UNIT IV MESSAGE AUTHENTICATION AND INTEGRITY

UNIT IV MESSAGE AUTHENTICATION AND INTEGRITY Authentication requirement – Authentication function – MAC – Hash function – Security of hash function and MAC – SHA – Digital signature and authentication protocols – DSS- Entity Authentication: Biometrics, Passwords,Challenge Response protocols- Authentication applications - Kerberos, X.509

1. AUTHENTICATION REQUIREMENT

Communication across the network, the following attacks can be identified.

Disclosure — release of message contents to any person or process not possessing the appropriate cryptographic key.

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.

Masquerade — insertion of messages into the network from fraudulent source. This can be creation of message by the attacker using the authorized port.

Content modification – changes to the contents of a message, including insertion, deletion, transposition, and modification.

Sequence modification – any modification to a sequence of messages between parties,including insertion, deletion, and reordering.

Timing modification – delay or replay of messages.

- In a connection oriented application, an entire session or sequence of messages couldbe replay of some previous valid session, or individual messages in the sequence could be delayed or replayed.
- In a connectionless application, an individual message could be delayed or replayed.

Source repudiation – denial of transmission of message by source.

Destination repudiation – denial of receipt of message by destination.

1.1. AUTHENTICATION FUNCTION

Any message authentication or digital signature mechanism can be viewed as having fundamentally two levels.

At the lower level, there must be some sort of function that produces an authenticator, a valueto be used to authenticate a message.

At the higher-level, low-level function is then used as primitive in a higher-level authenticationprotocol that enables a receiver to verify the authenticity of a message.

The types of function that may be used to produce an authenticator are grouped into threeclasses.

Message Encryption – the ciphertext of the entire message serves as its authenticator.

Message Authentication Code (MAC) – a public function of the message and a secret key that produces a fixed length value that serves as the authenticator.

Hash Function – a public function that maps a message of any length into a fixed-length hash value, which serves as the authenticator.

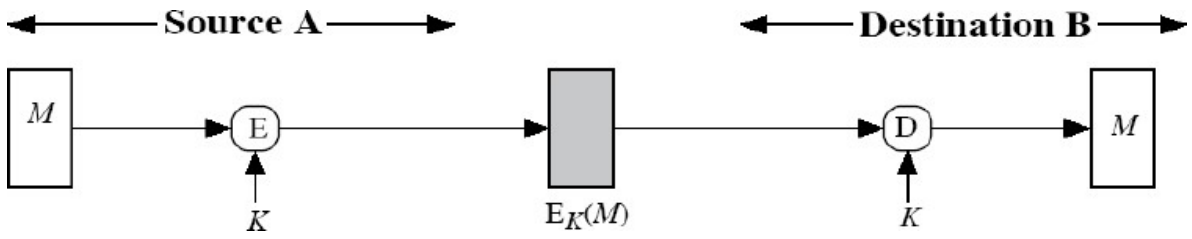
Message Encryption:

Message encryption Message encryption by itself can provide a measure of authentication.

The analysis differs from symmetric and public key encryption schemes.

(a) If symmetric encryption(fig.a) is used then:

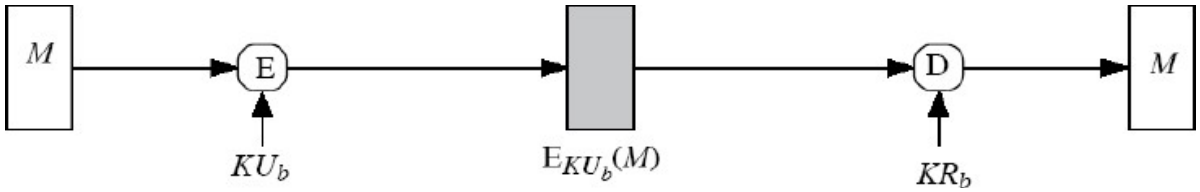
- A message m, transmitted from source A to destination B is encrypted using a secret key shared by A and B.
- Since only sender and receiver knows key used
- Receiver knows sender must have created it. Hence authentication is provided.
- Know content cannot have been altered. Hence confidentiality is also provided.
- If message has suitable structure, redundancy or a checksum to detect any changes
- Therefore Symmetric Encryption provides authentication and confidentiality.



(a).Symmetric key encryption confidentiality, authentication and signature

(b) If public-key encryption(Fig b) is used:

This method is the use of public key cryptography which provides confidentiality only. The sender A makes use of the public key of the receiver to encrypt the message. Here there is no authentication because any user can use B‟s public key to send a message and claim that only A has sent it.



(b) Public key encryption confidentiality

In this method(Fig c) to have only authentication, the message is encrypted with the sender‟s A‟s private key. The receiver B uses the sender‟s A‟s public key to decrypt the message. NowA cannot deny that it has not transmitted since it only knows its private key. This is called as authentication or Digital Signature. Hence the problem is the,

- Receiver cannot determine whether the packet decrypted contains some useful message or random bits.
- The problem is that anyone can decrypt the message when they know the public key of sender A.

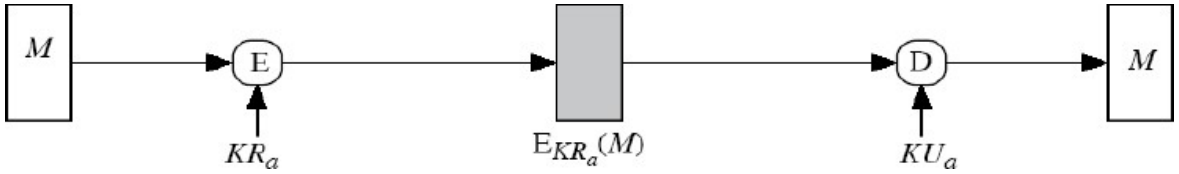


Figure (c) Public key encryption authentication and signature

This method (Fig d) provides authentication, confidentiality and digital signature. But the problem with this method is the complex public key cryptography algorithm should be applied twice during encryption and twice during decryption.

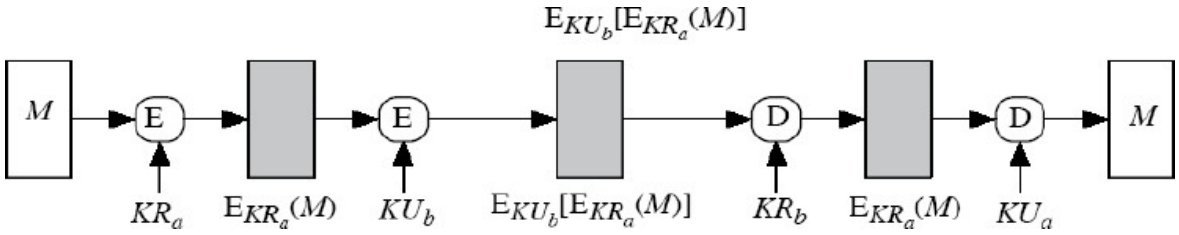


Figure (d) Public key encryption confidentiality, authentication and signature

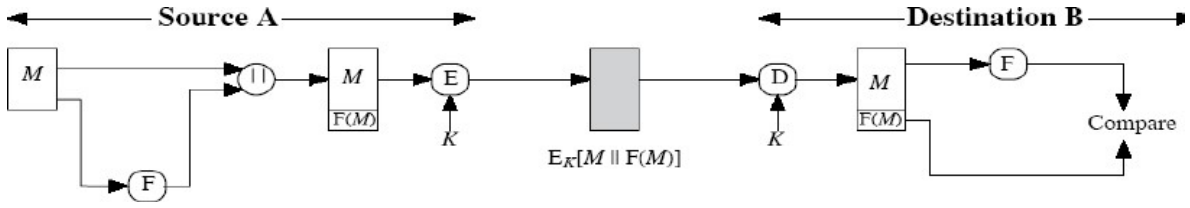
Suppose the message can be any arbitrary bit pattern, in that case, there is no way to determine automatically, at the destination whether an incoming message is the ciphertext of a legitimate message.

One solution to this problem is to force the plaintext to have some structure that is easily recognized but that cannot be replicated without recourse to the encryption function.

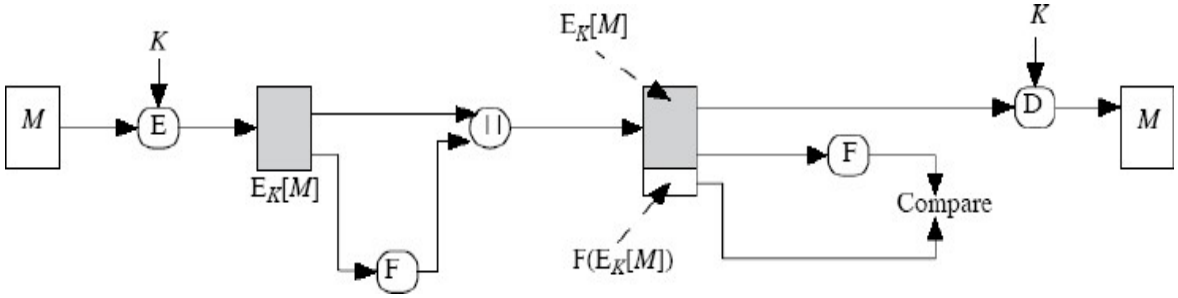
Append an error detecting code, also known as Frame Check Sequence (FCS) orchecksum to each message before encryption „A‟ prepares a plaintext message M and then provides this as input to a function F that produces an FCS. The FCS is appended to M and the entire block is then encrypted.

At the destination, B decrypts the incoming block and treats the result as a message with an appended FCS. B applies the same function F to attempt to reproduce the FCS.

If the calculated FCS is equal to the incoming FCS, then the message is considered authentic. *In the internal error control, the function F is applied to the plaintext, whereas in external error control, F is applied to the ciphertext (encrypted message fig e and d).*



(e) Internal error control



(f) External error

control4.2.MAC

An alternative authentication technique involves the use of secret key to generate a small fixed size block of data, known as cryptographic checksum or MAC that is appended tothe message.

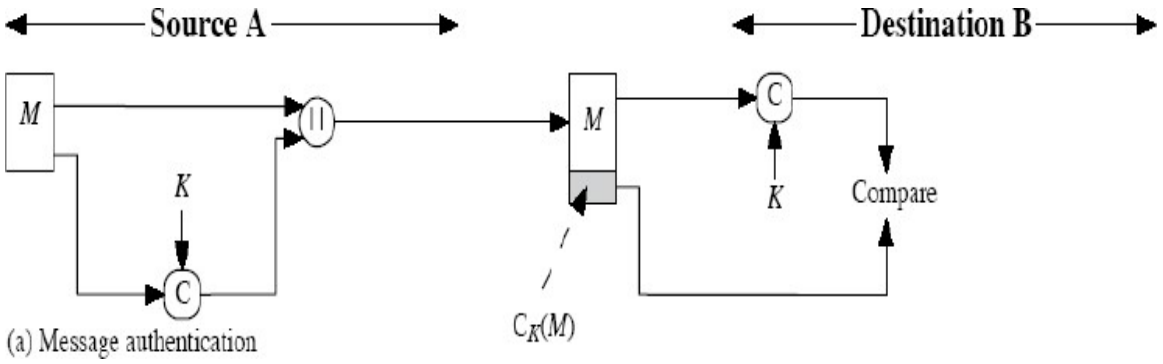
This technique assumes that two communication parties say A and B, share a common secret key „k". When A has to send a message to B, it calculates the MAC as a function of the message and the key.

$$MAC = C_K (M)$$

Where M – input message C – MAC function K – Shared secret key

The message plus MAC are transmitted to the intended recipient. The recipient performs the same calculation on the received message, using the shared secret key, to generate a new MAC.

The received MAC is compared to the calculated MAC. If it is equal, then the message is considered authentic(Fig g and h). A MAC function is similar to encryption. One difference is that MAC algorithm need not be reversible, as it must for decryption. In general, the MAC function is a many-to-one function.



(g) Message Authentication

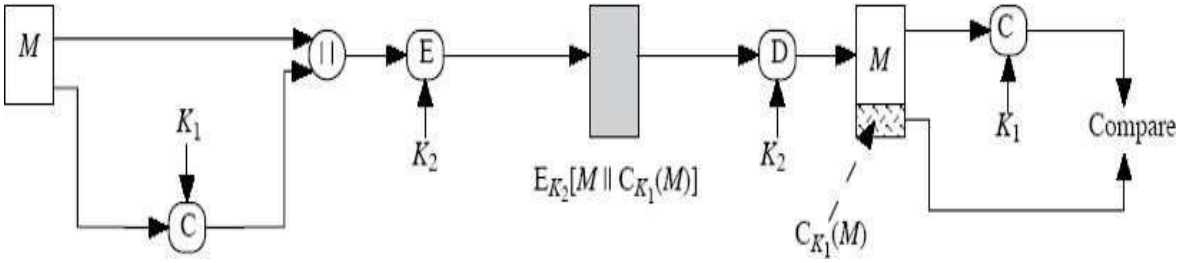


Figure (h) Message authentication and confidentiality, authentication tied to plain text

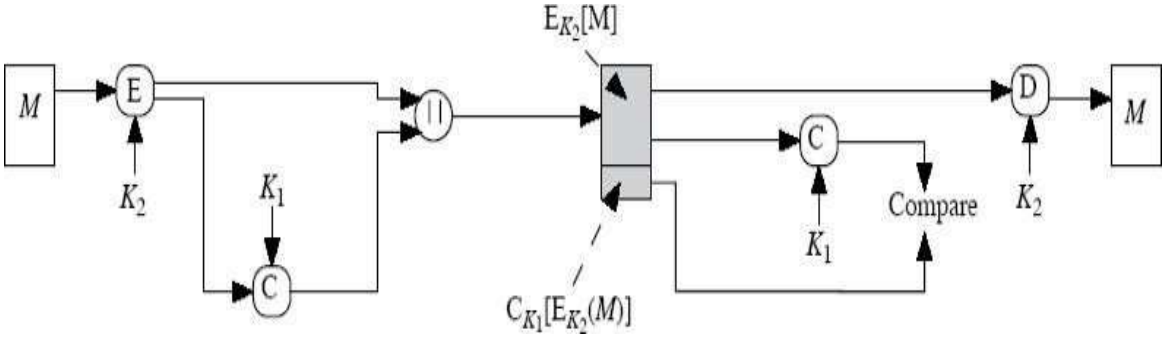


Figure (i) Message authentication and confidentiality, authentication tied to ciphertext

Requirements for MAC:

When an entire message is encrypted for confidentiality, using either symmetric or asymmetric encryption, the security of the scheme generally depends on the bit length of the key.

Barring some weakness in the algorithm, the opponent must resort to a brute-force attack using all possible keys. On average, such an attack will require 2(k-1) attempts for a k-bit key.

If confidentiality is not employed, the opponent has access to plaintext messages and their associated MACs. Suppose k > n; that is, suppose that the key size is greater than the

MAC size. Then, given a known M_1 and MAC_1, with $MAC_1 = C_K(M_1)$, the cryptanalyst can perform $MAC_i = CK_i(M1)$ for all possible key values Ki.

At least one key is guaranteed to produce a match of $MAC_i = MAC_1$.

Note that a total of 2^k MACs will be produced, but there are only $2^n < 2^k$ different MAC values. Thus, a number of keys will produce the correct MAC and the opponent has no way of knowing the correct key. On average, a total of $2^k/2^n = 2^{(k-n)}$ keys will produce a match. Thus, the opponent must iterate the attack:

Round 1

Given: M_1, $MAC_1 = C_K(M_1)$
Compute $MAC_i = C_{Ki}(M_1)$ for all 2^k
keysNumber of matches $\approx 2^{(k-n)}$

Round 2

Given: M_2, $MAC_2 = C_K(M_2)$
Compute $MAC_i = C_{Ki}(M_2)$ for the $2^{(k-n)}$ keys resulting from
Round 1Number of matches $\approx 2^{(k-2xn)}$ and so on

Consider the following MAC algorithm. Let $M = (X_1||X_2||...||X_m)$ be a message that is treated as a concatenation of 64-bit blocks X_i. Then define

$$\Delta(M) = X_1 + X_2 \ldots X_m$$
$$C_k(M) = E_k(\Delta(M))$$

Thus, the key length is 56 bits and the MAC length is 64 bits. If an opponent observes {M||C(K, M)}, a brute-force attempt to determine K will require at least 2^{56} encryptions.

But the opponent can attack the system by replacing X1 through X_{m-1} with any desired values Y_1 through Y_{m-1} and replacing X_m with Y_m where Y_m is calculated as follows:

$$Y_m = Y_1 + Y_2 \ldots\ldots Y_{m1} + \Delta(M)$$

The opponent can now concatenate the new message, which consists of Y_1 through Y_m, with the original MAC to form a message that will be accepted as authentic by the receiver. With this tactic, any message of length 64 $X_{(m-1)}$ bits can be fraudulently inserted.

Then the MAC function should satisfy the following requirements: The MAC function should have the following properties:

- If an opponent observes M and $C_K(M)$, it should be computationally infeasible for the opponent to construct a message M‟ such that $C_K(M') = C_K(M)$
- $C_K(M)$ should be uniformly distributed in the sense that for randomly chosen messages, M and M‟, the probability that $C_K(M) = C_K(M')$ is 2^{-n} where n is the number of bits in the MAC.

- Let M‟ be equal to some known transformation on M. i.e., M‟ = f(M).

MAC based on DES

One of the most widely used MACs, referred to as Data Authentication Algorithm (DAA) is based on DES.

The algorithm(Fig 2) can be defined as using cipher block chaining (CBC) mode of operation of DES with an initialization vector of zero. The data to be authenticated are grouped into contiguous 64-bit blocks: $D_1, D_2 \ldots D_n$. if necessary, the final block is padded on the right with zeros to form a full 64-bit block. Using the DES encryption algorithm and a secret key, a data authentication code (DAC) is calculated as follows:

$O_1 = E_K(D_1)$
$O_2 = E_K(D_2 +$
$O_1)O_3 = E_K(D_3 +$
$O_2)$
. . . . . .
$O_N = E_K(D_N + O_{N-1})$

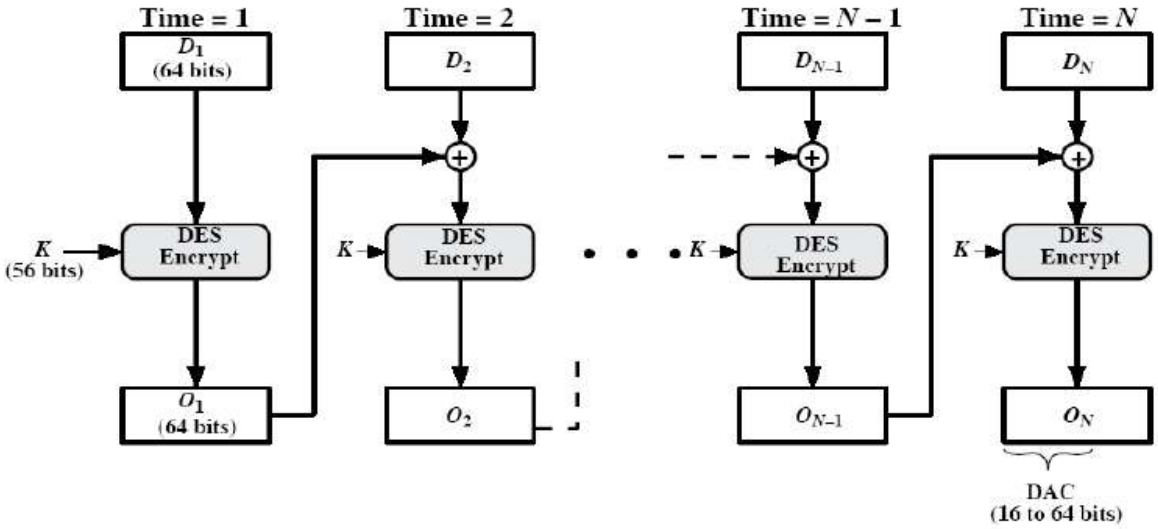


Figure.2 Data Authentication

Algorithm4.3.HASH FUNCTION

A variation on the message authentication code is the one way hash function. As with MAC, a hash function accepts a variable size message M as input and produces a fixed-size output, referred to as hash code H(M).

Unlike a MAC, a hash code does not use a key but is a function only of the input message. The hash code is also referred to as a message digest or hash value. There are varieties of ways in which a hash code can be used to provide message authentication, as follows:

In Fig (a) The message plus the hash code is encrypted using symmetric encryption. This is identical to that of internal error control strategy. Because encryption is applied to the entire message plus the hash code, confidentiality is also provided.

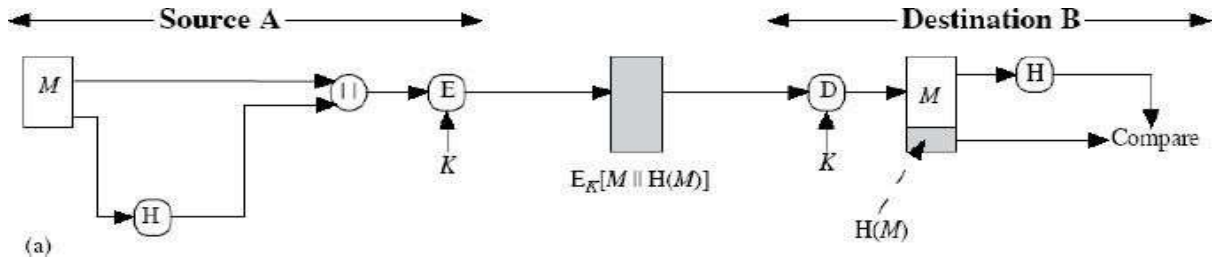
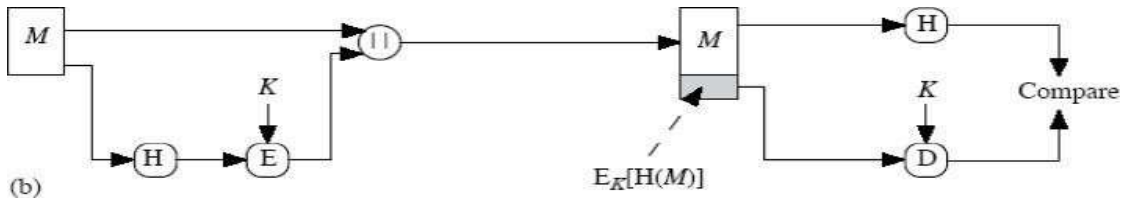


Figure (a) Hash Function

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



In Fig (c) Only the hash code is encrypted, using the public key encryption and using the sender"s private key. It provides authentication plus the digital signature.

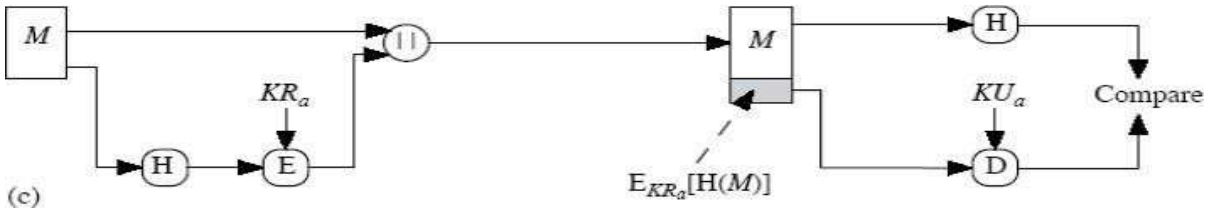
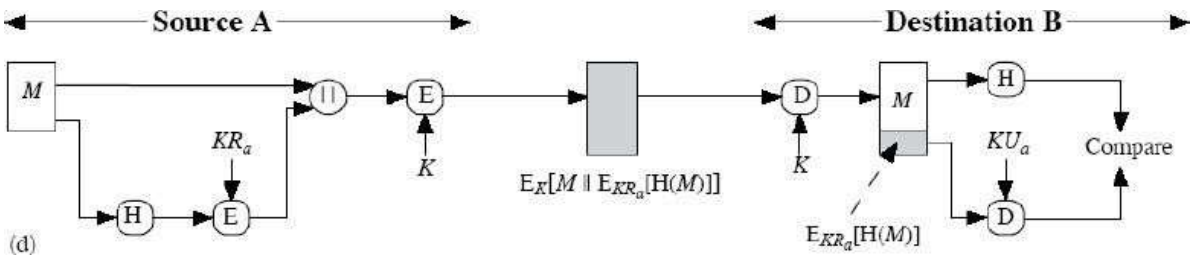
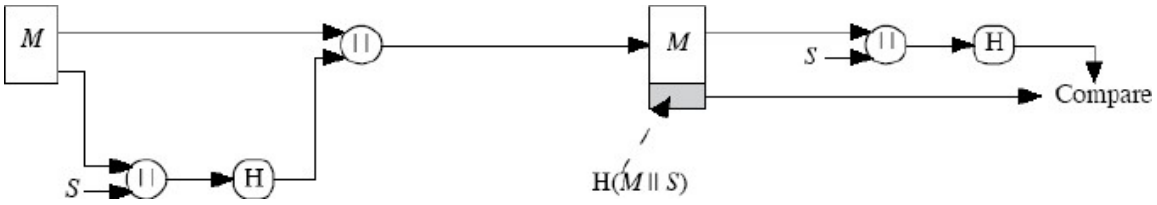


Figure (b & c) Basic use of Hash Function

In Fig (d) If confidentiality as well as digital signature is desired, then the message plus the public key encrypted hash code can be encrypted using a symmetric secret key.



In Fig (e) This technique uses a hash function, but no encryption for message authentication. This technique assumes that the two communicating parties share a common secret value „S". The source computes the hash value over the concatenation of M and S and appends the resulting hash value to M.



In Fig(f) Confidentiality can be added to the previous approach by encrypting the entire message plus the hash code.

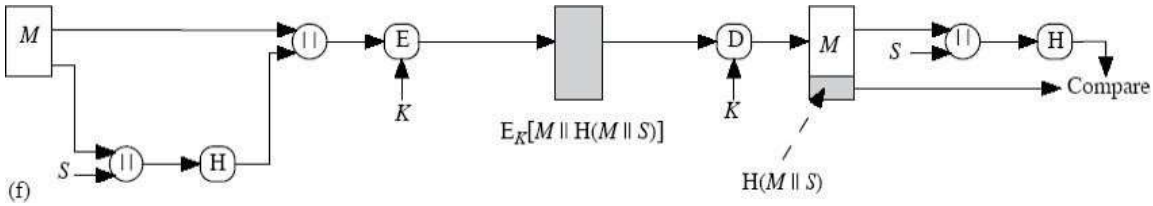


Figure (d,e & f) Basic use of Hash Function

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

$$h = H(M)$$

where M is a variable-length message and H(M) is the fixed-length hash value.

The hash value is appended to the message at the source at a time when the message is assumed or known to be correct. The receiver authenticates that message by recomputing the hash value.

Requirements for a Hash Function

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 softwareimplementations practical.

4. For any given value h, it is computationally infeasible to find x such that H(x) = h. This issometimes referred to in the literature as the one-way property.

5. For any given block x, it is computationally infeasible to find y x such that H(y) = H(x). This issometimes referred to as weak collision resistance.

6. It is computationally infeasible to find any pair (x, y) such that H(x) = H(y). This is sometimesreferred to as strong collision resistance.
The first three properties are requirements for the practical application of a hash function to message authentication.

The fourth property, the one-way property, states that it is easy to generate a code givena message but virtually impossible to generate a message given a code.

The fifth property guarantees that an alternative message hashing to the same value asa given message cannot be found. This prevents forgery when an encrypted hash code is used.

The sixth property refers to how resistant the hash function is to a type of attack known as the birthday attack.

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 follows: $C_i = b_{i1} + b_{i1} \ldots + b_{im}$

where

$C_i = i^{th}$ bit of the hash code, $1 \leq i \leq n$
m = number of n-bit blocks in the
inputb_{ij}= ith bit in jth block

Procedure:

1. Initially set the n-bit hash value to zero.

2. Process each successive n-bit block of data as follows:

   a. Rotate the current hash value to the left by one bit.
   b. XOR the block into the hash value.

Birthday Attacks

Suppose that a 64-bit hash code is used. One might think that this is quite secure. For example, if an encrypted hash code C is transmitted with the corresponding unencrypted message M, then an opponent would need to find an M' such that H(M') = H(M) to substitute another message and fool the receiver.

On average, the opponent would have to try about 2^{63} messages to find one thatmatches the hash code of the intercepted message.

However, a different sort of attack is possible, based on the birthday paradox. The source, A, is prepared to "sign" a message by appending the appropriate m-bit hash code and encrypting that hash code with A's private key.

1. The opponent generates $2^{m/2}$ variations on the message, all of which convey essentially the same meaning. (Fraudulent message)

2. The two sets of messages are compared to find a pair of messages that produces the same hash code. The probability of success, by the birthday paradox, is greater than 0.5. If no matchis found, additional valid and fraudulent messages are generated until a match is made.

3. The opponent offers the valid variation to A for signature. This signature can then be attached to the fraudulent variation for transmission to the intended recipient. Because the two variations have the same hash code, they will produce the same signature; the opponent is assured of success even though the encryption key is not known.

Thus, if a 64-bit hash code is used, the level of effort required is only on the order of 2^{32}

MEET-IN-THE-MIDDLE ATTACK.

Divide a message M into fixed-size blocks $M_1, M_2, ..., M_N$ and use a symmetric encryption system such as DES to compute the hash code G as follows:

H_o = initial value
$H_i = E_{Mi}[H_{i-1}]$
$G = H_N$

This is similar to the CBC technique, but in this case there is no secret key. As with any hash code, this scheme is subject to the birthday attack, and if the encryption algorithm is DES and only a 64-bit hash code is produced, then the system is vulnerable.

Furthermore, another version of the birthday attack can be used even if the opponent has access to only one message and its valid signature and cannot obtain multiple signings.

Here is the scenario; we assume that the opponent intercepts a message with a signature in the form of an encrypted hash code and that the unencrypted hash code is m bits long:

1. Calculate the unencrypted hash code G.

2. Construct any desired message in the form $Q_1, Q_2, ..., Q_{N2}$.

3. Compute for $H_i = EQ_i [H_{i-1}]$ for $1 \leq i \leq (N-2)$.

4. Generate $2^{m/2}$ random blocks; for each block X, compute $E_X[H_{N-2}.]$ Generate an additional $2^{m/2}$ random blocks; for each block Y, compute $D_Y[G]$, where D is the decryption function corresponding to E.

5. Based on the birthday paradox, with high probability there will be an X and Y such that $E_X [H_{N-2}] = D_Y[G]$.

6. Form the message $Q_1, Q_2, ..., Q_{N-2}$, X, Y. This message has the hash code G and therefore can be used with the intercepted encrypted signature.

4.4. SECURITY OF HASH FUNCTION AND MAC

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

Brute-Force Attacks

The nature of brute-force attacks differs somewhat for hash functions and MACs.

Hash Functions

The strength of a hash function against brute-force attacks depends solely on the length of thehash code produced by the algorithm.

Requirements of Hash Function:

One-way: For any given code h, it is computationally infeasible to find x such that H(x) = h.

Weak collision resistance: For any given block x, it is computationally infeasible to find y x with H(y) = H(x).

Strong collision resistance: It is computationally infeasible to find any pair (x, y) such that H(x) = H(y).

For a hash code of length n, the level of effort required, as we have seen is proportional to the following:

One way	2^n
Weak collision resistance	2^n
Strong collision resistance	$2^{n/2}$

Message Authentication Codes

A brute-force attack on a MAC is a more difficult undertaking because it requires known message-MAC pairs.. To attack a hash code, we can proceed in the following way. Given a fixed message x with n-bit hash code h = H(x), a brute-force method of finding a collision is to pick a random bit string y and check if H(y) = H(x). The attacker can do this repeatedly offline.

To proceed, we need to state the desired security property of a MAC algorithm, which can be expressed as follows:

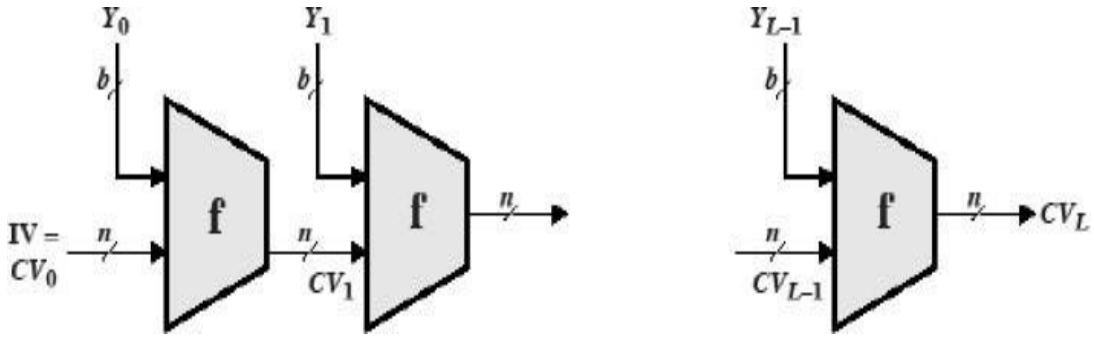
Cryptanalysis

As with encryption algorithms, cryptanalytic attacks on hash functions and MAC algorithms seek to exploit some property of the algorithm to perform some attack other than an exhaustive search.

Hash Functions

The hash function takes an input message and partitions it into L fixed-sized blocks of b bits each. If necessary, the final block is padded to b bits. The final block also includes the value of the total length of the input to the hash function(Fig 3.4). The inclusion of the length makes the job of the opponent more difficult.

Either the opponent must find two messages of equal length that hash to the same value or two messages of differing lengths that, together with their length values, hash to the same value.



IV=Initial Value
Y_i = ith input block
n=Length of Hash code

CV=Changing Variable
L=number of input blocks
b=Length of input block

General structure of secure hash code

The hash algorithm involves repeated use of a compression function,f, that takes twoinputs (an n-bit input from the previous step, called the chaining variable, and a b-bit block) and produces an n-bit output.

At the start of hashing, the chaining variable has an initial value that is specified as part of the algorithm. The final value of the chaining variable is the hash value. Often b > n; hence the term compression.

The hash function can be summarized as follows:

CV_0 = IV = initial n-bit
valueCV_I = f(CV_{i-1}, Yi_{-1}) 1
≤i ≤L H(M) = CV_L
Where the input to the hash function is a message M consisting of the blocks Y_o, Y_1,..., Y_{L-1}. The structure can be used to produce a secure hash function to operate on a message of any length.

Message Authentication Codes :

There is much more variety in the structure of MACs than in hash functions, so it is difficult to generalize about the cryptanalysis of MACs.

4.5. SHA

The algorithm takes as input a message with a maximum length of less than bits and produces as output a 512-bit message digest. The input is processed in 1024-bit blocks. Figure **3.1** depicts the overall processing of a message to produce a digest.

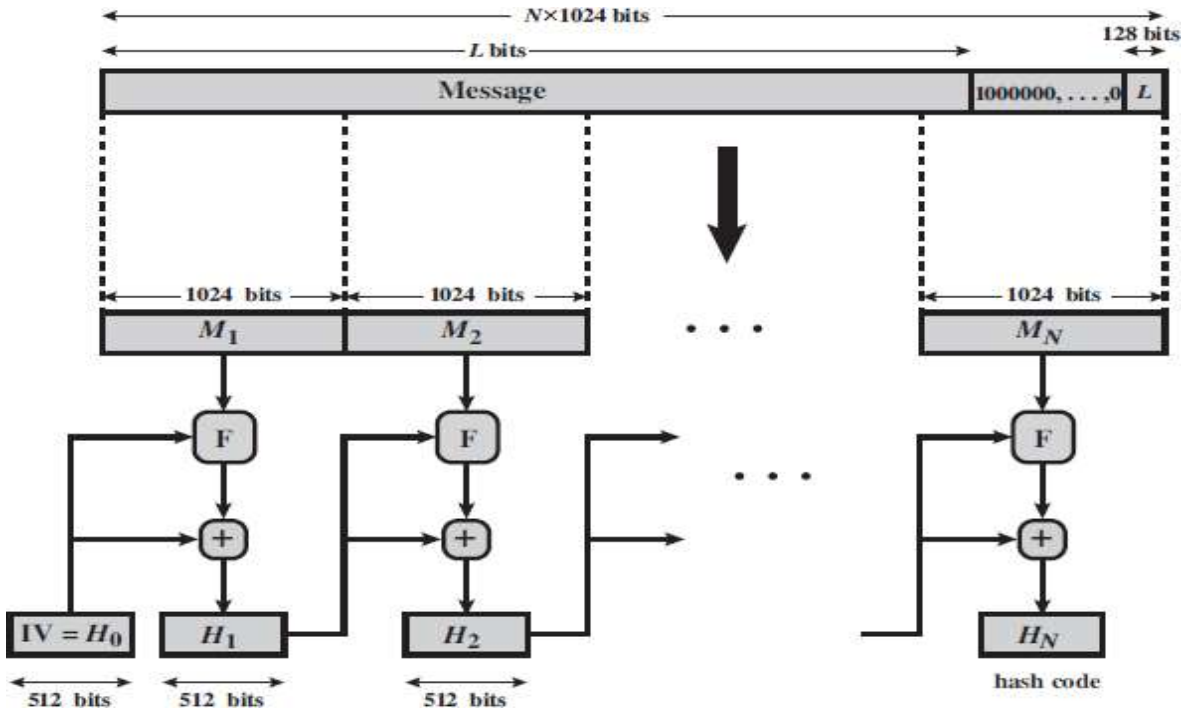


Fig .Message Digest Generation Using SHA-512

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. 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 3,8 , the expanded message is represented as the sequence of 1024-bit blocks M_1, M_2,.....M_N , so that the total length of the expanded message is NX1024 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 =
1F83D9ABFB41BD6Bd = A54FF53A5F1D36F1 h =
5BE0CD19137E2179

These values are stored in big-endian format, which is the most significant byte of a word in the low-address (leftmost) byte position. These words were obtained by taking the first sixty-four bits of the fractional parts of the square roots of the first eight prime numbers.

Step 4: Process message in 1024-bit (128-word) blocks.

The heart of the algorithm (Fig 3.9) is a module that consists of 80 rounds; Each round takes as input the 512-bit buffer value, abcdefgh, and updates the contents of the buffer. At input to the first round, the buffer has the value of the intermediate hash value,H_{i-1}

Each round makes use of a 64-bit value W_t, derived from the current 1024-bit block (M_i) being processed .These values are derived using a message schedule described subsequently.

Each round also makes use of an additive constant k_t, where 0<=t<=79 indicates one of the 80 rounds.

The output of the eightieth round is added to the input to the first round (H_i-1) to produce H_i.The addition is done independently for each of the eight words in the buffer with each of the corresponding words in Hi-1, using addition modulo 264 .

Step 5: Output.

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

The behavior of SHA-1 is summarized as follows:

$H_0 = IV$

$H_i = SUM_{64} (H_{i-1},$

$ABCDEFGH_i)MD = H_N$

Where

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

$ABCDE_q$ = the output of the last round of processing of the *i*th message block

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

SUM_{32} = Addition modulo 2^{32} performed separately on each word of the pair of inputs

MD = final message digest value

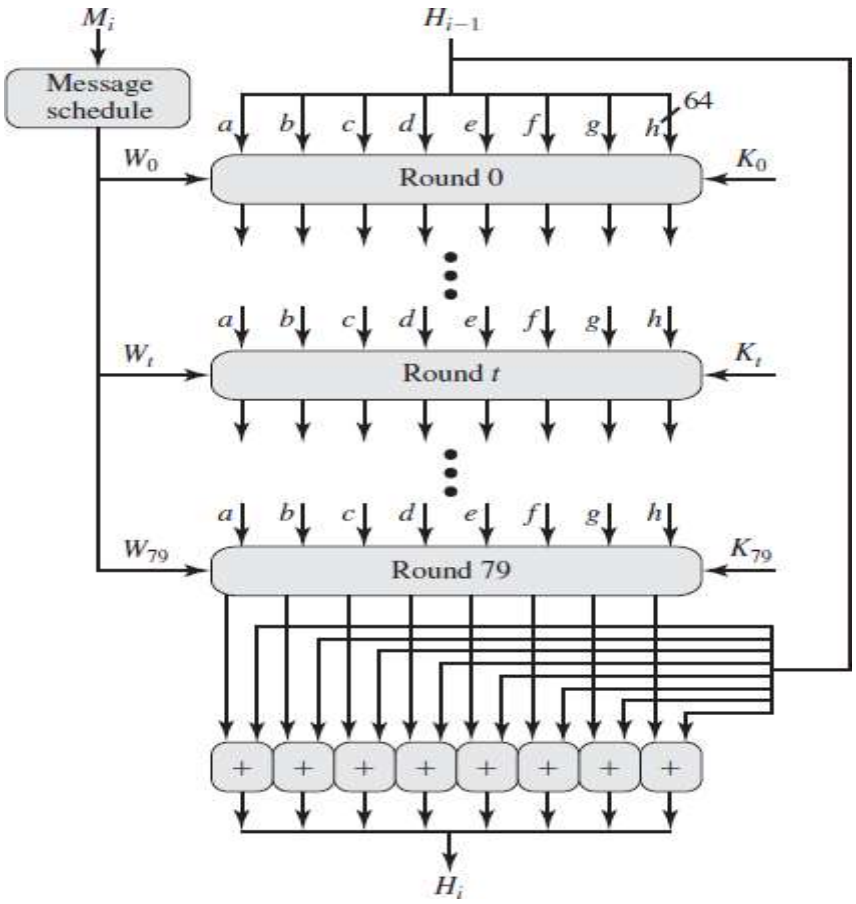


Figure. SHA-512 Processing of a Single 1024-Bit Block

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. Each round is defined by the following set of equations:

$$T_1 = h + Ch(e,f,g) + (\Sigma_1^{512} e) + Wt + Kt)$$

$$T_2 = (\Sigma_0^{512} a) + Maj(a,b,c)$$

$h = g$ $g = f$ $f = e$ $e = d + T_1$ $d = c$
$c = b$ $b = a$ $a = T_1 + T_2$

Where

T = Step number; $0 \leq t \leq 79$

Ch(e,f,g) = (a AND f) $\oplus$ (NOT e AND

g)

The conditional function: If e then f else g (Fig 3.10)

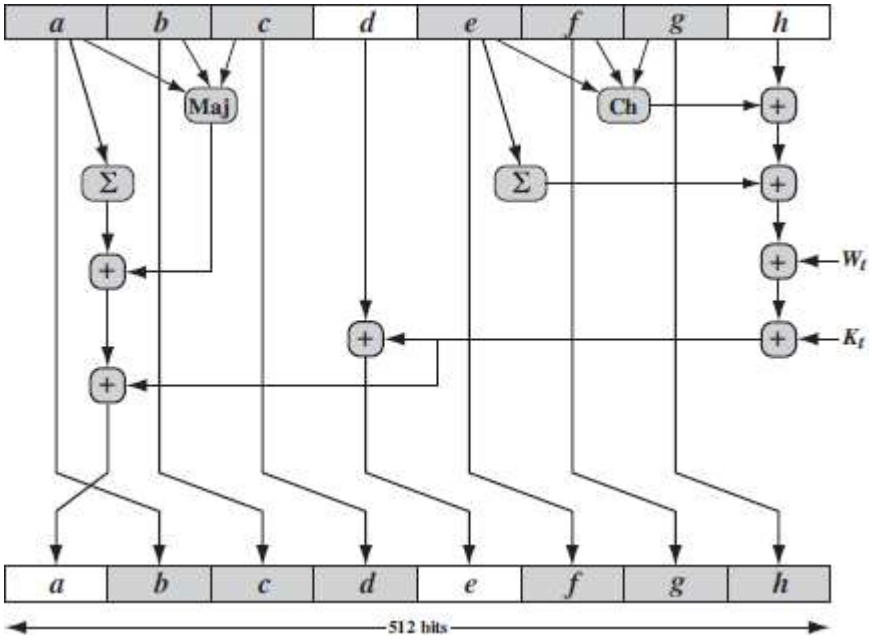


Fig .Elementary SHA Operation(single step)

The functions can be summarized as follows:

Steps	Function Name	Function Value
0≤t≤9	f1 = f(t,B,C,D)	(B Λ C) V (B! Λ D)
20≤t≤39	f2 = f(t,B,C,D)	B ⊕C ⊕D
40≤t≤59	f3 = f(t,B,C,D)	(B Λ C) V (B Λ D) V (C Λ D)
60≤t≤79	f4 = f(t,B,C,D)	B ⊕C ⊕D

The logical operators AND,OR,NOT,XOR, are represented by the symbols Λ V !

⊕ Only three different functions are used.

For, 0≤ t ≤ 19 the function is the conditional function.

For 20 ≤ t ≤ 39 and 60 ≤ t ≤ 79 the function produces a parity bit.

For A0 ≤ t ≤ 59 the function is true if two or three of the argument are true.

The following diagram illustrates how the 32bit word values wt are derived from the 512 bit message.

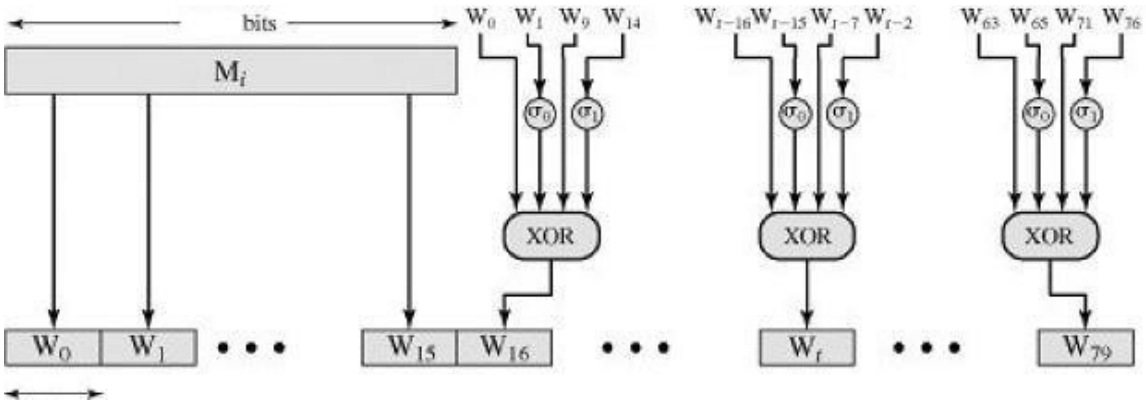


Figure.Creation of 80-word Input Sequence for SHA-512 Processing of Single Block

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

$$w_t = S^{\text{‘‘}}(w_{t-16} + w_{t-14} + w_{t-8} + w_{t-3})$$

Thus in the first 16 steps of processing the values 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 shiftby one bit of the XOR of four of the processing values of w_t.

Both MD5 and RIPEMD-160 uses one of the 16 words of a message block directly as input to each step function only the order of the word is permuted from round to round.

SHA-1 expands the 16 block words to 80 words for use in the compression function.

Comparison of SHA-1 and MD5

Because both are derived from MD4,SHA-1 and MD5 are similar to one another.

1. Security against brute – force attacks:

The most important difference is that the SHA-1 digest is 32bits longer than the MD5 digest.

Using a brute force technique the difficulty of producing any message having a given message digest is on the order of 2^{128} operations for MD5 and 2^{160} for SHA-1.

Using brute force technique the difficulty of producing two messages having the same message digest is on the order of 2^{64} operations for MD5 and 2^{80} for SHA-1.Thus SHA-1 is considerably stronger against brute force attacks.

2. Security against cryptanalysis:

MD5 is vulnerable to cryptanalytic attacks.

SHA-1 is not vulnerable to such attacks.

3. Speed:

Both algorithms rely on addition module 2^{32}, so both do well on 32 bit architecture

SHA-1 involves more steps (80) and must process a 160 bit buffer compared to MD5‟s 128 bit buffer.

Thus SHA-1 should execute more slowly than MD5 on the same hardware.

4. Simplicity and compactness:

Both algorithms are simple to describe and simple to implement and do not require large programs or substitution tables.

5. Little endian versus big endian architecture:

MD5 uses a little endian scheme and SHA-1 uses a big endian scheme.

HMAC

HMAC Design Objectives:

- To use hash functions that perform well in software and for which code is freely and widely available.
- To allow for easy replacement 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 significantdegradation.
- 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. In this way, the bulk of the HMAC code is prepackaged and ready to use without modification.

Second, if it is ever desired to replace a given hash function in an HMAC implementation, remove the existing hash function module and drop in the new module.

HMAC Algorithm:

Definition of terms used in algorithm(Fig 3.12).

H = embedded hash function (e.g., MD5, SHA-1, RIPEMD-160)

IV = initial value input to hash function

M = message input to HMAC

Y_i = *i* th block of *M*, $0 \leq i \leq (L - 1)$

L =number of blocks in *M*

b = number of bits in a block

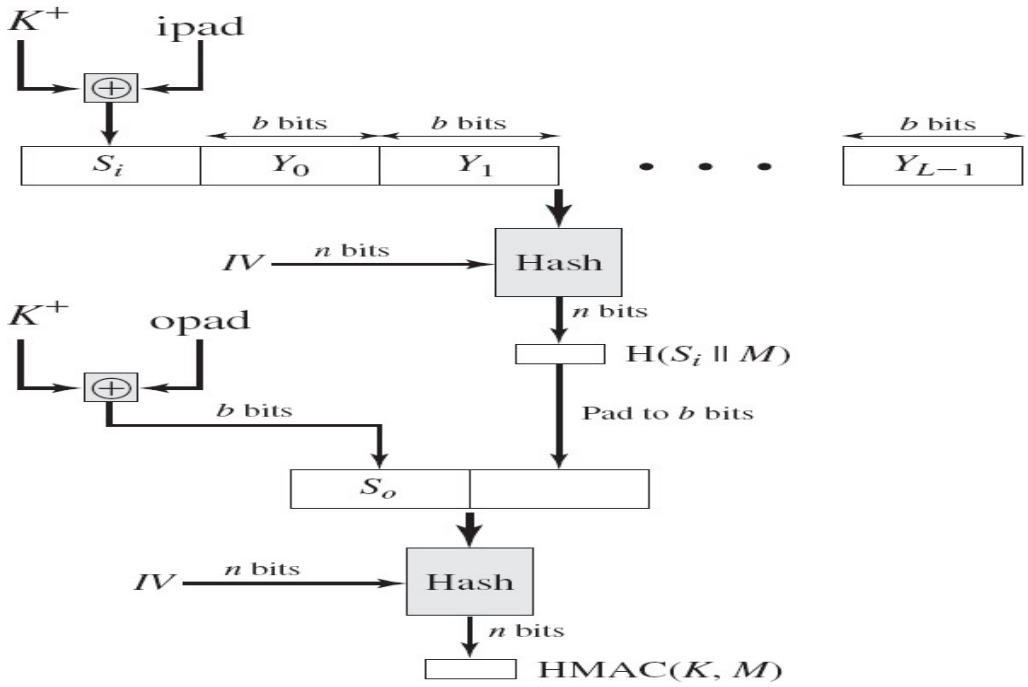


Figure .HMAC Structure

n = length of hash code produced by embedded hash function

K = secret key; recommended length is *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

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

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

Then HMAC can be expressed as

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

We can describe the algorithm as follows:

1. Append zeros to the left end of to create a -bit string (e.g., if is of length 160 bits and , then will be appended with 44 zeroes).

2. XOR (bitwise exclusive-OR) with ipad to produce the b -bit block S_i .

3. Append M to S_i.

4. Apply H to the stream generated in step 3.

5. XOR K^+ with opad to produce the b-bit block S_0.

6. Append the hash result from step 4 to S_0.

7. Apply H to the stream generated in step 6 and output the result.

A more efficient implementation is possible. Two quantities are precomputed:

$$f(IV, (K^+ \oplus \text{ipad}))$$

$$f(IV, (K^+ \oplus \text{opad}))$$

In effect, the precomputed quantities substitute for the initial value (IV) in the hash function. With this implementation (Fig 3.13), only one additional instance of the compression function is added to the processing normally produced by the hash function.

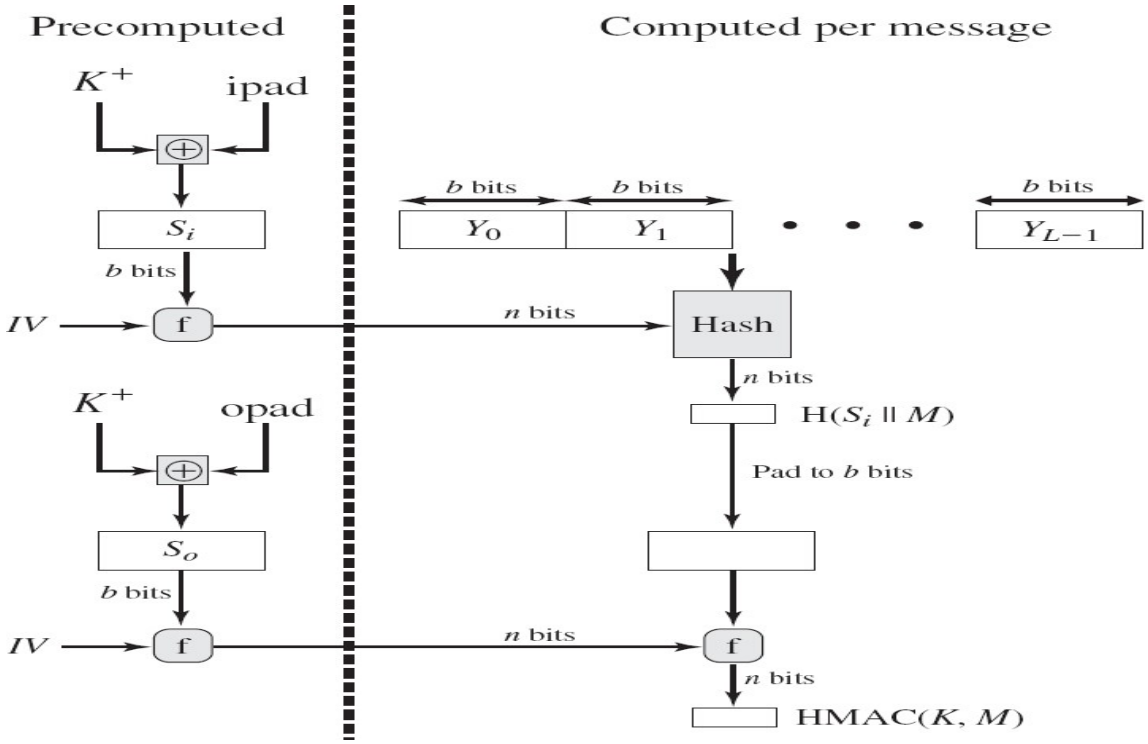


Figure .Efficient Implementation of HMAC

Security of HMAC

The security of a MAC function is generally expressed in terms of the probability of successful forgery with a given amount of time spent by the forger and a given number of message–tag pairs created with the same key.

In essence, it is proved in that for a given level of effort (time, message–tag pairs) on messages generated by a legitimate user and seen by the attacker, 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 that is random, secret, and unknown to the attacker.

2. The attacker finds collisions in the hash function even when theIV 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 thehash function is replaced by a secret, random value of 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 2^n, or a birthday attack.

In the second attack, the attacker is looking for two messages M & M‟and that produce the same hash: H(*M*) = H(*M*‟) .

CMAC

Only messages of one fixed length of *mn* bits are processed, where *n* is the cipher block size and *m* is a fixed positive integer. a simple example, notice that given the CBC MAC of a one- block message *X*, say *T* = MAC(*K*, *X*), the adversary immediately knows the CBC MAC for the two block message $X \| (X \oplus T)$since this is once again *T*.

Black and Rogaway [BLAC00] demonstrated that this limitation could be overcome using three keys: one key *K* of length *k* to be used at each step of the cipher block chaining and two keys of length *b*, where *b* is the cipher block length.

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

First, let us define the operation of CMAC when the message is an integer multiple *n* of the cipher block length *b*. For AES, *b* = 128, and for triple DES, *b* = 64. The message is divided into*n* blocks (*M*1, *M*2,c, *Mn*). The algorithm makes use of a *k*-bit encryption key *K* and a *b*-bit constant, *K*1. For AES, the key size *k* is 128, 192, or 256 bits; for triple DES, the key size is 112 or 168 bits. CMAC is calculated as follows

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

where

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

The CMAC operation(Fig 3.14) then proceeds as before, except that a different b-bit key $K2$ is used instead of $K1$.

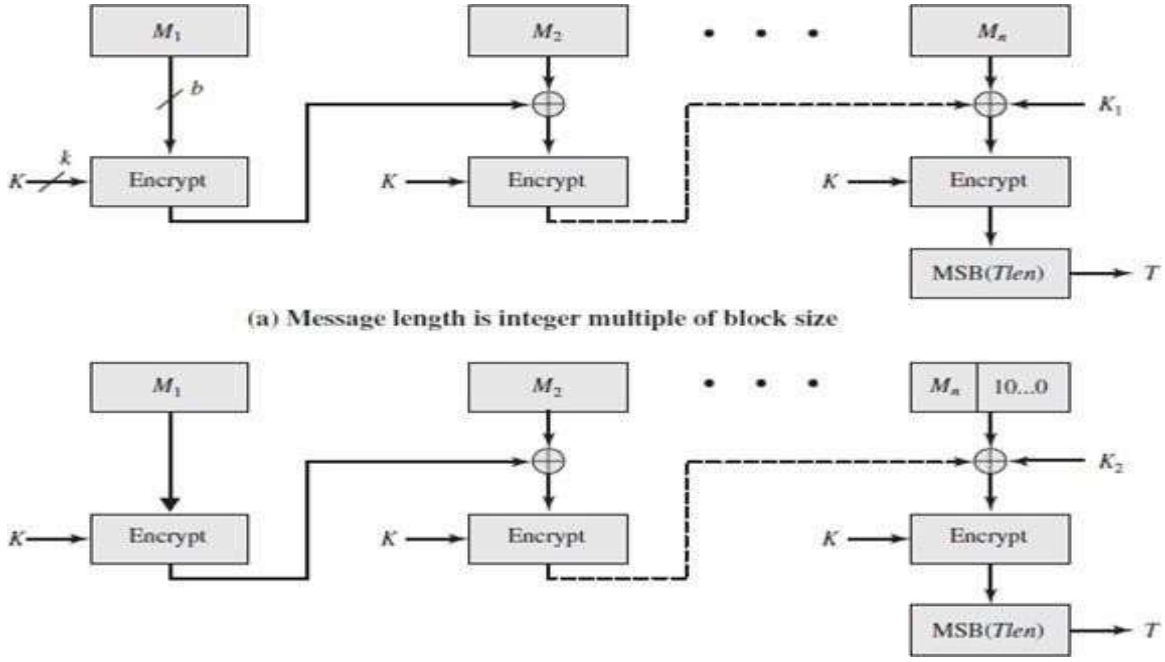


Fig .Cipher-based Message Authentication Code

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

$$L = \mathrm{E}(K, 0^b)$$
$$K_1 = L \cdot x$$
$$K_2 = L \cdot x^2 = (L \cdot x) \cdot x$$

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.

4.6. DIGITAL SIGNATURE AND AUTHENTICATION PROTOCOLS

Digital Signature Requirements

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

Disputes created by message authentication are:

- Creation of fraud message.

- Deny the sending of message

For example, suppose that John sends an authenticated message to Mary, the following disputes that could arise:

1 Mary may forge a different message and claim that it came from John. Mary would simply have to create a message and append an authentication code using the key that John and Mary share.

2. John can deny sending the message. Because it is possible for Mary to forge a message, there is no way to prove that John did in fact send the message.

Properties of digital signature :

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

Requirements for a digital signature:

- The signature must be a bit pattern that depends on the message being signed.
- 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 of the digital signature in storage.

Direct Digital Signature

The term **direct digital signature** refers to a digital signature scheme that involves only the communicating parties (source, destination). It is assumed that the destination knows the public key of the source.

Confidentiality can be provided by encrypting the entire message plus signature with a shared secret key (symmetric encryption). Note that it is important to perform the signature function first and then an outer confidentiality function. In case of dispute, some third party must view the message and its signature.

If the signature is calculated on an encrypted message, then the third party also needs access to the decryption key to read the original message. However, if the signature is the inner operation, then the recipient can store the plaintext message and its signature for later use in dispute resolution.

The validity of the scheme just described depends on the security of the sender‟s private key.

Weakness of Direct Digital Signature:

- If a sender later wishes to deny sending a particular message, the sender can claim that the private key was lost or stolen and that someone else forged his or her signature.

- Another threat is that some private key might actually be stolen from X at time T. The opponent can then send a message signed with X‟s signature and stamped with a time before or equal to T.

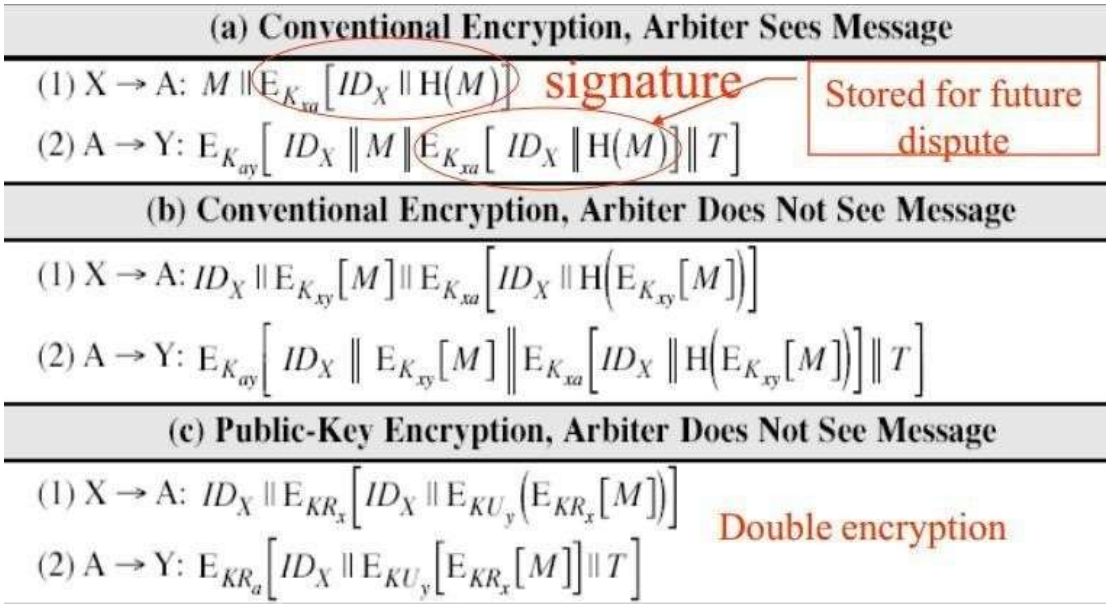
Arbitrated Digital Signatures

The problem associated with the Direct digital signature can be overcome by using arbitrated schemes.

In the arbitrated scheme, the entire signed message from the sender goes to the arbiter A. The arbiter subjects the message and signature to a number of tests to check the origin and control. The date and time is attached to the message. This indicates that the digital signature has been verified and is satisfied. The message is then transmitted to the receiver.

Requirement of the arbiter:

- As the arbiter plays a sensitive and crucial role, it should be a trusted third party.



Notation:X = Sender Y = Recipient A = Arbiter M=Message T=Timestamp

Scheme 1: Conventional encryption, Arbiter sees the message:

The sender X and arbiter A share the master key K_{ax} the receiver y and the arbiter Ashare the master key K_{ay}

When X wants to send a message M to Y, construct a message computes the hash value H(M). This hash is encrypted using symmetric encryption with the key Kax which acts as signature. The message along with the signature is transmitted to A.

At A, it decrypts the signature and checks the hash value to validate the message. A transmit the message to Y, encrypted with K_{ay}. Y decrypt to extract the message and signature.

Disadvantage:

Eaves dropper can read the message as there is no confidentiality.

Scheme 2: Conventional encryption, Arbiter does not see the message:

- K_{ax} and K_{ay} are the master keys.
- K_{xy} is the key shared between the X and Y
- When x wants to transmit a message to Y, the packet goes to arbiter.
- The same procedure as that of I scheme is used X transmit an identifier, a copy of the message encrypted with K_{xy} and a signature to A.
- The signature is the hash of the message encrypted with K_{xa}
- A decrypt the signature, and checks the hash value to validate the message.
- A cannot read the message, A attaches to it the time stamps, encrypt with K_{xa} and transmit to Y.

Attack: The arbiter can join with an attacker and deny a message with sender‟s signature.

Scheme 2: Public key encryption, Arbiter does not see the message:

This method uses the public key cryptography which gives authentication and digital signature.

The doubly encrypted message is concatenated with ID_x and sent to arbiter.

- A can decrypt the outer encryption to ensure that the message has come from X.
- A then transmit the message with ID_x and time stamp.

Advantages:

- No information is shared among parties before communication, hence fraud is avoided.
- No incorrectly dated message can be sent.

Disadvantages:

The complex public key algorithm is to be twice for encryption and twice for decryption.

Authentication Protocols

Authentication Protocols used to convince parties of each other‟s identity and to exchangesession keys.

Mutual Authentication

An important application area is that of mutual authentication protocols. Such protocols enable communicating parties to satisfy themselves mutually about each other‟s identity and to

exchange session keys.

Key issues are

- **confidentiality** — to protect session keys and prevent masqueraded and compromised
- **timeliness** – to prevent replay attacks

Replay Attacks
Where a valid signed message is copied and later resent

- Simple replay
  The opponent simply copies the message and replays it later.

- Repetition that can be logged
  The opponent replay a time stamped message within a valid time window.

- Repetition that cannot be detected
  The attacker would have suppressed the original message from the receiver. Only the replay message alone arrives.

- Backward replay without modification
  This replay back to the sender itself.This is possible if the sender cannot easily recognize the difference between the message sent and the message received based on the content.

Countermeasures include

One approach to coping with replay attacks is to attach a sequence number to each message used in an authentication exchange. A new message is accepted only if its sequence number is in the proper order.

The difficulty with this approach is that it requires each party to keep track of the last sequence number for each claimant it has dealt with. Because of this overhead, sequence numbers are generally not used for authentication and key exchange. Instead, one of the following two general approaches is used:

- **Timestamps:** Party A accepts a message as fresh only if the message contains a **Timestamp** that is close enough to A‟s knowledge of current time. This approach requires that clocks among the various participants be synchronized.
- **Challenge/response:** Party A, expecting a fresh message from B, first sends B a **nonce** (challenge) and requires that the subsequent message (response) received from B contain the correct nonce value.

Using Symmetric Encryption

- Use a two-level hierarchy of keys
- Usually with a trusted Key Distribution Center (KDC)
  - Each party shares own master key with KDC
  - KDC generates session keys used for connections between parties
  - Master keys used to distribute the session ke

Needham-Schroeder Protocol

- Original third-party key distribution protocol
- For session between A and B mediated by KDC
- Protocol overview is:

1. A→KDC: $ID_A || ID_B || N_1$
2. KDC→A: $EK_a[K_s || ID_B || N_1 || EK_b[K_s || ID_A]]$
3. A→B: $EK_b[K_s || ID_A]$
4. B→A: $EK_s[N_2]$
5. A→B: $EK_s[f(N_2)]$

Step 1: A to KDC ,transmit the id of source and destination and a nonce N_1 as a request. Step 2: A securely acquires the session key in step 2 and a packet to B encrypted with EK_b isalso received from KDC.
Step 3: A transmit to B the message it got from KDC.
Step 4: As a hand shake, B encrypts a new nonce N_2 and transmit to A with K_s.Step 5: As a hand shake, A encrypt the function of N_2 with K_s
Step 4 and Step 5 as used as hand shake and prevent the reply attacks.

Attacks:

- Used to securely distribute a new session key for communications between A &B
- But is vulnerable to a replay attack if an old session key has been Compromised
  - Then message 3 can be resent convincing B that is communicating With A
- Modifications to address this require:
  - Timestamps
  - Using an extra nonce

DenningProtocol:
To overcome the above weakness by a modification to the Needham/Schroeder protocol that includes the addition of a timestamp to steps 2 and 3.

A→KDC: $ID_A || ID_B$

KDC → A: $E(K_a, [K_s II ID_B II T II E(K_b, [K_s III D_A II T])])$A→B: $E(K_b, [K_s II ID_A IIT])$

B→A: $E(K_s, N1)$

A→ B: $E(K_s,$

$f(N1))$

T is a timestamp that assures A and B that the session key has only just been generated. Thus, both A and B know that the key distribution is a fresh exchange.A and B can verify timeliness by checking that

$$I\ Clock - T\ I < \Delta t1 + \Delta t2$$

The **Denning protocol** seems to provide an increased degree of security compared to the Needham/Schroeder protocol. However, a new concern is raised: namely, that this new scheme requires reliance on clocks that are synchronized throughout the network.

suppress-replay attacks:

The problem occurs when a sender‟s clock is ahead of the intended recipient‟s clock. In this case, an opponent can intercept a message from the sender and replay it later when the timestamp in the message becomes current at the recipient‟s site. This replay could cause unexpected results.

Method to overcome:

One way to counter suppress-replay attacks is to enforce the requirement that parties regularly check their clocks against the KDC‟s clock. The other alternative, which avoids the need for clock synchronization, is to rely on handshaking protocols using nonce.

This latter alternative is not vulnerable to a suppress-replay attack, because the nonce the recipient will choose in the future are unpredictable to the sender.

An attempt is made to respond to the concerns about suppress replay attacks and at the sametime fix the problems in the Needham/Schroeder protocol.

The protocol is

1. A: →B: $ID_A||N_a$
2. B: → KDC: $ID_B||N_b||E(K_b, [ID_A||N_a||T_b])$
3. KDC: → A: $E(K_a, [ID_B f\ fN_a f\ fKs\ f\ fT_b])$ „ $E(Kb , [ID_A,, Ks „ T_b]) „ N_b$
4. A: →B: $Kb, [ID_A,, K_s,, T_b]) „ E(K_s, N_b)$

1. A initiates the authentication exchange by generating a nonce,N_a , and sending that plus its identifier to B in plaintext. This nonce N_a will be returned to A in an encrypted message that includes the session key, assuring A of its timeliness.

2. B alerts the KDC that a session key is needed. Its message to the KDC includes its identifier and a nonce, . This nonce will be returned to B in an encrypted message that includes the session key, assuring B of its timeliness.B‟s message to the KDC also includes a block encrypted with the secret key shared by B and the KDC. This block is used to instruct the KDCto issue credentials to A; the block specifies the intended recipient of the credentials, a suggested expiration time for the credentials, and the nonce received from A.

3. The KDC passes on to A B‟s nonce and a block encrypted with the secret key that B shares with the KDC. The block serves as a “ticket” that can be used by A for subsequent authentications, as will be seen. The KDC also sends to A block encrypted with the secret key shared by A and the KDC. This block verifies that B has received A‟s initial message (ID_B) and that this is a timely message and not a replay (N_a), and it provides A with a session key (K_S)and the time limit on its use (T_b).

4. A transmits the ticket to B, together with the B‟s nonce, the latter encrypted with the session key. The ticket provides B with the secret key that is used to decrypt $E(K_S, N_b)$ to recover the nonce. The fact that B‟s nonce is encrypted with the session key authenticates that themessage came from A and is not a replay.

Using Public-Key Encryption

- Have a range of approaches based on the use of public-key encryption
- Need to ensure have correct public keys for other parties
- Using a central authentication server (AS)
- Various protocols exist using timestamps or nonces

Denning AS Protocol

- Denning presented the following:

1. A→AS: $ID_A \| ID_B$
2. AS→A: $E_{KRas}[ID_A \| KU_a \| T] \| EKRas[ID_B \| KU_b \| T]$
3. A→B: $E_{KRas}[ID_A \| KU_a \| T] \| EKRas[ID_B \| KU_b \| T] \| EKU_b[E_{KRas}[Ks \| T]]$

- Note session key is chosen by A, hence AS need not be trusted to protect it
- timestamps prevent replay but require synchronized clocks

Another approach proposed by woo and lam makes use of nonce.

1. A→KDC: $ID_A \| ID_B$
2. *KDC*→A: $E_{KRauth}[ID_B \text{ II } K_{Ub}]$
3. a→b: $EK_{Ub}[N_a \text{ II } ID_A]$
4. B→KDC: $ID_B \text{ II } ID_A \text{ II } EK_{Uauth}\ [N_a]$
5. KDC→B: $E_{KRauth}\ [IDA \text{ II } KUa] \text{ II } E_{KUb}[E_{KRauth}[N_a \text{ II } K_s \text{ II } ID_B]]$
6. B→A: $E_{KUa}[E_{KRauth}[N_a \text{ II } K_s \text{ II } ID_B] \text{ II } N_b]$
7. A→B: $E_{Ks}[N_b]$

Step 1: A informs the KDC of its intention to establish a secure connection with B.

Step 2: The KDC returns to A a copy of B‟s public key certificate.

Step 3: A informs B of its desire to communicate and sends a nonce N_a.

Step 4: B asks the KDC for A‟s public key certificate and request a session key.;B includes A‟s nonce so that the KDC can stamp the session key with that nonce.The nonce is protected using the KDC‟s public key.

Step 5: The KDC returns to B a copy of A‟s public key certificate, plus the information$[N_a, K_s, ID_B]$.

Step 6: The triple $[N_a, K_s, ID_B]$, still encrypted with the KDC‟s private key,is relayed to A,together with a nonce N_b generated by B.

All the foregoing are encrypted using A‟s public key. A retrieves the session key K_s and uses it to encrypt N_b and return it to B.

Step 7: Assures B of A‟s knowledge of the session key.

One-Way Authentication

- Required when sender & receiver are not in communications at same time (eg. Email)
- Have header in clear so can be delivered by email system
- May want contents of body protected & sender authenticated

Using Symmetric Encryption

- can refine use of KDC but can‟t have final exchange of nonce

1. A→KDC: $ID_A || ID_B || N_1$
2. KDC→A: $EK_a[K_s || ID_B || N_1 || EK_b[K_s || ID_A]]$
3. A→B: $EK_b[K_s || ID_A] || EK_s[M]$

- Does not protect against replays
  - could rely on timestamp in message, though email delays make this problematic

Public-Key Approaches

- If confidentiality is major concern, can use:

$$A \rightarrow B: EKU_b[K_s] || EK_s[M]$$

In this case, the message is encrypted with a one-time secret key. A also encrypts this one-time key with B‟s public key. Only B will be able to use the corresponding private key to recover the one-time key and then use that key to decrypt the message. This scheme is more efficient than simply encrypting the entire message with B‟s public key.

- If authentication needed use a digital signature with a digital certificate:

$$A \rightarrow B: M,,II\ E_{KRa}(\ H(M))$$

This method guarantees that A cannot later deny having sent the message. However, this technique is open to another kind of fraud. Bob composes a message to his boss Alice that contains an idea that will save the company money. He appends his digital signature and sends it into the e-mail system.

Eventually, the message will get delivered to Alice‟s mailbox. But suppose that Max has heard of Bob‟s idea and gains access to the mail queue before delivery. He finds Bob‟s message, strips off his signature, appends his, and requeues the message to be delivered to Alice. Max gets credit for Bob‟s idea.

To counter such a scheme, both the message and signature can be encrypted with therecipient‟s public key:

$$A \rightarrow B: E_{KUb}, [M || E_{KRa}, H(M)]$$

The latter two schemes require that B know A‟s public key and be convinced that it is timely. An effective way to provide this assurance is the digital certificate

$$A \rightarrow B: M \,||\, EKR_a [H(M)] \,||\, EKR_{as} \text{ II } T|| \, ID_A \,||\, KU_a]$$

In addition to the message,A sends B the signature encrypted with A‟s private key and A‟s certificate encrypted with the private key of the authentication server. The recipient of the message first uses the certificate to obtain the sender‟s public key and verify that it is authentic and then uses the public key to verify the message itself.

4.7. DSS

The DSS Approach

The DSS uses an algorithm that is designed to provide only the digital signature function. Unlike RSA, it cannot be used for encryption or key exchange. Nevertheless, it is a public-key technique.

RSA approach

The message to be signed is input to a hash function that produces a secure hash code of fixed length. This hash code is then encrypted using the sender‟s private key to form the signature.

Both the message and the signature are then transmitted. The recipient takes the message and produces a hash code. The recipient also decrypts the signature using thesender‟s public key.

If the calculated hash code matches the decrypted signature, the signature is accepted as valid. Because only the sender knows the private key, only the sender could have produceda valid signature.

DSS approach

The DSS approach also makes use of a hash function. The hash code is provided as input to a signature function along with a random number generated for this particular signature.

The signature function also depends on the sender‟s private key (*PRa*) and a set of parameters known to a group of communicating principals. We can consider this set to constitute a global public key (PU_G). The result is a signature consisting of two components, labeled *s* and *r(fig 3.15)*.

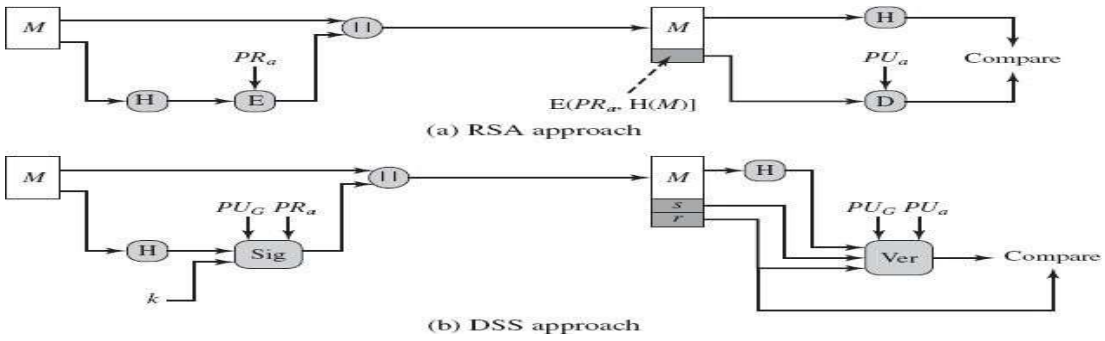


Figure 3.15 Two Approaches to Digital Signatures

At the receiving end, the hash code of the incoming message is generated. This plus the signature is input to a verification function. The verification function also depends on the global public key as well as the sender‟s public key, which is paired with the sender‟s private key.

The output of the verification function is a value that is equal to the signature component if the signature is valid. The signature function is such that only the sender, with knowledge of the private key, could have produced the valid signature.

The Digital Signature Algorithm:

There are three parameters that are public and can be common to a group of users.

- A 160-bit prime number q is chosen.
- Next, a prime number p is selected with a length between 512 and 1024 bits such that q divides (p - 1).
- Finally, g is chosen to be of the form $h^{(p-1)/q} \bmod p$, where h is an integer between 1 and (p-1).

With these numbers in hand, each user selects a private key and generates a public key.The private key x must be a number from 1 to (q-1) and should be chosen randomly. T

The public key is calculated from the private key as $y = g^x \bmod p$ .The calculation of given(Fig 3.16) is relatively straightforward. However, given the public key y, it is believed to be computationally infeasible to determine x, which is the discrete logarithm of y to the base g, mod p.

At the receiving end, verification is performed using the formulas. The receiver generates a quantity v that is a function of the public key components, the sender‟s public key, and the hash code of the incoming message. If this quantity matches the component of the signature, then the signature is validated.

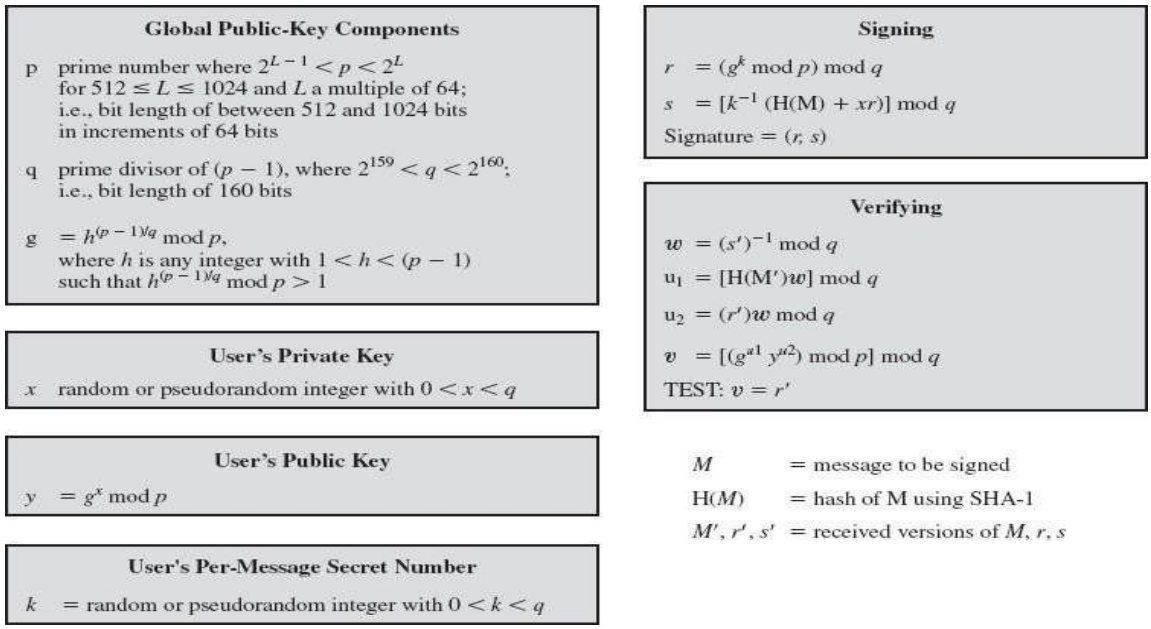


Figure.The Digital Signature Algorithm (DSA)

The value r does not depend on the message at all. Instead, r is a function of k and the three global public-key components.

The multiplicative inverse of k (mod q) is passed to a function that also has as inputs the message hash code and the user‟s private key.

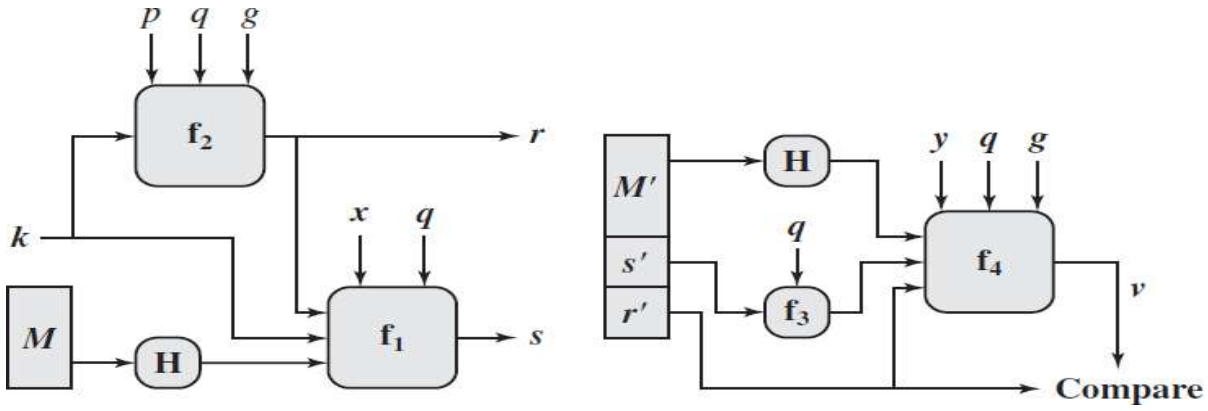
The structure of this function is such that the receiver can recover using the incoming message and signature, the public key of the user, and the global public key. Given the difficulty of taking discrete logarithms, it is infeasible for an opponent to recover k from r to recover x from s.

The only computationally demanding task in signature generation is the exponential calculation $g^k \bmod p$. Because this value does not depend on the message to be signed, it can be computed ahead of time.

Selects a private key and generates a public key. The private key x must be a number from 1 to (q1) and should be chosen randomly. The public key is calculated from the private key as $y = g^x \bmod p$.

To create a signature, a user calculates two quantities, r and s, that are functions of the public key components (p, q, g), the user's private key (x), the hash code of the message, H(M), and an additional integer k that should be generated randomly and be unique for each signing.

At the receiving end, verification is performed using the formulas. The receiver generates a quantity v that is a function of the public key components, the sender's public key, and the hash code of the incoming message. If this quantity matches the r component of the signature, then the signature is validated (Fig).



4.8. AUTHENTICATION APPLICATIONS

One of the key aspects of cryptography and network security is authentication. It helps to establish trust by identifying a particular user or a system. There are many ways to authenticate a user. Traditionally, user ids and passwords have been used.

1.Authentication Requirements

During communication across networks, following attacks can be identified.

1. **Disclosure:** Releases of message contents to any person or process not possessingthe appropriate cryptographic key.

2. **Traffic analysis:** Discovery of the pattern of traffic between parties.

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

4. Content modification: Changes to the content of the 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.

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

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

The security measures for the above mentioned attacks are as follows

For 1,2 - Message Confidentiality
3,4,5,6 - Message Authentication
7 - Digital Signatures
8 - Digital signature with protocol designed to counter the attack

2.Authentication Functions

Any message authentication or digital signature mechanism can be viewed as having fundamentally two levels.

1. **Lower level:** Some function that produces an authenticator: a value to be used to authenticate a message.

2. **Higher Level:** Lower layer functions are used to create a protocol that enables a receiverto verify the authenticity of message

The different types of functions that may be used to produce an authenticator are as follows:

1. **Message encryption:** The cipher text of the entire message serves as its authenticator.

2. **Message Authentication Code (MAC):** A public function of the message and a secret key that produces a fixed length value serves as the authenticator.

3. **Hash function:** A public function that maps a message of any length into a fixed length hash value, which serves as the authenticator.

4.9. KERBEROS

Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users. Kerberos relies exclusively on conventional encryption, making no use of public-key encryption.

Motivation

A distributed architecture consists of dedicated user workstations (clients) and distributed or centralized servers. In this environment, there are three approaches to security:

- Rely on each individual client workstation to assure the identity of its user or users and rely on each server to enforce a security policy based on user identification (ID).
- Require that client systems authenticate themselves to servers, but trust the client system concerning the identity of its user.
- Require the user to prove his or her identity for each service invoked. Also require that servers prove their identity to clients.

The following are the **requirements for Kerberos**:

- **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 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 Needham and Schroeder.

It is trusted in the sense that clients and servers trust Kerberos to mediate their mutual authentication. Assuming the Kerberos protocol is well designed, and then the authentication service is secure if the Kerberos server itself is secure.

Two versions of Kerberos are in common use. **Version 4** and **Version 5**

Kerberos Version 4

Version 4 of Kerberos makes use of DES, in a rather elaborate protocol, to provide the authentication service

1.A Simple Authentication Dialogue

In an unprotected network environment, any client can apply to any server for service. The obvious security risk is that of impersonation. To counter this threat, servers must be able to confirm the identities of clients who request service. But in an open environment, this placesa substantial burden on each 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. The simple authentication dialogue is as follows:

1. C >> AS: $ID_c||P_c||ID_v$
2. AS >> C: Ticket
3. C >> V: $ID_c||$Ticket

   Ticket=

   $EK_v(ID_c||ADc||ID_v)$

C : Client,
AS : Authentication Server,
V : Server, ID_c : ID of the client,P_c : Password of the client,
AD_c : Address of client, ID_v : ID of the server,K_v : secret key shared by AS and V,
|| : concatenation.

2.A More Secure Authentication Dialogue

There are two major problems associated with the previous approach:

- Plaintext transmission of the password.
- Each time a user has to enter the password.

To solve these problems, we introduce a scheme for avoiding plaintext passwords, anda new server, known as ticket granting server (TGS). The hypothetical scenario is as follows:

Once per user logon session:-

1. C >> AS: $ID_c||ID_{tgs}$
2. AS >> C: $Ek_c\,(Ticket_{tgs})$

Once per type of service:

3. C >> TGS: $ID_c||ID_v||Ticket_{tgs}$
4. TGS >> C: $ticket_v$

Once per service session:

5. C >> V: $ID_c||\ Ticket_v$

$Ticket_{tgs}=$
$Ekt_{gs}(ID_c||AD_c||IDt_{gs}||TS_1||Lifetime_1)Ticket_v=$
$Ek_v(ID_c||AD_c||ID_v||TS2||Lifetime_2)$

C: Client, AS: Authentication Server, V: Server,
IDc : ID of the client, Pc:Password of the client, ADc: Address of client,IDv : ID of the server, Kv: secret key shared by AS and V,
|| : concatenation, IDtgs: ID of the TGS server, TS1, TS2: time stamps, lifetime:lifetime of the ticket.

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.

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 and password 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.

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.

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 Kerberoswithout having to transmit the password in plaintext. Now that the client has a ticket-granting ticket, access to any server can be obtained with steps 3 and 4:

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 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 authenticatethe user. If

the user is permitted access to the server V, the TGS issues a ticket to grant access to the requested service.

The service-granting ticket has the same structure as the ticket-granting ticket. Indeed, because the TGS is a server, we would expect that the same elements are needed to authenticate a client to the TGS and to authenticate a client to an application server.

Again, the ticket contains a timestamp and lifetime. If the user wants access to the same service at a later time, the client can simply use the previously acquired service-granting ticket and need not bother the user for a password.

Note that the ticket is encrypted with a secret key (K_v) known only to the TGS and the server, preventing alteration.

Finally, with a particular service-granting ticket, the client can gain access to the corresponding service with step 5:

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.

This new scenario satisfies the two requirements of only one password query per user session and protection of the user password.

Kerberos V4 Authentication Dialogue Message Exchange

Two additional problems remain in the more secure authentication dialogue:

- Lifetime associated with the ticket granting ticket. If the lifetime is very short, then the user will be repeatedly asked for a password. If the lifetime is long, then the opponent has the greater opportunity for replay.

- Requirement for the servers to authenticate themselves to users.

The actual Kerberos protocol version 4 is as follows
:

- A basic third-party authentication scheme
- Have an Authentication Server (AS)
  - Users initially negotiate with AS to identify self
  - AS provides a non-corruptible authentication credential (ticket grantingticket TGT)

- Have a Ticket Granting
  - Users subsequently request access to other services from TGS on basisof users TGT

(a) Authentication service exchange: to obtain ticket granting ticket
(1) $C \rightarrow AS : ID_C \text{ II } ID_{tgs} \text{ II } TS_1$ (2) $AS \rightarrow C : EKc [K_{c,tgs} \text{ II } ID_{tgs} \text{ II } TS_2 \text{ II } Lifetime_2 \text{ II } Ticket_{tgs}]$
(b) Ticket-Granting Service Exchange: to obtain service-granting ticket

(3) C → TGS: ID_v II $Ticket_{tgs}$ II $Authenticator_c$

(4) TGS → C: $EK_{c,tgs}[K_{c,y}$ II ID_v II TS_4 II $Ticket_v]$

$Ticket_{tgs} = E_{K,tgs}[K_{c,tgs}$ II ID_C II AD_C II ID_{tgs} IITS_2 II $Lifetime_2]$

$Ticket_v = E_{Kv}[K_{c,v}$ II ID_C II AD_C II ID_v IITS_4 II $Lifetime_4]$

$Authenticator_C = E_{Ktgs}$ [ID_C II AD_C II $TS_3]$

(c) Client/Server Authentication Exchange: to obtain service

(5) C → V : $Ticket_v$ II $Authenticator_c$

(6) V → C: $E_{kc,v}[TS_5 +1]$

$Ticket_v = EK_v[K_{c,v}$ II ID_C II AD_C II Id_v II TS_4 II $Lifetime_4]$

$Authenticator_c = EK_{tgs}$ [ID_C II AD_C II $TS_3]$

Kerberos 4 Overview

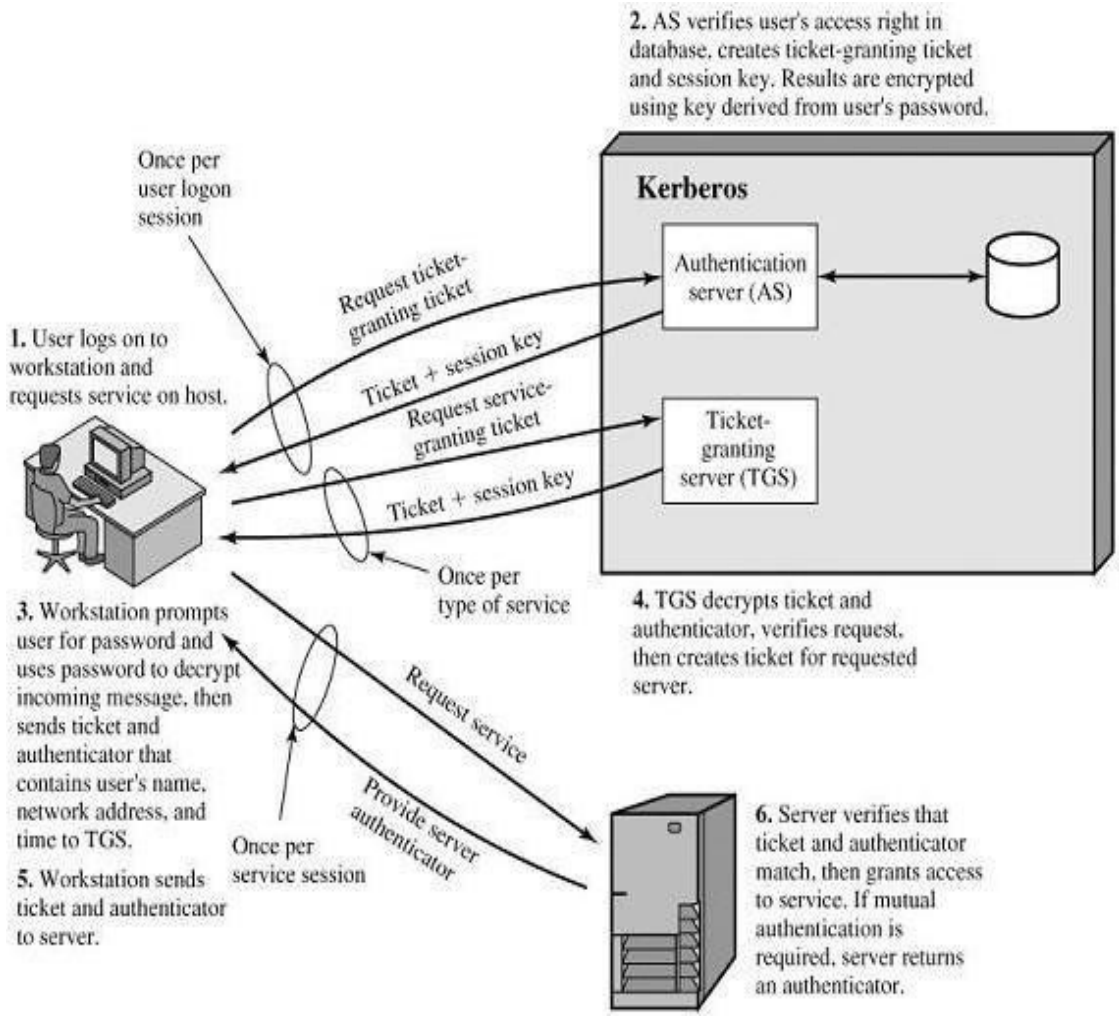


Fig 4.1 Overview of Kerberos 4

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:

4. 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.
5. 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.

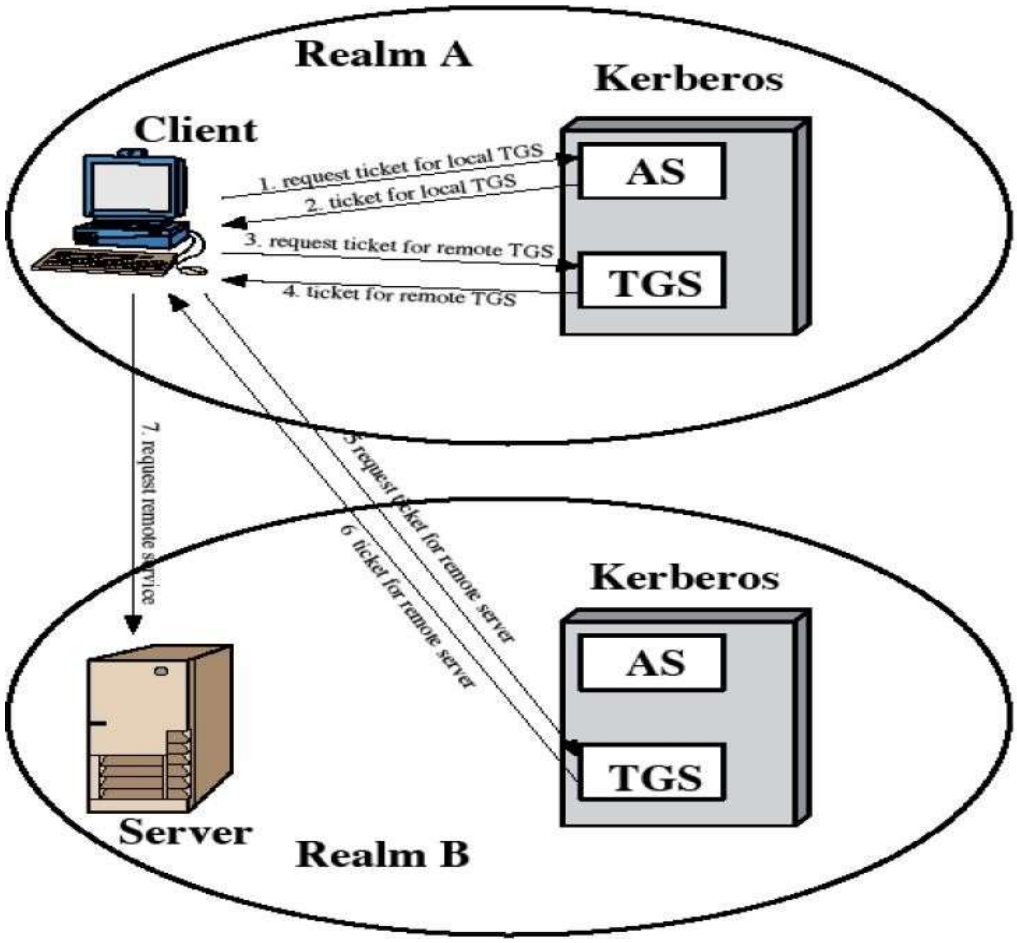


Fig .Request for service in another Realm

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 keptin 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 inter realm authentication. For two realms to support inter realm authentication, a third requirement is added:

6. 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.

The scheme requires that the Kerberos server in one realm trust the Kerberos server in the other realm to authenticate its users. Furthermore, the participating servers in the second realm must also be willing to trust the Kerberos server in the first realm.

The details of the exchanges illustrated in Fig 2are as follows:

$C \rightarrow AS$:ID_C II ID_{tgs} II TS_1

$AS \rightarrow C$:$EK_c[K_{c,tgs}$ ii ID_{tgs} II TS_2 II $Lifetime_2$ II

$Ticket_{tgs}C \rightarrow TGS$:ID_{tgsrem} II $Ticket_{tgs}$ II $Authenticator_c$

$TGS \rightarrow C$:E $K_{c,tgs}[K_{c,tgsrem}$ II $IDtgsrem$ II TS4 II

$Ticket_{tgsrem}C \rightarrow TGS_{rem}$:ID_{vrem} II $Ticket_{tgsrem}$ II

$Authenticator_c$

$TGS_{rem} \rightarrow C$:$EK_{c,tgsrem}$ [$K_{c,vrem}$ II ID_{vrem} II TS_6 II $Ticket_{vrem}$:

$C \rightarrow V_{rem}$:$Ticket_{vrem}$ II$Authenticator_c$

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.

Environmental shortcomings:

7. **Encryption system dependence:**

Version 4 requires the use of DES. In version 5, ciphertext is tagged with an encryption type identifier so that any encryption technique may be used.

8. **Internet protocol dependence:**

Version 4 requires the use of Internet Protocol (IP) addresses. Version 5 network addresses are tagged with type and length, allowing any network address type to be used.

9. **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 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 ordering.

10. **Ticket lifetime:**

Lifetime values in version 4 are encoded in an 8-bit quantity in units of five minutes. In version 5, tickets include an explicit start time and end time, allowing tickets with arbitrary lifetimes.

11. **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. Version 5 provides this capability.

Technical deficiencies in the version 4 protocol:

- Double encryption
- PCBC encryption
- Session keys
- Password attacks

The Version 5 Authentication Dialogue

(a) Authentication Service Exchange: to obtain ticket-granting ticket
(1) $C \rightarrow AS$: Options II ID_c II $Realm_c$ II Times II $Nonce_1$
(2) $AS \rightarrow C$: $Realm_c$ II ID_c II $Ticket_{tgs}$ II EK_c [$K_{c,tgs}$ II Times II $Nonce_1$ II $Realm_{tgs}$ II ID_{tgs}]$Ticket_{tgs} = EK_{tgs}$ [Flags II $K_{c,tgs}$ II $Realm_c$ II ID_c II AD_c II Times]
(b) Ticket – Granting Service Exchange:to obtain service-granting ticket
(3) $C \rightarrow TGS$: Optionns II ID_v II Times II $Nonce_1$
(4) $TGS \rightarrow C$: $Realm_c$ II ID_c II $Ticket_v$ II $EK_{c,tgs}$[$K_{c,v}$ II Times II $Nonce_2$ II $Realm_v$ II ID_v]$Ticket_{tgs} = EK_{tgs}$[Flags II $K_{c,tgs}$ II $Realm_c$ II ID_c II AD_c II Times] $Ticket_v = Ek_v$[[Flags II $K_{c,v}$ II $Realm_c$ II ID_c II AD_c II Times]

$Authenticator_c = EK_{c,tgs}[ID_c \text{ II } Realm_c \text{ II } TS_1]$
(c) Client/Server AUTHENTICATION Exchange: to obtain service
(5) C → V : Options II $Ticket_v$ II $Authenticator_c$ (6) V → C : $EK_{c,v}$ [TS_2 II subkey II Seq#] $Ticketv = EK_v[Flags \text{ II } K_{c,v} \text{ II } Realm_c \text{ II } ID_c \text{ II } AD_c \text{ II Times}]$ $Authenticator_c = E_{Kc,v}[ID_c \text{ II } Realm_c \text{ II } TS_2 \text{ II Subkey II Seq\#}]$

First, consider the authentication service exchange. Message (1) is a client request for a ticket-granting ticket. 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
  - r_{time} : requested renew-till time

Nonce: A random value to be repeated in message (2) to assure that the response is fresh andhas not been replaced by an opponent .

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.

Let us now compare the ticket-granting service exchange for versions 4 and 5.

We see that message (3) for both versions include 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.

The authenticator itself is essentially the same as the one used in version 4.

Message (4) has the same structure as message (2), returning a ticket plus information needed by the client, the latter encrypted with 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 as follows:

- **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 (Kc,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 maybe sequence numbered to detect replays.

If mutual authentication is required, the server responds with message (6). This message includes the timestamp from the authenticator. Note that in version 4, the timestamp was incremented by one. This is not necessary in version 5 because the nature of the format of messages is such that it is not possible for an opponent to create message (6) without knowledge of the appropriate encryption keys.

X.509 AUTHENTICATION SERVICES

X.509 defines a framework for authentication services by the X.500 directory to its users. The directory consists of public-key certificates.

Each certificate contains the public key of a user and is signed with the private key of a trusted certification authority.

X.509 defines authentication protocols based on public key certificates. X.509 standard certificate format used in S/MIME, IP Security and SSL/TLS and SET.

Certificates

The certificates are created and stored in the directory by the trusted Certification Authority (CA). The directory server not having certification functions and not create public key. But the user obtains the certificate from some easily accessible location

The general format of the certificate as shown below Fig 4.3

The elements of the certificates are

1. **Version(V):** The default version is 1. The issuer and subject unique identifier are present in version 2. If one or more extensions are present in version 3.
2. **Serial Number (SN):** Unique integer value issued by CA
3. **Signature Algorithm Identifier (AI):** This algorithm is used to sign the certificate withsome parameters
4. **Issuer Name (CA):** The name of the CA that created and signed this certificate
5. **Period of validity (T_A):** The first and last on which the certificate is valid
6. **Subject Name (A):** The name of the user to whom this certificate refers
7. **Subject's Public Key Information (AP):** The public key of the subject plus identifier ofthe algorithm for which this key is to be used, with associated parameters.

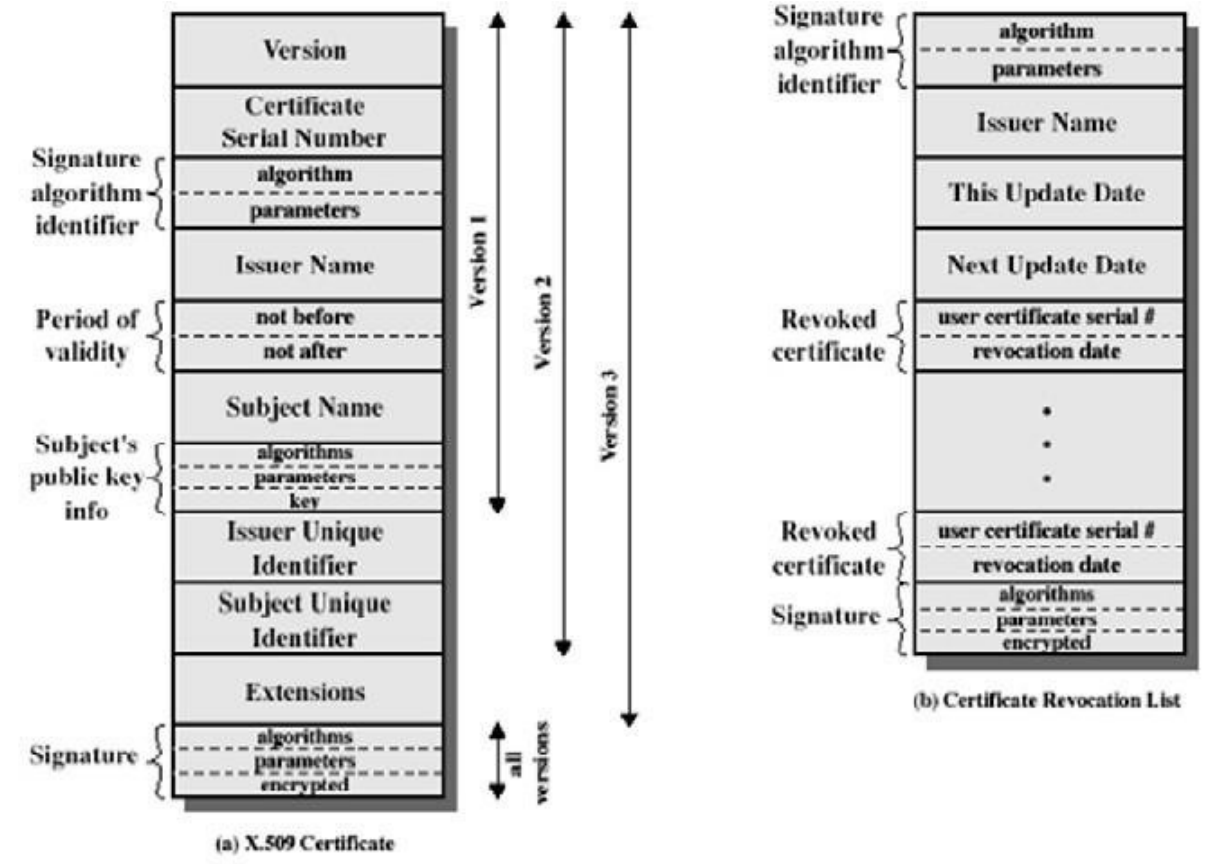


Fig X.509 AUTHENTICATION SERVICES

Issuer Unique Identifier: It is used to identify uniquely the issuing CA

8. **Subject Unique Identifier:** It is used to identify uniquely the subject
9. **Extensions:** A set of one or more extension fields
10. **Signature:** Covers all of the other fields of certificate; it contains hash code of other fields, encrypted with the CA‟s private key.

[**Note:** Unique identifier is used to avoid reuse of subject and issuer names over time]

Notation to define a certificate

CA<<A>> = CA {V, SN, AI, CA, TA, A, Ap}
where
Y<<X>> = The certificate of user X issued by certification authority Y.
Y {I} = The signing of I by Y. It consists of I with an encrypted hash code appended.

The CA signs the certificate with its private key. If the corresponding public key is knownto a user, then that user can verify that a certificate signed by the CA is valid.

Generation and usage of certificate by a user

The user certificates generated by CA have the following characteristics:

11. Any user with access to the public key of the CA can verify the user publickey that was certified.
12. No party other than the Certification Authority (CA) can modify the certificate withoutthis being detected.
13. Certificates are unforgeable,

If all users belong to the same CA, the certificates can be placed in the directory for access by all users. If the number of users increased, single CA could not be satisfied all user requirements.

For example, User A has obtained the certificate from certificate authority x_1 and user B from x_2. Now the two CAs (x_1 and x2) securities exchange their own public keys in the form of certificates. That is x_1 must hold x_2"s certificate and x_2 holds x_1"s certificate

Now A want s to access B"s public key, it follows the following chain to obtain B"s public key.

x_1<< x_2>> x_2<<B>>

i.e., first A gets x_2"s certificate from x_1"s directory to obtain x_2"s public key. Then using x_2"s public key to obtain B"s certificate from x_2"s directory to obtain„s public key.

In the same method, B can obtain A"s public key with the reverse chain
x_2<< x_1>> x_1<<A>>

Hierarchy of CAs

To obtain public key certificate of user efficiently, more than one CAs can be arranged in a hierarchy, so that navigation in easy.

The connected circles indicate the hierarchical(Fig 4.4) relationship among the CAs; the associated boxes indicate certificates maintained in the directory for each CA entry. The directory entry for each CA includes two types of certificates:

- **Forward certificates**: Certificates of X generated by other CAs
- **Reverse certificates**: Certificates generated by X that are the certificates of other CAs

User A can acquire the following certificates from the directory to establish a certification path to B:

X<<W>> W <<V>> V <<Y>><<Z>> Z <<B>>

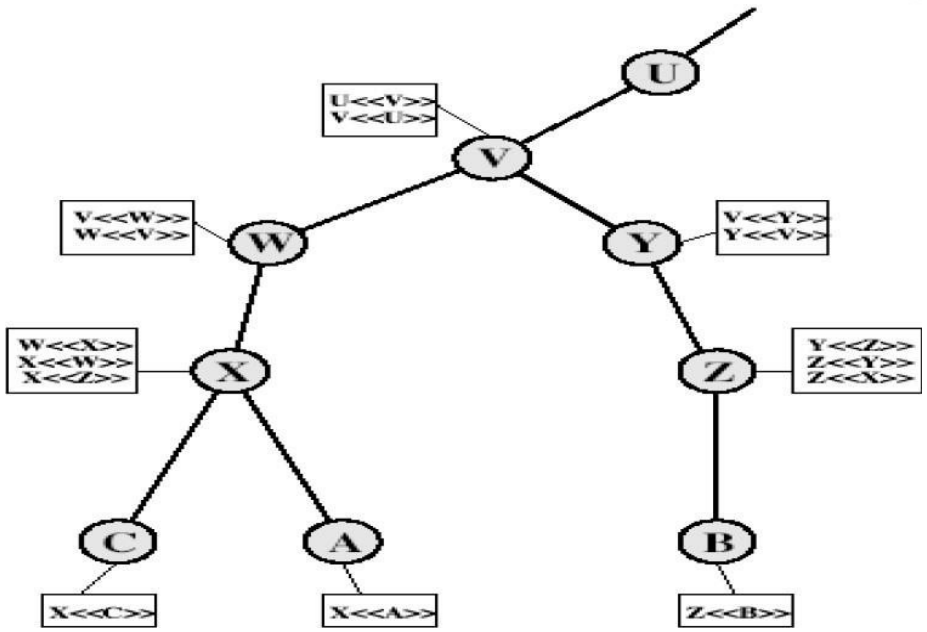


Fig4.4 : Hierarchy of X.509

Revocation of certificates

- Certificates have a period of validity
- May need to revoke before expiry, for the following reasons eg:
  - ✓ User's private key is compromised
  - ✓ User is no longer certified by this CA
  - ✓ CA's certificate is compromised
- CA‟s maintain list of revoked certificates
  - ✓ The Certificate Revocation List (CRL)
- Users should check Certificates with CA‟s CRL

Authentication Procedures

X.509 includes three alternative authentication procedures:

- One-Way Authentication
- Two-Way Authentication
- Three-Way Authentication

One-Way Authentication

One message (A→B) used to establish

- The identity of A and that message is from A
- Message was intended for B
- Integrity & originality of message

Message must include timestamp, nonce, B's identity and is signed by A

Two-Way Authentication

Two messages (A→B, B→A) which also establishes in addition:

- The identity of B and that reply is from B
- That reply is intended for A
- Integrity & originality of reply

Reply includes original nonce from A, also timestamp and nonce from B

Three-Way Authentication

Three messages (A→B, B→A, A→B) which enables above authentication without synchronized clocks (Fig 4.5)

- Has reply from A back to B containing signed copy of nonce from B
- Means that timestamps need not be checked or relied upon

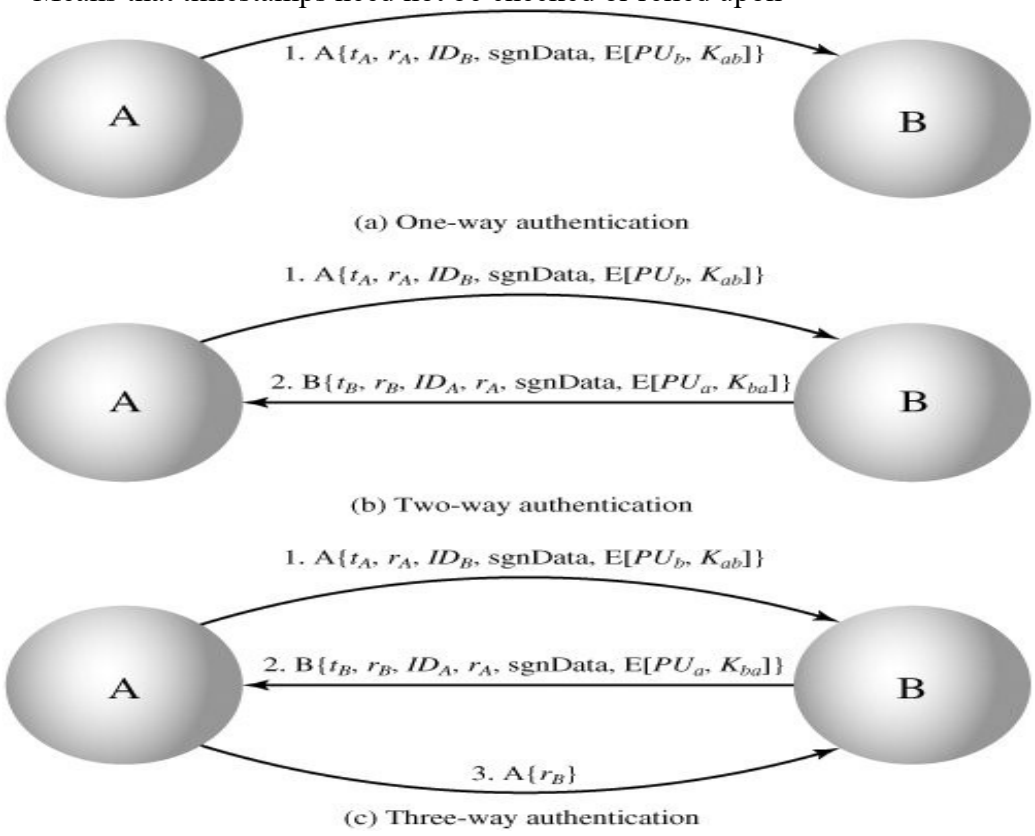


Fig: X509 Strong Authentication Procedure

X.509 Version 3

The following requirements not satisfied by version 2:

14. The Subject field is inadequate to convey the identity of a key owner to a public-key user.
15. The Subject field is also inadequate for many applications, which typically recognize entities by an Internet e-mail address, a URL, or some other Internet-related identification.
16. There is a need to indicate security policy information. There is a need to limit the damage that can result from a faulty or malicious CA by setting constraints on the applicability of a particular certificate.
17. It is important to be able to identify different keys used by the same owner at different times.

The certificate extensions fall into three main categories: key and policy information, subject and issuer attributes, and certification path constraints.

Key and Policy Information

These extensions convey additional information about the subject and issuer keys, plus indicators of certificate policy.. For example, a policy might be applicable to the authentication of electronic data interchange (EDI) transactions for the trading of goods within a given price range.

This area includes the following:

Authority key identifier: Identifies the public key to be used to verify the signature on this certificate or CRL.
Subject key identifier: Identifies the public key being certified.
Key usage: Indicates a restriction imposed as to the purposes for which, and the policies under which, the certified public key may be used.
Private-key usage period: Indicates the period of use of the private key corresponding to the public key. For example, with digital signature keys, the usage period for the signing private key is typically shorter than that for the verifying public key.
Certificate policies: Certificates may be used in environments where multiple policies apply.
Policy mappings: Used only in certificates for CAs issued by other CAs.

Certificate Subject and Issuer Attributes

These extensions support alternative names, in alternative formats, for a certificatesubject or certificate issuer and can convey additional information about the certificate subject, to increase a certificate user's confidence that the certificate subject is a particular person or entity. For example, information such as postal address, position within a corporation, or picture image may be required.

The extension fields in this area include the following:

- **Subject alternative name:** Contains one or more alternative names, using any of a variety of forms

- **Subject directory attributes:** Conveys any desired X.500 directory attribute values for the subject of this certificate.

Certification Path Constraints

These extensions allow constraint specifications to be included in certificates issued for CAs by other CAs. The extension fields in this area include the following:

- **Basic constraints:** Indicates if the subject may act as a CA. If so, a certification path length constraint may be specified.
- **Name constraints**: Indicates a name space within which all subject names in subsequent certificates in a certification path must be located.
- **Policy constraints**: Specifies constraints that may require explicit certificate policy identification or inhibit policy mapping for the remainder of the certification path.