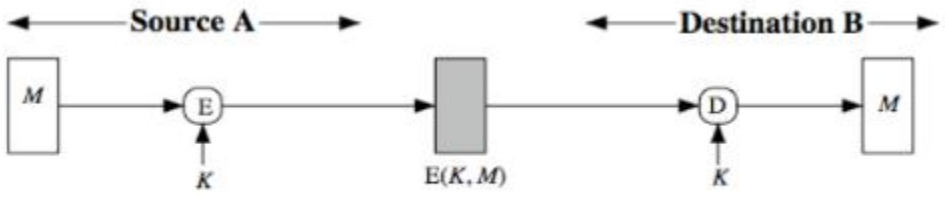
Message Authentication:

Message authentication is a procedure to verify that received messages come from the authorized source and have not been altered. It is intended against the attacks like content modification, sequence modification, timing modification and repudiation There are three classes functions that may be used to produce an authenticator. They are:

- Message encryption–the cipher text serves as authenticator
- Message authentication code (MAC)–a public function of the message and a secret key producing a fixed-length value to serve as authenticator. This does not provide a digital signature because A and B share the same key.
- Hash function–a public function mapping an arbitrary length message into a fixed length hash value to serve as authenticator. This does not provide a digital signature because there is no key.

Message Encryption:

If use symmetric encryption, No other party knows the key, then confidentiality is provided. Here in the symmetric encryption the ciphertext of the entire message serves as its authenticator, see the following diagram,…. it shows that the cipher text acts as an authenticator



(a) Symmetric encryption: confidentiality and authentication

If use public-key techniques (Asymmetric techniques) , can use a digital signature which can only have been created by key owner to validate the integrity of the message contents. To provide both confidentiality and authentication,

A can encrypt M first using its private key, which provides the digital signature, and then using B's public key, which provides confidentiality (Stallings Figure 12.1d).

(d) Public-key encryption: confidentiality, authentication, and signature

(b)Message Authentication Code

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

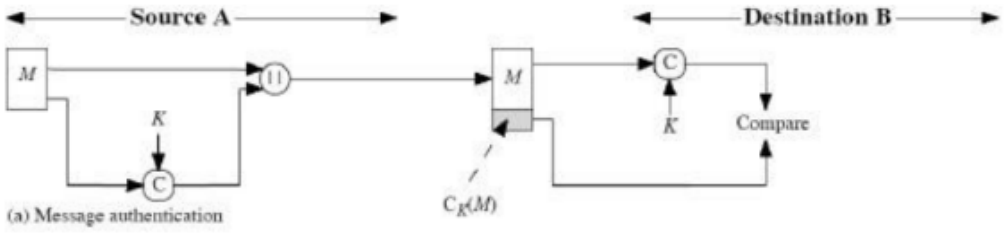
When A has a message to send to B, it calculates MAC as a function C of key and message given as: $MAC=C_k(M)$

The message and the MAC are transmitted to the intended recipient, Here the recipient uses same shared key and generate MAC value and compare the received value from sender.If they both are same The sender is authenticated .The receiver has to check the following conditions also:

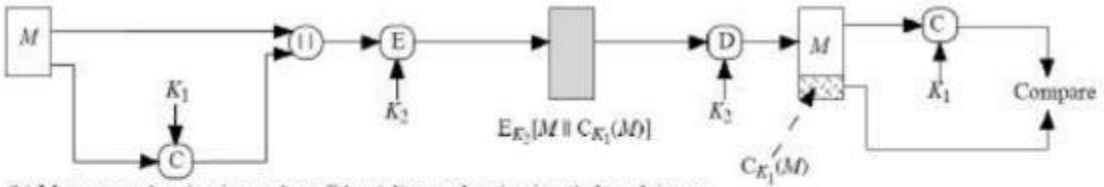
1. The receiver is assured that the message has not been altered: Any alternations been done the MAC's do not match.

2. The receiver is assured that the message is from the alleged sender: No one except the sender has the secret key and could prepare a message with a proper MAC.
3. If the message includes a sequence number, then receiver is assured of proper sequence as an attacker cannot successfully alter the sequence number.
4.

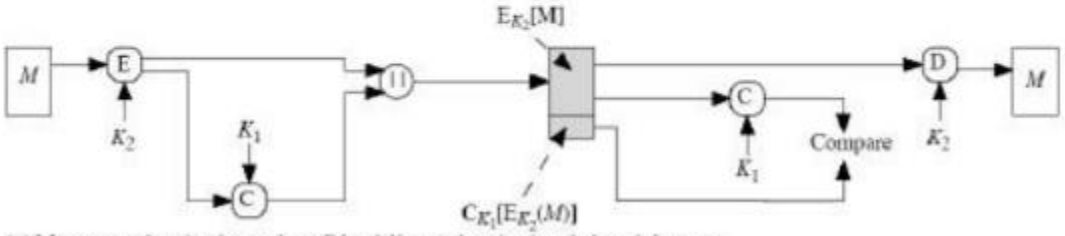
Basic uses of Message Authentication Code (MAC) are shown in the figure:



(a) Message authentication



(b) Message authentication and confidentiality; authentication tied to plaintext



(c) Message authentication and confidentiality; authentication tied to ciphertext

There are three different situations where use of a MAC is desirable:

- If a message is broadcast to several destinations in a network (such as a military control center), then it is cheaper and more reliable to have just one node responsible to evaluate the authenticity – message will be sent in plain with an attached authenticator.
- If one side has a heavy load, it cannot afford to decrypt all messages –it will just check the authenticity of some randomly selected messages.
- Authentication of computer programs in plaintext is very attractive service as they need not be decrypted every time wasting of processor resources. Integrity of the program can always be checked by MAC.

Hash Function:

A hash function H accepts a variable-length block of data M as input and produces a fixed-size hash value h = H(M). The kind of hash function needed for security applications is referred to as a cryptographic hash function. A cryptographic hash function is an algorithm for which it is computationally infeasible (because no attack is significantly more efficient than brute force) to find either (a) a data object that maps to a pre-specified hash result (the one-way property) Because of these characteristics, hash functions are often used to determine whether data has changed or not.

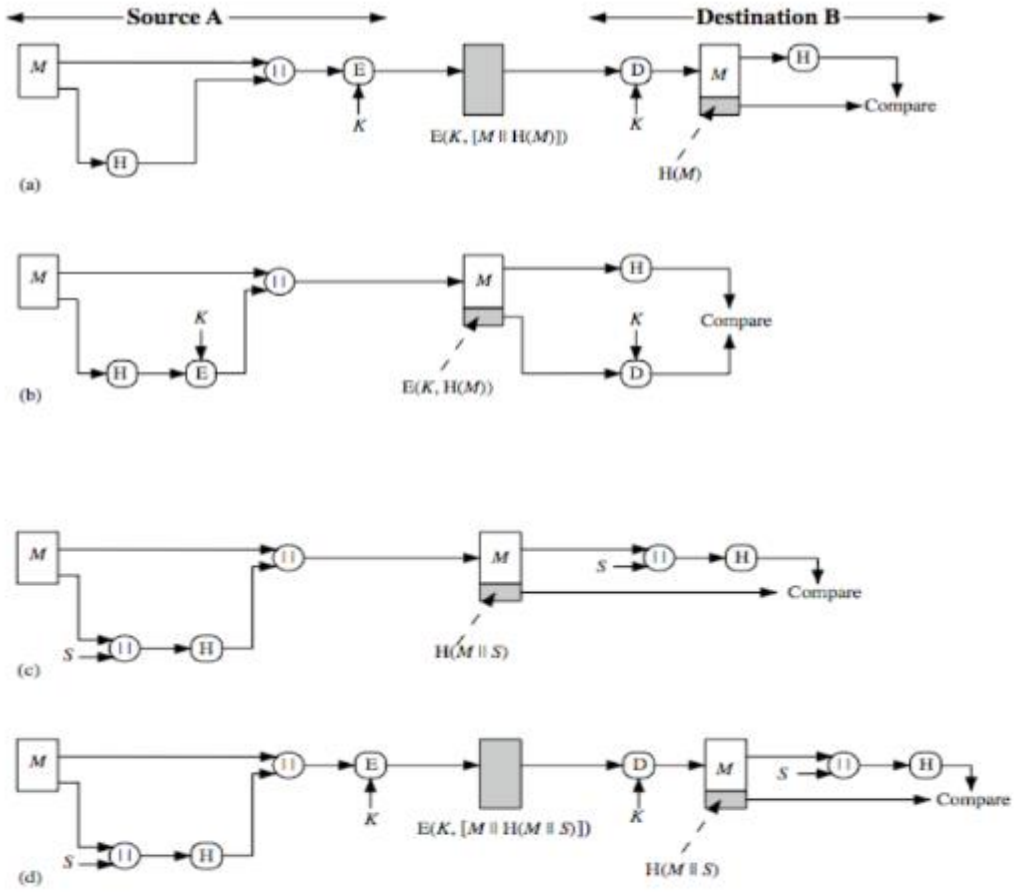


Figure 11.2 illustrates a variety of ways in which a hash code can be used to provide message authentication, as follows:

a. The message plus concatenated hash code is encrypted using symmetric encryption. Since 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.

b. Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications not requiring confidentiality.

c. Shows the use of 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 S, it can recompute 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 (c) by encrypting the entire message plus the hash code.

SHA-512 Logic

The algorithm takes as input a message with a maximum length of $< 2^{128}$ bits and produces as output a 512-bit message digest. The input is processed in 1024-bit blocks .This follows the general structure depicted in Figure 11.9. The processing consists of the following steps.

Step 1 Append padding bits. The message is padded so that its length is congruent to 896 modulo 1024 [length K 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 11.9, the expanded message is represented as the sequence of 1024-bit blocks M1, M2, c, MN, so that the total length of the expanded message is N * 1024 bits.

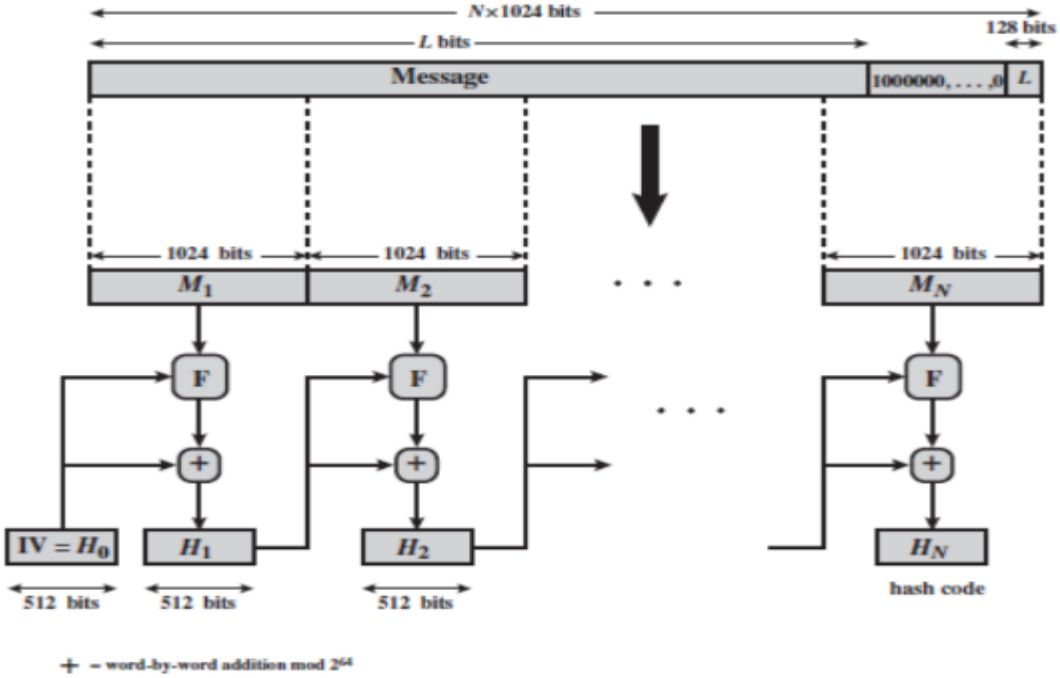


Figure 11.9 Message Digest Generation Using SHA-512

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):

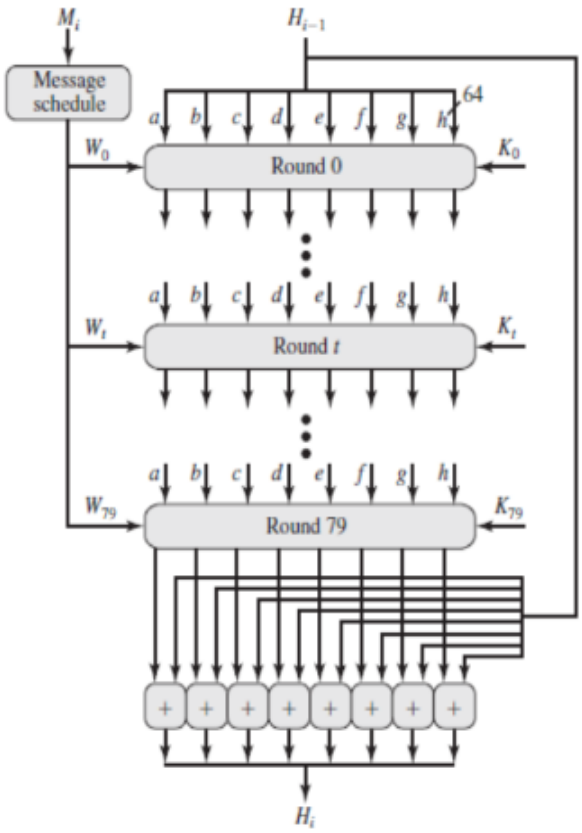
a = 6A09E667F3BCC908 e = 510E527FADE682D1

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 11.9. The logic is illustrated in Figure 11.10.



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, Hi-1. Each round t makes use of a 64-bit value Wt, derived from the current 1024-bit block being processed

(Mi). These values are derived using a message schedule described subsequently. Each round also makes use of an additive constant Kt, where 0 … t … 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.

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

SHA-512 Round Function

Let us look in more detail at the logic in each of the 80 steps of the processing of one 512-bit block

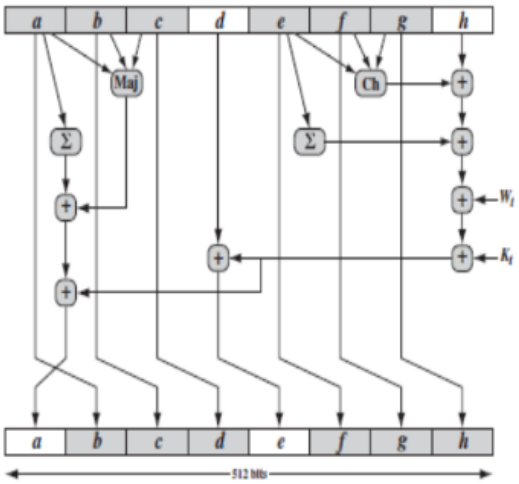


Figure 11.11 Elementary SHA-512 Operation (single round)

Each round is defined by the following set of equations:

$$T_1 = h + \text{Ch}(e, f, g) + \left(\sum_1^{512} e\right) + W_t + K_t$$
$$T_2 = \left(\sum_0^{512} a\right) + \text{Maj}(a, b, c)$$
$$h = g$$
$$g = f$$
$$f = e$$
$$e = d + T_1$$
$$d = c$$
$$c = b$$
$$b = a$$
$$a = T_1 + T_2$$

where

t = step number; $0 \leq t \leq 79$

$\text{Ch}(e, f, g)$ = $(e \text{ AND } f) \oplus (\text{NOT } e \text{ AND } g)$
the conditional function: If e then f else g

$\text{Maj}(a, b, c)$ = $(a \text{ AND } b) \oplus (a \text{ AND } c) \oplus (b \text{ AND } c)$
the function is true only of the majority (two or three) of the arguments are true

$\left(\sum_0^{512} a\right)$ = $\text{ROTR}^{28}(a) \oplus \text{ROTR}^{34}(a) \oplus \text{ROTR}^{39}(a)$

$\left(\sum_1^{512} e\right)$ = $\text{ROTR}^{14}(e) \oplus \text{ROTR}^{18}(e) \oplus \text{ROTR}^{41}(e)$

$\text{ROTR}^n(x)$ = circular right shift (rotation) of the 64-bit argument x by n bits

W_t = a 64-bit word derived from the current 1024-bit input block
K_t = a 64-bit additive constant, + = addition modulo 2^{64}

SHA-512 Key Expansion

Two observations can be made about the round function.

1. Six of the eight words of the output of the round function involve simply permutation (b, c, d, f , g, h) by means of rotation. This is indicated by shading in Figure 11.11.
2. Only two of the output words (a, e) are generated by substitution. Word e is a function of input variables (d, e, f , g, h), as well as the round word Wt and the constant Kt. Word a is a function of all of the input variables except d, as well as the round word Wt and the constant Kt. It remains to indicate how the 64-bit word values Wt are derived from the 1024-bit message.

Figure 11.12 illustrates the mapping.

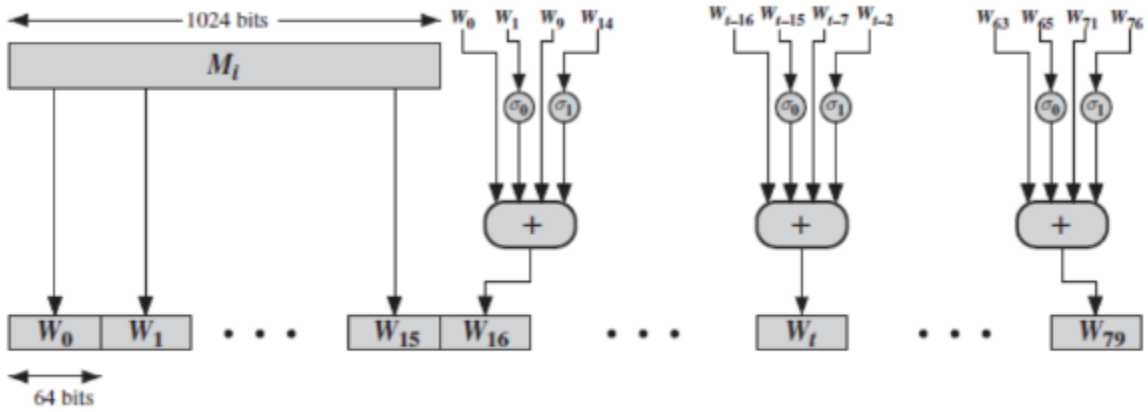


Figure 11.12 Creation of 80-word Input Sequence for SHA-512 Processing of Single Block

The first 16 values of Wt are taken directly from the 16 words of the current block. The remaining values are defined as

$$Wt = s1\ 512(Wt-2) + Wt-7 + s0\ 512(Wt-15) + Wt-16$$

where

$$s0\ 512(x) = ROTR1(x) \oplus ROTR8(x) \oplus SHR7(x)$$

$$s1\ 512(x) = ROTR19(x) \oplus ROTR61(x) \oplus SHR6(x)$$

ROTRn(x) = circular right shift (rotation) of the 64-bit argument x by n bits

SHRn(x) = left shift of the 64-bit argument x by n bits with padding by zeros on the right

+ = addition modulo 264

Thus, in the first 16 steps of processing, the value of Wt is equal to the corresponding word in the message block. For the remaining 64 steps, the value of Wt consists of the circular left shift by one bit of the XOR of four of the preceding values of Wt, with two of those values subjected to shift and rotate operations.

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. In either a connection-oriented or connectionless environment, the number and length of messages between parties could be determined.
3. Masquerade: Insertion of messages into the network from a fraudulent source. This includes the creation of messages by an opponent that are purported to come from an authorized entity. Also included are fraudulent acknowledgments of message receipt or nonreceipt by someone other than the message recipient.
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. In a connectionless application, an individual message (e.g., datagram) 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.

HMAC

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 SSL. HMAC has also been issued as a NIST standard (FIPS 198).

RFC 2104 lists the following design objectives for HMAC.

- 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.

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.

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)

Yi = i th block of M, 0 … i … (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 Ú n; if key length is greater than b, the key is input to the hash function to produce an n-bit key

K+ = K padded with zeros on the left so that the result is b bits in length

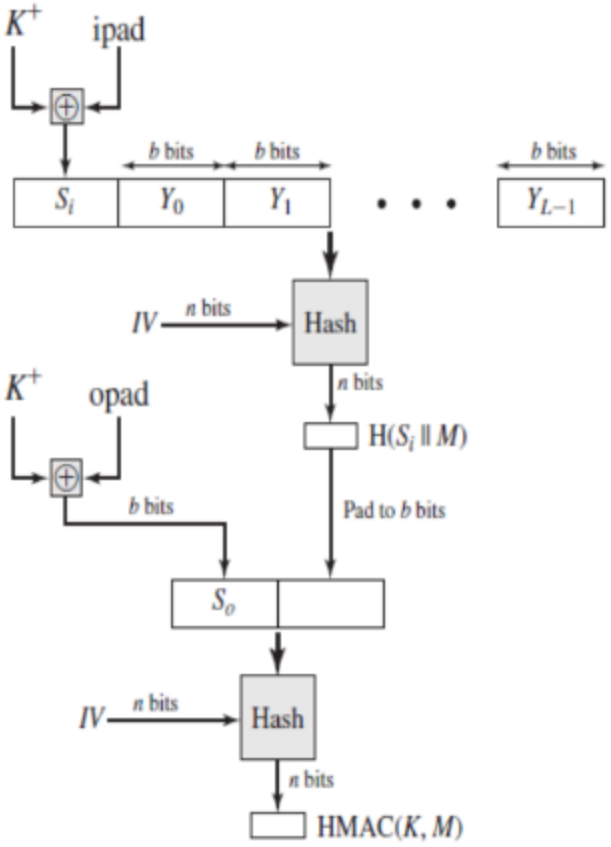


Figure 12.5 HMAC Structure

ipad = 00110110 (36 in hexadecimal) repeated b/8 times

opad = 01011100 (5C in hexadecimal) repeated b/8 times

Then HMAC can be expressed as

$$HMAC(K, M) = H[(K+ \_opad) \} H[(K+ \_ipad) \} M]]$$

We can describe the algorithm 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 zeroes).
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 So.
6. Append the hash result from step 4 to So.
7. Apply H to the stream generated in step 6 and output the result.

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

Security of HMAC

The probability of successful attack on HMAC is equivalent to one of the following attacks on the embedded hash function.

1. The attacker is able to compute an output of the compression function even with an IV that is random,
secret, and unknown to the attacker.
2. The attacker finds collisions in the hash function even when the IV is random and secret.

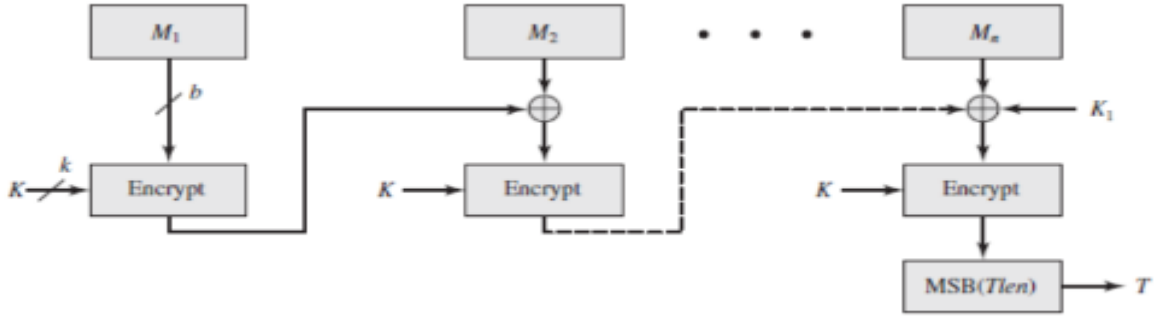
In the first attack, we can view the compression function as equivalent to the hash function applied to a message consisting of a single b-bit block. For this attack, the IV of the hash function is replaced by a secret, random value of n bits. An attack on this hash function requires either a brute-force attack on the key, which is a level of effort on the order of 2n, or a birthday attack, which is a special case of the second attack,

In the second attack, the attacker is looking for two messages M and M_ that produce the same hash: H(M) = H(M_). the attacker can choose any set of messages and work on these off line on a dedicated computing facility to find a collision. Because the attacker knows the hash algorithm and the default IV,

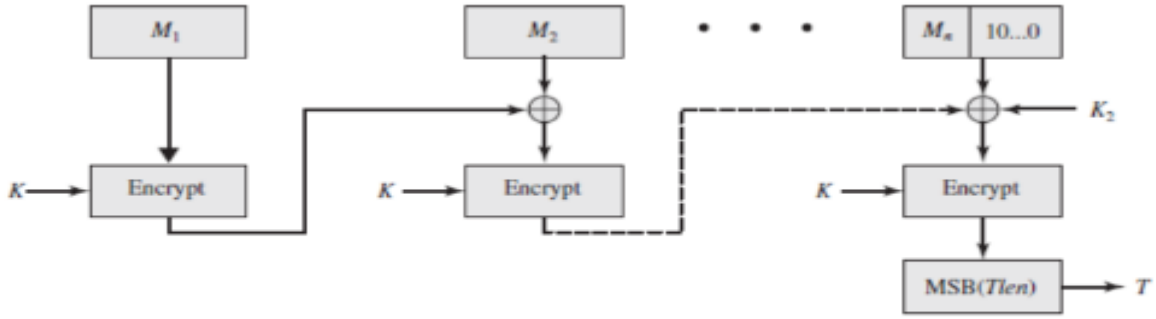
the attacker can generate the hash code for each of the messages that the attacker generates. However, when attacking HMAC, the attacker cannot generate message/ code pairs off line because the attacker does not know K. Therefore, the attacker must observe a sequence of messages generated by HMAC under the same key and perform the attack on these known messages.

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 (M1, M2,c, Mn). The algorithm makes use of a k-bit encryption key K and a b-bit constant, K1. 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)

$$C1 = E(K, M1)$$
$$C2 = E(K, [M2 \_C1])$$
$$C3 = E(K, [M3 \_C2])$$
$$.$$
$$.$$
$$.$$
$$Cn = E(K, [Mn \_Cn\text{-}1 \_K1])$$
$$T = MSBTlen(Cn)$$

where

T = message authentication code, also referred to as the tag

Tlen = bit length of T

MSBs(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 b-bit key K2 is used instead of K1.

The two b-bit keys are derived from the k-bit encryption key as follows.

$$L = E(K, 0b)$$
$$K1 = L \# x$$
$$K2 = L \# x2 = (L \# x) \# x$$

The two b-bit keys are derived from the k-bit encryption key as follows.

$$L = E(K, 0b)$$

$$K1 = L \# x$$

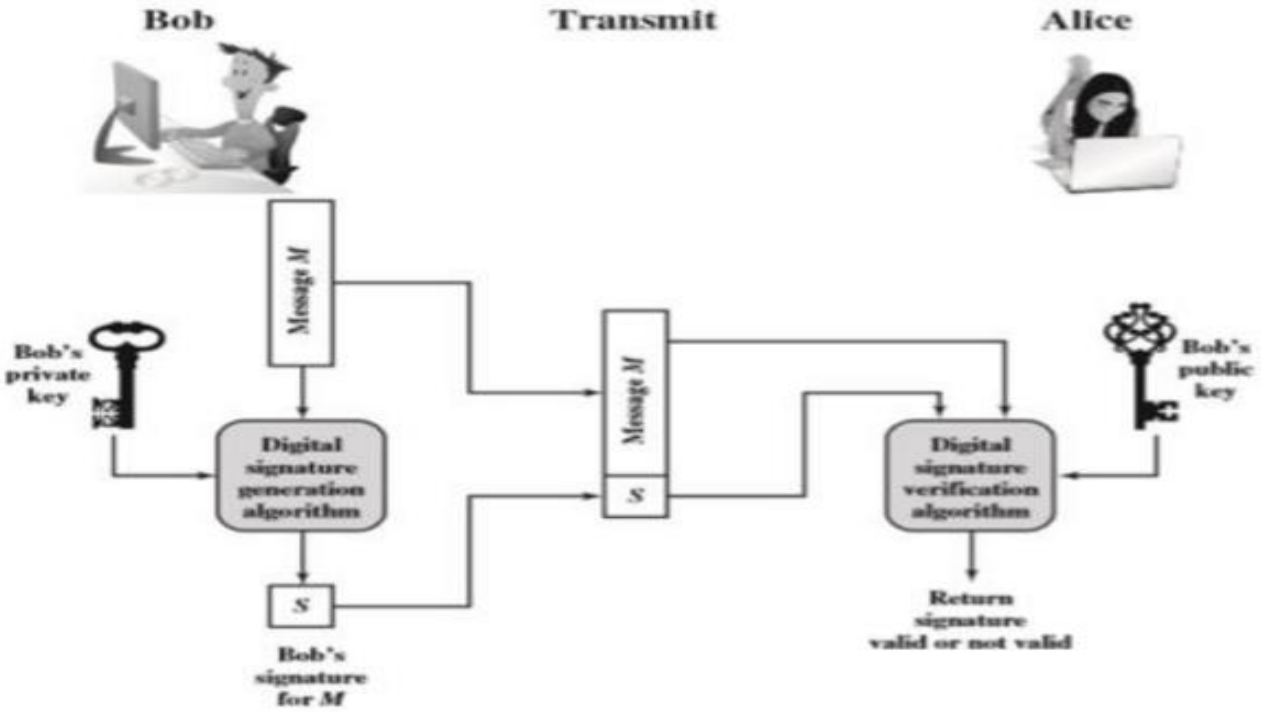
$$K2 = L \# x2 = (L \# x) \# x$$

where multiplication (#) is done in the finite field GF(2b) and x and x2 are first and second-order polynomials that are elements of GF(2b). Thus, the binary representation of x consists of b - 2 zeros followed by 10; the binary representation of x2 consists of b - 3 zeros followed by 100. The finite field is defined with respect to an irreducible polynomial that is lexicographically first among all such polynomials with the minimum possible number of nonzero terms. For the two approved block sizes, the polynomials are $x64 + x4 + x3 + x + 1$ and $x128 + x7 + x2 + x + 1$.

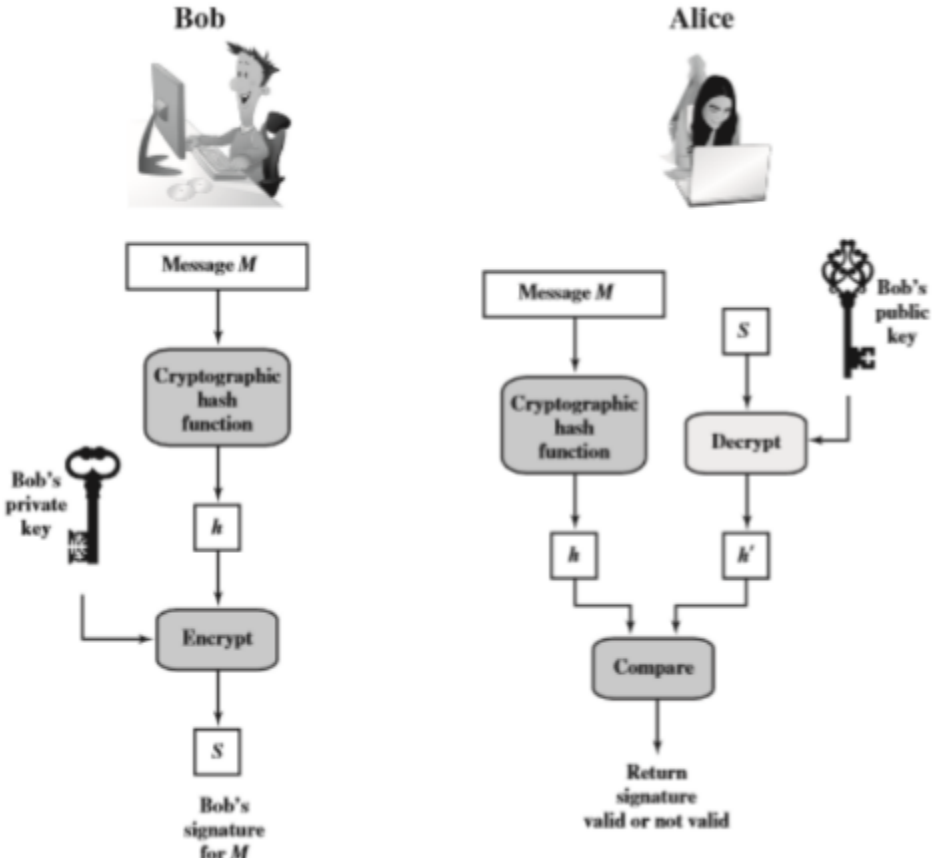
To generate K1 and K2, the block cipher is applied to the block that consists entirely of 0 bits. The first subkey is derived from the resulting ciphertext by a left shift of one bit and, conditionally, by XORing a constant that depends on the block size. The second subkey is derived in the same manner from the first subkey.

DIGITAL SIGNATURE

A digital signature is a digital code (generated and authenticated by public key encryption) which is attached to an electronically transmitted document to verify its contents and the sender's identity.



- In the above figure, represented the generic model of Digital signature process.
- Bob can sign a message using a digital signature generation algorithm.
- The inputs to the algorithm are the message and Bob's private key.
- Any other user, say Alice, can verify the signature using a verification algorithm, whose inputs are the message, the signature, and Bob's public key.



Properties

Message authentication protects two parties who exchange messages from any third party. However, it does not protect the two parties against each other.

The digital signature must have the following properties:

- It must verify the author and the date and time of the signature.
- It must authenticate the contents at the time of the signature.
- It must be verifiable by third parties, to resolve disputes.

Thus, the digital signature function includes the authentication function.

Digital Signature Requirements

Following are the requirements for a digital signature.

- The signature must use some information unique to the sender to prevent both forgery and denial.
- It must be relatively easy to produce the digital signature.
- It must be relatively easy to recognize and verify the digital signature.
- It must be computationally infeasible to forge a digital signature, either by constructing a new message for an existing digital signature or by constructing a fraudulent digital signature for a given message.
- It must be practical to retain a copy.
- Be practical to save digital signature in storage
- A secure hash function, embedded in a scheme, provides a basis for satisfying these requirements. However, care must be taken in the design of the details of the scheme.

Direct Digital Signature

- involves only sender and receiver
- assumed receiver has sender's public-key
- digital signature made by sender signing entire message or hash with private-key
- can encrypt using receivers public-key
- important that sign first then encrypt message and signature
- security depends on sender's private-key

Digital Signature Characteristics

- A public key scheme …

Two key pairs: a (long-time, permanent) durable private/public key pair

a (nonce-like, one-time, per-message) disposable private/public key pair

- Both key pairs generated by SENDER
- Signature is two numbers, depending on message hash and secret info
- A verification calculation succeeds iff the two numbers correctly depend on the secret info
- Disposable private/public key pair makes a collection of signatures of the sender uncorrelated, so hard to break, analytically or statistically.

ELGAMAL DIGITAL SIGNATURE SCHEME

The ElGamal signature scheme is a digital signature scheme which is based on the difficulty of computing discrete logarithms. It was described by Taher Elgamal in 1984.

- use private key for encryption (signing)
- uses public key for decryption (verification)
- each user generates their keys, for example: user A generates their key
  - chooses a secret key (number): $1 < x_A < q-1$
  - compute their public key: $y_A = a\ x_A \bmod q$
- Alice signs a message M to Bob by computing , the hash $m = H(M)$, $0 <= m <= (q-1)$
  - chose random integer K with $1 <= K <= (q-1)$ and $\gcd(K,q-1)=1$
  - compute temporary key: $S_1 = a^k \bmod q$
  - compute K^{-1} the inverse of K mod (q-1)
  - compute the value: $S_2 = K^{-1}(m-x_A S_1) \bmod (q-1)$
  - signature is:(S_1,S_2)
- any user B can verify the signature by computing
  - $V_1 = a^m \bmod q$
  - $V_2 = y_A^{S1}\ S_1^{S2} \bmod q$
  - signature is valid if $V_1 = V_2$
- Example problem:

For example, let us start with the prime field GF (19); that is, q = 19.

Table 8.3 Powers of Integers, Modulo 19

a	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}	a^{11}	a^{12}	a^{13}	a^{14}	a^{15}	a^{16}	a^{17}	a^{18}
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	1
3	9	8	5	15	7	2	6	18	16	10	11	14	4	12	17	13	1
4	16	7	9	17	11	6	5	1	4	16	7	9	17	11	6	5	1
5	6	11	17	9	7	16	4	1	5	6	11	17	9	7	16	4	1
6	17	7	4	5	11	9	16	1	6	17	7	4	5	11	9	16	1
7	11	1	7	11	1	7	11	1	7	11	1	7	11	1	7	11	1
8	7	18	11	12	1	8	7	18	11	12	1	8	7	18	11	12	1
9	5	7	6	16	11	4	17	1	9	5	7	6	16	11	4	17	1
10	5	12	6	3	11	15	17	18	9	14	7	13	16	8	4	2	1
11	7	1	11	7	1	11	7	1	11	7	1	11	7	1	11	7	1
12	11	18	7	8	1	12	11	18	7	8	1	12	11	18	7	8	1
13	17	12	4	14	11	10	16	18	6	2	7	15	5	8	9	3	1
14	6	8	17	10	7	3	4	18	5	13	11	2	9	12	16	15	1
15	16	12	9	2	11	13	5	18	4	3	7	10	17	8	6	14	1
16	9	11	5	4	7	17	6	1	16	9	11	5	4	7	17	6	1
17	4	11	16	6	7	5	9	1	17	4	11	16	6	7	5	9	1
18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1

It has primitive roots {2, 3, 10, 13, 14, 15}, as shown in the table,

We choose a = 10.

Alice generates a key pair as follows:

1. Alice chooses $X_A = 16$.

2. Then $Y_A = a^{XA} \bmod q = a^{16} \bmod 19 = 4$.

3. Alice's private key is 16; Alice's pubic key is $\{q, a, Y_A\} = \{19, 10, 4\}$.

Suppose Alice wants to sign a message with hash value m = 14.

1. Alice chooses K = 5, which is relatively prime to q - 1 = 18.

2. $S1 = a^K \bmod q = 105 \bmod 19 = 3$ (see Table).

3. $K\text{-}1 \bmod (q - 1) = 5\text{-}1 \bmod 18 = 11$.

4. $S2 = K\text{-}1 (m - X_A S1) \bmod (q - 1) = 11 (14 - (16) (3)) \bmod 18 = -374 \bmod 18 = 4$.

Bob can verify the signature as follows.

1. $V1 = a^m \bmod q = 1014 \bmod 19 = 16$.

2. $V2 = (Y^A)^{S1} (S1)^{S2} \bmod q = (4^3) (3^4) \bmod 19 = 5184 \bmod 19 = 16$.

As, V1=V2, thus, the signature is valid.

- any user B can verify the signature by computing
  - $V_1 = a^m \bmod q$
  - $V_2 = y_A^{S1} S_1^{S2} \bmod q$
  - signature is valid if $V_1 = V_2$

- Example problem:

For example, let us start with the prime field GF (19); that is, q = 19.

Table 8.3 Powers of Integers, Modulo 19

a	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}	a^{11}	a^{12}	a^{13}	a^{14}	a^{15}	a^{16}	a^{17}	a^{18}
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	1
3	9	8	5	15	7	2	6	18	16	10	11	14	4	12	17	13	1
4	16	7	9	17	11	6	5	1	4	16	7	9	17	11	6	5	1
5	6	11	17	9	7	16	4	1	5	6	11	17	9	7	16	4	1
6	17	7	4	5	11	9	16	1	6	17	7	4	5	11	9	16	1
7	11	1	7	11	1	7	11	1	7	11	1	7	11	1	7	11	1
8	7	18	11	12	1	8	7	18	11	12	1	8	7	18	11	12	1
9	5	7	6	16	11	4	17	1	9	5	7	6	16	11	4	17	1
10	5	12	6	3	11	15	17	18	9	14	7	13	16	8	4	2	1
11	7	1	11	7	1	11	7	1	11	7	1	11	7	1	11	7	1
12	11	18	7	8	1	12	11	18	7	8	1	12	11	18	7	8	1
13	17	12	4	14	11	10	16	18	6	2	7	15	5	8	9	3	1
14	6	8	17	10	7	3	4	18	5	13	11	2	9	12	16	15	1
15	16	12	9	2	11	13	5	18	4	3	7	10	17	8	6	14	1
16	9	11	5	4	7	17	6	1	16	9	11	5	4	7	17	6	1
17	4	11	16	6	7	5	9	1	17	4	11	16	6	7	5	9	1
18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1

It has primitive roots {2, 3, 10, 13, 14, 15}, as shown in the table,

We choose a = 10.

Alice generates a key pair as follows:

1. Alice chooses $X_A = 16$.

2. Then $Y_A = a^{XA} \bmod q = a^{16} \bmod 19 = 4$.

3. Alice's private key is 16; Alice's pubic key is $\{q, a, Y_A\} = \{19, 10, 4\}$.

Suppose Alice wants to sign a message with hash value m = 14.

1. Alice chooses K = 5, which is relatively prime to q - 1 = 18.

2. $S1 = a^K \bmod q = 105 \bmod 19 = 3$ (see Table).

3. $K\text{-}1 \bmod (q - 1) = 5\text{-}1 \bmod 18 = 11$.

4. $S2 = K\text{-}1 (m - X_A S1) \bmod (q - 1) = 11 (14 - (16) (3)) \bmod 18 = -374 \bmod 18 = 4$.

Bob can verify the signature as follows.

1. $V1 = a^m \bmod q = 1014 \bmod 19 = 16$.

2. $V2 = (Y^A)^{S1} (S1)^{S2} \bmod q = (4^3) (3^4) \bmod 19 = 5184 \bmod 19 = 16$.

As, V1=V2, thus, the signature is valid.

Symmetric Key Distribution Using Symmetric Encryption:

the strength of any cryptographic system rests with the key distribution technique, a 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.

Options 1 and 2 call for manual delivery of a key. For link encryption, this is a reasonable requirement, because each link encryption device is going to be exchanging data only with its partner on the other end of the link.

The scale of the problem depends on the number of communicating pairs that must be supported. If end-to-end encryption is done at a network or IP level, then a key is needed for each pair of hosts on the network that wish to communicate. Thus, if there are N hosts, the number of required keys is [N(N - 1)]/2.

Returning to our list, option 3 is a possibility for either link encryption or end-to-end encryption, but if an attacker ever succeeds in gaining access to one key, then all subsequent keys will be revealed. Furthermore, the initial distribution of potentially millions of keys still must be made. For end-to-end encryption, some variation on option 4 has been widely adopted.

In this scheme, a key distribution center is responsible for distributing keys to pairs of users (hosts, processes, applications) as needed. Each user must share a unique key with the key distribution center for purposes of key distribution. The use of a key distribution center is based on the use of a hierarchy of keys. At a minimum, two levels of keys are used (Figure 14.2).

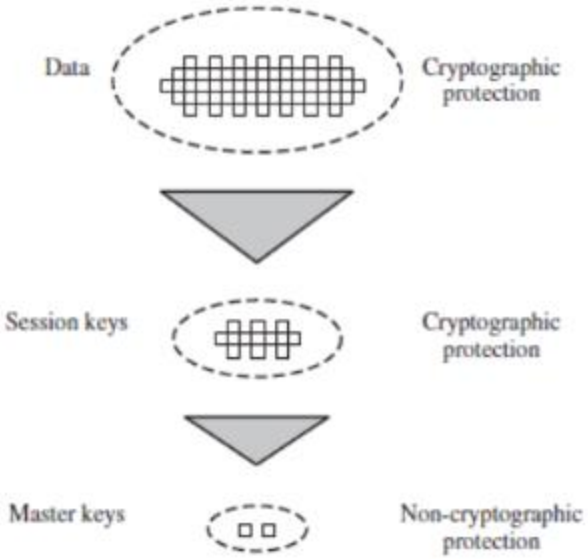


Figure 14.2 The Use of a Key Hierarchy

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, such as a frame relay connection or transport connection, and then discarded. Each session key is obtained from the key distribution center over the same networking facilities used for end-user communication. Accordingly, session keys are transmitted in encrypted form, using a master key that is shared by the key distribution center and an end system or user. For each end system or user, there is a unique master key that it shares with the key distribution center. Of course, these master keys must be distributed in some fashion.

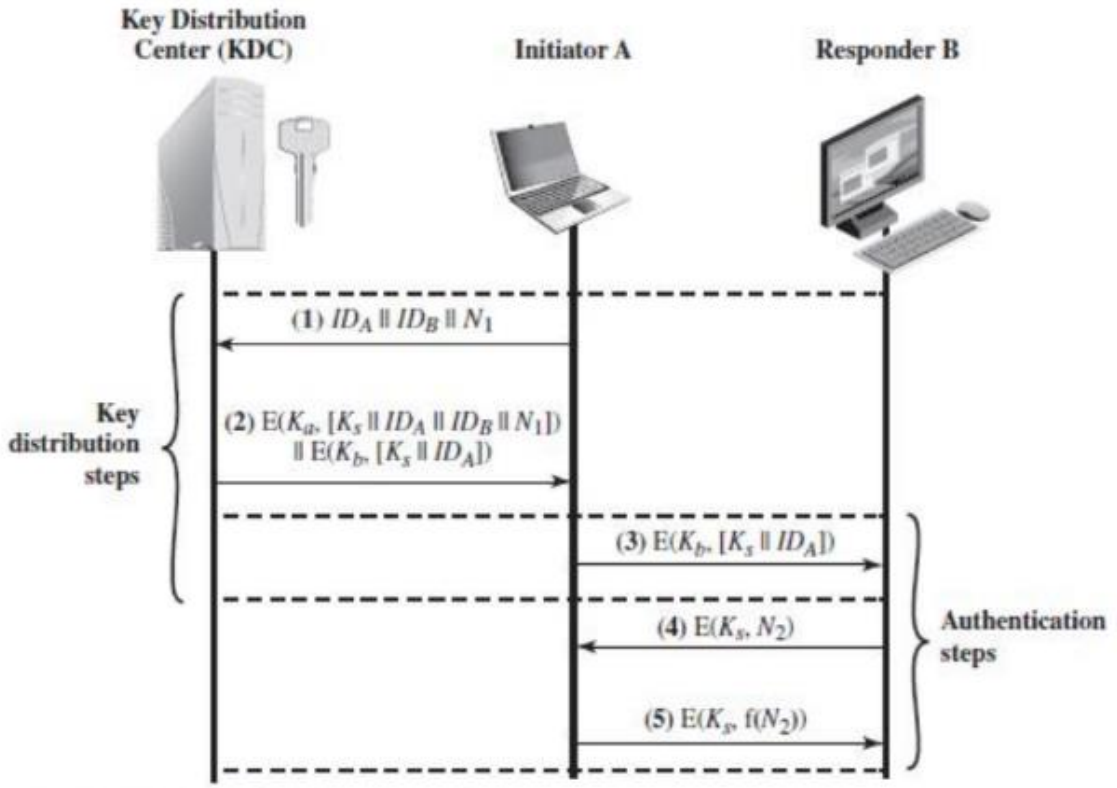


Figure 14.3 Key Distribution Scenario

Let us assume that user A wishes to establish a logical connection with B and requires a one-time session key to protect the data transmitted over the connection. A has a master key, Ka, known only to itself and the KDC; similarly, B shares the master key Kb with the KDC. The following steps occur.

1. A issues a request to the KDC for a session key to protect a logical connection to B. The message includes the identity of A and B and a unique identifier, N1, for this transaction, which we refer to as a nonce. The nonce may be a counter, or a random number; the minimum requirement is that it differs with each request. Also, to prevent masquerade, it should be difficult for an opponent to guess the nonce. Thus, a random number is a good choice for a nonce.

2. The KDC responds with a message encrypted using Ka. Thus, A is the only one who can successfully read the message, and A knows that it originated at the KDC. The message includes two items intended for A:

- The one-time session key, Ks, to be used for the session
- The original request message, including the nonce, to enable A to match this response with the appropriate request Thus, A can verify that its original request was not altered before reception by the KDC and, because of the nonce, that this is not a replay of some previous request.

In addition, the message includes two items intended for B:

- The one-time session key, Ks, to be used for the session
- An identifier of A (e.g., its network address), IDA. These last two items are encrypted with Kb (the master key that the KDC shares with B). They are to be sent to B to establish the connection and prove A's identity.

3. A stores the session key for use in the upcoming session and forwards to B the information that originated at the KDC for B, namely, E(Kb,[Ks } IDA]). Because this information is encrypted with Kb, it is protected from eavesdropping. B now knows the session key (Ks), knows that the other party is A (from IDA), and knows that the information originated at the KDC (because it is encrypted using Kb).

At this point, a session key has been securely delivered to A and B, and they may begin their protected exchange. However, two additional steps are desirable:

4. Using the newly minted session key for encryption, B sends a nonce, N2, to A.

5. Also, using Ks, A responds with f(N2), where f is a function that performs some transformation on N2 (e.g., adding one).

Different types of Key Distribution Centers:

Hierarchical Key Control

It is not necessary to limit the key distribution function to a single KDC. Indeed, for very large networks, it may not be practical to do so. As an alternative, a hierarchy of KDCs can be established. For example, there can be local KDCs, each responsible for a small domain of the overall internetwork, such as a single LAN or a single building. For communication among entities within the same local domain, the local KDC is responsible for key distribution. If two entities in different domains desire a shared key, then the corresponding local KDCs can communicate through a global KDC. In this case, any one of the three KDCs involved can actually select the key.

The hierarchical concept can be extended to three or even more layers, depending on the size of the user population and the geographic scope of the internetwork. A hierarchical scheme minimizes the effort involved in master key distribution, because most master keys are those shared by a local KDC with its local entities.

Session Key Lifetime

The more frequently session keys are exchanged, the more secure they are, because the opponent has less ciphertext to work with for any given session key. On the other hand, the distribution of session keys delays the start of any exchange and places a burden on network capacity. A security manager must try to balance these competing considerations in determining the lifetime of a particular session key.

A Transparent Key Control Scheme

The steps involved in establishing a connection are shown in Figure 14.4. When one host wishes to set up a connection to another host, it transmits a connection-request packet (step 1). The SSM saves that packet and applies to the KDC for permission to establish the connection (step 2). The communication between the SSM and the KDC is encrypted using a master key shared only by this SSM and the KDC. If the KDC approves the connection request, it generates the session key and delivers it to the two appropriate SSMs, using a unique permanent key for each SSM (step 3). The requesting SSM can now release the connection request packet, and a connection is set up between the two end systems (step 4). All user data exchanged between the two end systems are encrypted by their respective SSMs using the one-time session key.

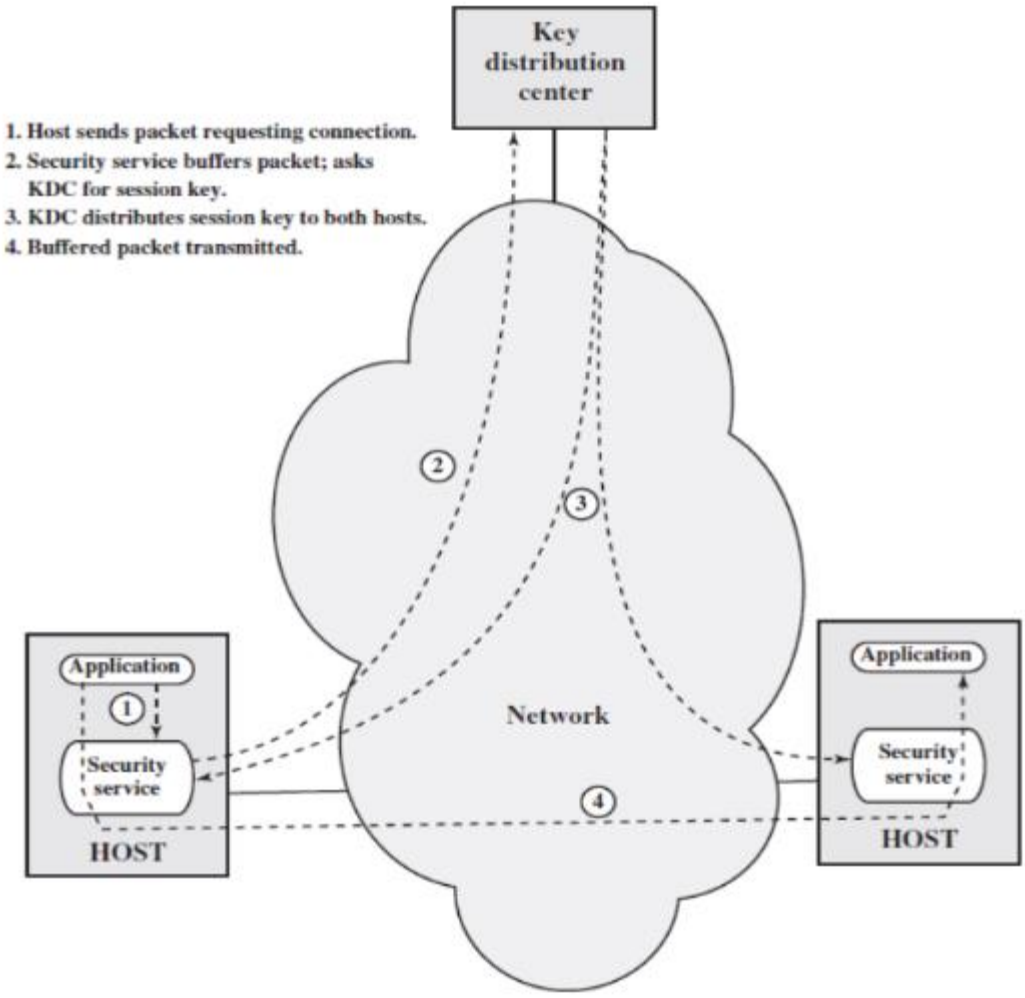


Figure 14.4 Automatic Key Distribution for Connection-Oriented Protocol

Decentralized Key Control

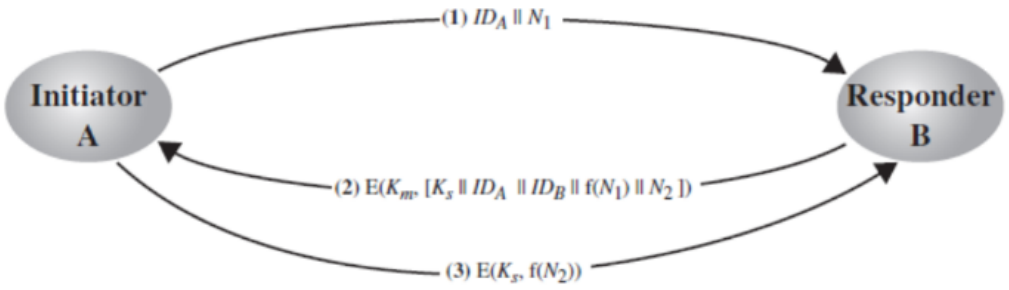


Figure 14.5 Decentralized Key Distribution

A session key may be established with the following sequence of steps (Figure 14.5).

1. A issues a request to B for a session key and includes a nonce, N1.
2. B responds with a message that is encrypted using the shared master key. The response includes the session key selected by B, an identifier of B, the value f(N1), and another nonce, N2.
3. Using the new session key, A returns f(N2) to B.

Symmetric Key Distribution Using Asymmetric Encryption

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 {PUa, PRa} and transmits a message to B consisting of PUa and an identifier of A, IDA.
2. B generates a secret key, Ks, and transmits it to A, which is encrypted with A's public key.

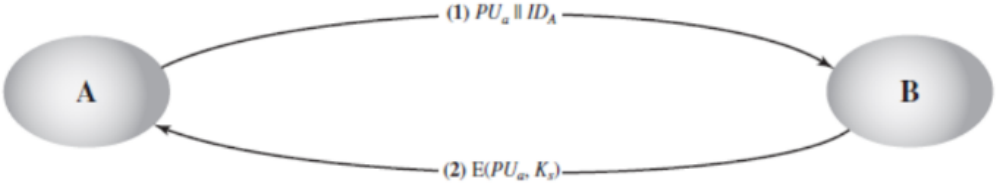


Figure 14.7 Simple Use of Public-Key Encryption to Establish a Session Key

3. A computes D(PRa, E(PUa, Ks)) to recover the secret key. Because only A can decrypt the message, only A and B will know the identity of Ks.
4. A discards PUa and PRa and B discards PUa. A and B can now securely communicate using conventional encryption and the session key Ks. At the completion of the exchange, both A and B discard Ks.

The protocol depicted in Figure 14.7 is insecure against an adversary who can intercept messages and then either relay the intercepted message or substitute another message (see Figure 1.3c). Such an attack is known as a man-in-the-middle attack [RIVE84]. We saw this type of attack in the present case, if an adversary, D, has control of the intervening communication channel, then D can compromise the communication in the following fashion without being detected (Figure 14.8).

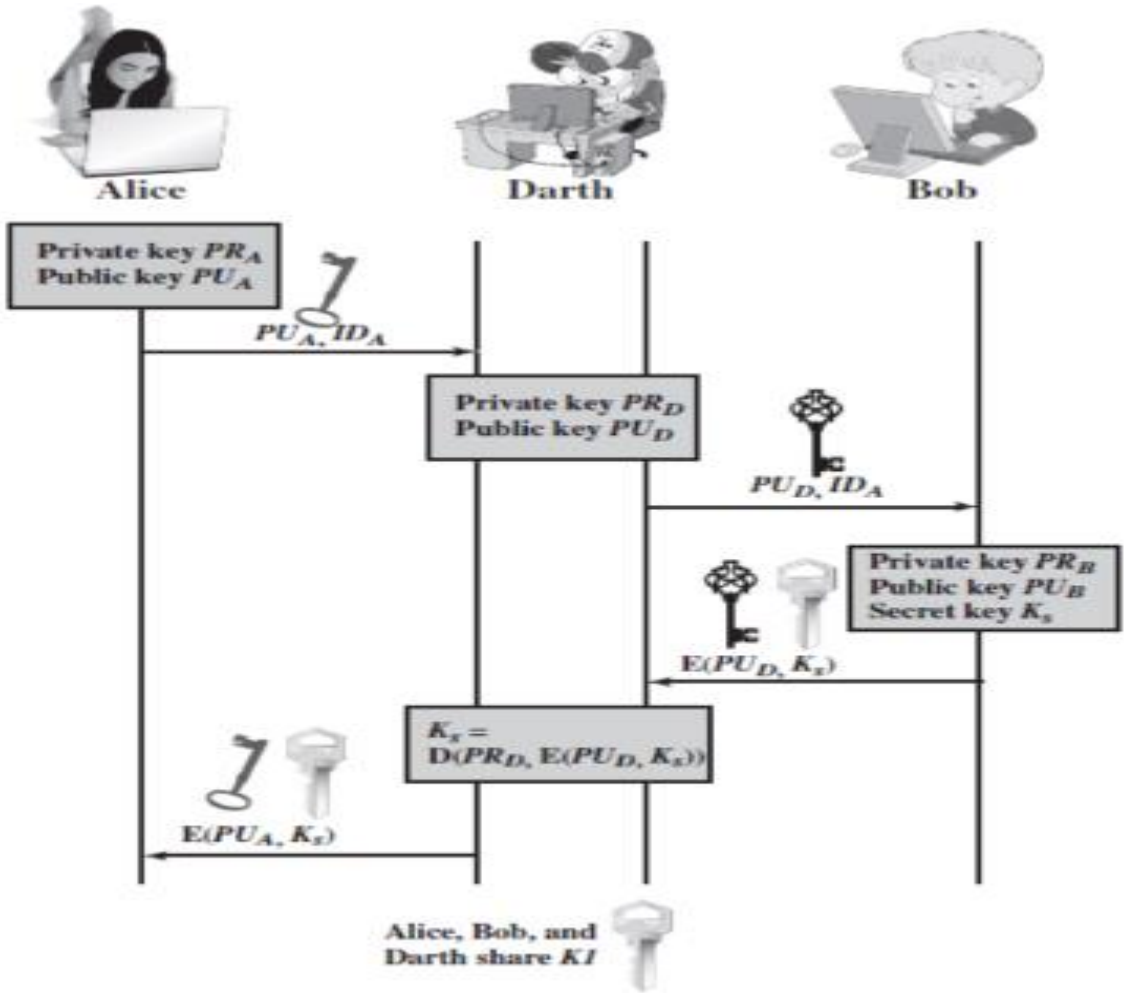


Figure 14.8 Another Man-in-the-Middle Attack

1. A generates a public/private key pair {PUa, PRa} and transmits a message intended for B consisting of PUa and an identifier of A, IDA.
2. D intercepts the message, creates its own public/private key pair {PUd, PRd} and transmits PUs 0 0 IDA to B.
3. B generates a secret key, Ks, and transmits E(PUs, Ks).
4. D intercepts the message and learns Ks by computing D(PRd, E(PUd, Ks)).
5. D transmits E(PUa, Ks) to A.

The result is that both A and B know Ks and are unaware that Ks has also been revealed to D. A and B can now exchange messages using Ks. D no longer actively interferes with the communications channel but simply eavesdrops. Knowing Ks, S can decrypt all messages, and both A and B are unaware of the problem. Thus, this simple protocol is only useful in an environment where the only threat is eavesdropping.

Secret Key Distribution with Confidentiality and Authentication

Figure 14.9, based on an approach suggested in [NEED78], provides protection against both active and passive attacks. We begin at a point when it is assumed that A and B have exchanged public keys by one of the schemes described subsequently in this chapter. Then the following steps occur.

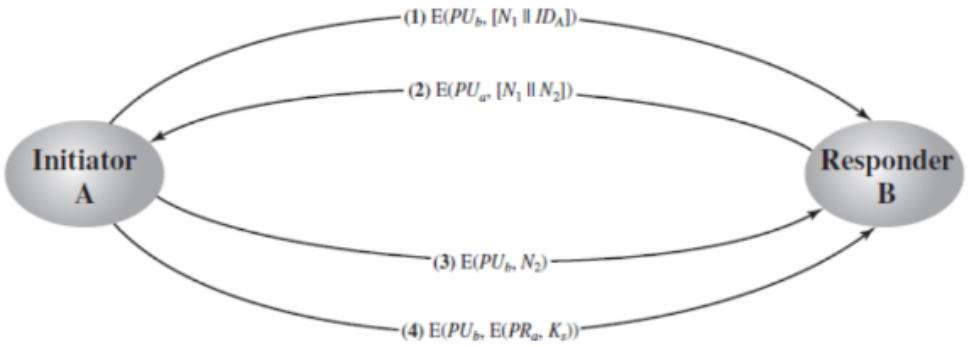


Figure 14.9 Public-Key Distribution of Secret Keys

1.

1.A uses B's public key to encrypt a message to B containing an identifier of A(IDA) and a nonce (N1), which is used to identify this transaction uniquely.

2. B sends a message to A encrypted with PUa and containing A's nonce (N1) as well as a new nonce generated by B (N2). Because only B could have decrypted message (1), the presence of N1 in message (2) assures A that the correspondent is B.

3. A returns N2, encrypted using B's public key, to assure B that its correspondent is A.

4. A selects a secret key Ks and sends M = E(PUb, E(PRa, Ks)) to B. Encryption of this message with B's public key ensures that only B can read it; encryption with A's private key ensures that only A could have sent it.

5. B computes D(PUa, D(PRb, M)) to recover the secret key.

The result is that this scheme ensures both confidentiality and authentication in the exchange of a secret key.

Distribution of Public Keys

Several techniques have been proposed for the distribution of public keys. Virtually all these proposals can be grouped into the following general schemes:

- Public announcement
- Publicly available directory
- Public-key authority
- Public-key certificates

Public Announcement of Public Keys:

Figure 14.10 Uncontrolled Public-Key Distribution

On the face of it, the point of public-key encryption is that the public key is public.Thus, if there is some broadly accepted public-key algorithm, such as RSA, any participant can send his or her public key to any other participant or broadcast the key to the community at large (Figure 14.10). it has a major weakness. Anyone can forge such a public announcement. That is, 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 intended for A and can use the forged keys for authentication .

Publicly Available Directory:

A greater degree of security can be achieved by maintaining a publicly available dynamic directory of public keys. Maintenance and distribution of the public directory would have to be the responsibility of some trusted entity or organization (Figure 14.11).

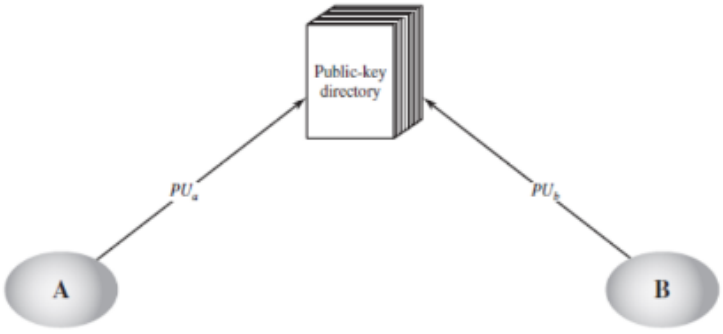


Figure 14.11 Public-Key Publication

Such a scheme would include the following elements:

1. The authority maintains a directory with a {name, public key} entry for each participant.
2. Each participant registers a public key with the directory authority. Registration would have to be in person or by some form of secure authenticated communication.
3. A participant may replace the existing key with a new one at any time, either because of the desire to replace a public key that has already been used for a large amount of data, or because the corresponding private key has been compromised in some way.
4. Participants could also access the directory electronically. For this purpose, secure, authenticated communication from the authority to the participant is mandatory.

This scheme is clearly more secure than individual public announcements but still has vulnerabilities. If an adversary succeeds in obtaining or computing the private key of the directory authority, the adversary could authoritatively pass out counterfeit public keys and subsequently impersonate any participant and eavesdrop on messages sent to any participant. Another way to achieve the same end is for the adversary to tamper with the records kept by the authority.

Public-Key Authority

Stronger security for public-key distribution can be achieved by providing tighter control over the distribution of public keys from the directory. The following steps (matched by number to Figure 14.12) occur.

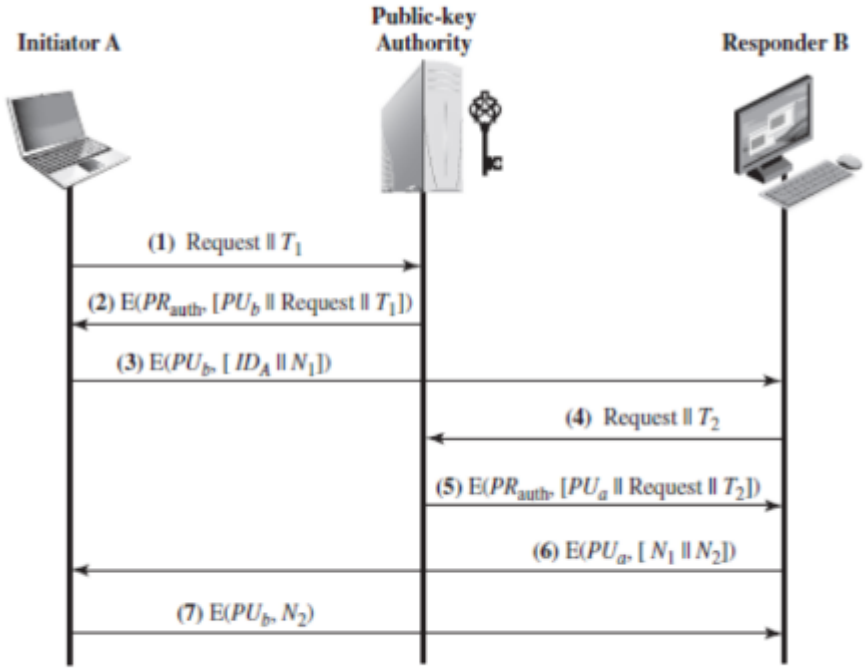


Figure 14.12 Public-Key Distribution Scenario

1. A sends a time stamped message to the public-key authority containing a request for the current public key of B.

2. The authority responds with a message that is encrypted using the authority's private key, PRauth. Thus, A is able to decrypt the message using the authority's public key. Therefore, A is assured that the message originated with the authority. The message includes the following:

- B's public key, PUb, which A can use to encrypt messages destined for B
- The original request used to enable A to match this response with the corresponding earlier request and to verify that the original request was not altered before reception by the authority
- The original timestamp given so A can determine that this is not an old message from the authority containing a key other than B's current public key

3. A stores B's public key and also uses it to encrypt a message to B containing an identifier of A (IDA) and a nonce (N1), which is used to identify this transaction uniquely.

4, 5. B retrieves A's public key from the authority in the same manner as A retrieved B's public key.

At this point, public keys have been securely delivered to A and B, and they may begin their protected exchange. However, two additional steps are desirable:

6. B sends a message to A encrypted with PUa and containing A's nonce (N1) as well as a new nonce generated by B (N2). Because only B could have decrypted message (3), the presence of N1 in message (6) assures A that the correspondent is B.

7. A returns N2, which is encrypted using B's public key, to assure B that its correspondent is A.

Public-Key Certificates

The scenario of Figure 14.12 is attractive, yet it has some drawbacks. The public-key authority could be somewhat of a bottleneck in the system, for a user must appeal to the authority for a public key for every other user that it wishes to contact. As before, the directory of names and public keys maintained by the authority is vulnerable to tampering.

An alternative approach, first suggested by Kohnfelder [KOHN78], is to use certificates that can be used by participants to exchange keys without contacting a public-key authority, in a way that is as reliable as if the keys were obtained directly from a public-key authority. In essence, a certificate consists of a public key, an identifier of the key owner, and the whole block signed by a trusted third party.

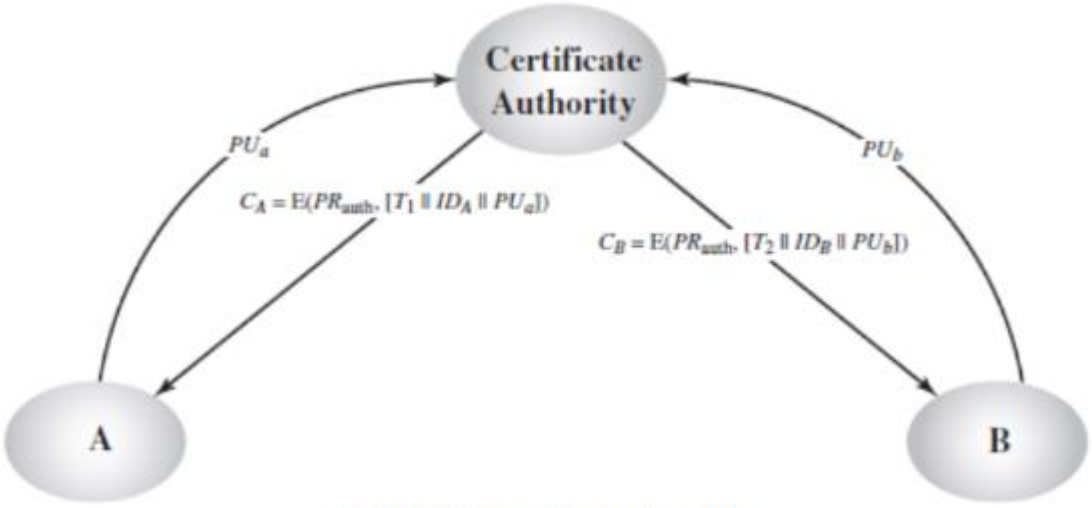
Typically, the third party is a certificate authority, such as a government agency or a financial institution, that is trusted by the user community. A user can present his or her public key to the authority in a secure manner and obtain a certificate. The user can then publish the certificate. Anyone

needing this user's public key can obtain the certificate and verify that it is valid by way of the attached trusted signature.

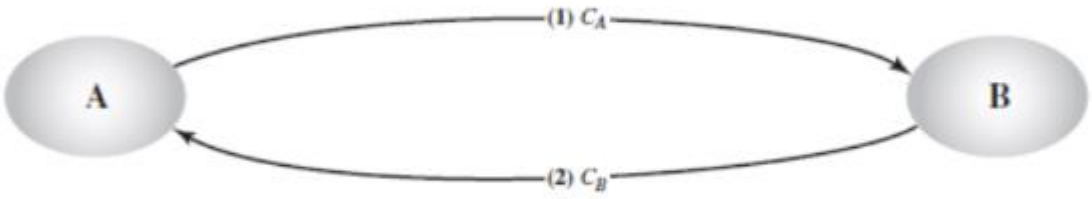
A participant can also convey its key information to another by transmitting its certificate. Other participants can verify that the certificate was created by the authority. We can place the following requirements on this scheme:

1. Any participant can read a certificate to determine the name and public key of the certificate's owner
2. Any participant can verify that the certificate originated from the certificate authority and is not counterfeit.
3. Only the certificate authority can create and update certificates.
4. Any participant can verify the currency of the certificate.

A certificate scheme is illustrated in Figure 14.13. Each participant applies to the certificate authority, supplying a public key and requesting a certificate. Application must be in person or by some form of secure authenticated communication.



(a) Obtaining certificates from CA



(b) Exchanging certificates

Figure 14.13 Exchange of Public-Key Certificates

For participant A, the authority provides a certificate of the form

CA = E(PRauth, [T} IDA }PUa])

where PRauth is the private key used by the authority and T is a timestamp. A may then pass this certificate on to any other participant, who reads and verifies the certificate as follows:

D(PUauth, CA) = D(PUauth, E(PRauth, [T} IDA }PUa])) = (T} IDA }PUa)

- The recipient uses the authority's public key, PUauth, to decrypt the certificate. Because the certificate is readable only using the authority's public key, this verifies that the certificate came from the certificate authority.

- The timestamp T validates the currency of the certificate.
- The timestamp counters the following scenario. A's private key is learned by an adversary. A generates a new private/public key pair and applies to the certificate authority for a new certificate. Meanwhile, the adversary replays the old certificate to B. If B then encrypts messages using the compromised old public key, the adversary can read those messages.

X.509 Certificate

X.509 is based on the use of public-key cryptography and digital signatures. The standard does not dictate the use of a specific algorithm but recommends RSA. The digital signature scheme is assumed to require the use of a hash function Figure 14.14 illustrates the generation of a public-key certificate.

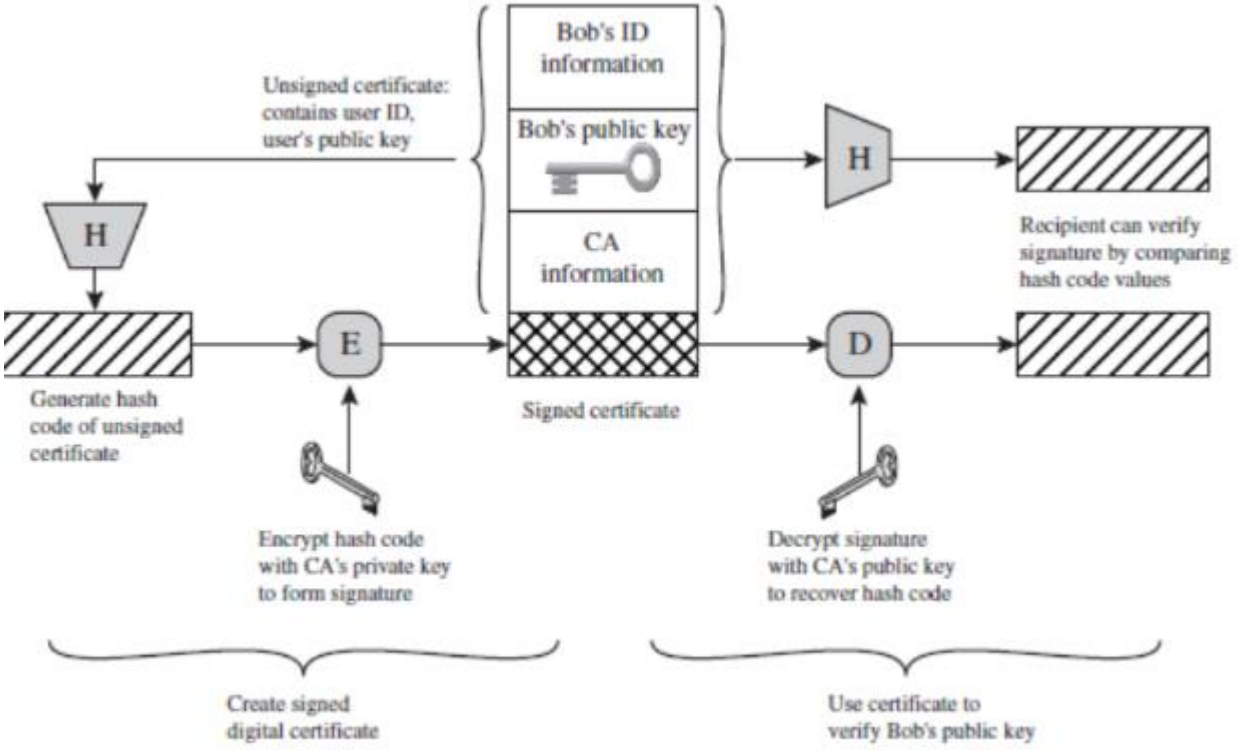


Figure 14.14 Public-Key Certificate Use

Certificates

The heart of the X.509 scheme is the public-key certificate associated with each user. These user certificates are assumed to be created by some trusted certification authority (CA) and placed in the directory by the CA or by the user. The directory server itself is not responsible for the creation of public keys or for the certification function; it merely provides an easily accessible location for users to obtain certificates. Figure 14.15a shows the general format of a certificate, which includes the following elements.

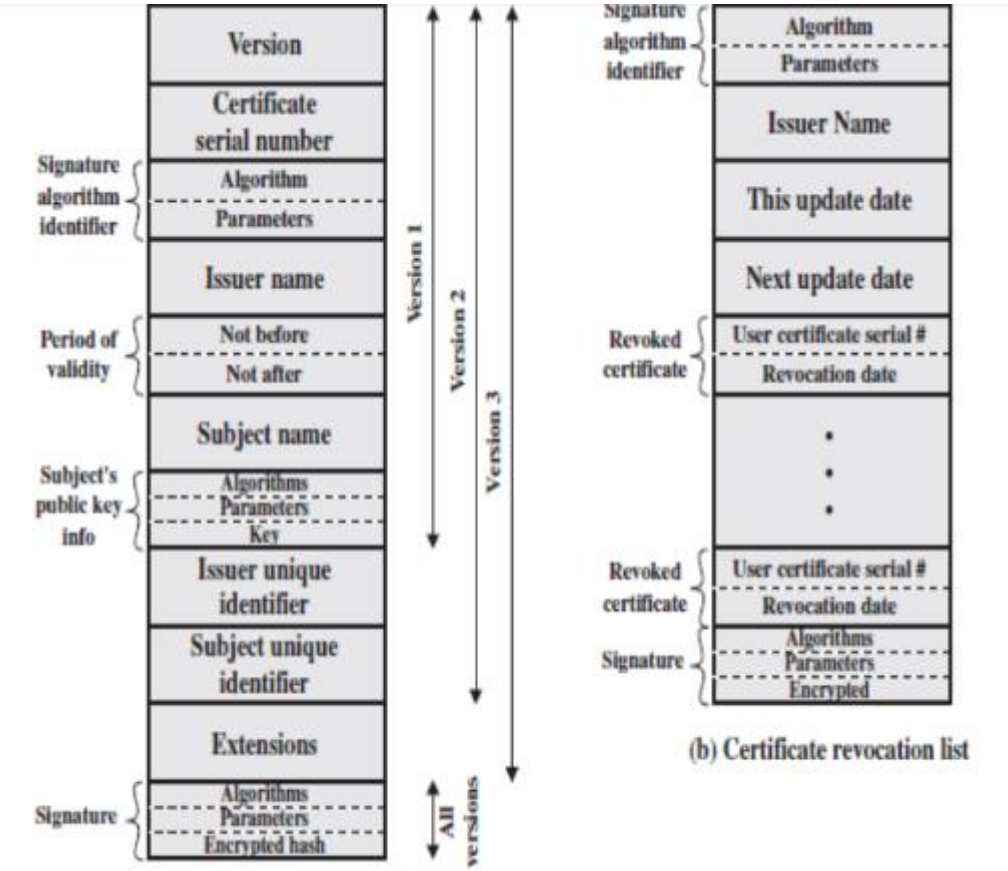


Figure 14.15 X.509 Formats

Obtaining a User's Certificate :

Step 1 A obtains from the directory the certificate of X2 signed by X1. Because A securely knows X1's public key, A can obtain X2's public key from its certificate and verify it by means of X1's signature on the certificate.

Step 2 A then goes back to the directory and obtains the certificate of B signed by X2. Because A now has a trusted copy of X2's public key, A can verify the signature and securely obtain B's public key.

A has used a chain of certificates to obtain B's public key. In the notation of X.509, this chain is expressed as

$$X_1 \ll X_2 \gg X_2 \ll B \gg$$

In the same fashion, B can obtain A's public key with the reverse chain:

$$X_2 \ll X_1 \gg X_1 \ll A \gg$$

This scheme need not be limited to a chain of two certificates. An arbitrarily long path of CAs can be followed to produce a chain. A chain with N elements would be expressed as

$$X_1 \ll X_2 \gg X_2 \ll X_3 \gg \ldots X_N \ll B \gg$$

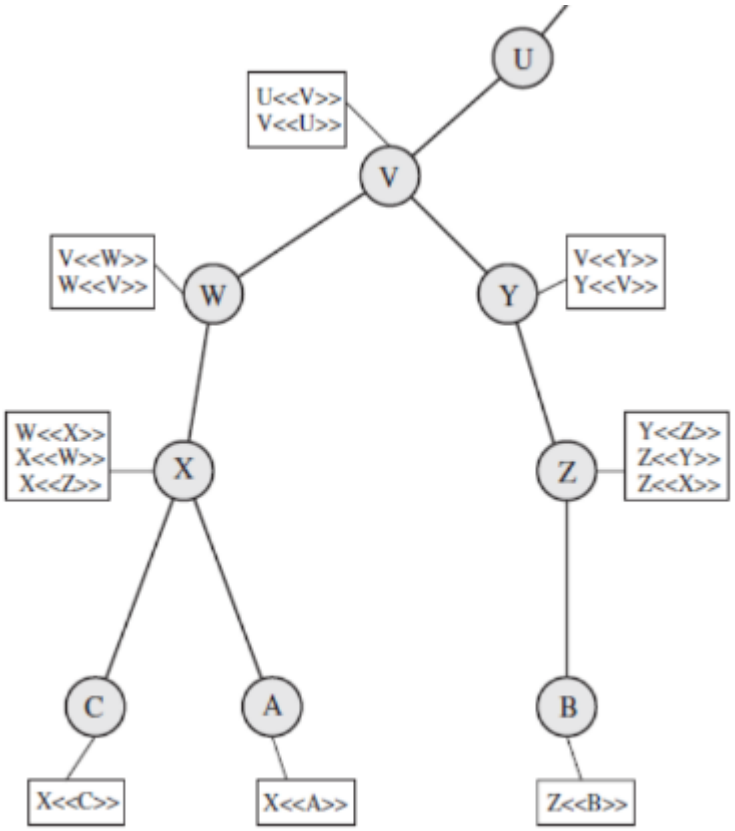


Figure 14.16 X.509 Hierarchy: A Hypothetical Example

Revoc ation of Certificates: Recall from Figure 14.15 that each certificate includes a period of validity, much like a credit card. Typically, a new certificate is issued just before the expiration of the old one. In addition, it may be desirable on occasion to revoke a certificate before it expires, for one of the following reasons.

1. The user's private key is assumed to be compromised.
2. The user is no longer certified by this CA. Reasons for this include that the subject's name has changed, the certificate is superseded, or the certificate was not issued in conformance with the CA's policies.
3. The CA's certificate is assumed to be compromised.

Each CA must maintain a list consisting of all revoked but not expired certificates issued by that CA, including both those issued to users and to other CAs. These lists should also be posted on the directory. Each certificate revocation list (CRL) posted to the directory is signed by the issuer and includes (Figure 14.15b) the issuer's name, the date the list was created, the date the next CRL is scheduled to be issued, and an entry for each revoked certificate. Each entry consists of the serial number of a certificate and revocation date for that certificate. Because serial numbers are unique within a CA, the serial number is sufficient to identify the certificate. When a user receives a certificate in a message, the user must determine whether the certificate has been revoked. The user could check the directory each time a certificate is received. To avoid the delays (and possible costs) associated with directory searches, it is likely that the user would maintain a local cache of certificates and lists of revoked certificates.

Kerberos

Kerberos4 is an authentication service developed as part of Project Athena at MIT. The problem that Kerberos addresses is this: Assume an open distributed environment in which users at workstations wish to access services on servers distributed throughout the network. We would like for servers to be able to restrict access to authorized users and to be able to authenticate requests for service.

In this environment, a workstation cannot be trusted to identify its users correctly to network services. In particular, the following three threats exist:

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.

.Motivation

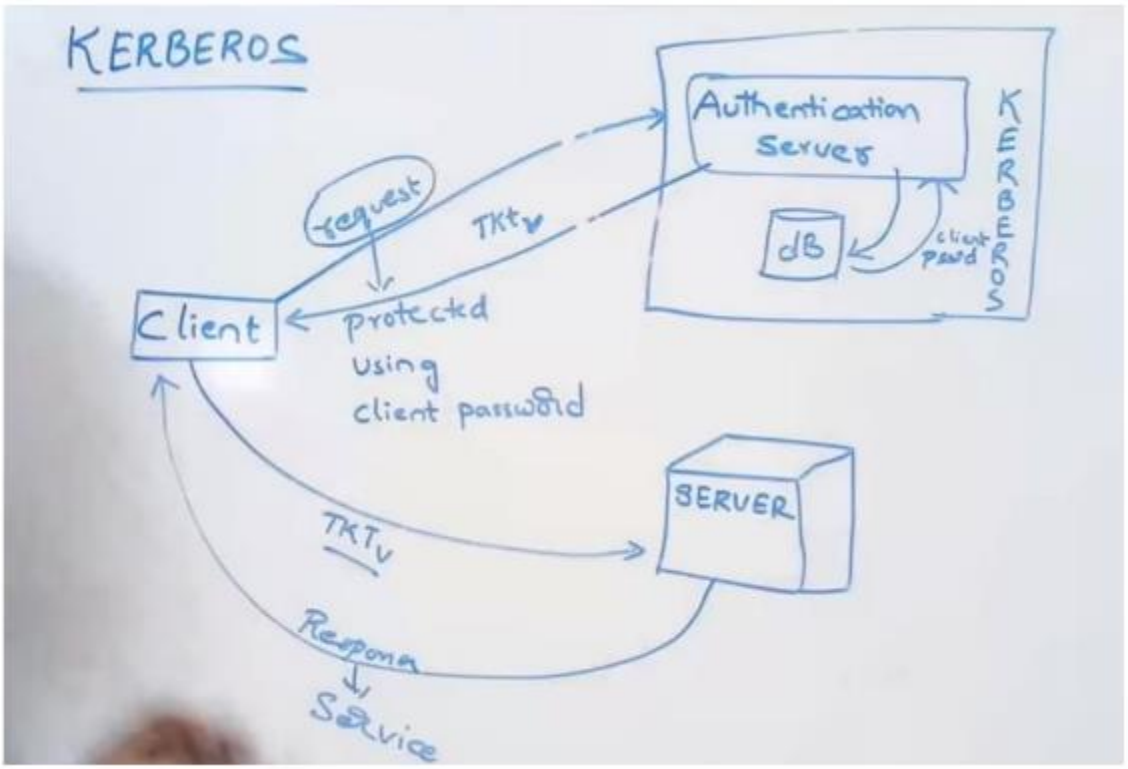
If a set of users is provided with dedicated personal computers that have no network connections, then a user's resources and files can be protected by physically securing each personal computer. When these users instead are served by a centralized time-sharing system, the time-sharing operating system must provide the security. The operating system can enforce access-control policies based on user identity and use the logon procedure to identify users.

The first published report on Kerberos [STEI88] listed the following requirements.

- Secure: A network eavesdropper should not be able to obtain the necessary information to impersonate a user. More generally, Kerberos should be strong enough that a potential opponent does not find it to be the weak link.
- Reliable: For all services that rely on Kerberos for access control, lack of availability of the Kerberos service means lack of availability of the supported services. Hence, Kerberos should be highly reliable and should employ a distributed server architecture with one system able to back up another.
- Transparent: Ideally, the user should not be aware that authentication is taking place beyond the requirement to enter a password.
- Scalable: The system should be capable of supporting large numbers of clients and servers. This suggests a modular, distributed architecture. To support these requirements, the overall scheme of Kerberos is that of a trusted third-party authentication service that uses a protocol based on that proposed by Needham and Schroeder [NEED78], It is trusted in the sense that clients and servers trust Kerberos to mediate their mutual authentication. Assuming the Kerberos protocol is well designed, then the authentication service is secure if the Kerberos server itself is secure.7

Kerberos Version 4

Version 4 of Kerberos makes use of DES, in a rather elaborate protocol, to provide the authentication service. Viewing the protocol as a whole, it is difficult to see the need for the many elements contained therein. Therefore, we adopt a strategy used by Bill Bryant of Project Athena [BRYA88] and build up to the full protocol by looking first at several hypothetical dialogues. Each successive dialogue adds additional complexity to counter security vulnerabilities revealed in the preceding dialogue. After examining the protocol, we look at some other aspects of version 4.

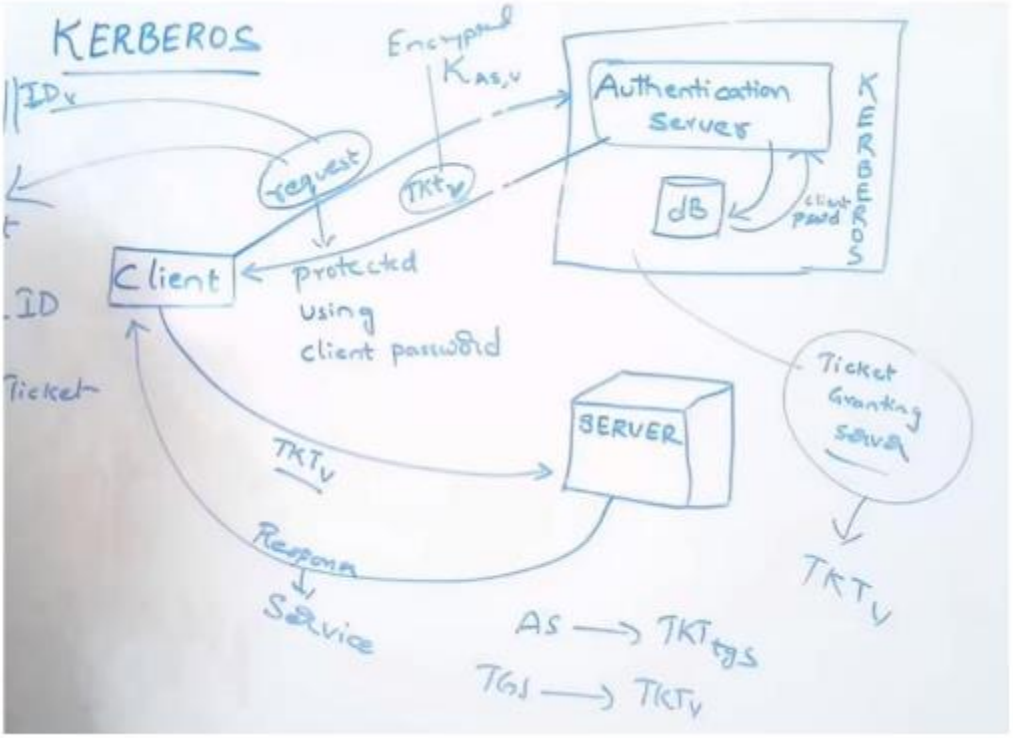


In this scenario,

- 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 password 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 problem, we would like to minimize the number of times that a user has to enter a password. The second problem is that the earlier scenario involved a plaintext transmission of the password



The new service, TGS, issues tickets to users who have been authenticated to AS. Thus, the user first requests a ticket-granting ticket (Tickettgs) 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.

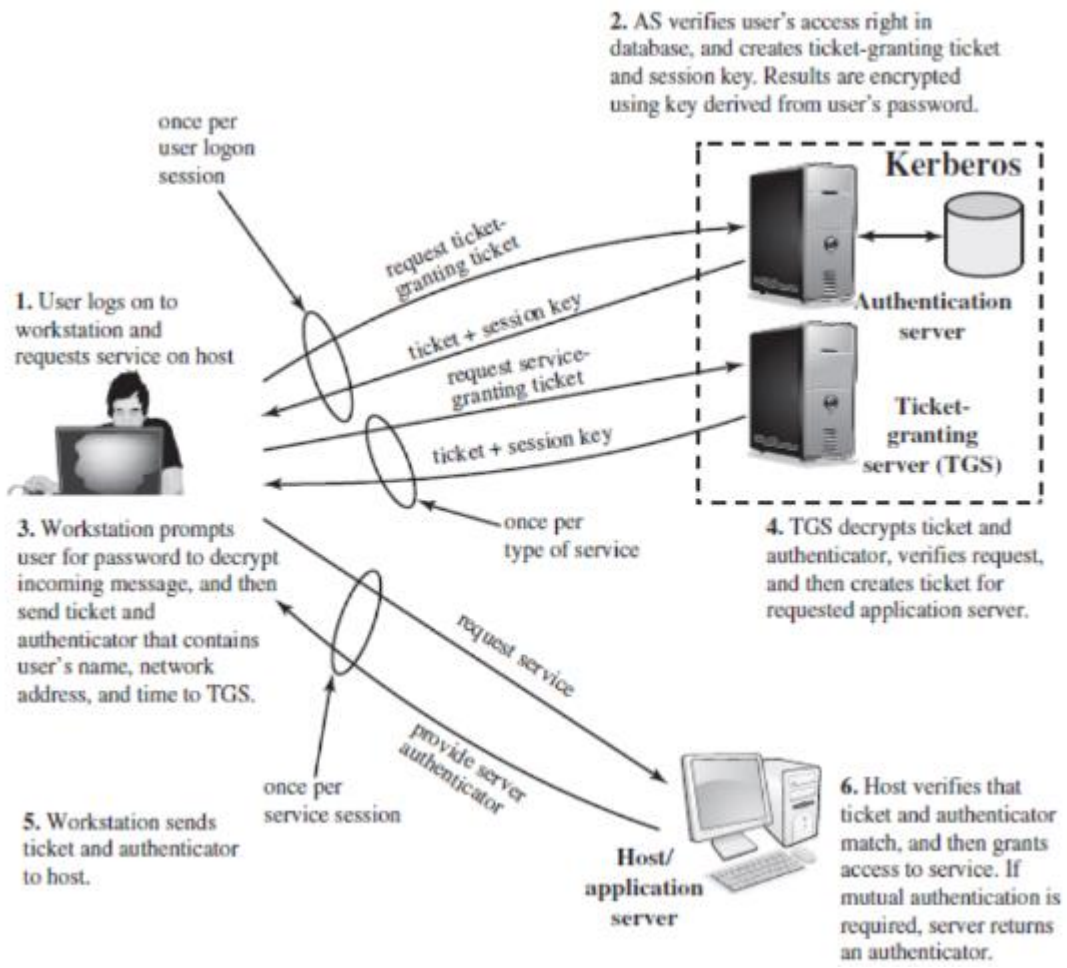
Let us look at the details of this scheme:

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 password, generates the key, and attempts to decrypt the incoming message. If the correct password is supplied, the ticket is successfully recovered.
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. 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. An opponent could eavesdrop on the network and capture a copy of the ticket-granting ticket and then wait for the legitimate user to log out.

Then the opponent could forge the legitimate user's network address and send the message of step (3) to the TGS. This would give the opponent unlimited access to Similarly, if an opponent captures a service-granting ticket and uses it before it expires, the opponent has access to the corresponding service. 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.

The second problem is that there may be a requirement for servers to authenticate themselves to users. Without such authentication, an opponent could sabotage the configuration so that messages to a server were directed to another location. The false server would then be in a position to act as a real server and capture any information from the user and deny the true service to the user.



PUBLIC KEY INFRASTRUCTURE

Introduction

- PKI is a set of roles, policies, and procedures needed to create, manage, distribute, use, store & revoke digital certificates and manage public-key encryption.
- The purpose of a PKI is to facilitate the secure electronic transfer of information for a range of network activities such as e-commerce, internet banking and confidential email.
- It is required for activities where simple passwords are an inadequate authentication method and more rigorous proof is required to confirm the identity of the parties involved in the communication and to validate the information being transferred

Design of PKI

- Public key cryptography is a cryptographic technique that enables entities to securely communicate on an insecure public network, and reliably verify the identity of an entity via digital signatures.
- A public key infrastructure (PKI) is a system for the creation, storage, and distribution of digital certificates which are used to verify that a particular public key belongs to a certain entity.
- The PKI creates digital certificates which map public keys to entities, securely stores these certificates in a central repository and revokes them if needed.

Components of PKI

- Certificate Authority:
  - A certificate authority or certification authority (CA) is an entity that issues digital certificates.
  - A digital certificate certifies the ownership of a public key by the named subject of the certificate.
  - This allows others (relying parties) to rely upon signatures or on assertions made about the private key that corresponds to the certified public key.
  - A CA acts as a trusted third party—trusted both by the subject (owner) of the certificate and by the party relying upon the certificate.
  - The format of these certificates is specified by the X.509 standard.

Components of PKI

- Registration Authority
  - Assures valid and correct registration is called a registration authority (RA).
  - An RA is responsible for accepting requests for digital certificates and authenticating the entity making the request.
  - In a Microsoft PKI, a registration authority is usually called a subordinate CA.

- Validation Authority
  - A validation authority (VA) is an entity that provides a service used to verify the validity of a digital certificate per the mechanisms described in the X.509 standard

Structure of X.509 Certificate