establishment of the security parameters is done via a mechanism called **security association (SA).**

IP, as we have seen, is a connectionless protocol: Each datagram is independent of the others. For this type of communication, the security parameters can be established in one of three ways.

1. Security parameters related to each datagram can be included in each datagram. The designer of IPSec did not choose this option probably because of overhead. Adding security parameters to each datagram creates a large overhead, particularly if the datagram is fragmented several times during its journey.
2. A set of security parameters can be established for each datagram. This means that before each datagram is transmitted, a set of packets needs to be exchanged between the sender and receiver to establish security parameters. This is probably less efficient than the first choice, and it is not used in IPSec.
3. IPSec uses the third choice. A set of security parameters can be established between a sender and a particular receiver the first time the sender has a datagram to send to that particular receiver. The set can be saved for future transmission of IP packets to the same receiver.

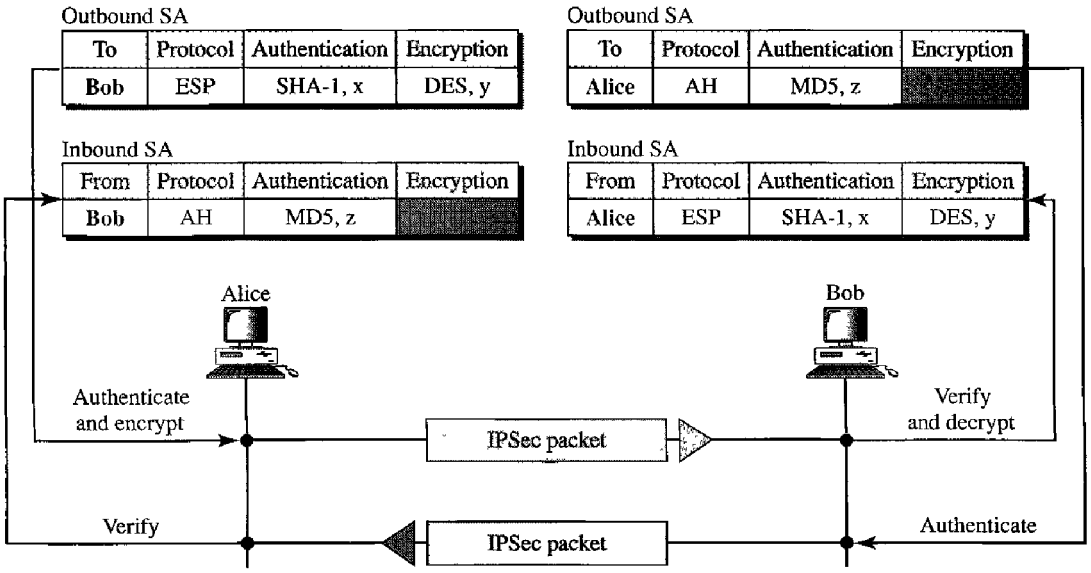
Security association is a very important aspect of IPSec. Using security association, IPSec changes a connectionless protocol, IP, to a connection-oriented protocol. We can think of an association as a connection. We can say that when Alice and Bob agree upon a set of security parameters between them, they have established a logical connection between themselves (which is called association). However, they may not use this connection all the time. After establishing the connection, Alice can send a datagram to Bob today, another datagram a few days later, and so on. The logical connection is there and ready for sending a secure datagram. Of course, they can break the connection, or they can establish a new one after a while (which is a more secure way of communication).

A Simple Example

A security association is a very complex set of pieces of information. However, we can show the simplest case in which Alice wants to have an association with Bob for use in

a two-way communication. Alice can have an outbound association (for datagrams to Bob) and an inbound association (for datagrams from Bob). Bob can have the same. In this case, the security associations are reduced to two small tables for both Alice and Bob as shown in Figure 32.8.

Figure 32.8 *Simple inbound and outbound security associations*



The figure shows that when Alice needs to send a datagram to Bob, she uses the ESP Protocol of IPSec. Authentication is done by using SHA-1 with key x. The encryption is done by using DES with key y. When Bob needs to send a datagram to Alice, he uses the AH Protocol of IPSec. Authentication is done by using MD5 with key z. Note that the inbound association for Bob is the same as the outbound association for Alice, and vice versa.

Security Association Database (SADB)

A security association can be very complex. This is particularly true if Alice wants to send messages to many people and Bob needs to receive messages from many people. In addition, each site needs to have both inbound and outbound SAs to allow bidirectional communication. In other words, we need a set of SAs that can be collected into a database. This database is called the **security association database (SADB).** The database can be thought of as a two-dimensional table with each row defining a single SA. Normally, there are two SADBs, one inbound and one outbound.

Security Parameter Index

To distinguish one association from the other, each association is identified by a parameter called the **security parameter index (SPI).** This parameter, in conjunction with the destination address (outbound) or source address (inbound) and protocol (AH or ESP), uniquely defines an association.

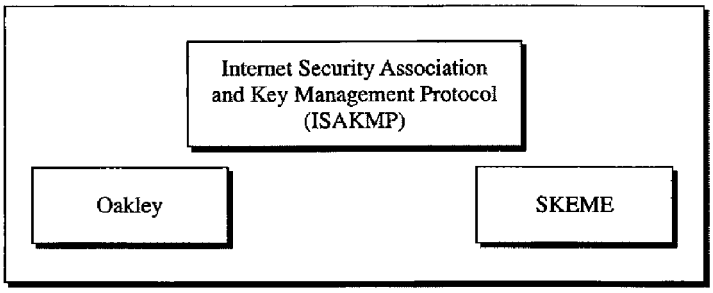
Internet Key Exchange (IKE)

Now we come to the last part of the puzzle—how SADBs are created. The **Internet Key Exchange (IKE)** is a protocol designed to create both inbound and outbound security associations in SADBs.

IKE creates SAs for IPSec.

IKE is a complex protocol based on three other protocols—Oakley, SKEME, and ISAKMP—as shown in Figure 32.9.

Figure 32.9 *IKE components*



The **Oakley** Protocol was developed by Hilarie Orman. It is a key creation protocol based on the Diffie-Hellman key-exchange method, but with some improvements. Oakley is a free-formatted protocol in the sense that it does not define the format of the message to be exchanged.

SKEME, designed by Hugo Krawcyzk, is another protocol for key exchange. It uses public-key encryption for entity authentication in a key-exchange protocol.

The **Internet Security Association and Key Management Protocol (ISAKMP)** is a protocol designed by the National Security Agency (NSA) that actually implements the exchanges defined in IKE. It defines several packets, protocols, and parameters that allow the IKE exchanges to take place in standardized, formatted messages to create SAs.

One may ask how ISAKMP is carried from the sender to the receiver. This protocol is designed so as to be applicable with any underlying protocol. For example, the packet can be used as the payload in the network layer or transport layer. When we use IPSec, it is natural that this packet be considered as a payload for the IP protocol and carried in the datagram. Now the next question is, How are the datagrams that carry ISAKMP securely exchanged? The answer is that there is no need. There is nothing in the ISAKMP packets that needs to be secured.

Virtual Private Network

Virtual private network (VPN) is a technology that is gaining popularity among large organizations that use the global Internet for both intra- and interorganization communication, but require privacy in their internal communications. We discuss VPN here because it uses the IPSec Protocol to apply security to the IP datagrams.

Private Networks

A private network is designed for use inside an organization. It allows access to shared resources and, at the same time, provides privacy. Before we discuss some aspects of these networks, let us define two commonly used, related terms: *intranet* and *extranet.*

Intranet An **intranet** is a private network (LAN) that uses the Internet model. However, access to the network is limited to the users inside the organization. The network uses application programs defined for the global Internet, such as HTTP, and may have Web servers, print servers, file servers, and so on.

Extranet An **extranet** is the same as an intranet with one major difference: Some resources may be accessed by specific groups of users outside the organization under the control of the network administrator. For example, an organization may allow authorized customers access to product specifications, availability, and online ordering. A university or a college can allow distance learning students access to the computer lab after passwords have been checked.

Addressing A private network that uses the Internet model must use IP addresses. Three choices are available:

1. The network can apply for a set of addresses from the Internet authorities and use them without being connected to the Internet. This strategy has an advantage. If in the future the organization decides to be connected to the Internet, it can do so with relative ease. However, there is also a disadvantage: The address space is wasted in the meantime.
2. The network can use any set of addresses without registering with the Internet authorities. Because the network is isolated, the addresses do not have to be unique. However, this strategy has a serious drawback: Users might mistakenly confuse the addresses as part of the global Internet.
3. To overcome the problems associated with the first and second strategies, the Internet authorities have reserved three sets of addresses, shown in Table 32.2.

Table 32.2 *Addresses for private networks*

Prefix	*Range*	*Total*
10/8	10.0.0.0 to 10.255.255.255	2^{24}
172.16/12	172.16.0.0 to 172.31.255.255	2^{20}
192.168/16	192.168.0.0 to 192.168.255.255	2^{16}

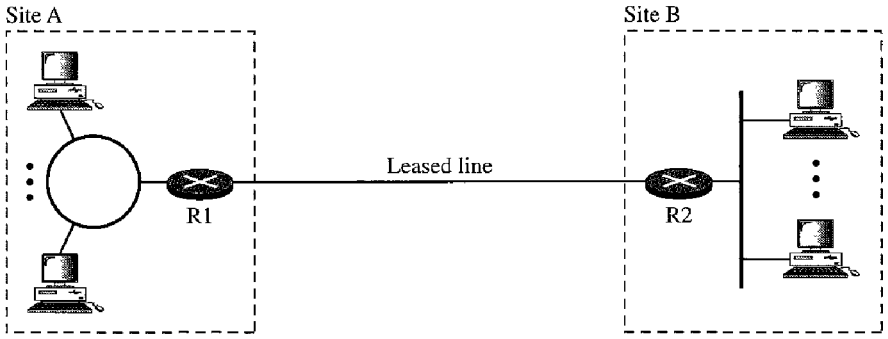
Any organization can use an address out of this set without permission from the Internet authorities. Everybody knows that these reserved addresses are for private networks. They are unique inside the organization, but they are not unique globally. No router will forward a packet that has one of these addresses as the destination address.

Achieving Privacy

To achieve privacy, organizations can use one of three strategies: private networks, hybrid networks, and virtual private networks.

Private Networks An organization that needs privacy when routing information inside the organization can use a **private network** as discussed previously. A small organization with one single site can use an isolated LAN. People inside the organization can send data to one another that totally remain inside the organization, secure from outsiders. A larger organization with several sites can create a private internet. The LANs at different sites can be connected to each other by using routers and leased lines. In other words, an internet can be made out of private LANs and private WANs. Figure 32.10 shows such a situation for an organization with two sites. The LANs are connected to each other by routers and one leased line.

Figure 32.10 *Private network*



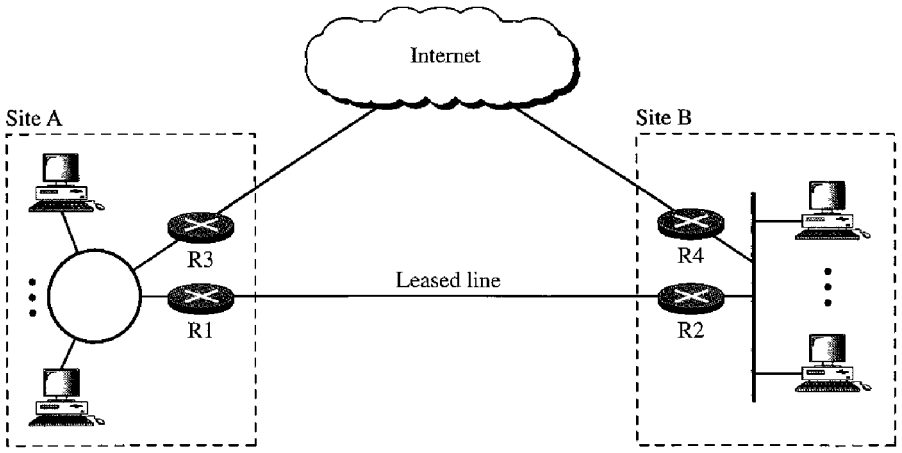
In this situation, the organization has created a private internet that is totally isolated from the global Internet. For end-to-end communication between stations at different sites, the organization can use the Internet model. However, there is no need for the organization to apply for IP addresses with the Internet authorities. It can use private IP addresses. The organization can use any IP class and assign network and host addresses internally. Because the internet is private, duplication of addresses by another organization in the global Internet is not a problem.

Hybrid Networks Today, most organizations need to have privacy in intraorganization data exchange, but, at the same time, they need to be connected to the global Internet for data exchange with other organizations. One solution is the use of a **hybrid network.** A hybrid network allows an organization to have its own private internet and, at the same time, access to the global Internet. Intraorganization data are routed through the private internet; interorganization data are routed through the global Internet. Figure 32.11 shows an example of this situation.

An organization with two sites uses routers R1 and R2 to connect the two sites privately through a leased line; it uses routers R3 and R4 to connect the two sites to the rest of the world. The organization uses global IP addresses for both types of communication. However, packets destined for internal recipients are routed only through routers R1 and R2. Routers R3 and R4 route the packets destined for outsiders.

Virtual Private Networks Both private and hybrid networks have a major drawback: cost. Private wide-area networks (WANs) are expensive. To connect several sites, an organization needs several leased lines, which means a high monthly fee. One solution

Figure 32.11 *Hybrid network*

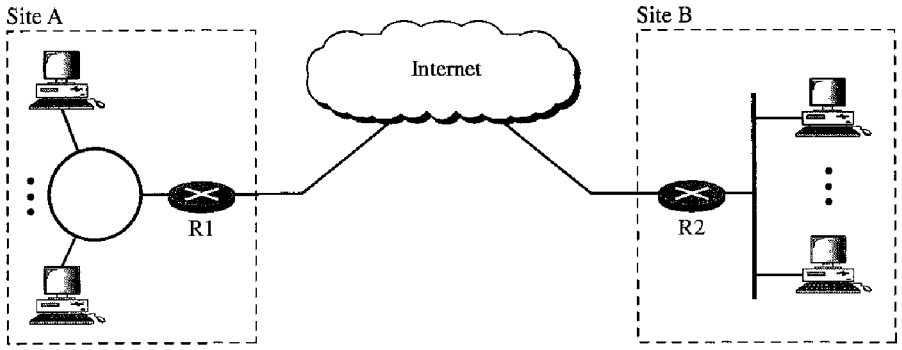


is to use the global Internet for both private and public communications. A technology called virtual private network allows organizations to use the global Internet for both purposes.

VPN creates a network that is private but virtual. It is private because it guarantees privacy inside the organization. It is virtual because it does not use real private WANs; the network is physically public but virtually private.

Figure 32.12 shows the idea of a virtual private network. Routers R1 and R2 use VPN technology to guarantee privacy for the organization.

Figure 32.12 *Virtual private network*

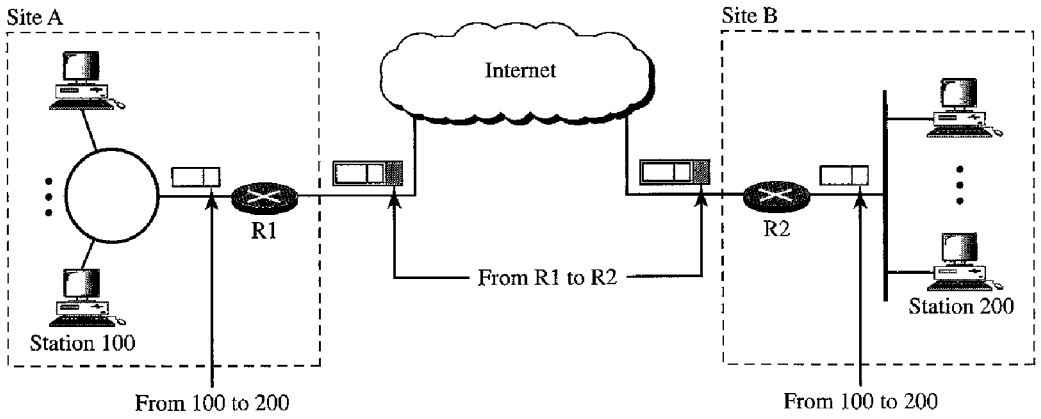


VPN Technology

VPN technology uses IPSec in the tunnel mode to provide authentication, integrity, and privacy.

Tunneling To guarantee privacy and other security measures for an organization, VPN can use the IPSec in the tunnel mode. In this mode, each IP datagram destined for private use in the organization is encapsulated in another datagram. To use IPSec in **tunneling,** the VPNs need to use two sets of addressing, as shown in Figure 32.13.

Figure 32.13 *Addressing in a VPN*



The public network (Internet) is responsible for carrying the packet from R1 to R2. Outsiders cannot decipher the contents of the packet or the source and destination addresses. Deciphering takes place at R2, which finds the destination address of the packet and delivers it.

32.2 SSL/TLS

A transport layer security provides end-to-end security services for applications that use a reliable transport layer protocol such as TCP. The idea is to provide security services for transactions on the Internet. For example, when a customer shops online, the following security services are desired:

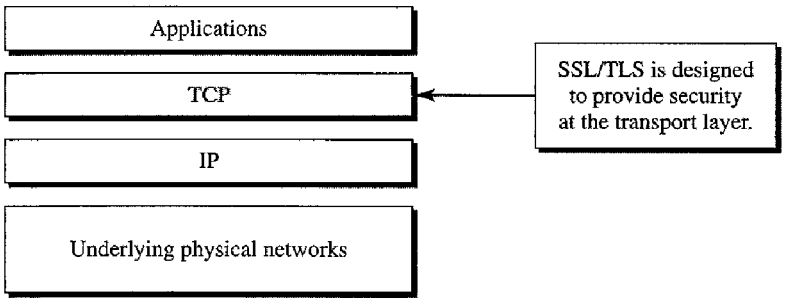
1. The customer needs to be sure that the server belongs to the actual vendor, not an imposter. The customer does not want to give an imposter her credit card number (entity authentication). Likewise, the vendor needs to authenticate the customer.
2. The customer and the vendor need to be sure that the contents of the message are not modified during transition (message integrity).
3. The customer and the vendor need to be sure that an imposter does not intercept sensitive information such as a credit card number (confidentiality).

Two protocols are dominant today for providing security at the transport layer: the Secure Sockets Layer (SSL) Protocol and the Transport Layer Security (TLS) Protocol. The latter is actually an IETF version of the former. First we discuss SSL, then we briefly mention the main differences between SSL and TLS. Figure 32.14 shows the position of SSL and TLS in the Internet model.

SSL Services

Secure Socket Layer (SSL) is designed to provide security and compression services to data generated from the application layer. Typically, SSL can receive data from any application layer protocol, but usually the protocol is HTTP. The data received from the

Figure 32.14 *Location of SSL and TLS in the Internet model*



application are compressed (optional), signed, and encrypted. The data are then passed to a reliable transport layer protocol such as TCP. Netscape developed SSL in 1994. Versions 2 and 3 were released in 1995. In this chapter, we discuss SSLv3. SSL provides several services on data received from the application layer.

Fragmentation

First, SSL divides the data into blocks of 2^{14} bytes or less.

Compression

Each fragment of data is compressed by using one of the lossless compression methods negotiated between the client and server. This service is optional.

Message Integrity

To preserve the integrity of data, SSL uses a keyed-hash function to create a MAC.

Confidentiality

To provide confidentiality, the original data and the MAC are encrypted using symmetric-key cryptography.

Framing

A header is added to the encrypted payload. The payload is then passed to a reliable transport layer protocol.

Security Parameters

When we discussed IPSec in the previous section, we mentioned that each of the two parties involved in data exchange needs to have a set of parameters for each association (SA). SSL has a similar goal, but a different approach. There are no SAs, but there are cipher suites and cryptographic secrets that together make the security parameters.

Cipher Suite

The combination of key exchange, hash, and encryption algorithms defines a **cipher suite** for each SSL session. Each suite starts with the term *SSL,* followed by the key-exchange

algorithm. The word *WITH* separates the key exchange algorithm from the encryption and hash algorithms. For example,

SSL_DHE_RSA_*WITH*_DES_CBC_SHA

defines DHE_RSA (ephemeral Diffie-Hellman with RSA digital signature) as the key exchange with DES_CBC as the encryption algorithm and SHA as the hash algorithm. Note that DH is fixed Diffie-Hellman, DHE is ephemeral Diffie-Hellman, and DH-anon is anonymous Diffie-Hellman. Table 32.3 shows the suites used in the United States. We have not included those that are used for export. Note that not all combinations of key-exchange algorithms (to establish keys for message authentication and encryption), encryption algorithms, and authentication algorithms are included in the cipher suite list. We have not defined or discussed several algorithms you can find in the table, but we wish to describe the whole picture so that the reader can have an idea of how general the suite is.

Table 32.3 *SSL cipher suite list*

Cipher Suite	*Key Exchange Algorithm*	*Encryption Algorithm*	*Hash Algorithm*
SSL_NULL_*WITH*_NULL_NULL	NULL	NULL	NULL
SSL_RSA_*WITH*_NULL_MD5	RSA	NULL	MD5
SSL_RSA_*WITH*_NULL_SHA	RSA	NULL	SHA
SSL_RSA_*WITH*_RC4_128_MD5	RSA	RC4_128	MD5
SSL_RSA_*WITH*_RC4_128_SHA	RSA	RC4_128	SHA
SSL_RSA_*WITH*_IDEA_CBC_SHA	RSA	IDEA_CBC	SHA
SSL_RSA_*WITH*_DES_CBC_SHA	RSA	DES_CBC	SHA
SSL_RSA_*WITH*_3DES_EDE_CBC_SHA	RSA	3DES_EDE_CBC	SHA
SSL_DH_anon_*WITH*_RC4_128_MD5	DH_anon	RC4_128	MD5
SSL_DH_anon_*WITH*_DES_CBC_SHA	DH_anon	DES_CBC	SHA
SSL_DH_anon_*WITH*_3DES_EDE_CBC_SHA	DH_anon	3DES_EDE_CBC	SHA
SSL_DHE_RSA_*WITH*_DES_CBC_SHA	DHE_RSA	DES_CBC	SHA
SSL_DHE_RSA_*WITH*_3DES_EDE_CBC_SHA	DHE_RSA	3DES_EDE_CBC	SHA
SSL_DHE_DSS_*WITH*_DES_CBC_SHA	DHE_DSS	DES_CBC	SHA
SSL_DHE_DSS_*WITH*_3DES_EDE_CBC_SHA	DHE_DSS	3DES_EDE_CBC	SHA
SSL_DH_RSA_*WITH*_DES_CBC_SHA	DH_RSA	DES_CBC	SHA
SSL_DH_RSA_*WITH*_3DES_EDE_CBC_SHA	DH_RSA	3DES_EDE_CBC	SHA
SSL_DH_DSS_*WITH*_DES_CBC_SHA	DH_DSS	DES_CBC	SHA
SSL_DH_DSS_*WITH*_3DES_EDE_CBC_SHA	DH_DSS	3DES_EDE_CBC	SHA
SSL_FORTEZZA_DMS_*WITH*_NULL_SHA	FORTEZZA_DMS	NULL	SHA
SSL_FORTEZZA_DMS_*WITH*_FORTEZZA_CBC_SHA	FORTEZZA_DMS	FORTEZZA_CBC	SHA
SSL_FORTEZZA_DMS_*WITH*_RC4_128_SHA	FORTEZZA_DMS	RC4_128	SHA

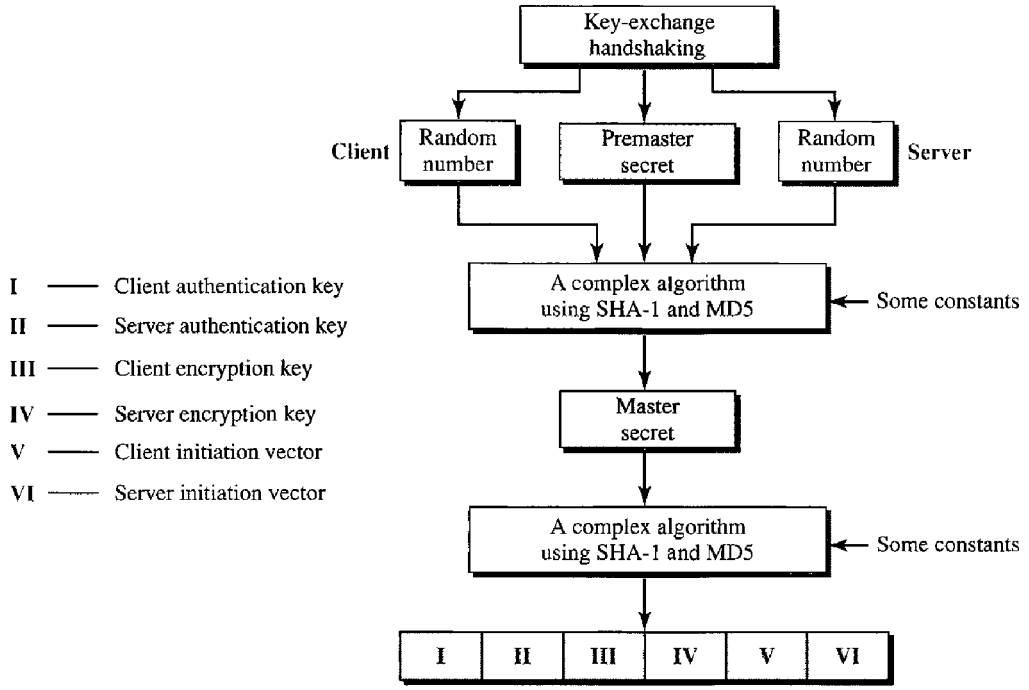
Cryptographic Secrets

The second part of security parameters is often referred to as cryptographic secrets. To achieve message integrity and confidentiality, SSL needs six cryptographic secrets, four keys, and two IVs.

The client and the server have six different cryptography secrets.

The process of creating these secrets is shown in Figure 32.15. The client needs one key for message authentication, one key for encryption, and one IV for block encryption. The server needs the same. SSL requires that the keys for one direction be different from those for the other direction. If there is an attack in one direction, the other direction is not affected. These parameters are generated by using a negotiation protocol, as we will see shortly.

Figure 32.15 *Creation of cryptographic secrets in SSL*



1. The client and server exchange two random numbers; one is created by the client and the other by the server.
2. The client and server exchange one **premaster secret** by using one of the key-exchange algorithms we discussed previously.
3. A 48-byte **master secret** is created from the premaster secret by applying two hash functions (SHA-1 and MD5).
4. The master secret is used to create variable-length secrets by applying the same set of hash functions and prepending with different constants.

Sessions and Connections

The nature of IP and TCP protocols is different. IP is a connectionless protocol; TCP is a connection-oriented protocol. An association in IPSec transforms the connectionless IP to a connection-oriented secured protocol. TCP is already connection-oriented.

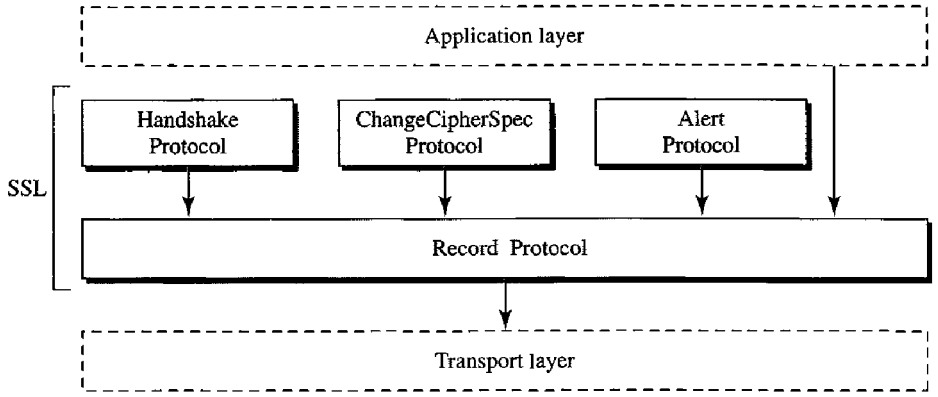
However, the designers of SSL decided that they needed two-levels of connectivity: **session** and **connection.** A session between two systems is an association that can last for a long time; a connection can be established and broken several times during a session.

Some of the security parameters are created during the session establishment and are in effect until the session is terminated (for example, cipher suite and master key). Some of the security parameters must be recreated (or occasionally resumed) for each connection (for example, six secrets).

Four Protocols

We have discussed the idea of SSL without showing how SSL accomplishes its tasks. SSL defines four protocols in two layers, as shown in Figure 32.16. The **Record Protocol** is the carrier. It carries messages from three other protocols as well as the data coming from the application layer. Messages from the Record Protocol are payloads to the transport layer, normally TCP. The **Handshake Protocol** provides security parameters for the Record Protocol. It establishes a cipher set and provides keys and security parameters. It also authenticates the server to the client and the client to the server, if needed. The **ChangeCipherSpec Protocol** is used for signaling the readiness of cryptographic secrets. The **Alert Protocol** is used to report abnormal conditions. We will briefly discuss these protocols in this section.

Figure 32.16 *Four SSL protocols*



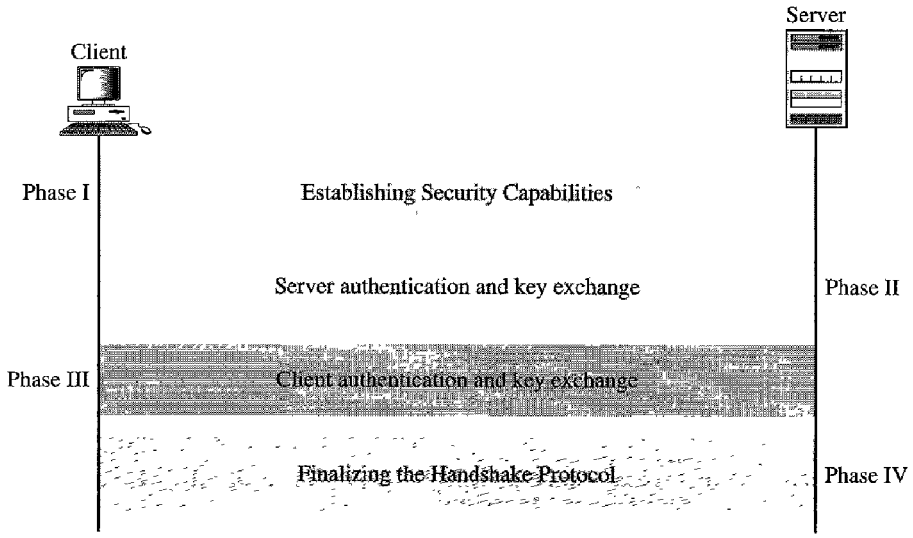
Handshake Protocol

The Handshake Protocol uses messages to negotiate the cipher suite, to authenticate the server to the client and the client to the server (if needed), and to exchange information for building the cryptographic secrets. The handshaking is done in four phases, as shown in Figure 32.17.

ChangeCipherSpec Protocol

We have seen that the negotiation of the cipher suite and the generation of cryptographic secrets are formed gradually during the Handshake Protocol. The question now is, When can the two parties use these parameter secrets? SSL mandates that the parties

Figure 32.17 *Handshake Protocol*



not use these parameters or secrets until they have sent or received a special message, the ChangeCipherSpec message, which is exchanged during the Handshake Protocol and defined in the ChangeCipherSpec Protocol. Before the exchange of any ChangeCipherSpec messages, only the pending columns have values.

Alert Protocol

SSL uses the Alert Protocol for reporting errors and abnormal conditions. It has only one message type, the alert message, that describes the problem and its level (warning or fatal).

Record Protocol

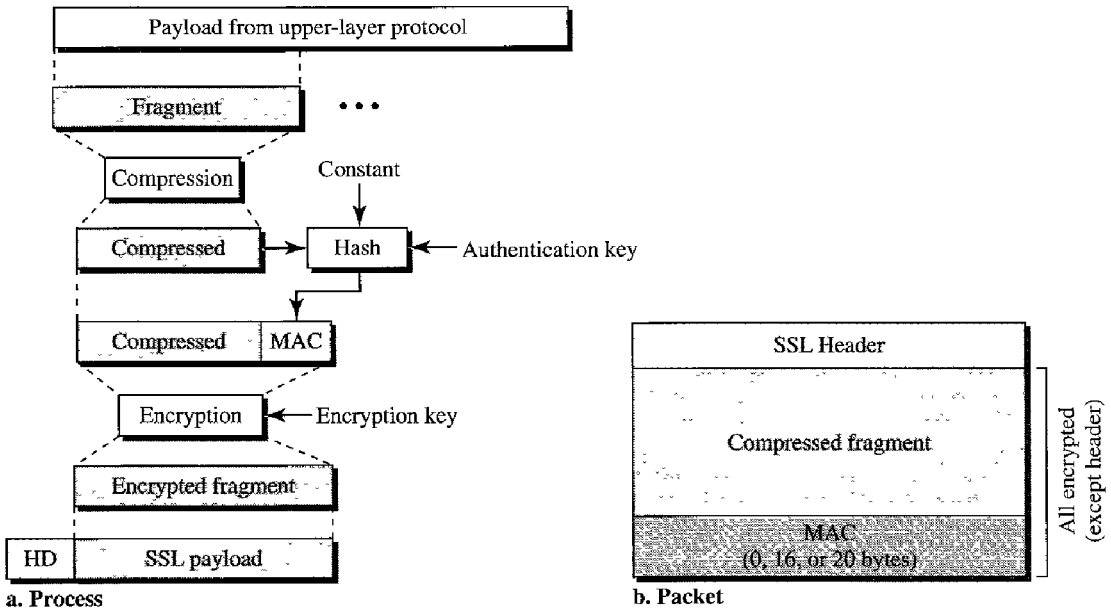
The Record Protocol carries messages from the upper layer (Handshake Protocol, ChangeCipherSpec Protocol, Alert Protocol, or application layer). The message is fragmented and optionally compressed; a MAC is added to the compressed message by using the negotiated hash algorithm. The compressed fragment and the MAC are encrypted by using the negotiated encryption algorithm. Finally, the SSL header is added to the encrypted message. Figure 32.18 shows this process at the sender. The process at the receiver is reversed.

Transport Layer Security

Transport Layer Security (TLS) is the IETF standard version of SSL. The two are very similar, with slight differences. We highlight the differences below:

- **Version.** The SSLv3.0 discussed in this section is compatible with TLSv1.0.
- **Cipher Suite.** TLS cipher suite does not support Fortezza.
- **Cryptography Secret.** There are several differences in the generation of cryptographic secrets. TLS uses a **pseudorandom function (PRF)** to create the master key and the key materials.

Figure 32.18 *Processing done by the Record Protocol*

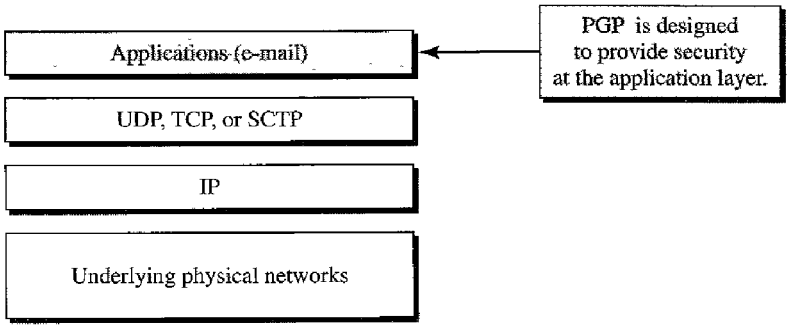


- **Alert Protocol.** TLS deletes some alert messages and adds some new ones.
- **Handshake Protocol.** The details of some messages have been changed in TLS.
- **Record Protocol.** Instead of using MAC, TLS uses the HMAC as defined in Chapter 31.

32.3 PGP

One of the protocols to provide security at the application layer is **Pretty Good Privacy (PGP).** PGP is designed to create authenticated and confidential e-mails. Figure 32.19 shows the position of PGP in the TCP/IP protocol suite.

Figure 32.19 *Position of PGP in the TCP/IP protocol suite*



Sending an e-mail is a one-time activity. The nature of this activity is different from those we have seen in the previous two sections. In IPSec or SSL, we assume that the two parties create a session between themselves and exchange data in both directions. In e-mail, there is no session. Alice and Bob cannot create a session. Alice sends a message to Bob; sometime later, Bob reads the message and may or may not send a reply. We discuss the security of a unidirectional message because what Alice sends to Bob is totally independent of what Bob sends to Alice.

Security Parameters

If e-mail is a one-time activity, how can the sender and receiver agree on the security parameters to use for e-mail security? If there is no session and no handshaking to negotiate the algorithms for encryption and authentication, how can the receiver know which algorithm the sender has chosen for each purpose? How can the receiver know the values of the keys used for encryption and authentication?

Phil Zimmerman, the designer and creator of PGP, has found a very elegant solution to the above questions. The security parameters need to be sent with the message.

In PGP, the sender of the message needs to include the identifiers of the algorithms used in the message as well as the values of the keys.

Services

PGP can provide several services based on the requirements of the user. An e-mail can use one or more of these services.

Plaintext

The simplest case is to send the e-mail message in plaintext (no service). Alice, the sender, composes a message and sends it to Bob, the receiver. The message is stored in Bob's mailbox until it is retrieved by him.

Message Authentication

Probably the next improvement is to let Alice sign the message. Alice creates a digest of the message and signs it with her private key. When Bob receives the message, he verifies the message by using Alice's public key. Two keys are needed for this scenario. Alice needs to know her private key; Bob needs to know Alice's public key.

Compression

A further improvement is to compress the message and digest to make the packet more compact. This improvement has no security benefit, but it eases the traffic.

Confidentiality with One-Time Session Key

As we discussed before, confidentiality in an e-mail system can be achieved by using conventional encryption with a one-time session key. Alice can create a session key, use the session key to encrypt the message and the digest, and send the key itself with the message. However, to protect the session key, Alice encrypts it with Bob's public key.

Code Conversion

Another service provided by PGP is code conversion. Most e-mail systems allow the message to consist of only ASCII characters. To translate other characters not in the ASCII set, PGP uses Radix 64 conversion. Each character to be sent (after encryption) is converted to Radix 64 code.

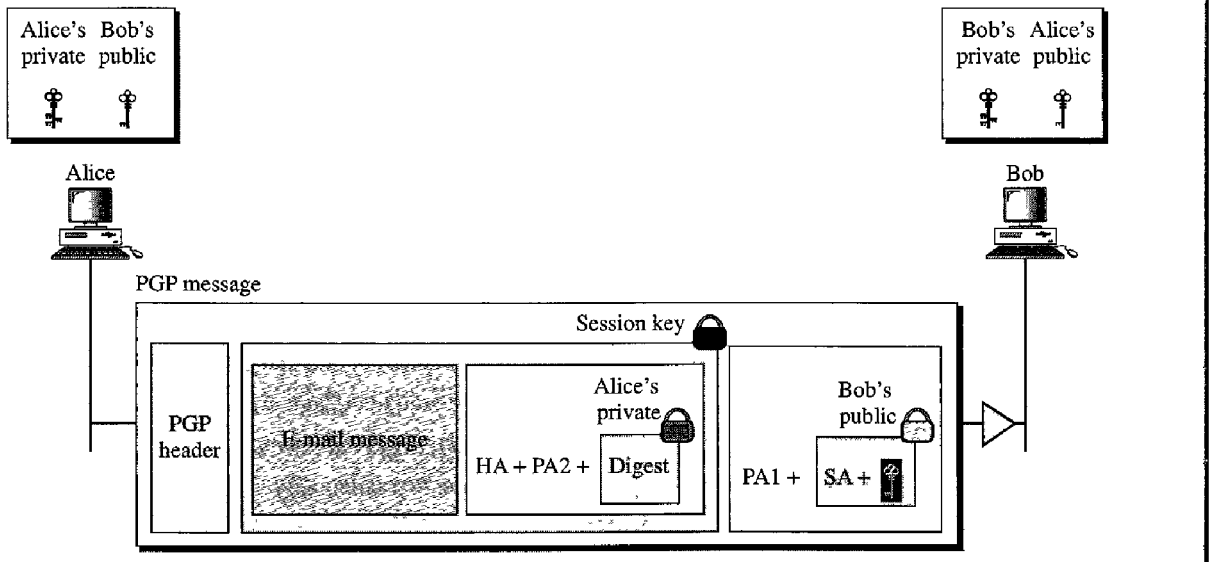
Segmentation

PGP allows segmentation of the message after it has been converted to Radix 64 to make each transmitted unit the uniform size allowed by the underlying e-mail protocol.

A Scenario

Let us describe a scenario that combines some of these services, authentication and confidentiality. The whole idea of PGP is based on the assumption that a group of people who need to exchange e-mail messages trust one another. Everyone in the group somehow knows (with a degree of trust) the public key of any other person in the group. Based on this single assumption, Figure 32.20 shows a simple scenario in which an authenticated and encrypted message is sent from Alice to Bob.

Figure 32.20 *A scenario in which an e-mail message is authenticated and encrypted*



PA1: Public-key algorithm 1 (for encrypting session key)
PA2: Public-key algorithm (for encrypting the digest)
SA: Symmetric-key algorithm identification (for encrypting message and digest)
HA: Hash algorithm identification (for creating digest)

Sender Site

The following shows the steps used in this scenario at Alice's site:

1. Alice creates a session key (for symmetric encryption/decryption) and concatenates it with the identity of the algorithm which will use this key. The result is encrypted

with Bob's public key. Alice adds the identification of the public-key algorithm used above to the encrypted result. This part of the message contains three pieces of information: the session key, the symmetric encryption/decryption algorithm to be used later, and the asymmetric encryption/decryption algorithm that was used for this part.

2.
   a. Alice authenticates the message (e-mail) by using a public-key signature algorithm and encrypts it with her private key. The result is called the signature. Alice appends the identification of the public key (used for encryption) as well as the identification of the hash algorithm (used for authentication) to the signature. This part of the message contains the signature and two extra pieces of information: the encryption algorithm and the hash algorithm.
   b. Alice concatenates the three pieces of information created above with the message (e-mail) and encrypts the whole thing, using the session key created in step 1.
3. Alice combines the results of steps 1 and 2 and sends them to Bob (after adding the appropriate PGP header).

Receiver Site

The following shows the steps used in this scenario at Bob's side after he has received the PGP packet:

1. Bob uses his private key to decrypt the combination of the session key and symmetric-key algorithm identification.
2. Bob uses the session key and the algorithm obtained in step 1 to decrypt the rest of the PGP message. Bob now has the content of the message, the identification of the public algorithm used for creating and encrypting the signature, and the identification of the hash algorithm used to create the hash out of the message.
3. Bob uses Alice's public key and the algorithm defined by PA2 to decrypt the digest.
4. Bob uses the hash algorithm defined by HA to create a hash out of message he obtained in step 2.
5. Bob compares the hash created in step 4 and the hash he decrypted in step 3. If the two are identical, he accepts the message; otherwise, he discards the message.

PGP Algorithms

Table 32.4 shows some of the algorithms used in PGP. The list is not complete; new algorithms are continuously added.

Table 32.4

Algorithm	*ID*	*Description*
Public key	1	RSA (encryption or signing)
	2	RSA (for encryption only)
	3	RSA (for signing only)
	17	DSS (for signing)

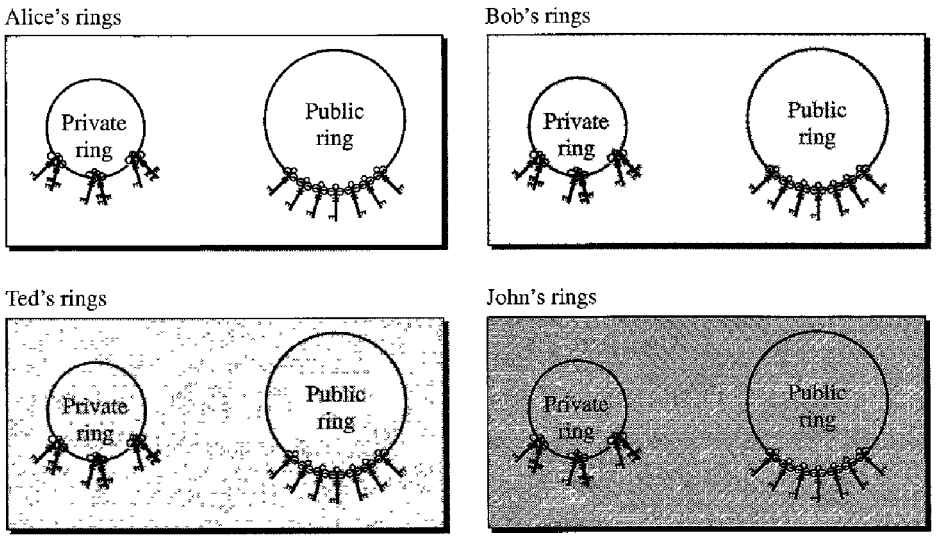
Table 32.4 *(continued)*

Algorithm	*ID*	*Description*
Hash algorithm	1	MD5
	2	SHA-1
	3	RIPE-MD
Encryption	0	No encryption
	1	IDEA
	2	Triple DES
	9	AES

Key Rings

In the previous scenarios, we assumed that Alice needed to send a message to only Bob. That is not always the case. Alice may need to send messages to many people. In this case, Alice needs a **key ring** of public keys, with a key belonging to each person with whom Alice needs to correspond (send or receive messages). In addition, the PGP designers specified a ring of private/public keys. One reason is that Alice may wish to change her pair of keys from time to time. Another reason is that Alice may need to correspond with different groups of people (friends, colleagues, and so on). Alice may wish to use a different key pair for each group. Therefore, each user needs to have two sets of rings: a ring of private/public keys and a ring of public keys of other people. Figure 32.21 shows a community of four people, each having a ring of pairs of private/public keys and, at the same time, a ring of four public keys belonging to the other four people in the community. The figure shows seven public keys for each public ring. Each person in the ring can keep more than one public key for each other person.

Figure 32.21 *Rings*



Alice, for example, has several pairs of private/public keys belonging to her and public keys belonging to other people. Note that everyone can have more than one public key. Two cases may arise.

1. Alice needs to send a message to one of the persons in the community.
   a. She uses her private key to sign the digest.
   b. She uses the receiver's public key to encrypt a newly created session key.
   c. She encrypts the message and signs the digest with the session key created.
2. Alice receives a message from one of the persons in the community.
   a. She uses her private key to decrypt the session key.
   b. She uses the session key to decrypt the message and digest.
   c. She uses her public key to verify the digest.

PGP Certificates

To trust the owner of the public key, each user in the PGP group needs to have, implicitly or explicitly, a copy of the certificate of the public-key owner. Although the certificate can come from a certificate authority (CA), this restriction is not required in PGP. PGP has its own certificate system.

Protocols that use X509 certificates depend on the hierarchical structure of the trust. There is a predefined chain of trust from the root to any certificate. Every user fully trusts the authority of the CA at the root level (prerequisite). The root issues certificates for the CAs at the second level, a second-level CA issues a certificate for the third level, and so on. Every party that needs to be trusted presents a certificate from some CA in the tree. If Alice does not trust the certificate issuer for Bob, she can appeal to a higher-level authority up to the root (which must be trusted for the system to work). In other words, there is one single path from a fully trusted CA to a certificate.

In PGP, there is no need for CAs; anyone in the ring can sign a certificate for anyone else in the ring. Bob can sign a certificate for Ted, John, Anne, and so on. There is no hierarchy of trust in PGP; there is no tree. As a result of the lack of hierarchical structure, Ted may have one certificate from Bob and another certificate from Liz. If Alice wants to follow the line of certificates for Ted, it has two paths: one starts from Bob and the other starts from Liz. An interesting point is that Alice may fully trust Bob, but only partially trust Liz. There can be multiple paths in the line of trust from a fully or partially trusted authority to a certificate. In PGP, the issuer of a certificate is usually called an **introducer.**

In PGP, there can be multiple paths from fully or partially trusted authorities to any subject.

Trusts and Legitimacy

The entire operation of PGP is based on introducer trust, the certificate trust, and the legitimacy of the public keys.

Introducer Trust Levels With the lack of a central authority, it is obvious that the ring cannot be very large if every user in the PGP ring of users has to fully trust everyone else.

(Even in real life we cannot fully trust everyone that we know.) To solve this problem, PGP allows different levels of trust. The number of levels is mostly implementation-dependent, but for simplicity, let us assign three levels of trust to any introducer: *none*, *partial*, and *full*. The **introducer trust** level specifies the trust levels issued by the introducer for other people in the ring. For example, Alice may fully trust Bob, partially trust Anne, and not trust John at all. There is no mechanism in PGP to determine how to make a decision about the trustworthiness of the introducer; it is up to the user to make this decision.

Certificate Trust Levels When Alice receives a certificate from an introducer, she stores the certificate under the name of the subject (certified entity). She assigns a level of trust to this certificate. The **certificate trust** level is normally the same as the introducer trust level that issued the certificate. Assume Alice fully trusts Bob, partially trusts Anne and Janette, and has no trust in John. The following scenarios can happen.

1. Bob issues two certificates, one for Linda (with public key K1) and one for Lesley (with public key K2). Alice stores the public key and certificate for Linda under Linda's name and assigns a *full* level of trust to this certificate. Alice also stores the certificate and public key for Lesley under Lesley's name and assigns a full level of trust to this certificate.
2. Anne issues a certificate for John (with public key K3). Alice stores this certificate and public key under John's name, but assigns a *partial* level for this certificate.
3. Janette issues two certificates, one for John (with public key K3) and one for Lee (with public key K4). Alice stores John's certificate under his name and Lee's certificate under his name, each with a *partial* level of trust. Note that John now has two certificates, one from Anne and one from Janette, each with a *partial* level of trust.
4. John issues a certificate for Liz. Alice can discard or keep this certificate with a signature trust of *none*.

Key Legitimacy The purpose of using introducer and certificate trusts is to determine the legitimacy of a public key. Alice needs to know how legitimate are the public keys of Bob, John, Liz, Anne, and so on. PGP defines a very clear procedure for determining **key legitimacy.** The level of the key legitimacy for a user is the weighted trust level of that user. For example, suppose we assign the following weights to certificate trust levels:

1. A weight of 0 to a nontrusted certificate
2. A weight of $\frac{1}{2}$ to a certificate with partial trust
3. A weight of 1 to a certificate with full trust

Then to fully trust an entity, Alice needs one fully trusted certificate or two partially trusted certificates for that entity. For example, Alice can use John's public key in the previous scenario because both Anne and Janette have issued a certificate for John, each with a certificate trust level of $\frac{1}{2}$. Note that the legitimacy of a public key belonging to an entity does not have anything to do with the trust level of that person. Although Bob can use John's public key to send a message to him, Alice cannot accept any certificate issued by John because, for Alice, John has a trust level of *none*.

Starting the Ring

You might have realized a problem with the above discussion. What if nobody sends a certificate for a fully or partially trusted entity? For example, how can the legitimacy of Bob's public key be determined if no one has sent a certificate for Bob? In PGP, the key legitimacy of a trusted or partially trusted entity can be also determined by other methods.

1. Alice can physically obtain Bob's public key. For example, Alice and Bob can meet personally and exchange a public key written on a piece of paper or to a disk.
2. If Bob's voice is recognizable to Alice, Alice can call him and obtain his public key on the phone.
3. A better solution proposed by PGP is for Bob to send his public key to Alice by e-mail. Both Alice and Bob make a 16-byte MD5 (or 20-byte SHA-1) digest from the key. The digest is normally displayed as eight groups of four digits (or 10 groups of four digits) in hexadecimal and is called a **fingerprint.** Alice can then call Bob and verify the fingerprint on the phone. If the key is altered or changed during the e-mail transmission, the two fingerprints do not match. To make it even more convenient, PGP has created a list of words, each representing a four-digit combination. When Alice calls Bob, Bob can pronounce the eight words (or 10 words) for Alice. The words are carefully chosen by PGP to avoid those similar in pronunciation; for example, if *sword* is in the list, *word* is not.
4. In PGP, nothing prevents Alice from getting Bob's public key from a CA in a separate procedure. She can then insert the public key in the public-key ring.

Web of Trust

PGP can eventually make a **web of trust** between a group of people. If each entity introduces more entities to other entities, the public-key ring for each entity gets larger and larger and entities in the ring can send secure e-mail to one another.

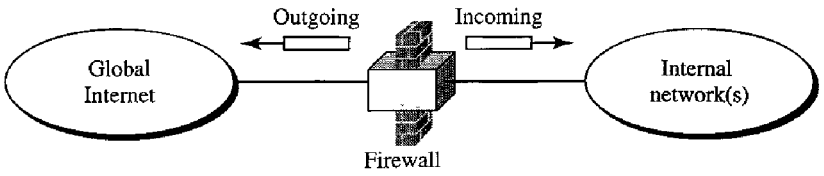
Key Revocation

It may become necessary for an entity to revoke his or her public key from the ring. This may happen if the owner of the key feels that the key is compromised (stolen, for example) or just too old to be safe. To revoke a key, the owner can send a revocation certificate signed by herself. The revocation certificate must be signed by the old key and disseminated to all the people in the ring who use that public key.

32.4 FIREWALLS

All previous security measures cannot prevent Eve from sending a harmful message to a system. To control access to a system, we need firewalls. A **firewall** is a device (usually a router or a computer) installed between the internal network of an organization and the rest of the Internet. It is designed to forward some packets and filter (not forward) others. Figure 32.22 shows a firewall.

Figure 32.22 *Firewall*



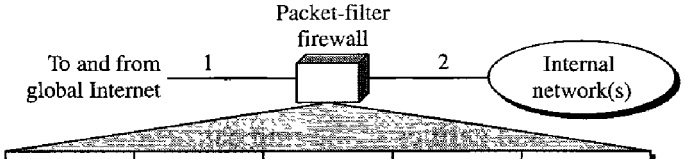
For example, a firewall may filter all incoming packets destined for a specific host or a specific server such as HTTP. A firewall can be used to deny access to a specific host or a specific service in the organization.

A firewall is usually classified as a packet-filter firewall or a proxy-based firewall.

Packet-Filter Firewall

A firewall can be used as a packet filter. It can forward or block packets based on the information in the network layer and transport layer headers: source and destination IP addresses, source and destination port addresses, and type of protocol (TCP or UDP). A **packet-filter firewall** is a router that uses a filtering table to decide which packets must be discarded (not forwarded). Figure 32.23 shows an example of a filtering table for this kind of a firewall.

Figure 32.23 *Packet-filter firewall*



Interface	Source IP	Source port	Destination IP	Destination port
1	131.34.0.0	*	*	*
1	*	*	*	23
1	*	*	194.78.20.8	*
2	*	80	*	*

According to Figure 32.23, the following packets are filtered:

1. Incoming packets from network 131.34.0.0 are blocked (security precaution). Note that the * (asterisk) means "any."
2. Incoming packets destined for any internal TELNET server (port 23) are blocked.
3. Incoming packets destined for internal host 194.78.20.8 are blocked. The organization wants this host for internal use only.
4. Outgoing packets destined for an HTTP server (port 80) are blocked. The organization does not want employees to browse the Internet.

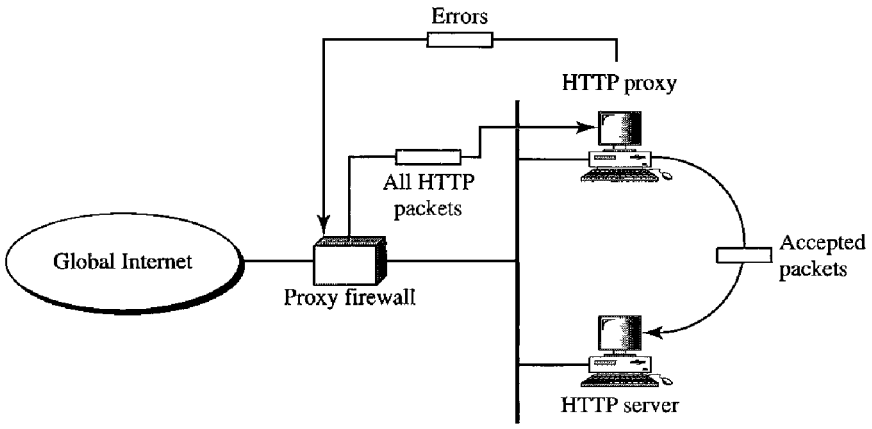
A packet-filter firewall filters at the network or transport layer.

Proxy Firewall

The packet-filter firewall is based on the information available in the network layer and transport layer headers (IP and TCP/UDP). However, sometimes we need to filter a message based on the information available in the message itself (at the application layer). As an example, assume that an organization wants to implement the following policies regarding its Web pages: Only those Internet users who have previously established business relations with the company can have access; access to other users must be blocked. In this case, a packet-filter firewall is not feasible because it cannot distinguish between different packets arriving at TCP port 80 (HTTP). Testing must be done at the application level (using URLs).

One solution is to install a proxy computer (sometimes called an application gateway), which stands between the customer (user client) computer and the corporation computer shown in Figure 32.24.

Figure 32.24 *Proxy firewall*



When the user client process sends a message, the **proxy firewall** runs a server process to receive the request. The server opens the packet at the application level and finds out if the request is legitimate. If it is, the server acts as a client process and sends the message to the real server in the corporation. If it is not, the message is dropped and an error message is sent to the external user. In this way, the requests of the external users are filtered based on the contents at the application layer. Figure 32.24 shows a proxy firewall implementation.

A proxy firewall filters at the application layer.

32.5 RECOMMENDED READING

For more details about subjects discussed in this chapter, we recommend the following books. The brackets, [. . .], refer to the reference list at the end of the text.

Books

IPSec is discussed in Chapter 7 of [Rhe03], Section 18.1 of [PHS03], and Chapters 17 and 18 of [KPS02]. A full discussion of IPSec can be found in [DH03]. SSL/TLS is discussed in Chapter 8 of [Rhe03], and Chapter 19 of [KPS02]. A full discusion of SSL and TLS can be found in [Res01] and [Tho00]. PGP is discussed in Chapter 9 of [Rhe03], and Chapter 22 of [KPS02]. Firewalls are discussed in Chapter 10 of [Rhe03] and Chapter 23 of [KPS02]. Firewalls are fully discussed in [CBR03]. Virtual private networks are fully discussed in [YS01] and [SWE99].

32.6 KEY TERMS

Alert Protocol
Authentication Header (AH) Protocol
certificate trust
ChangeCipherSpec Protocol
cipher suite
connection
Encapsulating Security Payload (ESP)
extranet
fingerprint
firewall
Handshake Protocol
hybrid network
Internet Key Exchange (IKE)
Internet Security Association and Key Management Protocol (ISAKMP)
intranet
introducer
introducer trust
IP Security (IPSec)
key legitimacy
key material
key ring
master secret
Oakley
packet-filter firewall
premaster secret
Pretty Good Privacy (PGP)
private network
proxy firewall
pseudorandom function (PRF)
Record Protocol
replay attack
Secure Socket Layer (SSL)
security association (SA)
security association database (SADB)
security parameter index (SPI)
session
SKEME
Transport Layer Security (TLS)
transport mode
tunnel mode
tunneling
virtual private network (VPN)
web of trust

32.7 SUMMARY

- IP Security (IPSec) is a collection of protocols designed by the IETF (Internet Engineering Task Force) to provide security for a packet at the network level.
- IPSec operates in the transport mode or the tunnel mode.
- In the transport mode, IPSec protects information delivered from the transport layer to the network layer. IPSec in the transport mode does not protect the IP header. The transport mode is normally used when we need host-to-host (end-to-end) protection of data.
- In the tunnel mode, IPSec protects the whole IP packet, including the original IP header.
- IPSec defines two protocols—Authentication Header (AH) Protocol and Encapsulating Security Payload (ESP) Protocol—to provide authentication or encryption or both for packets at the IP level.
- IPSec requires a logical relationship between two hosts called a security association (SA). IPSec uses a set of SAs called the security association database or SADB.
- The Internet Key Exchange (IKE) is the protocol designed to create security associations, both inbound and outbound. IKE creates SAs for IPSec.
- IKE is a complex protocol based on three other protocols: Oakley, SKEME, and ISAKMP.
- A private network is used inside an organization.
- An intranet is a private network that uses the Internet model. An extranet is an intranet that allows authorized access from outside users.
- The Internet authorities have reserved addresses for private networks.
- A virtual private network (VPN) provides privacy for LANs that must communicate through the global Internet.
- A transport layer security protocol provides end-to-end security services for applications that use the services of a reliable transport layer protocol such as TCP.
- Two protocols are dominant today for providing security at the transport layer: Secure Sockets Layer (SSL) and Transport Layer Security (TLS). The second is actually an IETF version of the first.
- SSL is designed to provide security and compression services to data generated from the application layer. Typically, SSL can receive application data from any application layer protocol, but the protocol is normally HTTP.
- SSL provides services such as fragmentation, compression, message integrity, confidentiality, and framing on data received from the application layer.
- The combination of key exchange, hash, and encryption algorithms defines a cipher suite for each SSL session. The name of each suite is descriptive of the combination.
- In e-mail, the cryptographic algorithms and secrets are sent with the message.
- One security protocol for the e-mail system is Pretty Good Privacy (PGP). PGP was invented by Phil Zimmerman to provide privacy, integrity, and authentication in e-mail.

- To exchange e-mail messages, a user needs a ring of public keys; one public key is needed for each e-mail correspondent.
- PGP has also specified a ring of private/public key pairs to allow a user to change her pair of keys from time to time. PGP also allows each user to have different user IDs (e-mail addresses) for different groups of people.
- PGP certification is different from X509. In X509, there is a single path from the fully trusted authority to any certificate. In PGP, there can be multiple paths from fully or partially trusted authorities.
- PGP uses the idea of certificate trust levels.
- When a user receives a certificate from an introducer, it stores the certificate under the name of the subject (certified entity). It assigns a level of trust to this certificate.

32.8 PRACTICE SET

Review Questions

1. Why does IPSec need a security association?
2. How does IPSec create a set of security parameters?
3. What are the two protocols defined by IPSec?
4. What does AH add to the IP packet?
5. What does ESP add to the IP packet?
6. Are both AH and ESP needed for IP security? Why or why not?
7. What are the two protocols discussed in this chapter that provide security at the transport layer?
8. What is IKE?
9. What is the difference between a session and a connection in SSL?
10. How does SSL create a set of security parameters?
11. What is the name of the protocol, discussed in this chapter, that provides security for e-mail?
12. How does PGP create a set of security parameters?
13. What is the purpose of the Handshake Protocol in SSL?
14. What is the purpose of the Record Protocol in SSL?
15. What is the purpose of a firewall?
16. What are the two types of firewalls?
17. What is a VPN and why is it needed?
18. How do LANs on a fully private internet communicate?

Exercises

19. Show the values of the AH fields in Figure 32.6. Assume there are 128 bits of authentication data.
20. Show the values of the ESP header and trailer fields in Figure 32.7.

21. Redraw Figure 32.6 if AH is used in tunnel mode.
22. Redraw Figure 32.7 if ESP is used in tunnel mode.
23. Draw a figure to show the position of AH in IPv6.
24. Draw a figure to show the position of ESP in IPv6.
25. Does the IPSec Protocol need the services of a KDC? Explain your answer.
26. Does the IPSec Protocol need the services of a CA? Explain your answer.
27. Does the SSL Protocol need the services of a KDC? Explain your answer.
28. Does the SSL Protocol need the services of a CA? Explain your answer.
29. Does the PGP Protocol need the services of a KDC? Explain your answer.
30. Does the PGP Protocol need the services of a CA? Explain your answer.
31. Are there any cipher suites in IPSec? Explain your answer.
32. Are there any cipher suites in PGP? Explain your answer.