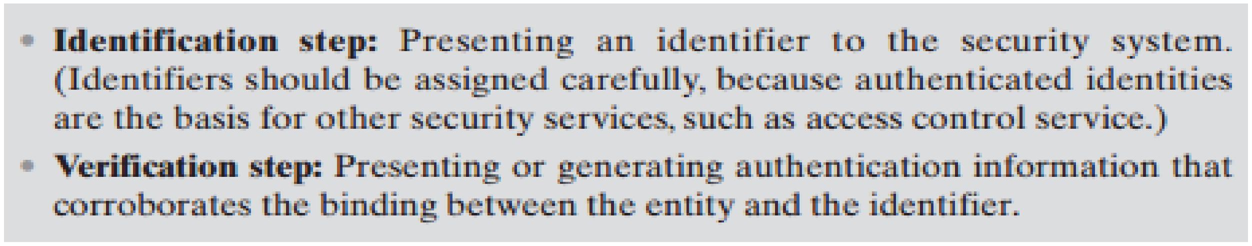
UNIT-V

REMOTE USER-AUTHENTICATION PRINCIPLES

In most computer security contexts, user authentication is the fundamental building block and the primary line of defense. User authentication is the basis for most types of access control and for user accountability

An Authentication process consists of two steps



- **Identification step:** Presenting an identifier to the security system. (Identifiers should be assigned carefully, because authenticated identities are the basis for other security services, such as access control service.)
- **Verification step:** Presenting or generating authentication information that corroborates the binding between the entity and the identifier.

Identification is the means by which a user provides a claimed identity to the system; user authentication is the means of establishing the validity of the claim .Note that user authentication is distinct from message authentication.

There are four general means of authenticating a user's identity, which can be used alone or in combination:

• **Something the individual knows:** Examples include a password, a personal identification number (PIN), or answers to a prearranged set of questions.

• **Something the individual possesses:** Examples include cryptographic keys, electronic keycards, smart cards, and physical keys. This type of authenticator is referred to as a token.

• **Something the individual is (static biometrics):** Examples include recognition by fingerprint, retina, and face.

• **Something the individual does (dynamic biometrics):** Examples include recognition by voice pattern, handwriting characteristics, and typing rhythm.

KERBEROS:

✓ **Introduction:**

Kerberos is a computer network authentication protocol which works on the basis of "tickets" to allow nodes communication over a non secure network.

(Or) It is a secure method (service) to authenticate a request for a service in a computer network.

It was developed in “Athena Project” at MIT.
It provide authentication by using Secret-Key cryptography.

- ✓ **Use of Kerberos:**
  Kerberos is used for decreasing the burden for server, means; Kerberos will takes responsibility of authentication.
  It is designed for providing for strong authentication for client/server applications by using secret-key.

- ✓ **Versions:**
  Kerberos Version4
  Kerberos Version5

Characteristics of KERBEROS:

- ✓ It is secure: it never sends a password unless it is encrypted.
- ✓ Only a single login is required per session. Credentials defined at login are then passed between resources without the need for additional logins.
- ✓ The concept depends on a trusted third party – a Key Distribution Center (KDC). The KDC is aware of all systems in the network and is trusted by all of them.
- ✓ It performs mutual authentication, where a client proves its identity to a server and a server proves its identity to the client.

Requirements Kerberos:

- ✓ **Secure**: Kerberos should be strong enough that a potential opponent does not find it to be the weak link.

- ✓ **Reliable:** Kerberos should be highly reliable and should employ distributed server architecture with one system able to back up another.
- ✓ **Transparent:** Ideally, the user should not be aware that authentication is taking place beyond the requirement to enter a password.
- ✓ **Scalable:** The system should be capable of supporting large numbers of clients and servers.

Kerberos Version 4

Version 4 of Kerberos makes use of DES

A SIMPLE AUTHENTICATION DIALOGUE

In an unprotected network environment, any client can apply to any server for service. The obvious security risk is that of impersonation. An opponent can pretend to be another client and obtain unauthorized privileges on server machines. To counter this threat, servers must be able to confirm the identities of clients who request service. Each server can be required to undertake this task for each client/server interaction, but in an open environment, this places a substantial burden on each server.

An alternative is to use an Authentication server (AS) that knows the passwords of all users and stores these in a centralized database. In addition, the AS shares a unique secret key with each server. These keys have been distributed physically or in some other secure manner. Consider the following hypothetical dialogue:

(1) C $\rightarrow$ AS: $ID_C \| P_C \| ID_V$

(2) AS $\rightarrow$ C: $Ticket$

(3) C $\rightarrow$ V: $ID_C \| Ticket$

$Ticket = E(K_v, [ID_C \| AD_C \| ID_V])$

where

C	= client
AS	= authentication server
V	= server
ID_C	= identifier of user on C
ID_V	= identifier of V
P_C	= password of user on C
AD_C	= network address of C
K_v	= secret encryption key shared by AS and V

1. The user logs on to a workstation and requests access to server V. The client module C in the user's workstation requests the user's password and then sends a message to the AS that includes the user's ID, the server's ID, and the user's password.
2. The AS checks its database to see if the user has supplied the proper password for this user ID and whether this user is permitted access to server V. If both tests are passed, the AS accepts the user as authentic and must now convince the server that this user is authentic. To do so, the AS creates a **ticket** that contains the user's ID and network address and the server's ID. This ticket is encrypted using the secret key shared by the AS and this server. This ticket is then sent back to C.

3. With this ticket, C can now apply to V for service. C sends a message to V containing C's ID and the ticket. V decrypts the ticket and verifies that the user ID in the ticket is the same as the unencrypted user ID in the message.

Problems:

The first problem is that the earlier scenario involved a plaintext transmission of the password [message (1)]. An eavesdropper could capture the password and use any services accessible to the victim

An opponent could capture the ticket transmitted in message (2), then use the name ID_C *a*nd transmit a message of form (3) another workstation. The server would receive a valid ticket that matches the user ID and grant access to the user on that other workstation. To prevent this attack, the AS includes in the ticket the network address from which the original request came.

The second problem is we would like to minimize the number of times that a user has to enter a password. Suppose each ticket can be used only once. If user C logs on to a workstation in the morning and wishes to check his or her mail at a mail server, C must supply a password to get a ticket for the mail server. If C wishes to check the mail several times during the day, each attempt requires reentering the password. We can improve matters by saying that tickets are reusable. For a single logon session, the workstation can store the mail server ticket after it is received and use it on behalf of the user .

A MORE SECURE AUTHENTICATION DIALOGUE

The main problem in ***A SIMPLE AUTHENTICATION DIALOGUE, the user*** must enter password for every individual service.

Kerberos overcome this by using a new server, known as Ticket granting server (TGS).

Now in Kerberos we have two servers; AS and TGS.

Once per user logon session:

(1) $\text{C} \rightarrow \text{AS:} \quad ID_C \| ID_{tgs}$

(2) $\text{AS} \rightarrow \text{C:} \quad \text{E}(K_c, Ticket_{tgs})$

Once per type of service:

(3) $\text{C} \rightarrow \text{TGS:} \quad ID_C \| ID_V \| Ticket_{tgs}$

(4) $TGS \rightarrow \text{C:} \quad Ticket_v$

Once per service session:

(5) $\text{C} \rightarrow \text{V:} \quad ID_C \| Ticket_v$

$$Ticket_{tgs} = \text{E}(K_{tgs}, [ID_C \| AD_C \| ID_{tgs} \| TS_1 \| Lifetime_1])$$
$$Ticket_v = \text{E}(K_v, [ID_C \| AD_C \| ID_v \| TS_2 \| Lifetime_2])$$

The new service, TGS, issues tickets to users who have been authenticated to AS. Thus, the user first requests a ticket-granting ticket ($Ticket_{tgs}$) from the AS.

The client module in the user workstation saves this ticket. Each time the user requires access to a new service, the client applies to the TGS, using the ticket to authenticate itself. The TGS then grants a ticket for the particular service. The client saves each service-granting ticket and uses it to authenticate its user to a server each time a particular service is requested. Let us look at the details of this scheme:

1. The client requests a ticket-granting ticket on behalf of the user by sending its user's ID to the AS, together with the TGS ID, indicating a request to use the TGS service.
2. The AS responds with a ticket that is encrypted with a key that is derived from the user's password ( ), which is already stored at the AS. When this response arrives at the client, the client prompts the user for his or her password, generates the key, and attempts to decrypt the incoming message. If the correct password is supplied, the ticket is successfully recovered.
3. The client requests a service-granting ticket on behalf of the user. For this purpose, the client transmits a message to the TGS containing the user's ID, the ID of the desired service, and the ticket-granting ticket.
4. The TGS decrypts the incoming ticket using a key shared only by the AS and the TGS (K_{tgs}) and verifies the success of the decryption by the presence of its ID. It checks to make sure that the lifetime has not expired. Then it compares the user ID and network address with the incoming information to authenticate the user. If the user is permitted access to the server V, the TGS issues a ticket to grant access to the requested service.
5. The client requests access to a service on behalf of the user. For this purpose, the client transmits a message to the server containing the user's ID and the service-granting ticket. The server authenticates by using the contents of the ticket.

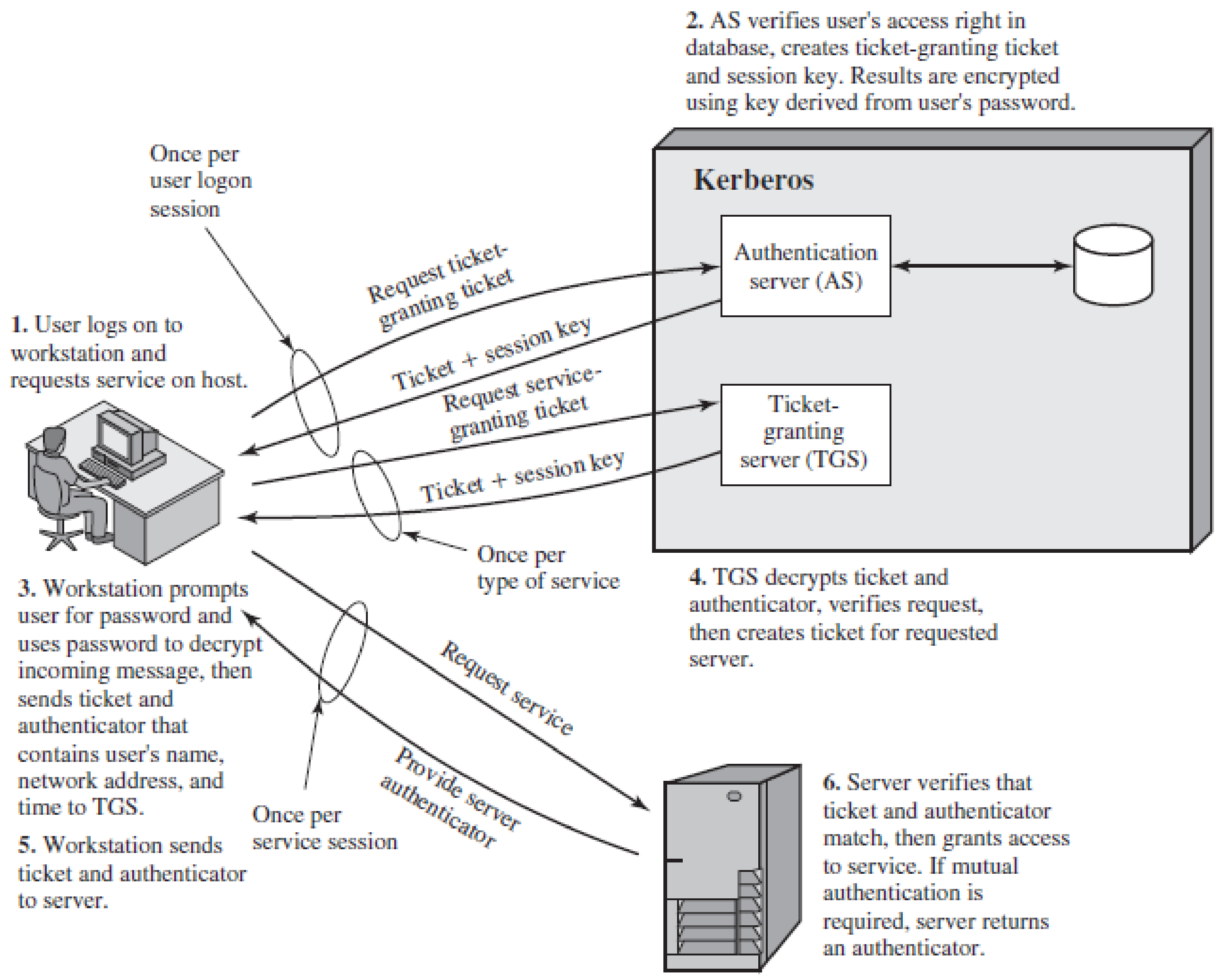
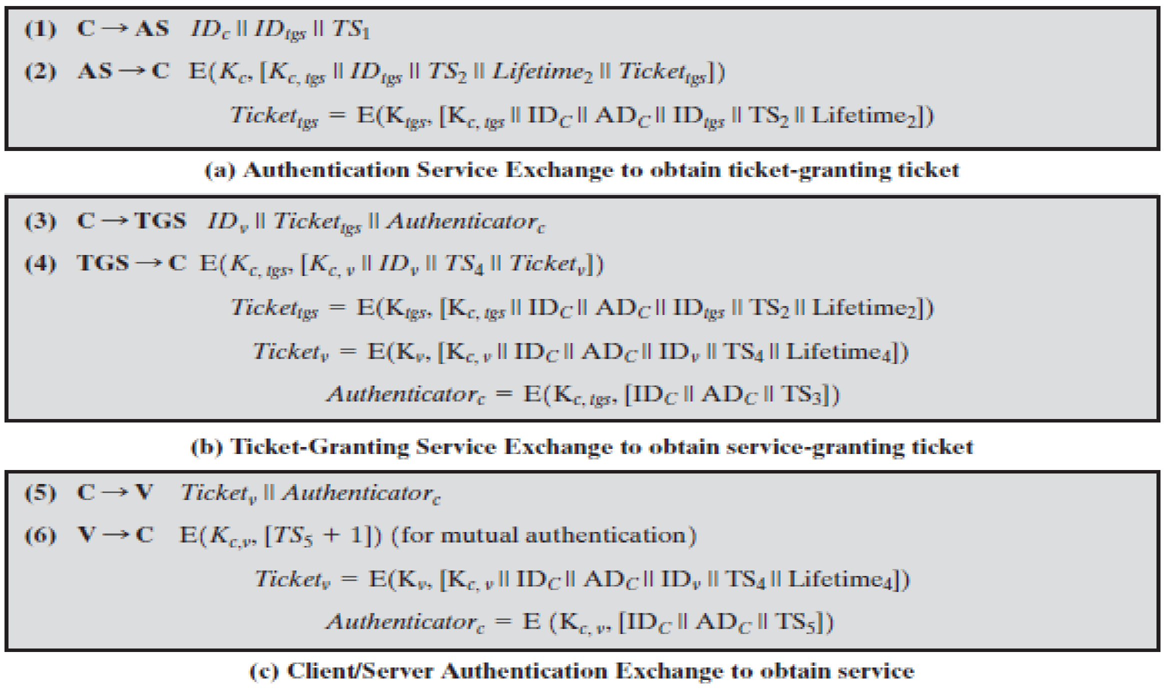

2. AS verifies user's access right in database, creates ticket-granting ticket and session key. Results are encrypted using key derived from user's password.
Once per user logon session
Kerberos
Authentication server (AS)
Request ticket-granting ticket
Ticket + session key
1. User logs on to workstation and requests service on host.
Request service-granting ticket
Ticket-granting server (TGS)
Ticket + session key
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
5. Workstation sends ticket and authenticator to server.
Once per service session
6. Server verifies that ticket and authenticator match, then grants access to service. If mutual authentication is required, server returns an authenticator.

Figure 15.1 Overview of Kerberos

Table 15.1 Summary of Kerberos Version 4 Message Exchanges



(1) $\mathbf{C \rightarrow AS}$ $ID_c \| ID_{tgs} \| TS_1$

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

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

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

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

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

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

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

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

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

(5) $\mathbf{C \rightarrow V}$ $Ticket_v \| Authenticator_c$

(6) $\mathbf{V \rightarrow C}$ $E(K_{c,v}, [TS_5 + 1])$ (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

WEB SECURITY

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.

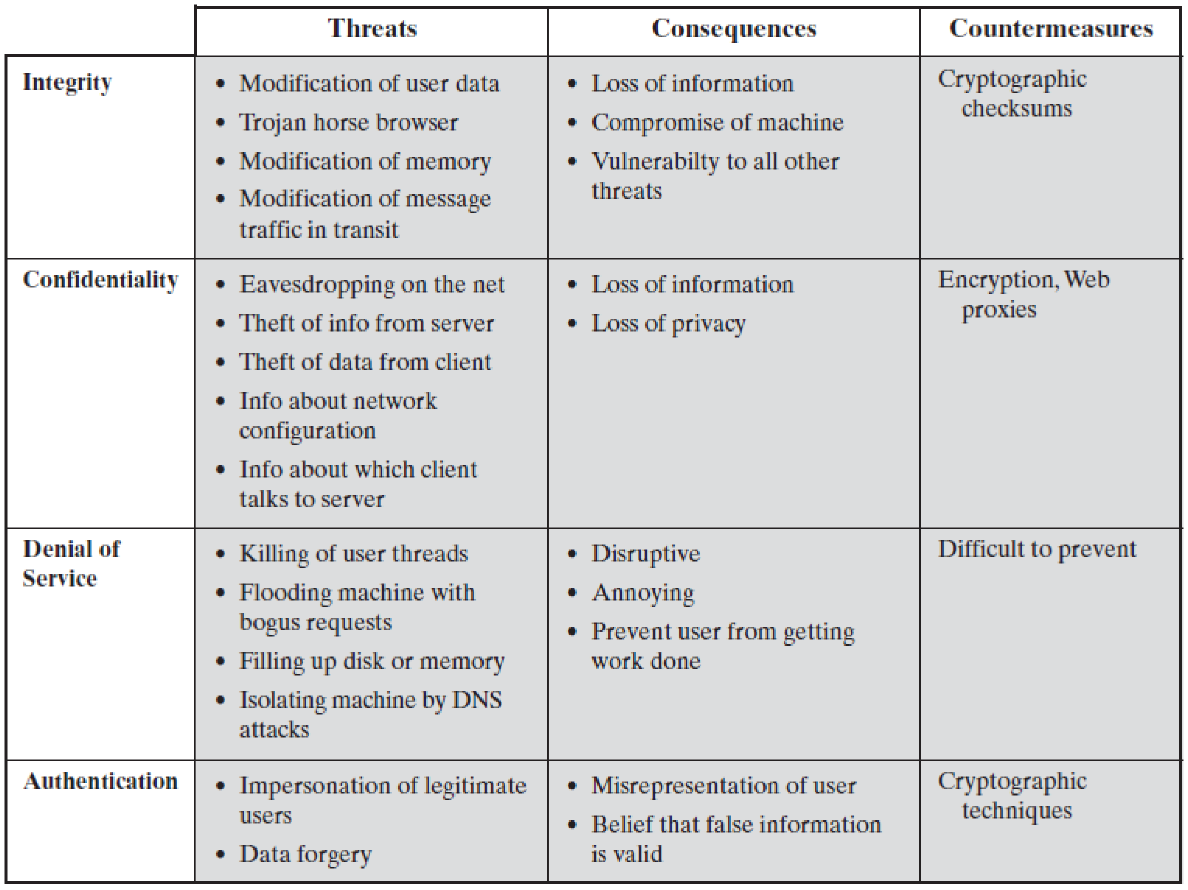
For the exchange of information and for commerce to be secure on any network, especially the Internet, a system or process must be put in place that satisfies requirements for confidentiality, access control, authentication, integrity, and non-repudiation. These requirements are achieved on the Web through 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:

Web Security Threats:

Table 16.1 provides a summary of the types of security threats faced when using the Web. One way to group these 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 Web site that is supposed to be restricted. Active attacks include impersonating another user, altering messages in transit between client and server, and altering information on a Web site. Another way to classify Web security threats is in terms of the location of the threat: Web server, Web browser, and network traffic between browser and server.

Table 16.1 A Comparison of Threats on the Web



	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 • Vulnerabilty 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 requests • 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

<u>SECURE SOCKET LAYER</u>

- Secure Socket Layer (SSL) provides security services between TCP and applications that use TCP. The Internet standard version is called Transport Layer Service (TLS).
- SSL/TLS provides confidentiality using symmetric encryption and message integrity using a message authentication code.
- SSL/TLS includes protocol mechanisms to enable two TCP users to determine the security mechanisms and services they will use.
- Netscape originated SSL.

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 rather two layers of protocols, as illustrated in Figure 16.2.

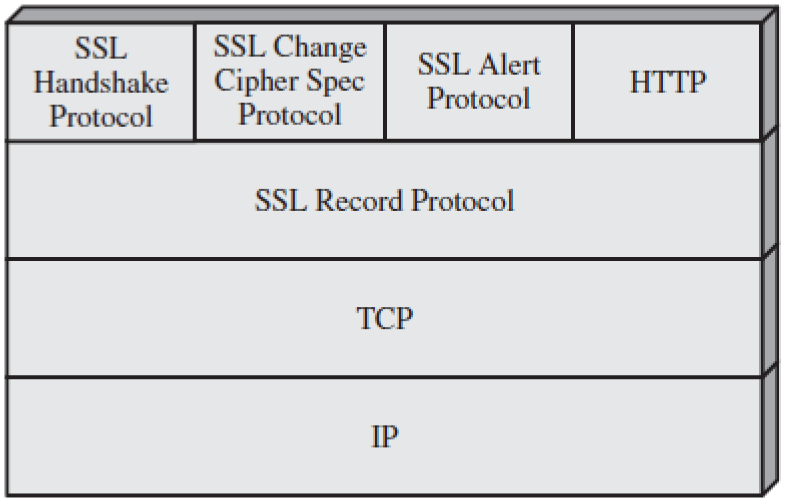


Figure 16.2 SSL Protocol Stack

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. These SSL-specific protocols are used in the management of SSL exchanges.

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.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..

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 secret encryption key for data encrypted by the server and decrypted by the client.
- **Client write key:** The symmetric 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.

- **Sequence numbers:** Each party maintains separate sequence numbers for transmitted and received messages for each connection.

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).

Figure 16.3 indicates 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 before being delivered to higher-level users.

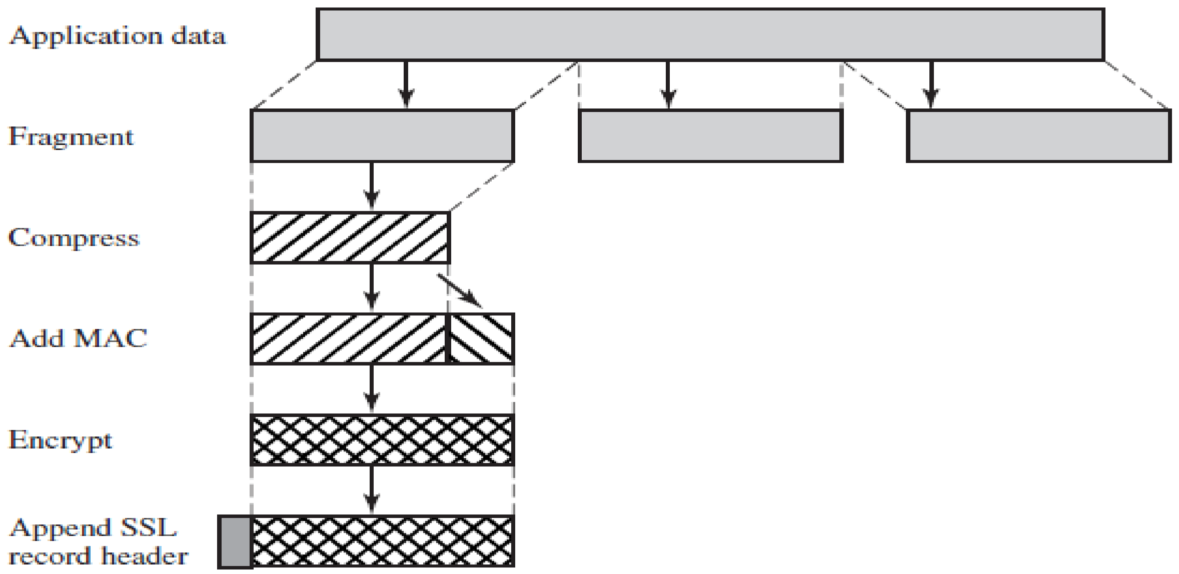


Figure 16.3 SSL Record Protocol Operation

- The first step is **fragmentation**. Each upper-layer message is fragmented into blocks of 2^{14} 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.1In SSLv3 (as well as the current version of TLS), no compression algorithm is specified, so the default compression algorithm is null.
- 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 next step is perform encryption and adds a header

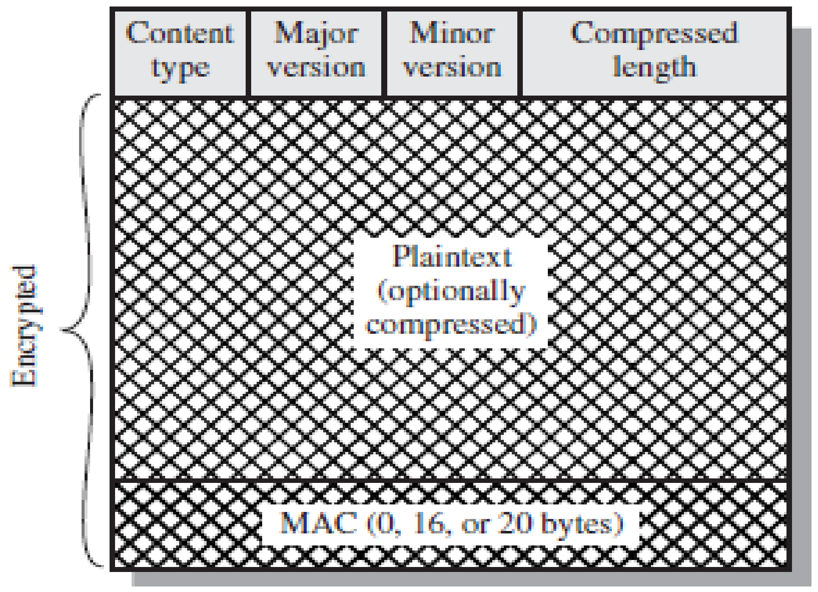


Figure 16.4 SSL Record Format

SSL Handshake Protocol

This phase is used to initiate a logical connection between client and server

The Handshake Protocol consists of a series of messages exchanged by client and server. All of these have the format shown in Figure 16.5c. Each message has three fields:

- **Type (1 byte):** Indicates one of 10 messages. Table 16.2 lists the defined message types.
- **Length (3 bytes):** The length of the message in bytes.
- **Content (≥ 0 bytes):** The parameters associated with this message; these are listed in Table 16.2.

Table 16.2 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

- It consists of 4 phases
  1. Establish Security Capabilities
  2. Server Authentication and Key Exchange
  3. Client Authentication and Key Exchange
  4. Finish

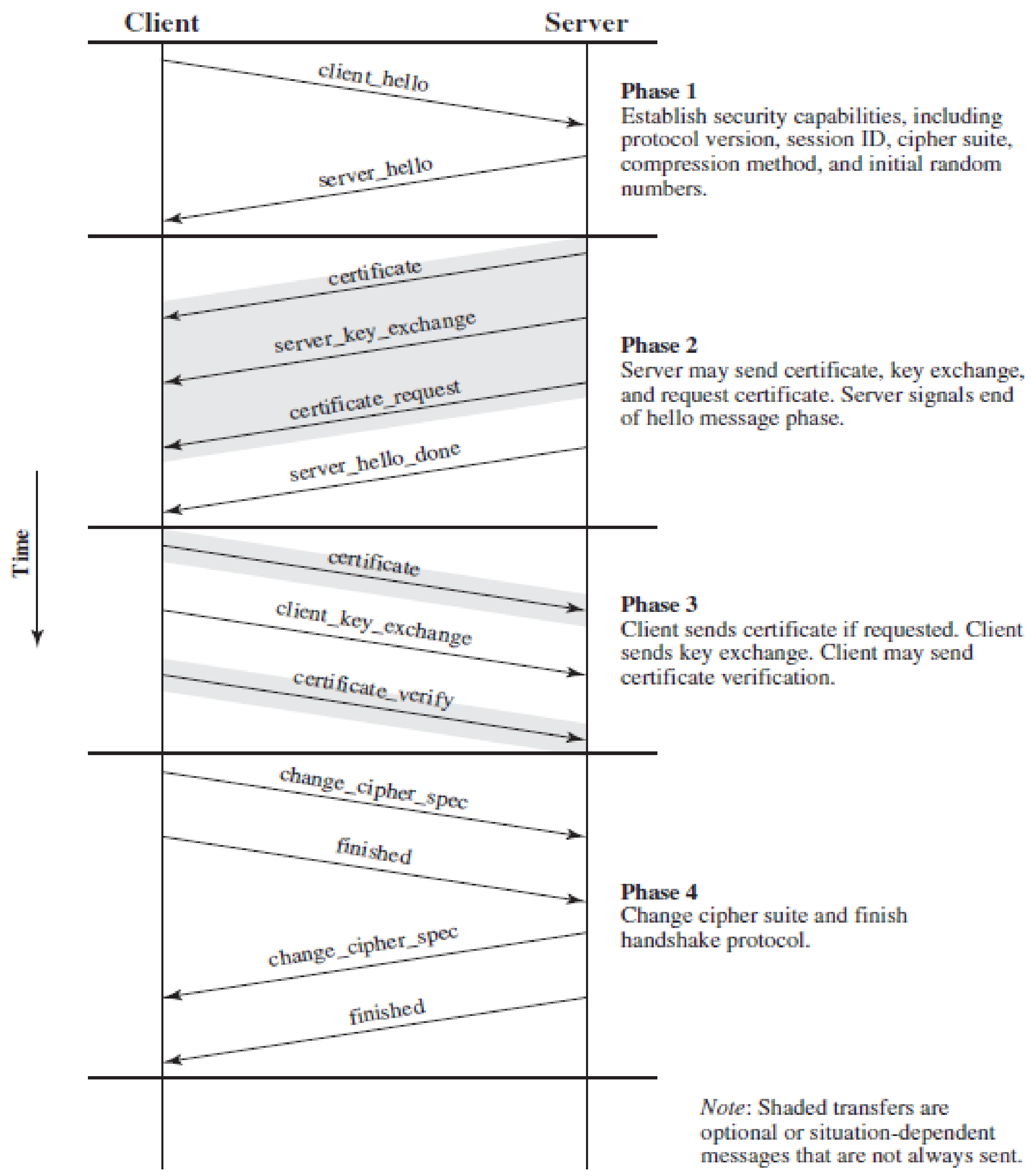

Client
Server
client_hello
server_hello
Phase 1
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.
certificate
server_key_exchange
certificate_request
server_hello_done
Phase 2
Server may send certificate, key exchange,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

Figure 16.6 Handshake Protocol Action

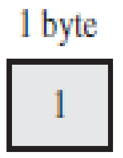
The exchange is initiated by the client, which sends a **client_hello message** with the following parameters:

- **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 to create a new connection on this session. A zero value indicates that the client 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; these are discussed subsequently.
- **Compression Method**: This is a list of the compression methods the client supports.

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 (Figure 16.5a), which consists of a single byte with the value 1.
- The sole purpose of this message is to cause the pending state to be copied into the current state,

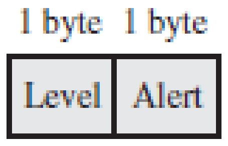


(a) Change Cipher Spec Protocol

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 (Figure 16.5b). 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
- The second byte contains a code that indicates the specific alert.



(b) Alert Protocol

Example Alerts

fatal: unexpected message, bad record mac, decompression failure, handshake failure, illegal parameter

warning: close notify, no certificate, bad certificate, unsupported certificate, certificate revoked, certificate expired, certificate unknown

unexpected_message: An inappropriate message was received.

bad_record_mac: An incorrect MAC was received.

TRANSPORT LAYER SECURITY

TLS was released in response to the Internet 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.

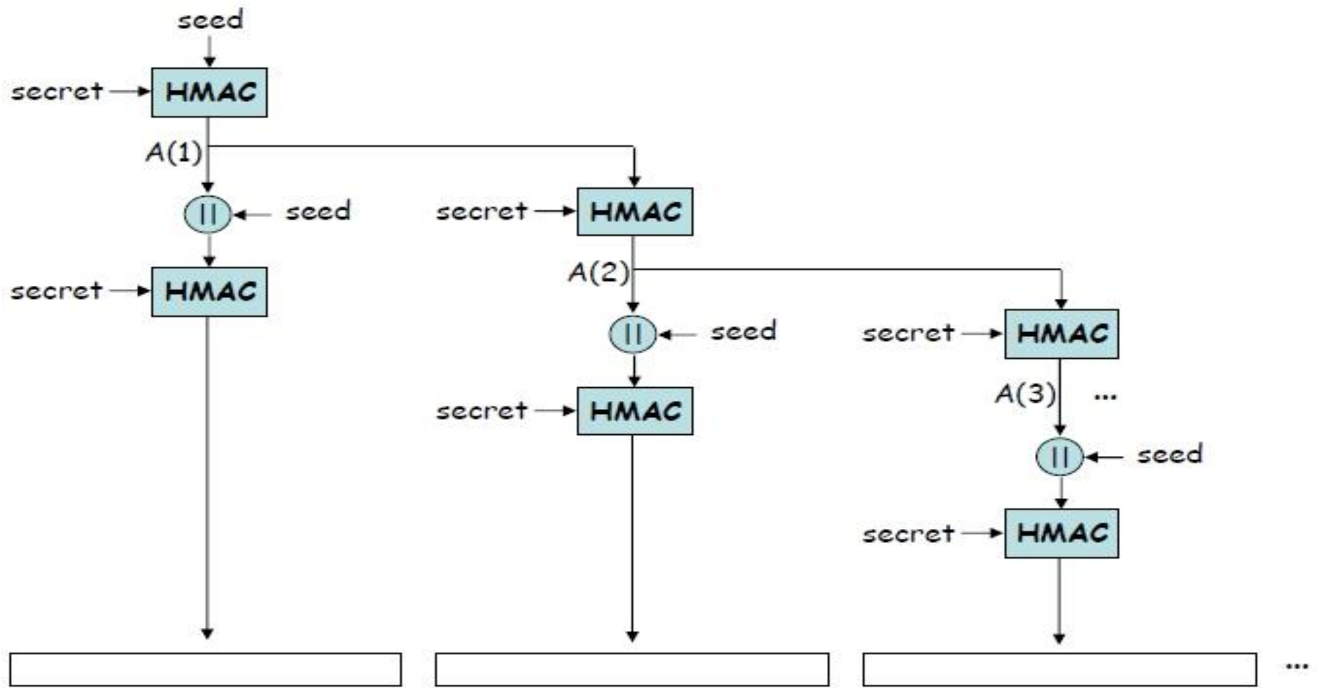
The TLS Handshake Protocol first negotiates key exchange using an asymmetric algorithm such as RSA or Diffie- Hellman .

TLS 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 3.

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.



```
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))
```

Alert Codes

TLS supports all of the alert codes defined in SSLv3 with the exception of no_certificate .A number of additional codes is defined in TLS. Some of them are

- **record_overflow**
- **unknown_ca**
- **access_denied**
- **protocol_version**
- **internal_error**
- **decrypt_error**

SECURE SHELL (SSH)

Secure Shell (SSH) is a protocol for secure network communications designed to be relatively simple and inexpensive to implement.The initial version, SSH1 was focused on providing a secure remote logon facility to replace TELNET and other remote logon schemes that provided no security. 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.

SSH2 is documented as a proposed standard in IETF RFCs 4250 through 4256.

SSH client and server applications are widely available for most operating systems.

It has become the method of choice for remote login and X tunneling and is rapidly becoming one of the most pervasive applications for encryption technology outside of embedded systems.

SSH is organized as three protocols that typically run on top of TCP (Figure 16.8):

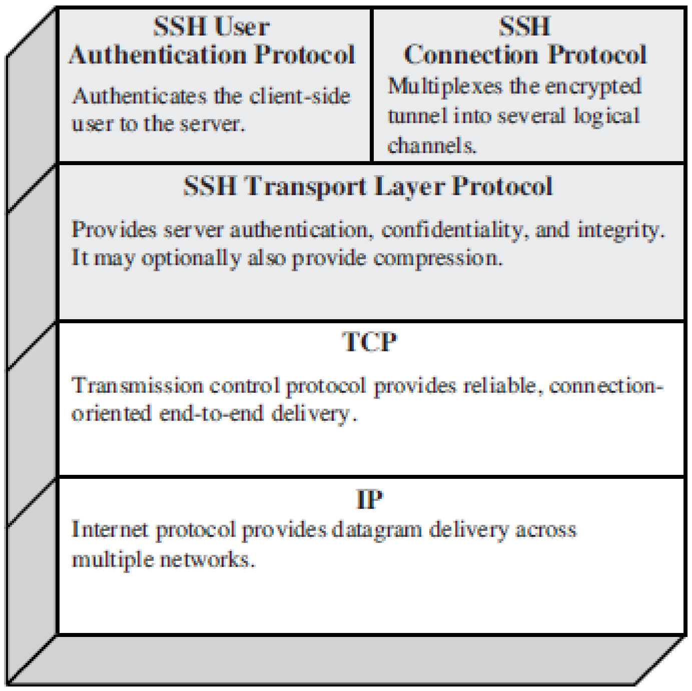

SSH User Authentication Protocol
Authenticates the client-side user to the server.
SSH Connection Protocol
Multiplexes the encrypted tunnel into several logical channels.
SSH Transport Layer Protocol
Provides server authentication, confidentiality, and integrity. It may optionally also provide compression.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

Figure 16.8 SSH Protocol Stack

1) Transport Layer Protocol: Provides server authentication, data confidentiality,and data integrity with forward secrecy (i.e., if a key is compromised during one session, the knowledge does not affect the security of earlier sessions).The transport layer may optionally provide compression..

Once the connection is established, the client and server exchange data, referred to as packets, in the data field of a TCP segment. Each packet is in the following format (Figure 16.10).

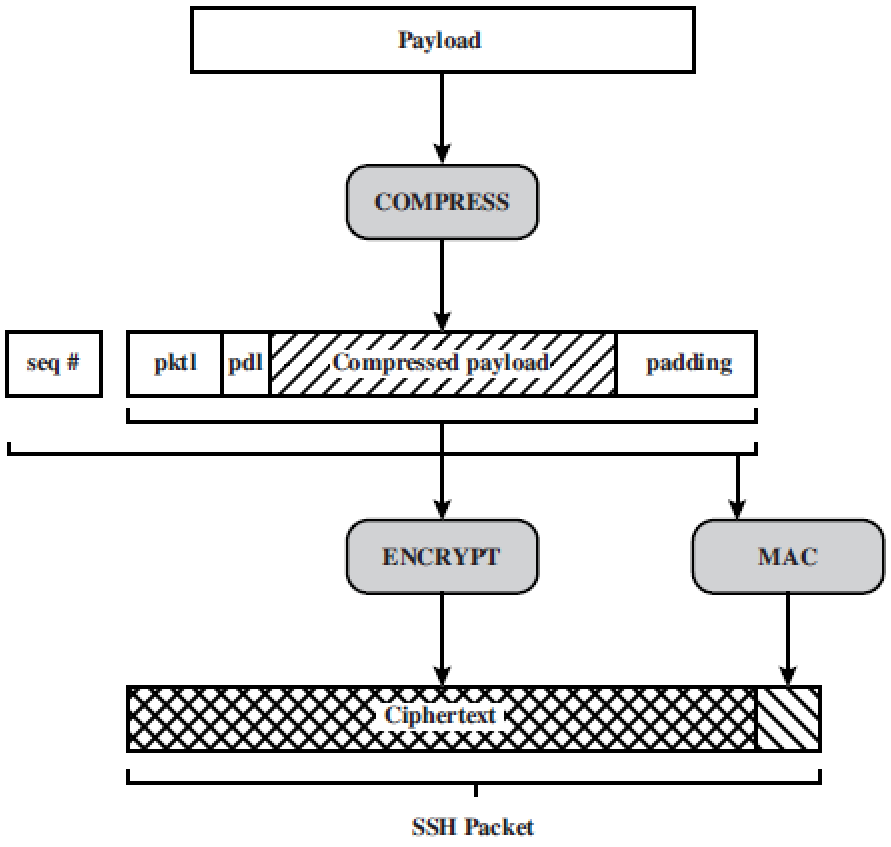
• **Packet length:** Length of the packet in bytes, not including the packet length and MAC fields.

• **Padding length:** Length of the random padding field.

• **Payload:** Useful contents of the packet. Prior to algorithm negotiation, this field is uncompressed. If compression is negotiated, then in subsequent packets, this field is compressed.

• **Random padding:** Once an encryption algorithm has been negotiated this field is added.

- **Message authentication code (MAC):** If message authentication has been negotiated, this field contains the MAC value.



pktl = packet length
pdl = padding length

Figure 16.10 SSH Transport Layer Protocol Packet Formation

2) User Authentication Protocol: Authenticates the user to the server.

Authentication Methods: The server may require one or more of the following authentication methods

- **Public key:** The details of this method depend on the public-key algorithm chosen. In essence, the client sends a message to the server that contains the client's public key, with the message signed by the client's private key.When the server receives this message, it checks whether the supplied key is acceptable for authentication and, if so, it checks whether the signature is correct.

• **Password:** The client sends a message containing a plaintext password, which is protected by encryption by the Transport Layer Protocol.

• **host based:** Authentication is performed on the client's host rather than the client itself

3) Connection Protocol: Multiplexes multiple logical communications channels over a single, underlying SSH connection.

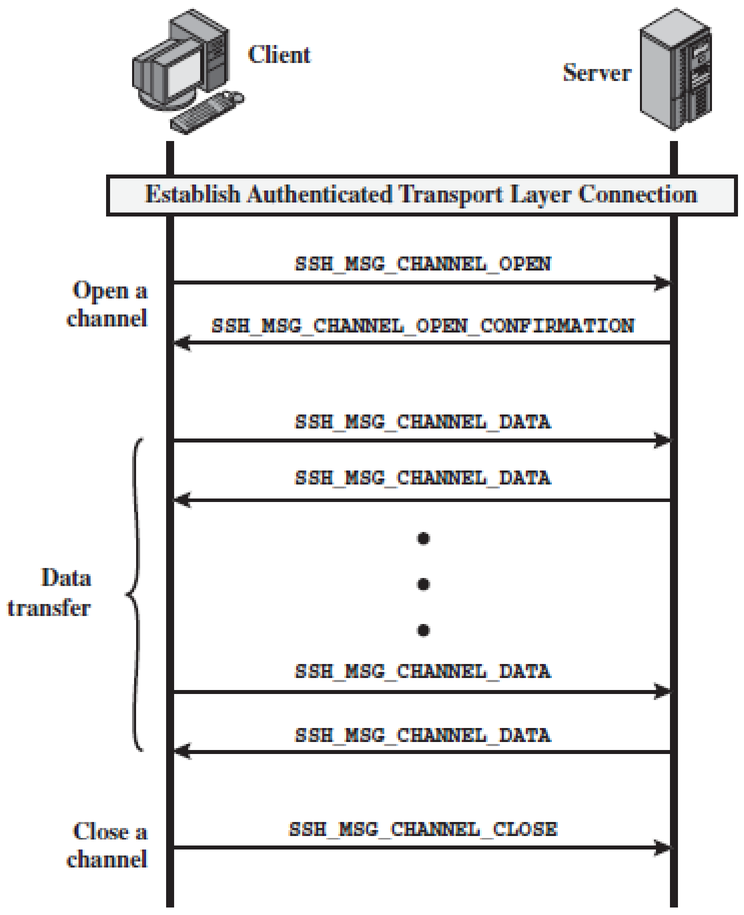


Figure 16.11 Example SSH Connection Protocol Message Exchange

ELECTRONIC MAIL SECURITY

The protection of email from unauthorized access and inspection is known as **electronic privacy**. There are mainly two methods for proving security for electronic mails

- Pretty Good Privacy
- S/MIME

Pretty Good Privacy:

In virtually all distributed environments, electronic mail is the most heavily used network based application.

Introduction:

- PGP is data encryption and decryption computer program that provides privacy (Confidentiality) and authentication for data communication.
- It was created by Phil Zimmermann in 1991

Use of PGP:

- It is used in Electronic mail
- File storage applications.
- PGP is an open-source, freely available software package for e-mail security. It provides **authentication** through the use of digital signature, **confidentiality** through the use of symmetric block encryption, **compression** using the ZIP algorithm, and **e-mail compatibility** using the radix-64 encoding scheme.
- PGP incorporates tools for developing a public-key trust model and public-key certificate management

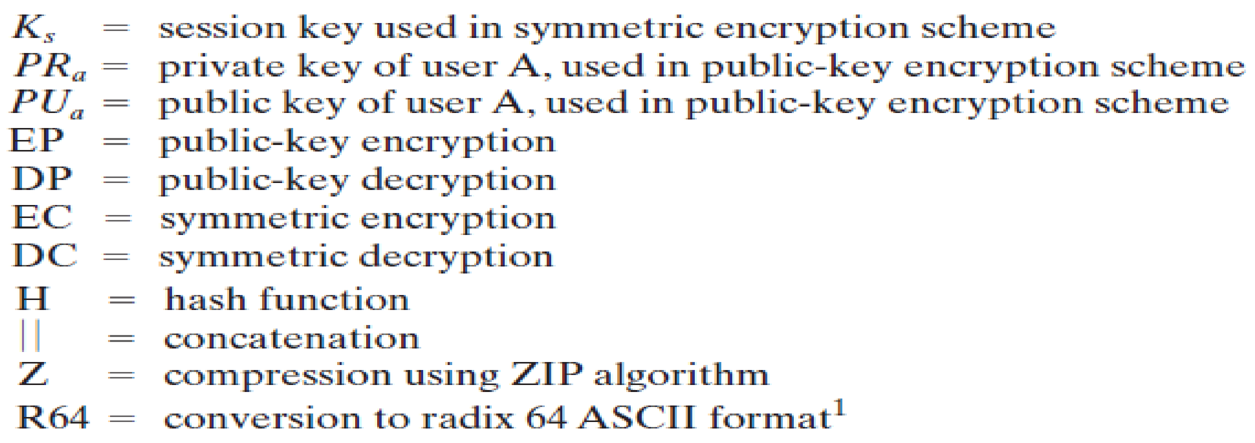
PGP has grown explosively and is now widely used, because of following reasons:

1. It is available free worldwide in versions that run on a variety of platforms, including Windows, UNIX, Macintosh, and many more. In addition, the commercial version satisfies users who want a product that comes with vendor support.
2. It is based on algorithms that have survived extensive public review and are considered extremely secure. Specifically, the package includes RSA, DSS, and Diffie-Hellman for public-key encryption; CAST-128, IDEA, and 3DES for symmetric encryption; and SHA-1 for hash coding.

3. It has a wide range of applicability, from corporations that wish to select and enforce a standardized scheme for encrypting files and messages to individuals who wish to communicate securely with others worldwide over the Internet and other networks.
4. It was not developed by, nor is it controlled by, any governmental or standards organization. For those with an instinctive distrust of “the establishment,” this makes PGP attractive.

NOTATIONS

The following symbols are used in PGP



K_s	=	session key used in symmetric encryption scheme
PR_a	=	private key of user A, used in public-key encryption scheme
PU_a	=	public key of user A, used in public-key encryption scheme
EP	=	public-key encryption
DP	=	public-key decryption
EC	=	symmetric encryption
DC	=	symmetric decryption
H	=	hash function
$\|\|$	=	concatenation
Z	=	compression using ZIP algorithm
R64	=	conversion to radix 64 ASCII format[1]

PGP SERVICES

1) authentication
2) confidentiality
3) compression
4) e-mail compatibility

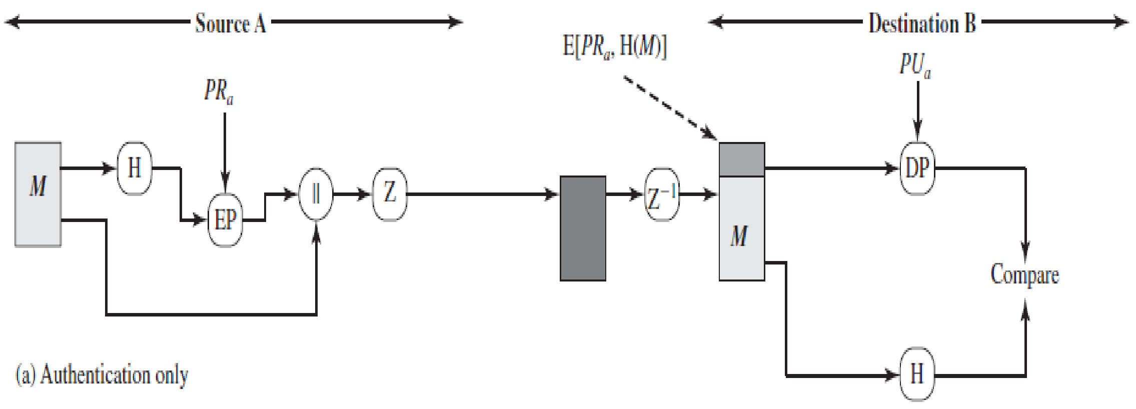
Function	Algorithms Used	Description

Digital signature	DSS/SHA or RSA/SHA	A hash code of a message is created using SHA-1.This message digest is encrypted using DSS or RSA with the sender's private key and included with the message.
Message encryption	CAST or IDEA or Three-key Triple DES with Diffie-Hellman or RSA	A message is encrypted using CAST-128 or IDEA or 3DES with a one-time session key generated by the sender. The session key is encrypted using Diffie-Hellman or RSA with the recipient's public key and included with the message.
Compression	ZIP	A message may be compressed for storage or transmission using ZIP.
E-mail compatibility	Radix-64 conversion	To provide transparency for e-mail applications, an encrypted message may be converted to an ASCII string using radix-64 conversion.

Providing authentication by using PGP:

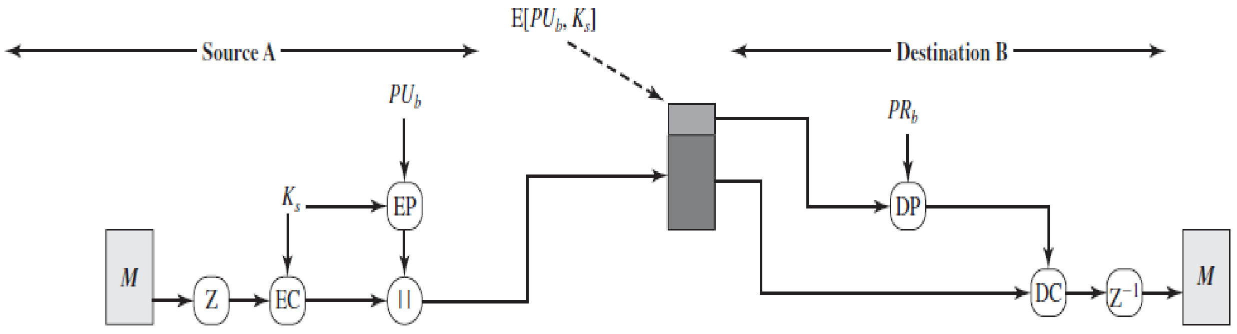
The sequence steps for providing authentication by using PGP

1. The sender creates a message.
2. SHA-1 is used to generate a 160-bit hash code of the message.
3. The hash code is encrypted with RSA using the sender's private key, and the result is prepended to the message.
4. The receiver uses RSA with the sender's public key to decrypt and recover the hash code.
5. The receiver generates a new hash code for the message and compares it with the decrypted hash code. If the two match, the message is accepted as authentic.



(a) Authentication only

Confidentiality by using PGP

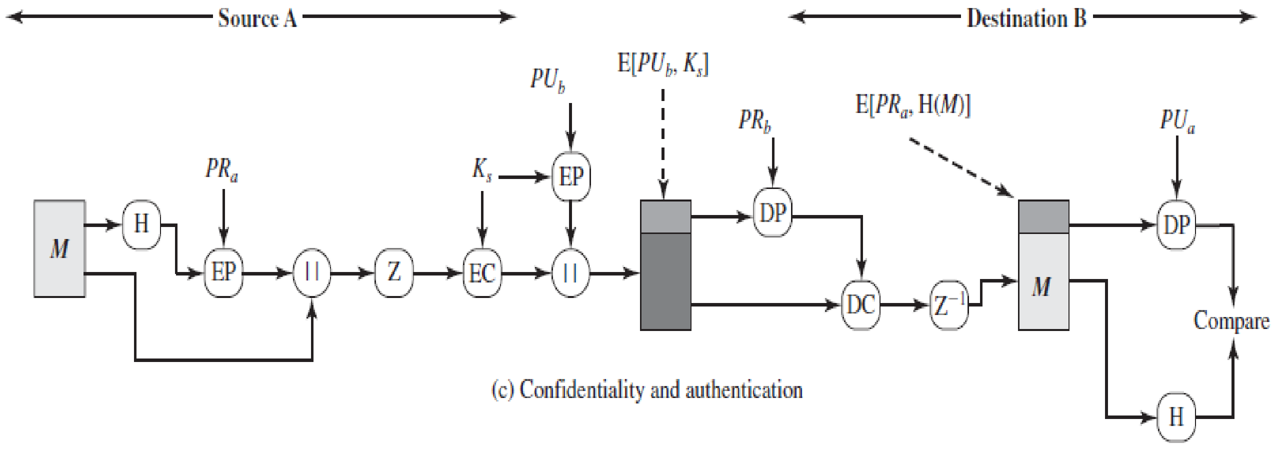


(b) Confidentiality only

Steps for providing confidentiality:

1. The sender generates a message and a random 128-bit number to be used as a session key for this message only.
2. The message is encrypted using CAST-128 (or IDEA or 3DES) with the session key.
3. The session key is encrypted with RSA using the recipient's public key and is prepended to the message.
4. The receiver uses RSA with its private key to decrypt and recover the session key.
5. The session key is used to decrypt the message.

PGP for both authentication and confidentiality:



(c) Confidentiality and authentication

COMPRESSION

As a default, PGP compresses the message after applying the signature but before Encryption. This has the benefit of saving space both for e-mail transmission and for file storage.

The signature is generated before compression for two reasons

- so can store uncompressed message & signature for later verification
- Message encryption is applied after compression to strengthen cryptographic security. Because the compressed message has less redundancy than the original plaintext, cryptanalysis is more difficult.

PGP uses ZIP compression algorithm

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 (with the sender's private key). If the confidentiality service is used, the message plus signature (if present) are encrypted (with a one-time symmetric key).Thus, part or the entire resulting block consists of a stream of arbitrary 8-bit octets.

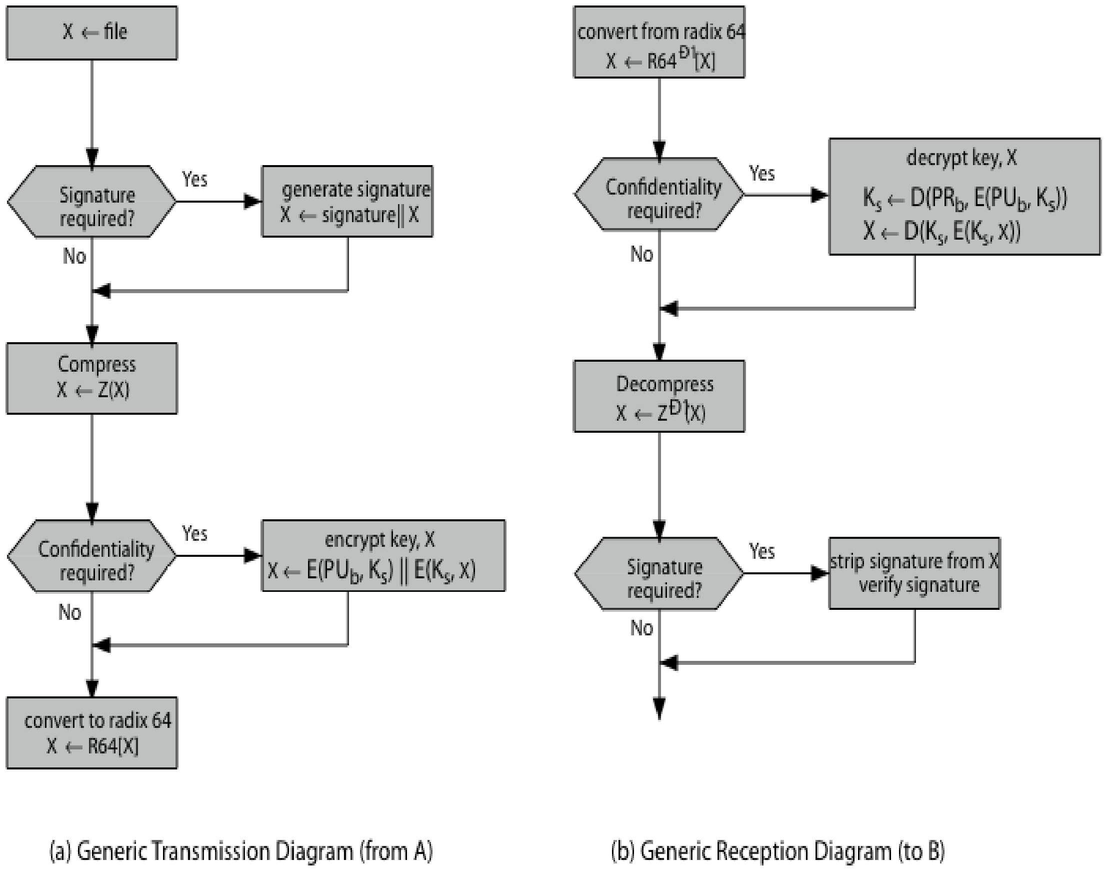
However, many electronic mail systems only permit the use of blocks consisting of ASCII text. To accommodate this restriction, PGP provides the service of converting the raw 8-bit binary stream to a stream of printable ASCII characters.

The scheme used for this purpose is radix-64 algorithm

- It maps 3 bytes to 4 printable chars
- It also appends a CRC to detect transmission errors

PGP also segments messages if too big

PGP Operation – Summary


X ← file
Signature required?
Yes
generate signature X ← signature|| X
No
Compress X ← Z(X)
Confidentiality required?
Yes
encrypt key, X
X ← E(PUb, Ks) || E(Ks, X)
No
convert to radix 64 X ← R64[X]
(a) Generic Transmission Diagram (from A)
convert from radix 64 X ← R64^-1[X]
Confidentiality required?
Yes
decrypt key, X
Ks ← D(PRb, E(PUb, Ks))
X ← D(Ks, E(Ks, X))
No
Decompress X ← Z^-1(X)
Signature required?
Yes
strip signature from X verify signature
No
(b) Generic Reception Diagram (to B)

S/MIME

- Secure/Multipurpose Internet Mail Extension is a security enhancement to the MIME internet email standard.
- S/MIME is for industry standard for commercial and organizational use.
- It defined in number of documents that is RFC 2630, RFC 2632, RFC 2633

E-mail format standard, RFC 822, which is still in common use. The most recent version of this format specification is RFC 5322 (Internet Message Format).

RFC 5322 (Internet Message Format).

RFC 5322 defines a format for text messages that are sent using electronic mail. It has been the standard for Internet-based text mail messages and remains in common use.

Message Structure

- A message consists of some number of header lines (*the header*) followed by unrestricted text (*the body*).
- A header line usually consists of a keyword, followed by a colon, followed by the keyword's arguments

  Example

  Date: October 8, 2009 2:15:49 PM EDT

  From: William Stallings <ws@shore.net>

  Subject: The Syntax in RFC 5322

  To: Smith@Other-host.com

  Cc: Jones@Yet-Another-Host.com

  Hello. This section begins the actual
  message body, which is delimited from the
  message heading by a blank line.

Multipurpose Internet Mail Extensions:

Multipurpose Internet Mail Extension (MIME) is an extension to the RFC 5322 framework that is intended to address some of the problems and limitations of the use of Simple Mail Transfer Protocol (SMTP).

The following are limitations of the SMTP/5322 scheme.

1. SMTP cannot transmit executable files or other binary objects. A number of schemes are in use for converting binary files into a text form that can be used by SMTP mail systems.
2. SMTP cannot transmit text data that includes national language characters, because these are represented by 8-bit codes, and SMTP is limited to 7-bit ASCII.
3. SMTP servers may reject mail message over a certain size.
4. SMTP gateways that translate between ASCII and the character code EBCDIC do not use a consistent set of mappings, resulting in translation problems.
5. SMTP gateways to X.400 electronic mail networks cannot handle non-textual data included in X.400 messages.
6. Some SMTP implementations do not adhere completely to the SMTP standards defined in RFC 821. Common problems include:
    a. Deletion, addition, or reordering of carriage return and linefeed
    b. Truncating or wrapping lines longer than 76 characters
    c. Removal of trailing white space (tab and space characters)
    d. Padding of lines in a message to the same length
    e. Conversion of tab characters into multiple space characters

MIME is intended to resolve these problems in a manner that is compatible with existing RFC 5322 implementations. The specification is provided in RFCs 2045 through 2049.

MIME has 5 header fields

The five header fields defined in MIME are

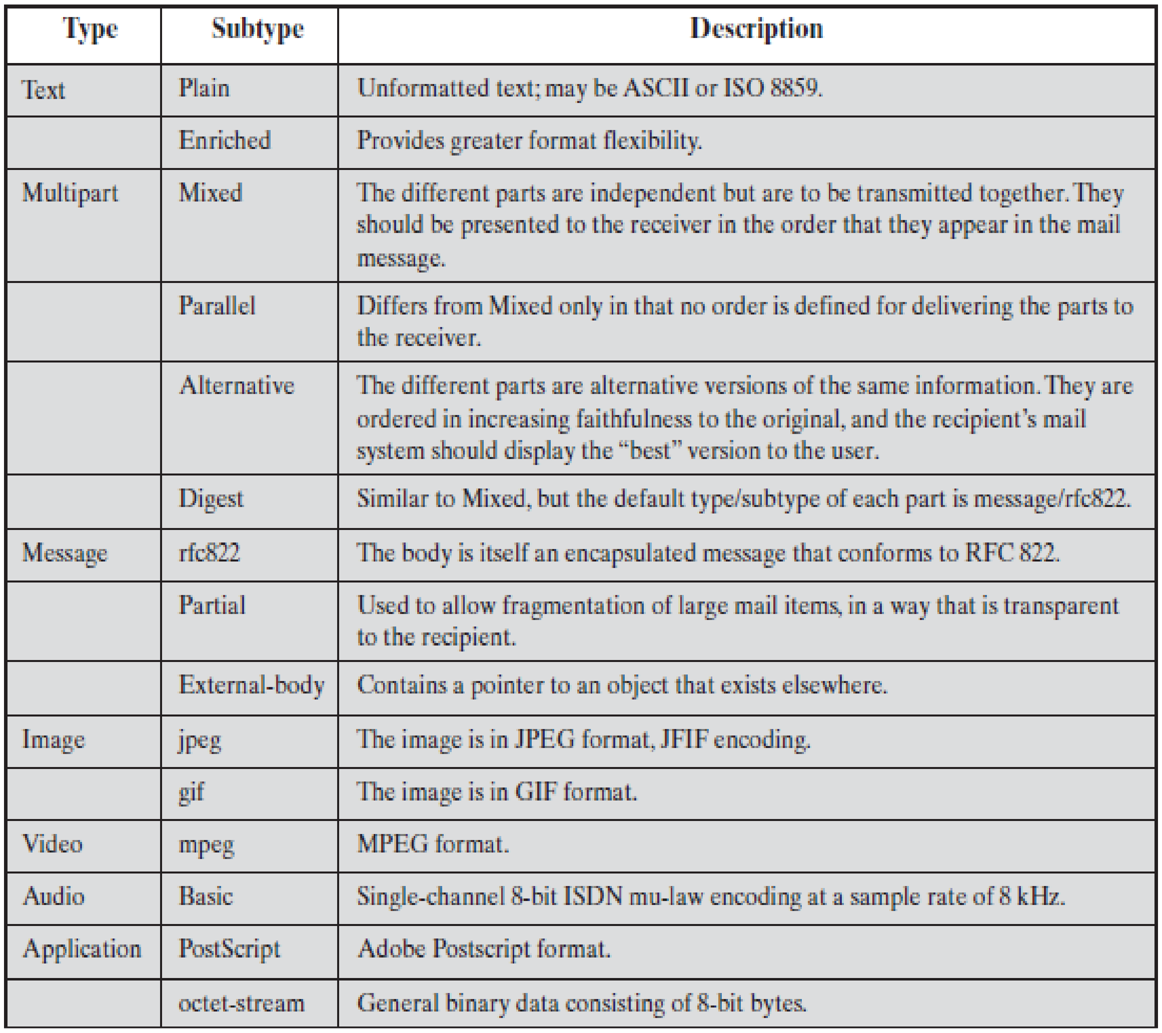
1. **MIME-Version**: Must have the parameter value 1.0.
2. **Content-Type**: Describes the data contained in the body with sufficient detail that the receiving user agent can pick an appropriate agent or mechanism to represent the data to the user
3. **Content-Transfer-Encoding**: Indicates the type of transformation that has been used to represent the body of the message in a way that is acceptable for mail transport.

4. **Content-ID**: Used to identify MIME entities uniquely in multiple contexts.
5. **Content-Description**: A text description of the object with the body; this is useful when the object is not readable (e.g., audio data).

Mime Content Types

Describes the data contained in the body with sufficient detail

Table 18.3 MIME Content Types



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 format.
	octet-stream	General binary data consisting of 8-bit bytes.

Example

Here is a simple example of a multipart message containing two parts--**both consisting of simple text**

From: Nathaniel Borenstein nsb@bellcore.com>

To: Ned Freed <ned@innosoft.com>

Subject: Sample message

MIME-Version: 1.0

Content-type: multipart/mixed; boundary="simple boundary“

This is the preamble. It is to be ignored, though it

is a handy place for mail composers to include an

explanatory note to non-MIME conformant readers.

—simple boundary

Content-type: text/plain; charset=us-ascii

This is explicitly typed plain ASCII text. It DOES end with a line break.

S/MIME Functionality

In terms of general functionality, S/MIME is very similar to PGP. Both offer the ability to sign and/or encrypt messages. In this subsection, we briefly summarize S/MIME capability.

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.
- **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 base64 encoding. A signed data message can only be viewed by a recipient with S/MIME capability.
    - encoded (message + signed digest)

- **Clear-signed data:** As with signed data, a digital signature of the content is formed. However, in this case, only the digital signature is encoded using base64.As a result, recipients without S/MIME capability can view the message content, although they cannot verify the signature.
    - clear text message + encoded (signed digest)

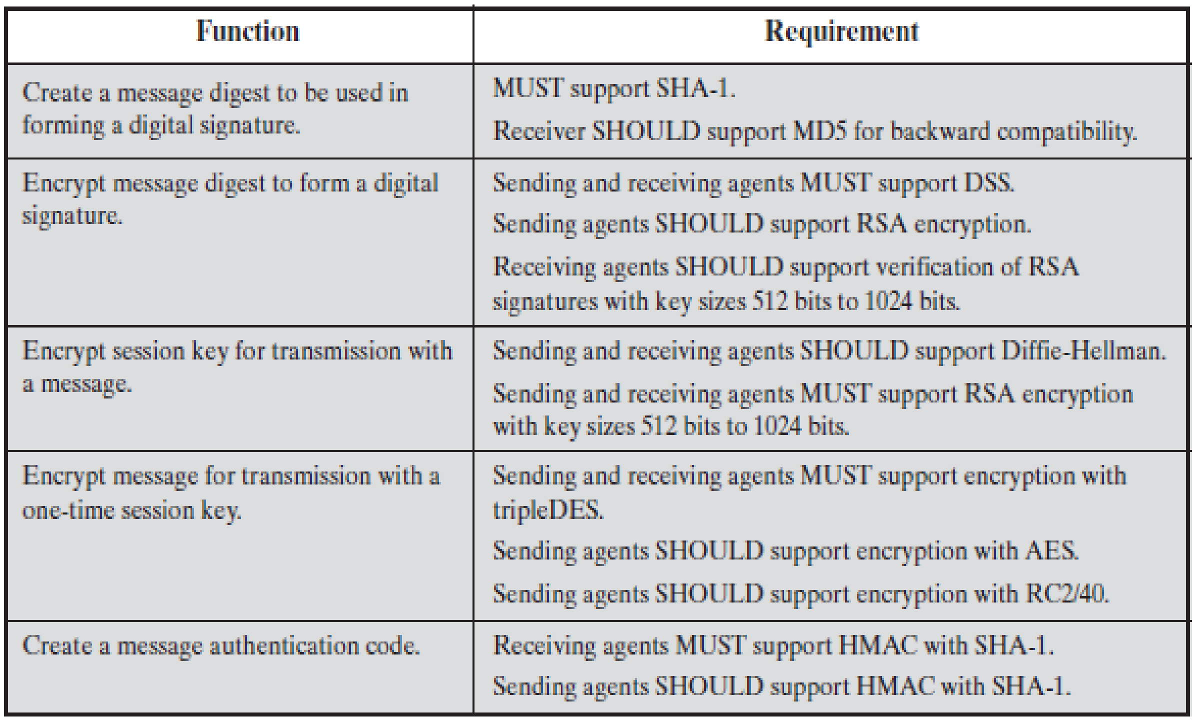
- **Signed and enveloped data:** Signed-only and encrypted-only entities may be nested, so that encrypted data may be signed and signed data or clear-signed data may be encrypted.
    - nesting of signed & encrypted entities

CRYPTOGRAPHIC ALGORITHMS

S/MIME uses the following terminology taken from RFC 2119 (Key Words for use in RFCs to Indicate Requirement Levels) to specify the requirement level:

- **MUST**: The definition is an absolute requirement of the specification. An implementation must include this feature or function to be in conformance with the specification.
- **SHOULD**: There may exist valid reasons in particular circumstances to ignore this feature or function, but it is recommended that an implementation include the feature or function.

Table 18.6 Cryptographic Algorithms Used in S/MIME



Function	Requirement
Create a message digest to be used in forming a digital signature.	MUST support SHA-1. Receiver SHOULD support MD5 for backward compatibility.
Encrypt message digest to form a digital signature.	Sending and receiving agents MUST support DSS. Sending agents SHOULD support RSA encryption. Receiving agents SHOULD support verification of RSA signatures with key sizes 512 bits to 1024 bits.
Encrypt session key for transmission with a message.	Sending and receiving agents SHOULD support Diffie-Hellman. Sending and receiving agents MUST support RSA encryption with key sizes 512 bits to 1024 bits.
Encrypt message for transmission with a one-time session key.	Sending and receiving agents MUST support encryption with tripleDES. Sending agents SHOULD support encryption with AES. Sending agents SHOULD support encryption with RC2/40.
Create a message authentication code.	Receiving agents MUST support HMAC with SHA-1. Sending agents SHOULD support HMAC with SHA-1.

S/MIME incorporates three public-key algorithms. The Digital Signature Standard (DSS) is the preferred algorithm for digital signature.