UNIT-IV

HASH FUNCTION:

It is a one of the authentication function; it accepts a variable size message M as input and produces a fixed size output.

A hash value 'h' is generated by a function H of the form

h=H (M)

M→ variable length message

H(M)→ fixed length hash value.

The hash code is also referred as Message Digest (MD) or hash value.

The main difference between Hash Function and MAC is a hash code does not use a key but is a function only of the input message.

The hash value is appended to the message at the source at a time when the message is assumed or known to be correct.

The receiver authenticates that message by re-computing the hash value.

Hash functions are often used to determine whether or not data has changed.

Figure 11.1 depicts the general operation of a cryptographic hash function

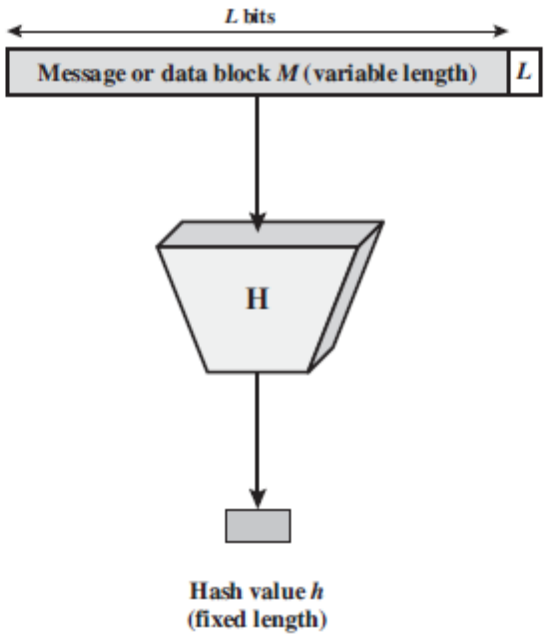


Figure 11.1 Black Diagram of Cryptographic Hash Function; $h = H(M)$

APPLICATIONS OF CRYPTOGRAPHIC HASH FUNCTIONS

It is used in a wide variety of security applications and Internet protocols

Message Authentication

Message authentication is a mechanism or service used to verify the integrity of a message. Message authentication assures that data received are exactly as sent (i.e., contain no modification, insertion, deletion, or replay)

When a hash function is used to provide message authentication, the hash function value is often referred to as a message digest.

Figure 11.2 illustrates a variety of ways in which a hash code can be used to provide message authentication, as follows.

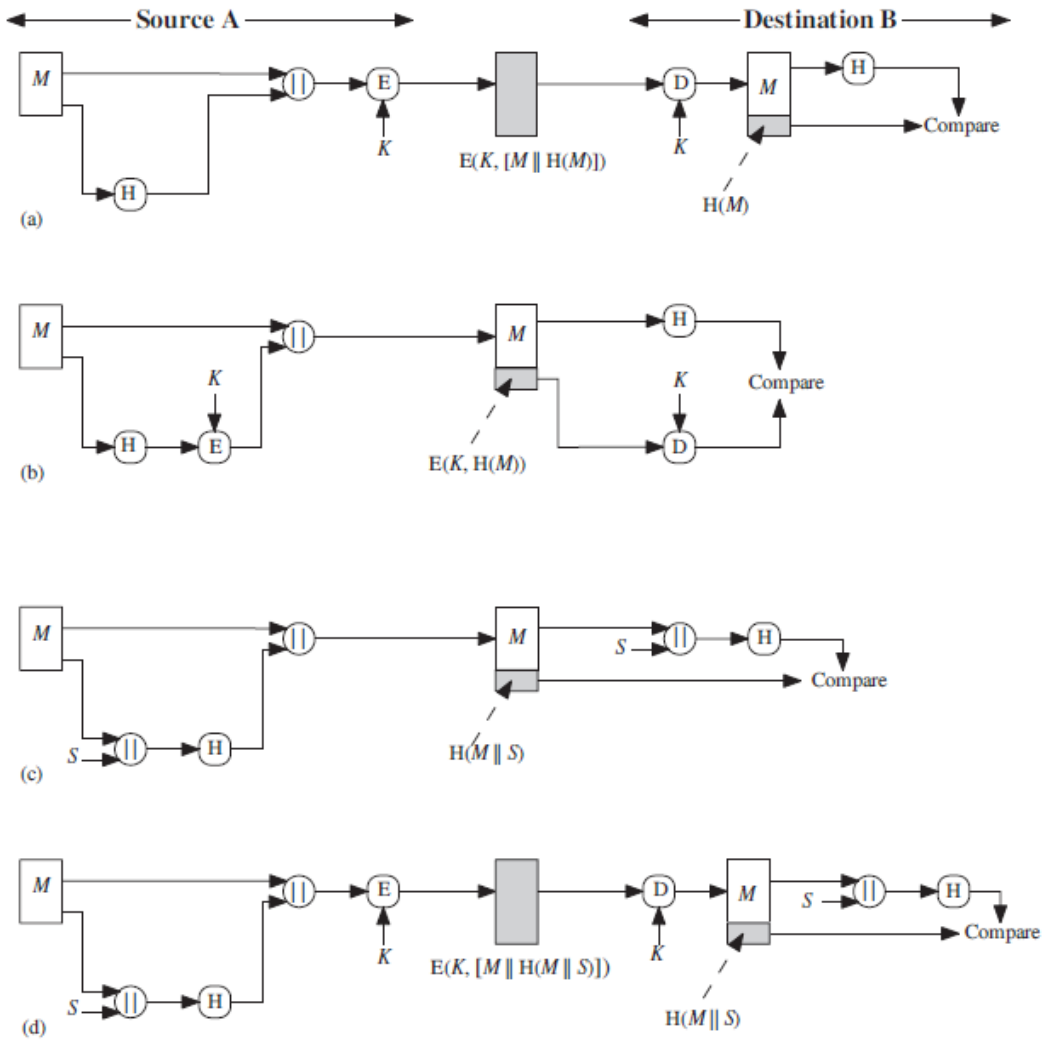


Figure 11.2 Simplified Examples of the Use of a Hash Function for Message Authentication

(a)The message plus concatenated hash code is encrypted using symmetric encryption. Because only A and B share the secret key, the message must have come from A and has not been altered.

The hash code provides the structure or redundancy required to achieve authentication. Because encryption is applied to the entire message plus hash code, confidentiality is also provided.

(b)Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality

(c)It is possible to use a hash function but no encryption for message authentication. The technique assumes that the two communicating parties share a common secret value S.A computes the hash value over the concatenation of M and S and appends the resulting hash value to M. Because B possesses, it can recomputed the hash value to verify. Because the secret value itself is not sent, an opponent cannot modify an intercepted message and cannot generate a false message.

(d)Confidentiality can be added to the approach of method (c) by encrypting the entire message plus the hash code.

Digital Signatures

Another important application, which is similar to the message authentication application, is the digital signature.

The operation of the digital signature is similar to that of the MAC. In the case of the digital signature, the hash value of a message is encrypted with a user's private key. Anyone who knows the user's public key can verify the integrity of the message that is associated with the digital signature.

Figure 11.3 illustrates, in a simplified fashion, how a hash code is used to provide a digital signature.

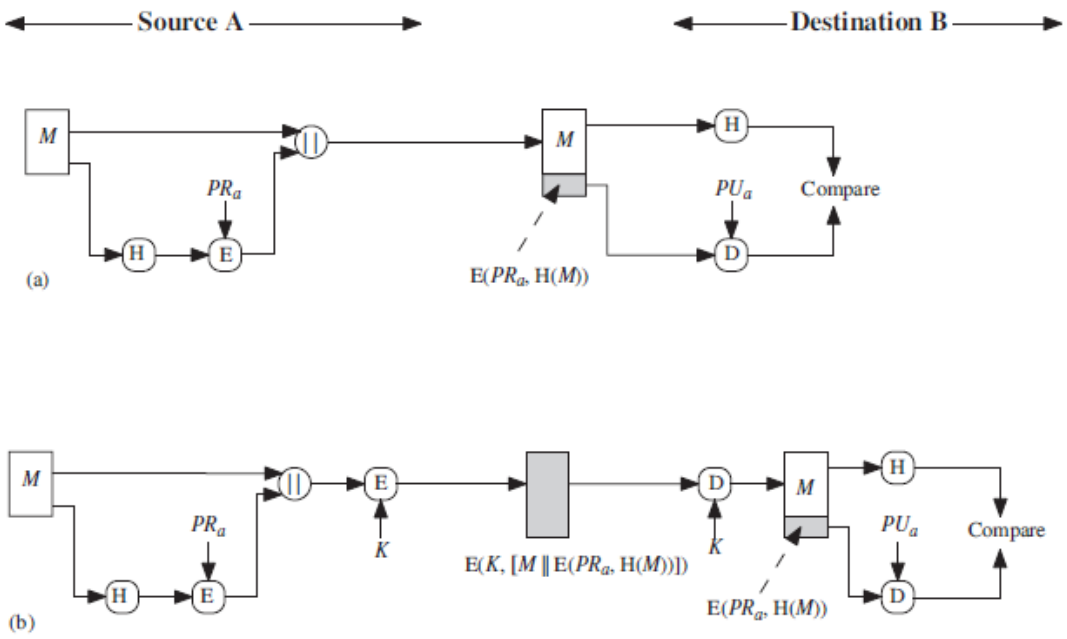


Figure 11.3 Simplified Examples of Digital Signatures

REQUIREMENTS& SECURITY FOR A HASH FUNCTION:

The purpose of a hash function is to produce a "fingerprint" of a file, message or other block of data. To be useful for message authentication, a hash function H must have the following properties:

H can be applied to a block of data of any size

H produces a fixed length output.

H(x) is relatively easy to compute for any given x, making both hardware and software implementations practical.

One-way property: - for any given value h, it is computationally infeasible to find x such that H(x)=h. this sometimes referred to in the literature as the one way property.

Weak collision resistance:- for any given block x. it is computationally infeasible to find y≠x with H(y)=H(x). this is referred as weak collision resistance.

Strong collision resistance:- it is computationally infeasible to find any pair (X,Y) such that H(x)=H(y). this is referred as strong collision resistance.

Requirements for a Cryptographic Hash Function H

Requirement	Description
Variable input size	H can be applied to a block of data of any size.
Fixed output size	H produces a fixed-length output.
Efficiency	$H(x)$ is relatively easy to compute for any given x, making both hardware and software implementations practical.
Preimage resistant (one-way property)	For any given hash value h, it is computationally infeasible to find y such that $H(y) = h$.
Second preimage resistant (weak collision resistant)	For any given block x, it is computationally infeasible to find $y \neq x$ with $H(y) = H(x)$.
Collision resistant (strong collision resistant)	It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$.
Pseudorandomness	Output of H meets standard tests for pseudorandomness.

A hash function that satisfies the first five properties in Table 11.1 is referred to as a weak hash function. If the sixth property, collision resistant, is also satisfied, then it is referred to as a strong hash function.

As with encryption algorithms, there are two categories of attacks on hash functions: brute-force attacks and cryptanalysis

Brute-Force Attacks

A brute-force attack does not depend on the specific algorithm but depends only on bit length. In the case of a hash function, a brute-force attack depends only on the bit length of the hash value. A cryptanalysis, in contrast, is an attack based on weaknesses in a particular cryptographic algorithm.

Cryptanalysis

As with encryption algorithms, cryptanalytic attacks on hash functions seek to exploit some property of the algorithm to perform some attack other than an exhaustive search. The way to measure the resistance of a hash algorithm to cryptanalysis is to compare its strength to the effort required for a brute-force attack.

That is, an ideal hash algorithm will require a cryptanalytic effort greater than or equal to the brute-force effort.

SHA(Secure Hash Algorithm):

In recent years, the most widely used hash function has been the Secure Hash Algorithm (SHA).

Introduction:

The Secure Hash Algorithm is a family of cryptographic hash functions developed by the NIST (National Institute of Standards & Technology).

SHA is based on the MD4 algorithm and its design closely models MD5.

SHA-1 is specified in RFC 3174.

Purpose: Authentication, not encryption.

SHA-1 produces a hash value of 160 bits. In 2002, NIST produced a revised version of the standard, FIPS 180-2, that defined three new versions of SHA, with hash value lengths of 256, 384, and 512 bits, known as SHA-256, SHA-384, and SHA-512, respectively.

SHA-1 logic:

The algorithm takes a message with maximum of length of less than 264 bits.

Produce output is 160 bits message digest.

The input is processed 512 bits block.

Processed Steps:

Algorithm processing Steps:

Step1: Append Padding Bits

Step 2: Append Length

Step 3: Initialize MD Buffer

Step 4: Process Message in 512 bit (16-Word) Blocks

Step 5: Output

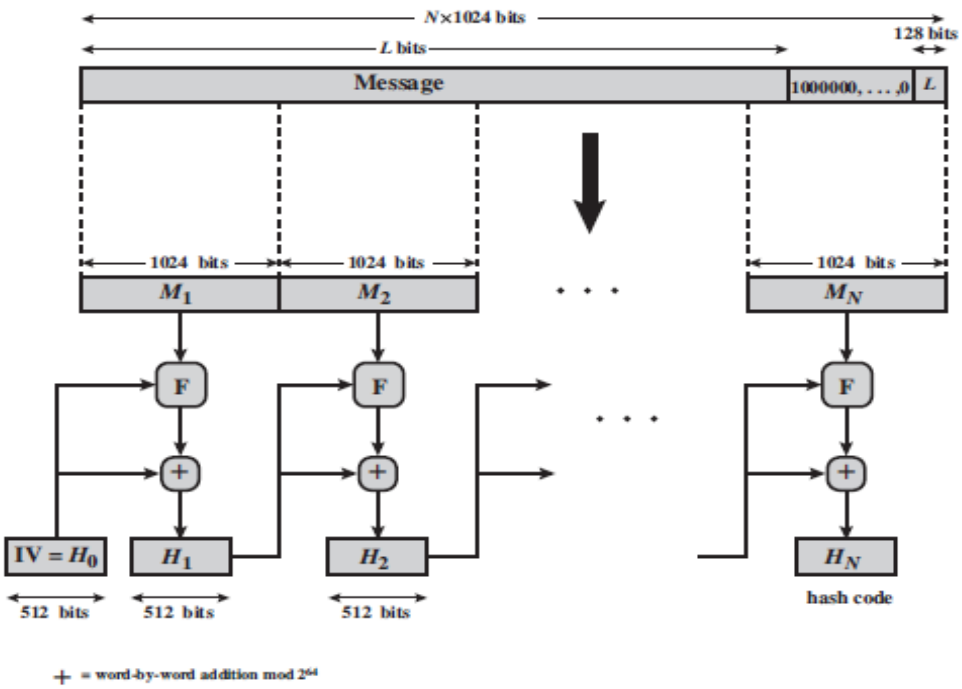


Figure 11.8 Message Digest Generation Using SHA-512

Step-1: **Appending Padding Bits**: The original message is "padded" (extended) so that its length (in bits) is congruent to 448, modulo 512. The padding rules are:

The original message is always padded with one bit "1" first.

Then zero or more bits "0" are padded to bring the length of the message up to 64 bits fewer than a multiple of 512.

Step-2: **append length**: a block of 64 bits is appended to the message. This block is treated as unsigned 64 bit integers (most significant byte first) and contains the length of the original message.

Step-3: **Initialize MD buffer**: 160 bit buffer is used to hold intermediate and final results of the hash function. This buffer can be represented as five 32 bit registers (A, B,C,D,E).

Step 4: Process message in 1024-bit (128-word) blocks. The heart of the algorithm is a module that consists of 80 rounds; this module is labeled F in Figure 11.8. The logic is illustrated in Figure 11.9.

Each round takes as input the 512-bit buffer value, abcdefgh, and updates the contents of the buffer. At input to the first round, the buffer has the value of the intermediate hash value, H_{i-1}. Each round t makes use of a 64-bit value W_t, derived from the current 1024-bit block being processed (M_i). These values are derived using a message schedule described subsequently. Each round also makes use of an additive constant K_t, where $0 \leq t \leq 79$ indicates one of the 80 rounds. These words represent the first 64 bits of the fractional parts of the cube roots of the first 80 prime numbers. The constants provide a "randomized" set of 64-bit patterns, which should eliminate any regularities in the input data. Table 11.4 shows these constants in hexadecimal format (from left to right).

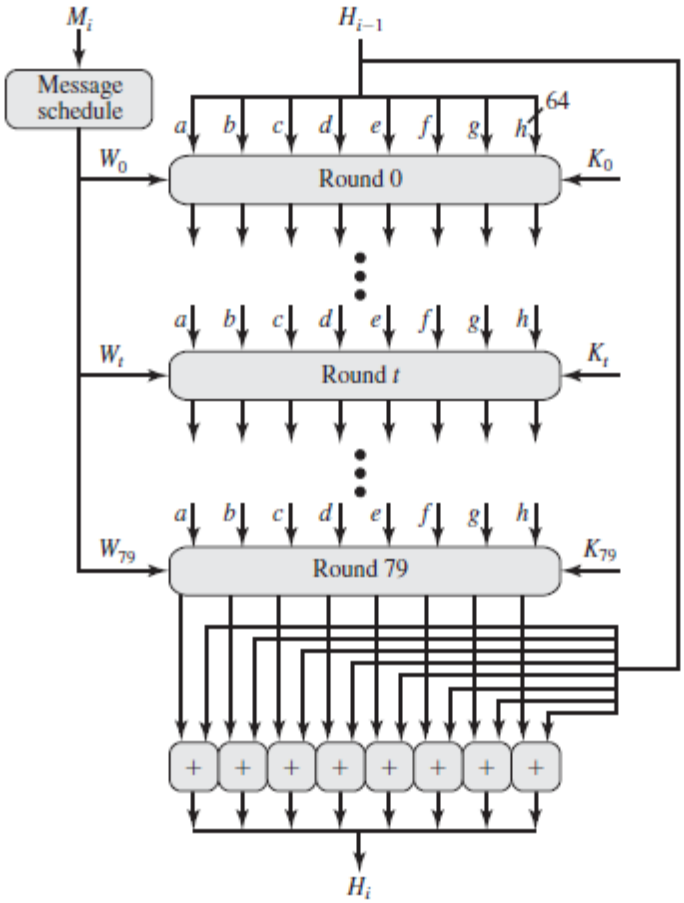


Figure 11.9 SHA-512 Processing of a Single 1024-Bit Block

Step 5 Output. After all N 1024-bit blocks have been processed, the output from the Nth stage is the 512-bit message digest.

We can summarize the behavior of SHA-512 as follows:

$$H_0 = \text{IV}$$
$$H_i = \text{SUM}_{64}(H_{i-1}, \text{abcdefgh}_i)$$
$$MD = H_N$$

where

IV = initial value of the abcdefgh buffer, defined in step 3

abcdefgh_i = the output of the last round of processing of the ith message block

N = the number of blocks in the message (including padding and length fields)

SUM_{64} = addition modulo 2^{64} performed separately on each word of the pair of inputs

MD = final message digest value

MESSAGE AUTHENTICATION

Message authentication is a mechanism or service used to verify the integrity of a message. Message authentication assures that data received are exactly as sent by (i.e., contain no modification, insertion, deletion, or replay) and that the purported identity of the sender is valid.

MESSAGE AUTHENTICATION REQUIREMENTS

In the context of communications across a network, the following attacks can be identified

1. Disclosure: Release of message contents to any person or process not possessing the appropriate cryptographic key.

2. Traffic analysis: Discovery of the pattern of traffic between parties. In a connection oriented application, the frequency and duration of connections could be determined.

3. Masquerade: Insertion of messages into the network from a fraudulent source.

4. Content modification: Changes to the contents of a message, including insertion, deletion, transposition, and modification.

5. Sequence modification: Any modification to a sequence of messages between parties, including insertion, deletion, and reordering.

6. Timing modification: Delay or replay of messages. In a connection-oriented application, an entire session or sequence of messages could be a replay of some previous valid session, or individual messages in the sequence could be delayed or replayed.

7. Source repudiation: Denial of transmission of message by source.

8. Destination repudiation: Denial of receipt of message by destination.

MESSAGE AUTHENTICATION FUNCTIONS

Any message authentication or digital signature mechanism has two levels of functionality. At the lower level, there must be some sort of function that produces an authenticator: a value to be used to authenticate a message. This lower-level function is then used as a primitive in a higher-level authentication protocol that enables a receiver to verify the authenticity of a message. there are 3 types of functions that may be used to produce an authenticator.

• **Hash function:** A function that maps a message of any length into a fixed length hash value, which serves as the authenticator
• **Message encryption:** The cipher text of the entire message serves as its authenticator
• **Message authentication code (MAC):** A function of the message and a secret key that produces a fixed-length value that serves as the authenticator

Message Encryption
Message encryption by itself can provide a measure of authentication. The analysis differs for symmetric and public-key encryption schemes.

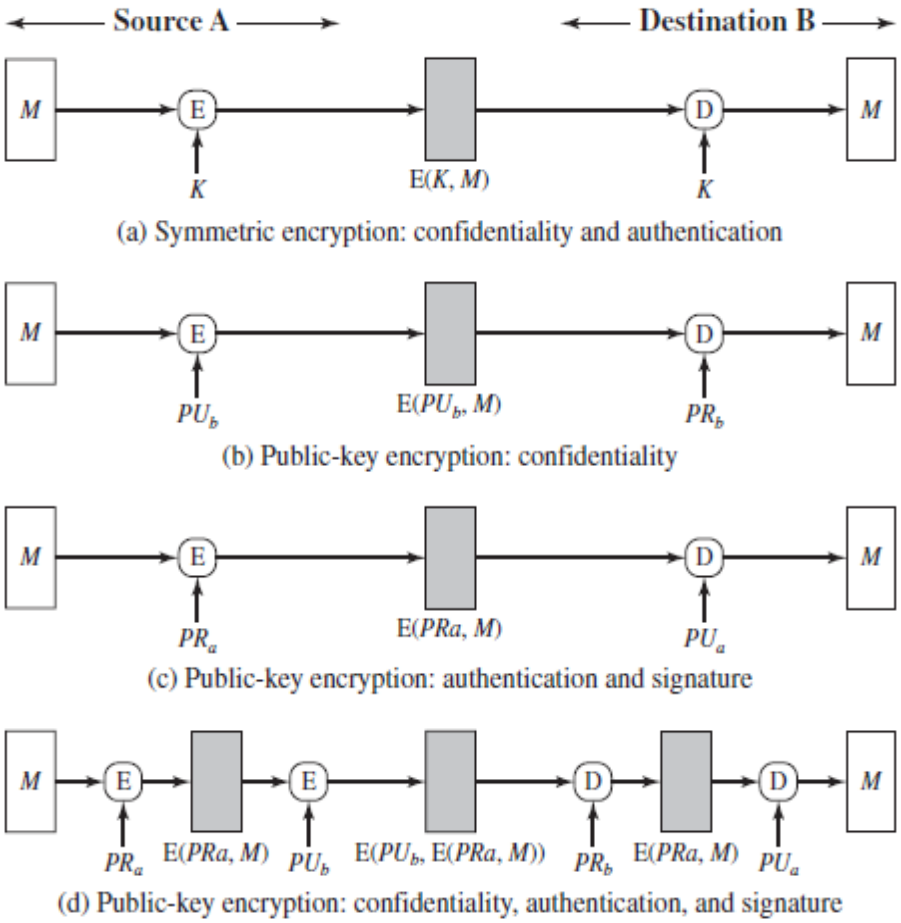


Figure 12.1 Basic Uses of Message Encryption

MESSAGE AUTHENTICATION CODE (MAC)

This authentication technique involves the use of a secret key to generate a small fixed-size block of data, known as a **cryptographic checksum** or MAC, that is appended to the message. This technique assumes that two communicating parties,say A and B, share a common secret key

When A has a message to send to B, it calculates the MAC as a function of the message and the key

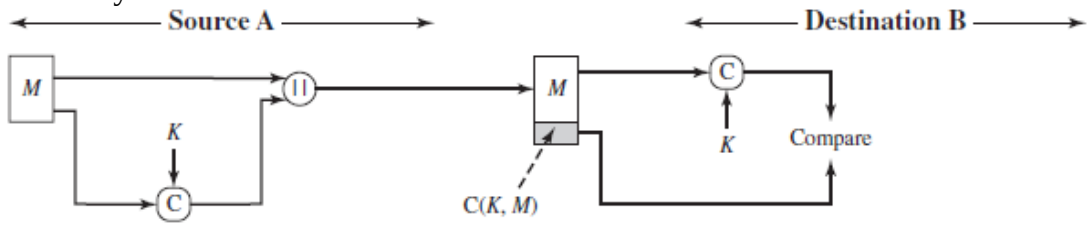
$$MAC = MAC(K, M)$$

where

M = input message
C = MAC function
K = shared secret key
MAC = message authentication code

The message plus MAC are transmitted to the intended recipient.The recipient performs the same calculation on the received message, using the same secret key, to generate a new MAC. The received MAC is compared to the calculated MAC (Figure 12.4a). If we assume that only the receiver and the sender know the identity of the secret key, and if the received MAC matches the calculated MAC, then

1. The receiver is assured that the message has not been altered. If an attacker alters the message but does not alter the MAC, then the receiver's calculation of the MAC will differ from the received MAC.
2. The receiver is assured that the message is from the alleged sender. Because no one else knows the secret key.



(a) Message authentication

SECURITY OF MACS:

Just as with symmetric and public-key encryption, we can group attacks on hash functions and MACs into two categories: **brute-force attacks and cryptanalysis.**

brute-force attacks

A brute-force attack on a MAC is a more difficult undertaking than a brute-force attack on a hash function because it requires known message-tag pairs. The strength of a hash function against brute-force attacks depends solely on the length of the hash code produced by the algorithm, with cost (2m/2). A brute-force attack on a MAC has cost related to min(2k, 2n), similar to symmetric encryption algorithms. It would appear reasonable to require that the key length and MAC length satisfy a relationship such as min(*k, n)* >= *N*, where N is perhaps in the range of 128 bits.

cryptanalysis.

As with encryption algorithms, cryptanalytic attacks on hash functions and MAC algorithms seek to exploit some property of the algorithm to perform some attack other than an exhaustive search. The way to measure the resistance of a hash or MAC algorithm to cryptanalysis is to compare its strength to the effort required for a bruteforce attack. That is, an ideal hash or MAC algorithm will require a cryptanalytic effort greater than or equal to the brute-force effort.

HMAC:

In recent years, there has been increased interest in developing a MAC derived from a cryptographic hash function, because they generally execute faster than symmetric block ciphers, and because code for cryptographic hash functions is widely available.

A hash function such as SHA 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,originally by just pre-pending a key to the message. Problems were found with these earlier, simpler proposals, but they resulted in the development of HMAC.

HMAC Design Objectives:

• To use, without modifications, available hash functions. In particular, to use 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 about the embedded hash function.

HMAC Algorithm:

HMAC Algorithm

Figure 12.5 illustrates the overall operation of HMAC. Define the following terms.

H = embedded hash function (e.g., MD5, SHA-1, RIPEMD-160)
IV = initial value input to hash function
M = message input to HMAC (including the padding specified in the embedded hash function)
Y_i = i th 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; recommended length is $\geq n$; if key length is greater than b, the key is input to the hash function to produce an n-bit key

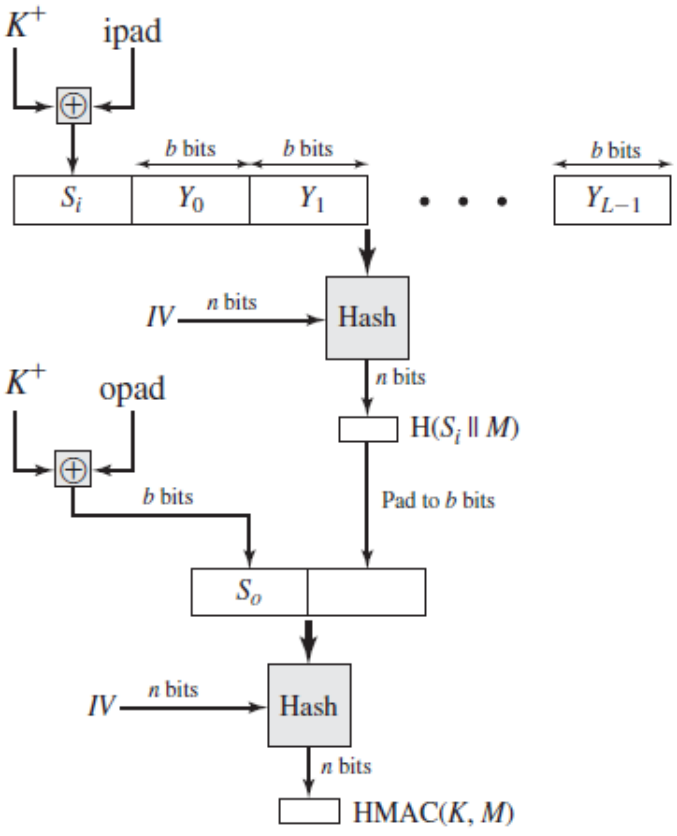
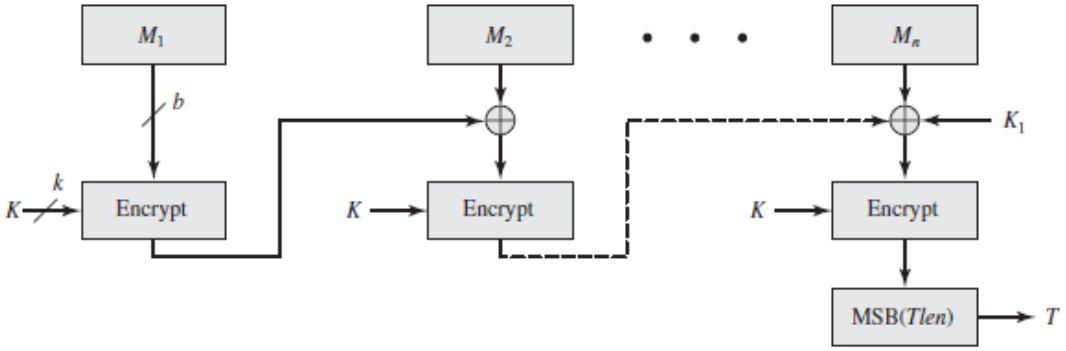


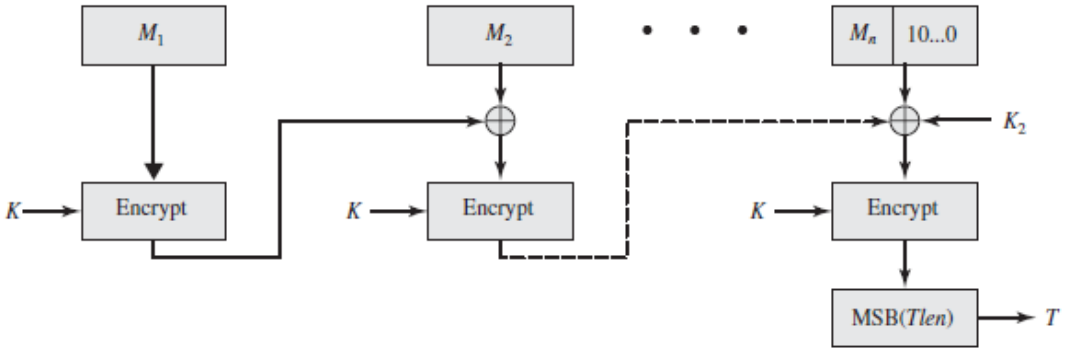
Figure 12.5 HMAC Structure

Cipher-Based Message Authentication Code (CMAC)

First, let us define 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 constant, 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 12.8).



(a) Message length is integer multiple of block size



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

Figure 12.8 Cipher-Based Message Authentication Code (CMAC)

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

where

T = message authetication 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.

DIGITAL SIGNATURES

A digital signature is an authentication mechanism that enables thecreator of a message to attach a code that acts as a signature. Typicallythe signature is formed by taking the hash of the message and encrypting the message with the creator's private key. The signature guarantees thesource and integrity of the message.

The digital signature standard (DSS) is an NIST standard that uses thesecure hash algorithm (SHA).

Properties

Message authentication protects two parties who exchange messages from any thirdparty. However, it does not protect the two parties against each other. Several formsof dispute between the two are possible.

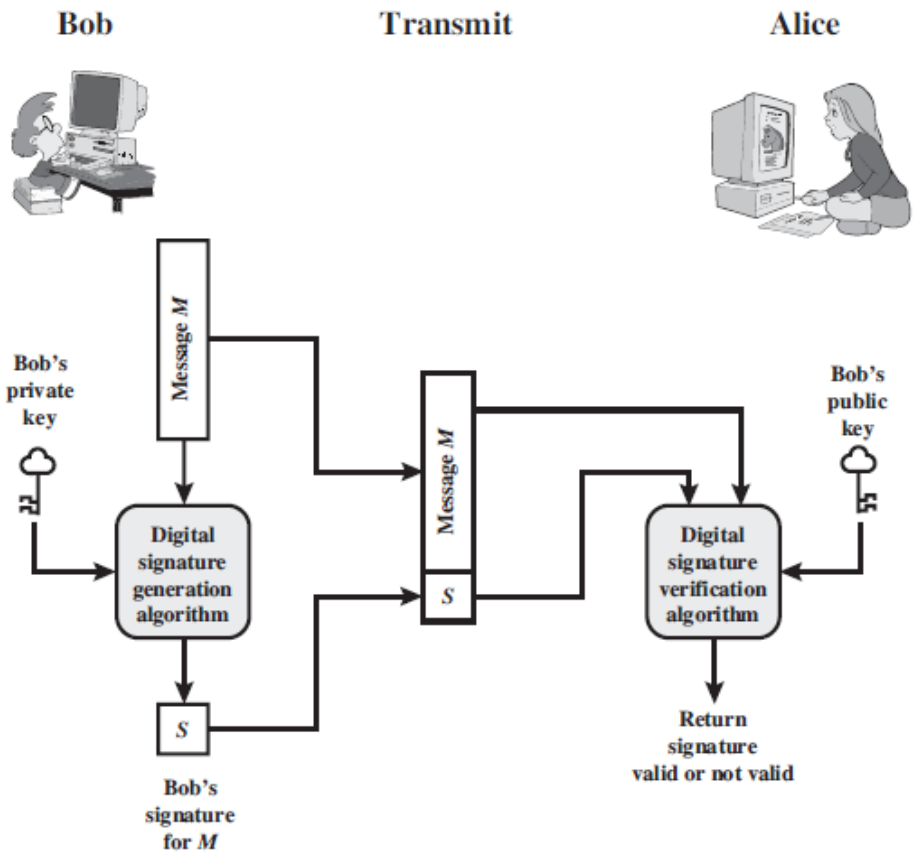


Figure 13.1 Generic Model of Digital Signature Process

DIGITAL SIGNATURE STANDARD

The Digital Signature Standard (DSS) makes use of the Secure Hash Algorithm (SHA) described and presents a new digital signature technique, the Digital Signature

Algorithm (DSA).

This latest version incorporates digital signature algorithms based on RSA and on elliptic curve cryptography. In this section, we discuss the original DSS algorithm. The DSS uses an algorithm that is designed to provide only the digital signature function.

Unlike RSA, it cannot be used for encryption or key exchange. Nevertheless, it is a public-key technique.

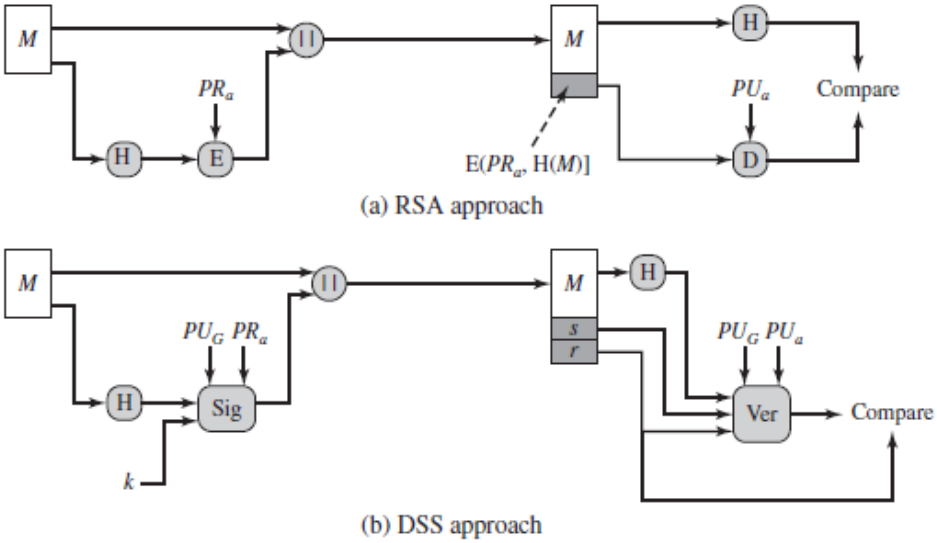


Figure 13.3 Two Approaches to Digital Signatures

In the RSA approach, the message to be signed is input to a hash function that produces a secure hash code of fixed length. This hash code is then encrypted using the sender's private key to form the signature. Both the message and the signature are thentransmitted. The recipient takes the message and produces a hash code.

The recipient also decrypts the signature using the sender's public key. If the calculated hash code matches the decrypted signature, the signature is accepted as valid. Because only the sender knows the private key, only the sender could have produced a valid signature.

Digital Signature Algorithm

The DSA is based on the difficulty of computing discrete logarithms and is based on schemes originally presented by ElGamal and Schnorr. The DSA signature scheme has advantages, being both smaller (320 vs 1024bit) and faster over RSA. Unlike RSA, it cannot be used for encryption or key exchange. Nevertheless, it is a public-key technique

DSA typically uses a common set of global parameters (p,q,g) for a community of clients, as shown. A 160-bit prime number q is chosen. Next, a prime number p is selected with a length between 512 and 1024 bits such that q divides (p – 1). Finally, g is chosen to be of the form $h^{(p-1)/q} \bmod p$ where h is an integer between 1 and (p – 1) withthe restriction that g must be greater than 1. Thus, the global public key components of DSA have the same for as in the Schnorr signature scheme.

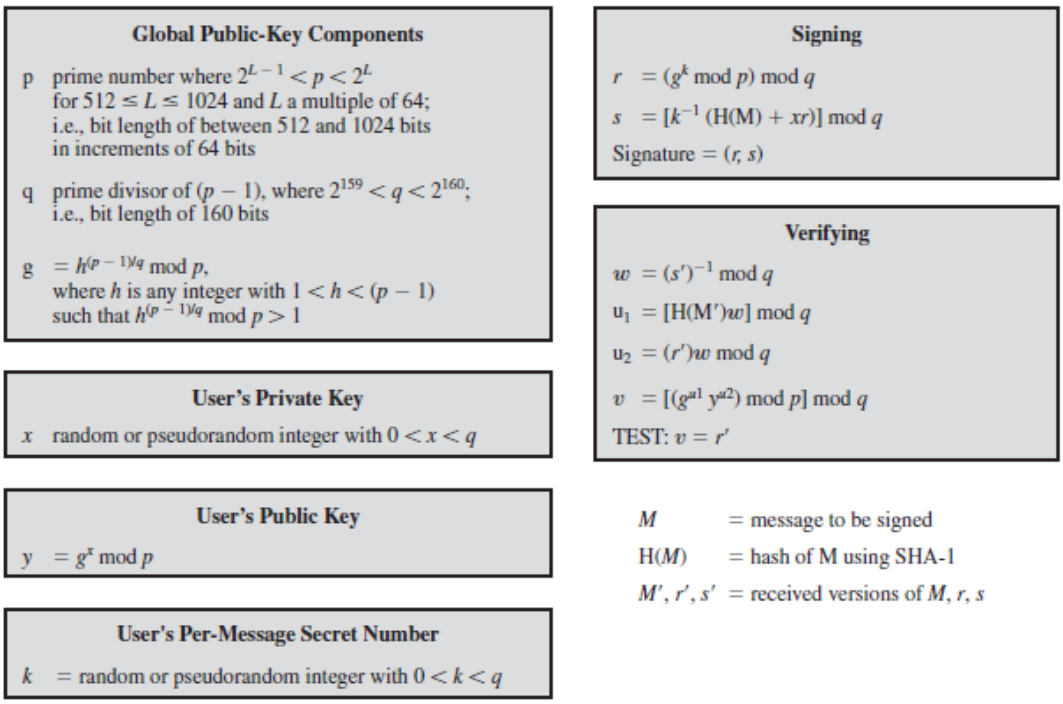
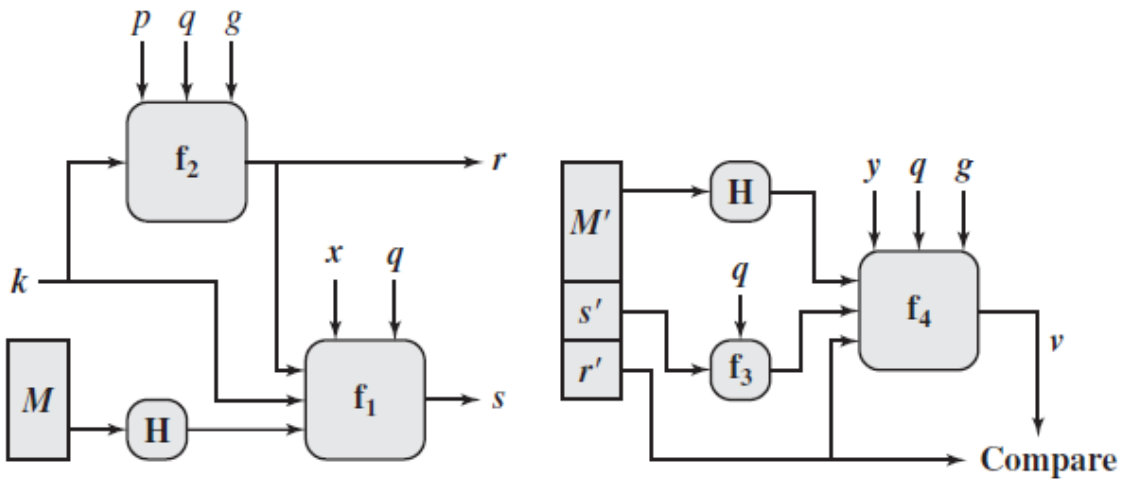


Figure 13.4 The Digital Signature Algorithm (DSA)

Signing and Verifying

The structure of the algorithm, as revealed here is quite interesting. Note that the test at the end is on the value r, which does not depend on the message at all. Instead, r is a function of k and the three global public-key components. The multiplicative inverse of k (mod q) is passed to a function that also has as inputs the message hash code and the user's private key. The structure of this function is such that the receiver can recover r using the incoming message and signature, the public key of the user, and the global public key.



$s = f_1(H(M), k, x, r, q) = (k^{-1} (H(M) + xr)) \bmod q$

$r = f_2(k, p, q, g) = (g^k \bmod p) \bmod q$

(a) Signing

$w = f_3(s', q) = (s')^{-1} \bmod q$

$v = f_4(y, q, g, H(M'), w, r')$

$= ((g^{(H(M')w) \bmod q} y^{r'w \bmod q}) \bmod p) \bmod q$

(b) Verifying

Figure 13.5 DSS Signing and Verifying

KEY MANAGEMENT AND DISTRIBUTION

Symmetric Key Distribution Using Symmetric Encryption For symmetric encryption to work, the two parties to an exchange must share the same key, and that key must be protected from access by others. Therefore, the term that refers to the means of delivering a key to two parties who wish to exchange data, without allowing others to see the key. For two parties A and B, key distribution can be achieved in a number of ways, as follows:

1. A can select a key and physically deliver it to B.

2. A third party can select the key and physically deliver it to A and B.

3. If A and B have previously and recently used a key, one party can transmit the new key to the other, encrypted using the old key.

4. If A and B each has an encrypted connection to a third party C, C can deliver a key on the encrypted links to A and B.

Physical delivery (1 & 2) is simplest - but only applicable when there is personal contact between recipient and key issuer. This is fine for link encryption where devices & keys occur in pairs, but does not scale as number of parties who wish to communicate grows.

3 is mostly based on 1 or 2 occurring first.A third party, whom all parties trust, can be used as a trusted intermediary to mediate the establishment of secure communications between them (4). Must trust intermediary not to abuse the knowledge of all session keys. As number of parties grow,some variant of 4 is only practical solution to the huge growth in number of keys potentially needed.

Key distribution centre:

 The use of a key distribution centeris based on the use of a hierarchy of keys. At a minimum, two levels of keys are used.

 Communication between end systems is encrypted using a temporary key, often referred to as a Session key.

 Typically, the session key is used for the duration of a logical connection and then discarded

 Master key is shared by the key distribution center and an end system or user and used to

encrypt the session key.

A Key Distribution Scenario

The key distribution concept can be deployed in a number of ways. A typical scenario is illustrated in Figure 14.3, which is based on a figure in [POPE79]. The scenario assumes that each user shares a unique master key with the key distribution center (KDC).

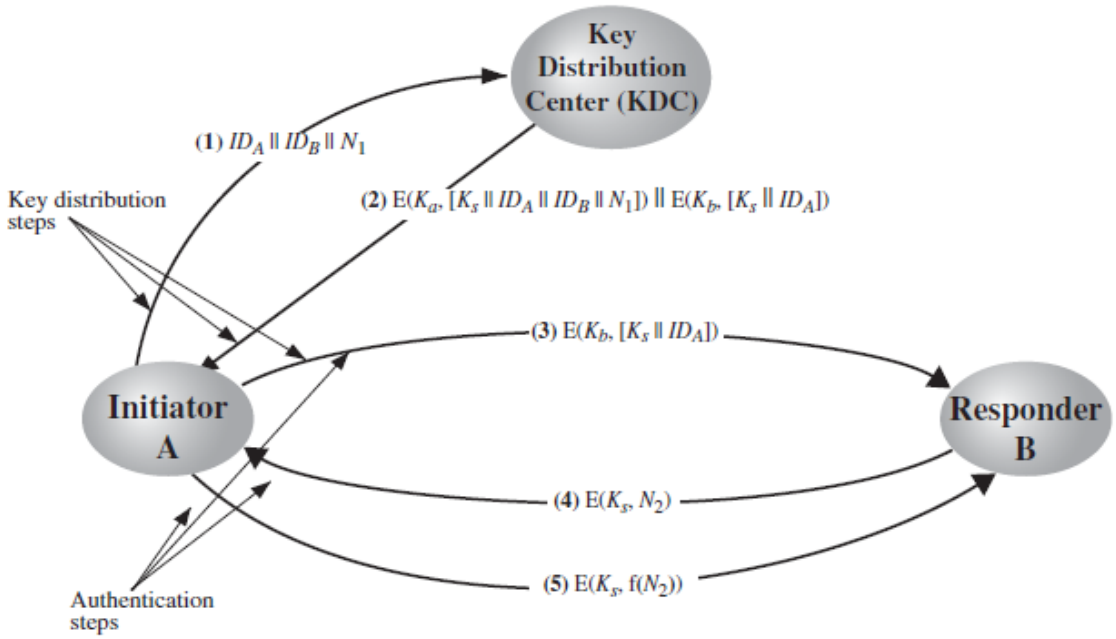


Figure 14.3 Key Distribution Scenario

Symmetric Key Distribution Usingasymmetric Encryption

Because of the inefficiency of public key cryptosystems, they are almost never usedfor the direct encryption of sizable block of data, but are limited to relatively smallblocks. One of the most important uses of a public-key cryptosystem is to encryptsecret keys for distribution.Wesee many specific examples of this in Part Five. Here,we discuss general principles and typical approaches.

Simple Secret Key Distribution

An extremely simple scheme was put forward by Merkle [MERK79], as illustrated in Figure 14.7. If A wishes to communicate with B, the following procedure is employed:

1. A generates a public/private key pair $\{PU_a, PR_a\}$ and transmits a message to B consisting of PU_a and an identifier of A, ID_A.
2. B generates a secret key, K_s, and transmits it to A, which is encrypted with A's public key.
3. A computes $D(PR_a, E(PU_a, K_s))$ to recover the secret key. Because only A can decrypt the message, only A and B will know the identity of K_s.
4. A discards PU_a and PR_a and B discards PU_a.

A and B can now securely communicate using conventional encryption and the session key K_s. At the completion of the exchange, both A and B discard K_s.

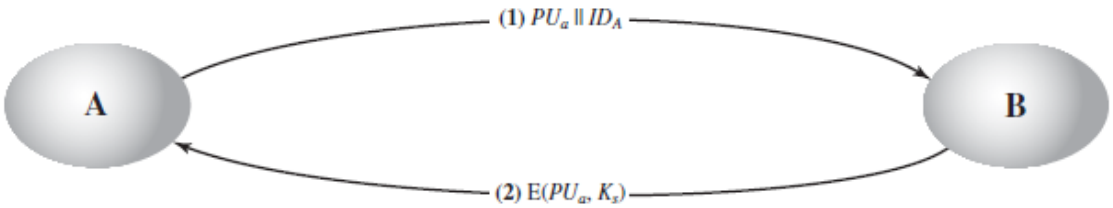


Figure 14.7 Simple Use of Public-Key Encryption to Establish a Session Key