USER AUTHENTICATION, EMAIL SECURITY & WEB SECURITY

Syllabus: Transport Level Security: Web Security Requirements, Secure Socket Layer (SSL) and Transport Layer Security (TLS), Secure Shell(SSH) Electronic Mail Security: Pretty Good Privacy (PGP) and S/MIME.

Electronic Mail Security

In all distributed environments e-mail is the most heavily used network based application. It is also the only distributed application that is widely used across all architectures and vendor platforms. Users expect to be able to, and do send mail to others who are connected directly or indirectly to the Internet.

With the explosively growing reliance on e-mail for every conceivable purpose, there grows a demand for confidentiality and authentication services. Here are two schemes that provide these two services by name PGP (Pretty Good Privacy) and S/MIME (Secure Multipurpose and Internet Mail Extensions).

Pretty Good Privacy: PGP is an effort of a single person Phil Zimmerman. PGP provides confidentiality and authentication service that can be used for e-mail and file storage applications. In essence he selected

1. The best available cryptographic algorithms
2. Integrated these algorithms into a general purpose application
3. PGP is a freely available software
4. PGP runs on variety of platforms DOS/Windows/Unix/Mac and many more.

Notations:

K_s : Session key used for conventional encryption

KR_a : Private key of user A used in public key encryption

KU_a : Public key of user A used in public key encryption

EP: Public key encryption

DP: Public Key decryption

EC: Conventional encryption

DC: Conventional decryption

H: Hash function

||: concatenation

Z: Compression

R64: Radix 64 conversion

Operational Description:

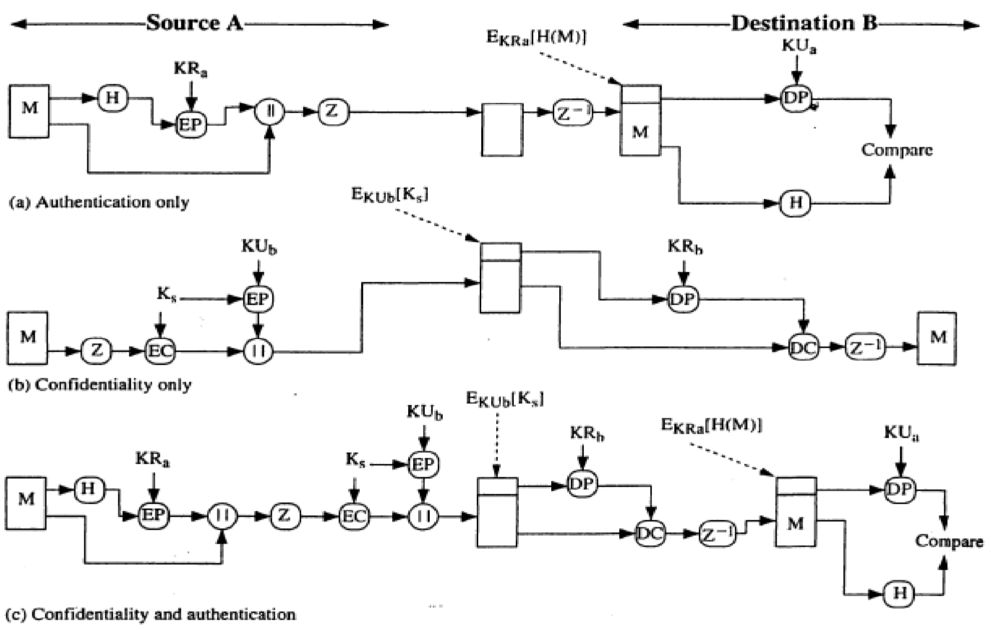
The actual operation of PGP implements the following five functions:

1. Digital Signature : Signature is created by using DSS or RSA algorithm. To create a signature, first a hash code is calculated on the message using SHA-1 or MD5. This hash code is encrypted with private key of the source KR_a using DSS or RSA.
2. Message encryption: A message is encrypted using CAST128 or IDEA or 3DES with a onetime session key. The session key is encrypted with receivers public key using public key encryption.
3. Compression: Messages are compressed for storage or transmission using ZIP. Compression must be provided always after authentication and before confidentiality serivce.

Why compression must be after authentication and before confidentiality?

- It is preferable to sign an uncompressed message so that one can store only the uncompressed message together with the signature for future verification. Suppose if one signed a compressed document, either he has to store compressed message for later verification or he has to decompress the message whenever verification is required. It is always preferable to authenticate the original contents of a message.
- Message encryption is applied after compression to strengthen cryptographic security. Because the compressed message has less redundancy than the original message, cryptanalysis is more.

4. E-mail Compatibility: When PGP is used at least part of the block to be transmitted is encrypted. If only the signature service is used, then the message digest is encrypted. If confidentiality service is used then the message plus signature must be encrypted. Many e-mail systems only permit ASCII text. To accommodate this restriction, PGP provides a service of converting the raw 8-bit stream to a stream of printable ASCII characters.
5. Segmentation and Reassembly: E-mail facilities are often restricted to a maximum length. To accommodate this restriction, PGP automatically subdivides a message that is too large into segments that are small enough to send via e-mail.



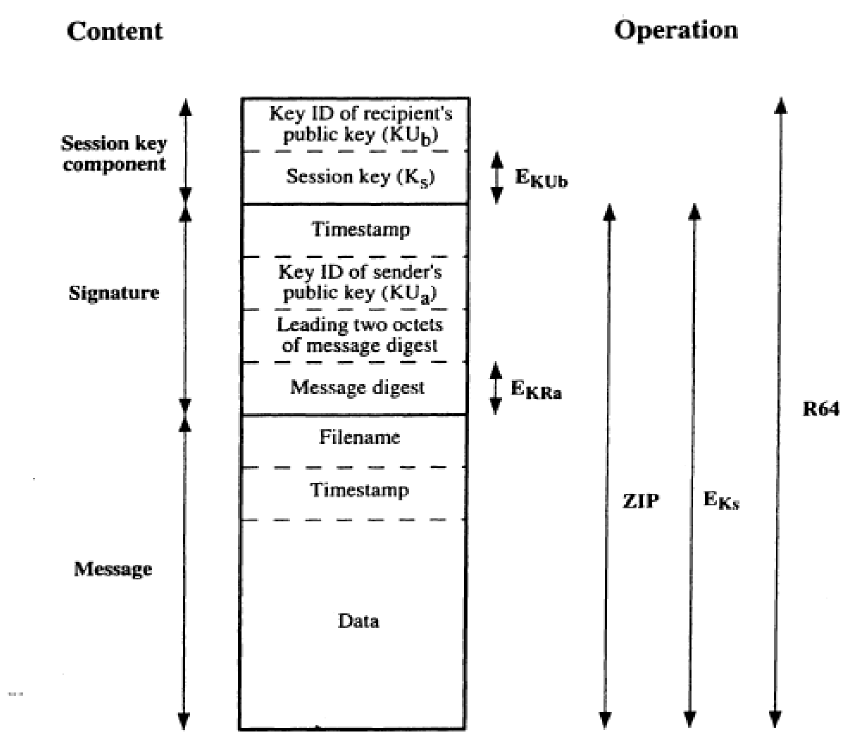
PGP Cryptographic Functions.

PGP Message Format:

A message consists of three components: a message component, signature component, session key component.

1. Message component includes the actual data to be stored or transmitted like file name, and a time stamp that specifies the time of creation.
2. Signature component includes the following components:

- Timestamp: The time at which the signature was made.
- Message Digest: The 160 SHA-1 message digest (hash code) encrypted with sender's private key.
- Leading two octets of message digest: To enable the recipient to determine if the correct public key was used to decrypt the message digest for authentication, by comparing this plain text copy of the first two octets with the first two octets of the message digest.
- Key Id of the sender's public key: Identifies the public key that should be used to decrypt the message digest and hence, identifies the private key that was used to encrypt the message digest. The message component and optional signature component may be compressed using ZIP and may be encrypted using a session key.



3. The session key component: It includes the session key and the identifier of the recipient's public key that was used by the sender to encrypt the session key.

All the three components are encoded with Radix 64 conversion.

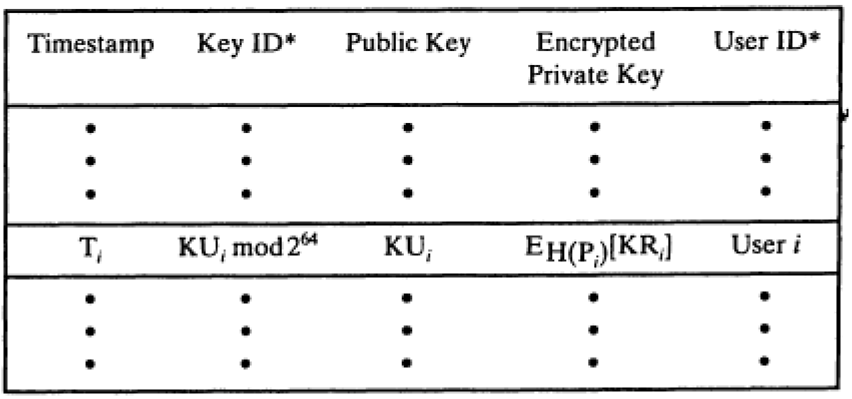
PGP Key Rings:

PGP maintains a pair of data structures at each node, one to store the public and private key pairs owned by that node and one to store the public keys of other users known at this node. These data structures are known as private key ring and public key ring respectively.

Private Key Ring: Each row represents one of the public/private key pairs owned by this user. Each row contains the following entries:

- Timestamp: The date/time when this key pair was generated
- Key Id: The least significant 64 bits of public key for this entry
- Public Key: The public key of user
- Private Key: The encrypted version of private key
- User Id: User's e-mail address.

Private-Key Ring

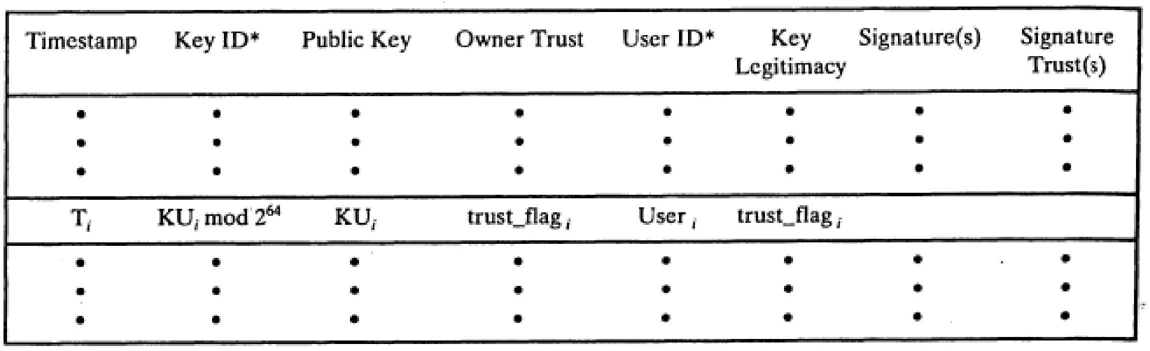


Timestamp	Key ID*	Public Key	Encrypted Private Key	User ID*
⋮	⋮	⋮	⋮	⋮
T_i	$KU_i \bmod 2^{64}$	KU_i	$E_{H(P_i)}[KR_i]$	User i
⋮	⋮	⋮	⋮	⋮

The private key ring can be indexed by either user id or key id.

Public Key Ring: This data structure is used to store public keys of other users that are known to this user.

Public-Key Ring



Timestamp	Key ID*	Public Key	Owner Trust	User ID*	Key Legitimacy	Signature(s)	Signature Trust(s)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
T_i	KU_i mod 2^{64}	KU_i	trust_flag$_i$	User $_i$	trust_flag$_i$		
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

* = field used to index table

Timestamp: The date/time when this entry was generated

Key Id: The least significant 64 bits of the public key for this entry

Public key: The public key of this entry

User Id: The owner of this key. Multiple user ids may be associated with a single public key.

Owner Trust: Represents Trust Flag value of owner.

Key Legitimacy: Key Trust Flag value

Signature: signature of the respective user

Signature Trust: signature of certification authority

PGP Message Generation:

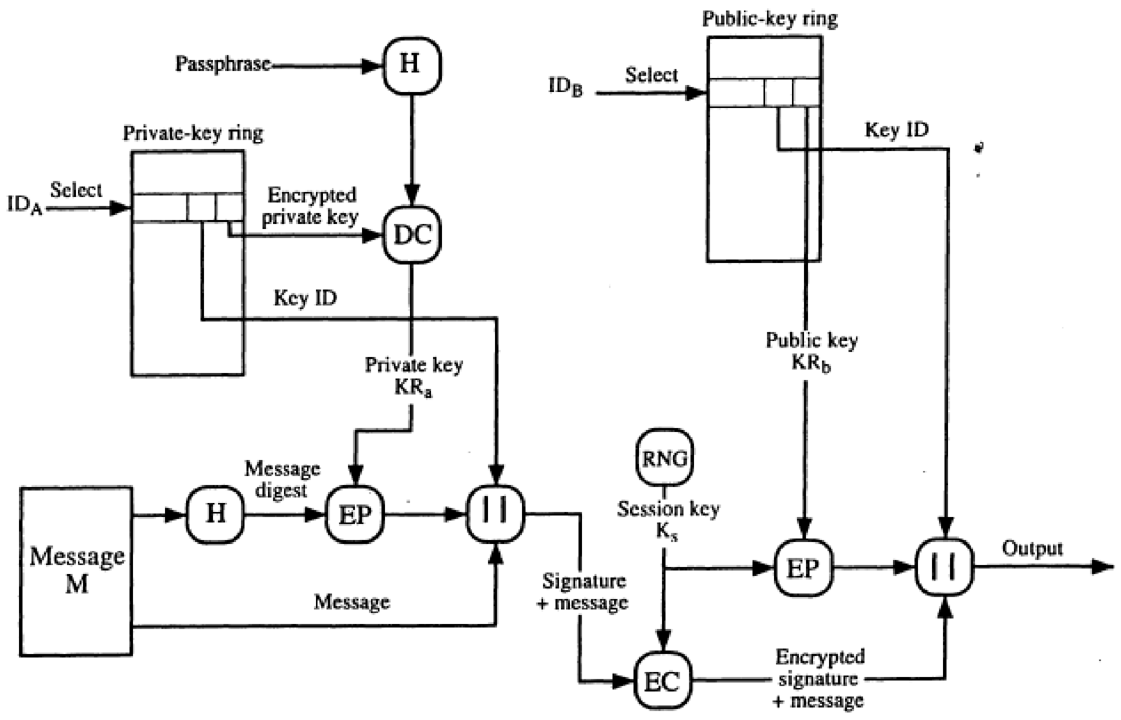
1. Sending the message

- PGP retrieves the sender's private key from the private key ring using the key id

- PGP prompts the user for the password to recover the encrypted private key which is stored in private key ring
- Then the signature is created with the retrieved private key.

2. Encrypting the message

- PGP generates a session key and encrypts the message
- PGP retrieves the recipient's public key from the public key ring using his/her key id.
- The session key component of the message is encrypted with the receiver's public key.



Now the sender forwards encrypted signature plus message and encrypted session key to the receiver.

PGP Message Reception:

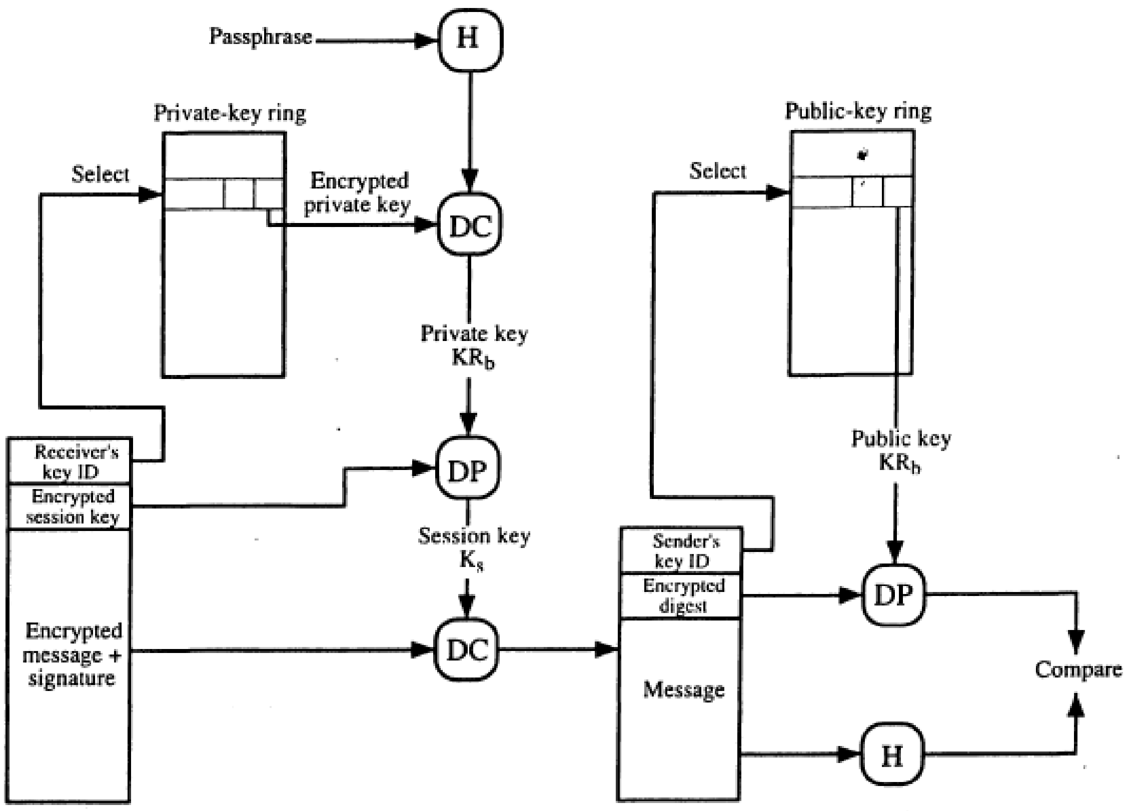
1. Decrypting the Message

- PGP retrieves the receiver's private key from the private key ring using key id field in the session key component of the message.
- PGP prompts the user for password to decrypt the encrypted private key which is stored in the private key ring.
- PGP recovers the session key by decrypting this using his private key.

2. Authenticating the Message

- PGP captures the sender's public key from the public key ring, using the key id field in the signature component of the message.

- Using the recovered public key receiver decrypts the message digest.
- PGP computes the message digest on the received message and compares this computed message digest with the decrypted message digest. If both are equal, then receiver accepts the message.

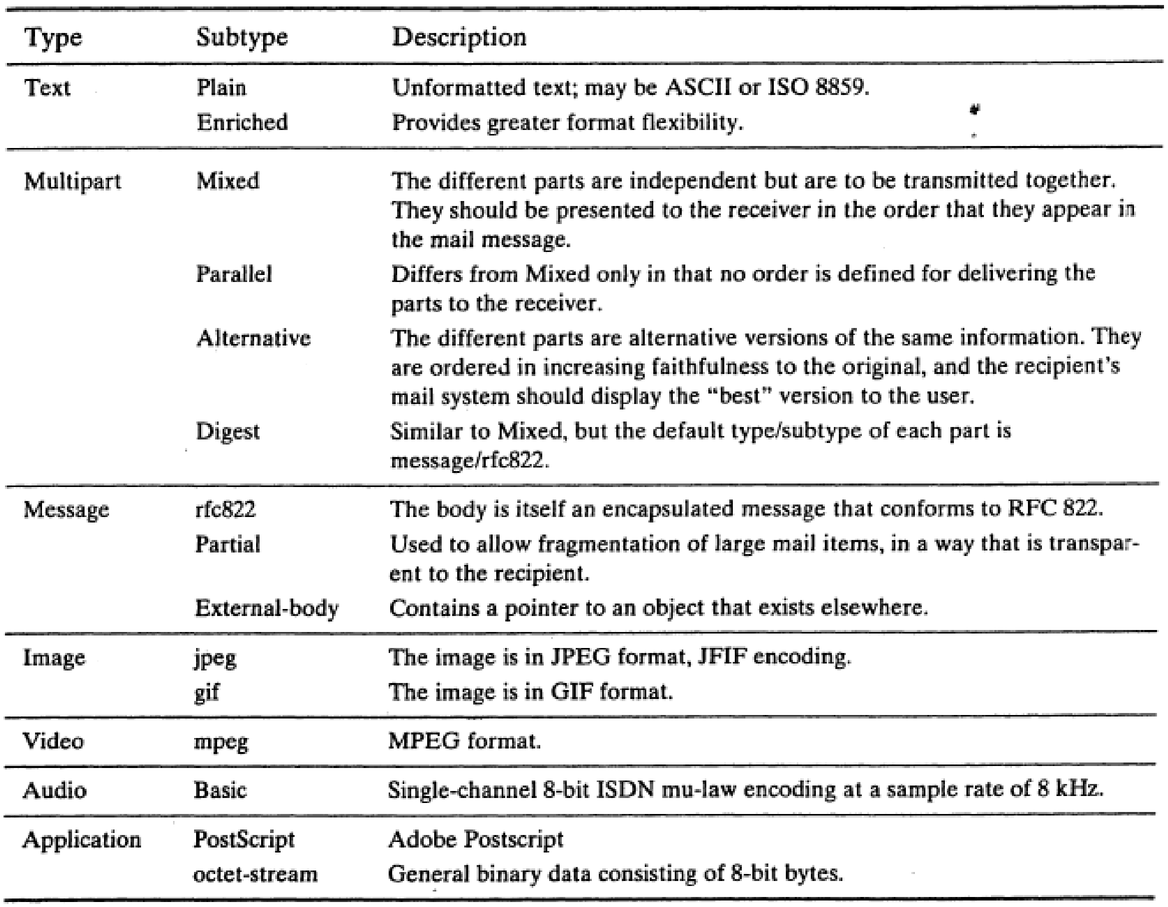


S/MIME: Secure Multipurpose Internet Mail Extension is a security enhancement to the MIME internet e-mail format standard based on RSA data security. MIME is an extension to the RFC 822 to address the limitations of the use of SMTP. Following are the limitations of SMTP:

- SMTP cannot transfer exe, and binary files
- SMTP cannot transfer textual data that includes national language characters
- SMTP servers rejects e-mail messages over a certain size
- SMTP gateways do not have compatibility between ASCII and EBCDIC formats
- SMTP gateways cannot handle non textual data

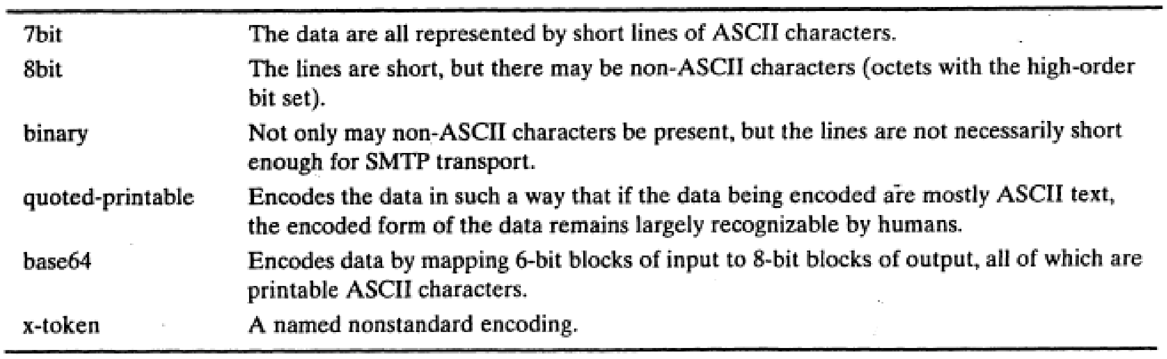
MIME consists of the following five headers:

- MIME version: Represents the version of MIME
- Content-Type: Describes the type of data contained in the body.



Type	Subtype	Description
Text	Plain	Unformatted text; may be ASCII or ISO 8859.
	Enriched	Provides greater format flexibility.
Multipart	Mixed	The different parts are independent but are to be transmitted together. They should be presented to the receiver in the order that they appear in the mail message.
	Parallel	Differs from Mixed only in that no order is defined for delivering the parts to the receiver.
	Alternative	The different parts are alternative versions of the same information. They are ordered in increasing faithfulness to the original, and the recipient's mail system should display the "best" version to the user.
	Digest	Similar to Mixed, but the default type/subtype of each part is message/rfc822.
Message	rfc822	The body is itself an encapsulated message that conforms to RFC 822.
	Partial	Used to allow fragmentation of large mail items, in a way that is transparent to the recipient.
	External-body	Contains a pointer to an object that exists elsewhere.
Image	jpeg	The image is in JPEG format, JFIF encoding.
	gif	The image is in GIF format.
Video	mpeg	MPEG format.
Audio	Basic	Single-channel 8-bit ISDN mu-law encoding at a sample rate of 8 kHz.
Application	PostScript	Adobe Postscript
	octet-stream	General binary data consisting of 8-bit bytes.

- Content-Transfer-Encoding: Indicates the type of transformation that has been used to represent the body of the message.



7bit	The data are all represented by short lines of ASCII characters.
8bit	The lines are short, but there may be non-ASCII characters (octets with the high-order bit set).
binary	Not only may non-ASCII characters be present, but the lines are not necessarily short enough for SMTP transport.
quoted-printable	Encodes the data in such a way that if the data being encoded are mostly ASCII text, the encoded form of the data remains largely recognizable by humans.
base64	Encodes data by mapping 6-bit blocks of input to 8-bit blocks of output, all of which are printable ASCII characters.
x-token	A named nonstandard encoding.

- Content Id: used to identify MIME entities uniquely
- Content-Description: A text description of the object with the body. This is useful when the object is not readable(audio)

S/MIME Functionality: It is very similar to PGP. Both offer the ability to sign and/or encrypt the messages. S/MIME provides the following functions:

- **Enveloped Data:** This consists of encrypted content of any type and encrypted content encryption keys for one or more recipients. The steps for developing enveloped data are:

a. Generates a pseudorandom session key for a particular symmetric key encryption
b. For each recipient, encrypt the session key with the recipient's public key
c. For each recipient, prepare a block such as recipient's info that contains the sender's public key certificate and the encrypted session key.
d. Encrypt the message content with the session key

- **Signed Data:** A digital signature is formed by taking the message digest of the content to be signed and then encrypting that with the private key of the signer. The content plus signature are then encoded using base 64 encoding. A signed data message can only be viewed by a recipient with S/MIME capability.
  i) Compute the message digest, or hash function of the content to be signed.
  ii) Encrypt the message digest with the signer's private key.
  iii) Prepare a block known as signerInfo that contains the signer's public key certificate and the encrypted message digest
- **Clear-Signed Data**: In signed data format, a digital signature of the content is formed. However in this case, only digital signature is encoded using base 64. So recipients without S/MIME capability can view the message contents but cannot verify the signature.
- **Signature and Enveloped Data**: Signed-only and encrypted-only entities may be nested, so that encrypted data may be signed and signed data or clear-signed data may be encrypted.

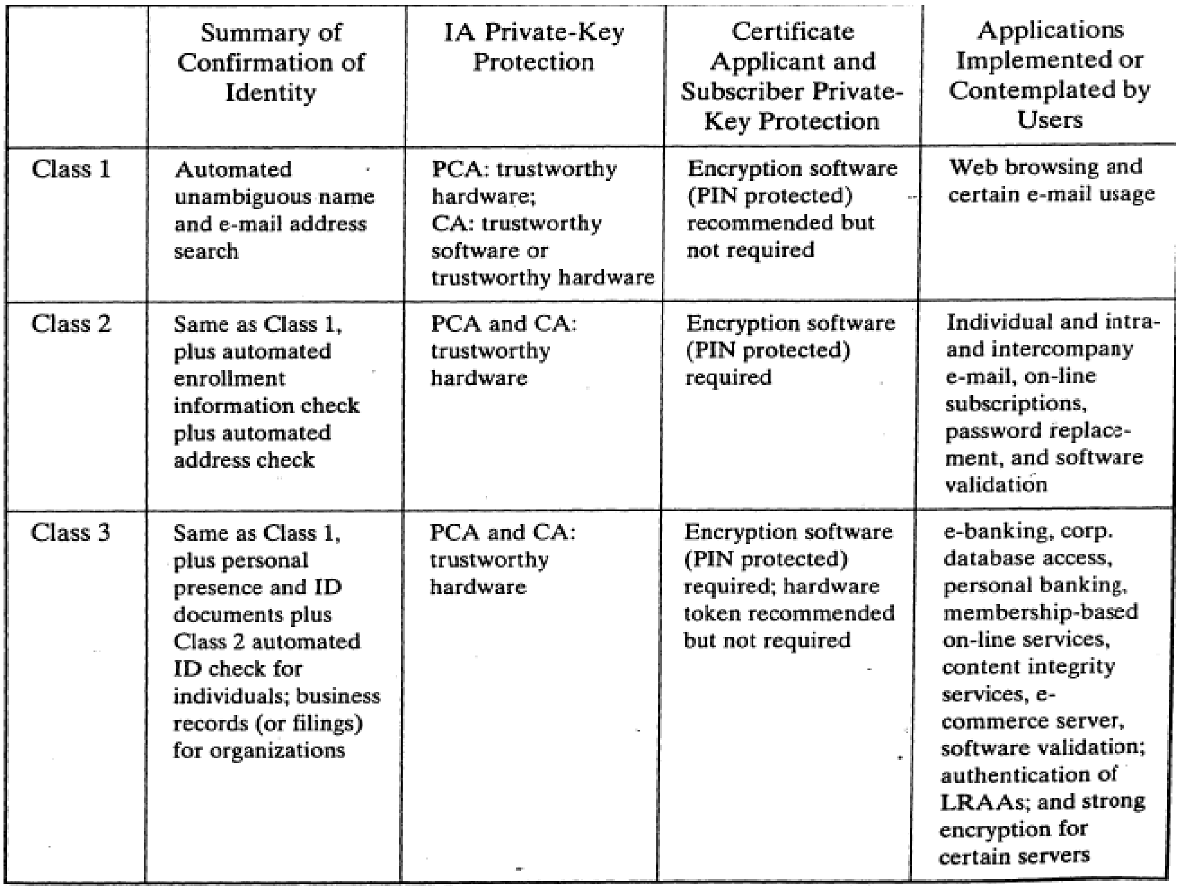
S/MIME Certificate Processing: S/MIME uses public key certificates that conform to version 3 of X.509. the key management scheme used by S/MIME is hybrid between X.509 hierarchy and PGP's web of trust. Certificates are used to verify incoming signatures and to encrypt outgoing messages.

Verisign Certificates: There are several companies that provide certification authority services. For example Nortel has designed an enterprise CA to provide S/MIME support within an organization. There are a number of Internet-based CA's including Verisign, GTE and the U.S Postal service. Verisign provides a CA service that is intended to be compatible with S/MIME. The information contained in a Digital ID depends on the type of Digital ID and its use. Each Digital ID contains the following:

- Owner's public key, Owner's name or alias
- Expiration date of the digital ID, Serial number of the Digital ID
- Name of the CA that issued the Digital ID
- Digital signature of the CA
- It also contains user supplied information like address
- E-mail address

- Basic registration information(country, zip code, age, and gender

Verisign provides three levels or classes of security for public key certificates. Here is a list of classes.

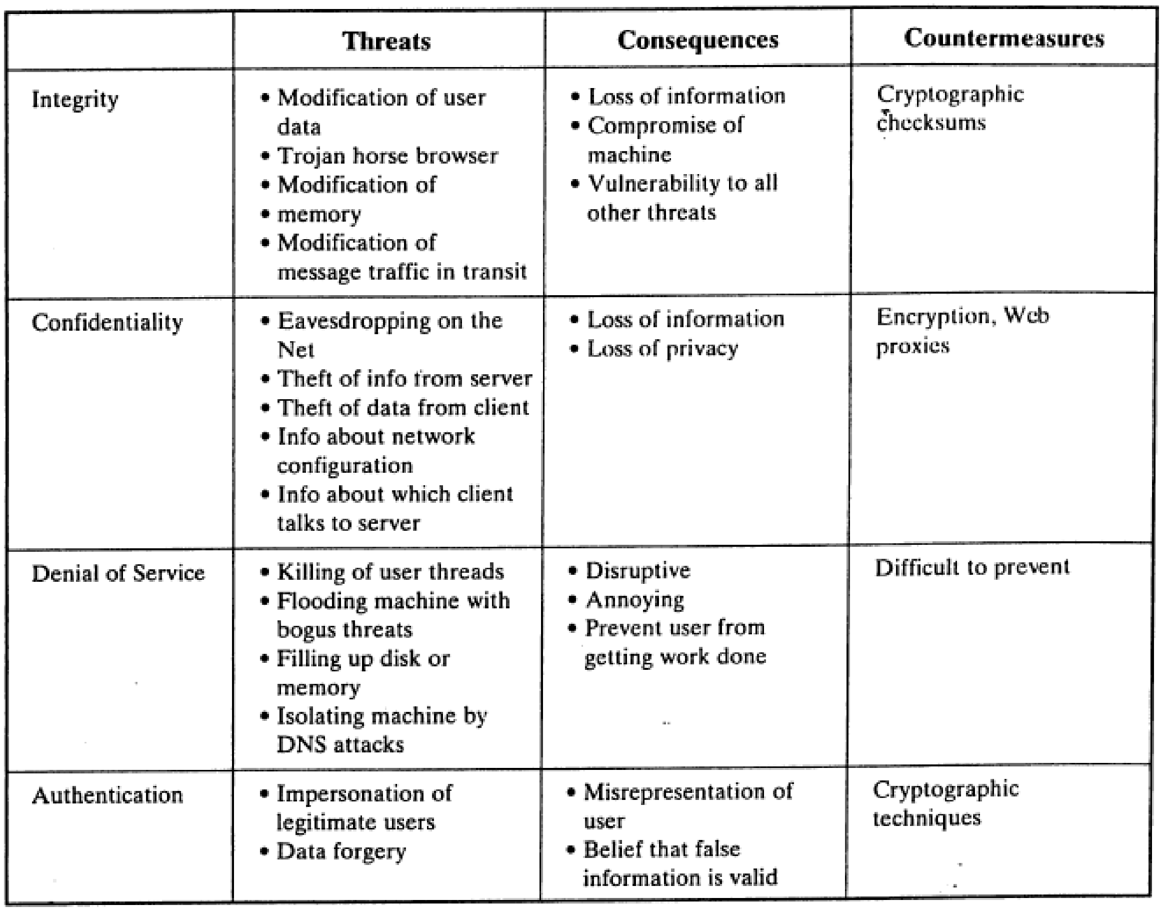


	Summary of Confirmation of Identity	IA Private-Key Protection	Certificate Applicant and Subscriber Private-Key Protection	Applications Implemented or Contemplated by Users
Class 1	Automated unambiguous name and e-mail address search	PCA: trustworthy hardware; CA: trustworthy software or trustworthy hardware	Encryption software (PIN protected) recommended but not required	Web browsing and certain e-mail usage
Class 2	Same as Class 1, plus automated enrollment information check plus automated address check	PCA and CA: trustworthy hardware	Encryption software (PIN protected) required	Individual and intra- and intercompany e-mail, on-line subscriptions, password replacement, and software validation
Class 3	Same as Class 1, plus personal presence and ID documents plus Class 2 automated ID check for individuals; business records (or filings) for organizations	PCA and CA: trustworthy hardware	Encryption software (PIN protected) required; hardware token recommended but not required	e-banking, corp. database access, personal banking, membership-based on-line services, content integrity services, e-commerce server, software validation; authentication of LRAAs; and strong encryption for certain servers

WEB SECURITY

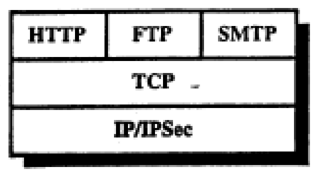
The World Wide Web is fundamentally a client/server application running over the internet and TCP/IP intranets.

Web Security Threats: Web security threats are located at web server, web browser and network traffic between browser and server. These threats grouped into two categories passive and active threats. Following is the list of web threats:

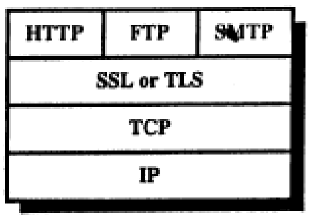


	Threats	Consequences	Countermeasures
Integrity	• Modification of user data • Trojan horse browser • Modification of • memory • Modification of message traffic in transit	• Loss of information • Compromise of machine • Vulnerability to all other threats	Cryptographic checksums
Confidentiality	• Eavesdropping 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

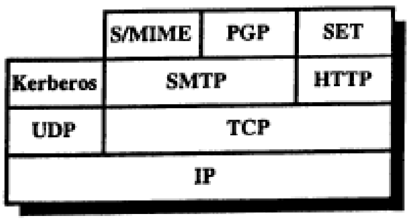
Web Traffic Security Approaches: Several approaches are possible to provide web security. One way to provide web security is to use IP security. The advantage of using IP security is that it is transparent to end users and applications and provides security. It includes filtering capability so that only selected traffic need to be secured.



(a) Network Level



(b) Transport Level



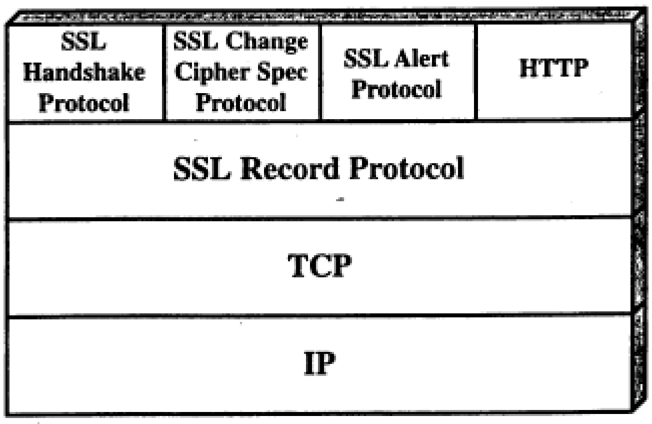
(c) Application Level

Another solution is to implement security at the top of TCP. SSL or TLS can be used to secure the web transactions. Application specific security services are embedded within the particular application. The advantage of this approach is that the service can be tailored to the specific needs of a given application. Ex: SET.

Secure Socket Layer and Transport Layer Security:

SSL was originated by Netscape. Version 3 of the protocol was designed with public review and input from industry. The first version of TLS can be viewed essentially an SSL version 3 .

SSL Architecture: SSL is designed to make use of TCP to provide a reliable end-to-end secure service. SSL is not a single protocol but it is a two layers of protocols. The SSL record protocol provides basic security services to various higher layer protocols. HTTP provides the transfer service for web client/server interaction and operates 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 SSL session and the SSL connection.

Connection: A connection is a transport that provides a suitable type of service. For SL, 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. Session define a set of cryptographic security parameters which are shared with multiple connections. Security sessions are used to avoid the expensive negotiation of new security parameters for each connection.

There are number of states associated with each session. Once a session is established, there is a current operating state for both read and write. A session state is defined by the following parameters:

- Session Identifier: A sequence selected by the server to identify an active or resumable session state
- Peer certificate: An X.509 certificate of the peer.
- Compression method: The algorithm used to compress prior to encryption
- Cipher spec: Specifies the bulk data encryption algorithm used for MAC calculation.
- Master secret: The forty-eight byte secret shared between the client and server
- Is resumable: Indicates whether the session is used to initiate new connections.

A connection state is defined by the following parameters.

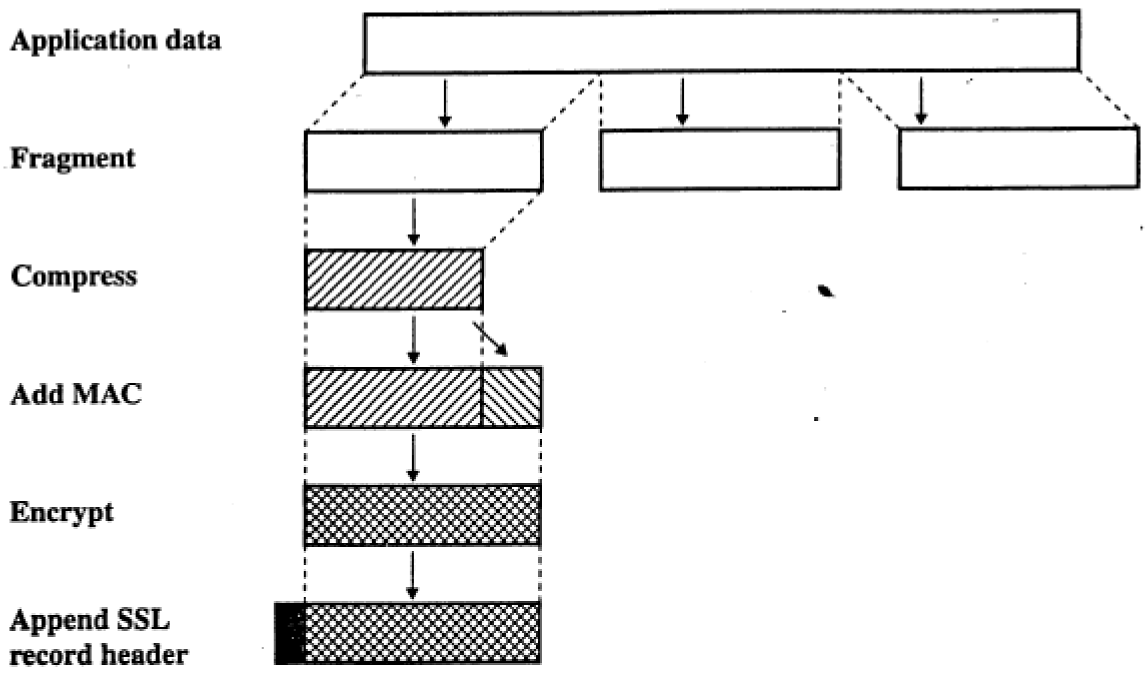
- Server and client random: byte sequences that are chosen by client and server for each connection
- Server write MAC secret: The secret key used in MAC operation on data sent by the server
- Client write MAC secret: The secret key used in MCA operation on data sent by the client.
- Server write key: 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 vector: when a block cipher uses CBC mode, an IV is used for each key. This is first initialized by handshake protocol.
- Sequence numbers: each party maintains separately sequence numbers for transmitted and received messages.

SSL Record Protocol:

The SSL record protocol provides two services for SSL connection.

- Confidentiality: Handshake protocol defines a shared secret key that is sued for conventional encryption of SSL payloads.
- Message Integrity: The Handshake protocol also defines a shared secret key that is used to form a MAC.

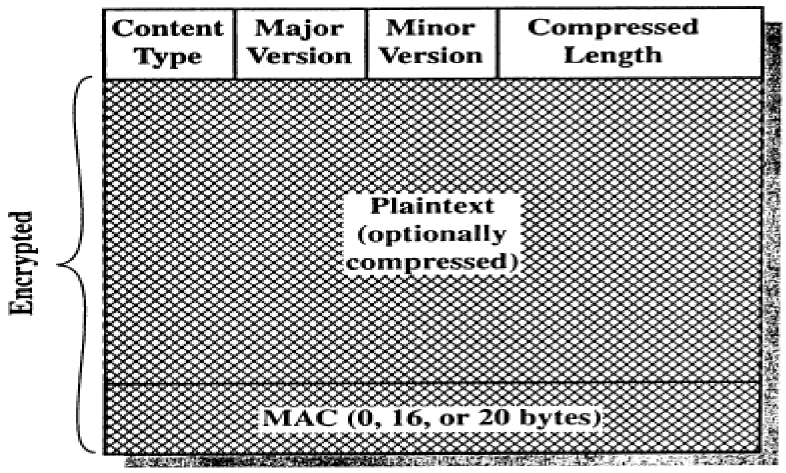
The Record protocol takes an application message to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies a MAC(using HMAC), encrypts (using IDEA, DES, 3DES, fortezza(used in smart card encryption)), adds a header, and transmits the resulting unit in a TCP segment. Received data are then decrypted, verified and decompressed and reassembled and then delivered to the higher level users.



SSL Record Protocol Operation.

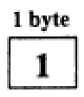
The final step of SSL record protocol processing is to prepend a header which contains the following fields:

- Content type(8 bits): The higher layer protocol used to process the enclosed fragment
- Major version(8 bits): Indicates a major version of SSL in use.it is 3.
- Minor version(8 bits): Indicates minor version of SSL in use. It is 0.
- Compressed length(16 bits): The length in bytes of the plain text fragment (or compressed fragment if compression is used) .

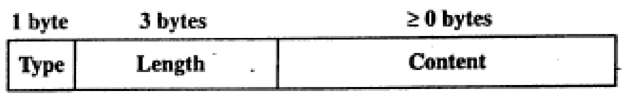


SSL Change Cipher Spec Protocol:

It is one of the three SSL-specific protocols that use the SSL record protocol, and it is simplest. It consists of a single message, which is a single byte with the value 1. The sole purpose fo this message is to cause the pending state to be copied into the current state, which updates the cipher suite to be used on this connection.



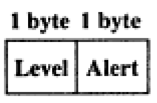
(a) Change Cipher Spec Protocol



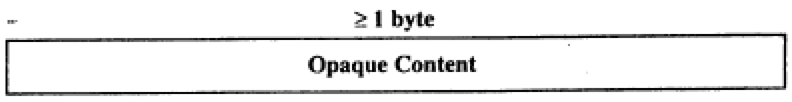
(c) Handshake Protocol

SSL Alert Protocol:

The alert protocol is used to convey SSL related alerts to the peer entity. Each message in this protocol consists of two bytes. The first byte takes the value warning (1) or fatal(2) to convey the security of message. If the level is fatal, SSL immediately terminates the connection. Other connections on the same session may continue, but no new connections on this session may be established. The second byte contains a code that indicates the specific alert. Here is a list of Fatal alerts:



(b) Alert Protocol



(d) Other Upper-Layer Protocol (e.g., HTTP)

- Unexpected_message: An inappropriate message was received

- Bad_record_mac: An incorrect mac was received.
- Decompression_failure: The received decompression function is improper.
- Handshake_failure: sender was unable to negotiate an acceptable set of security parameters.
- Illegal_parameter: A field in a handshake message was out of range

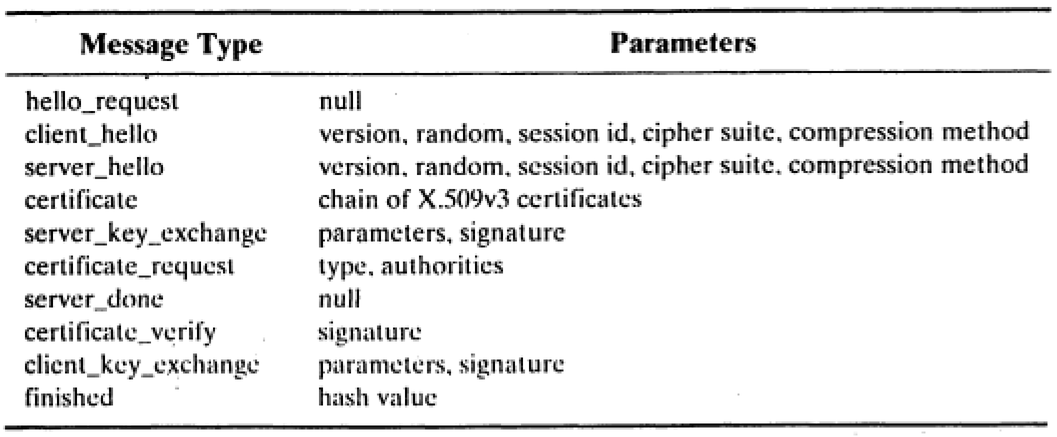
Rest of the alerts are:

- Close_notify: notifies the recipient that the sender will not send any more messages on this connection. Each party is required to send close_notify alert before closing the write side of a connection.
- No_certificate: no appropriate certificate is available
- Bad_certificate: A received certificate was corrupt.
- Unsupported_certificate: certificate is not supported
- Certificate_revoked: certificate is already revoked by its signer
- Certificate_expired: Certificate has expired.
- Certificate_unknown: Certificate CA is not known

SSL Handshake Protocol: The most complex part of SSL is the Handshake protocol. It allows the server and client to authenticate each other and to negotiate encryption and MAC algorithm and cryptographic keys to be used to protect data sent on the 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. Each message has three fields:

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

SSL Handshake protocol Message Types:



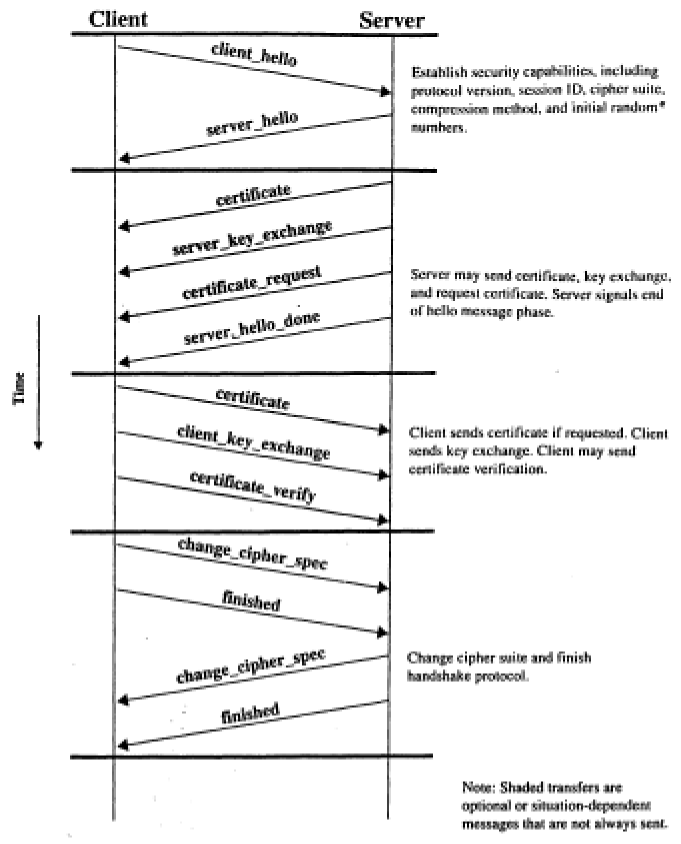
Message Type	Parameters
hello_request	null
client_hello	version, random, session id, cipher suite, compression method
server_hello	version, random, session id, cipher suite, compression method
certificate	chain of X.509v3 certificates
server_key_exchange	parameters, signature
certificate_request	type, authorities
server_done	null
certificate_verify	signature
client_key_exchange	parameters, signature
finished	hash value

Phase 1: Establishing security capabilities:

This phase is used to initiate 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: Gives details of SSL version
- Random: A client generated pseudorandom number and timestamp
- Session ID: A variable length session identifier.
- CipherSuite: Contains a list that combination of cryptographic algorithms
- Compression method: List of compression methods that client supports.

After sending the client_hello message, the client waits for the server_hello message, which contains the same parameters as the client_hello message.


Client
Server
client_hello
server_hello
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
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.
Time
certificate
client_key_exchange
certificate_verify
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.
change_cipher_spec
finished
change_cipher_spec
finished
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, if it needs to be authenticated; the message contains one or a chain of X.509 certificates. Next server_key_exchange message will best transmitted by the server. Server request client to forward client's public key certificate. The final message in phase 2 is server_done message which is sent by the server to indicate the end of server hello and associated messages. After sending this message, server will wait for a response from client. This message has no parameters.

Phase 3 Client Authentication and Key Exchange:

Upon receiving server_done message, the client verify that the server provided a valid certificate if required 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, which must be sent in this phase. The content of the message depends on the type of key exchange.

Finally in this pahse, the client sends a certificate_verify message to provide explicit verification of a client certificate.

Phase 4 Finish: IT 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. This message is not considered part of the Handshake protocol but is sent using the change cipher spec protocol. 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):

TLS is very similar to SSL3. The TLS record format is the same as that of the SSL. Record format and the fields in the header have the same meanings. There are two differences between the SSLv3 and TLS MAC schemes: the actual algorithm and the scope of the MAC calculation. TLS makes use of the HMAC algorithm. TLS supports all of the alert codes defined in SSLv3 with the exception of no certificate. A number of additional codes are define in TLS:

- Decryption failed : cipher text decrypted in an invalid way
- Record_overflow: TLS record was receive with a payload
- Unknown_CA: CA is not trusted and is invalid
- Access_denied: A valid certificate was received, but when access control is applied, the sender decided not to proceed with the negotiation.
- Decode_error: A message could not be decoded because a field was out of its specified range or length of the message was incorrect
- Export_restriction: A negotiation not in compliance with export restrictions on key length was detected.

- Protocol version: The protocol version the client attempted to negotiate is recognized but not supported
- Insufficient_security: returned instead of handshake_failure when a negotiation has failed specifically because the server requires ciphers more secure than those supported by the client.
- Internal_error: an internal error unrelated to the peer or correctness of the protocol makes it impossible to continue.

HTTPS: HTTPS (Http over SSL) refers to the combination of HTTP and SSL to implement secure communication between a web browser and a web server. The HTTPS capability is built into all modern browsers. The principal difference seen by a user of a web browser is that URL addresses begin with https:// rather than http://. A normal http uses a port 80. If https is specified, port 443 is sued which invokes SSL. When HTTPS is sued, the following elements of the communication are encrypted:

- URL of the requested document
- Contents of the document
- Contents of browser forms
- Cookies sent from browser to server and from server to browser
- Contents of HTTP header

Connection Initiation: For HTTPS, the agent acting as the HTTP client also acts as the TLS client. The client initiates a connection to the server on the appropriate port and then send the TLS clienthello to begin the TLS handshake. When the TLS handshake has finished, the client may then initiate the first HTTP request. All HTTP data is to be sent as TLS application data.

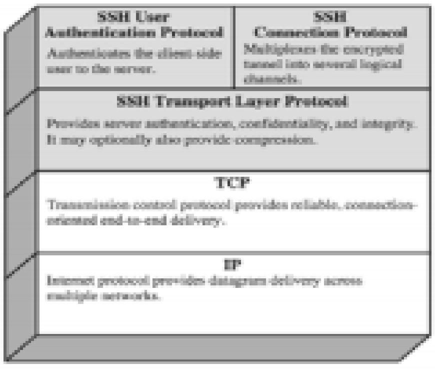
Connection Closure: As HTTP client or server can indicate the closing of a connection by including the following line in an HTTP record: connection: close. This indicates that the connection will be closed after this record is delivered.

The closure of an HTTPS connection requires that TLS close the connection with the peer TLS entity on the remote side, which will involve closing the underlying TCP connection.

Secure Shell(SSH)

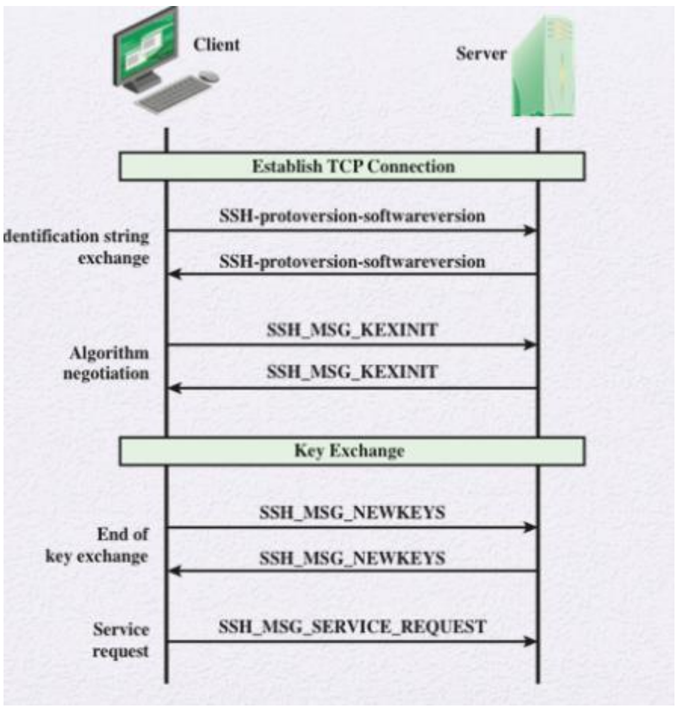
SSH is a protocol for secure network communication designed to be relatively simple and inexpensive to implement. The initial version SSH1 was focused on providing a secure remote login facility to replace TELNET and RLOGIN schemes. SSH also provides a more general client/server capability and can be used for such network functions as file transfer and e-mail. A new version, SSH2 fixes a number of security flaws in the original scheme.

SSH Protocol Stack

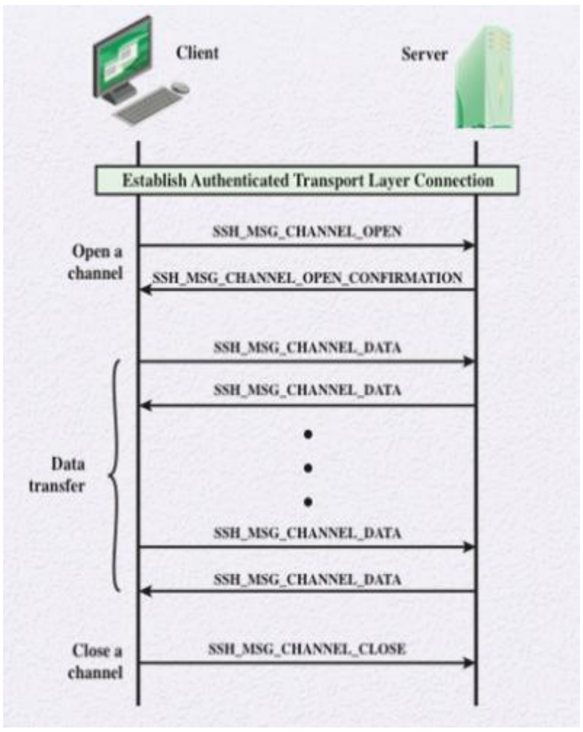

SSH User Authentication Protocol
Authenticates the client-side user to the server.
SSH Connection Protocol
Multiplexes the encrypted tunnel into several logical channels.
SSH Transport Layer Protocol
Provides server authentication, confidentiality, and integrity. It may optionally also provide compression.
TCP
Transmission control protocol provides reliable, connection-oriented end-to-end delivery.
IP
Internet protocol provides datagram delivery across multiple networks.

SSH client and server applications are widely available for most operating systems. SSH is organized as three protocols that typically run on top of TCP.

- SSH Transport Layer protocol: provides server authentication, data confidentiality and data integrity with forward secrecy. Transport layer may optionally provide compression.



- User Authentication Protocol: Authenticates the user to the server
- Connection Protocol: Multiplexes multiple logical communications channels over a single, underlying SSH connection.


Client
Server
Establish Authenticated Transport Layer Connection
Open a channel
SSH_MSG_CHANNEL_OPEN
SSH_MSG_CHANNEL_OPEN_CONFIRMATION
Data transfer
SSH_MSG_CHANNEL_DATA
SSH_MSG_CHANNEL_DATA
SSH_MSG_CHANNEL_DATA
SSH_MSG_CHANNEL_DATA
Close a channel
SSH_MSG_CHANNEL_CLOSE