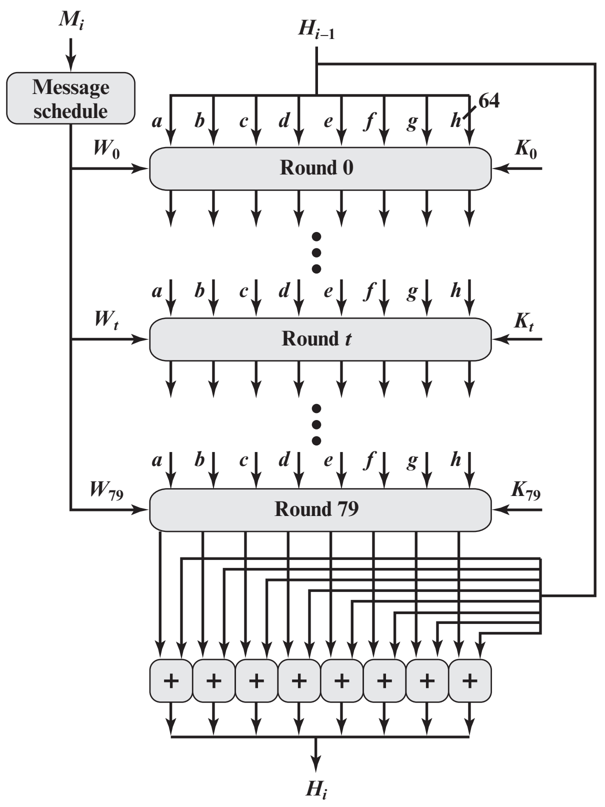


Figure 3.5 SHA-512 Processing of a Single 1024-Bit Block

The SHA-512 algorithm has the property that every bit of the hash code is a function of every bit of the input. The complex repetition of the basic function F produces results that are well mixed; that is, it is unlikely that two messages chosen at random, even if they exhibit similar regularities, will have the same hash code. Unless there is some hidden weakness in SHA-512, which has not so far been published, the difficulty of coming up with two messages having the same message digest is on the order of 2^{256} operations, while the difficulty of finding a message with a given digest is on the order of 2^{512} operations.

SHA-3

SHA-2, particularly the 512-bit version, would appear to provide unassailable security. However, SHA-2 shares the same structure and mathematical operations as its predecessors, and this is a cause for concern. Because it would take years to find a suitable replacement for SHA-2, should it become vulnerable, NIST announced in 2007 a competition to produce the next-generation NIST hash function, which is to be called SHA-3. Following are the basic requirements that must be satisfied by any candidate for SHA-3:

1. It must be possible to replace SHA-2 with SHA-3 in any application by a simple drop-in substitution. Therefore, SHA-3 must support hash value lengths of 224, 256, 384, and 512 bits.
2. SHA-3 must preserve the online nature of SHA-2. That is, the algorithm must process comparatively small blocks (512 or 1024 bits) at a time instead of requiring that the entire message be buffered in memory before processing it.

In 2012, NIST selected a winning submission and formally published SHA-3. A detailed presentation of SHA-3 is provided in Chapter 15.

3.3 MESSAGE AUTHENTICATION CODES

HMAC

In recent years, there has been increased interest in developing a MAC derived from a cryptographic hash code, such as SHA-1. The motivations for this interest are as follows:

- Cryptographic hash functions generally execute faster in software than conventional encryption algorithms such as DES.
- Library code for cryptographic hash functions is widely available.

A hash function such as SHA-1 was not designed for use as a MAC and cannot be used directly for that purpose because it does not rely on a secret key. There have been a number of proposals for the incorporation of a secret key into an existing hash algorithm. The approach that has received the most support is HMAC [BELL96a, BELL96b]. HMAC has been issued as RFC 2104, has been chosen as the mandatory-to-implement MAC for IP Security, and is used in other Internet protocols, such as Transport Layer Security (TLS) and Secure Electronic Transaction (SET).

HMAC Design Objectives RFC 2104 lists the following design objectives for HMAC.

- To use, without modifications, available hash functions. In particular, hash functions that perform well in software, and for which code is freely and widely available
- To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required
- To preserve the original performance of the hash function without incurring a significant degradation
- To use and handle keys in a simple way
- To have a well-understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions on the embedded hash function

The first two objectives are important to the acceptability of HMAC. HMAC treats the hash function as a "black box." This has two benefits. First, an existing implementation of a hash function can be used as a module in implementing HMAC.

In this way, the bulk of the HMAC code is prepackaged and ready to use without modification. Second, if it is ever desired to replace a given hash function in an HMAC implementation, all that is required is to remove the existing hash function module and drop in the new module. This could be done if a faster hash function were desired. More important, if the security of the embedded hash function were compromised, the security of HMAC could be retained simply by replacing the embedded hash function with a more secure one.

The last design objective in the preceding list is, in fact, the main advantage of HMAC over other proposed hash-based schemes. HMAC can be proven secure provided that the embedded hash function has some reasonable cryptographic strengths. We return to this point later in this section, but first we examine the structure of HMAC.

HMAC ALGORITHM Figure 3.6 illustrates the overall operation of HMAC. The following terms are defined:

H	=	embedded hash function (e.g., SHA-1)
M	=	message input to HMAC (including the padding specified in the embedded hash function)

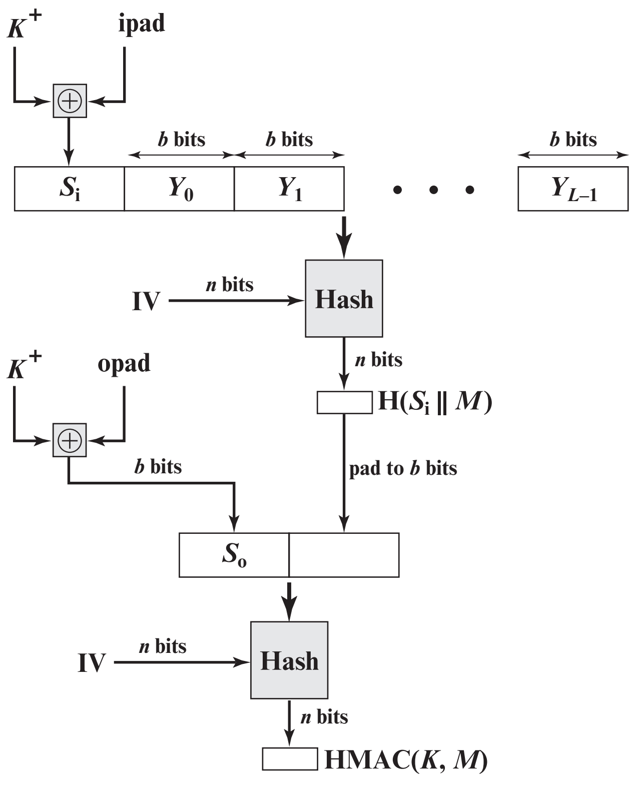


Figure 3.6 HMAC Structure

Y_i = ith block of M, $0 \leq i \leq (L - 1)$
L = number of blocks in M
b = number of bits in a block
n = length of hash code produced by embedded hash function
K = secret key; if key length is greater than b, the key is input to the hash function to produce an n-bit key; recommended length is $> n$
K^+ = K padded with zeros on the left so that the result is b bits in length
ipad = 00110110 (36 in hexadecimal) repeated $b/8$ times
opad = 01011100 (5C in hexadecimal) repeated $b/8$ times

Then HMAC can be expressed as

$$\text{HMAC}(K, M) = \text{H}[(K^+ \oplus \text{opad}) \| \text{H}[(K^+ \oplus \text{ipad}) \| M]]$$

In words, HMAC is defined as follows:

1. Append zeros to the left end of K to create a b-bit string K^+ (e.g., if K is of length 160 bits and $b = 512$, then K will be appended with 44 zero bytes).
2. XOR (bitwise exclusive-OR) K^+ with ipad to produce the b-bit block S_i.
3. Append M to S_i.
4. Apply H to the stream generated in step 3.
5. XOR K^+ with opad to produce the b-bit block S_o.
6. Append the hash result from step 4 to S_o.
7. Apply H to the stream generated in step 6 and output the result.

Note that the XOR with ipad results in flipping one-half of the bits of K. Similarly, the XOR with opad results in flipping one-half of the bits of K, but a different set of bits. In effect, by passing S_i and S_o through the hash algorithm, we have pseudorandomly generated two keys from K.

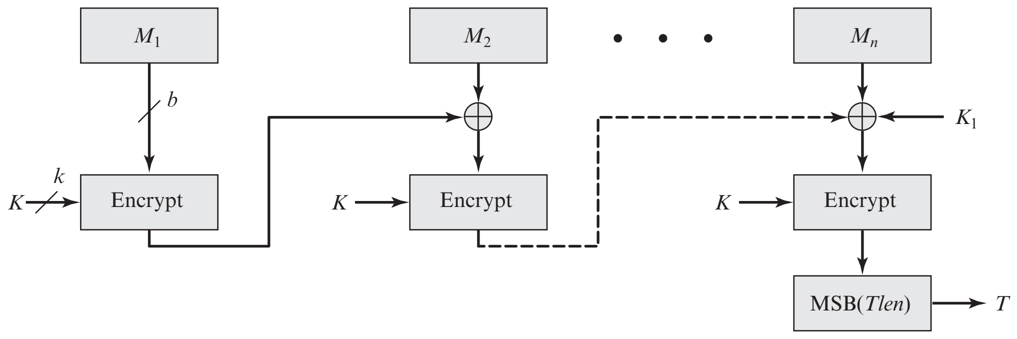
HMAC should execute in approximately the same time as the embedded hash function for long messages. HMAC adds three executions of the basic hash function (for S_i, S_o, and the block produced from the inner hash).

MACs Based on Block Ciphers

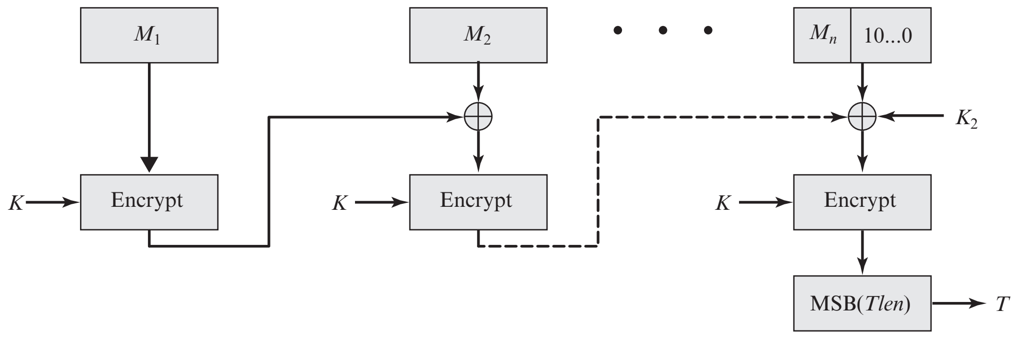
In this section, we look at several MACs based on the use of a block cipher.

Cipher-Based Message Authentication Code (CMAC) The Cipher-based Message Authentication Code mode of operation is for use with AES and triple DES. It is specified in SP 800-38B.

First, let us consider the operation of CMAC when the message is an integer multiple n of the cipher block length b. For AES, $b = 128$, and for triple DES, $b = 64$. The message is divided into n blocks ($M_1, M_2, \ldots, M_n$). The algorithm makes use of a k-bit encryption key K and an n-bit key, K_1. For AES, the key size k is 128, 192, or 256 bits; for triple DES, the key size is 112 or 168 bits. CMAC is calculated as follows (Figure 3.7).



(a) Message length is integer multiple of block size



(b) Message length is not integer multiple of block size

Figure 3.7 Cipher-Based Message Authentication Code (CMAC)

$$\begin{aligned}
C_1 &= \mathrm{E}(K, M_1)\\
C_2 &= \mathrm{E}(K, [M_2 \oplus C_1])\\
C_3 &= \mathrm{E}(K, [M_3 \oplus C_2])\\
&\cdot\\
&\cdot\\
&\cdot\\
C_n &= \mathrm{E}(K, [M_N \oplus C_{n-1} \oplus K_1])\\
T &= \mathrm{MSB}_{Tlen}(C_n)
\end{aligned}$$

where

T	= message authentication code, also referred to as the tag
$Tlen$	= bit length of T
$\mathrm{MSB}_s(X)$	= the s leftmost bits of the bit string X

If the message is not an integer multiple of the cipher block length, then the final block is padded to the right (least significant bits) with a 1 and as many

0s as necessary so that the final block is also of length b. The CMAC operation then proceeds as before, except that a different n-bit key K_2 is used instead of K_1.

To generate the two n-bit keys, the block cipher is applied to the block that consists entirely of 0 bits. The first subkey is derived from the resulting ciphertext by a left shift of one bit and, conditionally, by XORing a constant that depends on the block size. The second subkey is derived in the same manner from the first subkey.

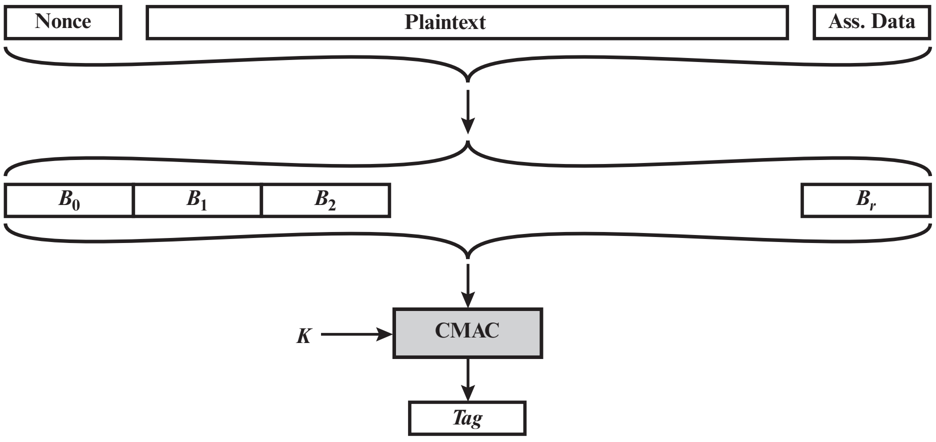
Counter with Cipher Block Chaining-Message Authentication Code The Counter with Cipher Block Chaining-Message Authentication Code (CCM) mode of operation, defined in SP 800-38C, is referred to as an **authenticated encryption** mode. "Authenticated encryption" is a term used to describe encryption systems that simultaneously protect confidentiality and authenticity (integrity) of communications. Many applications and protocols require both forms of security, but until recently the two services have been designed separately.

The key algorithmic ingredients of CCM are the AES encryption algorithm (Section 2.2), the CTR mode of operation (Section 2.5), and the CMAC authentication algorithm. A single key K is used for both encryption and MAC algorithms. The input to the CCM encryption process consists of three elements.

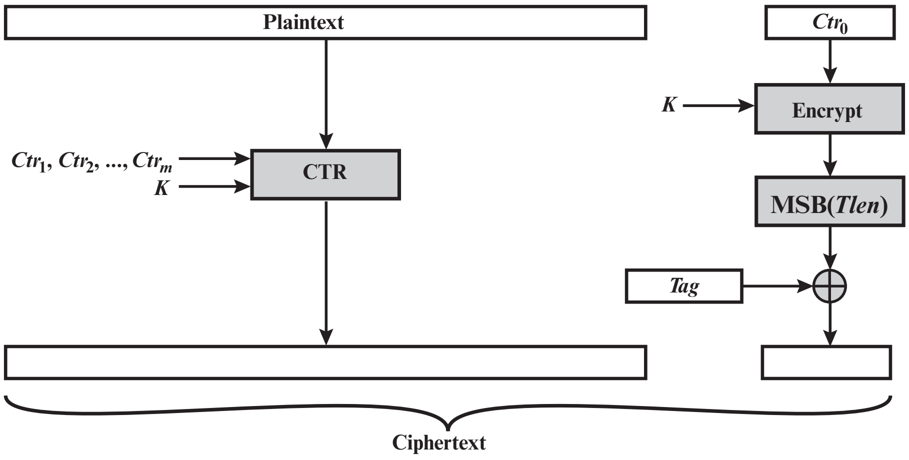
1. Data that will be both authenticated and encrypted. This is the plaintext message P of data block.
2. Associated data A that will be authenticated but not encrypted. An example is a protocol header that must be transmitted in the clear for proper protocol operation but which needs to be authenticated.
3. A nonce N that is assigned to the payload and the associated data. This is a unique value that is different for every instance during the lifetime of a protocol association and is intended to prevent replay attacks and certain other types of attacks.

Figure 3.8 illustrates the operation of CCM. For authentication, the input includes the nonce, the associated data, and the plaintext. This input is formatted as a sequence of blocks B_0 through B_r. The first block contains the nonce plus some formatting bits that indicate the lengths of the N, A, and P elements. This is followed by zero or more blocks that contain A, followed by zero or more blocks that contain P. The resulting sequence of blocks serves as input to the CMAC algorithm, which produces a MAC value with length *Tlen*, which is less than or equal to the block length (Figure 3.8a).

For encryption, a sequence of counters is generated that must be independent of the nonce. The authentication tag is encrypted in CTR mode using the single counter Ctr_0. The *Tlen* most significant bits of the output are XORed with the tag to produce an encrypted tag. The remaining counters are used for the CTR mode encryption of the plaintext (Figure 2.11). The encrypted plaintext is concatenated with the encrypted tag to form the ciphertext output (Figure 3.8b).



(a) Authentication



(b) Encryption

Figure 3.8 Counter with Cipher Block Chaining-Message Authentication Code

3.4 PUBLIC-KEY CRYPTOGRAPHY PRINCIPLES

Of equal importance to conventional encryption is **public-key encryption**, which finds use in message authentication and key distribution. This section looks first at the basic concept of public-key encryption and takes a preliminary look at key distribution issues. Section 3.5 examines the two most important public-key algorithms: RSA and Diffie–Hellman. Section 3.6 introduces digital signatures.

Public-Key Encryption Structure

Public-key encryption, first publicly proposed by Diffie and Hellman in 1976 [DIFF76], is the first truly revolutionary advance in encryption in literally thousands of years. Public-key algorithms are based on mathematical functions rather than on simple operations on bit patterns, such as are used in symmetric encryption algorithms. More important, public-key cryptography is asymmetric, involving the use of two separate keys—in contrast to the symmetric conventional encryption, which uses only one key. The use of two keys has profound consequences in the areas of confidentiality, key distribution, and authentication.

Before proceeding, we should first mention several common misconceptions concerning public-key encryption. One is that public-key encryption is more secure from cryptanalysis than conventional encryption. In fact, the security of any encryption scheme depends on (1) the length of the key and (2) the computational work involved in breaking a cipher. There is nothing in principle about either conventional or public-key encryption that makes one superior to another from the point of view of resisting cryptanalysis. A second misconception is that public-key encryption is a general-purpose technique that has made conventional encryption obsolete. On the contrary, because of the computational overhead of current public-key encryption schemes, there seems no foreseeable likelihood that conventional encryption will be abandoned. Finally, there is a feeling that key distribution is trivial when using public-key encryption, compared to the rather cumbersome handshaking involved with key distribution centers for conventional encryption. In fact, some form of protocol is needed, often involving a central agent, and the procedures involved are no simpler or any more efficient than those required for conventional encryption.

A public-key encryption scheme has six ingredients (Figure 3.9a).

- **Plaintext:** This is the readable message or data that is fed into the algorithm as input.
- **Encryption algorithm:** The encryption algorithm performs various transformations on the plaintext.
- **Public and private key:** This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the encryption algorithm depend on the public or private key that is provided as input.
- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.
- **Decryption algorithm:** This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

As the names suggest, the public key of the pair is made public for others to use, while the private key is known only to its owner. A general-purpose public-key cryptographic algorithm relies on one key for encryption and a different but related key for decryption.

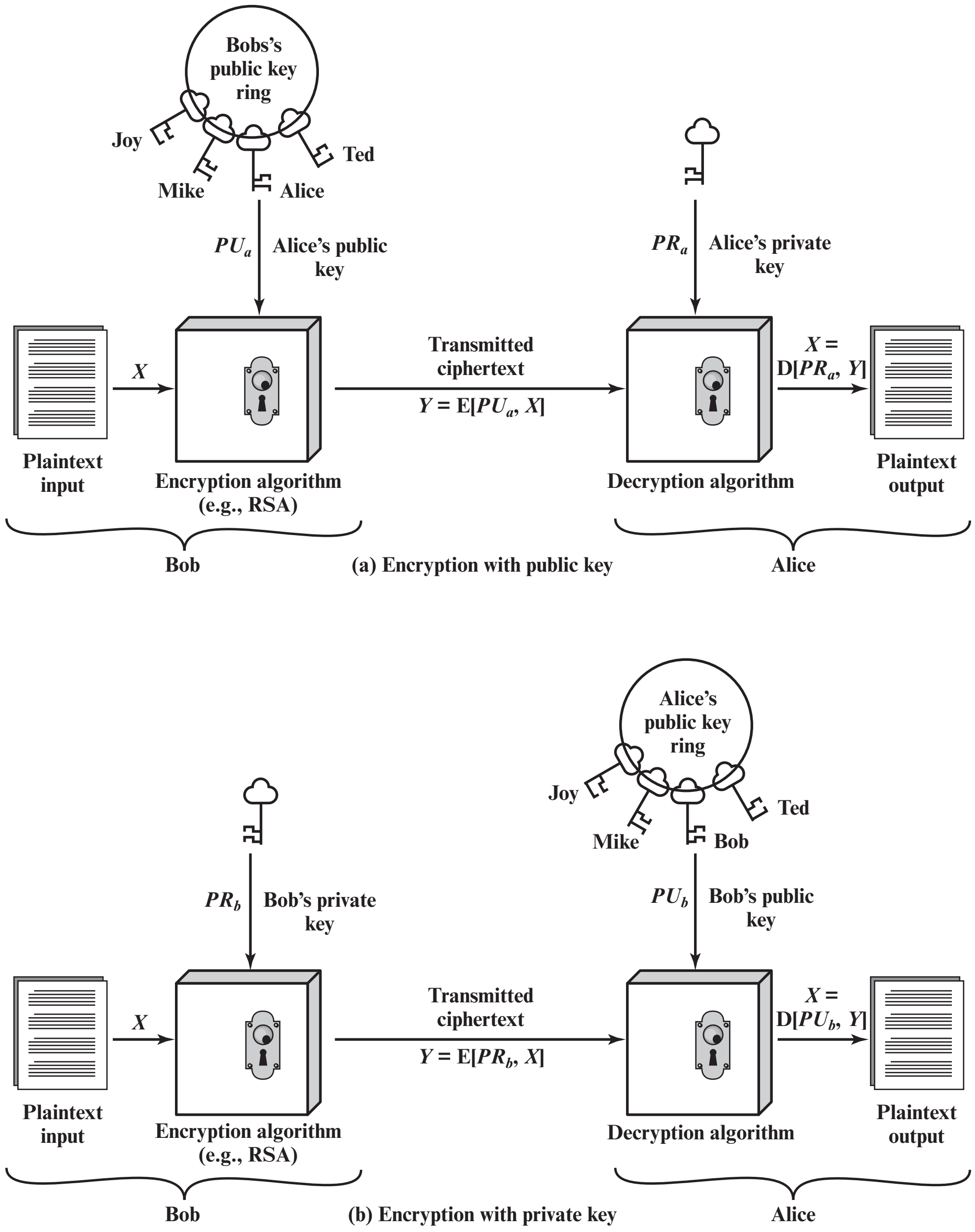


Figure 3.9 Public-Key Cryptography

The essential steps are the following:

1. Each user generates a pair of keys to be used for the encryption and decryption of messages.
2. Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private. As Figure 3.9a

suggests, each user maintains a collection of public keys obtained from others.

3. If Bob wishes to send a private message to Alice, Bob encrypts the message using Alice's public key.
4. When Alice receives the message, she decrypts it using her private key. No other recipient can decrypt the message because only Alice knows Alice's private key.

With this approach, all participants have access to public keys, and private keys are generated locally by each participant and therefore need never be distributed. As long as a user protects his or her private key, incoming communication is secure. At any time, a user can change the private key and publish the companion public key to replace the old public key.

The key used in conventional encryption is typically referred to as a **secret key**. The two keys used for public-key encryption are referred to as the **public key** and the **private key**. Invariably, the private key is kept secret, but it is referred to as a private key rather than a secret key to avoid confusion with conventional encryption.

Applications for Public-Key Cryptosystems

Before proceeding, we need to clarify one aspect of public-key cryptosystems that is otherwise likely to lead to confusion. Public-key systems are characterized by the use of a cryptographic type of algorithm with two keys, one held private and one available publicly. Depending on the application, the sender uses either the sender's private key, the receiver's public key, or both to perform some type of cryptographic function. In broad terms, we can classify the use of public-key cryptosystems into three categories:

- **Encryption/decryption:** The sender encrypts a message with the recipient's public key.
- **Digital signature:** The sender "signs" a message with its private key. Signing is achieved by a cryptographic algorithm applied to the message or to a small block of data that is a function of the message.
- **Key exchange:** Two sides cooperate to exchange a session key. Several different approaches are possible, involving the private key(s) of one or both parties.

Some algorithms are suitable for all three applications, whereas others can be used only for one or two of these applications. Table 3.2 indicates the applications supported by the algorithms discussed in this chapter: RSA and Diffie–Hellman. This table also includes the Digital Signature Standard (DSS) and elliptic-curve cryptography, also mentioned later in this chapter.

One general observation can be made at this point. Public-key algorithms require considerably more computation than symmetric algorithms for comparable security and a comparable plaintext length. Accordingly, public-key algorithms are used only for short messages or data blocks, such as to encrypt a secret key or PIN.