UNIT - II: PUBLIC -KEY CRYPTOGRAPHY: Approaches of Message Authentication, Secure Hash Functions (SHA-1, SHA-512) and HMAC Algorithm, Public Key Cryptography principles, Public Key Cryptography Algorithms, Digital Signatures, Public Key Infrastructure, Digital Certificates, Certificate Authority, Key Management,Kerberos,X.509 Directory Authentication Service.

TEXTBOOK William Stallings, Network Security Essentials (Applications and Standards), Pearson Education.

UNIT-II

APPROACHES TO MESSAGE AUTHENTICATION

Message authentication is a procedure that allows communicating parties to verify that received messages areauthentic.

The two important aspects are to verify that the contents of the message have not been altered and that the source is authentic. We may also wish to verify a message's timeliness (it has not been artificially delayed and replayed) and sequence relative to other messages flowing between two parties.

Authentication Using Conventional Encryption

It would seem possible to perform authentication simply by the use of symmetric encryption. If we assume that only the sender and receiver share a key (which is as it should be), then only the genuine sender would be able to encrypt a message successfully for the other participant, provided the receiver can recognize a valid message.

Furthermore, if the message includes an error-detection code and a sequence number, the receiver is assured that no alterations have been made and that sequencing is proper. If the message also includes a timestamp, the receiver isassured that the message has not been delayed beyond that normally expected for network transit.

In fact, symmetric encryption alone is not a suitable tool for data authentication.To give one simple example, in the ECB mode of encryption, if an attackerreorders the blocks of ciphertext, then each block will still decrypt successfully.However, the reordering may alter the meaning of the overall data sequence.Although sequence numbers may be used at some level (e.g., each IP packet), that a separate sequence number will not be associated with each*b*-bit block of plaintext.Thus, block reordering is a threat.

Message Authentication without Message Encryption

We examine several approaches to message authentication that do not rely on encryption. In all of these approaches, an authentication tag is generated and appended to each message for transmission.The message itself is not encrypted and can be read at the destination independent of the authentication function at the destination.

Message authentication is provided as a separate function from message encryption. [DAVI89] suggests three
situations in which message authentication without confidentiality is preferable:

1. There are a number of applications in which the same message is broadcast toa number ofdestinations.Two examples are notification to users that the networkis now unavailable and an alarm signal in a control center. Thus, the message must be broadcast in plaintext with an associatedmessage authentication tag. The responsible system performs authentication.
2. Another possible scenario is an exchange in which one side has a heavy load and cannot afford the time to decrypt all incoming messages. Authentication is carried out on a selective basis with messages being chosen at random for checking.
3. Authentication of a computer program in plaintext is an attractive service. The computer program can be executed without having to decrypt it every time, which would be wasteful of processor resources. However, if a message authentication tag were attached to the program, it could be checked whenever
assurance is required of the integrity of the program.

MESSAGE AUTHENTICATION CODE

One authentication technique involves the use of a secret key to generate a small block of data, known as a **message authentication code(MAC)**, that is appended to the message. This technique assumes that two communicating parties, say A and B, share a common secret key *KAB*. When A has a message to send to B, it calculates the message authentication code as a function of the message and the key: MAC*M* = F(*KAB*,*M*).The message plus code 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 message authentication code.
The received code is compared to the calculated code (Figure 3.1). If we assume that only the receiver and the sender know the identity of the secret key, and if the received code matches the calculated code, then the following statements apply:

1. The receiver is assured that the message has not been altered. If an attackeralters the message but does not alter the code, then the receiver's calculationof the code will differ from the received code. Because the attacker is assumednot to know the secret key, the attacker cannot alter the code to correspond to
the alterations in the message.

2. The receiver is assured that the message is from the alleged sender. Because no oneelse knows the secret key, no one else could prepare a message with a proper code.

3. If the message includes a sequence number (such as is used with HDLC andTCP), then the receiver can be assured of the proper sequence, because anattacker cannot successfully alter the sequence number.

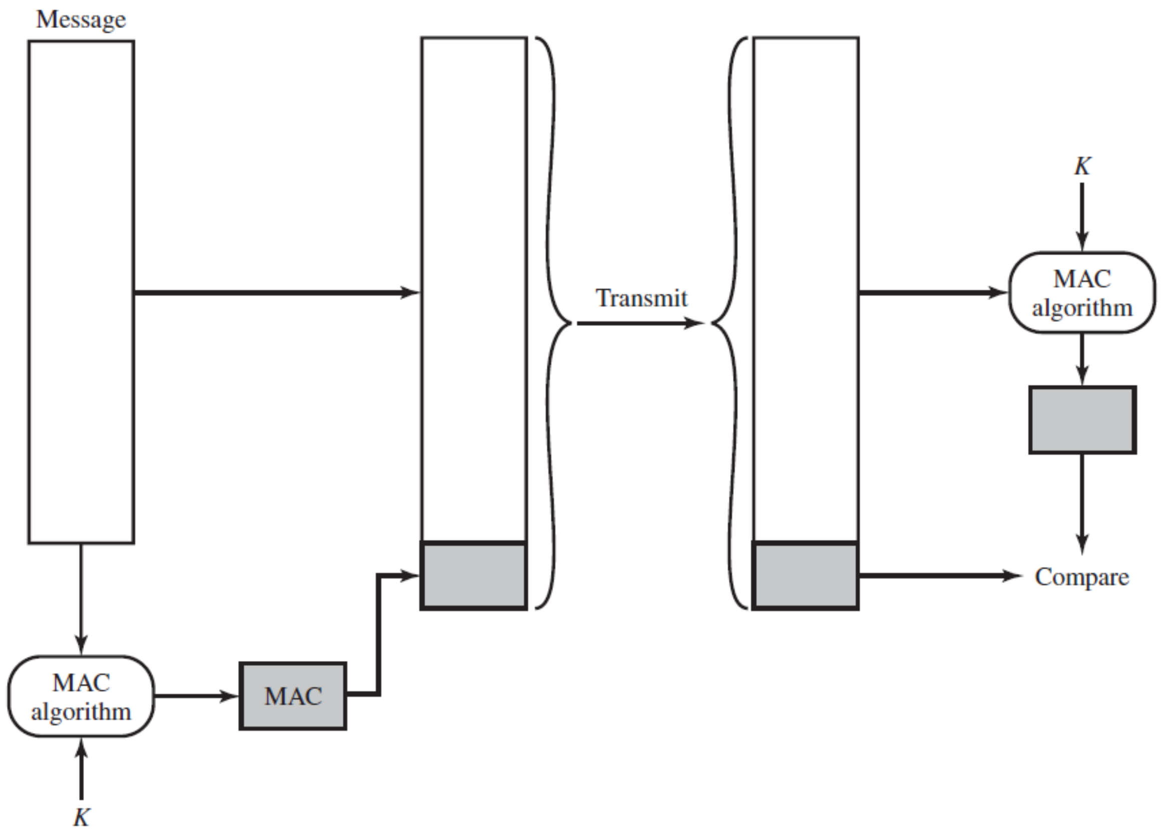


Figure 3.1 Message Authentication Using a Message Authentication Code (MAC)

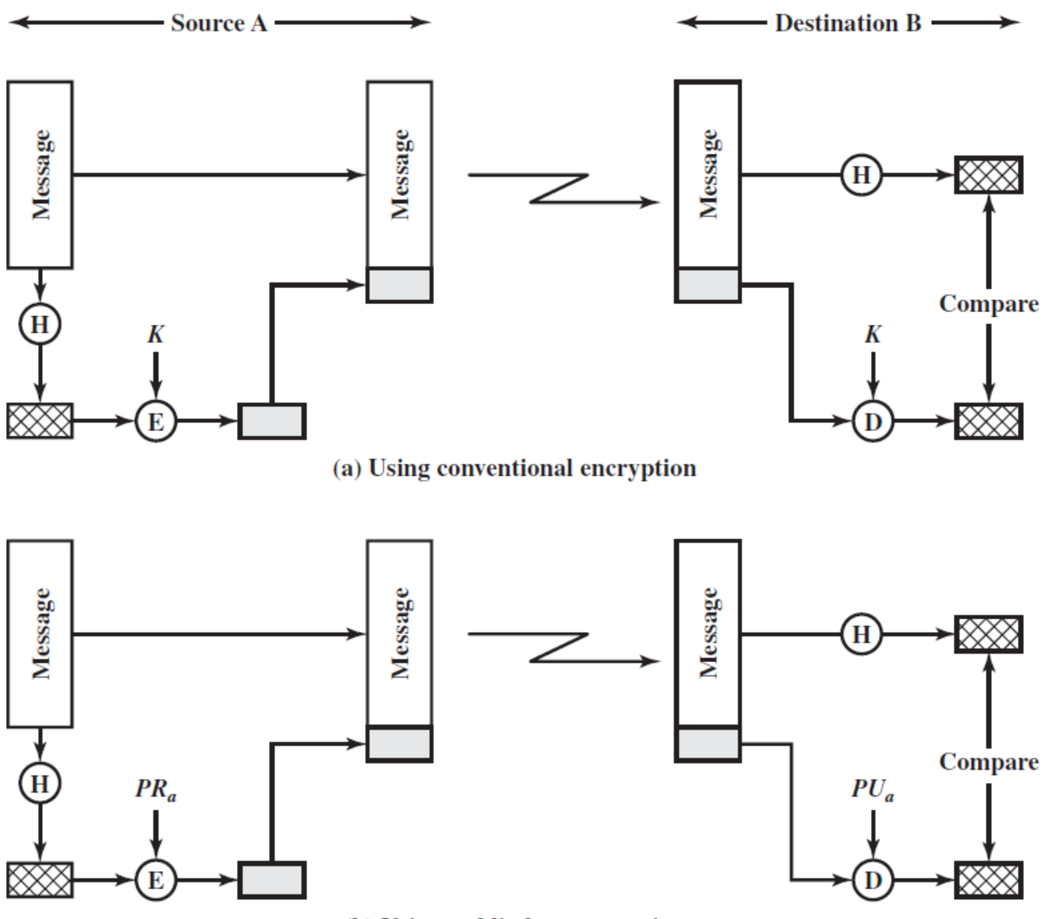
DES is used to generate an encrypted version of the message, and the last number of bits of ciphertext are used
as the code. A 16- or 32-bit code is typical. The process just described is similar to encryption. One difference is that the authentication algorithm need not be reversible.

ONE-WAY HASH FUNCTION

An alternative to the message authentication code is the **one-way hash function**. As with the message authentication code, a hash function accepts a variable-size message *M* as input and produces a fixed-size
message digest H(*M*) as output. Unlike the MAC, a hash function does not take a secret key as input. To authenticate a message, the message digest is sent with the message in such a way that the message digest is authentic.

Figure 3.2 illustrates three ways in which the message can be authenticated. The message digest can be encrypted using conventional encryption (part a); if it is assumed that only the sender and receiver share the encryption key, then authenticity is assured. The message digest can be encrypted using public-key encryption (part b).

The public-key approach has two advantages:
(1) It provides a digital signature as well as message authentication.
(2) It doesnot require the distribution of keys to communicating parties.

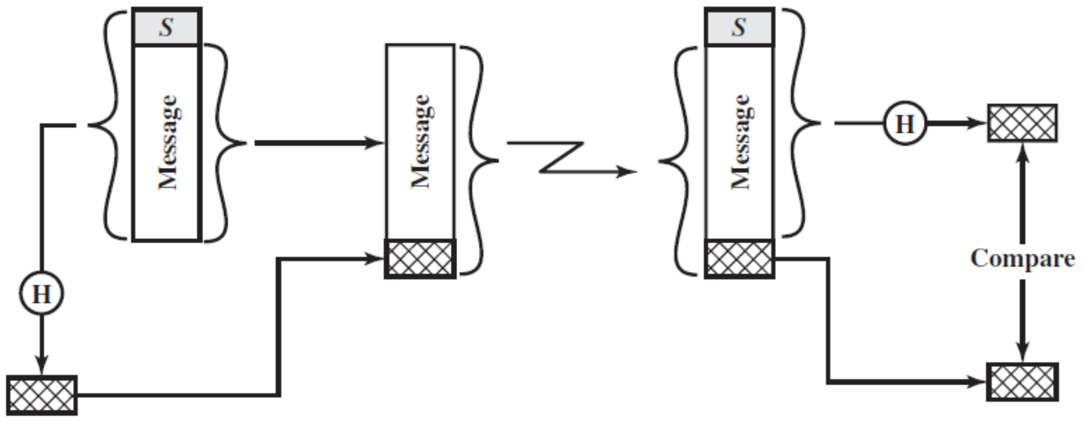


(a) Using conventional encryption

(b) Using public-key encryption

These two above approaches also have an advantage over approaches that encrypt the entire message in that less computation is required.

Figure 3.2c shows a technique that uses a hash function but no encryption for message authentication. This technique assumes that two communicating parties,say A and B, share a common secret value *SAB*.When A has a message to send to B,it calculates the hash function over the concatenation of the secret value and themessage: *MDM*=H(*SAB*||*M*).2 It then sends [*M*||*MDM*] to B. Because B possesses*SAB*, it can recompute H(*SAB*||*M*) and verify *MDM.* Because the secret value itself is not sent, it is not possible for an attacker to modify an intercepted message. As long as the secret value remains secret, it is also not possible for an attacker to generate a false message.A variation on the third technique, called HMAC, is the one adopted for IP



(c) Using secret value

Figure 3.2 Message Authentication Using a One-Way Hash Function

There has been interest in developing a technique that avoids encryption altogether. Several reasons
for this interest are pointed out in.

- Encryption software is quite slow. Even though the amount of data to be encrypted per message is small, there may be a steady stream of messages into and out of a system.
- Encryption hardware costs are nonnegligible. Low-cost chip implementations of DES are available, but the cost adds up if all nodes in a network must have this capability.
- Encryption hardware is optimized toward large data sizes. For small blocks of data, a high proportion of the time is spent in initialization/invocation overhead

SECURE HASH FUNCTIONS

The one-way hash function, or **secure hash function**, is important not only in message authentication but in digital signatures.

Hash Function Requirements

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:

1. H can be applied to a block of data of any size.
2. H produces a fixed-length output.
3. H(*x*) is relatively easy to compute for any given *x*, making both hardware and software implementations practical.
4. For any given code *h*, it is computationally infeasible to find *x* such that H(*x*) = *h*. A hash function with this property is referred to as **one-way** or **preimageresistant**.
5. For any given block *x*, it is computationally infeasible to find *y not equal to x* with H(*y*) not equal to H(*x*). A hash function with this property is

referred to as **second preimage resistant**. This is sometimes referred to as **weak collision resistant**.
6. It is computationally infeasible to find any pair (x, y) such that H(x) = H(y). A hash function with this property is referred to as **collision resistant**. This is sometimes referred to as **strong collision resistant**.

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

Security of Hash Functions

As with symmetric encryption, there are two approaches to attacking a secure hash function: cryptanalysis and brute-force attack.As with symmetric encryption algorithms, cryptanalysis of a hash function involves exploiting logical weaknesses in the algorithm. The strength of a hash function against brute-force attacks depends solely on the length of the hash code produced by the algorithm. For a hash code of length n, the level of effort required is proportional to the following

Preimage resistant	2^n
Second preimage resistant	2^n
Collision resistant	$2^{n/2}$

With a hash length of 160 bits, the same search machine would require over four thousand years to find a collision. With today's technology, the time would be much shorter, so that 160 bits now appears suspect.

Simple Hash Functions

All hash functions operate using the following general principles. The input (message, file, etc.) is viewed as a sequence of n-bit blocks. The input is processed one block at a time in an iterative fashion to produce an n-bit hash function. One of the simplest hash functions is the bit-by-bit exclusive-OR (XOR) of
every block. This can be expressed as

$$C_i = b_{i1} \oplus b_{i2} \oplus \ldots \oplus b_{im}$$

where

C_i = ith bit of the hash code, $1 \leq i \leq n$

m = number of n-bit blocks in the input

b_{ij} = ith bit in jth block

$\oplus$ = XOR operation

	bit 1	bit 2	• • •	bit n
Block 1	b_{11}	b_{21}		b_{n1}
Block 2	b_{12}	b_{22}		b_{n2}
	⋮	⋮	⋮	⋮
Block m	b_{1m}	b_{2m}		b_{nm}
Hash code	C_1	C_2		C_n

Figure 3.3 Simple Hash Function Using Bitwise XOR

The SHA Secure Hash Function

In recent years, the most widely used hash function has been the Secure Hash Algorithm (SHA). Indeed, because virtually every other widely used hash function had been found to have substantial cryptanalytic weaknesses, SHA was more or less the last remaining standardized hash algorithm by 2005. SHA was developed by the National Institute of Standards and Technology (NIST) and published as a federalinformation processing standard (FIPS 180) in 1993.When weaknesses were discoveredin SHA (now known as SHA-0), a revised version was issued as FIPS 180-1 in1995 and is referred to as **SHA-1**.

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. Collectively, these hash algorithms are known as **SHA-2**. These
new versions have the same underlying structure and use the same types of modular arithmetic and logical binary operations as SHA-1.

The algorithm takes as input a message with a maximum length of less than 2^{128} bits and produces as output a 512-bit message digest. The input is processed in 1024-bit blocks. Figure 3.4 depicts the overall processing of a message to produce a digest. The processing consists of the following steps.

Table 3.1 Comparison of SHA Parameters

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Message Digest Size	160	224	256	384	512
Message Size	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block Size	512	512	512	1024	1024
Word Size	32	32	32	64	64
Number of Steps	80	64	64	80	80
Security	80	112	128	192	256

Notes: 1. All sizes are measured in bits.
2. Security refers to the fact that a birthday attack on a message digest of size n produces a collision with a workfactor of approximately $2^{n/2}$.

Step 1 Append padding bits: The message is padded so that its length is congruent to 896 modulo 1024 [length 896 (mod 1024)]. Padding is always added, even if the message is already of the desired length. Thus, the number of padding bits is in the range of 1 to 1024.The padding consists of a single 1 bit
followed by the necessary number of 0 bits.

Step 2 Append length: A block of 128 bits is appended to the message. This block is treated as an unsigned 128-bit integer (most significant byte first) and contains the length of the original message (before the padding). The outcome of the first two steps yields a message that is an integer multiple of 1024 bits in length. In Figure 3.4, the expanded message is represented as the sequence of 1024-bit blocks *M*1, *M*2, . . ., *MN*, so that the total
length of the expanded message is $N \times 1024$ bits.

Step 3 Initialize hash buffer: A 512-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as eight 64-bit registers (*a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*).These registers are initialized to the following 64-bit integers (hexadecimal values):

a= 6A09E667F3BCC908 *e* = 510E527FADE682D1
b = BB67AE8584CAA73B *f* = 9B05688C2B3E6C1F
c = 3C6EF372FE94F82B *g* = 1F83D9ABFB41BD6B
d = A54FF53A5F1D36F1 *h* = 5BE0CD19137E2179

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 3.4.

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

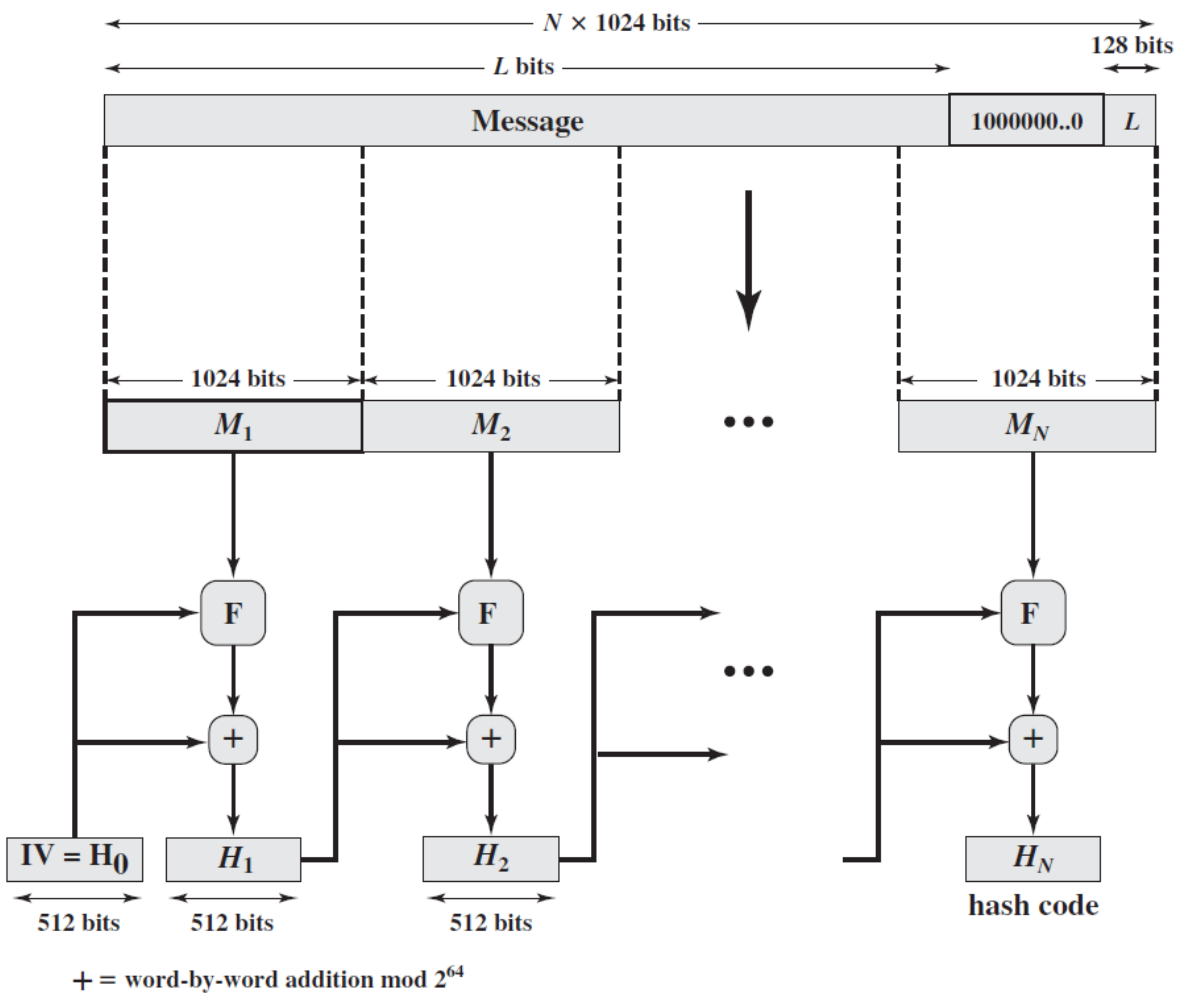


Figure 3.4 Message Digest Generation Using SHA-512

WORKING OF SHA-512 WITH 80 ROUNDS

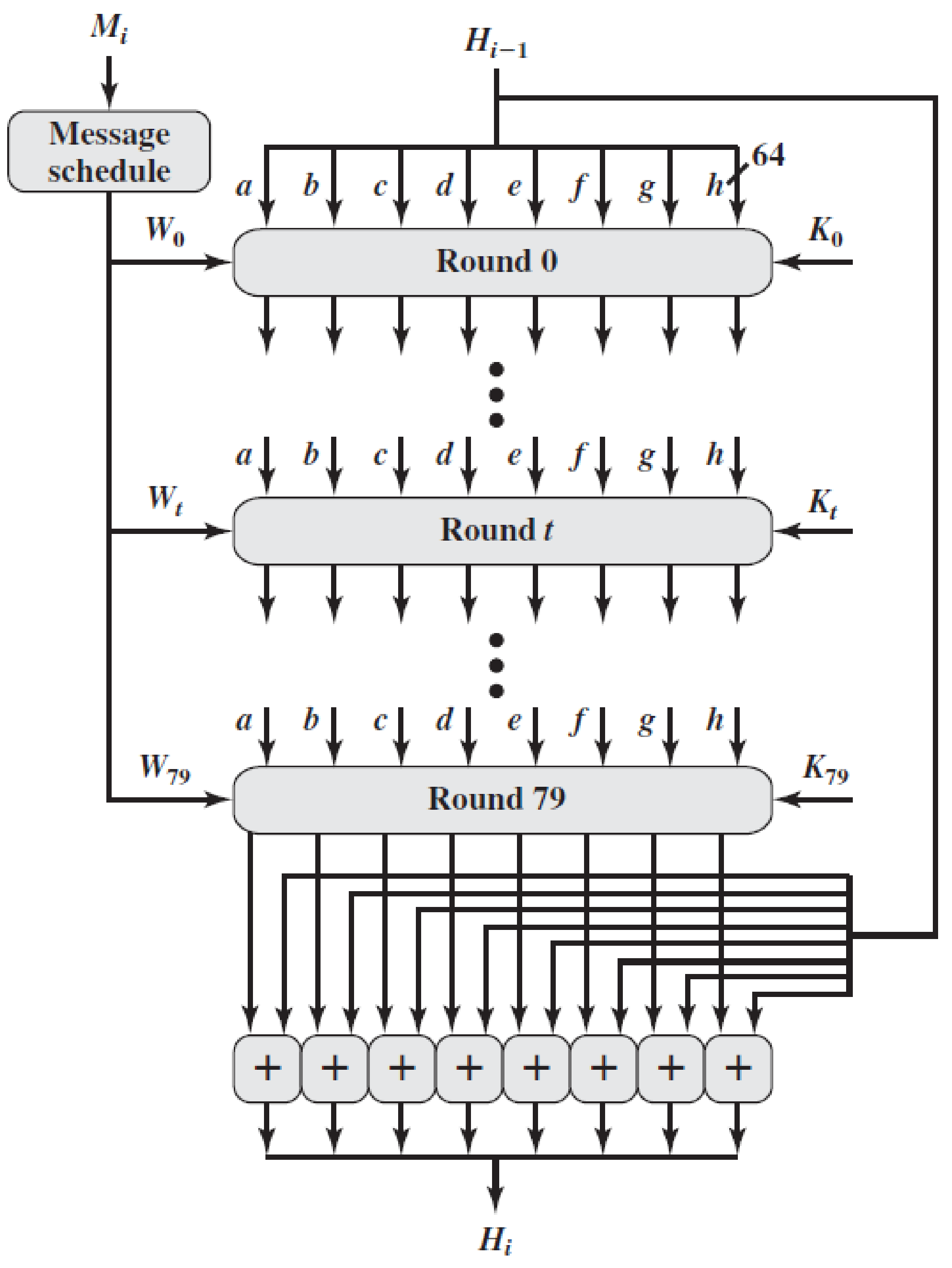


Figure 3.5 SHA-512 Processing of a Single 1024-Bit Block

MESSAGE AUTHENTICATION CODES-HMAC

In recent years, there has been increased interest in developing a MAC derivedfrom a cryptographic hash code, such as SHA-1.The motivations for this interest are
• Cryptographic hash functions generally execute faster in software than conventionalencryption 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 pre-packaged 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.

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)
Yi = *i*th block of *M*,
L = number of blocks in *M*

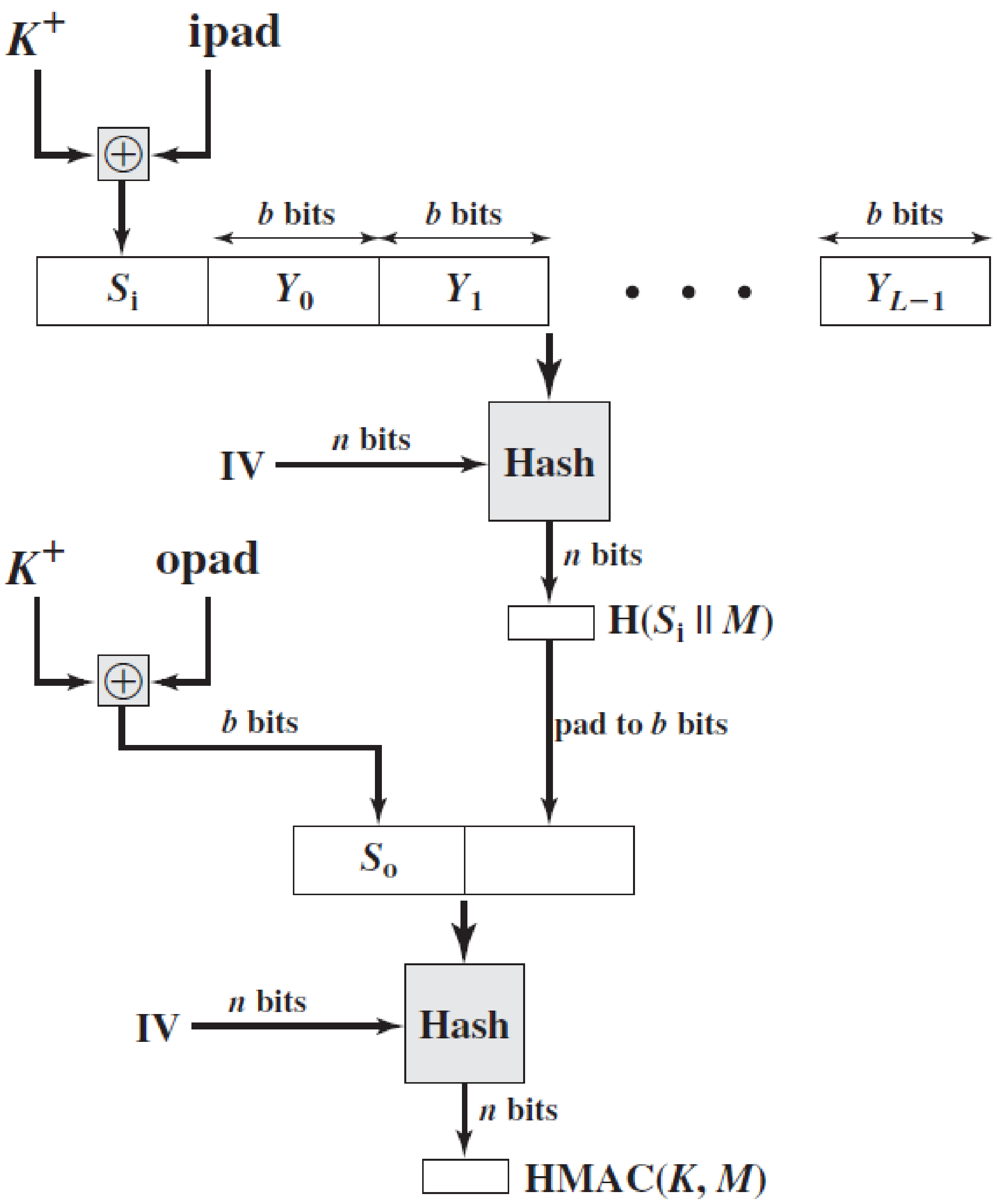


Figure 3.6 HMAC Structure

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 *Si* .
3. Append *M* to *Si*.
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

MACs Based on Block Ciphers

CIPHER-BASED MESSAGE AUTHENTICATION CODE (CMAC) The Cipher-basedMessage Authentication Code (CMAC) mode of operation is for use with AES andtriple DES.

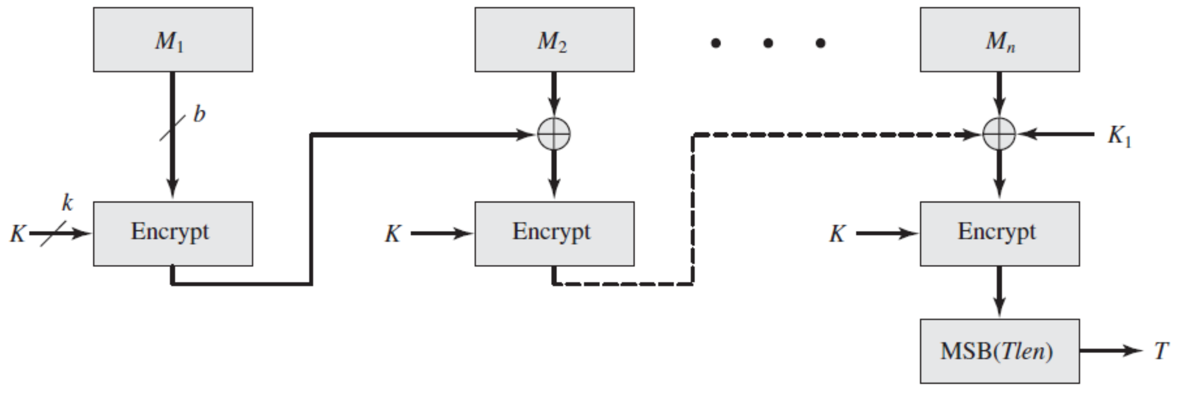
First, let us consider the operation of CMAC when the message is an integer multiple *n* of the cipher block length *b*. For AES, *b* 128, and for triple DES, *b* 64. The message is divided into *n* blocks (*M*1,*M*2, . . .,*Mn*).The algorithm makes use of a *k*-bit encryption key *K* and an *n*-bit key, *K*1. For AES, the key size *k* is 128, 192, or256 bits; for triple DES, the key size is 112 or 168 bits. CMAC is calculated as follows.

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

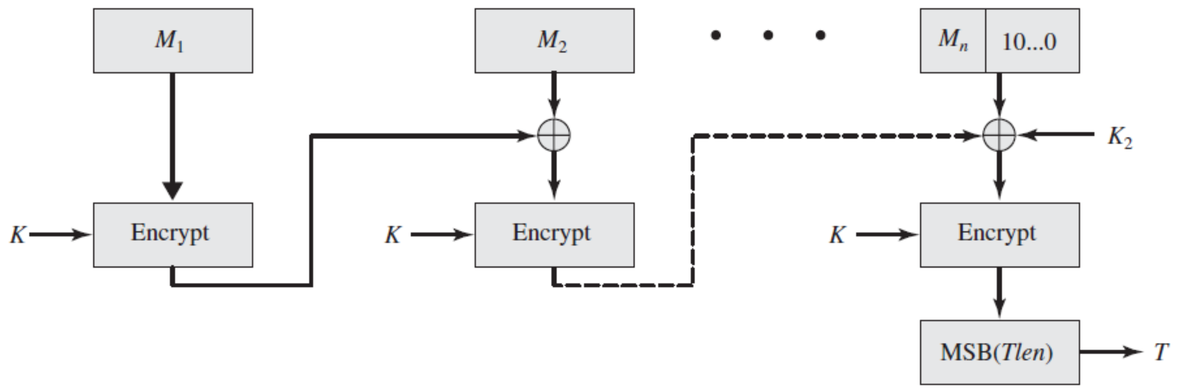
where

T	= message 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.*



(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)

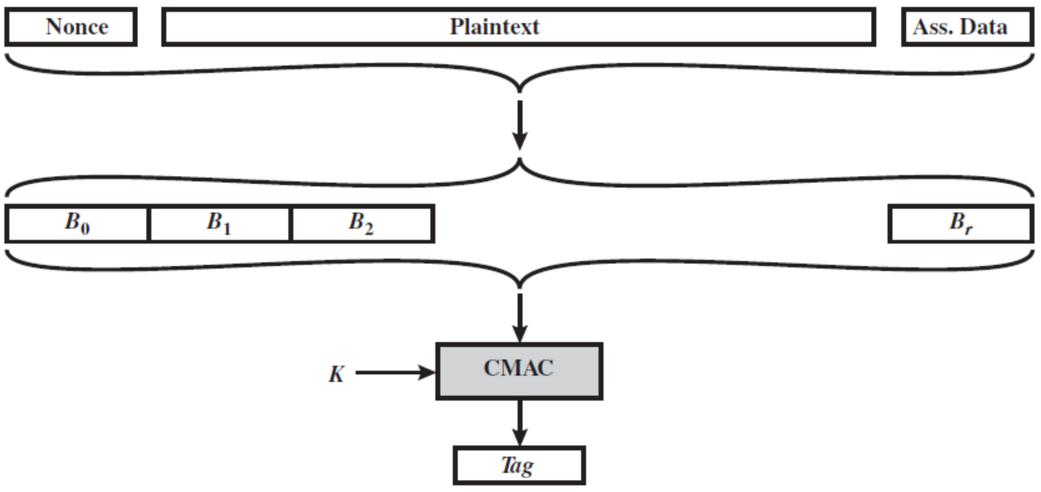
COUNTER WITH CIPHER BLOCK CHAINING-MESSAGE AUTHENTICATION CODE The CCM mode of operation, defined in NIST SP 800-38C, is referred to as an **authenticated encryption** mode. The key algorithmic ingredients of CCM are the AES encryption algorithm.

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.

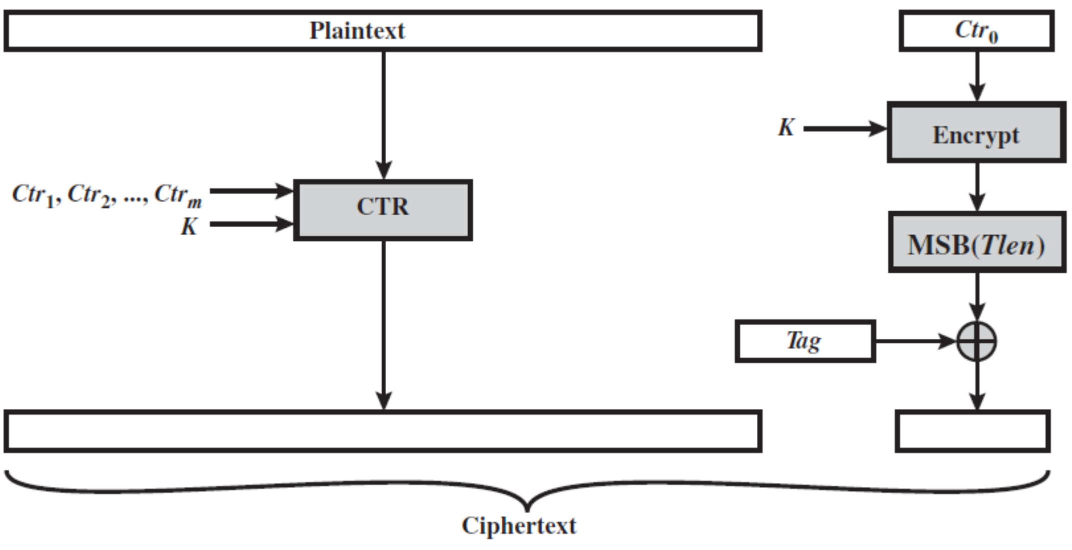
2. Associated data *A* that will be authenticated but not encrypted. An example is a protocol header that must be transmitted in the clear for proper protocol operation but which needs to be authenticated.

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.



(a) Authentication

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 toproduce an encrypted tag. The remaining counters are used for the CTR modeencryption of the plaintext. The encrypted plaintext is concatenatedwith the encrypted tag to form the ciphertext output.



(b) Encryption

Figure 3.8 Counter with Cipher Block Chaining-Message Authentication Code (CCM)

PUBLIC-KEY CRYPTOGRAPHY PRINCIPLES

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.

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.

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

- **Plaintext:** This is the readable message or data that is fed into the algorithm as input.
- **Encryption algorithm:**The encryption algorithm performs various transformationson 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.

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. 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.

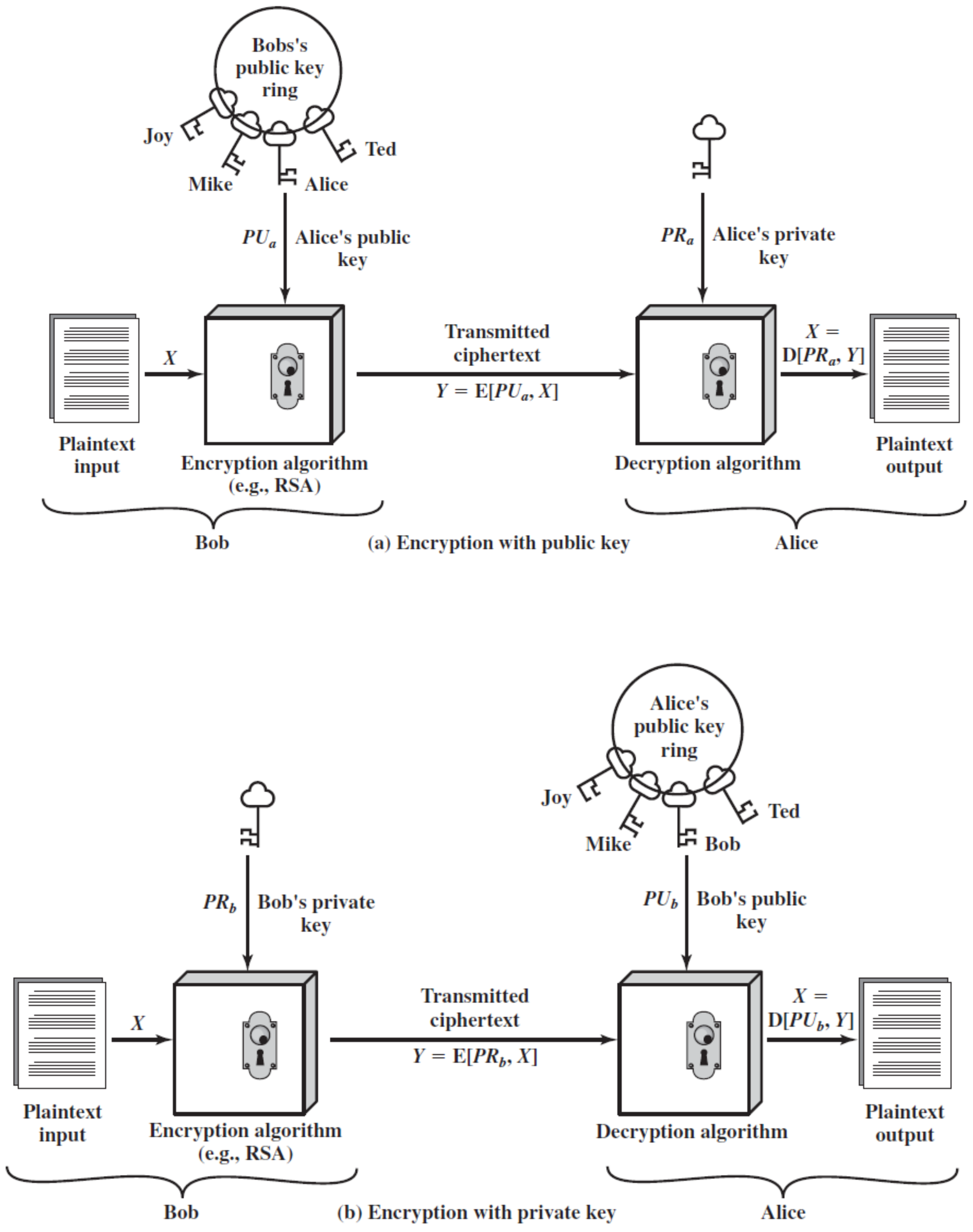


Figure 3.9 Public-Key Cryptography

Applications for Public-Key Cryptosystems

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.

Table 3.2 Applications for Public-Key Cryptosystems

Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No
Elliptic curve	Yes	Yes	Yes

Requirements for Public-Key Cryptography

The cryptosystem illustrated in Figure 3.9 depends on a cryptographic algorithm based on two related keys. Diffie and Hellman postulated this system without demonstrating that such algorithms exist. However, they did lay out the conditions that such algorithms must fulfill [DIFF76]:

1. It is computationally easy for a party B to generate a pair (public key PU_b, private key PR_b).
2. It is computationally easy for a sender A, knowing the public key and the message to be encrypted, M, to generate the corresponding ciphertext:

$$C = \mathrm{E}(PU_b, M)$$

3. It is computationally easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message:

$$M = \mathrm{D}(PR_b, C) = \mathrm{D}[PR_b, \mathrm{E}(PU_b, M)]$$

4. It is computationally infeasible for an opponent, knowing the public key, PU_b, to determine the private key, PR_b.
5. It is computationally infeasible for an opponent, knowing the public key, PU_b, and a ciphertext, C, to recover the original message, M.

PUBLIC-KEY CRYPTOGRAPHY ALGORITHMS

The two most widely used public-key algorithms are RSA and Diffie-Hellman.

(i) The RSA Public-Key Encryption Algorithm

One of the first public-key schemes was developed in 1977 by Ron Rivest, AdiShamir, and Len Adleman at MIT and first published in 1978. The RSAscheme has since that time reigned supreme as the most widely accepted and implementedapproach to public-key encryption. **RSA** is a block cipher in which theplaintext and ciphertext are integers between 0 and n-1 for some n.

Encryption and decryption are of the following form period for some plaintextblock M and ciphertext block C:

$$C = M^e \bmod n$$

$$M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$$

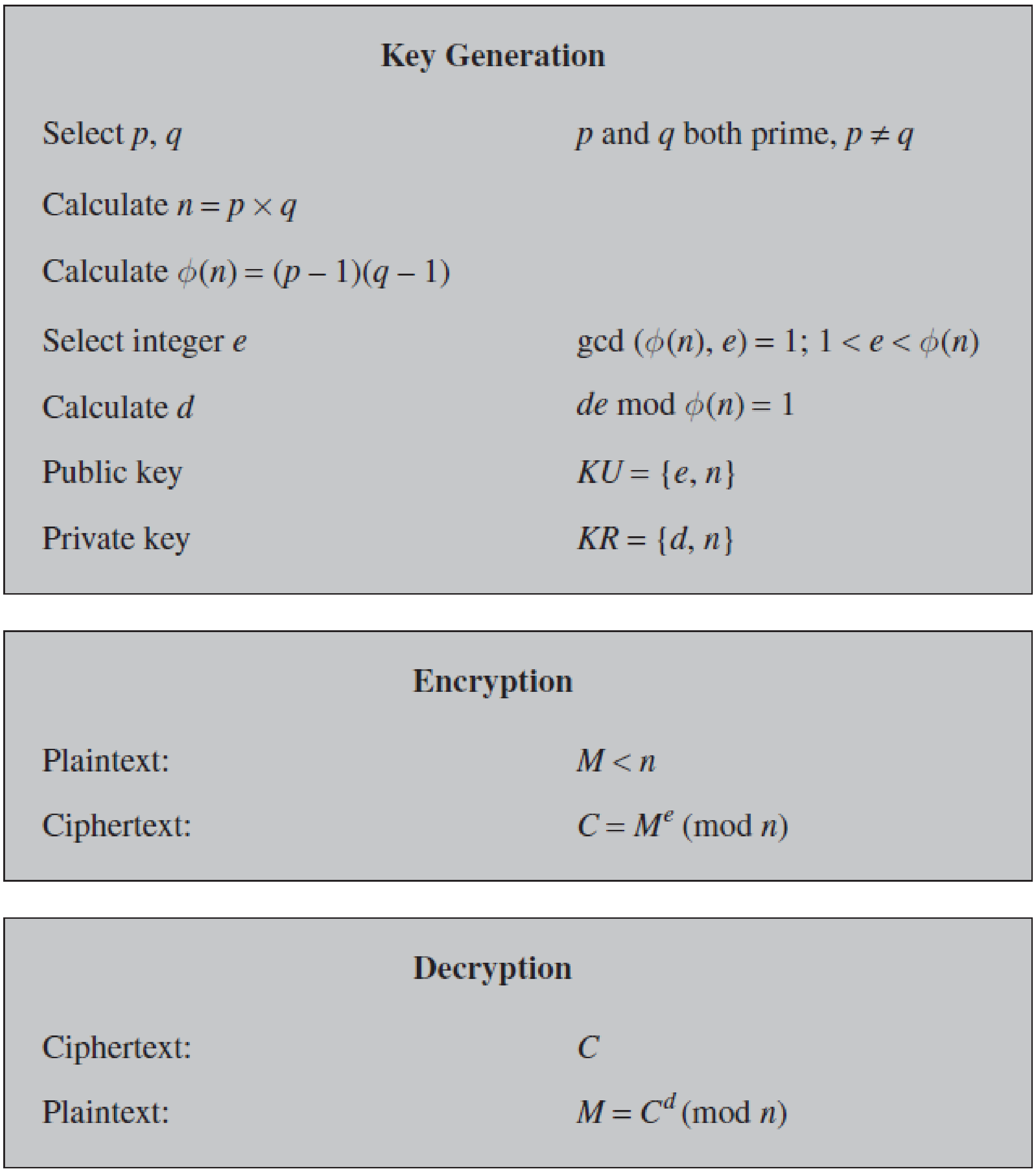
Both sender and receiver must know the values of n and e, and only thereceiver knows the value of d. This is a public-key encryption algorithm with a publickey of KU {e, n} and a private key of KR {d, n}. For this algorithm to be satisfactoryfor public-key encryption, the following requirements must be met.

1. It is possible to find values of e, d, n such that M^{ed} mod n $M$for all $M <n$.
2. It is relatively easy to calculate $M^e$and $C^d$for all values of $M<n$.
3. It is infeasible to determine d given e and n.

Steps of RSA:

1. Begin by selecting two primenumbers p and q and calculating their product n, which is the modulus for encryptionand decryption.
2. Next, we need the quantity f(n), referred to as the Eulertotient of n, which is the number of positive integers less than n and relatively primeto n.
3. Then select an integer e that is relatively prime to f(n) [i.e., the greatest commondivisor of e and f(n) is 1].
4. Finally, calculate d as the multiplicative inverse of e,modulo f(n). It can be shown that d and e have the desired properties.

Suppose that user A has published its public key and that user B wishes tosend the message M to A. Then B calculates $C= M^e$(mod n) and transmits C. Onreceipt of this ciphertext, user A decrypts by calculating $M = C^d$ (mod n).



Key Generation

Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	$de \bmod \phi(n) = 1$
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$

Encryption

Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod{n}$

Decryption

Ciphertext:	C
Plaintext:	$M = C^d \pmod{n}$

Figure 3.10 The RSA Algorithm

For Example:

1. Select two prime numbers, p= 17 and q=11.
2. Calculate n = pq= 17 x 11 = 187.
3. Calculate f(n) (p-1)(q-1)= 16 x 10= 160
4. Select e such that e is relatively prime to f(n)= 160 and less than f(n); we choose e=7.
5. Determine d such that de mod 160 = 1 and d < 160.The correct value is d = 23, because 23 x 7 = 161 = (1+160)

The resulting keys are public key PU {7, 187} and private key PR {23,187}.The example shows the use of these keys for a plaintext input of M = 88.

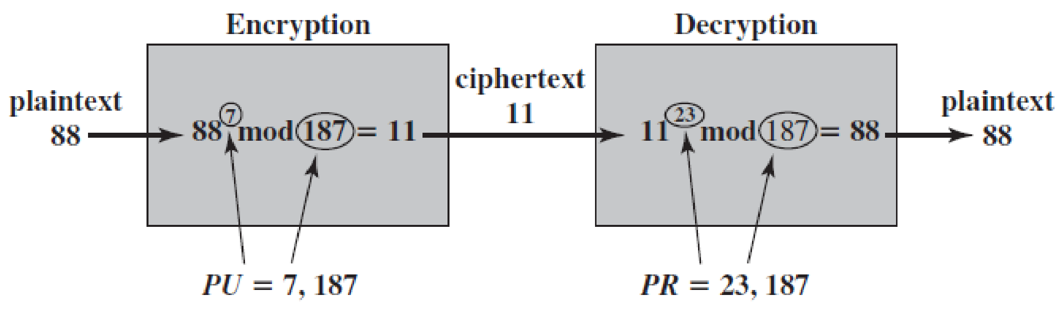


Figure 3.11 Example of RSA Algorithm

There are two possible approaches to defeating the RSA algorithm.**<u>The first isthe brute-force approach</u>**: Try all possible private keys. Thus, the **<u>larger the numberof bits in *e* and *d*</u>**, the more secure the algorithm. However, because the calculationsinvolved (both in key generation and in encryption/decryption) are complex, the<u>larger the size of the key, the slower the system will run.</u>

(ii) Diffie-Hellman Key Exchange

The first published public-key algorithm appeared in the seminal paper by Diffieand Hellman that defined public-key cryptography [DIFF76] and is generallyreferred to as the **Diffie-Hellman key exchange**. A number of commercial productsemploy this key exchange technique.

<u>The purpose of the algorithm is to enable two users to exchange a secret keysecurely</u> that then can be used for subsequent encryption of messages. The algorithmitself is limited to the exchange of the keys.

THE ALGORITHM With this background, we can define the Diffie-Hellman key exchange, which is summarized in Figure 3.12. For this scheme, there are two publicly known numbers: a prime number q and an integer α that is a primitive root of q. Suppose the users A and B wish to exchange a key. User A selects a random integer $X_A < q$ and computes $Y_A = \alpha^{X_A} \bmod q$. Similarly, user B independently selects a random integer $X_B < q$ and computes $Y_B = \alpha^{X_B} \bmod q$. Each side keeps the X value private and makes the Y value available publicly to the other side. User A computes the key as $K = (Y_B)^{X_A} \bmod q$ and user B computes the key as $K = (Y_A)^{X_B} \bmod q$. These two calculations produce identical results:

$$\begin{aligned} K &= (Y_B)^{X_A} \bmod q \\ &= (\alpha^{X_B} \bmod q)^{X_A} \bmod q \\ &= (\alpha^{X_B})^{X_A} \bmod q \\ &= \alpha^{X_B X_A} \bmod q \\ &= (\alpha^{X_A})^{X_B} \bmod q \\ &= (\alpha^{X_A} \bmod q)^{X_B} \bmod q \\ &= (Y_A)^{X_B} \bmod q \end{aligned}$$

The result is that the two sides have exchanged a secret value. Furthermore, because X_A and X_B are private, an adversary only has the following ingredients to work with: q, α, Y_A, and Y_B. Thus, the adversary is forced to take a discrete logarithm to determine the key. For example, to determine the private key of user B, an adversary must compute

$$X_B = \text{dlog}_{\alpha,q}(Y_B)$$

Global Public Elements

q	prime number
α	$\alpha < q$ and α a primitive root of q

User A Key Generation

Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{X_A} \bmod q$

User B Key Generation

Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{X_B} \bmod q$

Generation of Secret Key by User A

$K = (Y_B)^{X_A} \bmod q$

Generation of Secret Key by User B

$K = (Y_A)^{X_B} \bmod q$

Figure 3.12 The Diffie-Hellman Key Exchange Algorithm

Here is an example. Key exchange is based on the use of the prime number $q = 353$ and a primitive root of 353, in this case $\alpha = 3$. A and B select secret keys $X_A = 97$ and $X_B = 233$, respectively. Each computes its public key:

$$\text{A computes } Y_A = 3^{97} \bmod 353 = 40.$$
$$\text{B computes } Y_B = 3^{233} \bmod 353 = 248.$$

After they exchange public keys, each can compute the common secret key:

$$\text{A computes } K = (Y_B)^{X_A} \bmod 353 = 248^{97} \bmod 353 = 160.$$
$$\text{B computes } K = (Y_A)^{X_B} \bmod 353 = 40^{233} \bmod 353 = 160.$$

We assume an attacker would have available the following information:

$$q = 353; \quad \alpha = 3; \quad Y_A = 40; \quad Y_B = 248$$

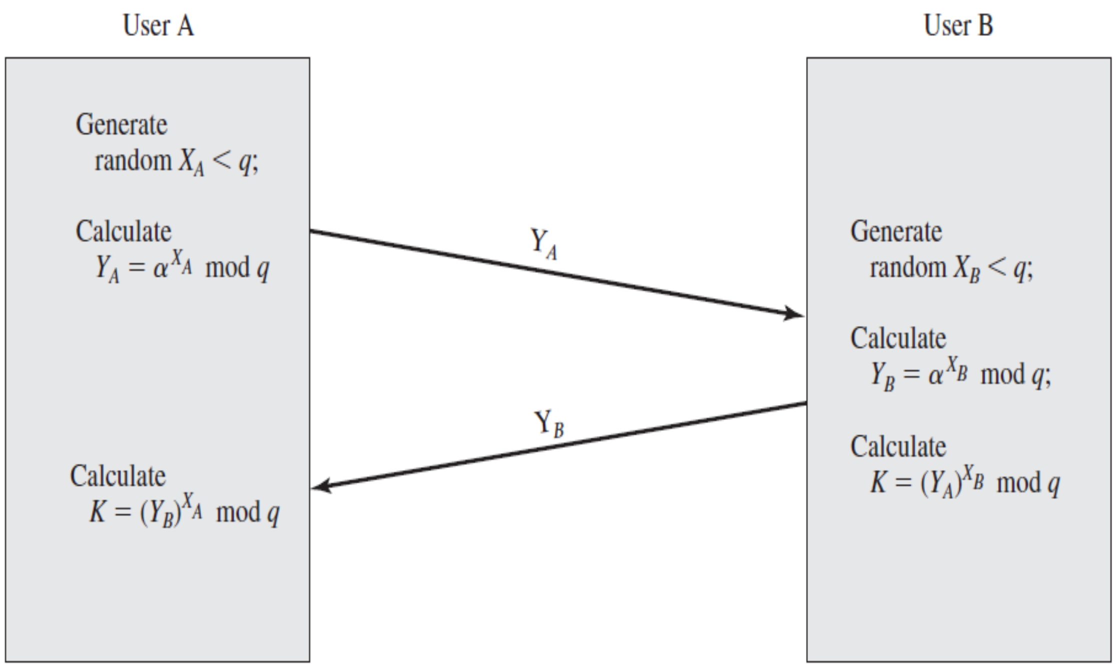

User A
User B
Generate random $X_A < q$;
Calculate $Y_A = \alpha^{X_A} \bmod q$
Y_A
Generate random $X_B < q$;
Calculate $Y_B = \alpha^{X_B} \bmod q$;
Y_B
Calculate $K = (Y_A)^{X_B} \bmod q$
Calculate $K = (Y_B)^{X_A} \bmod q$

Figure 3.13 Diffie-Hellman Key Exchange

MAN IN THE MIDDLEATTACK

The protocol depicted in Figure 3.13 is insecure against a man-in-the-middleattack. Suppose Alice and Bob wish to exchange keys, and Darth is theadversary. The attack proceeds as follows:

- Darth prepares for the attack by generating two random private keys X_{D1} andX_{D2}, and then computing the corresponding public keys YD1 and YD2.
- Alice transmits Y_A to Bob.
- Darth intercepts Y_A and transmits Y_{D1} to Bob. Darth also calculates
- Bob receives Y_{D1} and calculates $K1 = (Y_{D1}) X_B \bmod q$.
- Bob transmits Y_B to Alice.
- Darth intercepts Y_B and transmits Y_{D2} to Alice. Darth calculates
- Alice receives Y_{D2} and calculates $K2 = (Y_{D2}) X_A \bmod q$.
- Bob and Alice think that they share a secret key.
- Instead Bob and Darth share secret key K1, and Alice and Darth share secret key K2.
- All future communication between Bob and Alice is compromised in the following way:

  Alice sends an encrypted message M: E(K2, M).

  Darth intercepts the encrypted message and decrypts it to recover M.

  Darth sends Bob E(K1, M) or E(K1, M'), where M' is any message.
- In the first case, Darth simply wants to eavesdrop on the communication without altering it.
- In the second case, Darth wants to modify the message going to Bob.
- The key exchange protocol is vulnerable to such an attack because it does not authenticate the participants.
- This vulnerability can be overcome with the use of digital signatures and public-key certificates

<u>**DIGITAL SIGNATURES**</u>

Public-key encryption can be used in another way, as illustrated in Figure 3.9b.Suppose that Bob wants to send a message to Alice, and although it is not important that the message be kept secret, he wants Alice to be certain that the message is indeed from him. In this case, Bob uses his own private key to encrypt the message. When Alice receives the ciphertext, she finds that she can decrypt it with Bob’s public key, thus proving that the message must have been encrypted by Bob. No one else has Bob’s private key, and therefore no one else could have created a ciphertext that could be decrypted with Bob’s public key. Therefore, the entire encrypted message serves as a digital signature. In addition, it is impossible to alter the message without access to Bob’s private key, so the message is authenticated both in terms of source and in terms of data integrity.

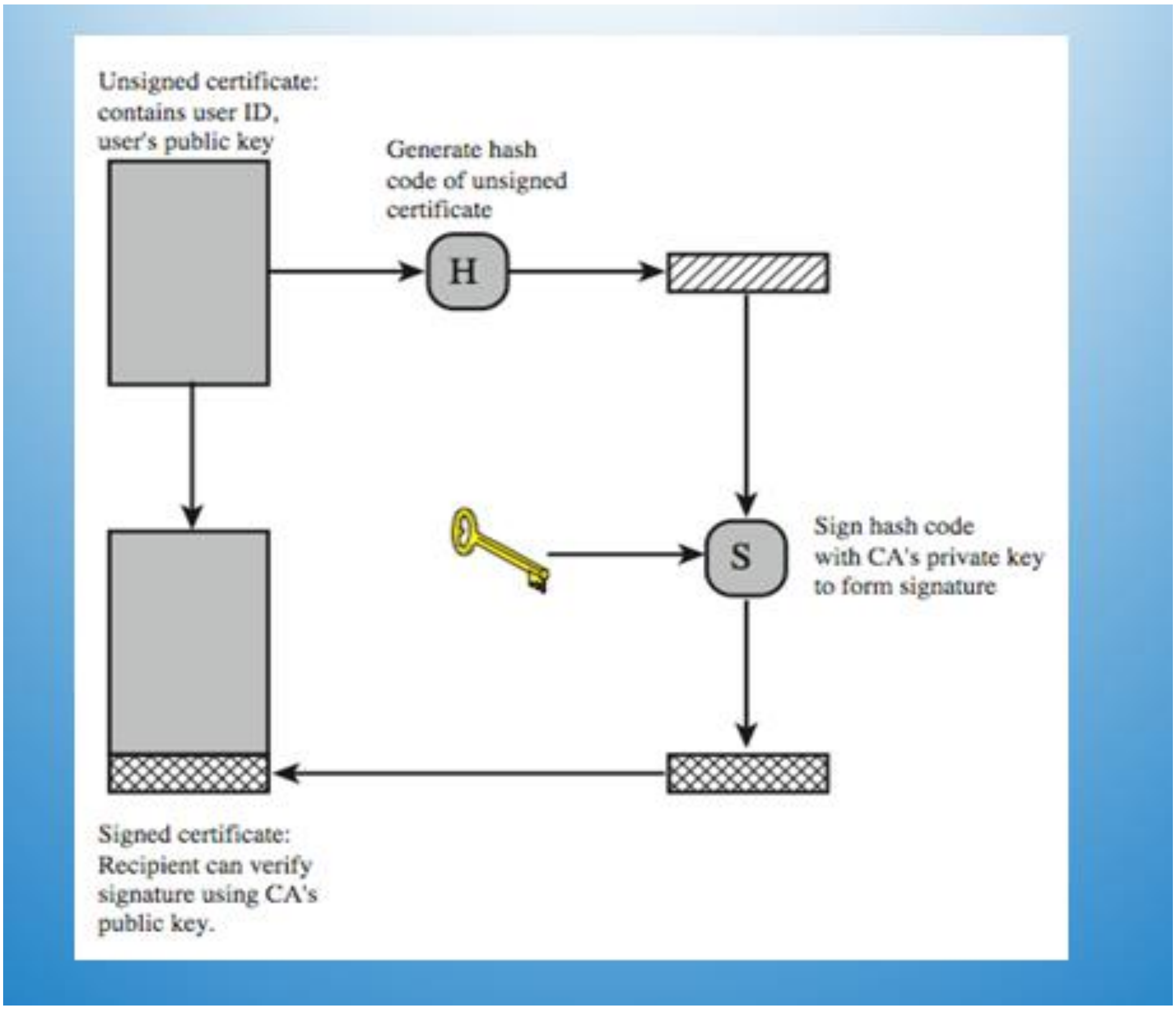
In the preceding scheme, the entire message is encrypted. Although validating both author and contents, this requires a great deal of storage. Each document must be kept in plaintext to be used for practical purposes. A copy also must be stored in ciphertext so that the origin and contents can be verified in case of a dispute. A more efficient way of achieving the same results is to encrypt a small block of bits that is a function of the document. Such a block, called an authenticator, must have the property that it is infeasible to change the document without changing the authenticator. If the authenticator is encrypted with the sender’s private key, it serves as a signature that verifies origin, content, and sequencing. A secure hash code such as SHA-1 can serve this function. Figure 3.2b illustrates this scenario.

It is important to emphasize that the encryption process just described doesnot provide confidentiality. That is, the message being sent is safe from alteration but not safe from eavesdropping. This is obvious in the case of a signature based ona portion of the message, because the rest of the message is transmitted in the clear. Even in the case of complete encryption, there is no protection of confidentiality because any observer can decrypt the message by using the sender’s public key.

<u>**The distribution of Public Keys (Digital Certificates)**</u>

- The weakness in Public key sharing- Any one can forge such a public announcement. Some user could pretend to be user A and send a public key to another participant or broadcast such a public key.
- Until such time as user A discovers the forgery and alerts other participants the forger is able to read all encrypted messages intented for A.
- Solution- PUBLIC KEY CERTIFICATES
- <u>A certificate consists of a Public key plus a user Id of the Key owner with the whole block signed by a trusted third party called Certificate Authority.</u>

- It is trusted by user community such as Government Agency.
- A user present his or her public key to the authority in a secure manner and obtain a Certificate.


Unsigned certificate: contains user ID, user's public key
Generate hash code of unsigned certificate
H
Sign hash code with CA's private key to form signature
S
Signed certificate: Recipient can verify signature using CA's public key.

KERBEROS

Kerberos is a key distribution and user authentication service developed at MIT. The problem that Kerberos addresses

1. A user may gain access to a particular workstation and pretend to be another user operating from that workstation.
2. A user may alter the network address of a workstation so that the requests sent from the altered workstation appear to come from the impersonated workstation.
3. A user may eavesdrop on exchanges and use a replay attack to gain entrance to a server or to disrupt operations.

In any of these cases, an unauthorized user may be able to gain access to services and data that he or she is not authorized to access. Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users. Kerberos relies exclusively on symmetric encryption, making no use of public-key encryption.

Kerberos Version 4

Version 4 of Kerberos makes use of DES.

A SIMPLE AUTHENTICATION DIALOGUEL: To Solve from the unauthorized access to the server. An alternative is to use an authentication server (AS) that knows the passwords of all users and stores these in a centralized database. In addition, the AS shares a unique secret key with each server. These keys have been distributed physically or in some other secure manner. Consider the following hypothetical dialogue:

$$\textbf{(1)}\ \mathrm{C} \rightarrow \mathrm{AS}:\quad ID_C \| P_C \| ID_V$$

$$\textbf{(2)}\ \mathrm{AS} \rightarrow \mathrm{C}:\quad Ticket$$

$$\textbf{(3)}\ \mathrm{C} \rightarrow \mathrm{V}:\quad ID_C \| Ticket$$

$$Ticket = \mathrm{E}(K_v, [ID_C \| AD_C \| ID_V])$$

where

C	= client
AS	= authentication server
V	= server
ID_C	= identifier of user on C
ID_V	= identifier of V
P_C	= password of user on C
AD_C	= network address of C
K_v	= secret encryption key shared by AS and V

In this scenario, the user logs on to a workstation and requests access to server V. The client module C in the user's workstation requests the user's password and then sends a message to the AS that includes the user's ID, the server's ID, and the user's password. The AS checks its database to see if the user has supplied the proper pass-word for this user ID and whether this user is permitted access to server V. If both tests are passed, the AS accepts the user as authentic and must now convince the server that this user is authentic. To do so, the AS creates a ticket that contains the user's ID and network address and the server's ID. This ticket is encrypted using the secret key shared by the AS and this server. This ticket is then sent back to C. Because the ticket is encrypted, it cannot be altered by C or by an opponent.

With this ticket, C can now apply to V for service. C sends a message to V containing C’s ID and the ticket. V decrypts the ticket and verifies that the user ID in the ticket is the same as the unencrypted user ID in the message. If these two match ,the server considers the user authenticated and grants the requested service.

A MORE SECURE AUTHENTICATION DIALOGUE:

Although the foregoing scenario solves some of the problems of authentication in an open network environment, problems remain. Two in particular stand out.

First, we would like to minimize the number of times that a user has to enter a password. Suppose each ticket can be used only once. If user C logs on to a workstation in the morning and wishes to check his or her mail at a mail server, C must supply a password to get a ticket for the mail server. If C wishes to check the mail several times during the day, each attempt requires entering the password

However, under this scheme, it remains the case that a user would need a new ticket for every different service. If a user wished to access a print server, a mail server, a file server, and so on

The second problem is that the earlier scenario involved a plaintext transmission of the password.

To solve these additional problems, we introduce a scheme for avoiding plain-text passwords and a new server, known as the ticket-granting server (TGS).

Once per user logon session:

$$\textbf{(1)}\ \mathrm{C} \rightarrow \mathrm{AS}: \quad ID_C \| ID_{tgs}$$

$$\textbf{(2)}\ \mathrm{AS} \rightarrow \mathrm{C}: \quad \mathrm{E}(K_c, Ticket_{tgs})$$

Once per type of service:

$$\textbf{(3)}\ \mathrm{C} \rightarrow \mathrm{TGS}: \quad ID_C \| ID_V \| Ticket_{tgs}$$

$$\textbf{(4)}\ \mathrm{TGS} \rightarrow \mathrm{C}: \quad Ticket_v$$

Once per service session:

$$\textbf{(5)}\ \mathrm{C} \rightarrow \mathrm{V}: \quad ID_C \| Ticket_v$$

$$Ticket_{tgs} = \mathrm{E}(K_{tgs}, [ID_C \| AD_C \| ID_{tgs} \| TS_1 \| Lifetime_1])$$

$$Ticket_v = \mathrm{E}(K_v, [ID_C \| AD_C \| ID_v \| TS_2 \| Lifetime_2])$$

The new service, TGS, issues tickets to users who have been authenticated to AS. Thus, the user first requests a ticket-granting ticket ($Ticket_{tgs}$) from the AS. The client module in the user workstation saves this ticket. Each time the user requires access to a new service, the client applies to the TGS, using the ticket to authenticate itself. The TGS then grants a ticket for the particular service. The client saves each service-granting ticket and uses it to authenticate its user to a server each time a particular service is requested.

1. The client requests a ticket-granting ticket on behalf of the user by sending its user's ID to the AS, together with the TGS ID, indicating a request to use the TGS service.
2. The AS responds with a ticket that is encrypted with a key that is derived from the user's password (Kc), which is already stored at the AS. When this response arrives at the client, the client prompts the user for his or her pass-word, generates the key, and attempts to decrypt the incoming message. If the correct password is supplied, the ticket is successfully recovered.

Because only the correct user should know the password, only the correct user can recover the ticket. Thus, we have used the password to obtain credentials from Kerberos without having to transmit the password in plaintext. The ticket itself consists of the ID and network address of the user and the ID of the TGS.

3. The client requests a service-granting ticket on behalf of the user. For this purpose, the client transmits a message to the TGS containing the user's ID, the ID of the desired service, and the ticket-granting ticket.

4. The TGS decrypts the incoming ticket using a key shared only by the AS and the TGS (Ktgs) and verifies the success of the decryption by the presence of its ID. It checks to make sure that the lifetime has not expired. Then it compares the user ID and network address with the incoming information to authenticate the user. If the user is permitted access to the server V, the TGS issues a ticket to grant access to the requested service.

5. Finally, with a particular service-granting ticket, the client can gain access to the corresponding service with step 5. The client requests access to a service on behalf of the user. For this purpose, the client transmits a message to the server containing the user's ID and the service granting ticket. The server authenticates by using the contents of the ticket.

THE VERSION 4 AUTHENTICATION DIALOGUE:
Although the foregoing scenario enhances security compared to the first attempt, two additional problems remain. The heart of the first problem is the lifetime associated with the ticket-granting ticket. If this lifetime is very short (e.g., minutes), then the user will be repeatedly asked for a password. If the lifetime is long (e.g., hours), then an opponent has a greater opportunity for replay.

Thus, we arrive at an additional requirement. A network service (the TGS or an application service) must be able to prove that the person using a ticket is the same person to whom that ticket was issued.

First, consider the problem of captured ticket-granting tickets and the need to determine that the ticket presenter is the same as the client for whom the ticket was issued. The threat is that an opponent will steal the ticket and use it before it expires.

Table 4.1 Summary of Kerberos Version 4 Message Exchanges

(1) C → AS $ID_c \| ID_{tgs} \| TS_1$

(2) AS → C $E(K_c, [K_{c,tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs}])$

$$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2])$$

(a) Authentication Service Exchange to obtain ticket-granting ticket

(3) C → TGS $ID_v \| Ticket_{tgs} \| Authenticator_c$

(4) TGS → C $E(K_{c,tgs}, [K_{c,v} \| ID_v \| TS_4 \| Ticket_v])$

$$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2])$$

$$Ticket_v = E(K_v, [K_{c,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| Lifetime_4])$$

$$Authenticator_c = E(K_{c,tgs}, [ID_C \| AD_C \| TS_3])$$

(b) Ticket-Granting Service Exchange to obtain service-granting ticket

(5) C → V $Ticket_v \| Authenticator_c$

(6) V → C $E(K_{c,v}, [TS_5 + 1])$ (for mutual authentication)

$$Ticket_v = E(K_v, [K_{c,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| Lifetime_4])$$

$$Authenticator_c = E(K_{c,v}, [ID_C \| AD_C \| TS_5])$$

(c) Client/Server Authentication Exchange to obtain service

To get around this problem, let us have the AS provide both the client and the TGS with a secret piece of information in a secure manner. Then the client can prove its identity to the TGS by revealing the secret information, again in a secure manner. An efficient way of accomplishing this is to use an encryption key as the secure information; this is referred to as a session key in Kerberos.

As before, the client sends a message to the AS requesting access to the TGS. The AS responds with a message, encrypted with a key derived from the user's password (K_C), that contains the ticket. The encrypted message also contains a copy of the session key, $K_{C,tgs}$, where the subscripts indicate that this is a session key for C and TGS. Because this session key is inside the message encrypted with K_C, only the user's client can read it. The same session key is included in the ticket, which can be read only by the TGS. Thus, the session key has been securely delivered to both C and the TGS.

Note that several additional pieces of information have been added to this first phase of the dialogue. Message (1) includes a timestamp, so that the AS knows that the message is timely. Message (2) includes several elements of the ticket in a form accessible to C. This enables C to confirm that this ticket is for the TGS and to learn its expiration time.

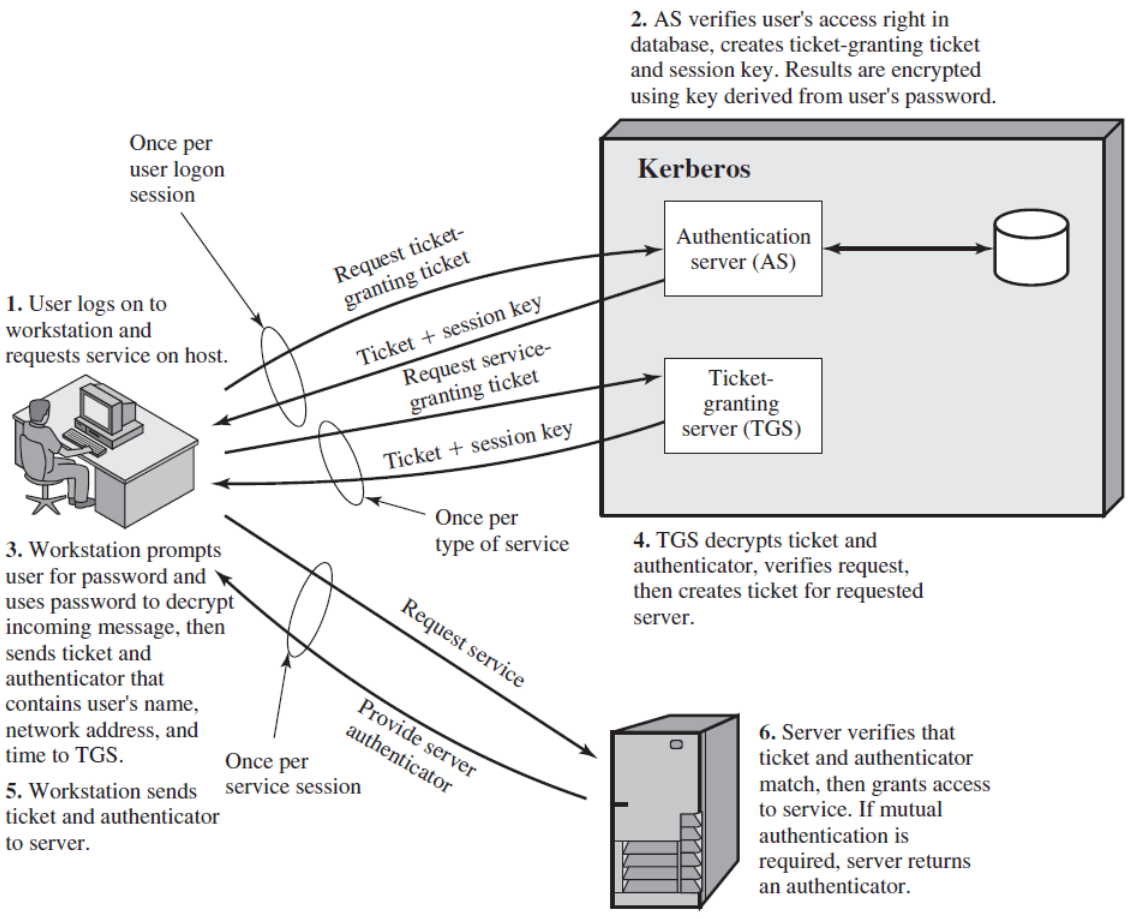
Armed with the ticket and the session key, C is ready to approach the TGS. As before, C sends the TGS a message that includes the ticket plus the ID of the requested service. In addition, C transmits an authenticator, which includes the ID and address of C's user and a timestamp. Unlike the ticket, which is reusable, the authenticator is intended for use only once and has a very short lifetime. The TGS can decrypt the ticket with the key that it shares with the AS. This ticket indicates that user C has been provided with the session key $K_{C,tgs}$. In effect, the ticket says, "Anyone who uses $K_{C,tgs}$ must be C." The TGS uses the session key to decrypt the authenticator. The TGS can then check the name and address from the authenticator with that of the ticket and with the network address of the incoming message. If all match, then the TGS is assured that the sender of the ticket is indeed the ticket's real owner. In effect, the authenticator says, "At time TS_3, I hereby use $K_{C,tgs}$." Note that the ticket does not prove anyone's identity but is a way to distribute keys securely. It is the authenticator that proves the client's identity. Because the authenticator can be used only once and has a short lifetime, the threat of an opponent stealing both the ticket and the authenticator for presentation later is countered.

The reply from the TGS in message (4) follows the form of message (2). The message is encrypted with the session key shared by the TGS and C and includes a session key to be shared between C and the server V, the ID of V, and the timestamp of the ticket.The ticket itself includes the same session key.

C now has a reusable service-granting ticket for V.When C presents this ticket, as shown in message (5), it also sends an authenticator. The server can decrypt the ticket, recover the session key, and decrypt the authenticator.

If mutual authentication is required, the server can reply.The server returns the value of the timestamp from the authenticator, incremented by 1, and encrypted in the session key. C can decrypt this message to recover the incremented timestamp. Because the message was encrypted by the session key, C is assured that it could have been created only by V. The contents of the message assure C that this is not a replay of an old reply.

Finally, at the conclusion of this process, the client and server share a secret key. This key can be used to encrypt future messages between the two or to exchange a new random session key for that purpose



2. AS verifies user's access right in database, creates ticket-granting ticket and session key. Results are encrypted using key derived from user's password.
Once per user logon session
Kerberos
Authentication server (AS)
Request ticket-granting ticket
Ticket + session key
1. User logs on to workstation and requests service on host.
Request service-granting ticket
Ticket-granting server (TGS)
Ticket + session key
Once per type of service
4. TGS decrypts ticket and authenticator, verifies request, then creates ticket for requested server.
3. Workstation prompts user for password and uses password to decrypt incoming message, then sends ticket and authenticator that contains user's name, network address, and time to TGS.
Request service
Provide server authenticator
Once per service session
5. Workstation sends ticket and authenticator to server.
6. Server verifies that ticket and authenticator match, then grants access to service. If mutual authentication is required, server returns an authenticator.

Figure 4.1 Overview of Kerberos

KERBEROS REALMS AND MULTIPLE KERBERI A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers requires the following:

1. The Kerberos server must have the user ID and hashed passwords of all participating users in its database. All users are registered with the Kerberos server.
2. The Kerberos server must share a secret key with each server. All servers are registered with the Kerberos server.

Such an environment is referred to as a **Kerberos realm**. The concept of **realm** can be explained as follows. A Kerberos realm is a set of managed nodes that share the same Kerberos database. The Kerberos database resides on the Kerberos master computer system, which should be kept in a physically secure room. A read-only copy of the Kerberos database might also reside on other Kerberos computer systems. However, all changes to the database must be made on the master computer system. Changing or accessing the contents of a Kerberos database requires the Kerberos master password. A related concept is that of a **Kerberos principal**, which is a service or user that is known to the Kerberos system. Each Kerberos principal is identified by its principal name. Principal names consist of three parts: a service or user name, an instance name, and a realm name

Networks of clients and servers under different administrative organizations typically constitute different realms. That is, it generally is not practical or does not conform to administrative policy to have users and servers in one administrative domain registered with a Kerberos server elsewhere. However, users in one realm may need access to servers in other realms, and some servers may be willing to provide service to users from other realms, provided that those users are authenticated.

Kerberos provides a mechanism for supporting such interrealm authentication. For two realms to support interrealm authentication, a third requirement is added

The Kerberos server in each interoperating realm shares a secret key with the server in the other realm. The two Kerberos servers are registered with each other.

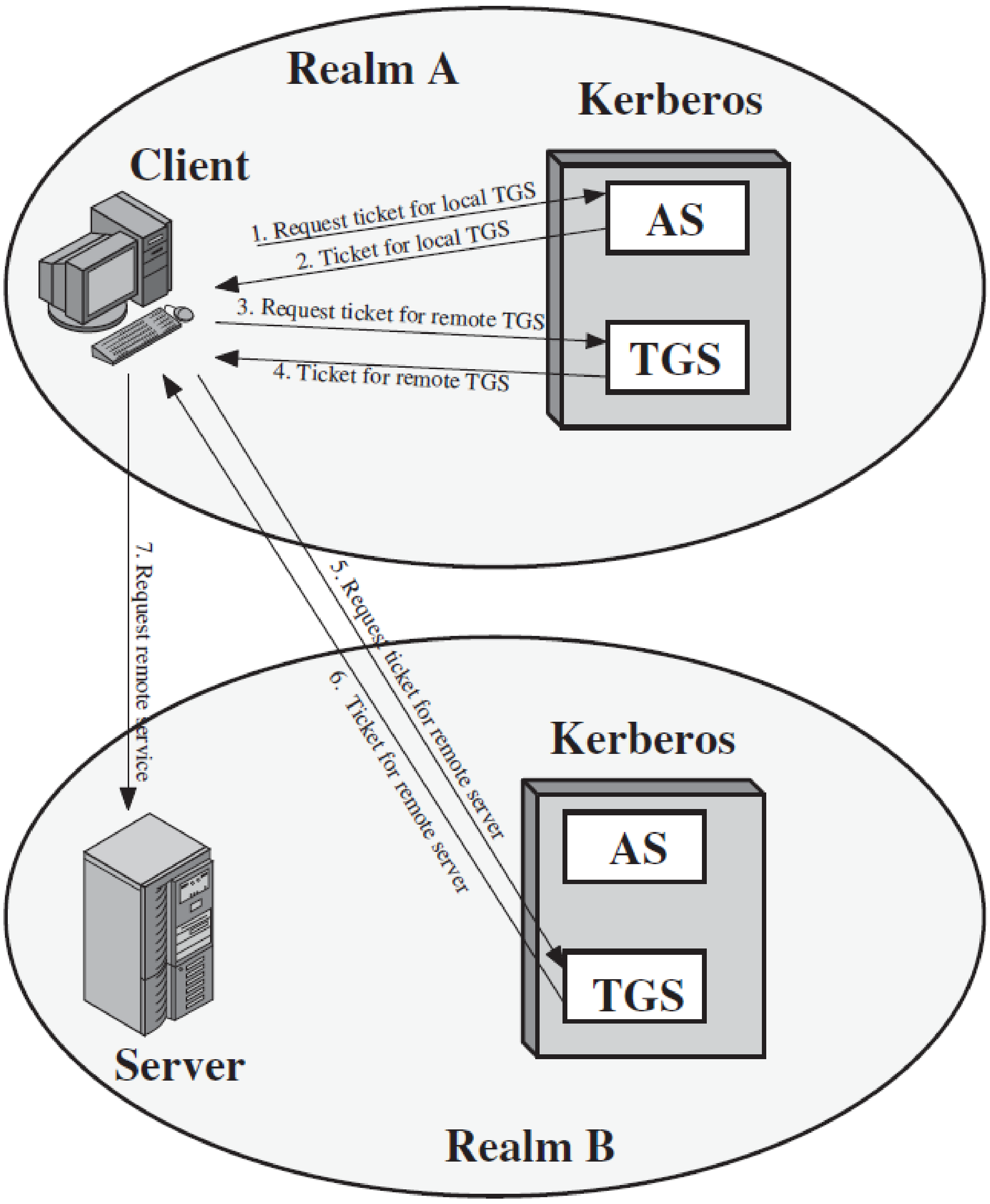

Realm A
Kerberos
Client
AS
TGS
1. Request ticket for local TGS
2. Ticket for local TGS
3. Request ticket for remote TGS
4. Ticket for remote TGS
7. Request remote service
5. Request ticket for remote server
6. Ticket for remote server
Kerberos
AS
TGS
Server
Realm B

Figure 4.2 Request for Service in Another Realm

The details of the exchanges illustrated in Figure 4.2 are as follows (compare Table 4.1).

$$
\begin{array}{ll}
\textbf{(1)}\ \mathrm{C} \rightarrow \mathrm{AS}: & ID_C \,\|\, ID_{tgs} \,\|\, TS_1 \\
\textbf{(2)}\ \mathrm{AS} \rightarrow \mathrm{C}: & \mathrm{E}(K_C, [K_{C,tgs} \,\|\, ID_{tgs} \,\|\, TS_2 \,\|\, Lifetime_2 \,\|\, Ticket_{tgs}]) \\
\textbf{(3)}\ \mathrm{C} \rightarrow \mathrm{TGS}: & ID_{tgsrem} \,\|\, Ticket_{tgs} \,\|\, Authenticator_C \\
\textbf{(4)}\ \mathrm{TGS} \rightarrow \mathrm{C}: & \mathrm{E}(K_{C,tgs}, [K_{C,tgsrem} \,\|\, ID_{tgsrem} \,\|\, TS_4 \,\|\, Ticket_{tgsrem}]) \\
\textbf{(5)}\ \mathrm{C} \rightarrow \mathrm{TGS_{rem}}: & ID_{Vrem} \,\|\, Ticket_{tgsrem} \,\|\, Authenticator_C \\
\textbf{(6)}\ \mathrm{TGS_{rem}} \rightarrow \mathrm{C}: & \mathrm{E}(K_{C,tgsrem}, [K_{C,Vrem} \,\|\, ID_{Vrem} \,\|\, TS_6 \,\|\, Ticket_{Vrem}]) \\
\textbf{(7)}\ \mathrm{C} \rightarrow \mathrm{V_{rem}}: & Ticket_{Vrem} \,\|\, Authenticator_C
\end{array}
$$

A user wishing service on a server in another realm needs a ticket for that server.The user's client follows the usual procedures to gain access to the local TGS and then requests a ticket-granting ticket for a remote TGS (TGS in another realm).The client can then apply to the remote TGS for a service-granting ticket for the desired server in the realm of the remote TGS.

The ticket presented to the remote server (*Vrem*) indicates the realm in which the user was originally authenticated. The server chooses whether to honor the remote request.

One problem presented by the foregoing approach is that it does not scale well to many realms. If there are *N* realms, then there must be *N*(*N* - 1)/2 secure key exchanges so that each Kerberos realm can interoperate with all other Kerberos realms.

Kerberos Version 5

Kerberos version 5 is specified in RFC 4120 and provides a number of improvements over version 4

DIFFERENCES BETWEEN VERSIONS 4 AND 5 Version 5 is intended to address the limitations of version 4 in two areas: environmental shortcomings and technical deficiencies.

1. **Encryption system dependence:** Version 4 requires the use of DES. Export restriction on DES as well as doubts about the strength of DES were thus of concern. In version 5, ciphertext is tagged with an encryption-type identifier so that any encryption technique may be used
2. **Internet protocol dependence:** Version 4 requires the use of Internet Protocol (IP) addresses. Other address types, such as the ISO network address, are not accommodated. Version 5 network addresses are tagged with type and length, allowing any network address type to be used.
3. **Message byte ordering:** In version 4, the sender of a message employs a byte ordering of its own choosing and tags the message to indicate least significant byte in lowest address or most significant byte in lowest address. This techniques works but does not follow established conventions. In version 5, all message structures are defined using Abstract Syntax Notation One (ASN.1) and Basic Encoding Rules (BER), which provide an unambiguous byte.
4. **Ticket lifetime:** Lifetime values in version 4 are encoded in an 8-bit quantity in units of five minutes. Thus, the maximum lifetime that can be expressed is $2^8 \times 5 = 1280$ minutes (a little over 21 hours).This may be inadequate for some applications (e.g., a long-running simulation that requires valid Kerberos credentials throughout execution). In version 5, tickets include an explicit start time and end time, allowing tickets with arbitrary lifetimes.
5. **Authentication forwarding:** Version 4 does not allow credentials issued to one client to be forwarded to some other host and used by some other client. This capability would enable a client to access a server and have that server access another server on behalf of the client. For example, a client issues a request to a print server that then accesses the client's file from a file server, using the client's credentials for access. Version 5 provides this capability.
6. **Interrealm authentication:** In version 4, interoperability among *N* realms requires on the order of N^2 Kerberos-to-Kerberos relationships, as described earlier. Version 5 supports a method that requires fewer relationships, as described shortly.

Apart from these environmental limitations, there are **technical deficiencies** in the version 4 protocol itself.

1. **Double encryption: T**ickets provided to clients are encrypted twice—once with the secret key of the target server and then again with a secret key known to the client. The second encryption is not necessary and is computationally wasteful.
2. **PCBC encryption:** Encryption in version 4 makes use of a nonstandard mode of DES known as **propagating cipher block chaining (PCBC)**. It has been demonstrated that this mode is vulnerable to an attack involving the interchange of ciphertext blocks [KOHL89]. PCBC was intended to provide an integrity check as part of the encryption operation. Version 5 provides explicit integrity mechanisms, allowing the standard CBC mode to be used for encryption
3. **Session keys:** Each ticket includes a session key that is used by the client to encrypt the authenticator sent to the service associated with that ticket. In addition, the session key subsequently may be used by the client and the server to protect messages passed during that session. However, because the same ticket may be used repeatedly to gain service from a particular server, there is the risk that an opponent will replay messages from an old session to the client or the server. In version 5, it is possible for a client and server to negotiate a sub-session key, which is to be used only for that one connection.
4. **Password attacks:** Both versions are vulnerable to a password attack. The message from the AS to the client includes material encrypted with a key based on the client's password.3 An opponent can capture this message and attempt to decrypt it by trying various passwords.

THE VERSION 5 AUTHENTICATION DIALOGUE : First, consider the **authentication service exchange**. Message (1) is a client request for a ticket-granting ticket. As before, it includes the ID of the user and the TGS. The following new elements are added:

- **Realm:** Indicates realm of user.
- **Options:** Used to request that certain flags be set in the returned ticket.
- **Times:** Used by the client to request the following time settings in the ticket:
  - **from**: the desired start time for the requested ticket
  - **till**: the requested expiration time for the requested ticket
  - **rtime**: requested renew-till time
- **Nonce:** A random value to be repeated in message (2) to assure that the response is fresh and has not been replayed by an opponent.

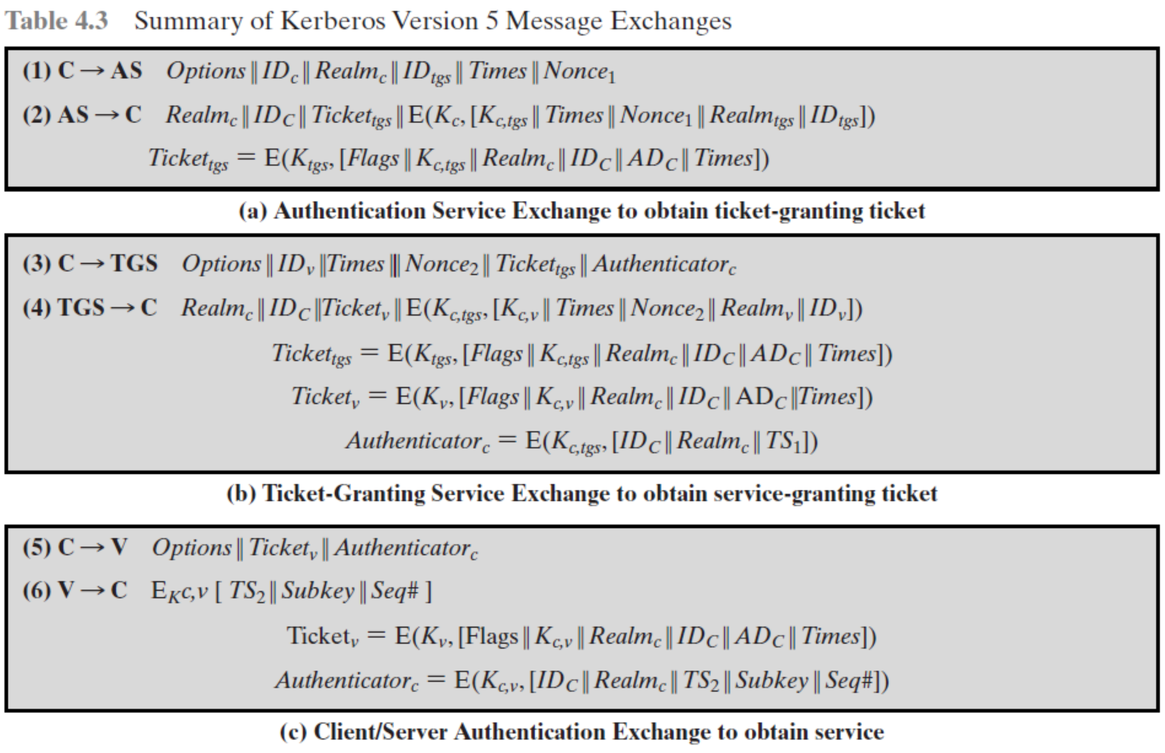


Table 4.3 Summary of Kerberos Version 5 Message Exchanges

(1) C → AS $Options \| ID_c \| Realm_c \| ID_{tgs} \| Times \| Nonce_1$

(2) AS → C $Realm_c \| ID_C \| Ticket_{tgs} \| E(K_c, [K_{c,tgs} \| Times \| Nonce_1 \| Realm_{tgs} \| ID_{tgs}])$

$$Ticket_{tgs} = E(K_{tgs}, [Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times])$$

(a) Authentication Service Exchange to obtain ticket-granting ticket

(3) C → TGS $Options \| ID_v \| Times \| Nonce_2 \| Ticket_{tgs} \| Authenticator_c$

(4) TGS → C $Realm_c \| ID_C \| Ticket_v \| E(K_{c,tgs}, [K_{c,v} \| Times \| Nonce_2 \| Realm_v \| ID_v])$

$$Ticket_{tgs} = E(K_{tgs}, [Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times])$$

$$Ticket_v = E(K_v, [Flags \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times])$$

$$Authenticator_c = E(K_{c,tgs}, [ID_C \| Realm_c \| TS_1])$$

(b) Ticket-Granting Service Exchange to obtain service-granting ticket

(5) C → V $Options \| Ticket_v \| Authenticator_c$

(6) V → C $E_{Kc,v} [TS_2 \| Subkey \| Seq\#]$

$$Ticket_v = E(K_v, [Flags \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times])$$

$$Authenticator_c = E(K_{c,v}, [ID_C \| Realm_c \| TS_2 \| Subkey \| Seq\#])$$

(c) Client/Server Authentication Exchange to obtain service

Message (2) returns a ticket-granting ticket, identifying information for the client, and a block encrypted using the encryption key based on the user's password. This block includes the session key to be used between the client and the TGS, times specified in message (1), the nonce from message (1), and TGS identifying information.

The ticket itself includes the session key, identifying information for the client, the requested time values, and flags that reflect the status of this ticket and the requested options. These flags introduce significant new functionality to version 5. For now, we defer a discussion of these flags and concentrate on the overall structure of the version 5 protocol.

Let us now compare the **ticket-granting service exchange** for versions 4 and 5. We see that message (3) for both versions includes an authenticator, a ticket, and the name of the requested service. In addition, version 5 includes requested times and options for the ticket and a nonce—all with functions similar to those of message (1). The authenticator itself is essentially the same as the one used in version 4. Message (4) has the same structure as message (2). It returns a ticket plus information needed by the client, with the information encrypted using the session key now shared by the client and the TGS.

Finally, for the **client/server authentication exchange**, several new features appear in version 5. In message (5), the client may request as an option that mutual authentication is required. The authenticator includes several new fields:

- **Subkey:** The client's choice for an encryption key to be used to protect this specific application session. If this field is omitted, the session key from the ticket ($K_{C,V}$) is used.
- **Sequence number:** An optional field that specifies the starting sequence number to be used by the server for messages sent to the client during this session. Messages may be sequence numbered to detect replays.

X.509

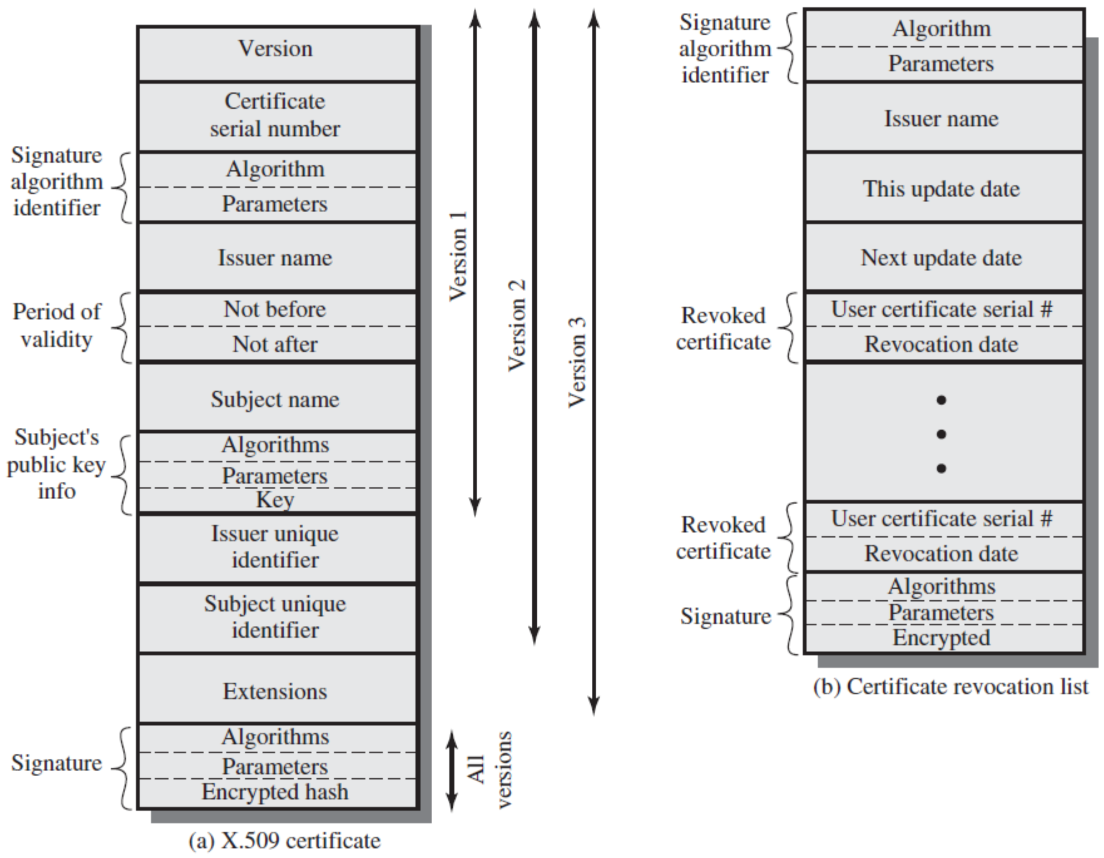


Figure 4.4 X.509 Formats

certification function; it merely provides an easily accessible location for users to obtain certificates.

Figure 4.4a shows the general format of a certificate, which includes the following elements.

- **Version:** Differentiates among successive versions of the certificate format; the default is version 1. If the Issuer Unique Identifier or Subject Unique Identifier are present, the value must be version 2. If one or more extensions are present, the version must be version 3.
- **Serial number:** An integer value, unique within the issuing CA, that is unambiguously associated with this certificate.
- **Signature algorithm identifier:** The algorithm used to sign the certificate, together with any associated parameters. Because this information is repeated in the Signature field at the end of the certificate, this field has little, if any, utility.
- **Issuer name:** X.500 name of the CA that created and signed this certificate.
- **Period of validity:** Consists of two dates: the first and last on which the certificate is valid.
- **Subject name:** The name of the user to whom this certificate refers. That is, this certificate certifies the public key of the subject who holds the corresponding private key.

- **Subject's public-key information:** The public key of the subject, plus an identifier of the algorithm for which this key is to be used, together with any associated parameters.
- **Issuer unique identifier:** An optional bit string field used to identify uniquely the issuing CA in the event the X.500 name has been reused for different entities.
- **Subject unique identifier:** An optional bit string field used to identify uniquely the subject in the event the X.500 name has been reused for different entities.
- **Extensions:** A set of one or more extension fields. Extensions were added in version 3 and are discussed later in this section.
- **Signature:** Covers all of the other fields of the certificate; it contains the hash code of the other fields encrypted with the CA's private key. This field includes the signature algorithm identifier.