UNIT-3

Message Authentication

Message authentication is a procedure to verify that received messages come from the alleged source and have not been altered. Message authentication may also verify sequencing and timeliness. It is intended against the attacks like content modification, sequence modification, timing modification and repudiation. For repudiation, concept of digital signatures is used to counter it. There are three classes by which different types of functions that may be used to produce an authenticator. They re:

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

MESSAGE ENCRYPTION:

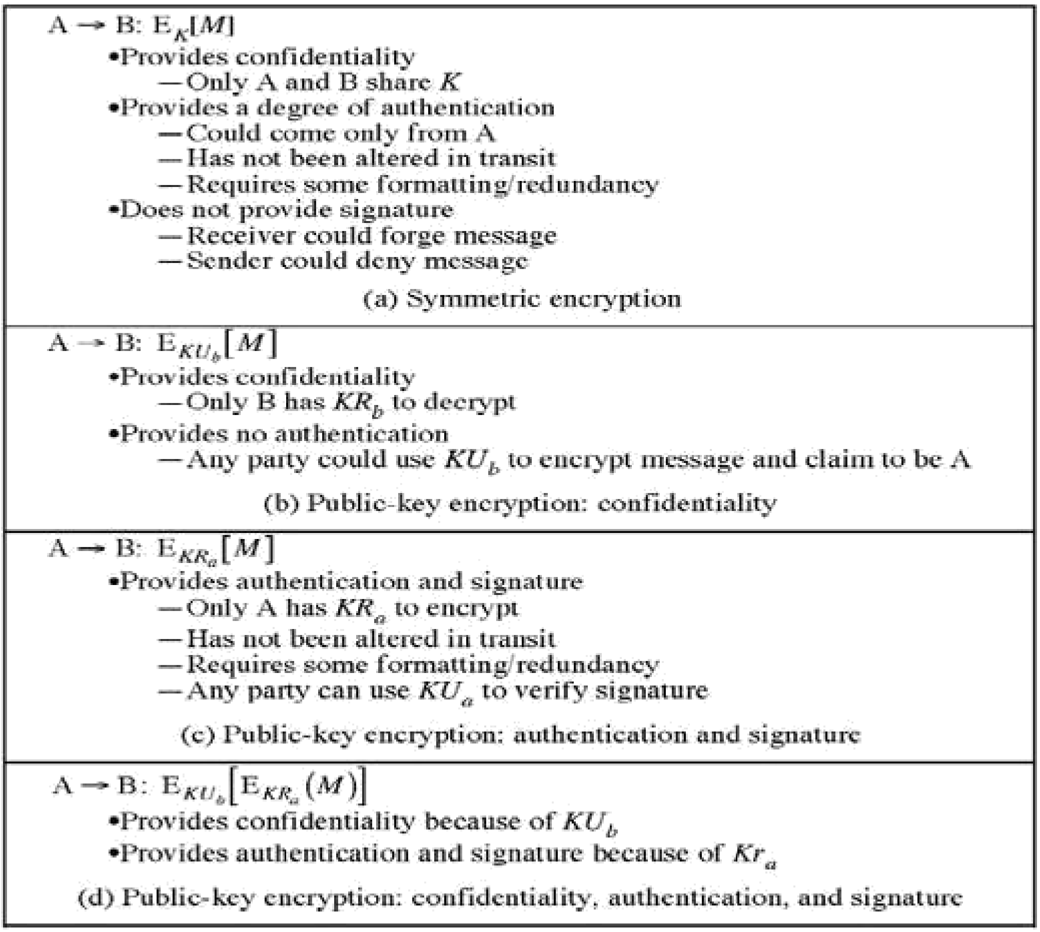
Message encryption by itself can provide a measure of authentication. The analysis differs for conventional and public-key encryption schemes. The message must have come from the sender itself, because the ciphertext can be decrypted using his (secret or public) key. Also, none of the bits in the message have been altered because an opponent does not know how to manipulate the bits of the ciphertext to induce meaningful changes to the plaintext. Often one needs alternative authentication schemes than just encrypting the message.

- Sometimes one needs to avoid encryption of full messages due to legal requirements.

- Encryption and authentication may be separated in the system architecture.

The different ways in which message encryption can provide authentication, confidentiality in both symmetric and asymmetric encryption techniques is explained with the table below:

Confidentiality and Authentication Implications of Message Encryption



A → B: $E_K[M]$
- Provides confidentiality
  - —Only A and B share K
- Provides a degree of authentication
  - —Could come only from A
  - —Has not been altered in transit
  - —Requires some formatting/redundancy
- Does not provide signature
  - —Receiver could forge message
  - —Sender could deny message

(a) Symmetric encryption

A → B: $E_{KU_b}[M]$
- Provides confidentiality
  - —Only B has KR_b to decrypt
- Provides no authentication
  - —Any party could use KU_b to encrypt message and claim to be A

(b) Public-key encryption: confidentiality

A → B: $E_{KR_a}[M]$
- Provides authentication and signature
  - —Only A has KR_a to encrypt
  - —Has not been altered in transit
  - —Requires some formatting/redundancy
  - —Any party can use KU_a to verify signature

(c) Public-key encryption: authentication and signature

A → B: $E_{KU_b}[E_{KR_a}(M)]$
- Provides confidentiality because of KU_b
- Provides authentication and signature because of Kr_a

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

MESSAGE AUTHENTICATION CODE

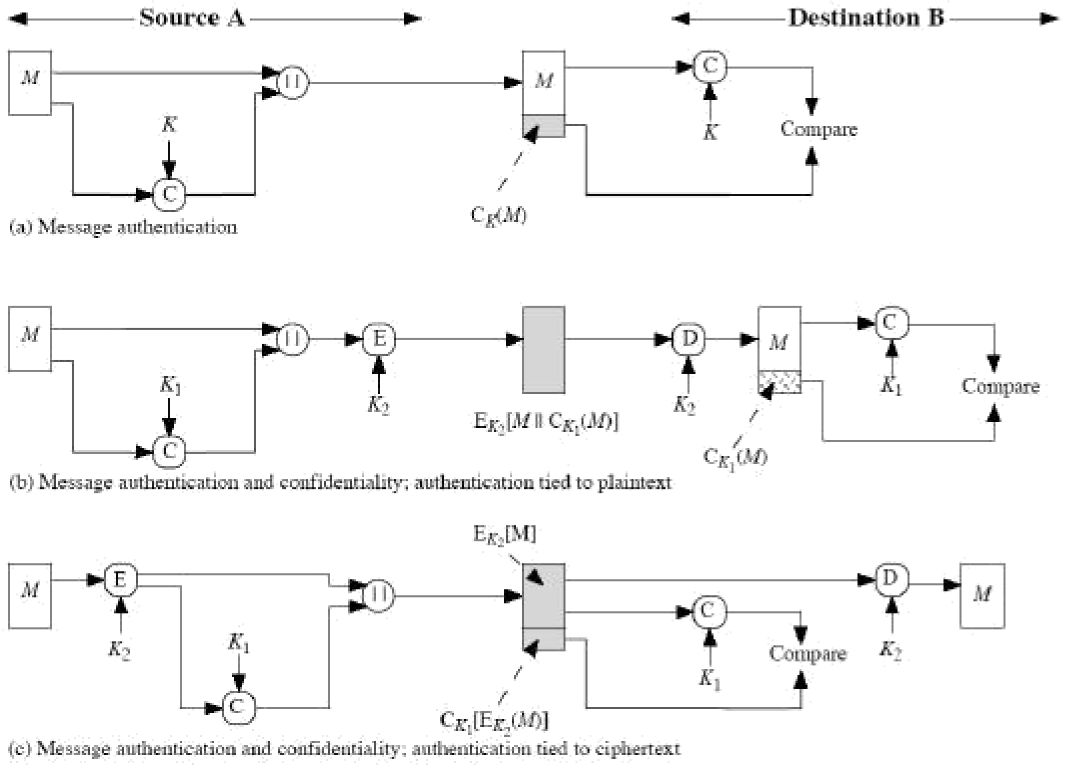
An alternative authentication technique involves the use of a secret key to generate a small fixed-size block of data, known as cryptographic checksum or MAC, which is appended to the message. This technique assumes that both the communicating parties say A and B share a common secret key K. When A has a message to send to B, it calculates MAC as a function C of key and message given as: **MAC=Ck(M)** The message and the MAC are transmitted to the intended recipient, who upon receiving 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 and only if they match, then:

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

2. The receiver is assured that the message is from the alleged sender: No one except the sender has the secret key and could prepare a message with a proper MAC.

3. If the message includes a sequence number, then receiver is assured of proper sequence as an attacker cannot successfully alter the sequence number.

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



(a) Message authentication

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

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

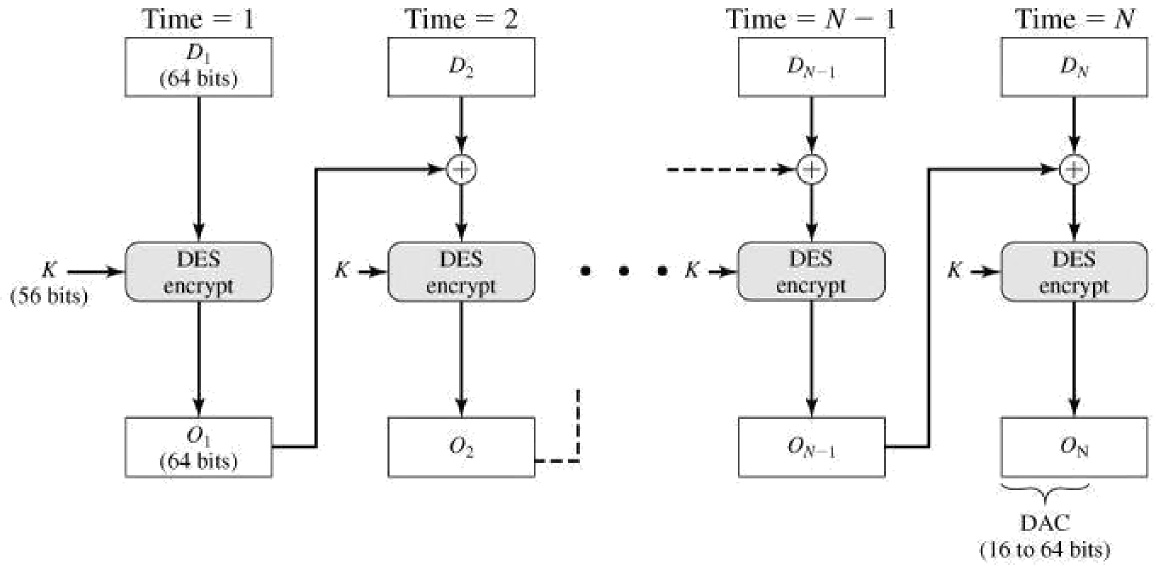
There are three different situations where use of a MAC is desirable:

- If a message is broadcast to several destinations in a network (such as a military control center), then it is cheaper and more reliable to have just one node responsible to evaluate the authenticity –message will be sent in plain with an attached authenticator.
- If one side has a heavy load, it cannot afford to decrypt all messages –it will just check the authenticity of some randomly selected messages.

- Authentication of computer programs in plaintext is very attractive service as they need not be decrypted every time wasting of processor resources. Integrity of the program can always be checked by MAC.

MESSAGE AUTHENTICATION CODE BASED ON DES

The Data Authentication Algorithm, based on DES, has been one of the most widely used MACs for a number of years. The algorithm is both a FIPS publication (FIPS PUB 113) and an ANSI standard (X9.17). But, security weaknesses in this algorithm have been discovered and it is being replaced by newer and stronger algorithms. The algorithm can be defined as using the cipher block chaining (CBC) mode of operation of DES shown below with an initialization vector of zero.

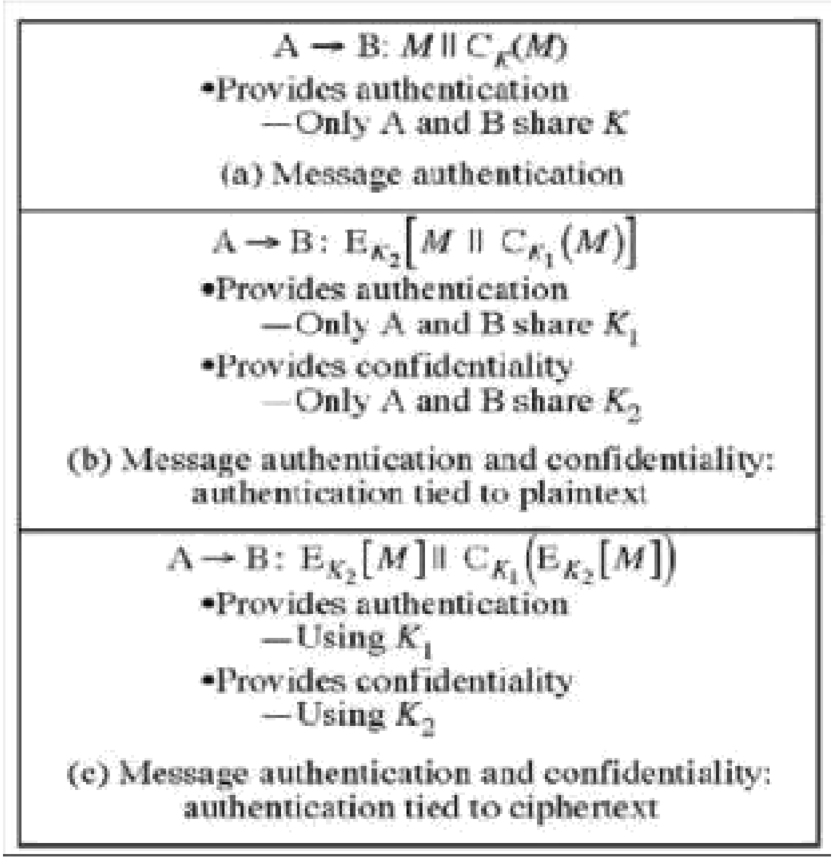


The data (e.g., message, record, file, or program) to be authenticated are grouped into contiguous 64-bit blocks: D1, D2,..., DN. If necessary, the final block is padded on the right with zeroes to form a full 64-bit block. Using the DES encryption algorithm, E, and a secret key, K, a data authentication code (DAC) is calculated as follows:

$$O_1 = E(K, D_1)$$
$$O_2 = E(K, [D_2 \oplus O_1])$$
$$O_3 = (K, [D_3 \oplus O_2])$$
$$\bullet$$
$$\bullet$$
$$\bullet$$
$$O_N = E(K, [D_N \oplus O_{N1}])$$

The DAC consists of either the entire block ON or the leftmost M bits of the block, with 16 ≤ M ≤ 64

Use of MAC needs a shared secret key between the communicating parties and also MAC does not provide digital signature. The following table summarizes the confidentiality and authentication implications of the approaches shown above.



$A \rightarrow B: M \| C_K(M)$

- Provides authentication
  - Only A and B share K

(a) Message authentication

$A \rightarrow B: E_{K_2}[M \| C_{K_1}(M)]$

- Provides authentication
  - Only A and B share K_1
- Provides confidentiality
  - Only A and B share K_2

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

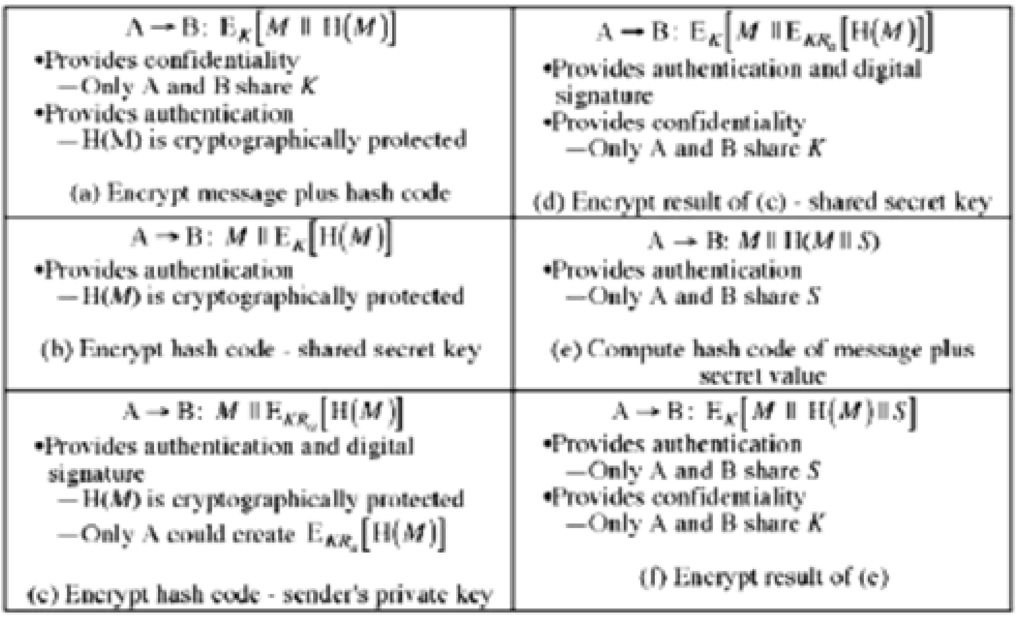
$A \rightarrow B: E_{K_2}[M] \| C_{K_1}(E_{K_2}[M])$

- Provides authentication
  - Using K_1
- Provides confidentiality
  - Using K_2

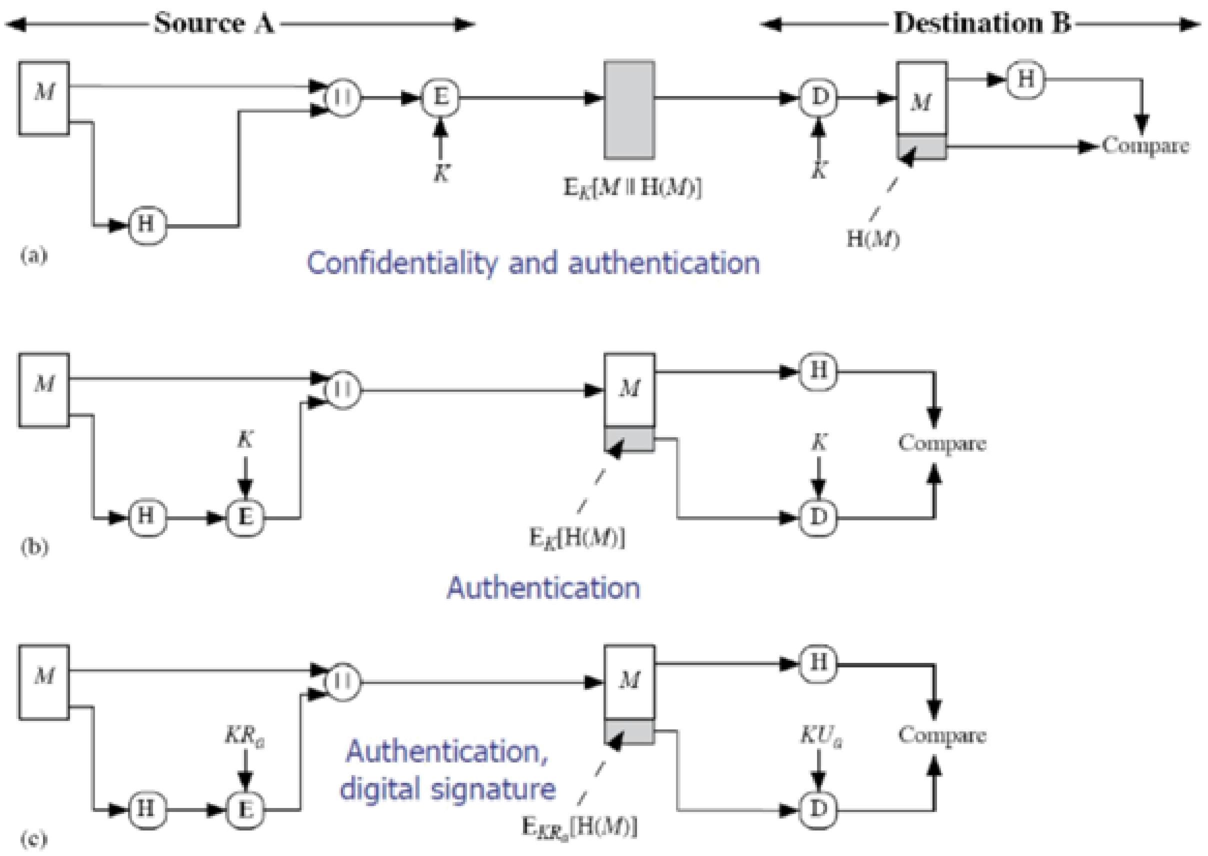
(c) Message authentication and confidentiality: authentication tied to ciphertext

HASH FUNCTION

A variation on the message authentication code is the one-way hash function. As with the message authentication code, the hash function accepts a variable-size message M as input and produces a fixed-size hash code H(M), sometimes called a message digest, as output. The hash code is a function of all bits of the message and provides an error-detection capability: A change to any bit or bits in the message results in a change to the hash code. A variety of ways in which a hash code can be used to provide message authentication is shown below and explained stepwise in the table.



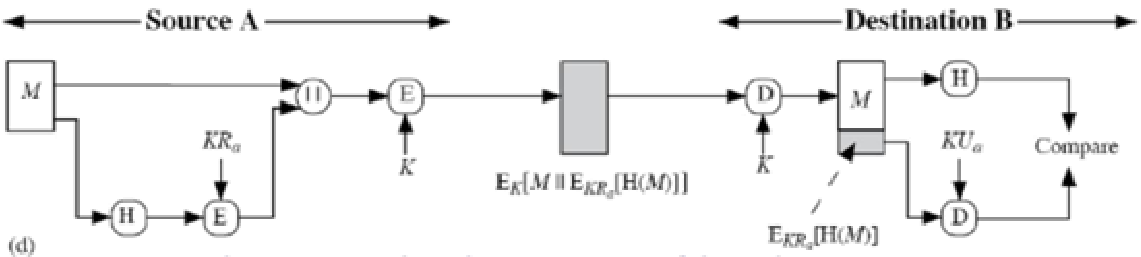
$A \rightarrow B: E_K[M \| H(M)]$ •Provides confidentiality —Only A and B share K •Provides authentication —H(M) is cryptographically protected (a) Encrypt message plus hash code	$A \rightarrow B: E_K[M \| E_{KR_a}[H(M)]]$ •Provides authentication and digital signature •Provides confidentiality —Only A and B share K (d) Encrypt result of (c) - shared secret key
$A \rightarrow B: M \| E_K[H(M)]$ •Provides authentication —$H(M)$ is cryptographically protected (b) Encrypt hash code - shared secret key	$A \rightarrow B: M \| H(M \| S)$ •Provides authentication —Only A and B share S (e) Compute hash code of message plus secret value
$A \rightarrow B: M \| E_{KR_a}[H(M)]$ •Provides authentication and digital signature —$H(M)$ is cryptographically protected —Only A could create $E_{KR_a}[H(M)]$ (c) Encrypt hash code - sender's private key	$A \rightarrow B: E_K[M \| H(M) \| S]$ •Provides authentication —Only A and B share S •Provides confidentiality —Only A and B share K (f) Encrypt result of (e)



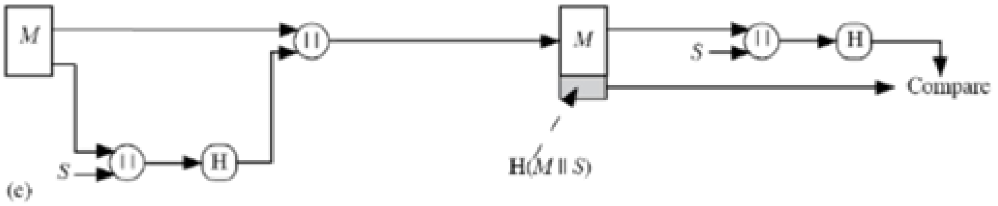
Confidentiality and authentication

Authentication

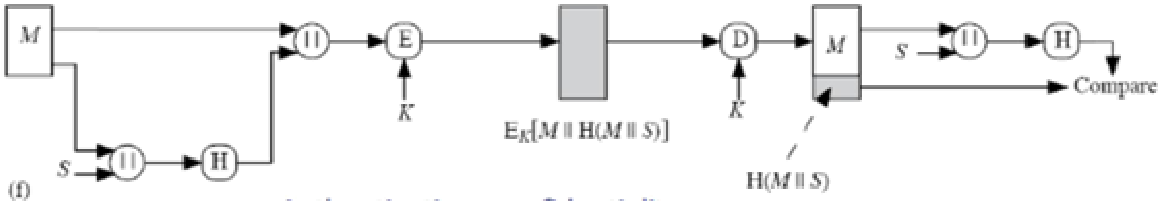
Authentication, digital signature



Authentication, digital signature, confidentiality



Authentication (no encryption needed!)



Authentication, confidentiality

In cases where confidentiality is not required, methods b and c have an advantage over those that encrypt the entire message in that less computation is required. Growing interest for techniques that avoid encryption is due to reasons like, Encryption software is quite slow and may be covered by patents. Also encryption hardware costs are not negligible and the algorithms are subject to U.S export control. A fixed-length hash value h is generated by a function H that takes as input a message of arbitrary length**: h=H(M).**

- A sends M and H(M)
- B authenticates the message by computing H(M) and checking the match

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

- H can be applied to a message of any size
- H produces fixed-length output
- Computationally easy to compute H(M) for any given M

- Computationally infeasible to find M such that H(M)=h, for a given h, referred to as the *one-way property*
- Computationally infeasible to find M’ such that H(M’)=H(M), for a given M, referred to as *weak collision resistance*.
- Computationally infeasible to find M,M’ with H(M)=H(M’) (to resist to birthday attacks), referred to as *strong collision resistance*.

Examples of simple hash functions are:

- Bit-by-bit XOR of plaintext blocks: h= D1⊕D2⊕…⊕DN
- Rotated XOR –before each addition the hash value is rotated to the left with 1 bit
- Cipher block chaining technique without a secret key.

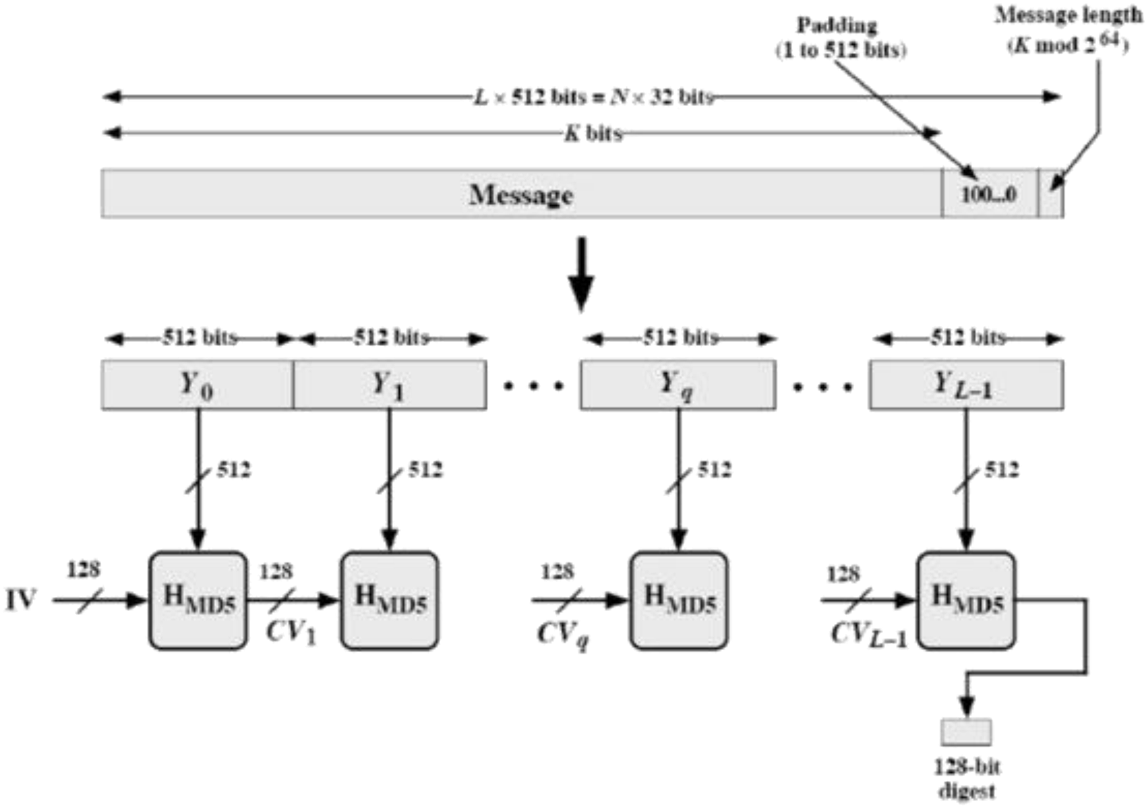
MD5 Message Digest Algorithm

The MD5 message-digest algorithm was developed by Ron Rivest at MIT and it remained as the most popular hash algorithm until recently. The algorithm takes as input, a message of arbitrary length and produces as output, 128-bit message digest. The input is processed in 512-bit blocks. The processing consists of the following steps:

1.) *Append Padding bits*: The message is padded so that its length in bits is congruent to 448 modulo 512 i.e. the length of the padded message is 64 bits less than an integer multiple of 512 bits. Padding is always added, even if the message is already of the desired length. Padding consists of a single 1-bit followed by the necessary number of 0-bits.

2.) *Append length*: A 64-bit representation of the length in bits of the original message (before the padding) is appended to the result of step-1. If the length is larger than 264, the 64 least representative bits are taken.

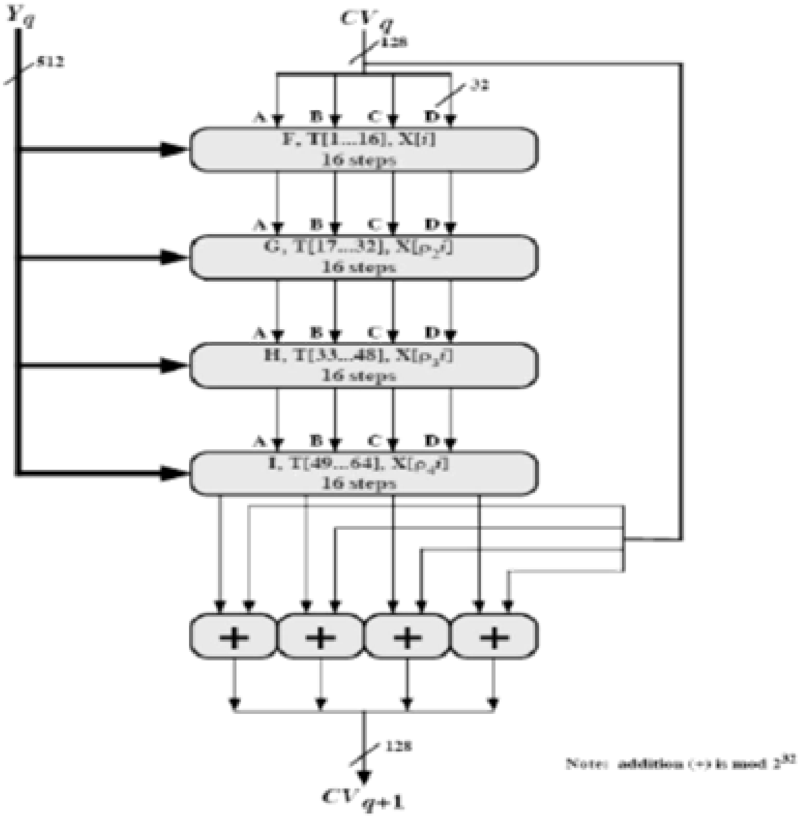
3.) *Initialize MD buffer*: A 128-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as four 32-bit registers (A, B, C, D) and are initialized with A=0x01234567, B=0x89ABCDEF, C=0xFEDCBA98, D=0x76543210 i.e. 32-bit integers (hexadecimal values).



Message Digest Generation Using MD5

4.) *Process Message in 512-bit (16-word) blocks* : The h art of algorithm is the compression function that consists of four rounds of processing and this module is labeled HMD5 in the above figure and logic is illustrated in the following figure. The four rounds have a similar structure, but each uses a different primitive logical function, referred to as F, G, H and I in the specification. Each block takes as input the current 512-bit block being processed Yq and the 128-bit buffer value ABCD and updates the contents of the buffer. Each round also makes use of one-fourth of a 64- element table T*1….64+, constructed

from the sine function. The ith element of T, denoted T[i], has the value equal to the integer part of 232 * abs(sin(i)), where i is in radians. As the value of abs(sin(i)) is a value between 0 and 1, each element of T is an integer that can be represented in 32-bits and would eliminate any regularities in the input data. The output of fourth round is added to the input to the first round (CVq) to produce CVq+1. The addition is done independently for each of the four words in the buffer with each of the corresponding words in CVq, using addition modulo 232. This operation is shown in the figure below:



5.) *Output*: After all L 512-bit blocks have been proc ssed, the output from the Lth stage is the 128- bit message digest. MD5 can be summarized as follows:

CV0 = IV CV_{q+1} = SUM_{32}(CV_q,$RF_I Y_q RF_H$[Y_q,RF $_G$[Y_q,RF_F[Y_q,CV_q]]]]) MD = CV_L Where,

IV = initial value of ABCD buffer, defined in step 3.

Y_q = the q_{th} 512-bit block of the message

L = the number of blocks in the message

CV_q = chaining variable processed with the q_{th} block of the message.

RFx = round function using primitive logical function x.

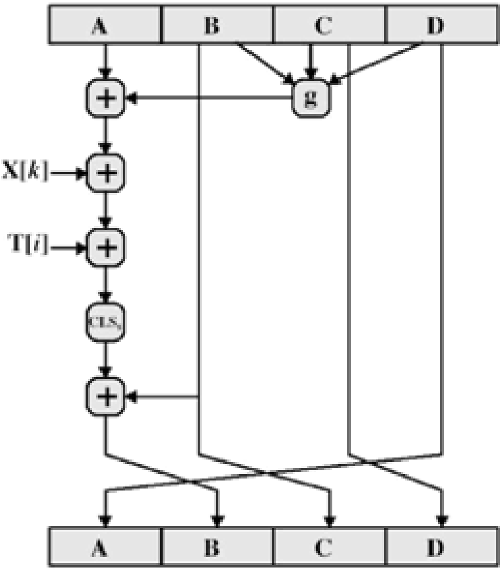
MD = final message digest value

SUM_{32} = Addition modulo 2_{32} performed separately.

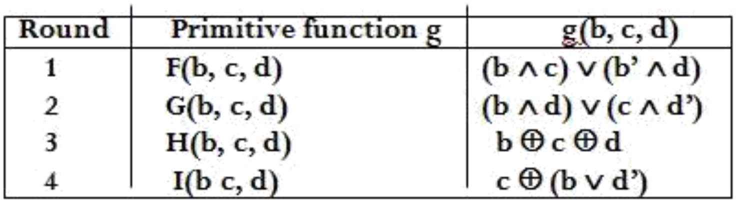
MD5 Compression Function:

Each round consists of a sequence of 16 steps operating on the buffer ABCD. Each step is of the form, **a = b+((a+g(b,c,d)+X[k]+T[i])<<<s)**

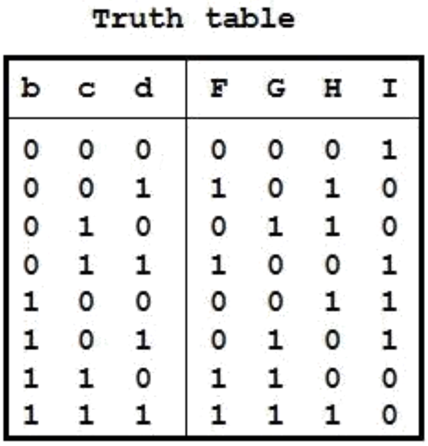
where a, b, c, d refer to the four words of the buffer but used in varying permutations. After 16 steps, each word is updated 4 times. g(b,c,d) is a different nonlinear function in each round (F,G,H,I). Elementary MD5 operation of a single step is shown below.



The primitive function g of the F,G,H,I is given as:



Round	Primitive function g	g(b, c, d)
1	F(b, c, d)	(b ∧ c) ∨ (b' ∧ d)
2	G(b, c, d)	(b ∧ d) ∨ (c ∧ d')
3	H(b, c, d)	b ⊕ c ⊕ d
4	I(b c, d)	c ⊕ (b ∨ d')



Truth table

b	c	d	F	G	H	I
0	0	0	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	0	1
1	1	0	1	1	0	0
1	1	1	1	1	1	0

Where the logical operators (AND, OR, NOT, XOR) are represented by the symbols (∧,v**, ~,(+)).**

Each round mixes the buffer input with the next "word" of the message in a complex, non-linear manner. A different non-linear function is used in each of the 4 rounds (but the same function for all 16 steps in a round). The 4 buffer words (a,b,c,d) are rotated from step to step so all are used and updated. g is one of the primitive functions F,G,H,I for the 4 rounds respectively. X[k] is the kth 32-bit word in the current message block. T[i] is the ith entry in the matrix of constants T. The addition of varying constants T and the use of different shifts helps ensure it is extremely difficult to compute collisions. The array of 32-bit words X[0..15] holds the value of current 512-bit input block being processed. Within a round, each of the 16 words of X[i] is used exactly once, during one step. The order in which these words is used varies from round to round. In the first round, the words are used in their original order. For rounds 2 through 4, the following permutations are used

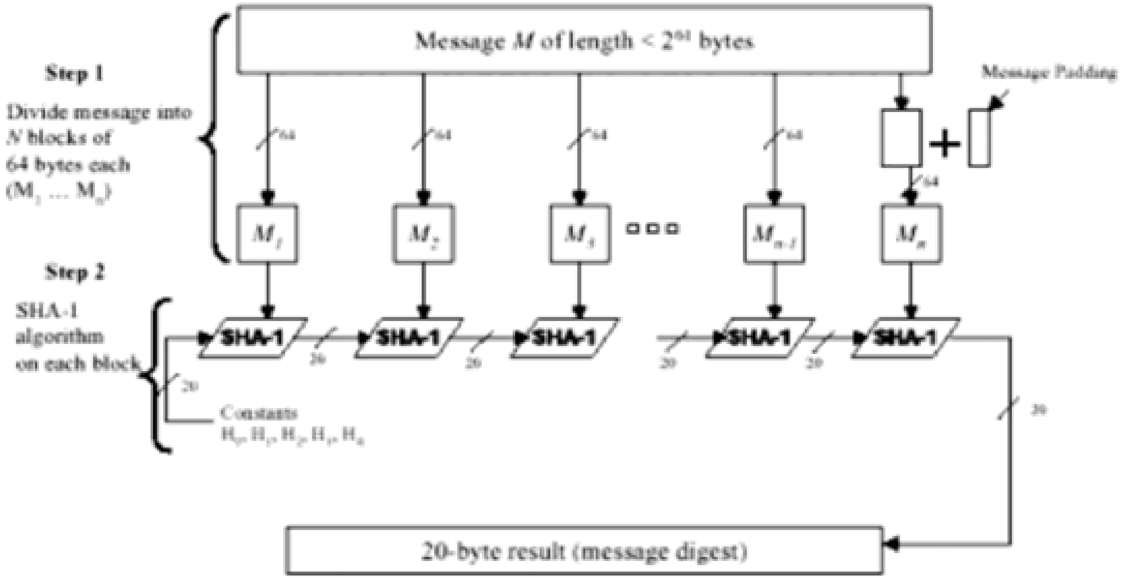
- p2(i) = (1 + 5i) mod 16
- p3(i) = (5 + 3i) mod 16
- p4(I) = 7i mod 16

MD4

- Precursor to MD5
- Design goals of MD4 (which are carried over to MD5)
- Security
- Speed
- Simplicity and compactness
- Favor little-endian architecture
- Main differences between MD5 and MD4
- A fourth round has been added.
- Each step now has a unique additive constant.
- The function g in round 2 was changed from (bc v bd v cd) to (bd v cd') to make g less symmetric.
- Each step now adds in the result of the previous step. This promotes a faster "avalanche effect".
- The order in which input words are accessed in rounds 2 and 3 is changed, to make these patterns less like each other.
- The shift amounts in each round have been approximately optimized, to yield a faster "avalanche effect." The shifts in different rounds are distinct.

Secure Hash Algorithm

The secure hash algorithm (SHA) was developed by the National Institute of Standards and Technology (NIST). SHA-1 is the best established of the existing SHA hash functions, and is employed in several widely used security applications and protocols. The algorithm takes as input a message with a maximum length of less than 264 bits and produces as output a 160-bit message digest.


Message M of length < 2^64 bytes
Message Padding
Step 1
Divide message into N blocks of 64 bytes each (M1 ... Mn)
64
64
64
64
+
64
M1
M2
M3
Mn-1
Mn
Step 2
SHA-1 algorithm on each block
SHA-1
SHA-1
SHA-1
SHA-1
SHA-1
20
20
20
20
20
20
Constants H0, H1, H2, H3, H4
20-byte result (message digest)

The input is processed in 512-bit blocks. The overall processing of a message follows the structure of MD5 with block length of 512 bits and hash length and chaining variable length of 160 bits. The processing consists of following steps:

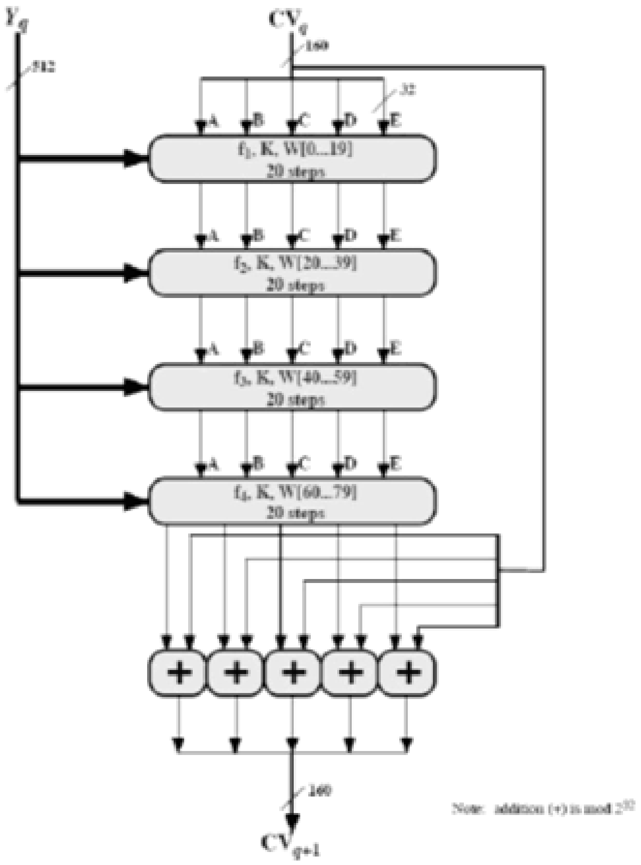
1.) ***Append Padding Bits:*** The message is padded so that length is congruent to 448 modulo 512; padding always added –one bit 1 followed by the necessary number of 0 bits.

2.) ***Append Length:*** a block of 64 bits containing the length of the original message is added.

3.) ***Initialize MD buffer*:**A160-bitbufferisued to hold intermediate and final results on the hash function. This is formed by 32-bit registers A,B,C,D,E. Initial values: A=0x67452301, B=0xEFCDAB89, C=0x98BADCFE, D=0x10325476, E=C3D2E1F0. Stores in big-endian format i.e. the most significant bit in low address.

4.) ***Process message in bloc 512-bit (16-word) blocks*:** The processing of a single 512-bit block is shown above. It consists of four rounds of processing of 20 steps each. These four rounds have similar structure, but uses a different primitive logical function, which we refer to as f1, f2, f3 and f4. Each round takes as input the current 512-bit block being processed and the 160-bit buffer value ABCDE and updates the contents of the buffer. Each round also makes use of four distinct additive constants Kt. The output of the fourth round i.e. eightieth step is added to the input to the first round to produce CVq+1.

5.) ***Output:*** After all L 512-bit blocks have been processed, the output from the Lth stage is the 160-bit message digest.

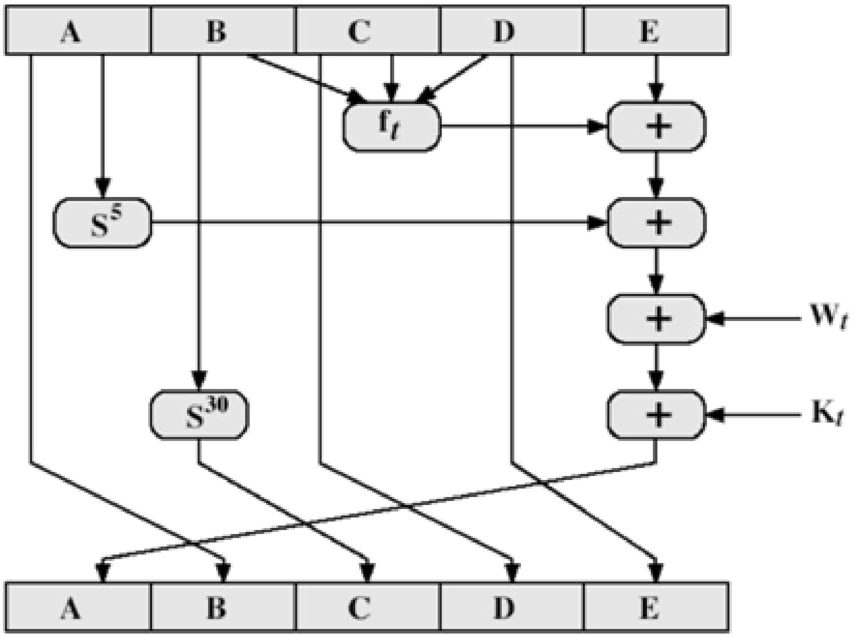


SHA-1 Processing of a Single 512-bit Block
(SHA-1 Compression Function)

The behavior of SHA-1 is as follows: **CV0 = IV CVq+1 = SUM32(CVq, ABCDEq) MD = CVL** Where, IV = initial value of ABCDE buffer ABCDEq = output of last round of processing of qth message block L = number of blocks in the message SUM32 = Addition modulo 232 MD = final message digest value.

SHA-1 Compression Function:

Each round has 20 steps which replaces the 5 buffer words. The logic present in each one of the 80 rounds present is given as **(A,B,C,D,E) <- (E + f(t,B,C,D) + S_5(A)+ Wt+ Kt),A,S_{30}(B),C,D** Where, A, B, C, D, E = the five words of the buffer t = step number; 0< t < 79 f(t,B,C,D) = primitive logical function for step t S_k = circular left shift of the 32-bit argument by k bits Wt **=** a 32-bit word derived from current 512-bit input block. Kt **=** an additive constant; four distinct values are used + = modulo addition.



SHA shares much in common with MD4/5, but with 20 instead of 16 steps in each of the 4 rounds. Note the 4 constants are based on sqrt(2,3,5,10).Note also that instead of just splitting the input block into 32-bit words and using them d recently, SHA-1 shuffles and

mixes them using rotates & XOR's to form more complex input, and greatly increases the difficulty of finding collisions. A sequence of logical functions f_0, f_1,..., f_{79} is used in the SHA-1. Each f_t, 0<=t<=79, operates on three 32-bit words B, C, D and produces a 32-bit word as output. f_t(B,C,D) is defined as follows: for words B, C, D, **f_t(B,C,D) = (B AND C) OR ((NOT B) AND D) (0 <= t <= 19) f_t(B,C,D) = B XOR C XOR D (20 <= t <= 39) f_t(B,C,D) = (B AND C) OR (B AND D) OR (C AND D) (40 <= t <= 59) f_t(B,C,D) = B XOR C XOR D (60 <= t <= 79).**

DIGITAL SIGNATURE

The most important development from the work on public-key cryptography is the digital signature. Message authentication protects two parties who exchange messages from any third party. However, it does not protect the two parties against each other. A digital signature is analogous to the handwritten signature, and provides a set of security capabilities that would be difficult to implement in any other way. It must have the following properties:

- 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 Thus, the digital signature function includes the authentication function. A variety of approaches has been proposed for the digital signature function. These approaches fall into two categories: direct and arbitrated.

Direct Digital Signature

Direct Digital Signatures involve the direct application of public-key algorithms involving only the communicating parties. A digital signature may be formed by encrypting the entire message with the sender’s private key, or by encrypting a hash code of the message with the sender’s private key. Confidentiality can be provided by further encrypting the entire message plus signature using either public or private key schemes. It is important to perform the signature function first and then an outer confidentiality function, since in case of dispute, some third party must view the message nd its signature. But these approaches are dependent on the security of the sender’s private-key. Will have problems if it is lost/stolen and signatures forged. Need time-stamps and timely key revocation.

Arbitrated Digital Signature

The problems associated with direct digital signatures can be addressed by using an arbiter, in a variety of possible arrangements. The arbiter plays a sensitive and crucial role in this sort of scheme, and all parties must have a great deal of trust that the arbitration mechanism is working properly. These schemes can be implemented with either private or public- ey algorithms, and the arbiter may or may not see the actual message contents. **Using Conventional encryption**

$$X \rightarrow A : M \| E(Kxa, [IDx \| H(M)])$$

$$A \rightarrow Y : E(Kay, [IDx \| M \| E(Kxa, [IDx \| H(M)]) \| T])$$

- It is assumed that the sender X and the arbiter A share a secret key Kxa and that A and Y share secret key Kay. X constructs a message M and computes its hash value H(m) . Then X transmits the message plus a signature to A. the signature consists of an identifier IDx of X plus the hash value, all encrypted using Kxa.
- A decrypts the signature and checks the hash value to validate the message. Then A transmits a message to Y, encrypted with Kay. The message includes IDx, the original message from X, the signature, and a timestamp.

- Arbiter sees message
- Problem : the arbiter could form an alliance with sender to deny a signed message, or with the receiver to forge the sender’s signature.

Using Public Key Encryption

X ⍰A : IDx ||E(PRx,[IDx|| E (PUy, E(PRx, M))])

A ⍰Y : E(PRa, [IDx ||E (PUy, E (PRx, M))|| T])

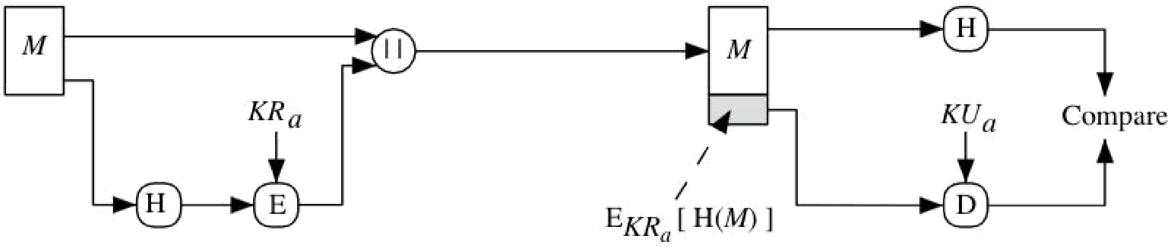
⍰⍰X double encrypts a message M first with X’s private key, PRx, and then with Y’s public key, PUy. This is a signed, secret version of the message. This signed message, together with X’s identifier , is encrypted again with PRx and, together with IDx, is sent to A. The inner, double encrypted message is secure from the arbiter (and everyone else except Y)

- A can decrypt the outer encryption to assure that the message must have come from X (because only X has PRx). Then A transmits a message to Y, encrypted with PRa. The message includes IDx, the double encrypted message, and timestamp.
- Arbiter does not see message

Digital Signature Standard (DSS)

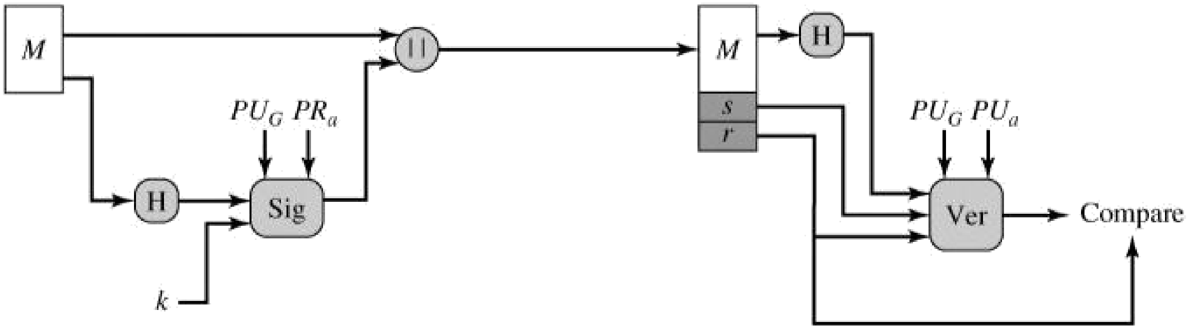
The National Institute of Standards and Technology (NIST) has published Federal Information Processing Standard FIPS 186, known as the Digital Signature Standard (DSS). The DSS makes use of the Secure Ha h Algorithm (SHA) and presents a new digital signature technique, the Digital Signature Algorithm (DSA). The DSS uses an algorithm that is designed to provide only the digital signature function and cannot be used for encryption or key exchange, unlike RSA.

The RSA approach is shown below. 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 the sender's public key. If the calculated hash code matches the decrypted signature, the signature is accepted as valid. Because only the sender knows the private key, only the sender could have produced a valid signature.

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 k 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 (PUG).The result is a signature consisting of two components, labeled s and r.



(b) DSS approach

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 (PUa), which is paired with the sender's private key. The outp t of the verification function is a value that is equal to the signature component r 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.

AUTHENTICATION APPLICATIONS

KERBEROS

Kerberos is an authentication service developed as part of Project Athena at MIT. It addresses the threats posed in an o en distributed environment in which users at workstations wish to access services on servers distributed throughout the network. Some of these threats are:

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

Two versions of Kerberos are in current use: Version-4 and Version-5. The first published report on Kerberos listed the following requirements:

Secure: A network eavesdropper should not be able to obtain the necessary information to impersonate a user. More generally, Kerberos should be strong enough that a potential opponent does not find it to be the weak link.

Reliable: For all services that rely on Kerberos for access control, lack of availability of the Kerberos service means lack of availability of the supported services. Hence, Kerberos should be highly reliable and should employ a distributed server architecture, with one system able to back up another.

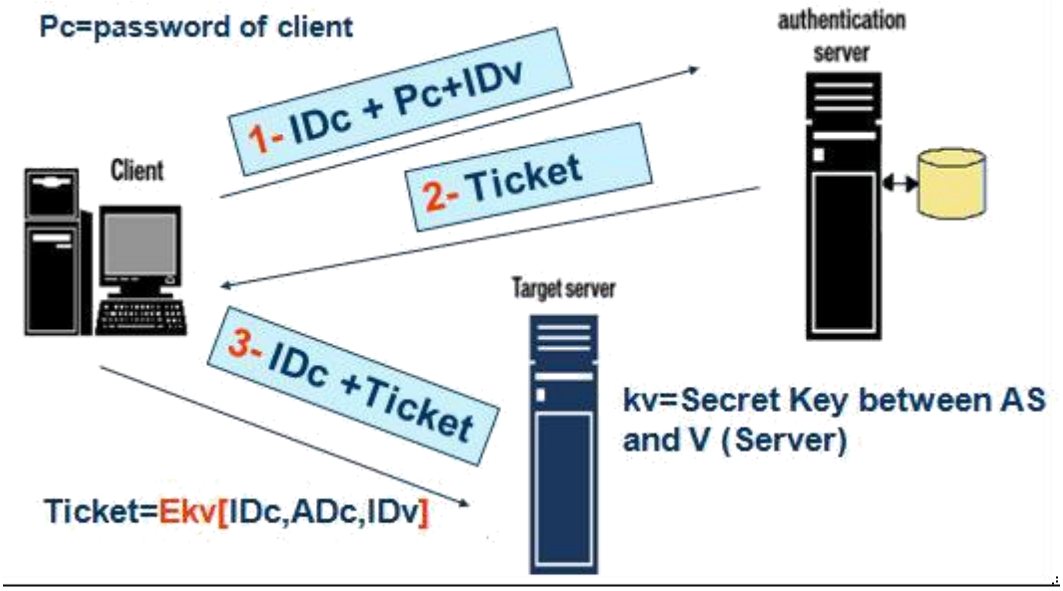
Transparent: Ideally, the user should not be aware that authentication is taking place, beyond the requirement to enter a password.

Scalable: The system should be capable of supporting large numbers of clients and servers. This suggests a modular, distributed architecture

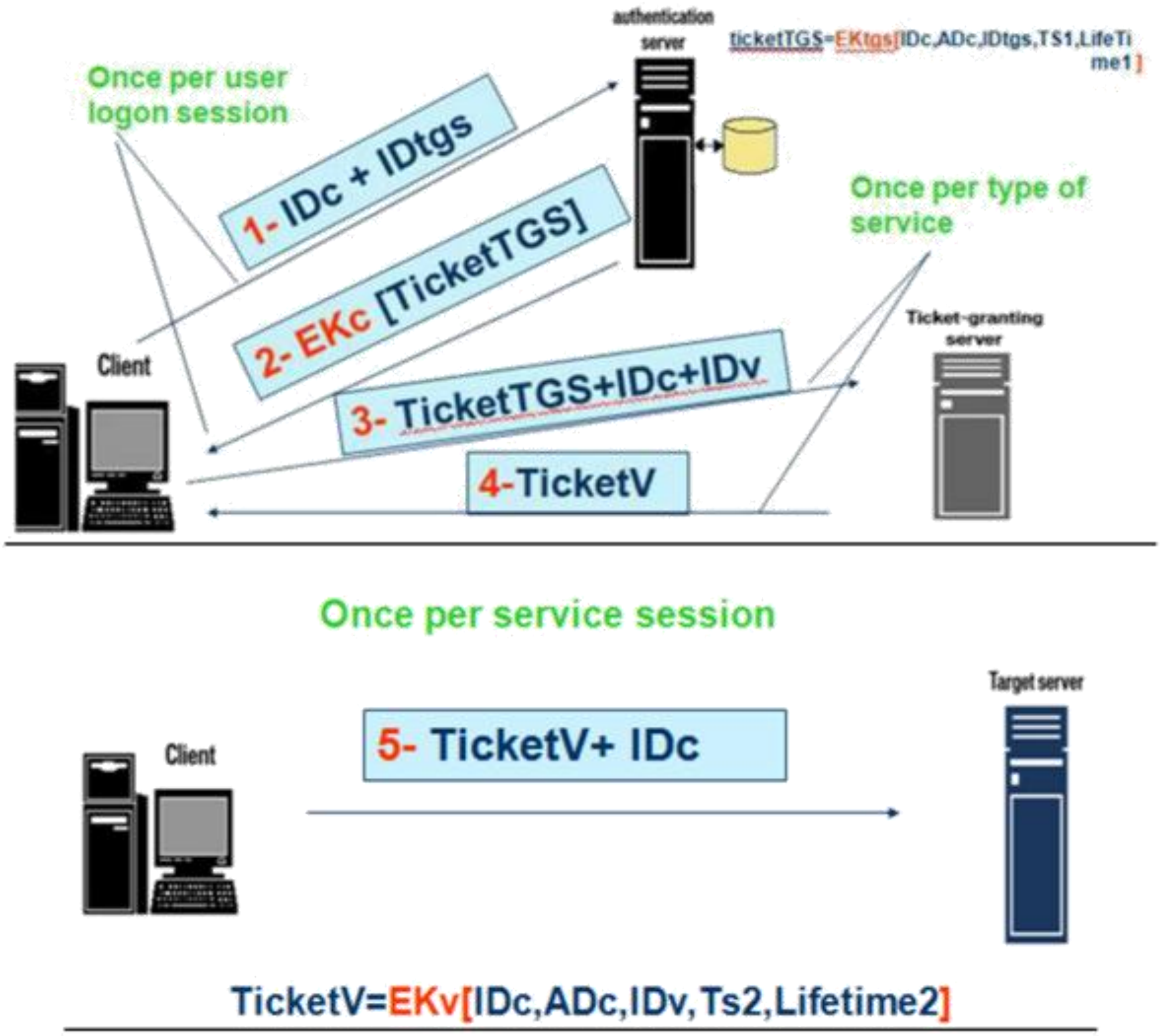
Two versions of Kerberos are in common use: Version 4 is most widely used version. Version 5 corrects some of the security deficiencies of Vers on 4. Version 5 has been issued as a draft Internet Standard (RFC 1510)

KERBEROS VERSION 4

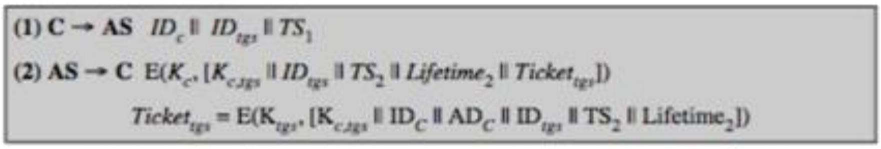
1.) SIMPLE DIALOGUE:



MORE SECURE DIALOGUE



The Version 4 Authentication Dialogue The full Kerberos v4 authentication dialogue is shown here divided into 3 phases.

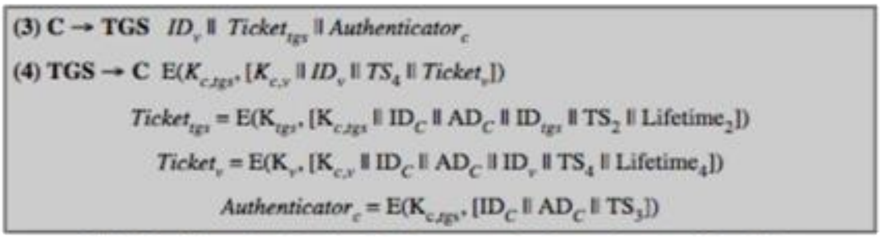


(1) C → AS $ID_c \| ID_{tgs} \| TS_1$

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

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

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



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

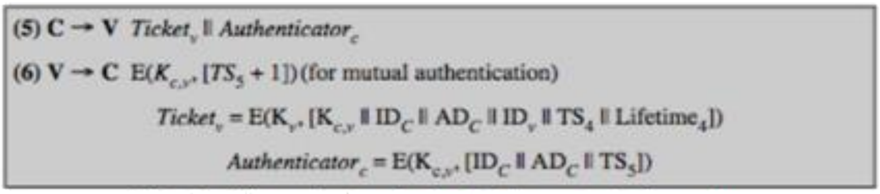
(4) TGS → C $E(K_{c,tgs}, [K_{c,v} \| ID_v \| TS_4 \| Ticket_v])$

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

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

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

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



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

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

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

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

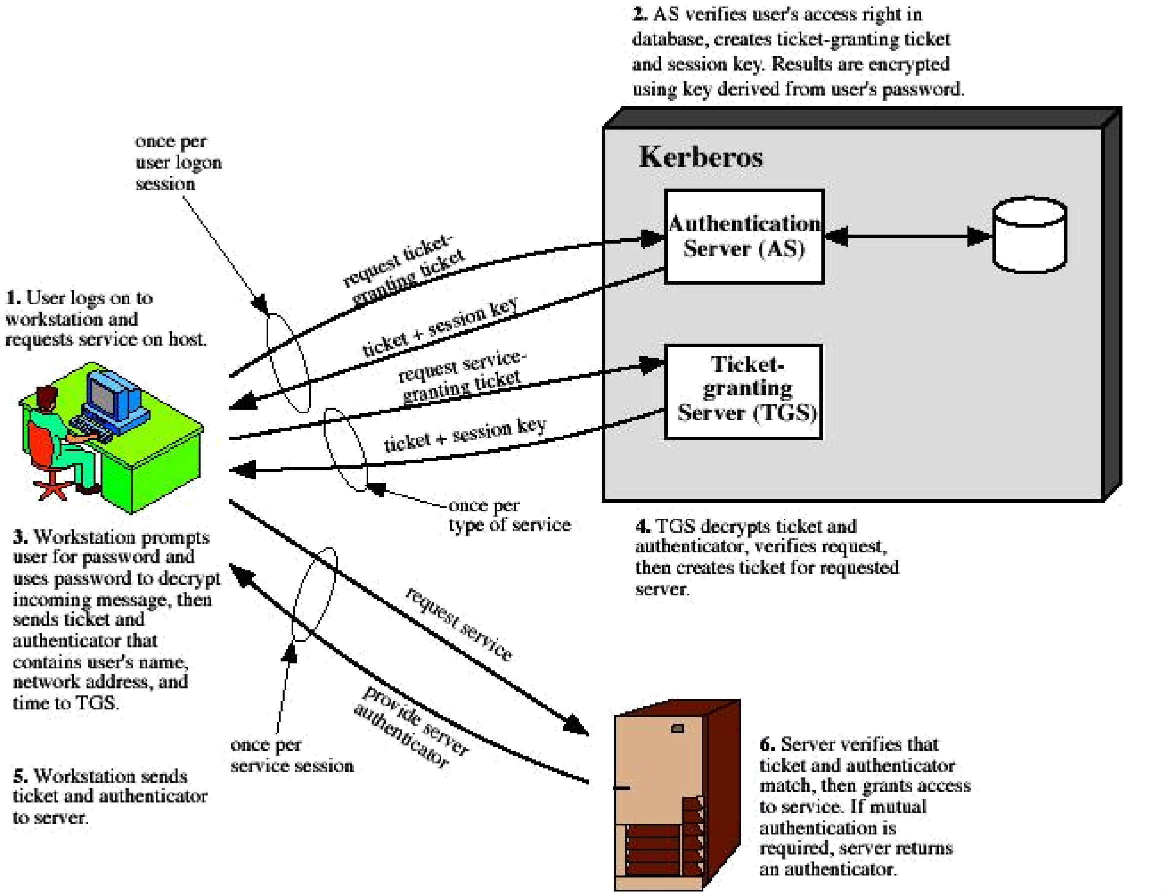
(c) Client/Server Authentication Exchange to obtain service

There is a problem of captured ticket-granting tickets nd the need to determine that the ticket presenter is the same as the client for whom the ticket was issued. An efficient way of doing this is to use a session encryption key to secure information.

Message (1) includes a time stamp, so that the AS knows that the message is timely. Message (2) includes several elements of the ticket in a form accessible to C. This enables C to confirm that this ticket is for the TGS and to learn its expiration time. Note that the ticket does not prove anyone's identity but is a way to distribute keys securely. It is the authenticator that proves the client's identity. Because the authenticator can be used only once and has a short lifetime, the threat of an opponent stealing both the ticket and the authenticator for presentation later is countered. C then sends the TGS a message that includes the ticket plus the ID of the requested service (message 3). The reply from the TGS, in message (4), follows the form of message (2). C now has a reusable service-granting ticket for V. When C presents this ticket, as shown in message (5), it also sends an authenticator.

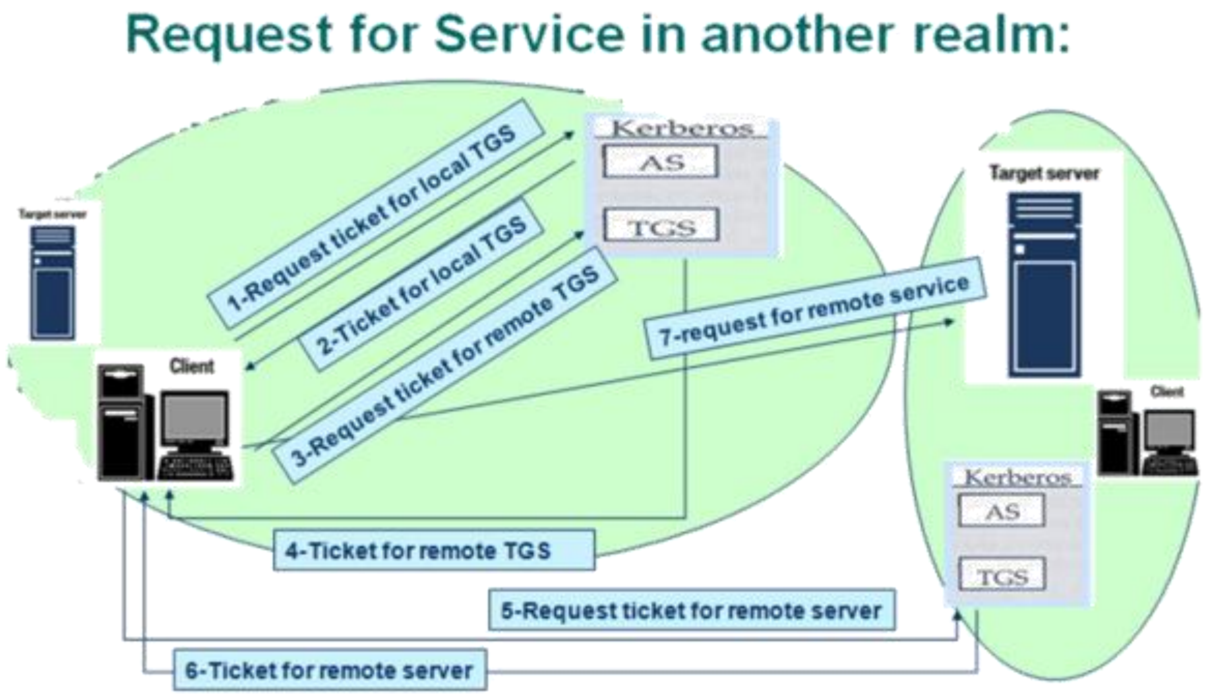
The server can decrypt the ticket, recover the session key, and decrypt the authenticator.If mutual authentication is required, the server can reply as shown in message (6).

Overview of Kerberos


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

Kerberos Realms A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers is referred to as a Kerberos realm. A Kerberos realm is a set of managed nodes that share the same Kerberos database, and are part of the same administrative domain. If have multiple realms, their Kerberos servers must share key and trust each other.

The following figure shows the authentication messages where service is being requested from another domain. The ticket presented to the remote server indicates the realm in which the user was originally authenticated. The server chooses whether to honor the remote request. One problem presented by the foregoing approach is that it does not scale well to many realms, as each pair of realms need to share a key.



Request for Service in another realm:

Kerberos
AS
TGS
Target server
Client
1-Request ticket for local TGS
2-Ticket for local TGS
3-Request ticket for remote TGS
4-Ticket for remote TGS
5-Request ticket for remote server
6-Ticket for remote server
7-request for remote service
Target server
Client
Kerberos
AS
TGS

The limitations of Kerberos version-4 are categorised into two types:

- Environmental shortcomings of Version 4:

– Encryption system dependence: DES

– Internet protocol dependence

– Ticket lifetime

– Authentication forwarding

➢ Inter-realm authentication Technical

- deficiencies of Version 4:

– Double encryption

– Session Keys

– Password attack

KERBEROS VERSION 5

Kerberos Version 5 is specified in RFC 1510 and provides a number of improvements over version 4 in the areas of environmental shortcomings and technical deficiencies. It includes some new elements such as:

➢ Realm: Indicates realm of the user

➢ Options

➢ Times

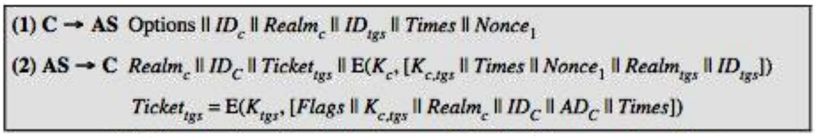
– From: the desired start time for the ticket

– Till: the requested expiration time

– Rtime: requested renew-till time

- Nonce: A random value to assure the response is fresh

The basic Kerberos version 5 authentication dialogue is shown here First, consider the **authentication service exchange.**

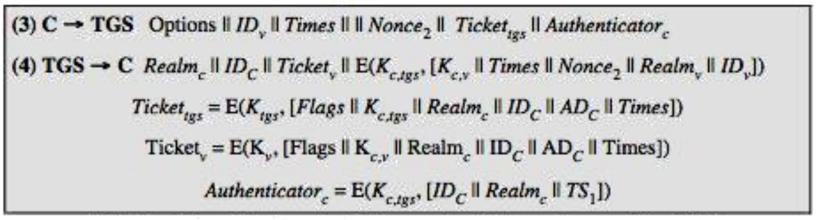


(1) C → AS $\text{Options} \parallel ID_c \parallel Realm_c \parallel ID_{tgs} \parallel Times \parallel Nonce_1$

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

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

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



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

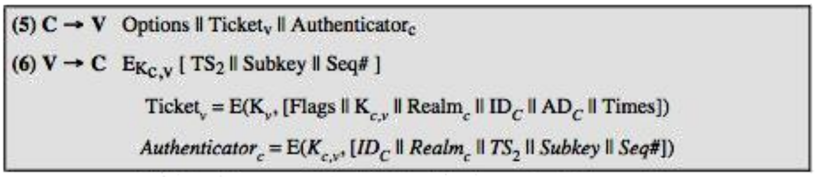
(4) TGS → C $Realm_c \parallel ID_C \parallel Ticket_v \parallel \text{E}(K_{c,tgs}, [K_{c,v} \parallel Times \parallel Nonce_2 \parallel Realm_v \parallel ID_v])$

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

$\text{Ticket}_v = \text{E}(\text{K}_v, [\text{Flags} \parallel \text{K}_{c,v} \parallel \text{Realm}_c \parallel \text{ID}_C \parallel \text{AD}_C \parallel \text{Times}])$

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

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



(5) C → V $\text{Options} \parallel \text{Ticket}_v \parallel \text{Authenticator}_c$

(6) V → C $\text{E}_{\text{K}_{c,v}} [\ \text{TS}_2 \parallel \text{Subkey} \parallel \text{Seq\#}\ ]$

$\text{Ticket}_v = \text{E}(\text{K}_v, [\text{Flags} \parallel \text{K}_{c,v} \parallel \text{Realm}_c \parallel \text{ID}_C \parallel \text{AD}_C \parallel \text{Times}])$

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

(c) Client/Server Authentication Exchange to obtain service

Message (1) is a client request for a ticket -granting ticket. 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. Now compare the **ticket-granting service** exchange for versions 4 and 5. See that message (3) for both versions includes an authenticator, a ticket, and the name of the requested service. In addition, version 5 includes requested times and options for the ticket and a nonce, all with functions similar to those of message (1). The authenticator itself is essentially the same as the one used in version 4. Message (4) has the same structure as message (2), 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, such as a request for mutual authentication. If required, the server responds with message (6) that includes the timestamp from the authenticator. The flags field included in tickets in version 5 supports expanded functionality compared to that available in version 4.

Advantages of Kerberos:

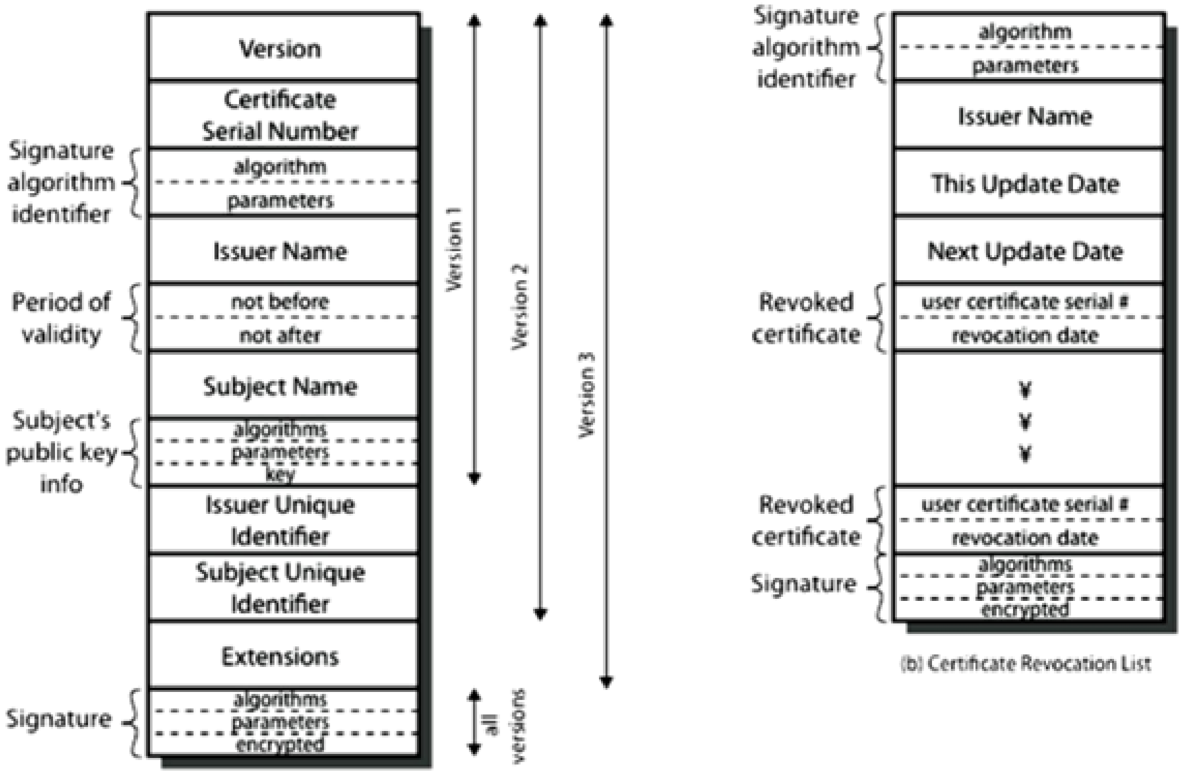
- User's passwords are never sent across the network, encrypted or in plain text
- Secret keys are *only* passed across the network in encrypted form
- Client and server systems mutually authenticate
- It limits the duration of their users' authentication.
- Authentications are **reusable** and **durable**
- Kerberos has been scrutinized by many of the top programmers, cryptologists and security experts in the industry

X.509 Authentication Service

ITU-T recommendation X.509 is partMediaoftheX.500series of recommendations that define a directory service. The directory is, in effect, server or distributed set of servers that maintains a database of information about users. The information includes a mapping from user name to network address, as w ll as other attributes and information about the users. X.509 is based on the use of public-key cryptography and digital signatures. The heart of the X.509 scheme is the public-key certificate associated with each user. These user certificates are a umed to be created by some trusted certification authority (CA) and placed in the directory by the CA or by the user. The directory server itself is not responsible for the creation of public keys or for the certification function; it merely provides an easily accessible location for users to obtain certificates.

The general format of a certificate is shown above, which includes the following elements:

- version 1, 2, or 3
- serial number (unique within CA) identifying certificate
- signature algorithm identifier issuer X.500 name (CA)
- period of validity (from - to dates)
- subject X.500 name (name of owner)
- subject public-key info (algorithm, parameters, key)
- issuer unique identifier (v2+)



(b) Certificate Revocation List

- subject unique identifier (v2+) Media
- extension fields (v3)
- signature (of hash of all fields in certificate)

The standard uses the following notation to define certificate:

CA<<A>> = CA {V, SN, AI, CA, TA, A, Ap}

Where Y<<X>>= the certificate of ser X issued by certification authority Y

Y {I} == the signing of I by Y. It consists of I with an encrypted hash code appended

User certificates generated by a CA have the following characteristics:

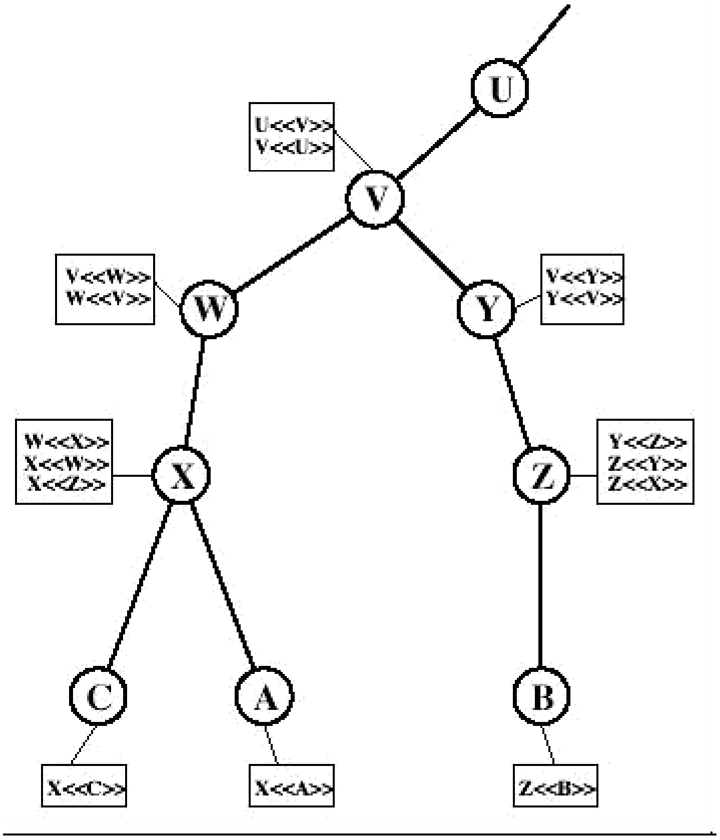
1. Any user with CA’s public key can verify the user public key that was certified
2. No party other than the CA can modify the certificate without being detected
3. because they cannot be forged, certificates can be placed in a public directory

Scenario: Obtaining a User Certificate If both users share a common CA then they are assumed to know its public key. Otherwise CA's must form a hierarchy and use certificates linking members of hierarchy to validate other CA's. Each CA has certificates for clients (forward) and parent (backward). Each client trusts parents certificates. It enables verification of any certificate from one CA by users of all other CAs in hierarchy. A has obtained a certificate from the CA X1. B has obtained a certificate from the CA X2. A can read the B’s certificate but cannot verify it. In order to solve the problem , the Solution: **X1<<X2> X2<<B>>.** A obtain the certificate of X2 signed by X1 from directory. obtain X2’s publi key. A goes back to directory and obtain the certificate of B signed by X2.

obtain B’s public key securely. 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

X.509 CA Hierarchy

A acquires B certificate using chain:
X<<W>>W<<V>>V<<Y>>Y<<Z>>
Z<<B>> B acquires A certificate using chain:
Z<<Y>>Y<<V>>V<<W>>W<<X>> X<<A>>



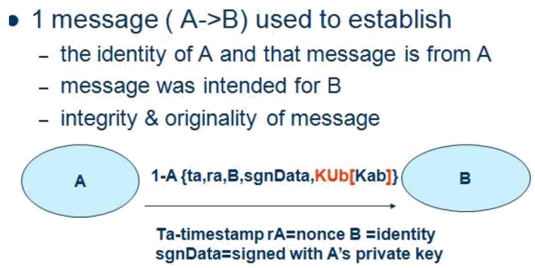
Revocation of Certificates Typically, a new certificate is issued just before the expiration of the old one. In addition, it may be desirable on occasion to revoke a certificate before it expires, for one of the following reasons:

- The user's private key is assumed to be compromised.
- The user is no longer certified by this CA.
- The CA's certificate is assumed to be compromised.

Each CA must maintain a list consisting of all revoked but not expired certificates issued by that CA, including both those issued to users and to other CAs. These lists should also be posted on the directory. Each **certificate revocation list (CRL) posted** to the directory is signed by the issuer and includes the issuer's name, the date the list was created, the date the next CRL is scheduled to be issued, and an entry for each revoked certificate. Each entry consists of the serial number of a certificate and revocation date for that certificate. Because serial numbers are unique within a CA, the serial number is sufficient to identify the certificate.

AUTHENTICATION PROCEDURES

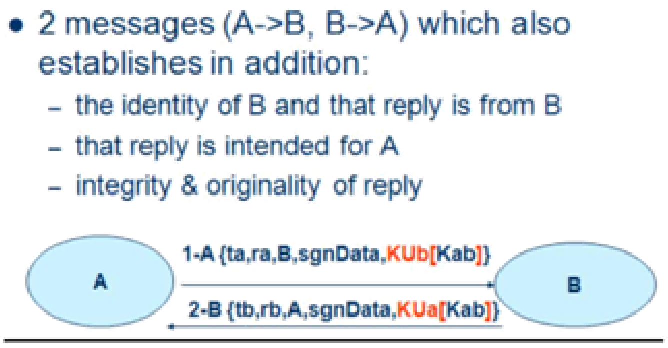
X.509 also includes three alternative authentication procedures that are intended for use across a variety of applications. All these procedures make use of public-key signatures. It is assumed that the two parties know each other's public key, there by obtaining each other's certificates from the directory or because the certificate is included in the initial message from each side. 1. One-Way Authentication: One way authentication involves a single transfer of information from one user (A) to another (B), and establishes the details shown above. Note that only the identity of the initiating entity is verified in this process, not that of the responding entity. At a minimum, the message includes a timestamp, a nonce, and the identity of B and is signed with A’s private key. The message may also include information to be conveyed, such as a session ey for B.



- 1 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

A
1-A {ta,ra,B,sgnData,KUb[Kab]}
B
Ta-timestamp rA=nonce B =identity
sgnData=signed with A’s private key

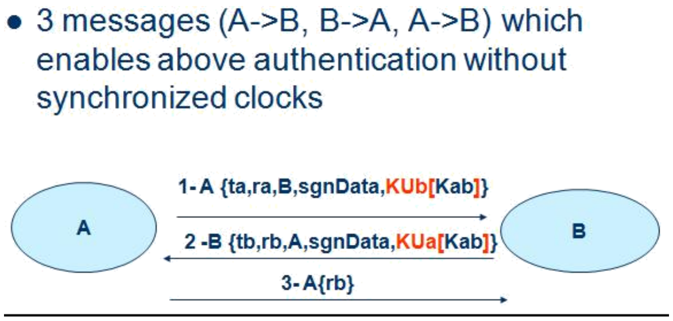
Two-Way Authentication: Two-way authentication thus permits both parties in a communication to verify the identity of the other, thus additionally establishing the above details. The reply message includes the nonce from A, to validate the reply. It also includes a timestamp and nonce generated by B, and possible additional information for A.



- 2 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

A
1-A {ta,ra,B,sgnData,KUb[Kab]}
2-B {tb,rb,A,sgnData,KUa[Kab]}
B

Three-Way Authentication: Three-Way Authentication includes a final message from A to B, which contains a signed copy of the nonce, so that timestamps need not be checked, for use when synchronized clocks are not available.



- 3 messages (A->B, B->A, A->B) which enables above authentication without synchronized clocks

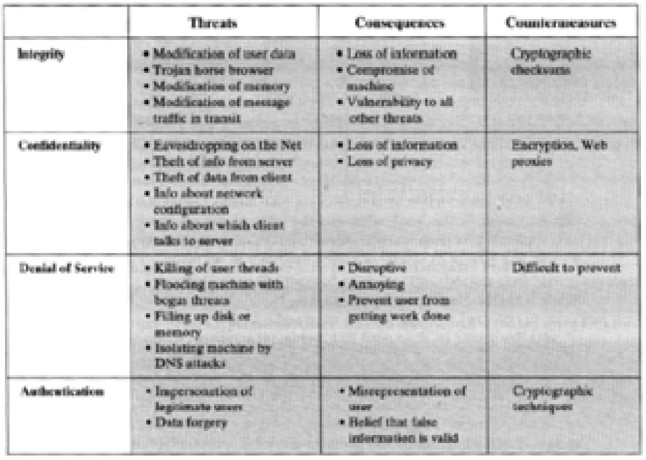
A
1- A {ta,ra,B,sgnData,KUb[Kab]}
2 -B {tb,rb,A,sgnData,KUa[Kab]}
3- A{rb}
B

Introduction:

Usage of internet for transferring or retrieving the data has got many benefits like speed, reliability, security etc. Much of the Internet's success and popularity lies in the fact that it is an open global network. At the same time, the fact that it is open and global makes it not very secure. The unique nature of the Internet makes exchanging information and transacting business over it inherently dangerous. The faceless, voiceless, unknown entities and individuals that share the Internet may or may not be who or what they profess to be. In addition, because the Internet is a global network, it does not recognize national borders and legal jurisdictions. As a result, the transacting parties may not be where they say they are and may not be subject to the same laws or regulations.

For the exchange of information and for commerce to be secure on any network, especially the Internet, a system or process must be put in place that satisfies requirements for confidentiality, access control, authentication, integrity, and non repudiation. These requirements are achieved on the Web through the use of encryption and by employing digital signature technology. There are many examples on the Web of the practical application of encryption. One of the most important is the SSL protocol.

A summary of types of security threats faced in using the Web is given below:

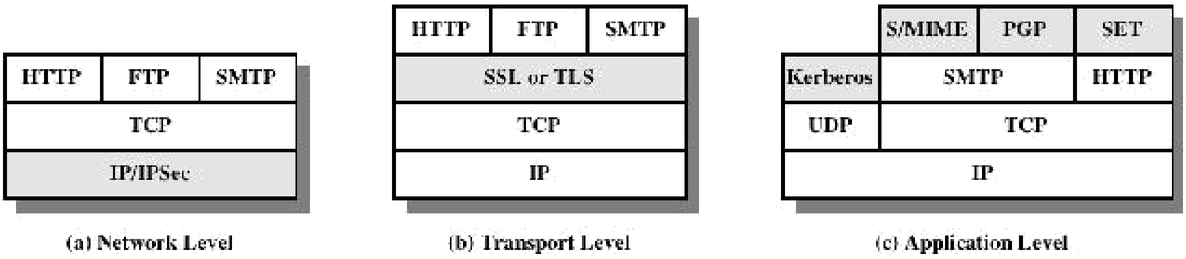


	Threats	Consequences	Countermeasures
Integrity	• Modification of user data • Trojan horse browser • Modification of memory • Modification of message traffic in transit	• Loss of information • Compromise of machine • Vulnerability to all other threats	Cryptographic checksums
Confidentiality	• Eavesdropping on the Net • Theft of info from server • Theft of data from client • Info about network configuration • Info about which client talks to server	• Loss of information • Loss of privacy	Encryption, Web proxies
Denial of Service	• Killing of user threads • Flooding machine with bogus threats • Filling up disk or memory • Isolating machine by DNS attacks	• Disruptive • Annoying • Prevent user from getting work done	Difficult to prevent
Authentication	• Impersonation of legitimate users • Data forgery	• Misrepresentation of user • Belief that false information is valid	Cryptographic techniques

One way of grouping the security threats is in terms of passive and active attacks. *Passive attacks* include eavesdropping on network traffic between browser and server and gaining access to information on a website that is supposed to be restricted. *Active attacks* include impersonating another user, altering messages in tr nsit between client and server and altering information on a website. Another way of classifying these security threats is in terms of location of the threat: Web server, Web browser and network traffic between browser and server.

Web Traffic Security Approaches

Various approaches for providing Web Security are available, where they are similar in the services they provide and also similar to some extent in the mechanisms they use. They differ with respect to their scope of applicability and their relative location within the TCP/IP protocol stack. The main approaches are IPSec, SSL or TLS and SET.



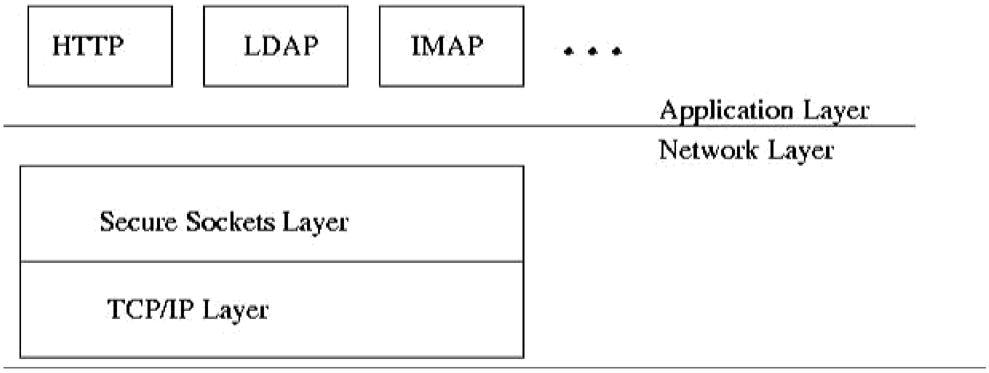
Relative location of Security Faculties in the TCP/IP Protocol Stack

IPSec provides security at the network level and the main advantage is that it is transparent to end users and applications. In addition, IPSec includes a filtering capability so that only selected traffic can be processed. **Secure Socket Layer or Transport Layer Security (SSL/TLS)** provides security just above the TCP at transport layer. Two implementation choices are present here. Firstly, the SSL/TLS can be implemented as a part of TCP/IP protocol suite, thereby being transparent to applications. Alternatively, SSL can be embedded in specific packages like SSL being implemented by Netscape and Microsoft Explorer browsers. **Secure Electronic Transaction (SET)** approach provides application-specific services i.e., according to the security requirements of a particular application. The main advantage of this approach is that service can be tailored to the specific needs of a given application.

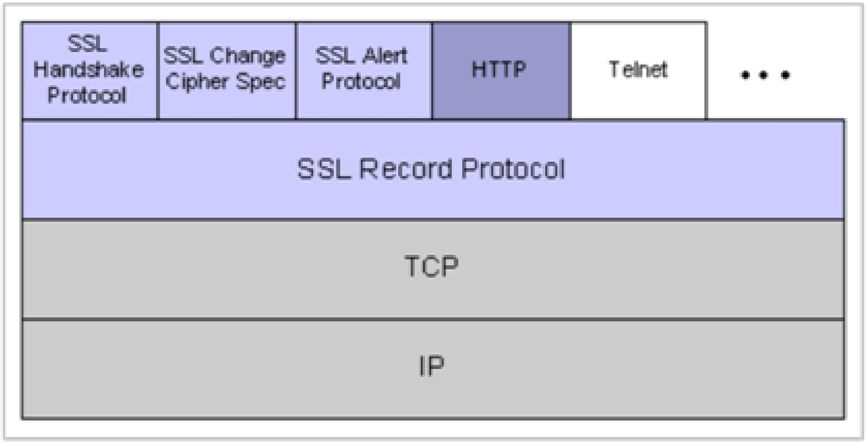
Secure Socket Layer/Transport Layer Security

SSL was developed by Netscape to provide security when transmitting information on the Internet. The Secure Sockets Layer protocol is a protocol layer which may be placed between a reliable connection-oriented network layer protocol (e.g. TCP/IP) and the application protocol layer (e.g. HTTP).

SSL runs above TCP/IP and below high-level application protocols



SSL provides for secure communication between client and server by allowing mutual authentication, the use of digital signatures for integrity and encryption for privacy. SSL protocol has different versions such as SSLv2.0, SSLv3.0, where SSLv3.0 has an advantage with the addition of support for certificate chain loading. SSL 3.0 is the basis for the Transport Layer Security [TLS] protocol standard. SSL is designed to make use of TCP to provide a reliable end-to-end secure service. SSL is not a single protocol, but rather two layers of protocols as shown below:



SSL Protocol Stack

The SSL Record Protocol provides basic security services to various higher-layer protocols. In particular, the Hypertext Transfer Protocol (HTTP), which provides the transfer service for Web client/server interaction, can operate on top of SSL. Three higher-layer protocols are defined as part of SSL: the Handshake Protocol, The Change Cipher Spec Protocol, and the Alert Protocol. Two important SSL concepts are the SSL session and the SSL connection, which are defined in the specification as follows:

- **Connection**: A connection is a transport (in the OSI layering model definition) that provides a suitable type of service. For SSL, such connections are peer-to-peer relationships. The connections are transient. Every connection is associated with one session.
- **Session**: An SSL session is an association between a client and a server. Sessions are created by the Handshake Protocol. Sessions define a set of cryptographic security parameters, which can be shared among multiple connections. Sessions are used to avoid the expensive negotiation of new security parameters for each connection.

An SSL session is *stateful*. Once a session is established, there is a current operating state for both read and write (i.e., receive and send). In addition, during the Handshake Protocol, pending read and write states are created. Upon successful conclusion of the Handshake Protocol, the pending states become the current states. An SSL session may include multiple secure connections; in addition, parties may have multiple simultaneous sessions.

A session state is defined by the following parameters:

- ***Session identifier***: An arbitrary byte sequence chosen by the server to identify an active or resumable session state.
- ***Peer certificate****:*An X509.v3 certificate of the peer. This element of the state may be null.
- ***Compression method***: The algorithm used to compress data prior to encryption.
- ***Cipher spec****:*Specifies the bulk data encryption algorithm (such as null, AES, etc.) and a hash algorithm (such as MD5 or SHA-1) used for MAC calculation. It also defines cryptographic attributes such as the hash_size.
- ***Master secret****:*48-byte secret shared between the client and server.
- ***Is resumable****:*A flag indicating whether the session can be used to initiate new connections.

A connection state is defined by the following parameters:

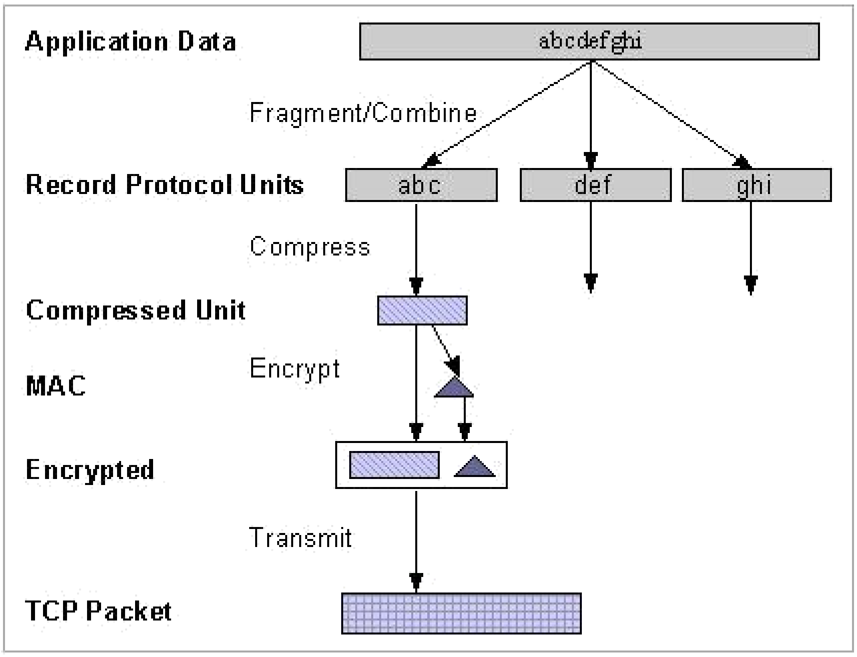
- ***Server and client random***: Byte sequences that are chosen by the server and client for each connection.
- ***Server write MAC secret***: The secret key used in MAC operations on data sent by the server.
- ***Client write MAC secret***: The secret key used in MAC operations on data sent by the client.
- ***Server write key:***The conventional encryption key for data encrypted by the server and decrypted by the client.
- ***Client write key***: The conventional encryption key for data encrypted by the client and decrypted by the server.
- ***Initialization vectors*****:** When a block cipher in CBC mode is used, an initialization vector (IV) is maintained for each key. This field is first initialized by the SSL Handshake Protocol. Thereafter the final ciphertext block from each record is preserved for use as the IV with the following record.
- ***Sequence numbers***: Each party maintains separate sequence numbers for transmitted and received messages for each connection. When a party sends or receives a change cipher spec message, the appropriate sequence number is set to zero. Sequence numbers may not exceed 264-1.

SSL Record Protocol

The SSL Record Protocol provides two services for SSL connections:

- Confidentiality: The Handshake Protocol defines a shared secret key that is used for conventional encryption of SSL payloads.
- Message Integrity: The Handshake Protocol also defines a shared secret key that is used to form a message authentication code (MAC).

The Record Protocol takes an application message to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies a MAC, encrypts, adds a header, and transmits the resulting unit in a TCP segment. Received data are decrypted, verified, decompressed, and reassembled and then delivered to higher-level users. The overall operation of the SSL Record Protocol is shown below:



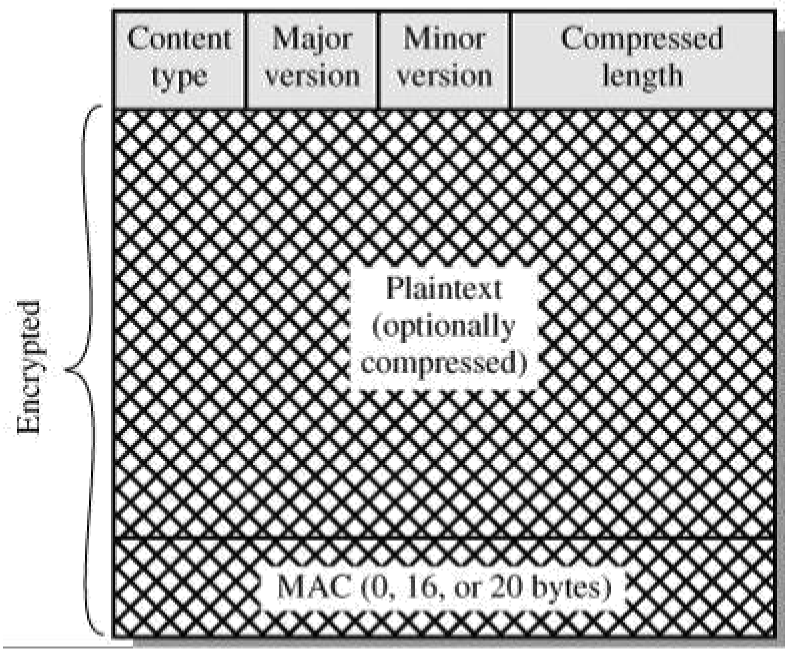
The first step is fragmentation. Each upper-layer message is fragmented into blocks of 214 bytes (16384 bytes) or less. Next, compression is optionally applied. Compression must be lossless and may not increase the content length by more than 1024 bytes. The next step in processing is to compute a message authentication code over the compressed data. For this purpose, a shared secret key is used. The calculation is defined as:

hash(MAC_write_secret || pad_2 ||
hash(MAC_write_secret || pad_1 || seq_num ||
SSLCompressed.type ||
SSLCompressed.length || SSLCompressed.fragment)) Where,

MAC_write_secret =	=	the byte 0x36 (0011 0110) repeated 48 times (384 bits) for MD5 and 40 times for
Secret shared key pad_1		
pad_2	=	the byte 0x5C (0101 1100) repeated 48 times for MD5 and 40 times for SHA-1

The main difference between HMAC and above calculation is that the two pads are concatenated in SSLv3 and are XORed in HMAC. Next, the compressed message plus the MAC are encrypted using symmetric encryption. Encryption may not increase the content length by more than 1024 bytes, so that the total length m y not exceed 2^{14} + 2048. The encryption algorithms allowed are AES-128/256, IDEA-128, DES-40, 3DES-168, RC2-40, Fortezza, RC4-40 and RC4-128. For stream encryption, the compressed message plus the MAC are encrypted whereas, for block encryption, padding may be added after the MAC prior to encryption.

SSL Record Format



The final step of SSL Record Protocol processing is to prepend a header, consisting of the following fields:

- Content Type (8 bits): The higher layer protocol used to process the enclosed fragment.
- Major Version (8 bits): Indicates major version of SSL in use. For SSLv3, the value is 3.
- Minor Version (8 bits): Indicates minor version in use. For SSLv3, the value is 0.
- Compressed Length (16 bits): The length in bytes of the plaintext fragment (or compressed fragment if compression is used). The maximum value is 2_{14} + 2048.

The content types that have been defined are change_cipher_spec, alert, handshake, and application_data.

SSL Change Cipher Spec Protocol

The Change Cipher Spec Protocol is one of the three SSL-specific protocols that use the SSL Record Protocol, and it is the simplest. This protocol consists of a single message, which consists of a single byte with the value 1.

The sole purpose of this message is to cause the pending st te to be copied into the current state, which updates the cipher suite to be used on th s connection.

SSL Alert Protocol

The Alert Protocol is used to convey SSL-r lat alerts to the peer entity. As with other applications that use SSL, alert messages are compressed and encrypted, as specified by the current state. Each me age in this protocol consists of two bytes.

The first byte takes the value warning(1) or fatal(2) to convey the severity of the message. If the level is fatal, SSL immediately terminates the connection. Other connections on the same session may continue, b t no new connections on this session may be established. The second byte contains a code that indicates the specific alert. The fatal alerts are listed below:

- unexpected_message: An inappropriate message was received.
- bad_record_mac: An incorrect MAC was received.
- decompression_failure: The decompression function received improper input (e.g., unable to decompress or decompress to greater than maximum allowable length).
- handshake_failure: Sender was unable to negotiate an acceptable set of security parameters given the options available.
- illegal_parameter: A field in a handshake message was out of range or inconsistent with other fields.

The remainder of the alerts are given below:

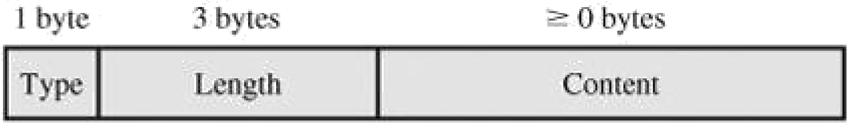
- close_notify: Notifies the recipient that the sender will not send any more messages on this connection. Each party is required to send a close_notify alert before closing the write side of a connection.
- no_certificate: May be sent in response to a certificate request if no appropriate certificate is available.
- bad_certificate: A received certificate was corrupt (e.g., contained a signature that did not verify).
- unsupported_certificate: The type of the received certificate is not supported.
- certificate_revoked: A certificate has been revoked by its signer.
- certificate_expired: A certificate has expired.
- certificate_unknown: Some other unspecified issue arose in processing the certificate, rendering it unacceptable.

SSL Handshake Protocol

SSL Handshake protocol ensures establishment of reliable nd secure session between client and server and also allows server & client to:

- authenticate each other
- to negotiate encryption & MAC algorithms
- to negotiate cryptographic keys to be used

The Handshake Protocol consists of a eries of messages exchanged by client and server. All of these have the format shown below and each message has three fields:



(c) Handshake Protocol

- **Type (1 byte**): Indicates one of 10 messages.
- **Length (3 bytes):** The length of the message in bytes.
- **Content (>=0 bytes):** The parameters associated with this message

The following figure shows the initial exchange needed to establish a logical connection between client and server. The exchange can be viewed as having four phases.

- Establish Security Capabilities
- Server Authentication and Key Exchange

o Client Authentication and Key Exchange

o Finish

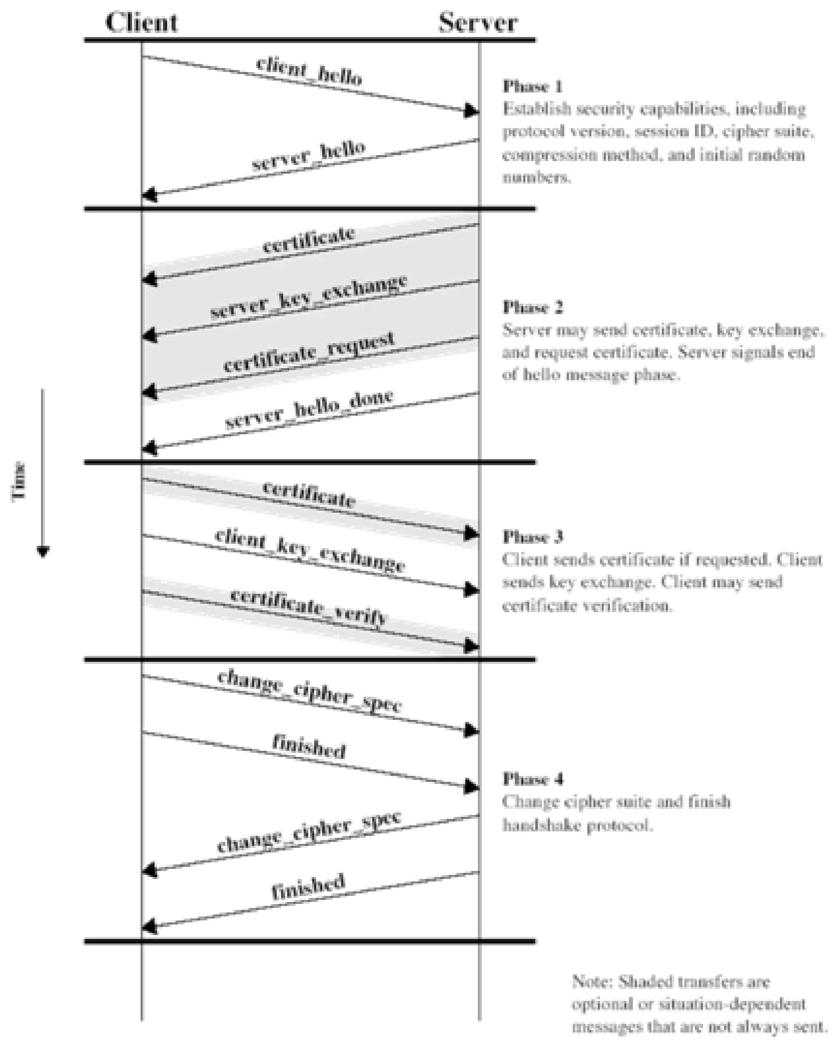
Phase 1. Establish Security Capabilities

This phase is used to initiate a logical connection and to establish the security capabilities that will be associated with it. The exchange is initiated by the client, which sends a client_hello message with the following parameters:

- Version: The highest SSL version understood by the client.
- Random: A client-generated random structure, consisting of a 32-bit timestamp and 28 bytes generated by a secure random number generator. These values serve as nonces and are used during key exchange to prevent replay attacks.

Session ID: A variable-length session identifier. A nonzero value indicates that the client wishes to update the parameters of an existing connection or create a new connection on this session. A zero value indicates that the cl ent wishes to establish a new connection on a new session.

- CipherSuite: This is a list that contains the combinations of cryptographic algorithms supported by the client, in decreasing order of preference. Each element of the list (each cipher suite) defines both a key exchange algorithm and a CipherSpec.
- Compression Method: This is a list of the compression methods the client supports.


Client
Server
client_hello
server_hello
Phase 1
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.
certificate
server_key_exchange
certificate_request
server_hello_done
Phase 2
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.
Time
certificate
client_key_exchange
certificate_verify
Phase 3
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.
change_cipher_spec
finished
change_cipher_spec
finished
Phase 4
Change cipher suite and finish handshake protocol.
Note: Shaded transfers are optional or situation-dependent messages that are not always sent.

Phase 2. Server Authentication and Key Exchange

The server begins this phase by sending its certificate via a certificate message, which contains one or a chain of X.509 certificates. The **certificate message** is required for any agreed-on key exchange method except anonymous Diffie-Hellman. Next, a **server_key_exchange** message may be sent if it is required. It is not required in two instances: (1) The server has sent a certificate with fixed Diffie-Hellman parameters, or (2) RSA key exchange is to be used.

Phase 3. Client Authentication and Key Exchange

Once the server_done message is received by client, it should verify whether a valid certificate is provided and check that the server_hello parameters are acceptable. If all is satisfactory, the client sends one or more messages back to the server. If the server has requested a certificate, the client begins this phase by sending a **certificate message**. If no suitable certificate is available, the client sends a no_certificate alert instead. Next is the **client_key_exchange** message, for which the content of the message depends on the type of key exchange.

Phase 4. Finish

This phase completes the setting up of a secure connection. The client sends a **change_cipher_spec** message and copies the pending CipherSpec into the current CipherSpec. The client then immediately sends the finished message under the new algorithms, keys, and secrets. The finished message verifies that the key exchange and authentication processes were successful.

Transport Layer Security

TLS was released in response to the Int rnet community's demands for a standardized protocol. TLS (Transport Layer Security), defined in RFC 2246, is a protocol for establishing a secure connection between a client and a server. TLS (Transport Layer Security) is capable of authenticating both the client and the server and creating a encrypted connection between the two. Many protocols use TLS (Transport Layer Security) to establish secure connections, including HTTP, IMAP, POP3, and SMTP. The TLS Handshake Protocol first negotiates key exchange using an asymmetric algorithm such as RSA or Diffie-Hellman. The TLS Record Protocol then begins opens an encrypted channel using a symmetric algorithm such as RC4, IDEA, DES, or 3DES. The TLS Record Protocol is also responsible for ensuring that the communications are not altered in transit. Hashing algorithms such as MD5 and SHA are used for this purpose. RFC 2246 is very similar to SSLv3. There are some minor differences ranging from protocol version numbers to generation of key material.

Version Number: The TLS Record Format is the same as that of the SSL Record Format and the fields in the header have the same meanings. The one difference is in version values. For the current version of TLS, the Major Version is 3 and the Minor Version is 1.

Message Authentication Code: Two differences arise one being the actual algorithm and the other being scope of MAC calculation. TLS makes use of the HMAC algorithm defined in RFC 2104. SSLv3 uses the same algorithm, except that the padding bytes are concatenated with the secret key rather than being XORed with the secret key padded to the block length. For TLS, the MAC calculation encompasses the fields indicated in the following expression:

HMAC_hash(MAC_write_secret, seq_num || TLSCompressed.type ||
TLSCompressed.version || TLSCompressed.length ||
TLSCompressed.fragment)

The MAC calculation covers all of the fields covered by the SSLv3 calculation, plus the field TLSCompressed.version, which is the version of the protocol being employed. Pseudorandom Function: TLS makes use of a pseudorandom function referred to as PRF to expand secrets into blocks of data for purposes of key generation or validation. The PRF is based on the following data expansion function:

P_hash(secret, seed) = HMAC_hash(secret, A(1) || seed) ||
HMAC_hash(secret, A(2) || seed) ||
HMAC_hash(secret, A(3) || seed) || ...

where A() is defined as

A(0) = seed

A(i) = HMAC_hash (secret, A(i - 1))

The data expansion function makes use of the HMAC algorithm, with either MD5 or SHA-1 as the underlying hash function. As can be seen, P_hash can be iterated as many times as necessary to produce the required quantity of data. each iteration involves two executions of HMAC, each of which in turn involves two executions of the underlying hash algorithm.