MODULE 4

Module-4 10 hours

User Authentication: Remote user authentication principles, Kerberos, Remote user authentication using asymmetric encryption.
Web security consideration, Transport layer security.
Email Threats and comprehensive email security, S/MIME, Pretty Good Privacy.

USER AUTHENTICATION

User authentication is the **fundamental component of computer security** that establishes the identity of users before granting access to system resources. It forms the **foundation for access control** and **user accountability**. According to **RFC 4949**, user authentication is defined as *the process of verifying an identity claimed by or for a system entity*

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."

In essence, 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.

The NIST Model for Electronic User Authentication

According to **NIST SP 800-63-2**, **electronic authentication** refers to *establishing confidence in user identities* presented electronically—typically while accessing online or distributed resources. This process involves several entities:

- The **Registration Authority (RA)** verifies the applicant's identity and vouches for it to the **Credential Service Provider (CSP)**.

- The **CSP** issues an **electronic credential** that binds the user's identity to an **authentication token** (such as a password, cryptographic key, or smart card).
- The registered user becomes a **subscriber**, and during authentication, the subscriber acts as a **claimant** who proves identity to a **verifier**.
- The **verifier** checks the claimant's credentials and provides an **assertion** of verified identity to a **relying party**, which decides whether to grant access.

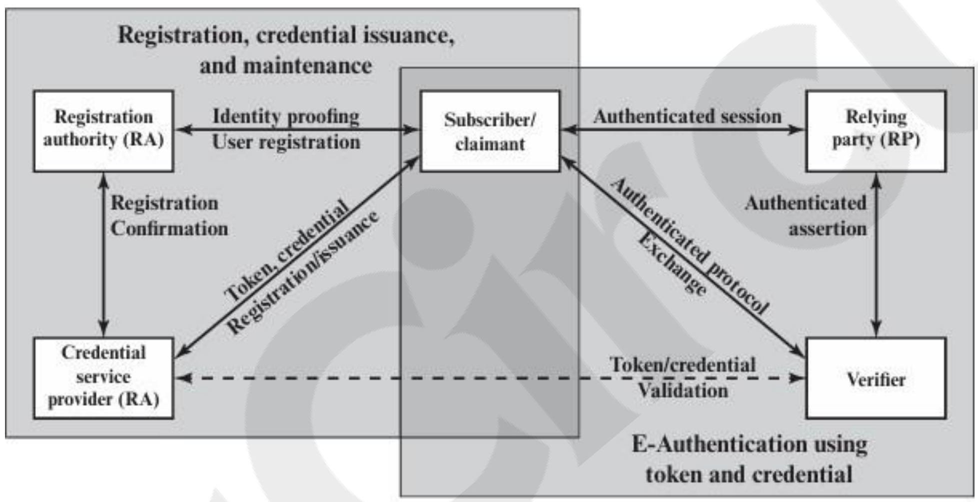


Figure 15.1 The NIST SP 800-63-2 E-Authentication Architectural Model

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

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

• **Something the individual possesses:** Examples include electronic keycards, smartcards, 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.

All of these methods, properly implemented and used, can provide secure user authentication. However, each method has problems. An adversary may be able to guess or steal a password. Similarly, an adversary may be able to forge or steal a token. A user may forget a password or lose a token. Further, there is a significant administrative overhead for managing password and token information on systems and securing such information on systems. With respect to biometric authenticators, there are a variety of problems, including dealing with false positives and false negatives, user acceptance, cost, and convenience.

Authentication Protocols

- An important application area is that of **mutual authentication** protocols. Such protocols enable communicating parties to satisfy themselves mutually about each other's identity and to exchange session keys. There, the focus was key distribution.
- Central to the problem of authenticated key exchange are two issues: **confidentiality and timeliness.**
- To prevent masquerade and to prevent compromise of session keys, essential identification and session key information must be communicated in encrypted form. The second issue, timeliness, is important because of the threat of message replays.

Replay Attacks: are where a valid signed message is copied and later resent. Such replays, at worst, could allow an opponent to compromise a session key or successfully impersonate another party. At minimum, a successful replay can disrupt operations by presenting parties with messages that appear genuine but are not.

Examples of replay attacks:

Simple replay: The opponent simply copies a message and replays it later.

Repetition that can be logged: An opponent can replay a timestamped message with in the valid time window.

Repetition that cannot be detected: This situation could arise because the original message could have been suppressed and thus did not arrive at its destination; only there play message arrives.

Backward replay without modification: This is a replay back to the message sender. This attack is possible if symmetric encryption is used and the sender cannot easily recognize the difference between messages sent and messages received on the basis of content.

Possible countermeasures include the use of:

• **Sequence numbers** (generally impractical since must remember last number used with every communicating party)

• **Timestamps** (needs synchronized clocks amongst all parties involved, which can be problematic)

• **Challenge/response** (using unique, random, unpredictable nonce, but not suitable for connectionless applications because of handshake overhead)

One-Way Authentication

One application for which encryption is growing in popularity is electronic mail (e-mail). The very nature of electronic mail, and its chief benefit, is that it is not necessary for the sender and receiver to be online at the same time. Instead, the e-mail message is forwarded to the receiver's electronic mailbox, where it is buffered until the receiver is available to read it. Accordingly, the e-mail message should be encrypted such that the mail-handling system is not in possession of the decryption key. A second requirement is that of authentication. Typically, the recipient wants some assurance that the message is from the alleged sender.

REMOTE USER-AUTHENTICATION USING SYMMETRICENCRYPTION

Mutual Authentication

As discussed earlier, A two-level hierarchy of symmetric encryption keys can be used to provide confidentiality for communication in a distributed environment. Usually involves the use of a trusted key distribution center (KDC). Each party in the network shares a secret master key with the KDC.

The KDC is responsible for generating session keys, and for distributing those keys to the parties involved, using the master keys to protect these session keys.

Needham-Schroeder Protocol

The Needham-Schroeder Protocol is the original, basic key exchange protocol. Used by 2 parties who both trusted a common key server, it gives one party the info needed to establish a session key with the other.

Note that all communications is between A&KDC and A&B, B&KDC don't talk directly (though indirectly a message passes from KDC via A to B, encrypted in B's key so that A is unable to read or alter it). Other variations of key distribution protocols can involve direct communications between B&KDC.

1. $A \rightarrow KDC$: $ID_A \| ID_B \| N_1$
2. $KDC \rightarrow A$: $E(K_a, [K_s \| ID_B \| N_1 \| E(K_b, [K_s \| ID_A])])$
3. $A \rightarrow B$: $E(K_b, [K_s \| ID_A])$
4. $B \rightarrow A$: $E(K_s, N_2)$
5. $A \rightarrow B$: $E(K_s, f(N_2))$ where f() is a generic function that modifies the value of the nonce.

Secret keys K_a and K_b are shared between A and the KDC and B and the KDC, respectively. The purpose of the protocol is to distribute securely a session key K_s to A and B.

There is a critical flaw in the protocol, as shown. The message in step 3 can be decrypted, and hence understood only by B. But if an opponent, X, has been able to compromise an old session key, then X can impersonate A and trick B into using the old key by simply replaying step 3. Admittedly, this is a much more unlikely occurrence than that an opponent has simply observed and recorded step 3.

Denning proposes to overcome this weakness by a modification to the Needham/Schroeder protocol that includes the addition of a timestamp to steps 2 and 3. Her proposal assumes that the master keys, K_a and K_b are secure, and it consists of the following steps.

1. A → KDC: $ID_A \| ID_B$
2. KDC → A: $E(K_a, [K_s \| ID_B \| T \| E(K_b, [K_s \| ID_A \| T])])$
3. A → B: $E(K_b, [K_s \| ID_A \| T])$
4. B → A: $E(K_s, N_1)$
5. A → B: $E(K_s, f(N_1))$

T is a timestamp that assures A and B that the session key has only just been generated. Thus, both A and B know that the key distribution is a fresh exchange.

The Denning protocol seems to provide an increased degree of security compared to the Needham/Schroeder protocol. However, a new concern is raised: namely, that this new scheme requires reliance on clocks that are synchronized throughout the network. It points out a risk involved. The risk is based on the fact that the distributed clocks can become unsynchronized as a result of faults in the clocks or the synchronization mechanism.

The problem occurs **when a sender's clock is ahead of the intended recipient's clock.** In this case, an opponent can intercept a message from the sender and replay it later when the timestamp in the message becomes current at the recipient's site. This replay could cause unexpected results. Gong refers to such attacks as **suppress-replay attacks.**

KERBEROS

Kerberos is an authentication service developed as part of Project Athena at MIT, and is one of the best known and most widely implemented **trusted third party** key distribution systems.

Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users. Unlike most other authentication schemes, Kerberos relies exclusively on symmetric encryption, making no use of public- key encryption. Two versions of Kerberos are in common use: v4 & v5.

In a more open environment, in which network connections to other machines are supported, an approach that requires the user to prove his or her identity for each service invoked, and also require that servers prove their identity to clients, is needed to protect user information and resources housed at the server. Kerberos supports this approach, and assumes a distributed client/server architecture that employs one or more Kerberos servers to provide an authentication service. The first published report on Kerberos[STEI88] listed the following requirements:

• **Secure:** A network eavesdropper should not be able to obtain the necessary information to impersonate a user.

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

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

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

To support these requirements, Kerberos is a trusted third-party authentication service that uses a protocol based on that proposed by Needham and Schroeder which was discussed earlier in this chapter.

The Version 4 Authentication Dialogue

- Kerberos V4 is a basic third-party authentication scheme.
  - The core of Kerberos is the **Authentication server** (AS) and **Ticket Granting Servers (TGS)** – these are trusted by all users and servers and must be securely administered.
  - The protocol includes a sequence of interactions between the client, AS, TGT and desired server. Version 4 of Kerberos makes use of DES, in a rather elaborate protocol, to provide the authentication service

The heart of the first problem is the lifetime associated with the ticket-granting ticket. If this lifetime is very short (e.g., minutes), then the user will be repeatedly asked for a password. If the lifetime is long (e.g., hours), then an opponent has a greater opportunity for replay. Similarly, if an opponent captures a service-granting ticket and uses it before it expires, the opponent has access to the corresponding service.

The second problem is that there may be a requirement for servers to authenticate themselves to users.

First, consider the problem of captured ticket-granting tickets and the need to determine that the ticket presenter is the same as the client for whom the ticket was issued. An efficient way of doing this is to use a session encryption key to secure information. Table 15.1a shows the technique for distributing the session key.

Table 14.1 Summary of Kerberos Version 4 Message Exchanges

(1) C → AS $ID_c \parallel ID_{tgs} \parallel TS_1$

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

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

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

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

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

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

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

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

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

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

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

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

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

(c) Client/Server Authentication Exchange to obtain service

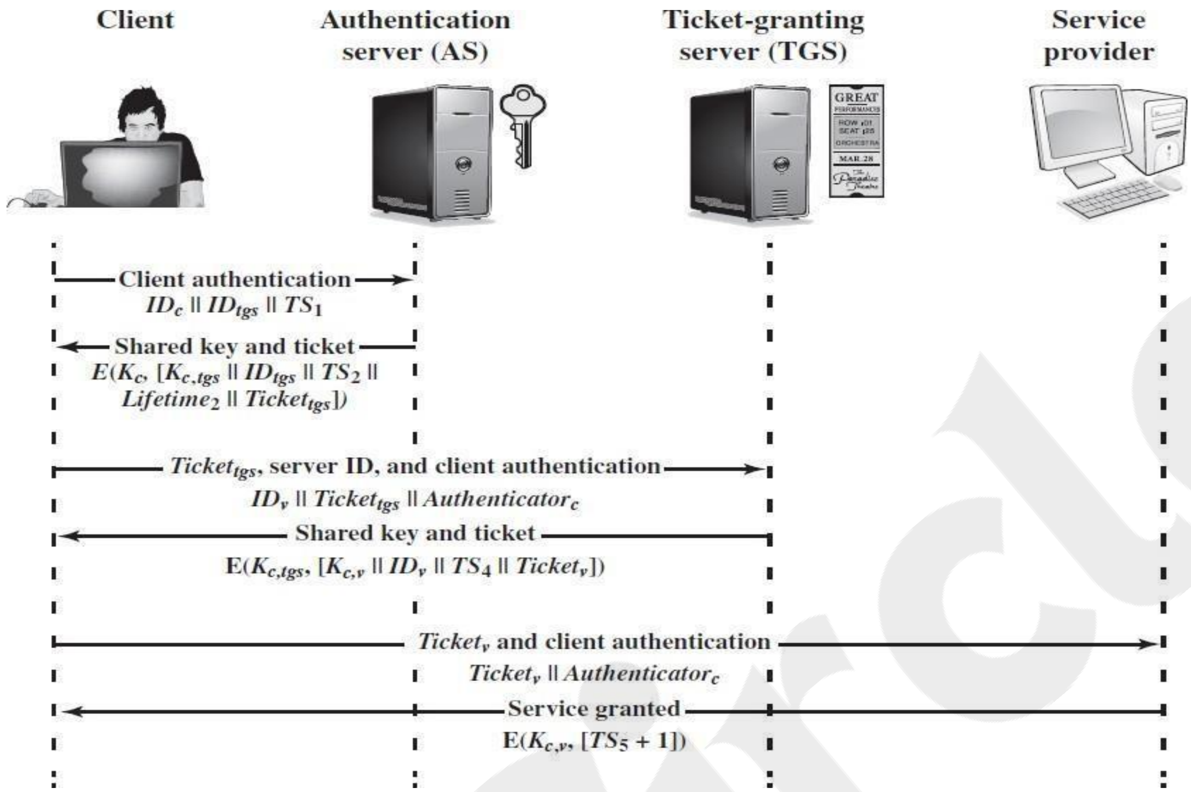
Table a shows the technique for distributing the session key. As before, the client sends a message to the AS requesting access to the TGS. The AS responds with a message, encrypted with a key derived from the user’s password (Kc) that contains the ticket. The encrypted message also contains a copy of the session key,(Kc,tgs), where the subscripts indicate that this is a session key for C and TGS. Because this session key is inside the message encrypted with (Kc), only the user’s client can read it. The same session key is included in the ticket, which can be read only by the TGS. Thus, the session key has been securely delivered to both C and the TGS.

Note that several additional pieces of information have been added to this first phase of the dialogue. Message (1) includes a timestamp, so that the AS knows that the message is timely. Message (2) includes several elements of the ticket in a form accessible to C. This enables C to confirm that this ticket is for the TGS and to learn its expiration time. Note that the ticket does not prove anyone's identity but is a way to distribute keys securely. It is the authenticator that proves the client's identity.

C sends the TGS a message that includes the ticket plus the ID of the requested service (message (3) in Table b). In addition, C transmits an authenticator, which includes the ID and address of C’s user and a timestamp. Unlike the ticket, which is reusable, the authenticator is intended for use only once and has a very short lifetime. The TGS can decrypt the ticket with the key that it shares with the AS. This ticket indicates that user C has been provided with the session key Kc,tgs. In effect, the ticket says,“Anyone who uses Kc,tgs must be C.” The TGS uses the session key to decrypt the authenticator.

The reply from the TGS, in message (4), follows the form of message (2). C now has a reusable service-granting ticket for V. When C presents this ticket, as shown in message (5), it also sends an authenticator. The server can decrypt the ticket, recover the session key, and decrypt the authenticator. If mutual authentication is required, the server can reply as shown in message (6).

Finally, at the conclusion of this process, the client and server share a secret key. This key can be used to encrypt future messages between the two or to exchange a new random session key for that purpose.



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

(a) Authentication Service Exchange

Message (3)	Client requests service-granting ticket.
ID_V	Tells TGS that user requests access to server V.
$Ticket_{tgs}$	Assures TGS that this user has been authenticated by AS.
$Authenticator_c$	Generated by client to validate ticket.
Message (4)	TGS returns service-granting ticket.
$K_{c,tgs}$	Key shared only by C and TGS protects contents of message (4).
$K_{c,v}$	Copy of session key accessible to client created by TGS to permit secure exchange between client and server without requiring them to share a permanent key.
ID_V	Confirms that this ticket is for server V.
TS_4	Informs client of time this ticket was issued.
$Ticket_V$	Ticket to be used by client to access server V.
$Ticket_{tgs}$	Reusable so that user does not have to reenter password.
K_{tgs}	Ticket is encrypted with key known only to AS and TGS, to prevent tampering.

Message (5)	Client requests service.
$Ticket_V$	Assures server that this user has been authenticated by AS.
$Authenticator_c$	Generated by client to validate ticket.
Message (6)	Optional authentication of server to client.
$K_{c,v}$	Assures C that this message is from V.
$TS_5 + 1$	Assures C that this is not a replay of an old reply.
$Ticket_v$	Reusable so that client does not need to request a new ticket from TGS for each access to the same server.
K_v	Ticket is encrypted with key known only to TGS and server, to prevent tampering.
$K_{c,v}$	Copy of session key accessible to client; used to decrypt authenticator, thereby authenticating ticket.
ID_C	Indicates the rightful owner of this ticket.
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket.
ID_V	Assures server that it has decrypted ticket properly.

KERBEROS REALMS:

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

1. The Kerberos server must have the user ID and hashed passwords of all participating users in its database. All users are registered with the Kerberos server.

2. The Kerberos server must share a secret key with each server. All servers are registered with the Kerberos server.

3. The Kerberos server in each interoperating realm shares a secret key with the server in the other realm. The two Kerberos servers are registered with each other.

A full-service Kerberos environment consisting of a Kerberos server, a number of

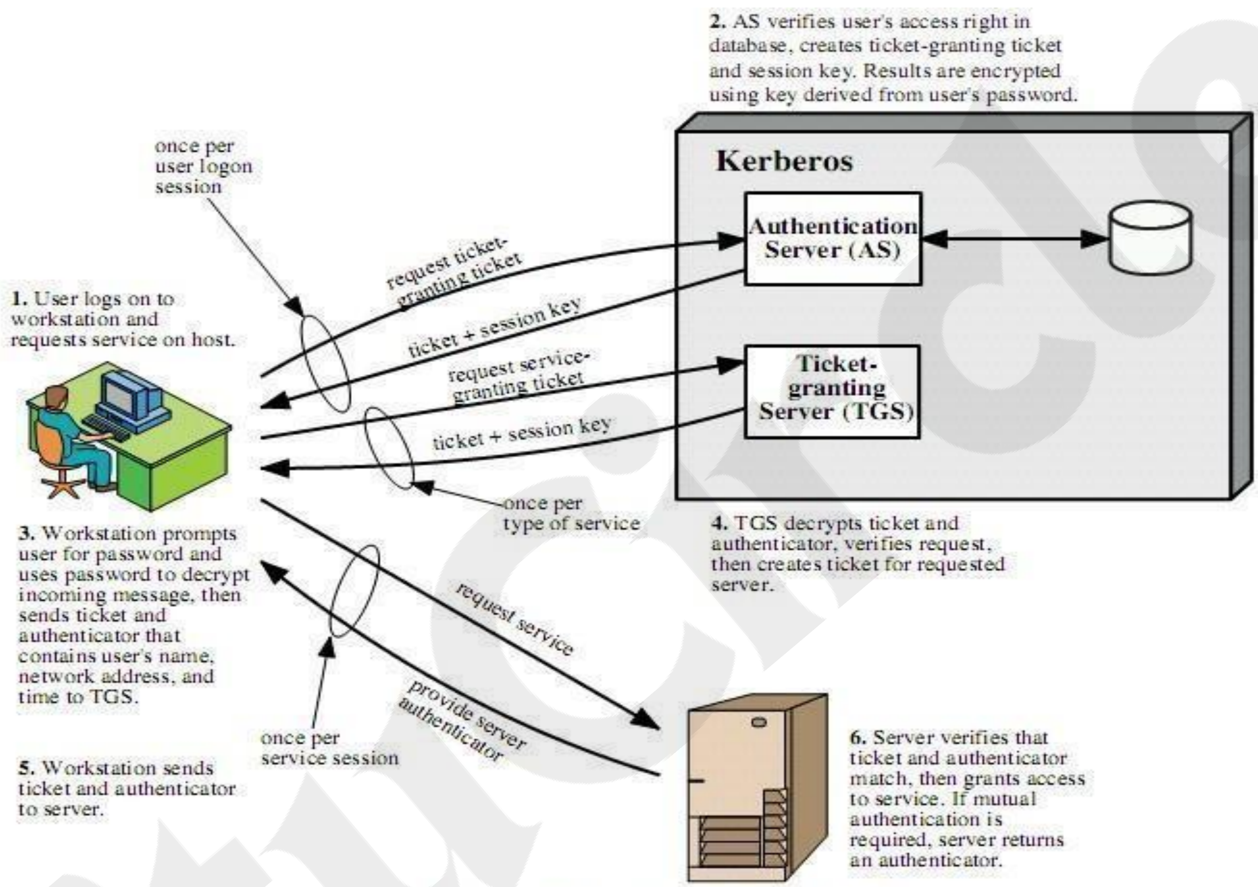


Figure 14.1 Overview of Kerberos

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

The details of the exchanges illustrated in below Figure are as follows

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

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

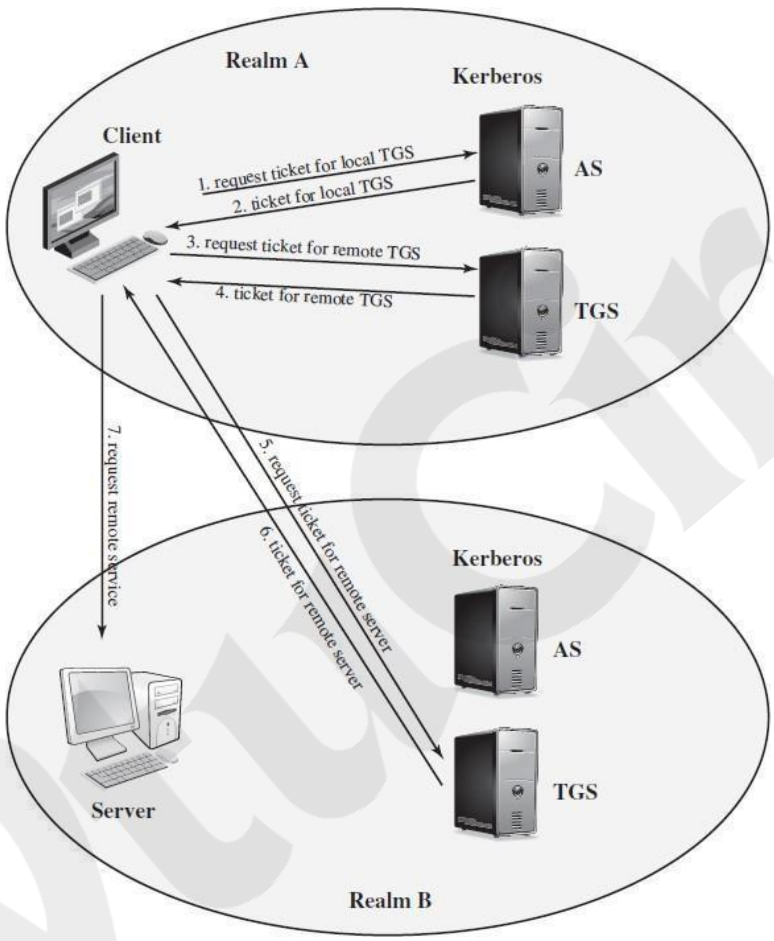
(3) $C \rightarrow TGS$: $ID_{tgsrem} \| Ticket_{tgs} \| Authenticator_c$

(4) $TGS \rightarrow C$: $E(K_{c,tgs}, [K_{c,tgsrem} \| ID_{tgsrem} \| TS_4 \| Ticket_{tgsrem}])$

(5) $C \rightarrow TGS_{rem}$: $ID_{vrem} \| Ticket_{tgsrem} \| Authenticator_c$

(6) $TGS_{rem} \rightarrow C$: $E(K_{c,tgsrem}, [K_{c,vrem} \| ID_{vrem} \| TS_6 \| Ticket_{vrem}])$

(7) $C \rightarrow V_{rem}$: $Ticket_{vrem} \| Authenticator_c$



The Version 5 Authentication Dialogue

Table 15.3 Summary of Kerberos Version 5 Message Exchanges

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

(2) **AS → C** $Realm_C \| ID_C \| Ticket_{tgs} \| \mathrm{E}(K_c, [K_{c,tgs} \| Times \| Nonce_1 \| Realm_{tgs} \| ID_{tgs}])$

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

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

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

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

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

$$Ticket_v = \mathrm{E}(K_v, [Flags \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times])$$

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

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

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

(6) **V → C** $\mathrm{E}_{K_{c,v}}[TS_2 \| Subkey \| Seq\ \#]$

$$Ticket_v = \mathrm{E}(K_v, [Flag \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times])$$

$$Authenticator_c = \mathrm{E}(K_{c,v}, [ID_C \| Relam_c \| TS_2 \| Subkey \| Seq\ \#])$$

(c) Client/Server Authentication Exchange to obtain service

INITIAL	This ticket was issued using the AS protocol and not issued based on a ticket-granting ticket.
PRE-AUTHENT	During initial authentication, the client was authenticated by the KDC before a ticket was issued.
HW-AUTHENT	The protocol employed for initial authentication required the use of hardware expected to be possessed solely by the named client.
RENEWABLE	Tells TGS that this ticket can be used to obtain a replacement ticket that expires at a later date.
MAY-POSTDATE	Tells TGS that a postdated ticket may be issued based on this ticket-granting ticket.
POSTDATED	Indicates that this ticket has been postdated; the end server can check the authtime field to see when the original authentication occurred.
INVALID	This ticket is invalid and must be validated by the KDC before use.
PROXIABLE	Tells TGS that a new service-granting ticket with a different network address may be issued based on the presented ticket.
PROXY	Indicates that this ticket is a proxy.
FORWARDABLE	Tells TGS that a new ticket-granting ticket with a different network address may be issued based on this ticket-granting ticket.
FORWARDED	Indicates that this ticket has either been forwarded or was issued based on authentication involving a forwarded ticket-granting ticket.

- Message (1) is a client request for a ticket-granting ticket. As before, it includes the ID of the user and the TGS. The following new elements are added:
  - Realm: Indicates realm of user
  - Options: Used to request that certain flags be set in the returned ticket
  - Times: Used by the client to request the following time settings in the ticket:

  —from: the desired start time for the requested ticket

  —till: the requested expiration time for the requested ticket

  —rtime: requested renew-till time
  - Nonce: A random value to be repeated in message (2) to assure that the response is fresh and has not been replayed by an opponent.
- Message (2) returns a ticket-granting ticket, identifying information for the client, and a block encrypted using the encryption key based on the user's password. This block includes the session key to be used between the client and the TGS, times specified in message (1), the nonce from message (1), and TGS identifying information. The ticket itself includes the session key, identifying information for the client, the requested time values, and flags that reflect the status of this ticket and the requested options. These flags introduce significant new functionality to version 5.
- Message (3) for both versions includes an authenticator, a ticket, and the name of the requested service. In addition, version 5 includes requested times and options for the ticket and a nonce— all with functions similar to those of message 1).
- Message (4) has the same structure as message (2). It returns a ticket plus information needed by the client, with the information encrypted using the session key now shared by the client and the TGS.
- Finally, for the client/server authentication exchange, several new features appear in version 5. In message (5), the client may request as an option that mutual authentication is required. The authenticator includes several new fields:
  - Subkey: The client's choice for an encryption key to be used to protect this specific application session. If this field is omitted, the session key from the ticket (Kc,v) is used.
  - Sequence number: An optional field that specifies the starting sequence number to be used by the server for messages sent to the client during this session. Messages may be sequence numbered to detect replays.

DIFFERENCES BETWEEN VERSIONS 4 AND 5

Version 5 is intended to address the limitations of version 4 in two areas: environmental shortcomings and technical deficiencies.

Environmental shortcomings.

1. **Encryption system dependence:** Version 4 requires the use of DES. Export restriction on DES as well as doubts about the strength of DES were thus of concern. In version 5, ciphertext is tagged with an encryption-type identifier so that any encryption technique may be used.

2. **Internet protocol dependence:** Version 4 requires the use of Internet Protocol (IP) addresses. Other address types, such as the ISO network address, are not accommodated. Version 5 network addresses are tagged with type and length, allowing any network address type to be used.

3. **Message byte ordering:** In version 4, the sender of a message employs a byte ordering of its own choosing and tags the message to indicate least significant byte in lowest address or most significant byte in lowest address. This technique works but does not follow established conventions. In version 5, all message structures are defined using Abstract Syntax Notation One (ASN.1) and Basic Encoding Rules (BER), which provide an unambiguous byte ordering.

4. **Ticket lifetime:** Lifetime values in version 4 are encoded in an 8-bit quantity in units of five minutes. Thus, the maximum lifetime that can be expressed is $2^8 * 5 = 1280$ minutes (a little over 21 hours).This may be inadequate for some applications. In version 5, tickets include an explicit start time and end time, allowing tickets with arbitrary lifetimes.

5. **Authentication forwarding:** Version 4 does not allow credentials issued to one client to be forwarded to some other host and used by some other client. This capability would enable a client to access a server and have that server access another server on behalf of the client. For example, a client issues a request to a print server that then accesses the client's file from a file server, using the client's credentials for access. Version 5 provides this capability.

6. **Interrealm authentication:** In version 4, interoperability among N realms requires on the order of N^2 Kerberos-to-Kerberos relationships, as described earlier. Version 5 supports a method that requires fewer relationships, as described shortly.

Technical deficiencies:

1. Double encryption: Note in Table 15.1 [messages (2) and (4)] that tickets provided to clients are encrypted twice - once with the secret key of the target server and then again with a secret key known to the client. The second encryption is not necessary and is computationally wasteful.

2. PCBC encryption: Encryption in version 4 makes use of a nonstandard mode of DES known as propagating cipher block chaining (PCBC).It has been demonstrated that this mode is vulnerable to an attack involving the interchange of ciphertext blocks. Version 5 provides explicit integrity mechanisms, a checksum or hash code is attached to the message prior to encryption using CBC.

3. Session keys: Each ticket includes a session key that is used by the client to encrypt the authenticator sent to the service associated with that ticket. In addition, the session key may subsequently be used by the client and the server to protect messages passed during that session. However, because the same ticket may be used repeatedly to gain service from a particular server, there is the risk that an opponent will replay messages from an old session to the client or the server. In version 5, it is possible for a client and server to negotiate a sub session key, which is to be used only for that one connection.

4. Password attacks: Both versions are vulnerable to a password attack. The message from the AS to the client includes material encrypted with a key based on the client's password. An opponent can capture this message and attempt to decrypt it by trying various passwords. If the result of a test decryption is of the proper form, then the opponent has discovered the client's password and may subsequently use it to gain authentication credentials from Kerberos. Version 5 does provide a mechanism known as pre authentication, which should make password attacks more difficult, but it does not prevent them.

REMOTE USER AUTHENTICATION USING ASYMMETRICENCRYPTION

Mutual Authentication

This protocol assumes that each of the two parties is in possession of the current public key of the other. It may not be practical to require this assumption.

1. A → AS: $ID_A \| ID_B$
2. AS → A: $E(PR_{as}, [ID_A \| PU_a \| T]) \| E(PR_{as}, [ID_B \| PU_b \| T])$
3. A → B: $E(PR_{as}, [ID_A \| PU_a \| T]) \| E(PR_{as}, [ID_B \| PU_b \| T]) \|$ $E(PU_b, E(PR_a, [K_s \| T]))$

A protocol using timestamps is provided in that uses a central system, referred to as an authentication server (AS), because it is not actually responsible for secret key distribution. Rather, the AS provides public-key certificates. The session key is chosen and encrypted by A; hence, there is no risk of exposure by the AS. The timestampsprotect against replays of compromised keys. See text for details. This protocol is compact but, as before, requires synchronization of clocks. Another approach, proposed by Woo and Lam, makes use of nonces.

1. A → KDC: $ID_A \| ID_B$
2. KDC → A: $E(PR_{auth}, [ID_B \| PU_b])$
3. A → B: $E(PU_b, [N_a \| ID_A])$
4. B → KDC: $ID_A \| ID_B \| E(PU_{auth}, N_a)$
5. KDC → B: $E(PR_{auth}, [ID_A \| PU_a]) \| E(PU_b, E(PR_{auth}, [N_a \| K_s \| ID_B]))$
6. B → A: $E(PU_a, [E(PR_{auth}, [(N_a \| K_s \| ID_B)]) \| N_b])$
7. A → B: $E(K_s, N_b)$

Note the authors themselves spotted a flaw in it and submitted a revised version of the algorithm in. In both this example and the protocols described earlier, protocols that appeared secure were revised after additional analysis. These examples highlight the difficulty of getting things right in the area of authentication.

One-Way Authentication

We have already presented public-key encryption approaches that are suited to electronic mail, including the straightforward encryption of the entire message for

confidentiality, authentication, or both. These approaches require that either the sender know the recipient's public key (confidentiality) or the recipient know the sender's public key (authentication) or both (confidentiality plus authentication).

If confidentiality is the primary concern, then better to encrypt the message with a one-time secret key, and also encrypt this one-time key with B's public key. If authentication is the primary concern, then a digital signature may suffice (but could be replaced by an opponent). To counter such a scheme, both the message and signature can be encrypted with the recipient's public key. The latter two schemes require that B know A's public key and be convinced that it is timely. An effective way to provide this assurance is the digital certificate.

Web Security Consideration

Web Security Overview

The **World Wide Web (WWW)** is primarily a **client/server application** running over the **Internet** and **TCP/IP-based intranets**. While general Internet security tools apply to Web environments, the **unique characteristics of Web usage** require **specialized security mechanisms**.

Although **Web browsers** are simple to use and **Web servers** easy to configure, the **underlying software is highly complex**, often hiding **security vulnerabilities**. Many newly installed systems have been found **vulnerable to attacks** despite being properly set up.

A compromised **Web server** can act as a **launching pad** for attackers to infiltrate the **entire organizational network**, accessing data and systems beyond the Web domain. Furthermore, **casual or untrained users** commonly access Web services without awareness of **security risks**, making them easy targets.

Web Security Threats

(Table 17.1) summarizes the major **Web security threats**, which can be grouped as follows:

Table 17.1 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 • 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 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

- **Passive attacks:**
  - **Eavesdropping** on network traffic between browser and server.
  - **Unauthorized access** to restricted Web information.
- **Active attacks:**
  - **Impersonation** of legitimate users.
  - **Modification of messages** during transmission.
  - **Alteration of Web content** on the server.

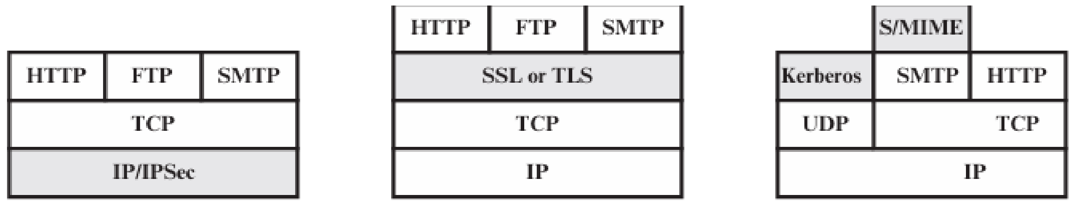
Threats can also be classified based on **where** they occur:

- **Web server threats**
- **Web browser threats**
- **Network traffic threats**

The first two categories relate to **computer system security**, while **traffic security** falls under **network security**, discussed in this chapter.

Web Traffic Security Approaches

Several approaches exist to secure Web traffic, differing mainly by their **placement in the TCP/IP stack** and **scope of protection** (Figure 17.1).



(a) Network level (b) Transport level (c) Application level

Figure 17.1 Relative Location of Security Facilities in the TCP/IP Protocol Stack

1. **Using IP Security (IPsec) — Figure 17.1(a):**
   - Operates at the **network layer**.
   - **Transparent to users and applications**.
   - Provides a **general-purpose** solution.
   - Includes **filtering** so only selected traffic is processed, reducing overhead.
2. **Using Security Above TCP — Figure 17.1(b):**
   - Example: **Secure Sockets Layer (SSL)** and **Transport Layer Security (TLS)**.
   - Can be implemented as:
     - A **protocol-level service** (transparent to applications).
     - **Embedded** within specific applications (e.g., browsers and servers).
   - Provides **encryption**, **authentication**, and **data integrity**.
3. **Application-Specific Security — Figure 17.1(c):**
   - Security features are **built directly into the application**.
   - Enables **customized protection** tailored to the application's needs.

Transport Layer Security (TLS)

Transport Layer Security (TLS)
One of the most widely used security services is **Transport Layer Security (TLS)**, currently at **Version 1.2** (RFC 5246). TLS evolved from **Secure Sockets Layer (SSL)**, which is now deprecated. TLS operates over **TCP** and can either be embedded in applications (like browsers and web servers) or provided as a transparent layer in the protocol stack.

TLS Architecture
TLS consists of two layers:

1. **TLS Record Protocol** – Provides basic **confidentiality** and **message integrity** to higher-layer protocols like HTTP.
2. **TLS-specific protocols** –
   - **Handshake Protocol**
   - **Change Cipher Spec Protocol**
   - **Alert Protocol**
   - **Heartbeat Protocol** (RFC 6250)

Key Concepts:

- **Connection:** A transient, peer-to-peer relationship providing transport services. Each connection is associated with one session.

- **Session:** An association between client and server that defines shared cryptographic parameters, avoiding repeated negotiation.

Session State:

- Session ID, Peer certificate, Compression method, Cipher spec, Master secret, Is resumable

Connection State:

- Server/client random, MAC secrets, write keys, IVs, sequence numbers

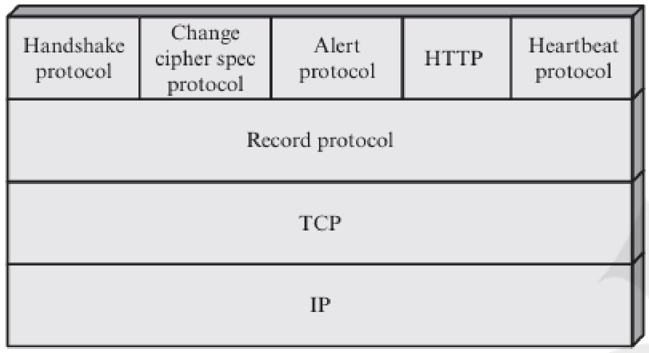


Figure 17.2 TLS Protocol Stack

TLS Record Protocol

The Record Protocol provides **confidentiality** and **message integrity**:

1. **Fragmentation:** Messages split into blocks ≤16,384 bytes.
2. **Compression:** Optional, lossless.

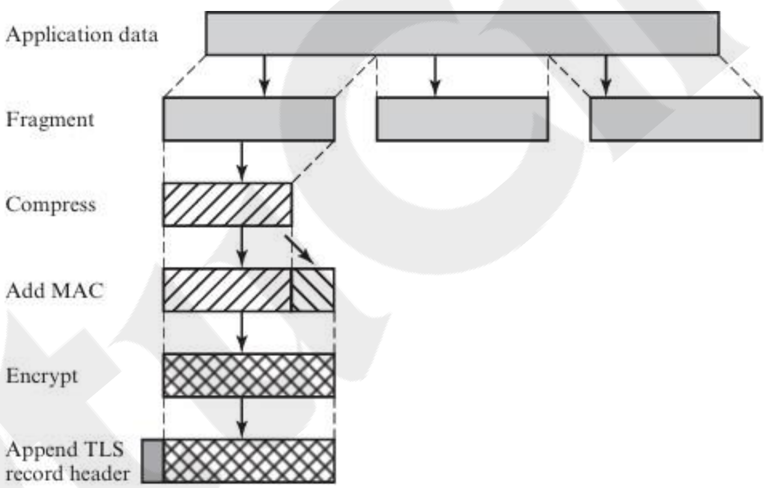


Figure 17.3 TLS Record Protocol Operation

3. **MAC computation:** Uses **HMAC** (MD5/SHA-1).
4. **Encryption:** Symmetric algorithms (AES, 3DES, RC4).
   - Stream ciphers encrypt compressed message + MAC
   - Block ciphers may include padding to match block length
5. **Header prepending:** Content Type, Major/Minor Version, Compressed Length
6. **Content types:** change_cipher_spec, alert, handshake, application_data (see Figure 17.4)

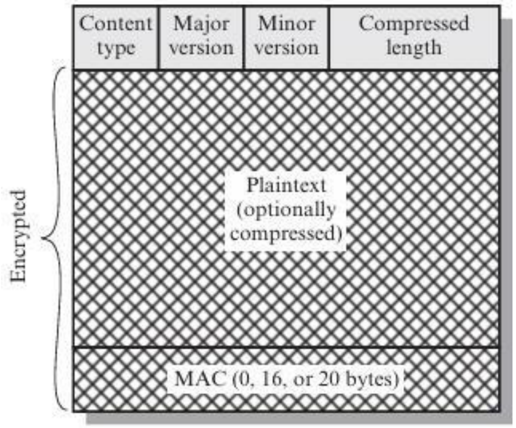


Figure 17.4 TLS Record Format

TLS-specific Protocols

1. Change Cipher Spec Protocol:

- Single-byte message (value 1) updates pending state to current state.

2. Alert Protocol:

- Two-byte messages indicating severity (warning or fatal) and alert code.
- Fatal alerts terminate the connection immediately (e.g., bad_record_mac, handshake_failure).
- Warnings include close_notify, bad_certificate, etc. (Figure 17.5b).

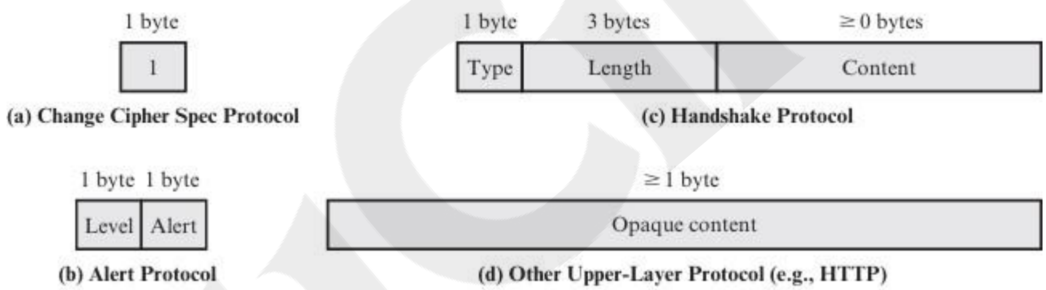


Figure 17.5 TLS Record Protocol Payload

3. Handshake Protocol:

- Authenticates client and server, negotiates encryption & MAC algorithms.
- Messages have **Type (1 byte), Length (3 bytes), Content** (see Table 17.2).

Table 17.2 TLS 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

- Four phases (Figure 17.6):

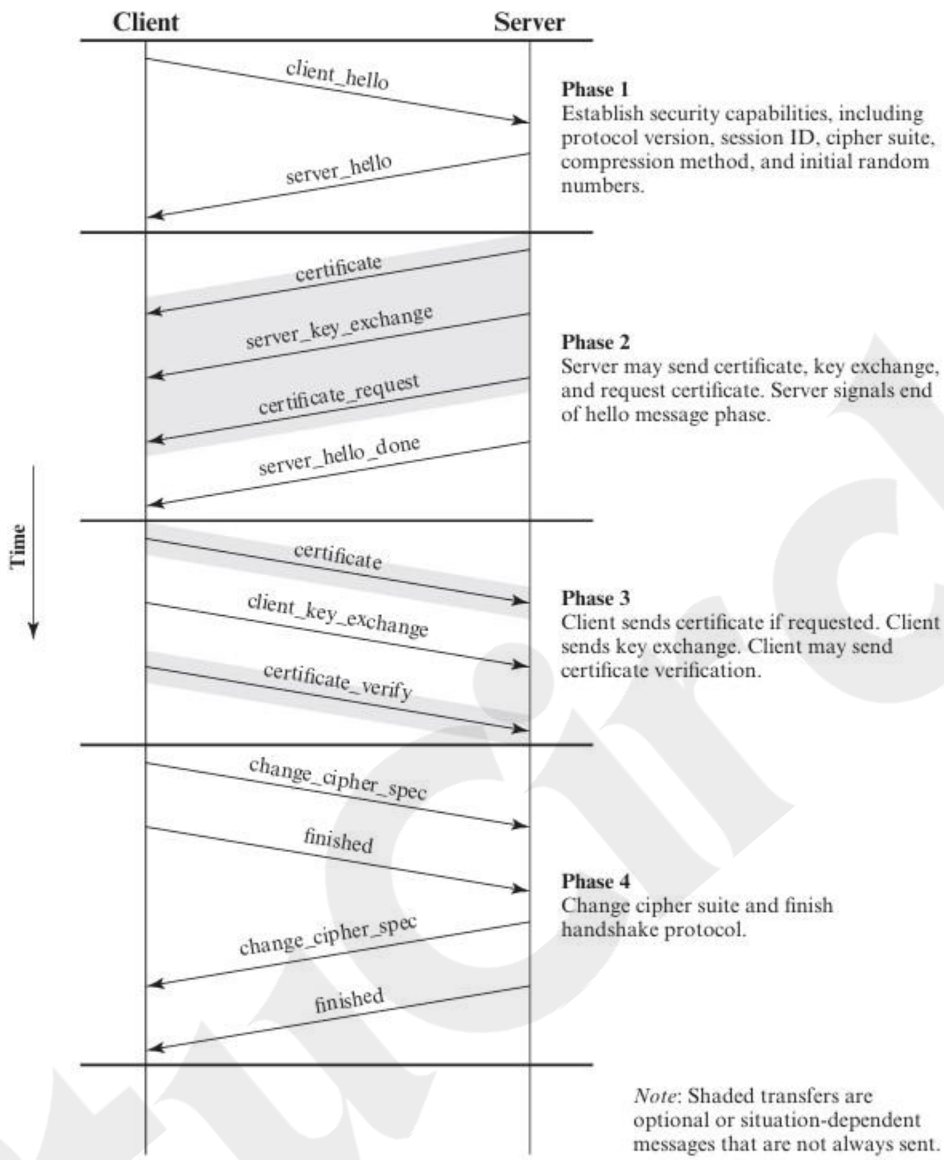


Figure 17.6 Handshake Protocol Action

1. **Establish Security Capabilities:** client_hello and server_hello exchanged. Includes version, random, session ID, CipherSuite, compression methods.
2. **Server Authentication & Key Exchange:** Server sends certificate, server_key_exchange, optional certificate request, and server_done. Supports RSA, Fixed/Ephemeral/Anonymous Diffie-Hellman.
3. **Client Authentication & Key Exchange:** Client sends certificate (if requested), client_key_exchange, and optional certificate_verify.
4. **Finish:** change_cipher_spec and finished messages confirm successful handshake.

Cryptographic Computations
Master Secret Creation:

- Shared 48-byte value computed using pre_master_secret via RSA or Diffie-Hellman.
- master_secret = PRF(pre_master_secret, "master secret", ClientHello.random ||

ServerHello.random)

Generation of Cryptographic Parameters:

- Keys (MAC, encryption) and IVs derived from master_secret using PRF.

Pseudorandom Function (PRF):

- Expands secrets into secure blocks for key generation/validation.
- PRF(secret, label, seed) = P_hash(secret, label || seed) using iterative HMAC (MD5/SHA-1).

Heartbeat Protocol (RFC 6250)

- Monitors availability and keeps TLS connections alive during idle periods.
- Messages: heartbeat request and heartbeat response.
- Payload between 16 bytes and 64 KB; echoed in response.

SSL/TLS Attacks

Categories:

1. **Handshake protocol attacks:** Exploit RSA formatting or implementation flaws.
2. **Record/application data attacks:** E.g., **BEAST** and **CRIME** attacks.
3. **PKI attacks:** Exploit certificate validation weaknesses.
4. **Other attacks:** DoS attacks exploiting TLS handshake computation.

TLSv1.3 Enhancements

- Removes unnecessary functions (compression, static RSA/DH, RC4, MD5/SHA-224).
- Uses **Diffie-Hellman/Elliptic Curve DH** for key exchange; RSA not permitted.
- Supports **1-round trip handshake**, improving efficiency and reducing attack surface.

ELECTRONIC MAIL SECURITY

- In virtually all distributed environments, electronic mail is the most heavily used network-based application.
- Users expect to be able to, and do, send e-mail to others who are connected directly or indirectly to the Internet, regardless of host operating system or communications suite.

Pretty Good Privacy

- ✓ 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 was developed by **Phil Zimmermann.**
- ✓ Zimmermann has done the following:

  1. Selected the best available **cryptographic algorithms** as building blocks.
  2. Integrated these algorithms into a general-purpose application that is independent of operating system and processor and that is based on a small set of easy-to-use commands.
  3. Made the package and its documentation, including the source code, freely available via the Internet, bulletin boards, and commercial networks such as AOL (America On Line).
  4. Entered into an agreement with a company (Via crypt, now Network Associates) to provide a fully compatible, low-cost commercial version of PGP.

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.
2. It is based on algorithms that are considered extremely secure such as **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.
5. PGP is now on an Internet standards track (RFC 3156; *MIME Security withOpenPGP*).

Notations

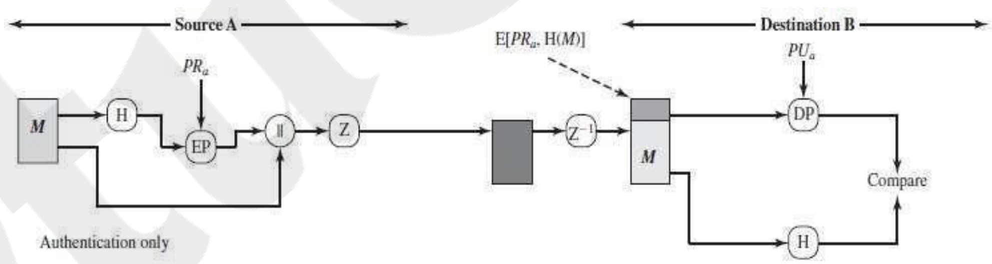
KS : Session key used in Symmetric Encryption Scheme
PRa : Private key of User A, used in Public Encryption
Scheme PUa : Public key of User A, used in Public Encryption
Scheme EP : Public Encryption
DP : Public Decryption
EC : Symmetric
Encryption DC :
Symmetric Decryption H
: Hash Function
|| : Concatenation
Z : Compression
R64 : Radix 64 conversion

Operational Description

The actual operation of PGP consists of **five series:**

- **Authentication**
- **Confidentiality**
- **Compression**
- **E-mail compatibility**
- **Segmentation**

1. Authentication



Authentication

- ✓ It is provided through the digital signatures. The sequence is as follows
  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.

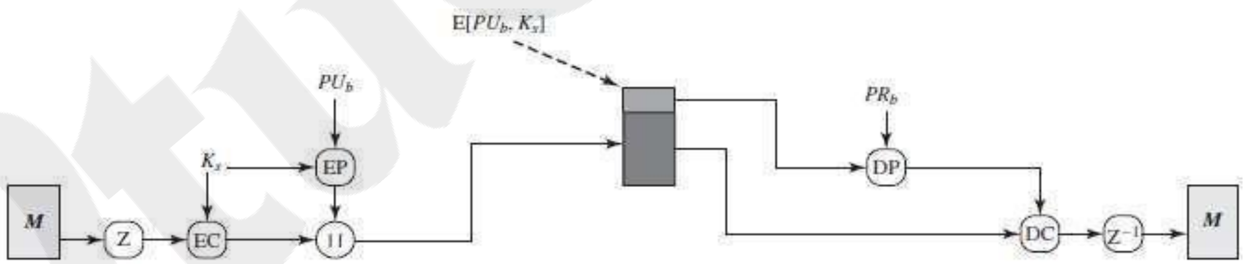
5. The receiver generates a **new hash code for the message and compares it with the decrypted hash code.** If the two match, the message is accepted as authentic.

- ✓ The combination of SHA-1 and RSA provides an effective digital signature scheme.
- ✓ Because of the strength of RSA, the recipient is assured that only the possessor of the matching private key can generate the signature.
- ✓ Because of the strength of SHA-1, the recipient is assured that no one else could generate a new message that matches the hash code and, hence, the signature of the original message
- ✓ Detached signatures are supported. A detached signature may be stored and transmitted separately from the message it signs.
- ✓ Finally, detached signatures can be used when more than one party must sign a document, such as a legal contract.

2. Confidentiality

It is provided by encrypting messages to be transmitted or to be stored locally as files. The sequence can be described as follows.

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.



(b) Confidentiality only

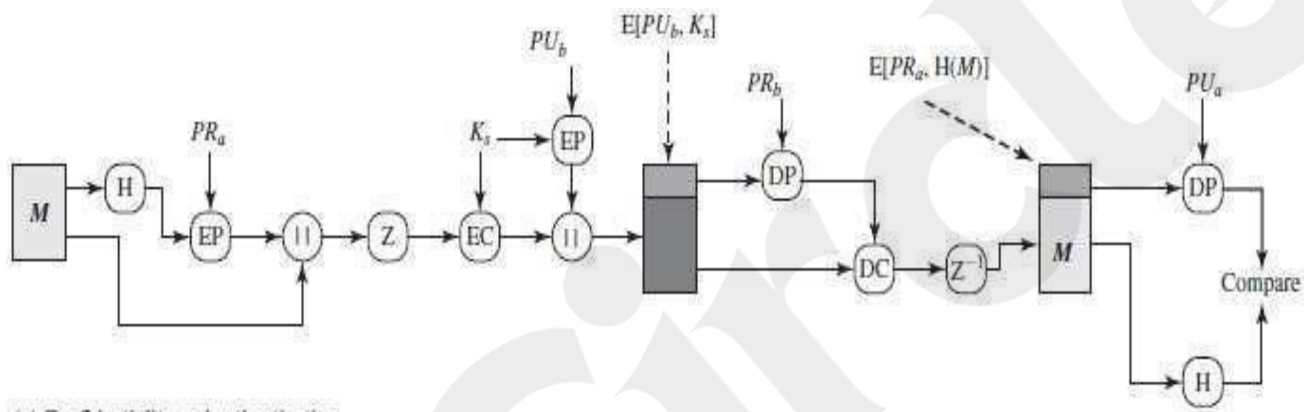
Confidentiality

- ✓ Instead of using the RSA also Diffie Hellman can be used.
- ✓ Diffie- Hellman is a key exchange algorithm.
- ✓ In fact, PGP uses a variant of Diffie-Hellman that does provide encryption/decryption, known as ElGamal.

3. Confidentiality and Authentication

1. **Here both services(**Confidentiality and Authentication)**may be used for the same message.**
2. **First, a signature is generated for the plaintext message and prepended to the message.**

3. Then the plaintext message plus signature is encrypted using CAST-128 (or IDEA or 3DES), and the session key is encrypted using RSA (or ElGamal).



(c) Confidentiality and authentication

Confidentiality and Authentication

4. Compression

- ✓ PGP compresses the message after applying the signature but before encryption .
- ✓ The signature is generated before compression for two reasons:
- ✓ It is preferable to sign an uncompressed message so that one can store only the uncompressed message together with the signature for future verification.
- ✓ If one signed a compressed document, then it would be necessary either to store a compressed version of the message for later verification or to recompress the message when verification is required.
- ✓ However, these different compression algorithms are interoperable because any version of the algorithm can correctly decompress the output of any other version.
- ✓ Applying the hash function and signature after compression would constrain all PGP implementations to the same version of the compression algorithm.

- ✓ 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. The compression algorithm used is ZIP

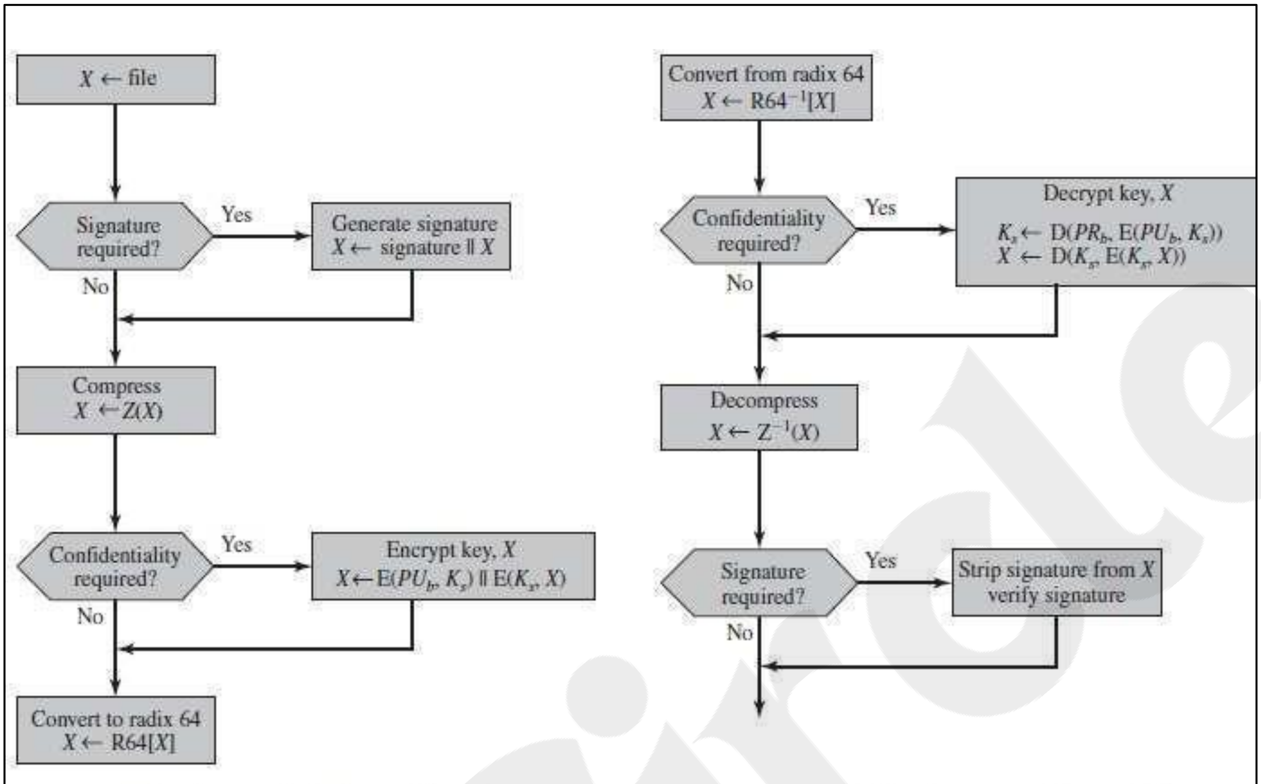
5. Email Compatibility

1. When PGP is used, at least part of the block to be transmitted is encrypted.
2. If only the **signature service is used, then the message digest(hash code) is encrypted** (with the sender's private key).
3. 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**.
4. PGP provides the service of **converting the raw 8-bit binary stream to a stream of printable ASCII characters.**
5. The scheme used for this purpose is **radix-64 conversion**.
6. Each group of **three octets of binary data is mapped into four ASCII characters**.
7. The **use of radix 64 expands a message by 33%.**
8. Each group of three octets of binary data is mapped into four ASCII characters.

6. Segmentation and Reassembly

1. Email facilities are often restricted to a **maximum message length.**
2. To accommodate this restriction, the PGP subdivides the message that is too large into segments that are small enough to send via email.
3. Thus the session key component, and signature component appear only once at the beginning of the first segment.
4. At the receiving end, PGP strips off all email headers and reassemble the entire original block of data.

The below Figure shows the relationship among **the four services**



(a) Generic transmission diagram (from A) (b) Generic reception diagram (to B)

Transmission and Reception of PGP Message

Transmission

- ✓ **On transmission** (if it is required), a signature is generated using a hash code of the uncompressed plaintext.
- ✓ Then the plaintext (plus signature if present) is compressed.
- ✓ Next, if confidentiality is required, the block (compressed plaintext or compressed signature plus plaintext) is encrypted and prepended with the public-key encrypted symmetric encryption key.
- ✓ Finally, the entire block is converted to radix-64 format.

Reception

- ✓ On reception, the incoming block is first converted back from radix-64 format to binary.
- ✓ Then, if the message is encrypted, the recipient recovers the session key and decrypts the message.
- ✓ The resulting block is then decompressed.
- ✓ If the message is signed, the recipient recovers the transmitted hash code and compares it to its own calculation of the hash code.

S/MIME (Security/Multipurpose Internet Mail Extension)

S/MIME is very similar to PGP. Both offer the ability to sign and/or encrypt messages.

- ✓ S/MIME (Secure/Multipurpose Internet Mail Extensions) is a security enhancement to the MIME internet e-mail format standard, based on RSA data security.
- ✓ S/MIME provides the following cryptographic security services for electronic messaging applications:
  - Authentication
  - Message integrity
  - Non-repudiation of origin using digital signatures
  - Data confidentiality using encryption

RFC 822

- ✓ RFC 822 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.
- ✓ In the RFC 822 context, messages are viewed as having an envelope and contents.
- ✓ The envelope contains whatever information is needed to accomplish transmission and delivery.
- ✓ The overall structure of a message that conforms to RFC 822 is very simple.
- ✓ A message consists of some number of header lines (the header) followed by unrestricted text (the body). The header is separated from the body by a blank line
- ✓ A header line consists of a keyword, followed by a colon, followed by the keyword's arguments; the format allows a long line to be broken up into several lines.
- ✓ The most frequently used keywords are From, To, Subject, and Date. Here is an example message.

Example:

Date: October 8, 2009 2:15:49 PM EDT
From: William Stallings <ws@shore.net>
Subject: The Syntax in RFC822
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 (MIME)

- ✓ Multipurpose Internet Mail Extension (MIME) is an extension to the RFC 822 framework that is intended to address some of the problems and limitations of the use of Simple Mail Transfer Protocol (SMTP).
- ✓ They are

  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, including the popular UNIX UUencode/Uudecode scheme. However, none of these is a standard or even a de facto standard.
  2. SMTP cannot transmit text data that includes national language characters, because these are represented by 8-bit codes with values of 128 decimal or higher, 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:
  - Deletion, addition, or reordering of carriage return and linefeed
  - Truncating or wrapping lines longer than 76 characters
  - Removal of trailing white space (tab and space characters)
  - Padding of lines in a message to the same length
  - Conversion of tab characters into multiple space characters.

OVERVIEW

The MIME specification includes the following elements

1. Five new message header fields are defined, which may be included in an RFC 822 header. These fields provide information about the body of the message.
2. A number of content formats are defined, thus standardizing representations that support multimedia electronic mail.
3. Transfer encodings are defined that enable the conversion of any content format into a form that is protected from alteration by the mail system.

(Explain the header fields of MIME protocol…)

The five header fields defined in MIME are

- **MIME-Version:** Must have the parameter value 1.0. This field indicates that the message conforms to RFCs 2045 and 2046.

- **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 or otherwise deal with the data in an appropriate manner.

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

(Explain **MIME** Content types ?)

MIME CONTENT TYPES:

- ✓ The bulk of the MIME specification is concerned with the definition of a variety of content types.
- ✓ There are seven different major types of content and a total of 15 subtypes. In general, a content type declares the general type of data, and the subtype specifies a particular format for that type of data.
- ✓ For the text type of body, no special software is required to get the full meaning of the text aside from support of the indicated character set.
- ✓ The primary subtype is plain text, which is simply a string of ASCII characters or ISO 8859 characters. The enriched subtype allows greater formatting flexibility.
- ✓ The multipart type indicates that the body contains multiple, independent parts.
- ✓ The Content-Type header field includes a parameter (called a boundary) that defines the delimiter between body parts.
- ✓ Each boundary starts on a new line and consists of two hyphens followed by the boundary value. The final boundary, which indicates the end of the last part, also has a suffix of two hyphens.
- ✓ Within each part, there may be an optional ordinary MIME header.

Example:

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
This is implicitly typed plain ASCII text. It does NOT end with a line break.
—simple boundary
Content-type: text/plain; charset=us-ascii
This is explicitly typed plain ASCII text. It DOES end with a linebreak.

—simple boundary—

Table 7.3 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.

✓ There are **four subtypes of the multipart type**, all of which have the same overall syntax.

1. The **multipart/mixed subtype** is used when there are multiple independent body parts that need to be bundled in a particular order.
2. **The multipart/parallel subtype**, the order of the parts is not significant. If the recipient's system is appropriate, the multiple parts can be presented in parallel.
3. For the **multipart/alternative subtype**, the various parts are different representations of the same information.
4. The **multipart/digest subtype** is used when each of the body parts is interpreted as an RFC 822 message with headers. This subtype enables the construction of a message whose parts are individual messages.

There are three sub types in **message type** in MIME.

1. The **message/rfc822 subtype:** indicates that the body is an entire message,

> **Example of multipart/alternative subtype:**
> From: Nathaniel Borenstein <nsb@bellcore.com>
> To: Ned Freed <ned@innosoft.com>
> Subject: Formatted text mail
> MIME-Version: 1.0
> Content-Type: multipart/alternative;
> boundary=boundary42
> —boundary42
> Content-Type: text/plain; charset=us-ascii
> ...plain text version of message goes here....
> —boundary42
> Content-Type: text/enriched
> .... RFC 1896 text/enriched version of same message
> goes here ...
> —boundary42—

including header and body.

2. **The message/partial subtype** enables fragmentation of a large message into a number of parts, which must be reassembled at the destination. For this subtype, three parameters are specified in the Content-Type: Message/Partial field: an id common to all fragments of the same message, a sequence number unique to each fragment, and the total number of fragments.
3. **The message/external-body subtype**: indicates that the actual data to be conveyed in this message are not contained in the body. Instead, the body contains the information needed to access the data

✓ **The application type** refers to other kinds of data, typically either uninterpreted binary data or information to be processed by a mail-based application. The quoted-printable transfer encoding is useful when the data consists largely of octets that correspond to printable ASCII characters.

(Explain MIME Transfer Encodings ?)

MIME TRANSFER ENCODINGS

✓ The other major component of the MIME specification, in addition to content type specification, is a definition of transfer encodings for message bodies.

✓ The Content-Transfer-Encoding field can actually take on **six values**, as listed in below Table However, three of these values (7bit, 8bit, and binary) indicate that no encoding has been done but provide some information about the nature of the data.

- ✓ For SMTP transfer, it is safe to use the 7bit form.
- ✓ The 8bit and binary forms may be usable in other mail transport contexts.
- ✓ x-token Encoding , which indicates that some other encoding scheme is used for which a name is to be supplied. This could be a vendor-specific or application-specific scheme.

- ✓ The **two actual encoding schemes defined are quoted-printable and base64.**
- ✓ Two schemes are defined to provide a choice between a transfer technique that is essentially human readable and one that is safe for all types of data in a way that is reasonably compact.
- ✓ The **quoted-printable** transfer encoding is useful when the data consists largely of **octets** that correspond to **printable ASCII characters.**
- ✓ In essence, it represents nonsafe characters by the hexadecimal representation of their code and introduces reversible (soft) line breaks to limit message lines to 76 characters.
- ✓ The **base64 transfer encoding,** also known as **radix-64 encoding,** is a common one for **encoding arbitrary binary data** in such a way as to be invulnerable to the processing by mail- transport programs.

Table 7.4 MIME Transfer Encodings

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.

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.

FUNCTIONS 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.

- **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.
- **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.

CRYPTOGRAPHIC ALGORITHM

- S/MIME uses the following terminology to specify 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.

- S/MIME incorporates three public-key algorithms.
  - ➢ The Digital Signature Standard (DSS) is the preferred algorithm for digital signature.
  - ➢ S/MIME lists Diffie-Hellman as the preferred algorithm for encrypting session keys
  - ➢ RSA, can be used for both signatures and session key encryption.

Table 7.6 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.

- For the hash function used to create the digital signature, the specification requires the 160-bit SHA-1 but recommends receiver support for the 128-bit MD5 for backward compatibility with older versions of S/MIME.
- For message encryption, three-key triple DES (tripleDES) is recommended, but compliant implementations must support 40-bit RC2.
- The following rules, in the following order, should be followed by a sending agent.
    1. If the sending agent has a list of preferred decrypting capabilities from an intended recipient, it SHOULD choose the first (highest preference) capability on the list that it is capable of using.
    2. If the sending agent has no such list of capabilities from an intended recipient but has received one or more messages from the recipient, then the outgoing message SHOULD use the same encryption algorithm as was used on the last signed and encrypted message received from that intended recipient.
    3. If the sending agent has no knowledge about the decryption capabilities of the intended recipient and is willing to risk that the recipient may not be able to decrypt the message, then the sending agent SHOULD use triple DES.
    4. If the sending agent has no knowledge about the decryption capabilities of the intended recipient and is not willing to risk that the recipient may not be able to decrypt the message, then the sending agent MUST use RC2/40.

S/MIME Messages
(Explain S/MIME content types)

- ✓ S/MIME makes use of a number of new MIME content types shown in table.
- ✓ All of the new application types use the designation PKCS. This refers to a set of public- key cryptography specifications issued by RSA Laboratories.

Table 7.7 S/MIME Content Types

Type	Subtype	smime Parameter	Description
Multipart	Signed		A clear-signed message in two parts: one is the message and the other is the signature.
Application	pkcs7-mime	signedData	A signed S/MIME entity.
	pkcs7-mime	envelopedData	An encrypted S/MIME entity.
	pkcs7-mime	degenerate signedData	An entity containing only public-key certificates.
	pkcs7-mime	CompressedData	A compressed S/MIME entity.
	pkcs7-signature	signedData	The content type of the signature subpart of a multipart/signed message.

Securing a MIME entity:

- ✓ **S/MIME** secures a MIME entity with a signature encryption, or both. A MIME entity may be an entire message , or if the MIME content type is multipart, then a MIME entity is one or more of the subparts of the message.

- ✓ Then the MIME entity plus some security-related data, such as algorithm identifiers and certificates, are processed by S/MIME to produce what is known as a PKCS object.
- ✓ A PKCS object is then treated as message content and wrapped in MIME. In all cases, the message to be sent is converted to canonical form.

Enveloped Data

- ✓ An application/pkcs7-mime subtype is used for one of four categories of S/MIME processing, each with a unique s/mime-type parameter.
- ✓ The steps for preparing an enveloped Data MIME entity are:

1. Generate a pseudorandom session key for a particular symmetric encryption algorithm (RC2/40 or triple DES).
2. For each recipient, encrypt the session key with the recipient's public RSA key.
3. For each recipient, prepare a block known as **Recipient Info** that contains an identifier of the recipient's public-key certificate, an identifier of the algorithm used to encrypt the session key, and the encrypted session key.
4. Encrypt the message content with the session key.

- ✓ The **Recipient Info** blocks followed by the encrypted content constitute the enveloped Data.

Signed data

- ✓ The signed Data s/mime-type can be used with one or more signers. For clarity, we confine our description to the case of a single digital signature.

The steps for preparing a signed Data MIME entity are:

1. Select a message digest algorithm (SHA or MD5).
2. Compute the message digest (hash function) of the content to be signed.
3. Encrypt the message digest with the signer's private key.
4. Prepare a block known as Signer Info that contains the signer's public key certificate, an identifier of the message digest algorithm, an identifier of the algorithm used to encrypt the message digest, and the encrypted message digest.

- ✓ The signed Data entity consists of a series of blocks, including a message digest algorithm identifier, the message being signed, and Signer Info.

Clear Signing

- ✓ Clear signing is achieved using the multipart content type with a signed subtype.
- ✓ A multipart/signed message has two parts. The first part can be any MIME type but must be prepared so that it will not be altered during transfer from source to destination.
- ✓ This second part has a MIME content type of application and a subtype of pkcs7- signature.

Registration Request

An application or user will apply to a certification authority for a public-key certificate. The application/pkcs10 S/MIME entity is used to transfer a certification request.

CERTIFICATES-ONLY MESSAGE

A message containing only certificates or a certificate revocation list (CRL) can be sent in response to a registration request. The message is an application/pkcs7-mime type/subtype with an s/mime-type parameter of degenerate.

S/MIME Certificate Processing

S/MIME uses public-key certificates that confirm to version 3 of X.509.

USER AGENT ROLE:

An S/MIME user has several key-management functions to perform.

- **Key generation**: The user of some related administrative utility MUST be capable of generating separate Diffie-Hellman and DSS key pairs and SHOULD be capable of generating RSA key pairs. Each key pair MUST be generated from a good source of a non-deterministic random input. A user agent SHOULD generate RSA key pairs with a length in the range of 768 – 1024 bits and MUST NOT generate less than 512 bits.
- **Registration**: A user's public key must be registered with a certification authority in order to receive an X.509 public-key certificate.
- **Certificate storage and retrieval**: A user requires access to a local list of certificates in order to verify incoming signatures and to encrypt outgoing messages.

VeriSign Certificates

- ✓ There are several companies that provide certification authority (CA) services. VeriSign provides a CA service that is intended to be compatible with S/MIME and a variety of other applications.
- ✓ VeriSign issues X.509 certificates with the product name VeriSign Digital ID.
- ✓ The information contained in a Digital ID depends on the type of Digital ID and its use.
- ✓ At a minimum, each Digital ID contains:
  - Owner's public key
  - Owner's name or alias
  - Expiration date of the Digital ID
  - Serial number of the Digital ID
  - Name of the certification authority that issued the Digital ID
  - Digital signature of the certification authority that issued the Digital ID.
- ✓ Digital IDs can also contain user supplied information :
  - Address

- Email address
- Basic registration information (Country ZIP code, age and gender)

- ✓ VeriSign provides three levels, or classes, of security for public-key certificates.
  - For Class 1 Digital IDs, VeriSign confirms the user's e-mail address by sending a PIN and Digital ID pick-up information to the e-mail address provided in the application.
  - For Class 2 Digital IDs, VeriSign verifies the information in the application through an automated comparison with a consumer database in addition to performing all of the checking associated with a Class 1 Digital ID.
  - For Class 3 Digital IDs, VeriSign requires a higher level of identity assurance.

Enhanced Security Services:

The three services are:

- **Signed receipts**: A signed receipt may be requested in a Signed Data object. e. In essence, the recipient signs the entire original message plus the original (sender's) signature and appends the new signature to form a new S/MIME messages.
- **Security labels:** A security label may be included in the authenticated attributes of a Signed Data object. The labels may be used for access control, by indicating which users are permitted access to an object.
- **Secure mailing lists**: When a user sends a message to multiple recipients, a certain amount of per-recipient processing is required, including the use of each recipient's public key. The user can be relieved of this work by employing the services of an S/MIME Mail List Agent (MLA). An MLA can take a single incoming message, perform the recipient-specific encryption for each recipient, and forward the message. The originator of a message need only send the message to the MLA with encryption performed using the MLA's public key.