

## CRYPTOGRAPHIC HASH FUNCTIONS & DIGITAL SIGNATURES

**Syllabus:** Application of Cryptographic hash Functions, Requirements & Security, Secure Hash Algorithm, Message Authentication Functions, Requirements & Security, HMAC & CMAC. Digital Signatures, NIST Digital Signature Algorithm. Key management & distribution.(refer third unit)

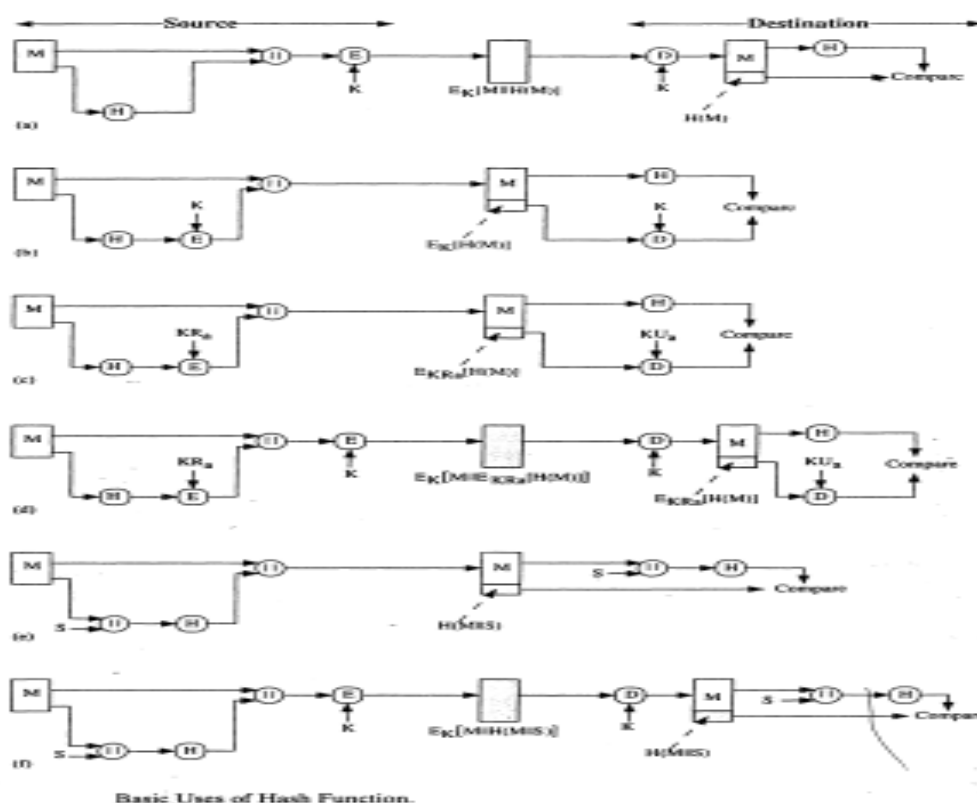
**Hash Functions:** A hash function  $H$  accepts a variable length of message  $M$  as input and produces a fixed size hash value  $h$ ,  $h = H(M)$ . In general terms the principal objective of hash function is data integrity. A change to any bit in message results with higher difference in the hash code. Thus the kind of function required for security applications is cryptographic hash function. A cryptographic hash function is an algorithm for which it is computationally infeasible to find either

- A data object that maps to a pre-specified hash result (or)
- Two data objects that maps to the same hash result.

Because of these two characteristics, hash functions are often used to determine whether or not data has changed.

**Applications of Cryptographic Hash functions:** This is used in a wide variety of security applications and internet protocols. Few of such applications are:

1. Message Authentication: It is a mechanism or service used to verify the integrity of a message.



Basic Uses of Hash Function.

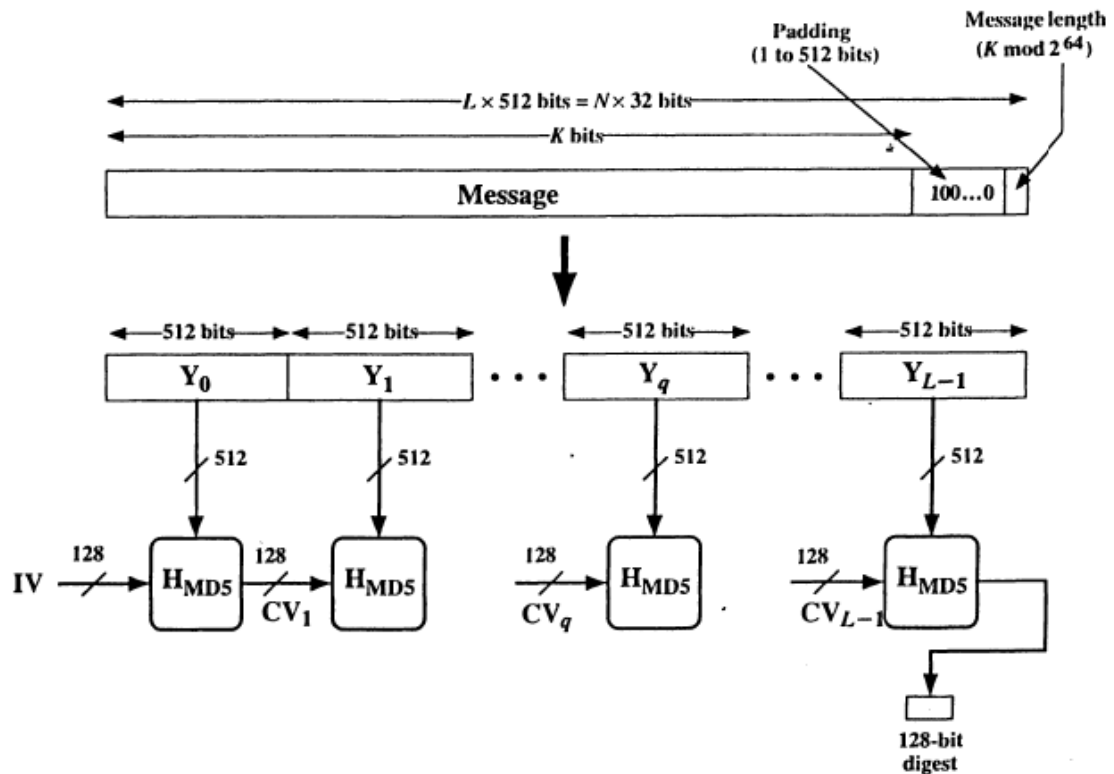
Message authentication assures that data received are exactly as sent. When a hash code is used to provide message authentication, the hash code is often referred to as a Message Digest. The following figure gives a variety of ways in which a hash code can be used to provide message authentication, as follows:

2. Digital Signatures: Another important application, which is similar to the message authentication application, is the digital signature. Its operation is similar to MAC. In digital signatures the hash value of a message is encrypted with the private key of a source. Anyone who knows the source public key can verify the signature.
3. Hash functions are commonly used to create a one-way password file. There is a scheme in which a hash of password is stored by an operating system.
4. Hash functions can be used for intrusion detection and virus detection
5. A cryptographic hash function can be used to construct a pseudorandom function or pseudorandom Number Generator (PRNG). A common application for hash based PRNG is for generation of symmetric keys.

### **Security Requirements for Cryptographic Hash Functions:**

1. H can be applied to a block of data of any size.
2. H produces a fixed-length output.
3.  $H(x)$  is relatively easy to compute for any given  $x$ , making both hardware and software implementations practical.
4. For any given code  $h$ , it is computationally infeasible to find  $x$  such that  $H(x)=h$ . This is sometimes referred to in the literature as the one-way property. Functions that lack this property are vulnerable to pre-image attacks.
5. For any given block  $x$ , it is computationally infeasible to find  $y \neq x$  with  $H(y)=H(x)$ . This is sometimes referred to as weak collision resistance. Functions that lack this property are vulnerable to second pre-image attacks.
6. It is computationally infeasible to find any pair  $(x,y)$  such that  $H(x)=H(y)$ . This is sometimes referred to as strong collision resistance. It requires a hash value at least twice as long as what required for pre-image resistance, otherwise collisions may be found by a birthday attack.

**Message Digest Algorithm (MD5):** MD5 was developed by Ron Rivest at MIT. The algorithm takes a variable length of message as input and outputs a 128 bit message digest. The input is processed in 512 bit blocks.



Message Digest Generation Using MD5.

1. Append Padding bits: Message is padded so that its length is 448 modulo 512.
2. Append length: After padding the bits, the expanded message is represented as a sequence of 512 bit blocks  $Y_0, Y_1, Y_2, \dots, Y_{L-1}$ ; so that the total length of message is  $L \times 512$  bits.
3. Initialize MD Buffer: A 128 buffer is used to hold intermediate and final results of the hash function. This buffer can be represented as four 32-bit registers A, B, C, D. These registers are initialized to the following 32-bit integers.

Ex: A=67452301

B=EFCDA89

C=-----

D=----

These values are stored in the little-endian format, means the least significant byte in a word is stored in the low address position.

That is word A=01 23 45 67

B=89 AB Cd EF

C=----

D=----

4. Process the message in 512 bit blocks: The four rounds have a similar structure, but each uses a different primitive logical function, referred as F,G,H and I

Each round takes as input (512) block and uses constants  $T[1], T[2], \dots, T[64]$  constructed from  $\sin 1$  to  $\sin 64$  functions.

That is  $T[i] = 2^{32} \times \text{abs}(\sin(i))$  where  $T[i]$  is the integer part of above equation.

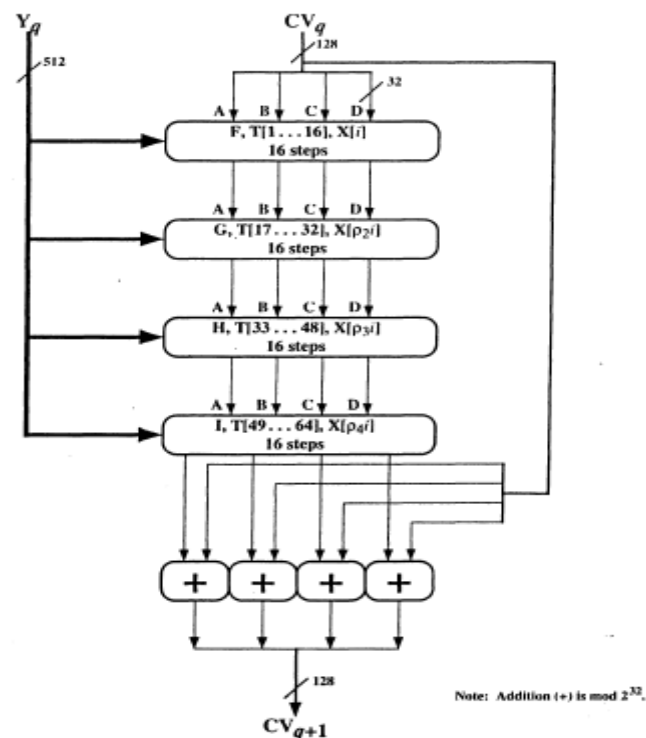
Each round uses 16 32-bit words by name  $X[i], X[p_2[i]], X[p_3[i]], X[p_4[i]]$  from the given 512 bit block.

$X[i] \rightarrow$  the first 16 32-bit words from the original 512-bit block.

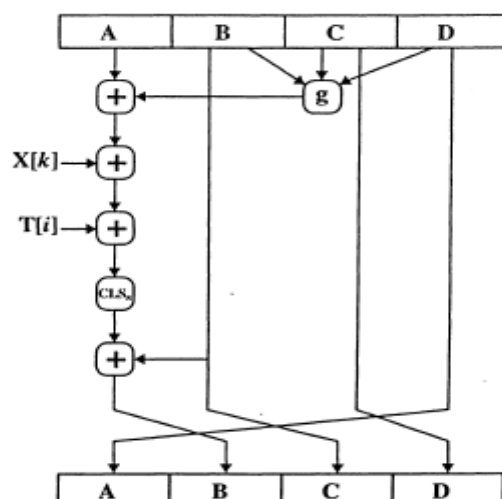
$X[p_2[i]] \rightarrow (1+5i) \bmod 16$  where  $i$  is the step count of that round

$X[p_3[i]] \rightarrow (5+3i) \bmod 16$

$X[p_4[i]] \rightarrow 7i \bmod 16$



**Details of Single Round:** since the algorithm uses little endian format, all operations are performed at low address register A.



1. A generator function  $g$  is calculated as

Round	Primitive function $g$	$g(b, c, d)$
1	$F(b, c, d)$	$(b \wedge c) \vee (b \wedge d)$
2	$G(b, c, d)$	$(b \wedge d) \vee (c \wedge d)$
3	$H(b, c, d)$	$b \oplus c \oplus d$
4	$I(b, c, d)$	$c \oplus (b \vee d)$

2. Apply integer modulo  $2^{32}$  addition on register A and on the output of function  $g$  which gives a 32-bit output.
3. The output of step 2 is taken as an input to  $\oplus$  with  $X[k]$  which yields again a 32-bit output.
4. Again, step 3 output is taken as an input to  $\oplus$  with  $T[i]$  and outputs a 32-bit output.
5. Apply circular left shift operation on the output of step 4 with  $s$  number of bits
6. Apply  $\oplus$  on the output of step 5 and with register B and copy this result to register B. Copy register B to C, C to D and D to A.

**Secure Hash Algorithm (SHA-I) :** The overall processing of message uses a variable length of message as input and outputs a 160 bit hash code. The input is processed in 512 bit blocks.

1. Append Padding bits: Message is padded so that its length is 448 modulo 512.
2. Append length: After padding the bits, the expanded message is represented as a sequence of 512 bit blocks  $Y_0, Y_1, Y_2, \dots, Y_{L-1}$ ; so that the total length of message is  $L \times 512$  bits.
3. Initialize MD Buffer: A 128 buffer is used to hold intermediate and final results of the hash function. This buffer can be represented as five 32-bit registers A, B, C, D and E. These registers are initialized to the following 32-bit integers.

Ex: A=67452301 , B=EFCDA89 , C=----- , D=----- , E=-----

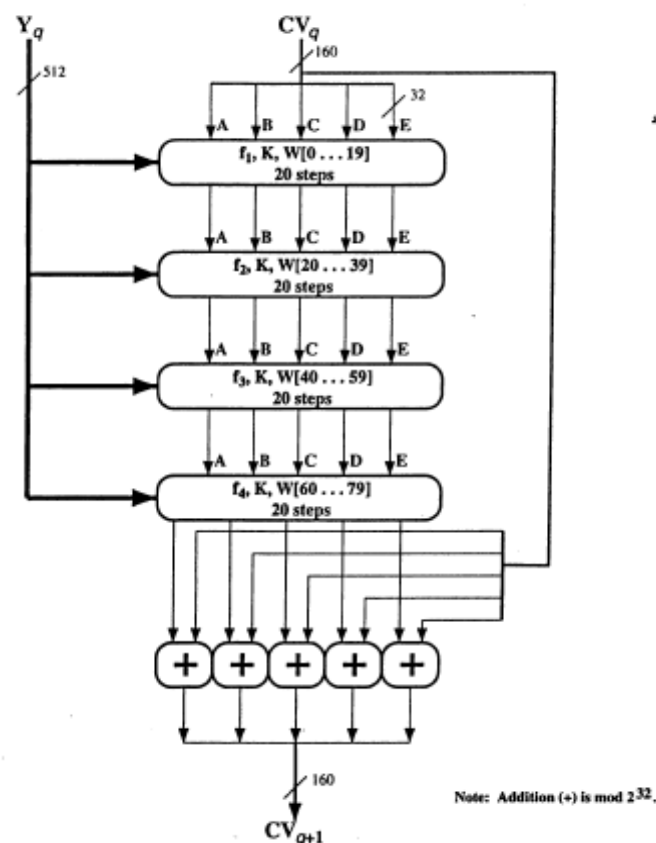
These values are stored in the big-endian format, means the most significant byte of a word is stored in the low address position.

That is word A=67 45 23 01, B=EF CD AB 89 , C=----- , D=----- , E=-----

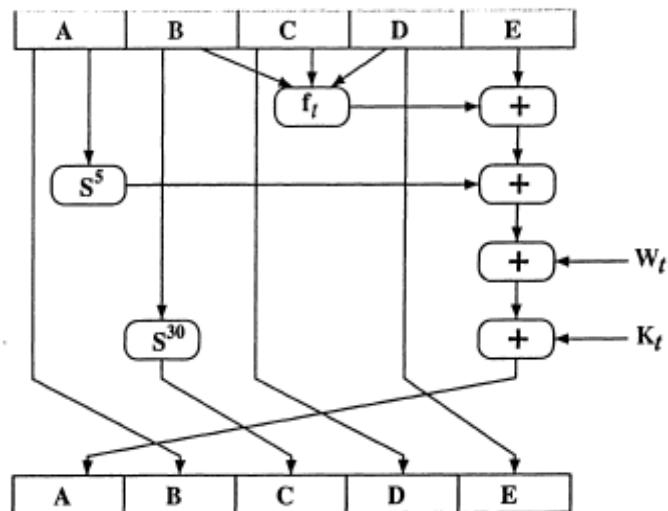
4. Process the message in 512 bit blocks: The four rounds have a similar structure, but each uses a different primitive logical function, referred as  $f_1, f_2, f_3$  and  $f_4$ . Each round has 20 steps.

Each round takes as input (512) block and uses an additive constant  $K_i$  where  $0 \leq i \leq 79$  indicates one of the 80 steps across four rounds. The following four distinct constants are used in each round and these values are represented in hexadecimal and decimal notation.

Step Number	Hexadecimal	Take Integer Part of:
$0 \leq i \leq 19$	$K_i = 5A827999$	$[2^{30} \times \sqrt{2}]$
$20 \leq i \leq 39$	$K_i = 6ED9EBA1$	$[2^{30} \times \sqrt{3}]$
$40 \leq i \leq 59$	$K_i = 8F1BBCDC$	$[2^{30} \times \sqrt{5}]$
$60 \leq i \leq 79$	$K_i = CA62C1D6$	$[2^{30} \times \sqrt{10}]$



**Details of single round:** since the algorithm uses Big-endian format all operations are performed at register E.



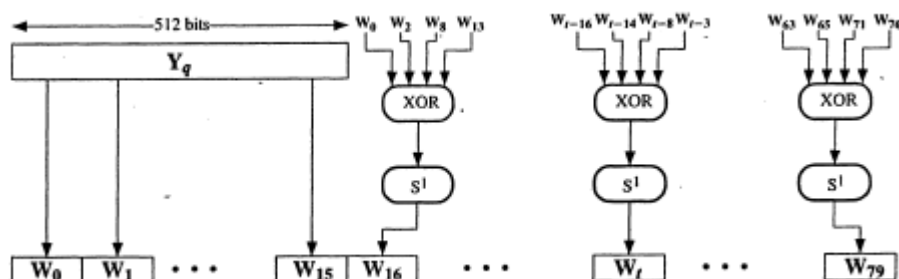
1. A generator function  $f_i$  is calculated as


Step	Function Name	Function Value
$(0 \leq i \leq 19)$	$f_1 = f(i, B, C, D)$	$(B \wedge C) \vee (B \wedge D)$
$(20 \leq i \leq 39)$	$f_2 = f(i, B, C, D)$	$B \oplus C \oplus D$
$(40 \leq i \leq 59)$	$f_3 = f(i, B, C, D)$	$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$
$(60 \leq i \leq 79)$	$f_4 = f(i, B, C, D)$	$B \oplus C \oplus D$

2. Apply integer modulo  $2^{32}$  addition on register E and on the output of function  $f_i$  which gives a 32-bit output.
3. Apply a circular left shift of 32 bit word A by k number of bits. On this output apply  $\oplus$  with the output of step 2.
4. Again, step 3 output is taken as an input to  $\oplus$  with wt.

$$W_t = S^l(W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3})$$

Where



5. Apply  on the output of step 4 and round additive constant  $K_t$  which is described in step 4 of architecture.
6. Copy the output of step 5 to register A, copy register A to B.

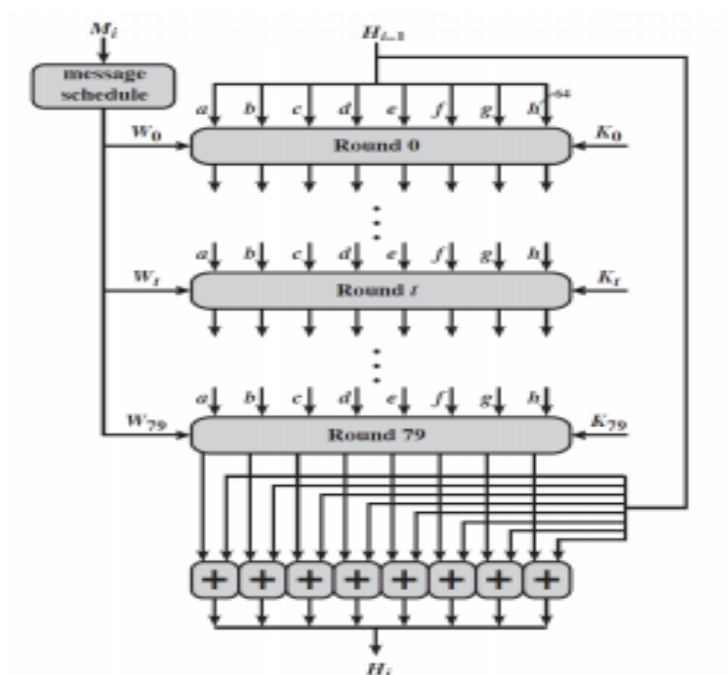
7. Apply circular left shift of 30 bits on Register B and copy it to register C, register C to D and D to E.

**SHA 512:** NIST issued revision FIPS 180-2 in 2002 adds 3 additional versions of SHA, SHA-256, SHA-384, and SHA-512 designed for compatibility with increased security provided by the AES cipher structure & detail is similar to SHA-1 hence analysis should be similar but security levels are rather higher.

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

#### Overview of SHA-512:

1. This algorithm process message in 1024-bit blocks
2. It has 80 rounds.



3. Each 1024 bit block is divided into 80 words of size 64 bits and are represented as  $W_t$ . where  $0 \leq t \leq 79$ .



- The first 16 words, that is  $w_0$  to  $w_{15}$  are directly generated from the given 1024 bit block by dividing it into 16 64-bit blocks.

- Remaining words that is  $w_{16}$  to  $w_{79}$  are generated using the formula

$$W_t = \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}$$

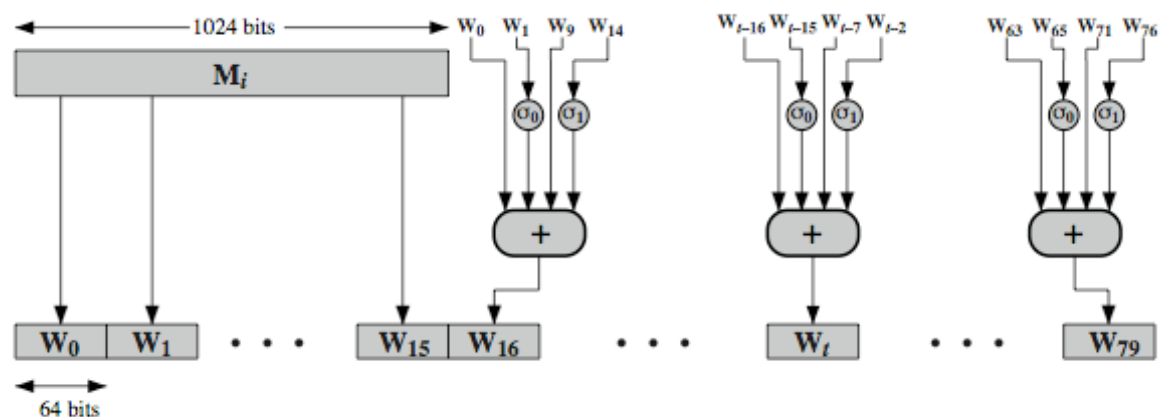
$$\sigma_0(x) = \text{ROTR1}(x) + \text{ROTR8}(x) + \text{SHR7}(x)$$

$$\sigma_1(x) = \text{ROTR19}(x) + \text{ROTR61}(x) + \text{SHR6}(x)$$

$\text{ROTR}_n(x)$  = rotate right by  $n$  bits

$\text{SHR}_n(x)$  = Left shift  $n$  bits with padding by 0's on the right

$+$  = Addition modulo  $2^{64}$



- The size of hash code is 512 bits.
- This algorithm uses 64-bit registers. Therefore it needs 8 64-bit registers by name A, B, C, D, E, F, G and H.
- Each round uses 80 different constants by calculating the cube roots of first 80 prime numbers and are denoted as  $K_t$ .

**SHA 512 Single Round Details:** All operations are performed on register H.

- Calculate two generator functions Maj and Ch on registers abc and efg respectively using the following formula



## Message Authentication Requirements:

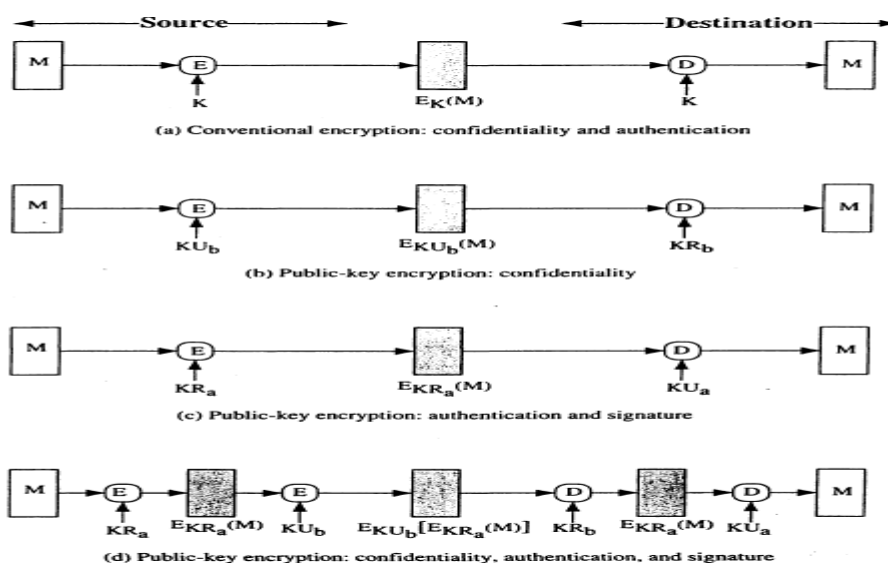
1. Disclosure: Release of message contents to any person or process not having the appropriate cryptographic key.
2. Traffic Analysis: Discovery of traffic between parties.
3. Masquerade: Insertion of messages into the network from a fraudulent source. This includes the creation of a message by an opponent that are purported to come from an authenticated entity.
4. Content Modification: Changes to the contents of a message, including insertion, deletion and recording.
5. Sequence Modification: Modifying sequence numbers between parties, including insertion deletion and recording.
6. Timing Modification: Delay or replay of messages.
7. Source repudiation: Denial of transmission of messages by source.
8. Destination repudiation: Denial of receipt of message by destination.

**Message Authentication Functions:** It is provided with three types of functions.

1. Message Encryption: The cipher text of entire message serves as its authenticator.
2. Message Authentication Code: A function of the message and a secret key that produces a fixed length value that serves as authenticator.
3. Hash Function: A function that maps a message of any length into a fixed length hash value which serves as authenticator.

## Message Encryption:

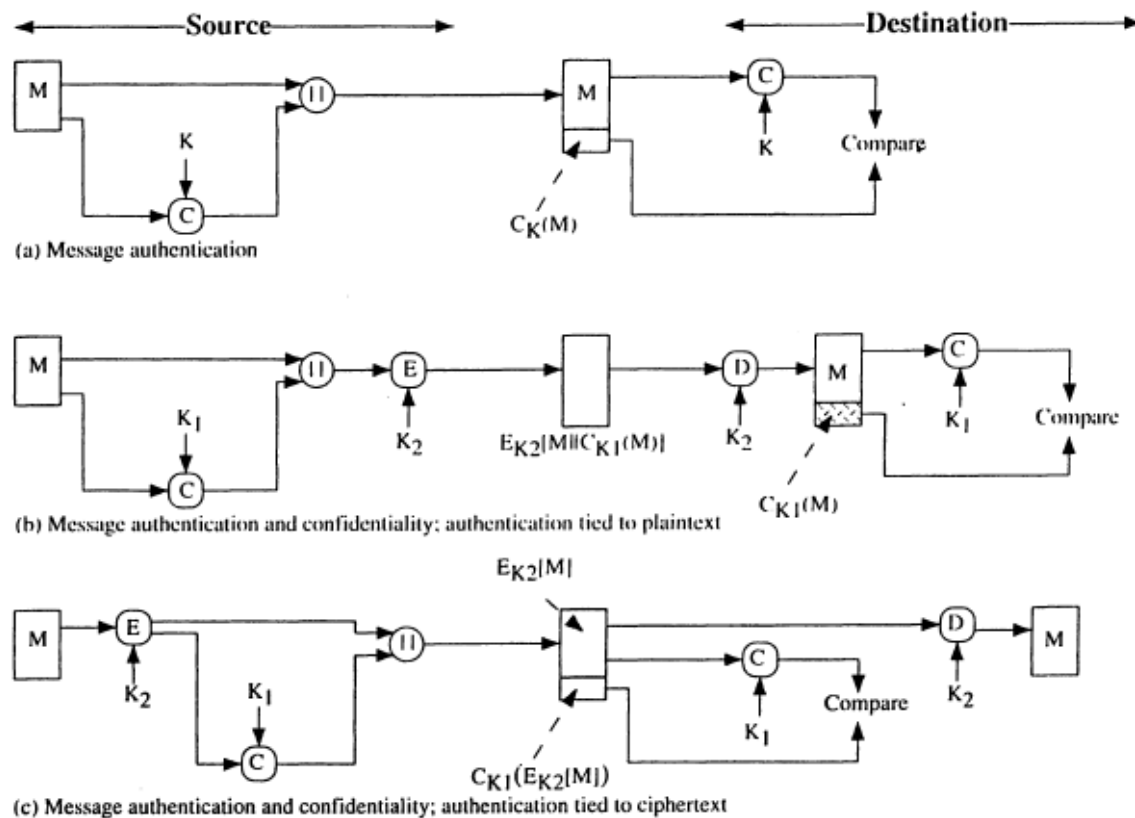
Message encryption can be provided with symmetric and asymmetric key cryptography. The following are the mechanisms to encrypt a message.



**Message Authentication Code:** It involves the use of secret key to generate a small fixed size block of data known as cryptographic checksum or MAC that is appended to the message.

$$\text{MAC} = C(K, M)$$

The messages plus MAC are transmitted to the intended recipient. The recipient performs the same calculation on the received message using the same secret key to generate a new MAC. The received MAC is compared to the calculated MAC. If both of them are equal, then receiver accepts the message, otherwise discards it.



**Security of MACs:** Just like encryption algorithm and hash functions, we can group attacks on MACs into two categories, brute force attacks and cryptanalysis.

1. **Brute-Force Attacks:** A brute force attack on a MAC is a more difficult undertaking rather than a brute-force attack on a hash function because it requires known message-tag pairs. To proceed with such kind of attacks, MAC algorithm must be computationally resistant. For example, take one-or more text MAC pairs  $[x, \text{MAC}(k, x)]$ , it is computationally infeasible to compute any text MAC pair  $[x, \text{MAC}(k, x)]$  for any new input  $x \neq x_i$
2. **Cryptanalysis:** Cryptanalytic attacks on MAC algorithm seek to exploit some property of the algorithm to perform some attack other than exhaustive search. The way to measure the resistance of a MAC algorithm to cryptanalysis is to compare its strength to the effort required

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

**HMAC:** A hash function such as SHA was not designed for use as a MAC and can not be used directly for that purpose, because it does not rely on a secret key. There have been number of proposals for the incorporation of a secret key into an existing hash function. HMAC is one such algorithm which has been issued NIST standard FIPAs 198. The following are the notations used in HMAC algorithm

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

IV=Initial value input to hash function

M=Message input to HMAC

$Y_i$ =ith block of M  $0 \leq i \leq L-1$

L=number of blocks in M

b=number of bits in a block

n=length of hash code produced by embedded hash function

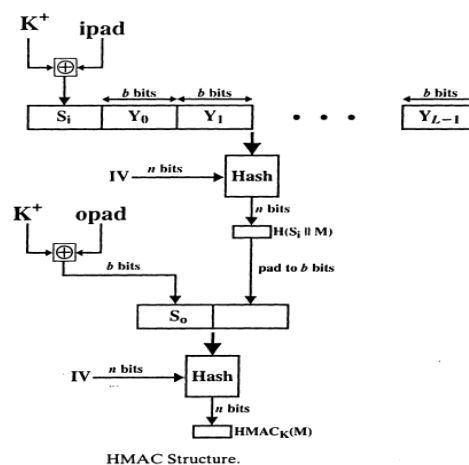
K=secret key of length  $\geq n$

$K^+$ =K padded with 0's on the left so that the result is b bits in length.

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

opad=0101100(5C in hexadecimal) repeated for b/8 number of times

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



### Algorithm:

1. Append zeros to the left end of the K to create a b bit string  $K^+$
2. XOR  $K^+$  with ipad to produce the b bit block  $S_i$
3. Append M to  $S_i$
4. Apply H to the stream generated in step 3
5. XOR  $K^+$  with opad to produce the b bit block  $S_o$

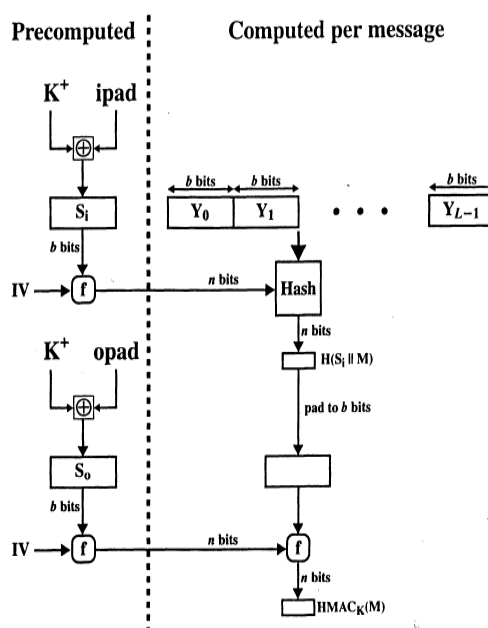
6. Append the hash result from step 4 to  $S_0$
7. Apply  $H$  to the stream generated in step 6 and output the result.

A more efficient implementation is possible by using the following two pre computed values

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

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

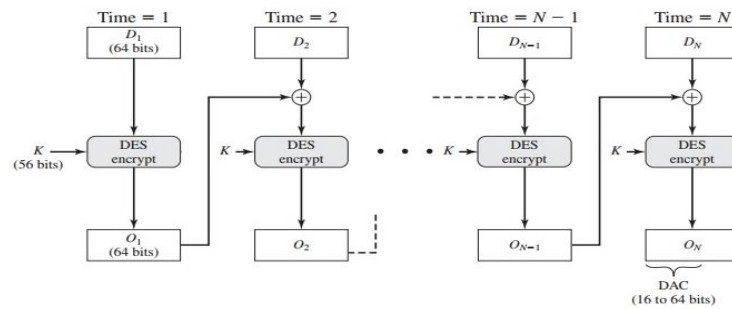
These quantities only need to be computed initially and every time the key changes. In effect the pre computed quantities substitute for the initial value (IV) in the function.



## MACs based on Block Ciphers : DAA and CMAC

**DAA:** Data Authentication Algorithm is based on DES. It is one of the most widely used MACs for a number of years. The algorithm can be defined as using the cipher block chaining (CBC) mode of operation of DES with an initialization vector of zeros. The data to be authenticated are grouped into contiguous 64-bit blocks  $D_1, D_2, \dots, D_N$ . If necessary the final block is padded on the right with zeroes to form a full 64 bit block. Using the DES encryption algorithm  $E$  and a secret key  $K$ , a data authentication code (DAC) is calculated as follows.

$$\begin{aligned}
 O_1 &= E(K, D) \\
 O_2 &= E(K, [D_2 \oplus O_1]) \\
 O_3 &= E(K, [D_3 \oplus O_2]) \\
 &\vdots \\
 O_N &= E(K, [D_N \oplus O_{N-1}])
 \end{aligned}$$



**Cipher based Message Authentication Code (CMAC):** This algorithm works with either AES or triple DES. The operation of the algorithm is as follows.

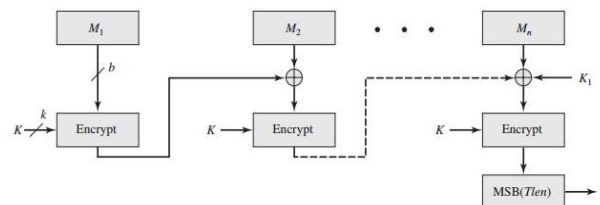
- The message must be an integer multiple  $n$  of the cipher block length  $b$
- For AES,  $b=128$ , and for triple DES,  $b=64$ .
- The message is divided into  $n$  blocks ( $M_1, M_2, M_3, \dots, M_n$ )
- Algorithm makes use of a  $k$  bit encryption key  $k$  and a  $b$  bit constant,  $k_1$ .
- For AES, the key size  $k$  is 128, 192, 256 bits; for triple DES, the key size is 112 or 168 bits.

CMAC is calculated as follows:

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

where

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

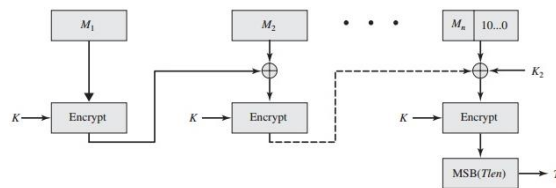


(a) Message length is integer multiple of block size

If the message length is not an integer multiple of the cipher block length, then the final block is padded to the right with a 1 and as many 0s as necessary so that the final block is also of length  $b$ . the CMA operation then proceeds as before, except that a different key  $K_2$  is used instead of  $K_1$ .

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

$$\begin{aligned}
 L &= E(K, 0^n) \\
 K_1 &= L \cdot x \\
 K_2 &= L \cdot x^2 = (L \cdot x) \cdot x
 \end{aligned}$$



Where multiplication (  $\cdot$  ) is done in the finite field  $GF(2^b)$ . To generate  $k_1$  and  $k_2$ , the block cipher is applied to the block that consists entirely 0 bits. The first sub key is derived from the resulting cipher text by a left shift of one bit and conditionally by xoring a constant that depends on the block size. The second sub key is derived in the same manner from the first sub key.

## Digital Signatures:

Message authentication protects two parties who exchange messages from any third party. however it does not protect the two parties against each other. Several forms of disputes are possible between the two parties.

1. A can forge a different message and claim that it came from B. A simple creates a message and append an authentication code using the key that A and B share.
2. B can deny sending the message because it is possible for A to forge a message, there is no way to prove that B had send the message.

In such type of situations where there is no complete trust between the two parties, something more than authentication is needed. The most attractive solution to this problem is digital signature. The digital signature is analogous to the handwritten signature and it must have the following properties.

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

Therefore digital signature also includes the authentication function. Based on these properties, the following requirements can be formulated for digital signatures.

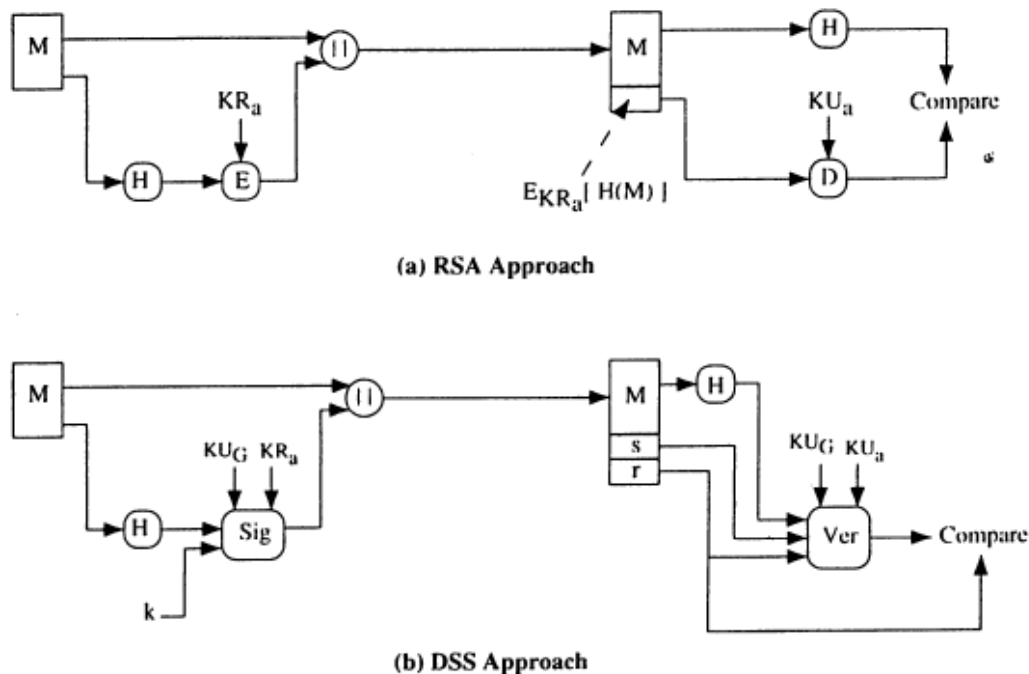
## Requirements:

- The signature must be a bit pattern
- The signature must use some personal information unique to the sender to prevent forgery and denial
- It must be relatively easy to produce digital signature
- It must be relatively easy to recognize and verify the digital signature
- It must be computationally infeasible to forge a digital signature
- It must be practical to retain a copy of digital signature



**Digital Signature Standard:** The National Institute of Standards and Technology has published Federal Information Processing Standard FIPS 186 known as Digital Signature Standard. The DSS makes use of the SHA.

**Two Approaches of DSS:**



Two Approaches to Digital Signatures.

**Digital Signature Algorithm:**

**A) Generation of global public key components:**

- Select a prime number  $p$  where  $2^{L-1} < p < 2^L$  for  $512 \leq L \leq 1024$  and  $L$  is a multiple of 64. That is bit length of between 512 and 1024 bits in increments of 64 bits.
- Select an integer  $q$  which is a prime divisor of  $(p-1)$ , where  $2^{159} < q < 2^{160}$  that is bit length of 160 bits
- Calculate global key  $g = h^{(p-1)/q} \mod p$  where  $h$  is any integer with  $1 < h < (p-1)$  such that  $h^{(p-1)/q} \mod p > 1$

**B) Generation of user's private and public keys**

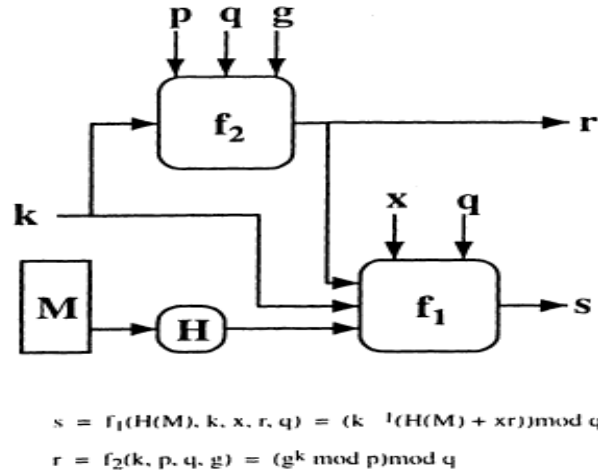
- Select an integer  $x$  where  $x$  is a pseudo random integer with  $0 < x < q$
- Calculate public key  $y = g^x \mod p$

**C) Generation of users per message secret number**

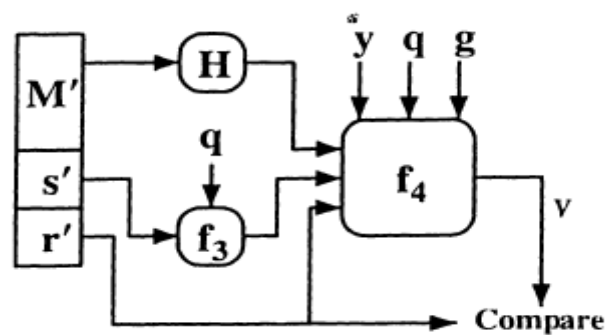
- Select pseudo random integer  $K$  where  $0 < k < q$

**D) Creation of Signature (by source)**

- Calculate  $r = (g^k \bmod p) \bmod q$
- Calculate  $s = [k^{-1} (H(M) + xr) \bmod q]$
- Signature =  $(r, s)$

**(a) Signing****E) Verification of signature (by receiver)**

- Calculate  $w = (s^{-1}) \bmod q$
- Calculate  $u_1 = [H(M') w] \bmod q$
- Calculate  $u_2 = (r^{-1}) w \bmod q$
- Calculate  $v = [(g^{u_1} y^{u_2}) \bmod p] \bmod q$
- Test whether  $v = r^{-1}$



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

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

$$= ((g^{u_1} y^{u_2}) \bmod p) \bmod q$$

**(b) Verifying**

**Remote User Authentication Principles:** User authentication is the fundamental building block and the primary line of defence in most security applications. It is the basis for many types of access control and user accountability. There are four general means of authenticating a user's identity, which can be used alone or in combination:

- **Something the individual knows:** Examples include a password, a personal identification number (PIN) or answers to a prearranged set of questions (security questions).
- **Something the individual possesses:** Examples include cryptographic keys, electronic key cards, smart cards and physical keys. This type of authentication is referred to as a token.
- **Something the individual is (static biometrics):** Examples include recognition by fingerprint, retina and face.
- **Something the individual does (dynamic biometrics):** Examples include recognition by voice pattern, handwriting characteristics, and typing rhythm.

All of these methods, properly implemented and used, can provide secure user authentication. Remote user authentication is of two ways.

1. **Mutual Authentication:** Mutual authentication protocols enable communicating parties to satisfy themselves mutually about each other's identity and to exchange session keys. Any authentication key exchange deals with two issues: confidentiality and timeliness. To prevent masquerading and compromise of session keys, identification details and session key must be communicated in an encrypted form. The second issue timeliness is important because of the threat of message replays. The following are the few examples of replay attacks:
  - a) **Simple Replay:** Opponent simply copies a message and replays it.
  - b) **Repetition that can be logged:** Opponent can replay a time stamped message within the valid time window.
  - c) **Repetition that cannot be detected:** Original message could have been suppressed and thus did not arrive at the destination only the replayed message arrives.
  - d) **Backward replay without modification:** This is a replay back to the sender

One approach to overcome with replay attacks is to use sequence numbers. A new message is accepted if its sequence number is in the proper order. But it requires each party to keep track of the last sequence number. Because of this overhead, sequence numbers are not generally used for authentication and key exchange. Instead, one of the following two approaches is used:

- Timestamps: A accepts a message as fresh only if the message contains a time stamp. This approach requires that clocks among the various participants be synchronized.
- Challenge/response: A expects a fresh message from B, first sends B a nonce(challenge) and requires that the subsequent message received from B contain the correct nonce value.

Timestamp approach should not be used for connection oriented applications because of the difficulties with this technique. On the other hand, the challenge-response approach is unsuitable for a connectionless type of application because it requires the overhead of a handshake before any connectionless transmission.

## 2. One-way Authentication:

**Kerberos:** Kerberos is an authentication service developed as part of project Athena at MIT. The problem that Kerberos addresses is: users at workstations wish to access services on servers distributed throughout the network. We would like for servers to be able to restrict access to

authorized users and to be able to authenticate requests for service. Therefore, in this situation a workstation cannot be trusted to identify its users correctly to network services. Therefore the following three threats may exist:

- A user may gain access to a particular system and pretend to be another user working from that system
- A user may alter the network address of a system.
- A user may eavesdrop on exchanges and use a replay attack to gain entrance to server or to disrupt operations.

In any of these cases, an unauthorized user may be able to gain access to services and data that he or she is not authorized to access. Instead of building elaborate authentication protocols at each server, Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users. Two versions of Kerberos are in common use. Version 4 is still widely used. Version 5 corrects some of the security deficiencies of version 4 and has been issues as a draft standard.

Kerberos Version 4: This version makes use of DES, to provide the authentication service. Each successive dialogue adds additional complexity to counter security vulnerabilities revealed in the preceding dialogue.

A Simple Authentication Dialogue: In an unprotected environment, any client can apply to any server for service. An opponent can pretend to be another client and obtain unauthorized privileges on server machines. To counter this threat, servers must confirm the identities of clients who request the service. But in an open environment it is possible.

Therefore, the alternative is to use authentication server(AS) that knows the passwords of all users and stores these in a centralized database. The authentication server shares unique secret key with each server. These key are physically distributed or in some secure manner.

$$\begin{aligned}
 (1) \quad C &\rightarrow AS: && ID_C || P_C || ID_V \\
 (2) \quad AS &\rightarrow C: && Ticket \\
 (3) \quad C &\rightarrow V: && ID_C || Ticket \\
 Ticket &= E_{K_V}[ID_C || AD_C || ID_V]
 \end{aligned}$$

where

<b>C</b>	=	client
<b>AS</b>	=	authentication server
<b>V</b>	=	server
<b>ID<sub>C</sub></b>	=	identifier of user on C
<b>ID<sub>V</sub></b>	=	identifier of V
<b>P<sub>C</sub></b>	=	password of user on C
<b>AD<sub>C</sub></b>	=	network address of C
<b>K<sub>V</sub></b>	=	secret encryption key shared by AS and V
<b>  </b>	=	concatenation

1. Client requests the AS to facilitate services of the server ID<sub>V</sub> by supplying his ID<sub>C</sub> and server id ID<sub>V</sub> and his password
2. The AS checks its database to if the user has supplied proper password and is he permitted to access this server. If the user enters a correct password and he is allowed to access this server AS generates a TICKET for client to access the server ID<sub>V</sub>. The AS also convinces the server by supplying following details in the ticket: I am issuing a ticket for this client ID<sub>C</sub> and he is requested me to access you, ID<sub>V</sub>(server) and his network address is AD<sub>C</sub>. AS also encrypts these details to avoid forgery or modification using server's master key E<sub>K<sub>V</sub></sub>.
3. After receiving this certificate client forwards his id ID<sub>C</sub> and Ticket to the server V. Upon receiving this certificate, server decrypts the ticket and verifies allows the client to avail its services.

A more secure Authentication Dialogue: The above scenario addresses two problems

1. To establish a connection with a server client has to request AS. This procedure need to be repeated for each server. So here, we minimize the number of time that a user has to enter the details: ID<sub>C</sub> || P<sub>C</sub> || ID<sub>V</sub>
2. In the previous scenario, password P<sub>C</sub> is transmitted in a plain text. An eavesdropper could capture the password and use any services available to the victim.

To solve these two problems, here a new scheme is introduced for avoiding plain text passwords, and a new server known as Ticket Granting Server.

**Once per user logon session:**

- (1) C → AS: ID<sub>C</sub> || ID<sub>TGS</sub>
- (2) AS → C: E<sub>K<sub>C</sub></sub>[Ticket<sub>TGS</sub>]

**Once per type of service:**

- (3) C → TGS: ID<sub>C</sub> || ID<sub>V</sub> || Ticket<sub>TGS</sub>
- (4) TGS → C: Ticket<sub>V</sub>

**Once per service session:**

$$\begin{aligned}
 (5) \quad C &\rightarrow V: && ID_C || Ticket_V \\
 Ticket_{TGS} &= E_{K_{TGS}}[ID_C || AD_C || ID_{TGS} || TS_1 || Lifetime_1] \\
 Ticket_V &= E_{K_V}[ID_C || AD_C || ID_V || TS_2 || Lifetime_2]
 \end{aligned}$$

The new service, TGS issues tickets to users who have been authenticated to AS. Thus the client first requests a ticket-granting ticket( $Ticket_{TGS}$ ) from the AS. Client saves this ticket in his terminal. Each time the user requires access to a new service, the client applies to the TGS, using the ticket to authenticate himself. The TGS then grants a ticket for the particular service.

1. The client request a ticket granting ticket on behalf of the user by sending its user ID to the AS.
2. The AS responds with a ticket that is encrypted with a client master key which is calculated from the user's password. When the response arrives to the client, the client prompts the user for this or her password, generates the key and attempts to decrypt the incoming message. If the correct password is supplied the ticket is successfully recovered.
3. The client requests a service granting ticket on behalf of the user. For this purpose, the client transmits a message to the TGS containing the user's ID, the ID of the desired service and the Ticket granting ticket.
4. The TGS decrypts the incoming ticket and verifies the success of the decryption by the presents of its ID and also verifies the lifetime of ticket. Then it compares the user ID and network address with the incoming information to authenticate the user. If the user is permitted to access server V, then TGS issues a ticket to grant access to the requested service.
5. The client requests access to a service on behalf of user by supplying his user ID and service granting ticket. The server authenticated by using the contents of the ticket.

Kerberos Version 4 Authentication Dialogue: The above scenario enhances security but still there are two problems:

1. Life time associated with the ticket granting ticket. If this lifetime is very short(minutes) then the user will be repeatedly asked for entering a password. If the lifetime is too long(hours) then an opponent have a chance to replay the messages. An opponent can eavesdrop and copy a ticket and can wait for a legitimate user till he login to the session.
2. There may a requirement for servers to authenticate themselves to users. Without such authentication, an opponent could sabotage the configuration so that the messages to the server are

directed to another location. Then the false server can then be in a position to act as a real server and capture any information from the user and deny the true service to the user.

Version 4 of Kerberos protocol addresses these two problems by facilitating Timestamp(TS) and Lifetime(LT) values in the tickets, and like servers, clients also authenticate themselves (Authenticator<sub>c</sub>) using the shared secret keys between the client and servers.

#### Summary of Kerberos Version 4 Message Exchanges

(a) Authentication Service Exchange: to obtain ticket-granting ticket	
(1) C → AS:	$ID_c    ID_{tgs}    TS_1$
(2) AS → C:	$E_{K_c} [K_{c,tgs}    ID_{tgs}    TS_2    Lifetime_2    Ticket_{tgs}]$
	$Ticket_{tgs} = E_{K_{tgs}} [K_{c,tgs}    ID_c    AD_c    ID_{tgs}    TS_2    Lifetime_2]$
(b) Ticket-Granting Service Exchange: to obtain service-granting ticket	
(3) C → TGS:	$ID_v    Ticket_{tgs}    Authenticator_c$
(4) TGS → C:	$E_{K_{c,tgs}} [K_{c,v}    ID_v    TS_4    Ticket_v]$
	$Ticket_{tgs} = E_{K_{tgs}} [K_{c,tgs}    ID_c    AD_c    ID_{tgs}    TS_2    Lifetime_2]$
	$Ticket_v = E_{K_v} [K_{c,v}    ID_c    AD_c    ID_v    TS_4    Lifetime_4]$
	$Authenticator_c = E_{K_{c,tgs}} [ID_c    AD_c    TS_3]$
(c) Client/Server Authentication Exchange: to obtain service	
(5) C → K:	$Ticket_v    Authenticator_c$
(6) K → C:	$E_{K_{c,v}} [TS_5 + 1]$ (for mutual authentication)
	$Ticket_v = E_{K_v} [K_{c,v}    ID_c    AD_c    ID_v    TS_4    Lifetime_4]$
	$Authenticator_c = E_{K_{c,v}} [ID_c    AD_c    TS_5]$

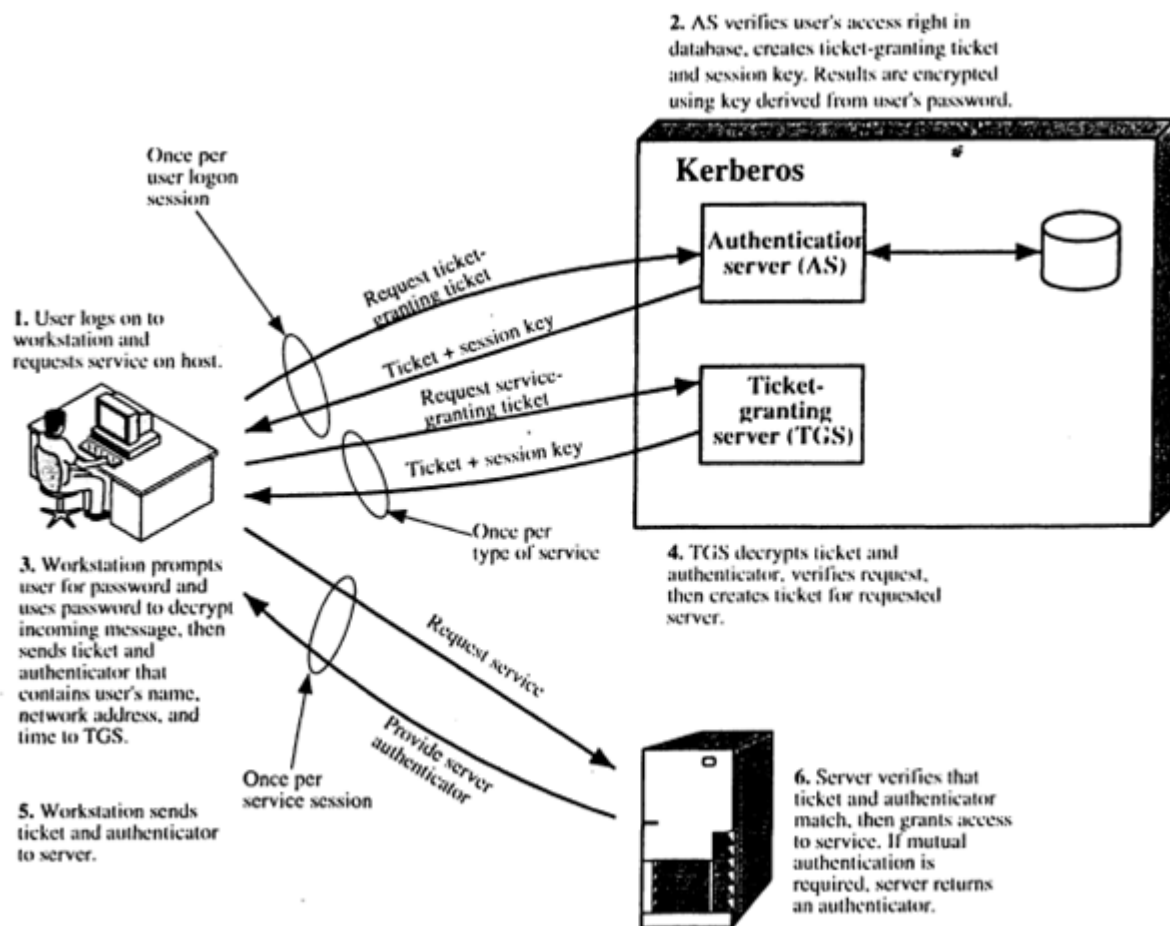
Overview of Kerberos:

#### Rationale for the Elements of the Kerberos Version 4 Protocol

(a) Authentication Service Exchange	
Message (1)	Client requests ticket-granting ticket
$ID_c$ :	Tells AS identity of user from this client
$ID_{tgs}$ :	Tells AS that user requests access to TGS
$TS_1$ :	Allows AS to verify that client's clock is synchronized with that of AS
Message (2)	AS returns ticket-granting ticket
$E_{K_c}$ :	Encryption is based on user's password, enabling AS and client to verify password, and protecting contents of message (2)
$K_{c,tgs}$ :	Copy of session key accessible to client; created by AS to permit secure exchange between client and TGS without requiring them to share a permanent key
$ID_{tgs}$ :	Confirms that this ticket is for the TGS
$TS_2$ :	Informs client of time this ticket was issued
$Lifetime_2$ :	Informs client of the lifetime of this ticket
$Ticket_{tgs}$ :	Ticket to be used by client to access TGS



(b) Ticket-Granting Service Exchange	
Message (3)	Client requests service-granting ticket
$ID_V$ :	Tells TGS that user requests access to server V
$Ticket_{TGS}$ :	Assures TGS that this user has been authenticated by AS
$Authenticator_c$ :	Generated by client to validate ticket
Message (4)	TGS returns service-granting ticket
$E_{K_{c,TGS}}$ :	Key shared only by C and TGS; protects contents of message (4)
$K_{c,TGS}$ :	Copy of session key accessible to client; created by TGS to permit secure exchange between client and server without requiring them to share a permanent key
$ID_V$ :	Confirms that this ticket is for server V
$TS_4$ :	Informs client of time this ticket was issued
$Ticket_V$ :	Ticket to be used by client to access server V
$Ticket_{TGS}$	Reusable so that user does not have to reenter password
$E_{K_{TGS}}$ :	Ticket is encrypted with key known only to AS and TGS, to prevent tampering
$K_{c,TGS}$ :	Copy of session key accessible to TGS; used to decrypt authenticator, thereby authenticating ticket
$ID_c$ :	Indicates the rightful owner of this ticket
$AD_c$ :	Prevents use of ticket from workstation other than one that initially requested the ticket
$ID_{TGS}$ :	Assures server that it has decrypted ticket properly
$TS_2$ :	Informs TGS of time this ticket was issued
$Lifetime_2$ :	Prevents replay after ticket has expired
$Authenticator_c$ :	Assures TGS that the ticket presenter is the same as the client for whom the ticket was issued; has very short lifetime to prevent replay
$E_{K_{c,TGS}}$ :	Authenticator is encrypted with key known only to client and TGS, to prevent tampering
$ID_c$ :	Must match ID in ticket to authenticate ticket
$AD_c$ :	Must match address in ticket to authenticate ticket
$TS_2$ :	Informs TGS of time this authenticator was generated
(c) Client/Server Authentication Exchange	
Message (5)	Client requests service
$Ticket_V$ :	Assures server that this user has been authenticated by AS
$Authenticator_c$ :	Generated by client to validate ticket
Message (6)	Optional authentication of server to client
$E_{K_{c,V}}$ :	Assures C that this message is from V
$TS_5 + 1$ :	Assures C that this is not a replay of an old reply
$Ticket_V$	Reusable so that client does not need to request a new ticket from TGS for each access to the same server
$E_{K_V}$ :	Ticket is encrypted with key known only to TGS and server, to prevent tampering
$K_{c,V}$ :	Copy of session key accessible to client; used to decrypt authenticator, thereby authenticating ticket
$ID_c$ :	Indicates the rightful owner of this ticket
$AD_c$ :	Prevents use of ticket from workstation other than one that initially requested the ticket
$ID_V$ :	Assures server that it has decrypted ticket properly
$TS_4$ :	Informs server of time this ticket was issued
$Lifetime_4$ :	Prevents replay after ticket has expired
$Authenticator_c$ :	Assures server that the ticket presenter is the same as the client for whom the ticket was issued; has very short lifetime to prevent replay
$E_{K_{c,V}}$ :	Authenticator is encrypted with key known only to client and server, to prevent tampering
$ID_c$ :	Must match ID in ticket to authenticate ticket
$AD_c$ :	Must match address in ticket to authenticate ticket
$TS_5$ :	Informs server of time this authenticator was generated



### Kerberos Realms and Multiple Kerber:

A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers requires:

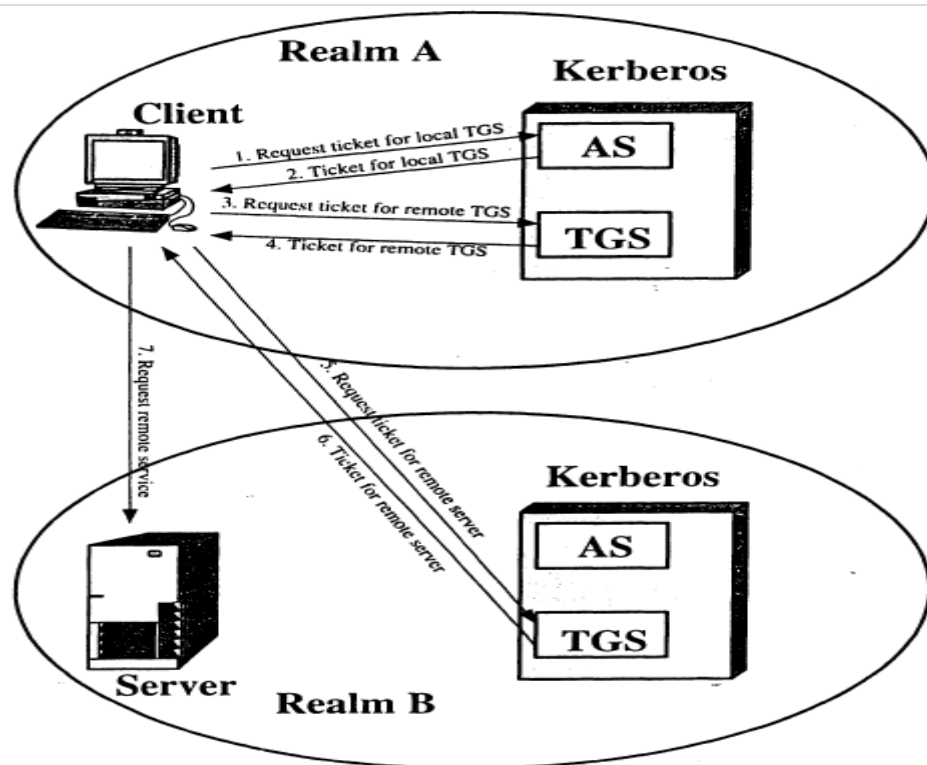
1. Kerberos server must have the user ID and hashed password of all participating users in its database. All users are registered with the Kerberos server.
2. The Kerberos server must have a secret key with each server. All servers are registered with the Kerberos server.

Such an environment is referred to as a realm. Networks of clients and servers under different administrative organizations typically constitute different realms. 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.

3. 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.

With all the above rules, we can describe the operation as follows: A user wishing service on a server in another realm needs a ticket for that server. The user's client follows the usual procedures to gain access to the local TGS and then requests a ticket-granting ticket for a remote TGS. The client can then apply to the remote TGS for a service granting ticket for the desired server in the realm of remote TGS.



Request for Service in Another Realm.

- (1)  $C \rightarrow AS:$   $ID_c || ID_{tgs} || TS_1$
- (2)  $AS \rightarrow C:$   $E_{K_c}[K_{c,tgs} || ID_{tgs} || TS_2 || Lifetime_2 || Ticket_{tgs}]$
- (3)  $C \rightarrow TGS:$   $ID_{tgsrem} || Ticket_{tgs} || Authenticator_c$
- (4)  $TGS \rightarrow C:$   $E_{K_{c,tgs}}[K_{c,tgsrem} || ID_{tgsrem} || TS_4 || Ticket_{tgsrem}]$
- (5)  $C \rightarrow TGS_{rem}:$   $ID_{vrem} || Ticket_{tgsrem} || Authenticator_c$
- (6)  $TGS \rightarrow C:$   $E_{K_{c,tgsrem}}[K_{c,vrem} || ID_{vrem} || TS_b || Ticket_{vrem}]$
- (7)  $C \rightarrow V_{rem}:$   $Ticket_{vrem} || Authenticator_c$

Kerberos Version 5 :

Differences between Versions 4 and 5:

Version 5 intended to address the limitations of version 4 in two area. Environmental shortcomings and technical deficiencies.

Environmental shortcomings:

1. Encryption System dependence: Version 4 requires the use of DES. Version 5 uses any encryption scheme.
2. Internet Protocol dependence: Version 4 requires the use of Internet Protocol addresses. Other address types such as ISO addresses are not accommodated. Whereas version 5 allows all types of network addresses.
3. Message byte ordering: In version 4, the sender of a message employs a byte ordering whereas in version 5, all message structures are defined using Abstract Syntax Notation One(ASN) and Basic Encoding Rules.
4. Ticket Lifetime: Lifetime values in version 4 are encoded in an 8-bit quantity in units of 5 minutes. Thus the maximum lifetime is  $2^8 \times 5 = 1280 \text{ mins} = 21 \text{ hours}$ . This may be inadequate for some applications. Inversion 5 tickets include an explicit start time and end time, allowing tickets with arbitrary lifetimes.
5. Authentication Forwarding: version 4 does not allow credentials issued to one client to be forwarded to some other host and used by some other client. Version 5 provides this capability.
6. Inter-realm authentication: In version 4 interoperability among N realms requires  $N^2$  Kerberos-to-Kerberos relationships. Version 5 supports a method that requires fewer relationships.

Apart from these environmental limitations, there are some technical deficiencies in version 4.

Technical Deficiencies:

1. Double encryption: Tickets provided to clients are encrypted twice, once with the secret key of the target server and then again with a secret key known to the client. The second encryption is not necessary and is computationally wasteful.
2. PCBC encryption: Encryption in version 4 makes use of a nonstandard mode of DES known Propagating Cipher Block Chaining mode. Version 5 makes use of CBC mode to be used for encryption.
3. Session Keys: Each ticket includes a session key that is used by the client to encrypt the authenticator sent to the service associated with that ticket. In version 5, it is possible for a client and server to negotiate a sub session key which is to be used for only one connection.

4. Password attacks: Both versions are vulnerable to password attacks.

Version 5 Authentication Dialogue:

- **Realm:** Indicates realm of user
- **Options:** Used to request that certain flags be set in the returned ticket, as explained below
- **Times:** Used by the client to request the following time settings in the ticket:

<b>(a) Authentication Service Exchange: to obtain ticket-granting ticket</b>	
(1) C → AS:	Options    ID <sub>c</sub>    Realm <sub>c</sub>    ID <sub>tgs</sub>    Times    Nonce <sub>1</sub>
(2) AS → C:	Realm <sub>c</sub>    ID <sub>c</sub>    Ticket <sub>tgs</sub>    E <sub>K<sub>c,tgs</sub></sub> [K <sub>c,tgs</sub>    Times    Nonce <sub>1</sub>    Realm <sub>tgs</sub>    ID <sub>tgs</sub> ] Ticket <sub>tgs</sub> = E <sub>K<sub>tgs</sub></sub> [Flags    K <sub>c,tgs</sub>    Realm <sub>c</sub>    ID <sub>c</sub>    AD <sub>c</sub>    Times]
<b>(b) Ticket-Granting Service Exchange: to obtain service-granting ticket</b>	
(3) C → TGS:	Options    ID <sub>v</sub>    Times    Nonce <sub>2</sub>    Ticket <sub>tgs</sub>    Authenticator <sub>c</sub>
(4) TGS → C:	Realm <sub>c</sub>    ID <sub>c</sub>    Ticket <sub>v</sub>    E <sub>K<sub>c,tgs</sub></sub> [K <sub>c,v</sub>    Times    Nonce <sub>2</sub>    Realm <sub>v</sub>    ID <sub>v</sub> ] Ticket <sub>tgs</sub> = E <sub>K<sub>tgs</sub></sub> [Flags    K <sub>c,tgs</sub>    Realm <sub>c</sub>    ID <sub>c</sub>    AD <sub>c</sub>    Times] Ticket <sub>v</sub> = E <sub>K<sub>v</sub></sub> [Flags    K <sub>c,v</sub>    Realm <sub>c</sub>    ID <sub>c</sub>    AD <sub>c</sub>    Times] Authenticator <sub>c</sub> = E <sub>K<sub>c,tgs</sub></sub> [ID <sub>c</sub>    Realm <sub>c</sub>    TS <sub>1</sub> ]
<b>(c) Client/Server Authentication Exchange: to obtain service</b>	
(5) C → TGS:	Options    Ticket <sub>v</sub>    Authenticator <sub>c</sub>
(6) TGS → C:	E <sub>K<sub>c,v</sub></sub> [TS <sub>2</sub>    Subkey    Seq#] Ticket <sub>v</sub> = E <sub>K<sub>v</sub></sub> [Flags    K <sub>c,v</sub>    Realm <sub>c</sub>    ID <sub>c</sub>    AD <sub>c</sub>    Times] Authenticator <sub>c</sub> = E <sub>K<sub>c,v</sub></sub> [ID <sub>c</sub>    Realm <sub>c</sub>    TS <sub>2</sub>    Subkey    Seq#]

**X.509 Authentication Service:** ITU-T recommendation X.509 is part of the X.500 series of recommendations that define a directory service.

**Certificates:** The heart of the X.509 scheme is the public key certificate associated with each user. These user certificates are assumed to be created by some trusted certification authorities. The public key certificate contains the following items:

**Version:** This field represents the public key certificate version

**Serial Number:** This is an integer value unique within the issuing CA that is associated in the certificate.

**Signature Algorithm Identifier:** This is the algorithm used to sign the certificate

**Issuer Name:** X.500 name of Certification authority that created and signed this certificate

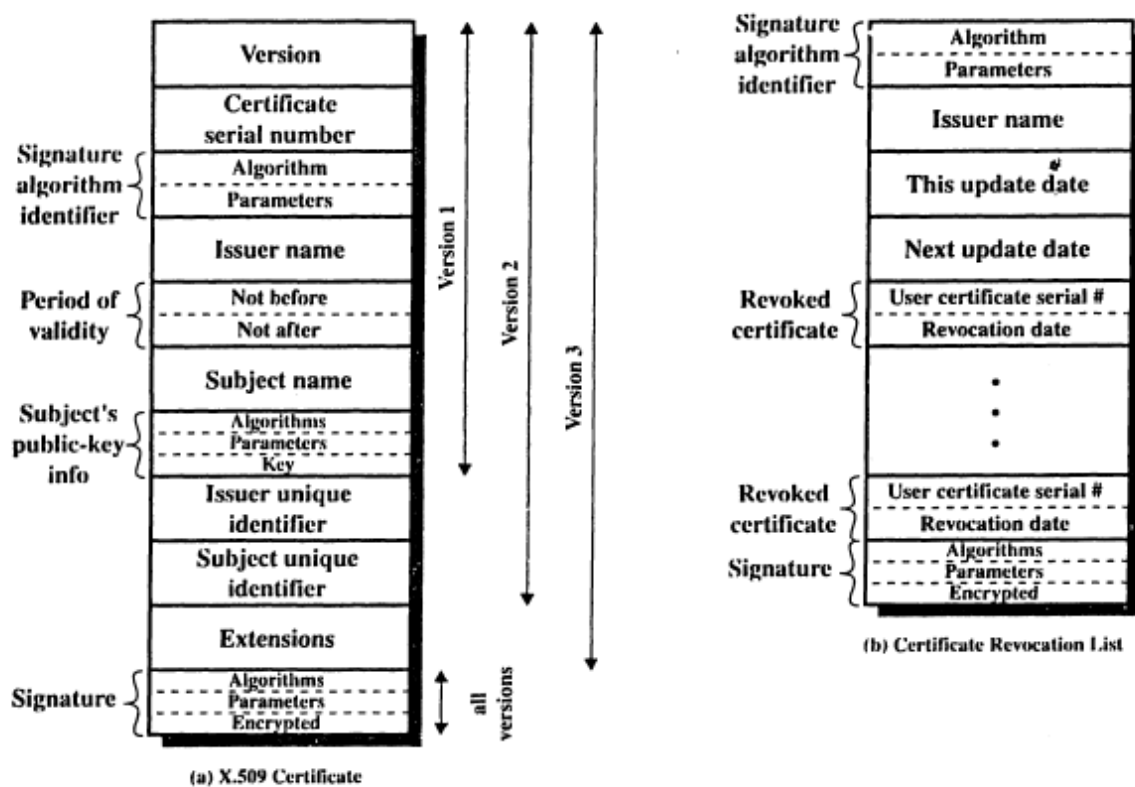
**Period of Validity:** Represents the validity period (from date and till date) of the certificate

**Subject name:** This is the name of the owner of the public key certificate

**Subjects Public Key Information:** Gives information about public key and its key id

**Issuer Unique Identifier:** An optional bit string field used to identify uniquely the issuing CA

**Subject Unique Identifier:** An optional bit string field used to identify the owner uniquely



X.509 Formats.

Extensions A set of one or more extension fields which are added in version 3

Signature: Calculates a hash code on all of the other fields of the certificate and encrypts this hash code with a private key of CA.

