AUTHENTICATION REQUIREMENTS

In the context of communication across a network, the following attacks can be identified:

- **Disclosure** – releases of message contents to any person or process not possessing the appropriate cryptographic key.
- **Traffic analysis** – discovery of the pattern of traffic between parties.
- **Masquerade** – insertion of messages into the network fraudulent source.
- **Content modification** – changes to the content of the message, including insertion deletion, transposition and modification.
- **Sequence modification** – any modification to a sequence of messages between parties, including insertion, deletion and reordering.
- **Timing modification** – delay or replay of messages.
- **Source repudiation** – denial of transmission of message by source.
- **Destination repudiation** – denial of transmission of message by destination.

To deal with first two attacks are in the realm of message confidentiality. Measures to deal with 3 through 6 are regarded as message authentication. Item 7 comes under digital signature and dealing with item 8 may require a combination of digital signature and a protocol to counter this attack.

AUTHENTICATION FUNCTIONS

Any message authentication or digital signature mechanism can be viewed as having fundamentally two levels. At the lower level, there may be some sort of function that produces an authenticator: a value to be used to authenticate a message. This lower layer function is then used as primitive in a higher-layer authentication protocol that enables a receiver to verify the authenticity of a message.

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

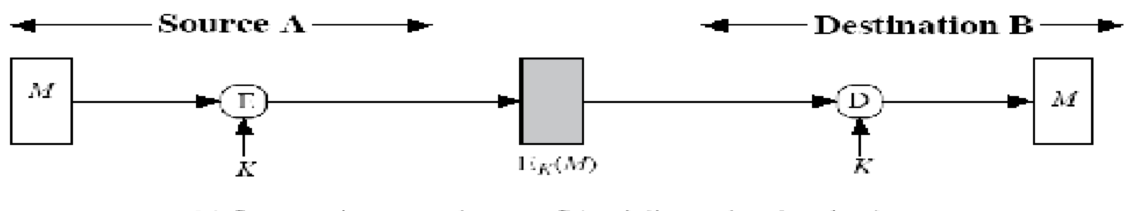
Message encryption – the cipher text of the entire message serves as its authenticator.

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

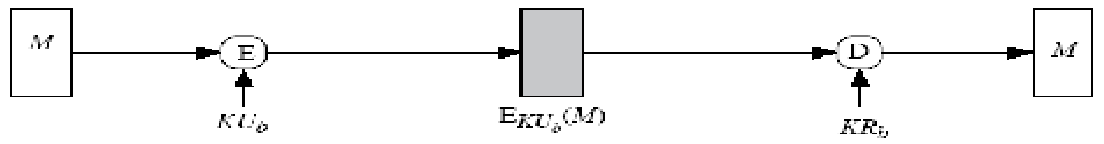
Hash function – a public function that maps a message of any length into a fixed length hash value, which serves as the authenticator. (NO KEY is used)

- **Message encryption**

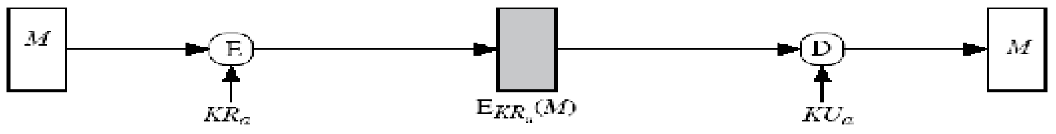
Message encryption by itself can provide a measure of authentication. The analysis differs from symmetric and public key encryption schemes.



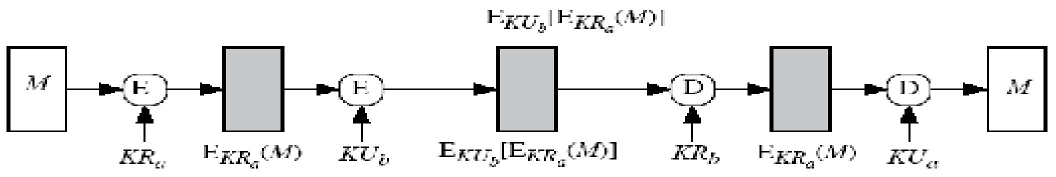
(a) Symmetric encryption: confidentiality and authentication



(b) Public key encryption: confidentiality



(c) Public-key encryption: authentication and signature



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

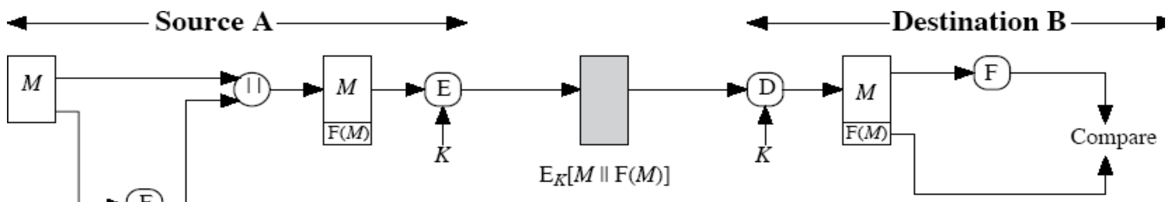
- Suppose the message can be any arbitrary bit pattern. In that case, there is no way to determine automatically, at the destination whether an incoming message is the ciphertext of a legitimate message.
- One solution to this problem is to force the plaintext to have some structure that is easily recognized but that cannot be replicated without recourse to the encryption function. for example, append an error detecting code, also known as Frame Check Sequence (FCS) or checksum to each message before encryption
- ‘A’ prepares a plaintext message M and then provides this as input to a function F that produces

an FCS. The FCS is appended to M and the entire block is then encrypted.

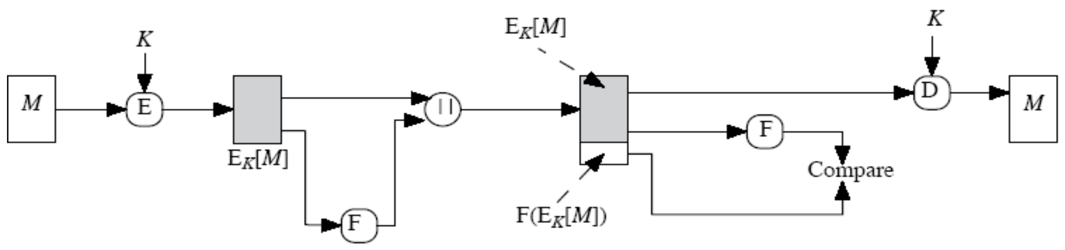
- At the destination, B decrypts the incoming block and treats the result as a message with an appended FCS. B applies the same function F to attempt to reproduce the FCS.

- If the calculated FCS is equal to the incoming FCS, then the message is considered authentic.

In the internal error control, the function F is applied to the plaintext, whereas in external error control, F is applied to the ciphertext (encrypted message).



(a) Internal error control



(b) External error control

- **MESSAGE AUTHENTICATION CODE (MAC)**

  - An alternative authentication technique involves the use of message and secret key to generate a small fixed size block of data, known as cryptographic checksum or MAC that is appended to the message.
  - This technique assumes that two communication parties say A and B, share a common secret key ‘k’. When A has to send a message to B, it calculates the MAC as a function of the message and the key.

$$MAC=C_K(M)$$

Where M – input message

C – MAC function

K – Shared secret key

- The message plus MAC are transmitted to the intended recipient. The recipient performs the same calculation on the received message, using the shared secret key, to generate a new MAC. The received MAC is compared to the calculated MAC. If it is equal, then the message is considered authentic.
- A MAC function is similar to encryption. One difference is that MAC algorithm need not be reversible, as it must for decryption. In general, the MAC function is a many- to-one function.

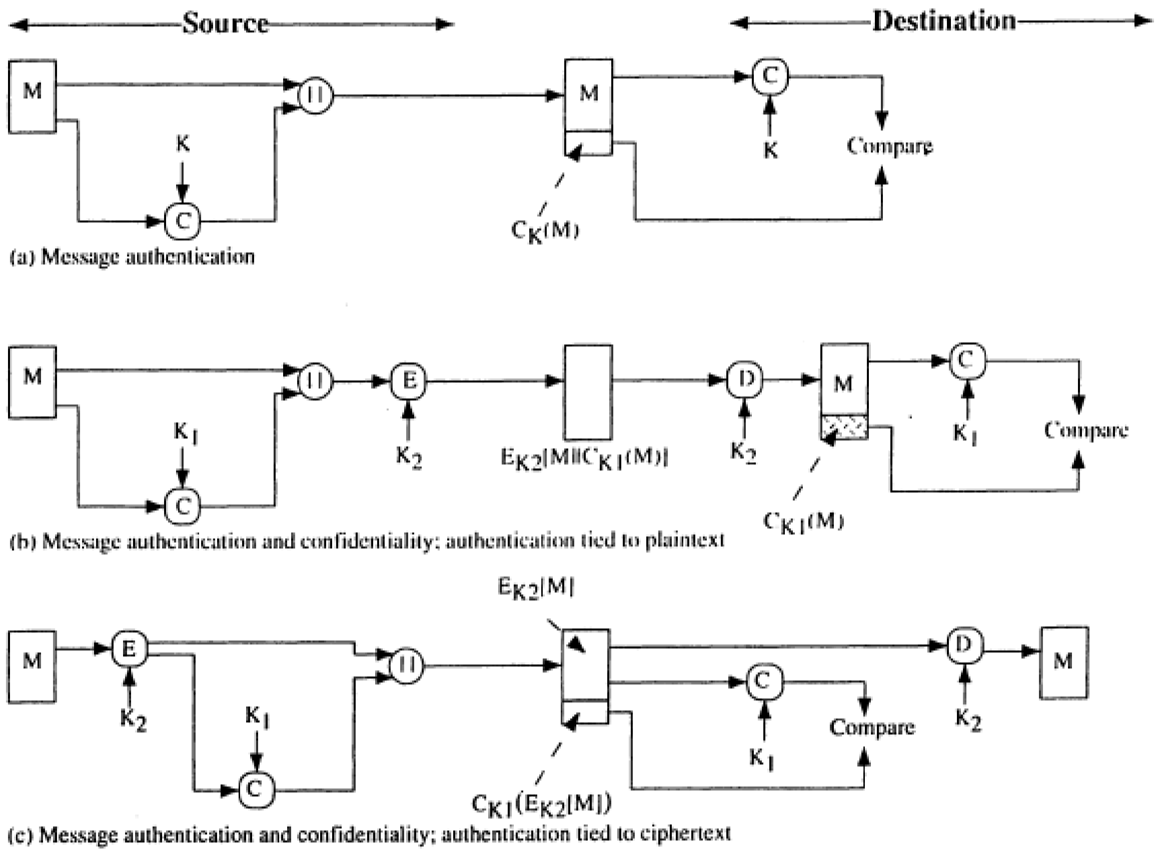

Source
Destination
M
K
C
||
Compare
C_K(M)
(a) Message authentication
K1
K2
E
D
E_K2[M||C_K1(M)]
C_K1(M)
(b) Message authentication and confidentiality; authentication tied to plaintext
E_K2[M]
C_K1(E_K2[M])
(c) Message authentication and confidentiality; authentication tied to ciphertext

Fig .Basic uses of Message Authentication Code

MAC based on DES

- One of the most widely used MACs, referred to as Data Authentication Algorithm (DAA) is based on DES.
- The algorithm can be defined as using cipher block chaining (CBC) mode of operation of DES with an initialization vector of zero.
- The data to be authenticated are grouped into contiguous 64-bit blocks: D_1, D_2 … D_n. if necessary, the final block is padded on the right with zeros to form a full 64-bit block.

- Using the DES encryption algorithm and a secret key, a data authentication code (DAC) is calculated as follows:

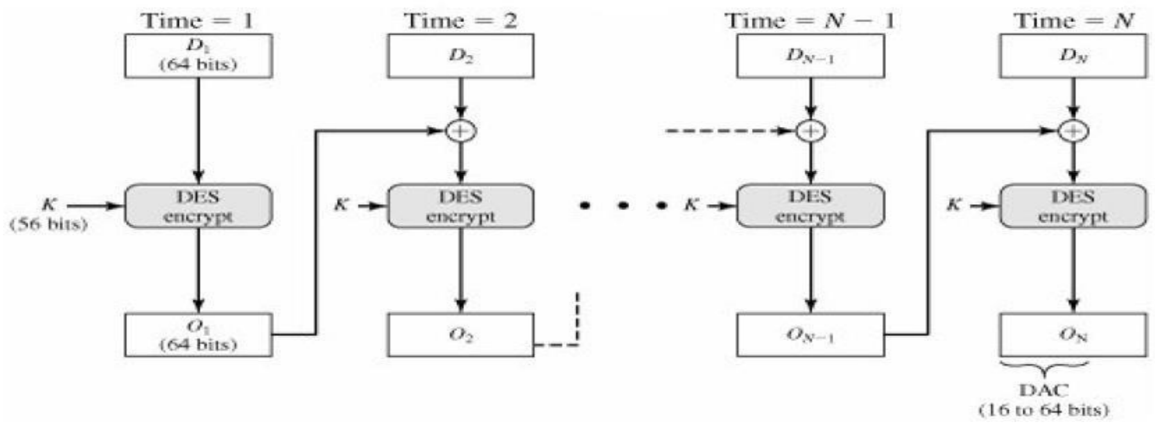
$$O_1 = E_K(D_1)$$

$$O_2 = E_K(D_2 \oplus O_1)$$

$$O_3 = E_K(D_3 \oplus O_2)\ldots$$

$$O_N = E_K(D_N \oplus O_{N-1})$$

Figure Data Authentication Algorithm (DAA)

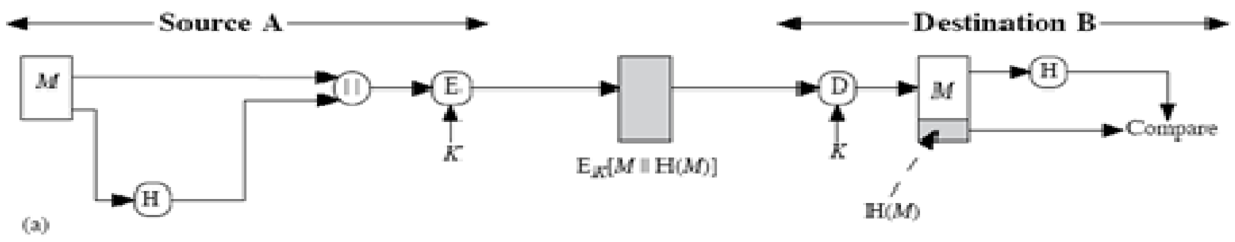


- **HASH FUNCTIONS**

A variation on the message authentication code is the one way hash function. As with MAC, a hash function accepts a variable size message M as input and produces a fixed-size output, referred to as hash code H (M). Unlike a MAC, a hash code does not use a key but is a function only of the input message. The hash code is also referred to as a message digest or hash value.

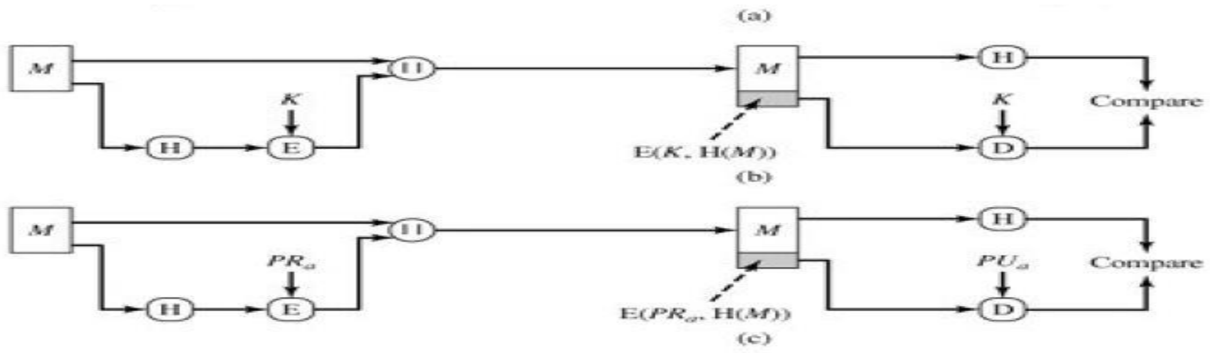
There are varieties of ways in which a hash code can be used to provide message authentication, as follows:

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

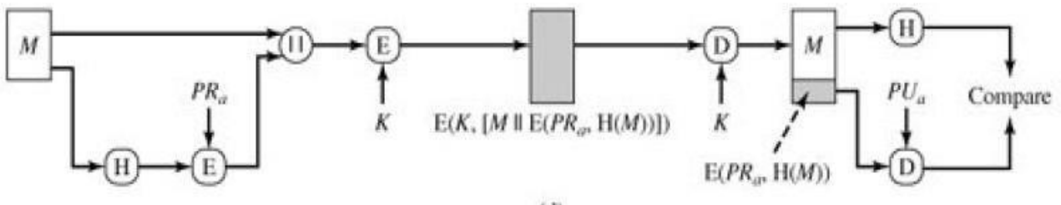


b) Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality.

c) Only the hash code is encrypted, using public-key encryption and using the sender's private key. As with (b), this provides authentication. It also provides a digital signature, because only the sender could have produced the encrypted hash code. In fact, this is the essence of the digital signature technique.

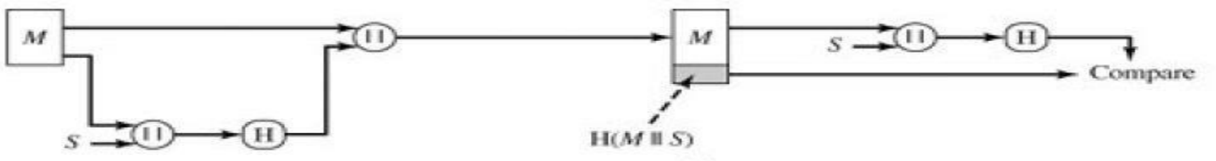


d) If confidentiality as well as a digital signature is desired, then the message plus the private-key encrypted hash code can be encrypted using a symmetric secret key. This is a common technique.



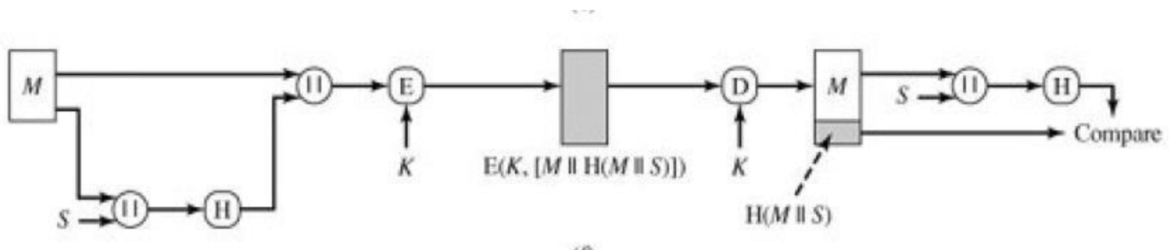
(d)

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



(e)

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



(f)

A hash value h is generated by a function H of the form h = H (M)
Where M is a variable-length message and H (M) is the fixed-length hash value. The hash value is appended to the message at the source at a time when the message is assumed or known to be correct. The receiver authenticates that message by re-computing the hash value.

Requirements for a Hash Function (H)

1. H can be applied to a block of data of anysize.
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 value 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.
5. For any given block x, it is computationally infeasible to find y. x such that H(y) = H(x). This is sometimes referred to as **weak collision resistance**.
6. It is computationally infeasible to find any pair (x, y) such that H(x) = H(y). This is sometimes referred to as **strong collision resistance**.

The first three properties are requirements for the practical application of a hash function to message authentication. The fourth property, the one-way property, states that it is easy to generate a code given a message but virtually impossible to generate a message given a code. The fifth property guarantees that an alternative message hashing to the same value as a given message cannot be found. This prevents forgery when an encrypted hash code is used. The sixth property refers to how resistant the hash function is to a type of attack known as the birthday attack, which we examine shortly.

Simple Hash Functions

All hash functions operate using the following general principles. The input (message, file, etc.) is viewed as a sequence of n-bit blocks. The input is processed one block at a time in an iterative fashion to produce an n-bit hash function.

One of the simplest hash functions is the bit-by-bit exclusive-OR (XOR) of every block. This can be expressed as follows:

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

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

m = number of n-bit blocks in the input bij= i_{th} bit in j_{th} block

$\oplus$= XOR operation

A simple way to improve matters is to perform a one-bit circular shift, or rotation, on the hash value after each block is processed. The procedure can be summarized as follows:

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

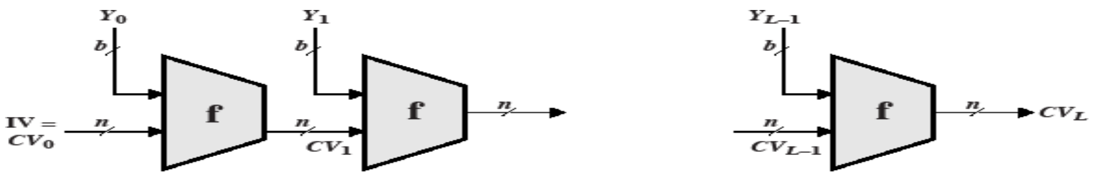
2. Process each successive n-bit block of data as follows:

a. Rotate the current hash value to the left by one bit.

b. XOR the block into the hash value.

Hash Functions

- In recent years, there has been considerable effort, and some successes, in developing cryptanalytic attacks on hash functions. To understand these, we need to look at the overall structure of a typical secure hash function, and is the structure of most hash functions in use today, including SHA and Whirlpool.
- The hash function takes an input message and partitions it into L fixed-sized blocks of b bits each. If necessary, the final block is padded to b bits.
- The final block also includes the value of the total length of the input to the hash function. The inclusion of the length makes the job of the opponent more difficult.
- Either the opponent must find two messages of equal length that hash to the same value or two messages of differing lengths that, together with their length values, hash to the same value.



IV	=	Initial value
CV	=	chaining variable
Y_i	=	ith input block
f	=	compression algorithm
L	=	number of input blocks
n	=	length of hash code
b	=	length of input block

Figure 11.10 General Structure of Secure Hash Code

- The hash algorithm involves repeated use of a **compression function**, f that takes two inputs (an n-bit input from the previous step, called the chaining variable, and a b-bit block) and produces an n-bit output.
- At the start of hashing, the chaining variable has an initial value that is specified as part of the algorithm. The final value of the chaining variable is the hash value. Often, b > n; hence the term compression. The hash function can be summarized as follows:

$$CV_0 = IV = \text{initial n-bit value}$$

$$CV_i = f(CV_{i-1}, Y_{i-1}) \quad 1 \leq i \leq L$$

$$H(M) = CV_L$$

- Where the input to the hash function is a message M consisting of the blocks Y_0, Y_1..Y_{L-1} the structure can be used to produce a secure hash function to operate on a message of any length.

MD5

- Family of one-way hash functions by Ronald Rivest
- MD2 is the oldest, produces a 128-bit hash value, and is regarded as slower and less secure than MD4 and MD5
- MD4 produces a 128-bit hash of the message, using bit operations on 32-bit operands for fast implementation
- MD5 was designed as a strengthened version, using four rounds, a little more complex than in MD4
- A little progress at crypt analyzing MD5 has been made with a small number of collisions having been found

- Both MD4 and MD5 are still in use and considered secure in most practical applications
- Both are specified as Internet standards.

MD5 LOGIC

- Step.1 : Append padding bits
- Step.2 : Append length
- Step.3 : Initialize MD buffer
- Step.4: Process message in 512 bits

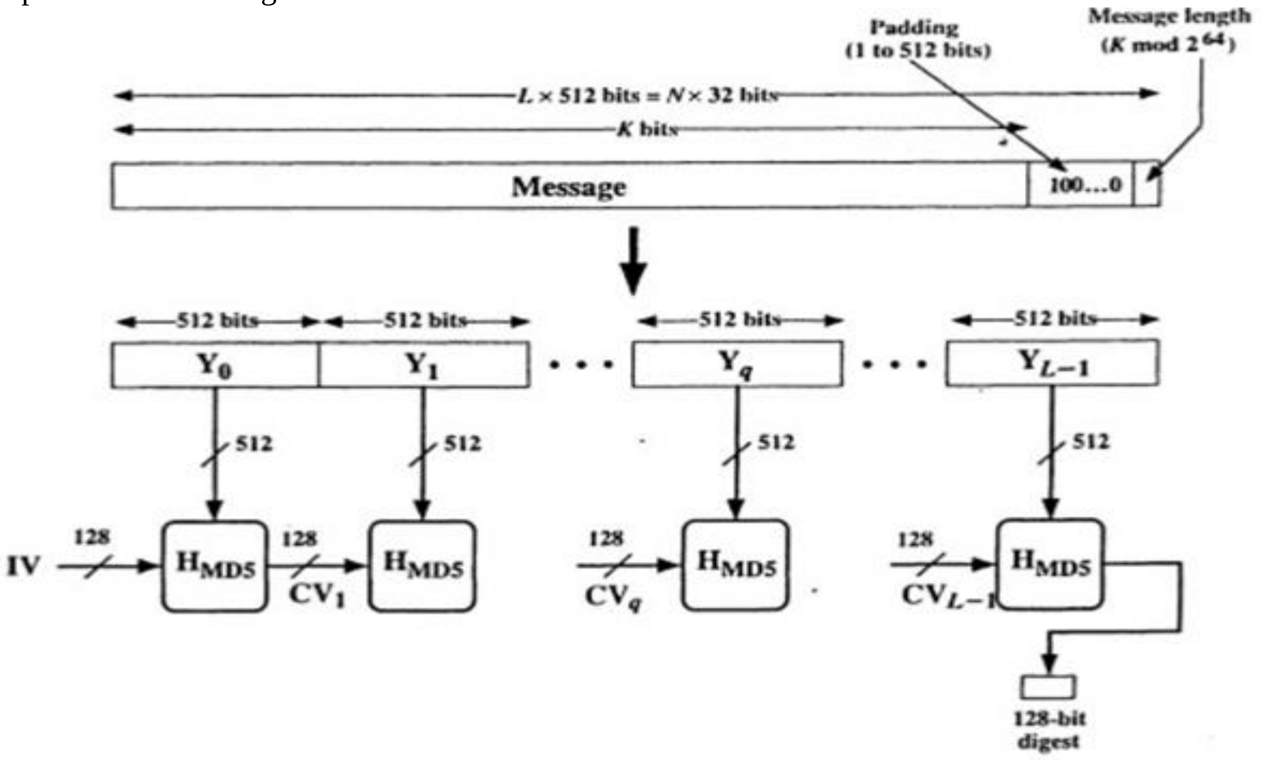


Fig. Message Digest generation by MD5

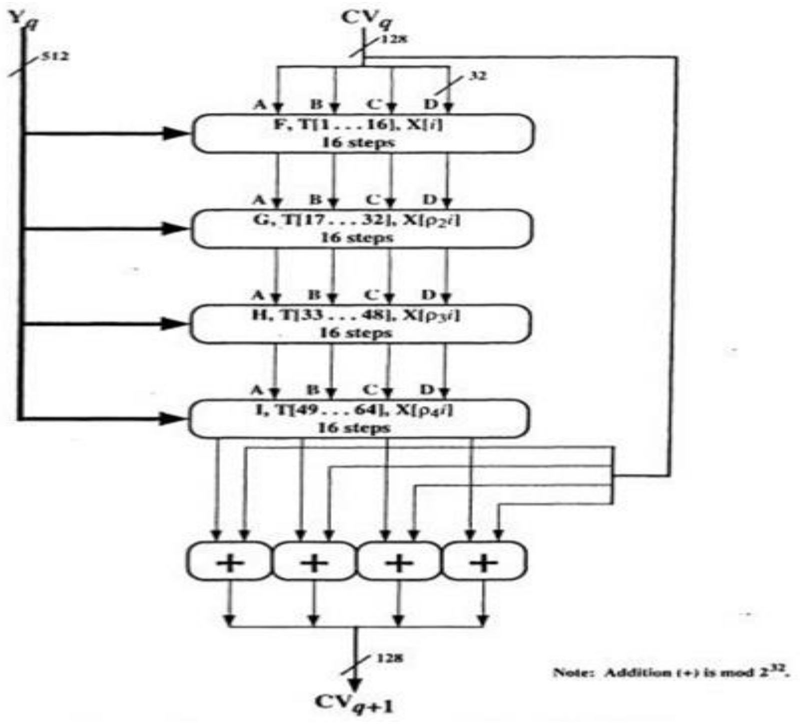

Y$_q$
512
CV$_q$
128
32
A B C D
F, T[1 . . . 16], X[i]
16 steps
G, T[17 . . . 32], X[ρ$_2$i]
16 steps
H, T[33 . . . 48], X[ρ$_3$i]
16 steps
I, T[49 . . . 64], X[ρ$_4$i]
16 steps
+
128
CV$_{q+1}$
Note: Addition (+) is mod 2^{32}.

Fig. Processing of single 512 bits by MD5

DIGITAL SIGNATURES

Have looked at message authentication- but does not address issues of lack of trust

- Digital signatures provide the ability to:
  - Verify author, date & time of signature
    - ♦ Authenticate message contents
    - ♦ Be verified by third parties to resolve disputes
      - ➢ Hence include authentication function with additional capabilities

Digital Signature Properties

- must depend on the message signed
- must use information unique to sender-to prevent both forgery and denial
- must be relatively easy to produce
- must be relatively easy to recognize & verify
- be computationally infeasible to forge
- with new message for existing digital signature
- with fraudulent digital signature for given message
- be practical save digital signature in storage

DIRECT DIGITAL SIGNATURES

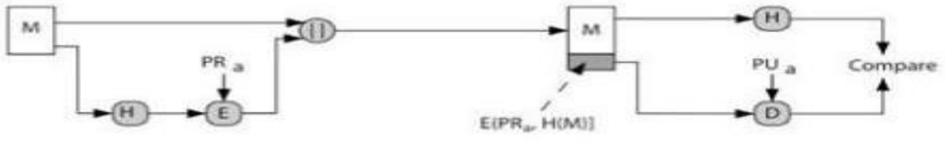
- Involve only sender & receiver
- Assumed receiver has sender's public-key
- Digital signature made by sender signing entire message or hash with private-key
- Can encrypt using receivers public-key
- Important that sign first then encrypt message & signature
- Security depends on sender's private-key

ARBITRATED DIGITAL SIGNATURES

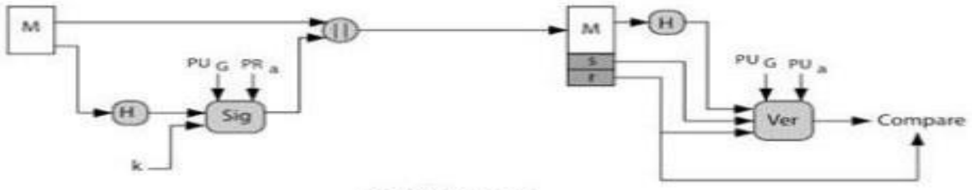
- Involves use of arbiter A
- Validates any signed message then dated and sent to recipient
- Requires suitable level of trust in arbiter
- Can be implemented with either private or public-key algorithms
- Arbiter may or may not see message

DIGITAL SIGNATURE STANDARD (DSS)

US Govt approved signature scheme designed by NIST & NSA in early 90's revised in 1993, 1996 & then 2000 uses the SHA hash algorithm .DSS is the standard, DSA is the algorithm



(a) RSA Approach



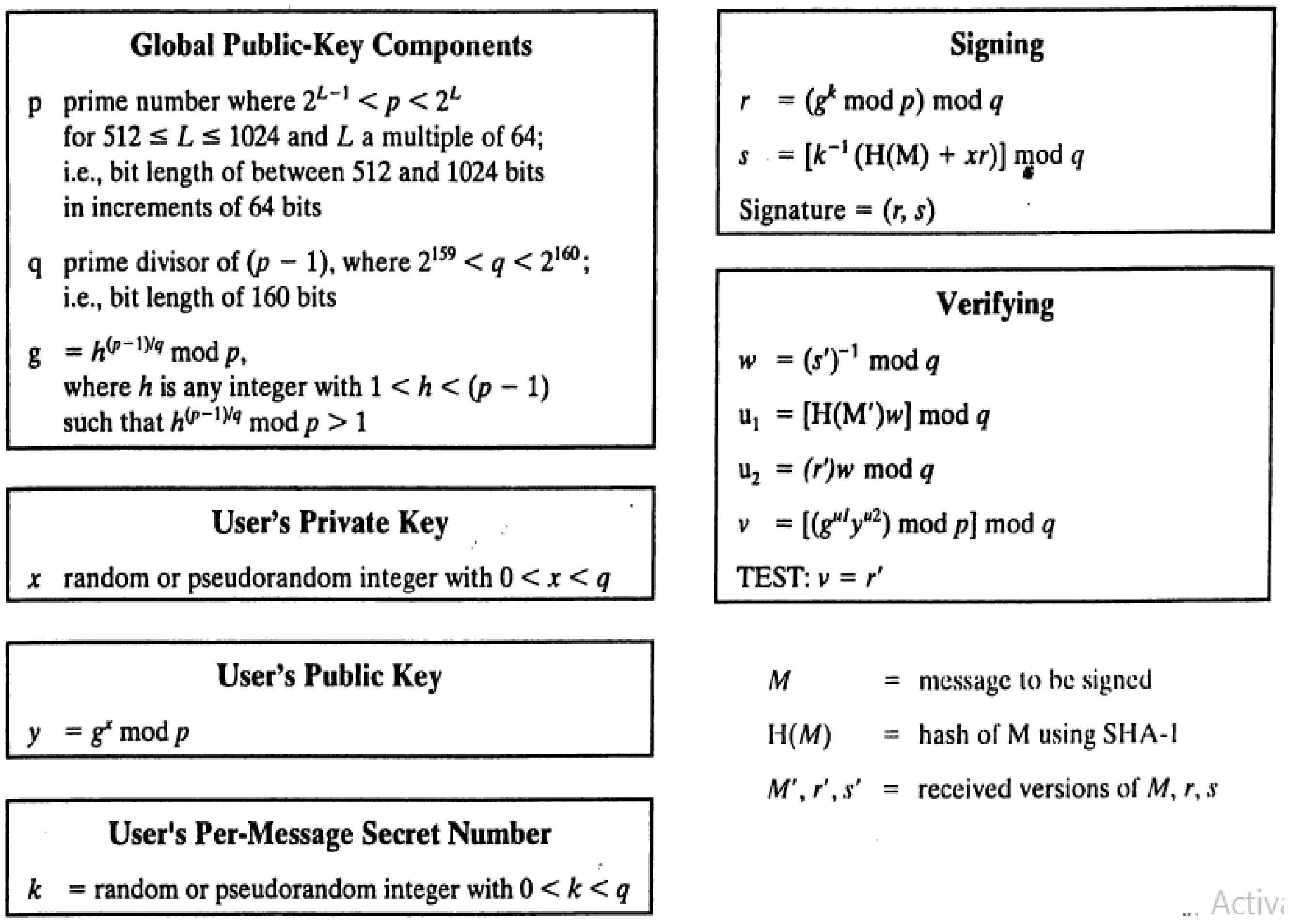
(b) DSS Approach

The Digital Signature Algorithm

The DSA is based on the difficulty of computing discrete logarithms.

There are three parameters in the algorithm that are public and can be common to a group of users. A 160-bit prime number q is chosen. Next, a prime number p is selected with a length between 512 and 1024 bits such that q divides (p - 1). Finally, g is chosen to be of the form h(p1)/qmod p where h is an integer between 1 and (p - 1) with the restriction that g must be greater than 1.

With these numbers in hand, each user selects a private key and generates a public key. The private key x must be a number from 1 to (q- 1) and should be chosen randomly. The public key is calculated from the private key as y = gx mod p. The calculation of y given x is relatively straightforward. However, given the public key y, it is believed to be computationally infeasible to determine x, which is the discrete logarithm of y to the base g, mod p



Global Public-Key Components

p prime number where $2^{L-1} < p < 2^{L}$ for $512 \le L \le 1024$ and L a multiple of 64; i.e., bit length of between 512 and 1024 bits in increments of 64 bits

q prime divisor of $(p - 1)$, where $2^{159} < q < 2^{160}$; i.e., bit length of 160 bits

g $= h^{(p-1)/q} \bmod p$, where h is any integer with $1 < h < (p - 1)$ such that $h^{(p-1)/q} \bmod p > 1$

User's Private Key

x random or pseudorandom integer with $0 < x < q$

User's Public Key

$y = g^{x} \bmod p$

User's Per-Message Secret Number

k = random or pseudorandom integer with $0 < k < q$

Signing

$r = (g^{k} \bmod p) \bmod q$

$s = [k^{-1}(H(M) + xr)] \bmod q$

Signature = (r, s)

Verifying

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

$u_1 = [H(M')w] \bmod q$

$u_2 = (r')w \bmod q$

$v = [(g^{u1} y^{u2}) \bmod p] \bmod q$

TEST: $v = r'$

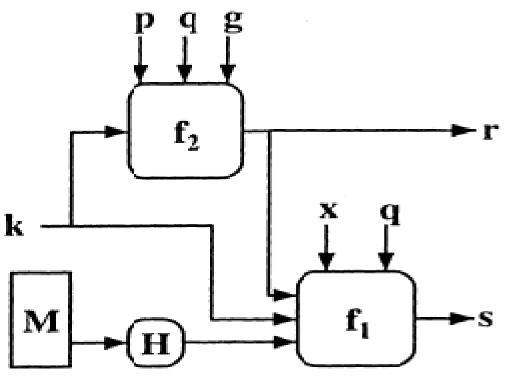
M	=	message to be signed
$H(M)$	=	hash of M using SHA-1
M', r', s'	=	received versions of M, r, s

Fig. **Digital Signature Algorithm**

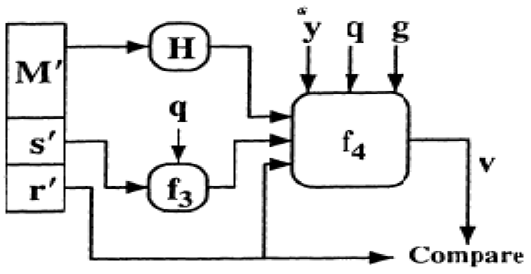
To create a signature, a user calculates two quantities, r and s, that are functions of the public key components (p, q, g), the user's private key (x), the hash code of the message, H(M), and an additional integer k that should be generated randomly and be unique for each signing.

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

The structure of the algorithm, as revealed in Figure below, is quite interesting. Note that the test at the end is on the value r, which does not depend on the message at all. Instead, r is a function of k and the three global public-key components. The multiplicative inverse of k (mod q) is passed to a function that also has as inputs the message hash code and the user's private key. The structure of this function is such that the receiver can recover r using the incoming message and signature, the public key of the user, and the global public key Given the difficulty of taking discrete logarithms, it is infeasible for an opponent to recover k from r or to recover x from s



$$s = f_1(H(M), k, x, r, q) = (k^{-1}(H(M) + xr)) \bmod q$$
$$r = f_2(k, p, q, g) = (g^k \bmod p) \bmod q$$



$$w = f_3(s', q) = (s')^{-1} \bmod q$$
$$v = f_4(y, q, g, H(M'), w, r')$$
$$= ((g^{(H(M')w) \bmod q}\, y^{r'w \bmod q}) \bmod p) \bmod q$$

Fig(a) signing **Fig(b) Verifying**

WEB SECURITY

Web now widely used by business, government, individuals but Internet & Web are vulnerable have a variety of threats

- integrity
- confidentiality
- denial of service
- authentication

Hence it needed added security mechanisms

SECURE SOCKET LAYER (SSL)

Transport layer security service originally developed by Netscape. Version 3 designed with public input and subsequently became Internet standard known as TLS (Transport Layer Security) uses TCP to provide a reliable end-to-end service.

SSL probably most widely used Web security mechanism. It's implemented at the Transport layer; of IPSec at Network layer; or various Application layer mechanisms eg.S/MIME & SET.

SSL is not a single protocol but rather two layers of protocol, as shown in the below figure. The figure shows the 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 also defined as part of SSL: the Handshake Protocol, Change Cipher Spec Protocol, and Alert Protocol. These SSL-specific protocols are used in the management of SSL exchanges.

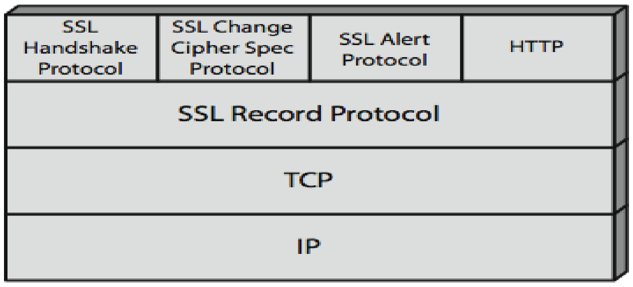


Fig. SSL Protocol stack

SSL Architecture

Two important SSL concepts are the SSL connection and the SSL session:

- Connection: A connection is a network transport that provides a suitable type of service, such connections are transient, peer-to-peer relationships, associated with one session
- Session: An SSL session is an association between a client and a server, 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.

SSL Record Protocol Services

SSL Record Protocol defines two services for SSL connections:

• Message Integrity: The Handshake Protocol also defines a shared secret key that is used to form a message authentication code (MAC), which is similar to HMAC

• Confidentiality: The Handshake Protocol defines a shared secret key that is used for conventional encryption of SSL payloads. The message is compressed before being concatenated with the MAC and encrypted, with a range of ciphers being supported as shown.

SSL Record Protocol Operation

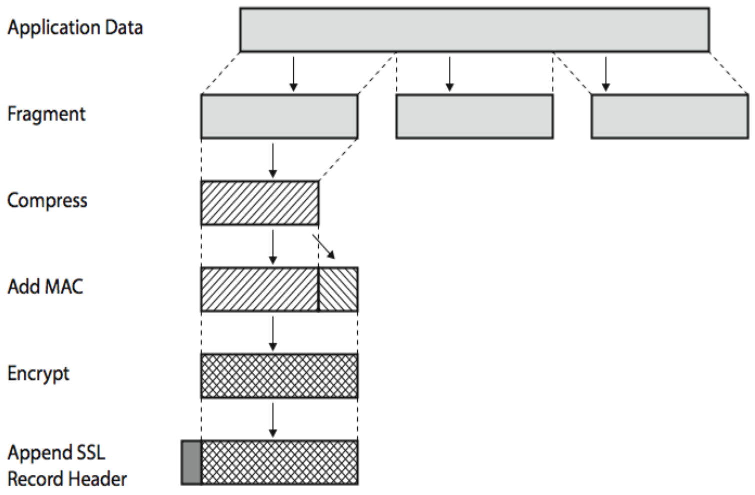


Fig. SSL Record Protocol operation

Figure above shows the overall operation of the SSL Record Protocol. 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-layer applications.

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, consisting of a single message. Its purpose is to cause the pending state to be copied into the current state, which updates the cipher suite to be used on this connection

SSL Alert Protocol

The Alert Protocol is used to convey SSL-related 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 message in this protocol consists of two bytes; the first takes the value warning (1) or fatal (2) to convey the severity of the message. The second byte contains a code that indicates the specific alert. The first group shown is the fatal alerts, the others are warnings

SSL Handshake Protocol

The most complex part of SSL is the Handshake Protocol. This protocol allows the server and client to authenticate each other and to negotiate an encryption and MAC algorithm and cryptographic keys to be used to protect data sent in an SSL record. The Handshake Protocol is used before any application data is transmitted. The Handshake Protocol consists of a series of messages exchanged by client and server, which can be viewed in 4 phases:

- Phase 1. Establish Security Capabilities - this phase is used by the client to initiate a logical connection and to establish the security capabilities that will be associated with it
- Phase 2. Server Authentication and Key Exchange - the server begins this phase by sending its certificate if it needs to be authenticated.
- Phase 3. Client Authentication and Key Exchange - the client should verify that the server provided a valid certificate if required and check that the server_hello parameters are acceptable
- 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

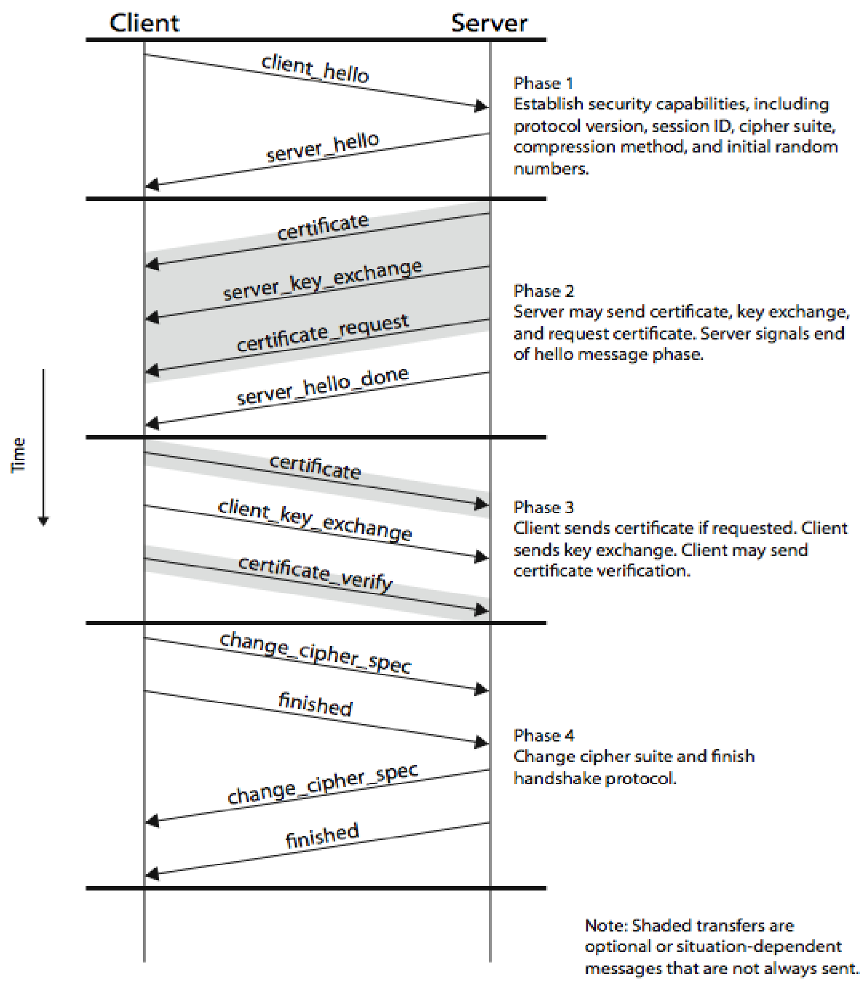

Client
Server
client_hello
server_hello
Phase 1
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.
certificate
server_key_exchange
certificate_request
server_hello_done
Phase 2
Server may send certificate, key exchange, 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.

Fig. **SSL Handshake Protocol action**

Figure above shows the initial exchange needed to establish a logical connection between client and server. The exchange can be viewed as having the four phases discussed previously

Transport Layer Security (TLS)

TLS is an IETF standardization initiative whose goal is to produce an Internet standard version of SSL. TLS is defined as a Proposed Internet Standard in RFC 2246. RFC 2246 is very similar to SSLv3, but with a number of minor differences in record format version number

- Uses HMAC for MAC
- A pseudo-random function expands secrets
- Has additional alert codes
- Some changes in supported ciphers

- Changes in certificate types & negotiations
- Changes in crypto computations & padding

Secure Electronic Transactions (SET)

SET is an open encryption and security specification designed to protect credit card transactions on the Internet. SETv1 emerged from a call for security standards by MasterCard and Visa in 1996. Beginning in 1996, there have been numerous tests of the concept, and by 1998 the first wave of SET-compliant products was available. SET is not itself a payment system, rather it is a set of security protocols and formats that enables users to employ the existing credit card payment infrastructure on an open network, such as the Internet, in a secure fashion, by providing:

• A secure communications channel among all parties involved in a transaction

• Trust through the use of X.509v3 digital certificates

• Privacy because the information is only available to parties in a transaction when and where necessary.

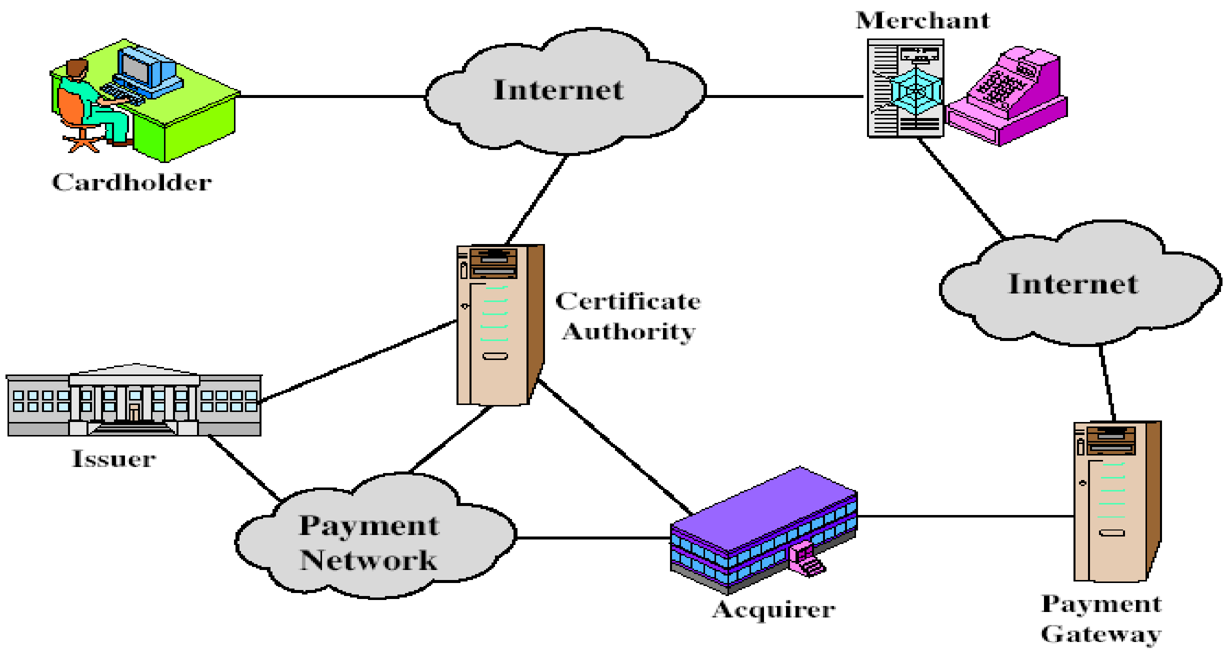


Fig. SET Component

Figure above indicates the participants in the SET system, being:

➢ Cardholder: purchasers interact with merchants from personal computers over the Internet

➢ Merchant: a person or organization that has goods or services to sell to the cardholder

➢ Issuer: a financial institution, such as a bank, that provides the cardholder with the payment card.

- Acquirer: a financial institution that establishes an account with a merchant and processes payment card authorizations and payments
- Payment gateway: a function operated by the acquirer or a designated third party that processes merchant payment messages
- Certification authority (CA): an entity that is trusted to issue X.509v3 public-key certificates for cardholders, merchants, and payment gateways

Detailed sequence of events that are required for a transaction as

1. customer opens account
2. customer receives a certificate
3. merchants have their own certificates
4. customer places an order
5. merchant is verified
6. order and payment are sent
7. merchant requests payment authorization
8. merchant confirms order
9. merchant provides goods or service
10. merchant requests payment

Dual signature

The purpose of the SET dual signature is to link two messages that are intended for two different recipients, the order information (OI) for the merchant and the payment information (PI) for the bank. The merchant does not need to know the customer's credit card number, and the bank does not need to know the details of the customer's order, however the two items must be linked in a way that can be used to resolve disputes if necessary. The customer takes the hash of the PI and the hash of the OI, concatenates them, and hashes the result. Finally, the customer encrypts the final hash with his or her private signature key, creating the dual signature. This can be summarized as:

$$DS=E(PRc, [H(H(PI)||H(OI))])$$