

Unit-3

Authentication Applications:

Kerberos :

Kerberos4 is an authentication service developed as part of Project Athena at MIT. The problem that Kerberos addresses is this: Assume an open distributed environment in which users at workstations wish to access services on servers distributed throughout the network. We would like for servers to be able to restrict access to authorized users and to be able to authenticate requests for service. In this environment, a workstation cannot be trusted to identify its users correctly to network services.

In particular, the following three threats exist:

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

Three approaches to security can be envisioned.

1. Rely on each individual client workstation to assure the identity of its user or users and rely on each server to enforce a security policy based on user identification (ID).
2. Require that client systems authenticate themselves to servers, but trust the client system concerning the identity of its user.
3. Require the user to prove his or her identity for each service invoked. Also require that servers prove their identity to clients.

The first published report on Kerberos [STEI88] listed the following requirements.

- Secure
- Reliable
- Transparent

- Scalable

Kerberos Version 4:

Version 4 of Kerberos makes use of DES, in a rather elaborate protocol, to provide the authentication service. Viewing the protocol as a whole, it is difficult to see the need for the many elements contained therein. Therefore, we adopt a strategy used by Bill Bryant of Project Athena [BRYA88] and build up to the full protocol by looking first at several hypothetical dialogues.

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

To solve these additional problems, we introduce a scheme for avoiding plaintext passwords and a new server, known as the ticket-granting server (TGS). The new (but still hypothetical) scenario is

as follows.

Once per user logon session:

- (1) $C \rightarrow AS: ID_C \parallel ID_{tgs}$
- (2) $AS \rightarrow C: E(K_c, Ticket_{tgs})$

Once per type of service:

- (3) $C \rightarrow TGS: ID_C \parallel ID_V \parallel Ticket_{tgs}$
- (4) $TGS \rightarrow C: Ticket_v$

Once per service session:

- (5) $C \rightarrow V: ID_C \parallel Ticket_v$

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

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 t is successfully recovered.

Now that the client has a ticket-granting ticket, access to any server can be obtained with steps 3 and 4.

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 () 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.

Finally, with a particular service-granting ticket, the client can gain access to the corresponding service with step 5.

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.

Table 15.1 Summary of Kerberos Version 4 Message Exchanges

(1) $C \rightarrow AS$ $ID_C \parallel ID_{TGS} \parallel TS_1$

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

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

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

(3) $C \rightarrow TGS$ $ID_V \parallel Ticket_{TGS} \parallel Authenticator_C$

(4) $TGS \rightarrow C$ $E(K_{c, tgs}, [K_{c, v} \parallel ID_V \parallel TS_4 \parallel Ticket_V])$

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

$$Ticket_V = E(K_{v, tgs}, [K_{c, v} \parallel ID_C \parallel AD_C \parallel ID_V \parallel TS_4 \parallel Lifetime_4])$$

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

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

(5) $C \rightarrow V$ $Ticket_V \parallel Authenticator_C$

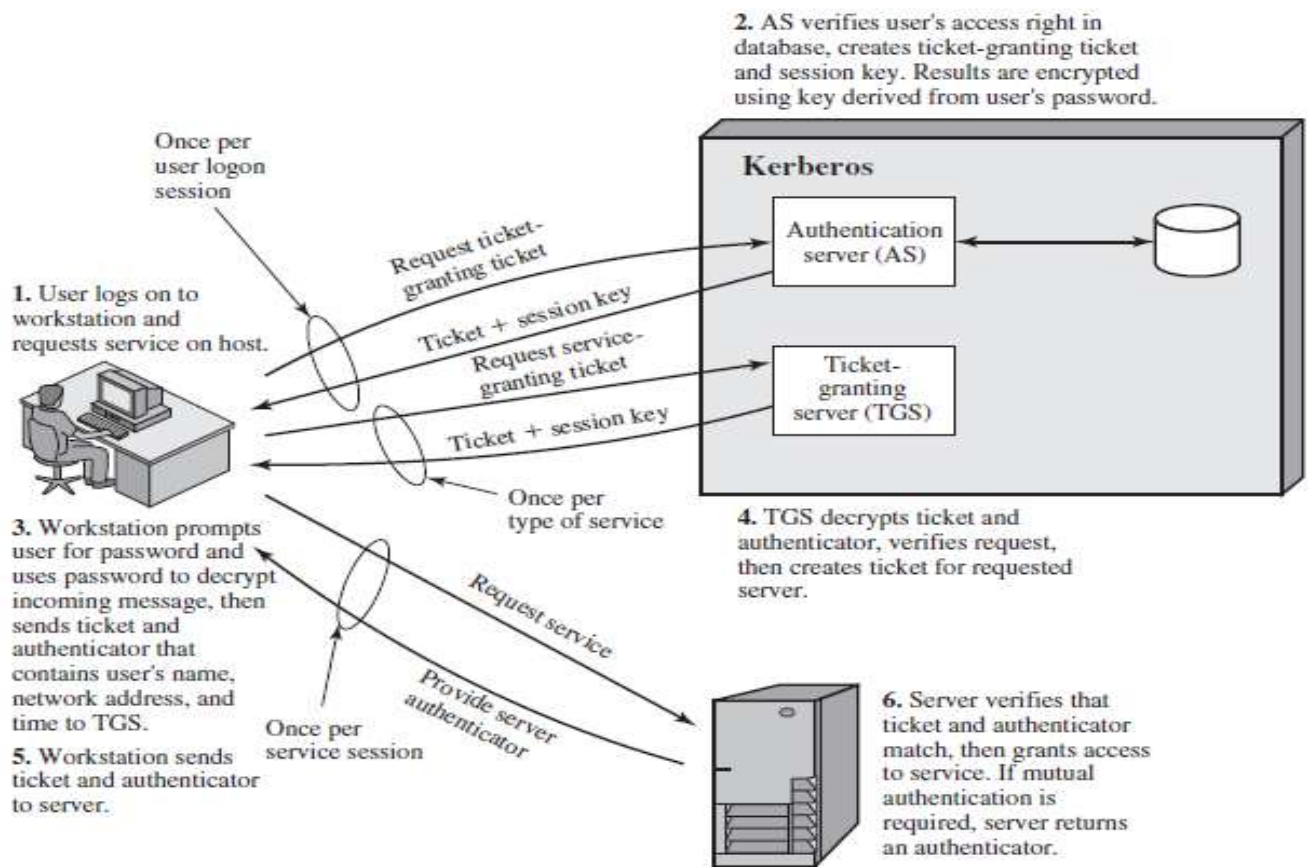
(6) $V \rightarrow C$ $E(K_{c, v}, [TS_5 + 1])$ (for mutual authentication)

$$Ticket_V = E(K_{v, tgs}, [K_{c, v} \parallel ID_C \parallel AD_C \parallel ID_V \parallel TS_4 \parallel Lifetime_4])$$

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

(c) Client/Server Authentication Exchange to obtain service

Overview of Kerberos



X.509 directory authorization service

X.509 is a digital certificate that is built on top of a widely trusted standard known as ITU or International Telecommunication Union X.509 standard, in which the format of PKI certificates is defined. X.509 digital certificate is a certificate-based authentication security framework that can be used for providing secure transaction processing and private information. These are primarily used for handling the security and identity in computer networking and internet-based communications. The heart of the X.509 scheme is the public-key certificate associated with each user. These user certificates are assumed to be created by some trusted **certification authority (CA)** and placed in the directory by the CA or by the user. These directory servers are only used for providing an effortless reachable location for all users so that they can acquire certificates. X.509 standard is built on an IDL known as ASN.1. With the help of Abstract Syntax Notation, **the X.509 certificate format uses an associated public and private key pair for encrypting and decrypting a message.** Once an X.509 certificate is provided to a user by the certified authority, that certificate is attached

to it like an identity card. The chances of someone stealing it or losing it are less, unlike other unsecured passwords. With the help of this analogy, it is easier to imagine how this authentication works: the certificate is basically presented like an identity at the resource that requires authentication.

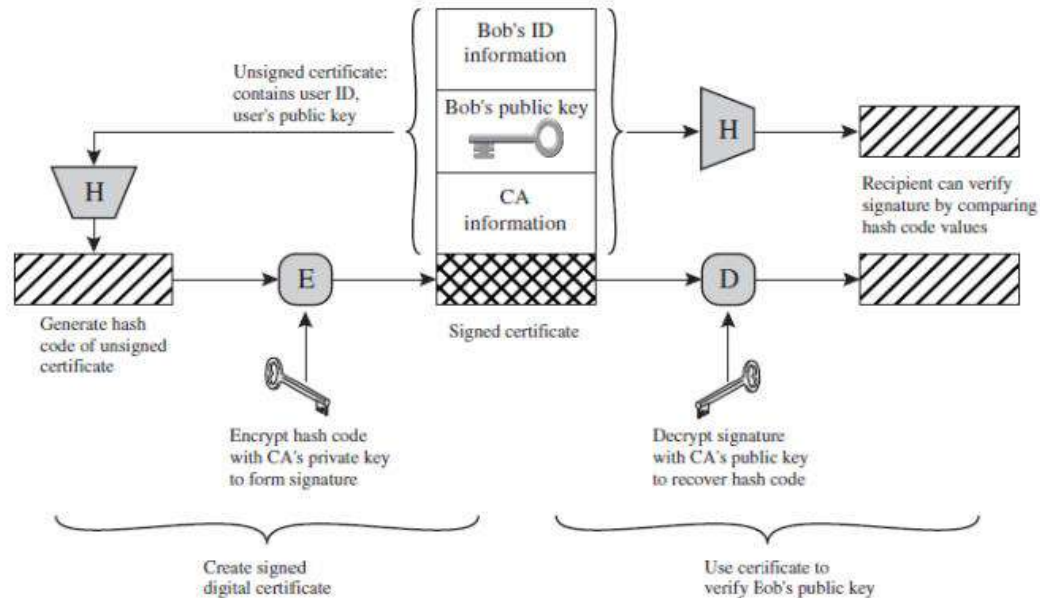


Fig: Public-key Certificate Use

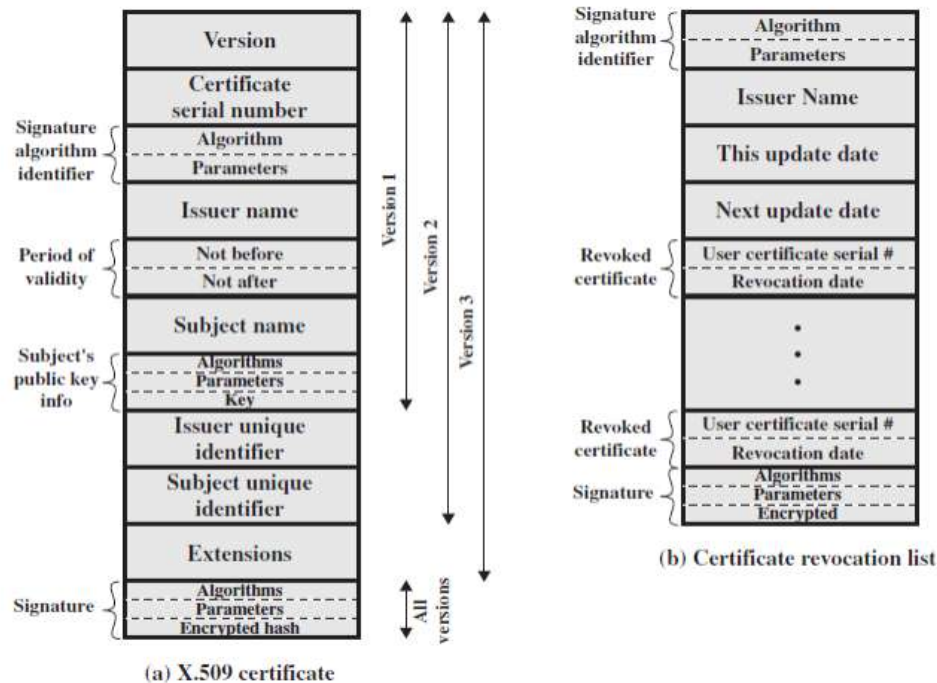


Fig: X.509 Formats

Generally, the certificate includes the below elements:

Version: Differentiates among successive versions of the certificate format; the default is version

1. If the issuer unique identifier or subject unique identifier are present, the value must be version
2. If one or more extensions are present, the version must be version3.

Serial number: An integer value unique within the issuing CA that is unambiguously associated with this certificate.

Signature algorithm identifier: The algorithm used to sign the certificate together with any associated parameters. Because this information is repeated in the signature field at the end of the certificate, this field has little, if any, utility.

Issuer name: X.500 is the name of the CA that created and signed this certificate.

Period of validity: Consists of two dates: the first and last on which the certificate is valid.

Subject name: The name of the user to whom this certificate refers. That is, this certificate certifies the public key of the subject who holds the corresponding private key.

Subject's public-key information: The public key of the subject, plus an identifier of the algorithm for which this key is to be used, together with any associated parameters.



Issuer unique identifier: An optional-bit string field used to identify uniquely the issuing CA in the event the X.500 name has been reused for different entities.

Subject unique identifier: An optional-bit string field used to identify uniquely the subject in the event the X.500 name has been reused for different entities.

Extensions: A set of one or more extension fields. Extensions were added in version 3 and are discussed later in this section.

Signature: Covers all of the other fields of the certificate; it contains the hash code of the other fields encrypted with the CA's private key. This field includes the signature algorithm identifier. The unique identifier fields were added in version 2 to handle the possible reuse of subject and/or issuer names over time. These fields are rarely used.

Example :

Certificate format version			version 3
Certificate serial number			12345678
Signature algorithm identifier for CA			RSA with MD5
Issuer X.500 name			c=US, o=ACME
Validity period			start=01/08/96, expiry=01/08/98
Subject X.500 name			c=US, o=ACME, cn=John Smith + ...
Subject public key information			 RSA with MD5
Issuer unique identifier			version 2
Subject unique identifier			version 2
version 3	Type	Criticality	Value
version 3	Type	Criticality	Value
version 3	...		
version 3	Type	Criticality	Value
CA Signature 			

Extensions

The standard uses the following notation to define a certificate:

$CA\langle\rangle = CA \{V, SN, AI, CA, UCA, A, UA, Ap, TA\}$

where $Y\langle\rangle =$ the certificate of user X issued by certification authority Y

$Y\{I\} =$ the signing of I by Y.

It consists of I with an encrypted hash code appended

V = version of the certificate

SN = serial number of the certificate

AI = identifier of the algorithm used to sign the certificate

CA = name of certificate authority

UCA = optional unique identifier of the CA

A = name of user A

UA = optional unique identifier of the user A

AP = public key of user A

TA = period of validity of the certificate

The CA signs the certificate with its private key. If the corresponding public key is known to a user, then that user can verify that a certificate signed by the CA is valid

Pretty Good Privacy (PGP) :

PGP is a remarkable phenomenon. Largely the effort of a single person, Phil Zimmermann, PGP provides a confidentiality and authentication service that can be used for electronic mail and file storage applications. In essence, 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 (Viacrypt, now Network Associates) to provide a fully compatible, low-cost commercial version of PGP.

PGP has grown explosively and is now widely used. A number of reasons can be cited for this growth

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 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.

3. 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.
4. PGP is now on an Internet standards track (RFC 3156; MIME Security with OpenPGP). Nevertheless, PGP still has an aura of an antiestablishment endeavor.

We begin with an overall look at the operation of PGP. Next, we examine how cryptographic keys are created and stored. Then, we address the vital issue of public-key management.

Notation

Most of the notation used in this chapter has been used before, but a few terms are new. It is perhaps best to summarize those at the beginning. The following symbols are used.

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

The PGP documentation often uses the term *secret key* to refer to a key paired with a public key in a public-key encryption scheme. As was mentioned earlier, this practice risks confusion with a secret key used for symmetric encryption. Hence, we use the term *private key* instead.

S/MIME :

Secure/Multipurpose Internet Mail Extension (S/MIME) is a security enhancement to the MIME Internet e-mail format standard based on technology from RSA Data Security.

Multipurpose Internet Mail Extension (MIME) is a standard that was proposed by Bell Communications in 1991 in order to expand the limited capabilities of email. MIME is a kind of add-on *or a supplementary protocol* that allows non-ASCII data to be sent through SMTP. It allows the users to exchange different kinds of data files on the Internet: audio, video, images, application programs as well.

Why do we need MIME?

Limitations of Simple Mail Transfer Protocol (SMTP):

1. SMTP has a very simple structure
2. Its simplicity however comes with a price as it only sends messages in NVT 7-bit ASCII format.
3. It cannot be used for languages that do not support 7-bit ASCII format such as French, German, Russian, Chinese and Japanese, etc. so it cannot be transmitted using SMTP. So, in order *to make SMTP more broad, we use MIME*.
4. It cannot be used to send binary files or video or audio data.

Features of MIME :

1. It is able to send multiple attachments with a single message.
2. Unlimited message length.
3. Binary attachments (executables, images, audio, or video files) may be divided if needed.
4. MIME provided support for varying content types and multi-part messages.

Transport level Security, Web Security Considerations:

Web Security Threats :

Web security is also known as “Cybersecurity”. It basically means protecting a **website** or **web** application by detecting, preventing and responding to cyber threats. Websites and **web** applications are just as prone to **security** breaches as physical homes, stores, and government locations.

WEB SECURITY :

- Measures to protect data during their transmission over a collection of interconnected networks.
- The World Wide Web is fundamentally a client/server application running over the internet and TCP/IP intranets.



Web security Requirement

- The web is very visible.
- The WWW is widely used by:-
- Business, Government agencies and many individuals.
- These can be described as passive attacks including eavesdropping on network traffic between browser and gaining access to information on a website that is supposed to be restricted.
- Active attacks including impersonating another user, altering information on a website.
- The web needs added security mechanisms to address these threats .

Table 16.1 provides a summary of the types of security threats faced when using the Web.

Table 16.1 A Comparison of Threats on the Web

	Threats	Consequences	Countermeasures
Integrity	<ul style="list-style-type: none"> • Modification of user data • Trojan horse browser • Modification of memory • Modification of message traffic in transit 	<ul style="list-style-type: none"> • Loss of information • Compromise of machine • Vulnerability to all other threats 	Cryptographic checksums
Confidentiality	<ul style="list-style-type: none"> • 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 	<ul style="list-style-type: none"> • Loss of information • Loss of privacy 	Encryption, Web proxies
Denial of Service	<ul style="list-style-type: none"> • Killing of user threads • Flooding machine with bogus requests • Filling up disk or memory • Isolating machine by DNS attacks 	<ul style="list-style-type: none"> • Disruptive • Annoying • Prevent user from getting work done 	Difficult to prevent
Authentication	<ul style="list-style-type: none"> • Impersonation of legitimate users • Data forgery 	<ul style="list-style-type: none"> • Misrepresentation of user • Belief that false information is valid 	Cryptographic techniques

Web Traffic Security Approaches :

- A number of approaches to providing Web security are possible.
- One way to provide Web security is to use IP security (IPsec). The advantage of using IPsec is that it is transparent to end users and applications and provides a general-purpose solution.
- Furthermore, IPsec includes a filtering capability so that only selected traffic need incur the overhead of IPsec processing.
- Figure 16.1c shows examples of this architecture. The advantage of this approach is that the service can be tailored to the specific needs of a given application.

A number of approaches to providing web security are possible figure illustrate this difference.

1. Network Level.
2. Transport Level.
3. Application Level.

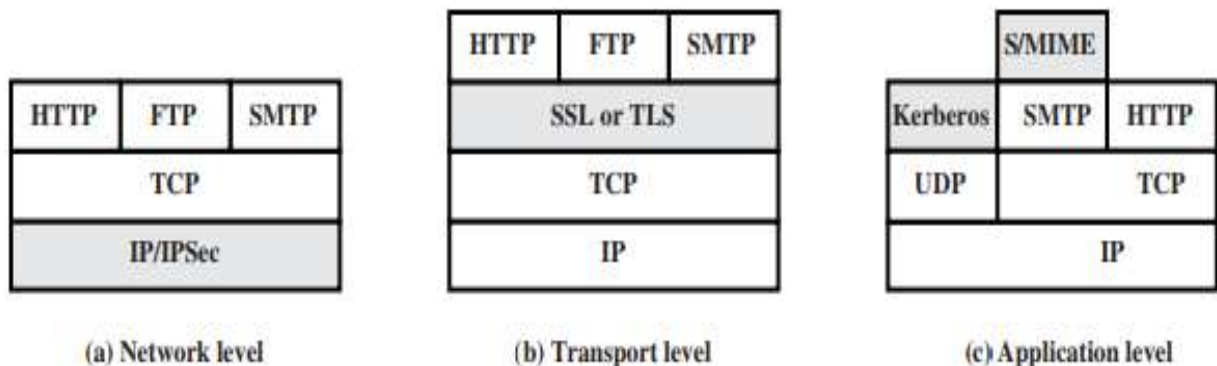


Figure 16.1 Relative Location of Security Facilities in the TCP/IP Protocol Stack

- ✓ **IP/IPSec** : The functional areas general IP Security mechanisms provides
- ✓ Authentication: This provides source authentication.
- ✓ Confidentiality: encryption
- ✓ key management: transfer of keys securely.
- ✓ applicable to use over LANs, across public & private WANs, & for the Internet.

SSL Architecture, SSL Record Protocol, Change Cipher Spec Protocol, Alert Protocol :

- ✓ 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 However, it is feasible to prevent the success of these attacks, usually by means of encryption. Thus, the emphasis in dealing with passive attacks is on prevention rather than detection.
- ✓ The SSL Record Protocol provides basic security services to various higher layer protocols. In particular, the Hypertext Transfer Protocol

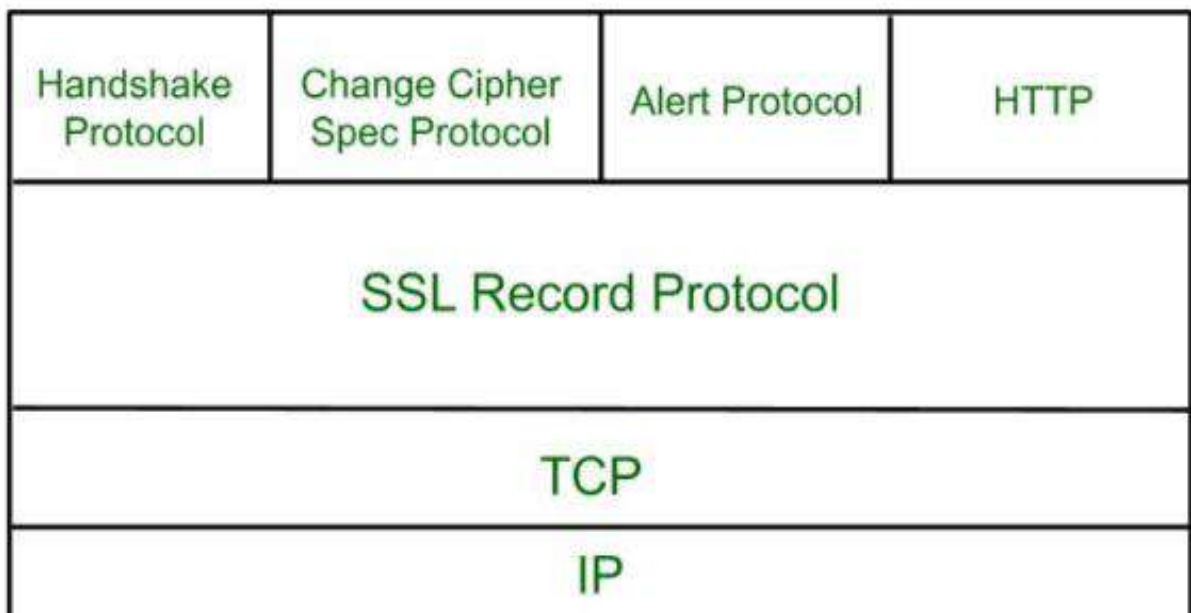
(HTTP), which provides the transfer service for Web client/server interaction, can operate on top of SSL.

- ✓ Three higher-layer protocols are defined as part of SSL:
 - SSL Record Protocol
 - The Handshake Protocol.
 - The Change Cipher Spec Protocol
 - The Alert Protocol.

Two important SSL concepts are the SSL session and the SSL connection, which are defined in the specification as follows.

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

SSL Protocol Stack:



SSL Record Protocol:

SSL Record provides two services to SSL connection.

- Confidentiality

➤ Message Integrity

In the SSL Record Protocol application data is divided into fragments. The fragment is compressed and then encrypted MAC (Message Authentication Code) generated by algorithms like SHA (Secure Hash Protocol) and MD5 (Message Digest) is appended.

After that encryption of the data is done and in last SSL header is appended to the data.

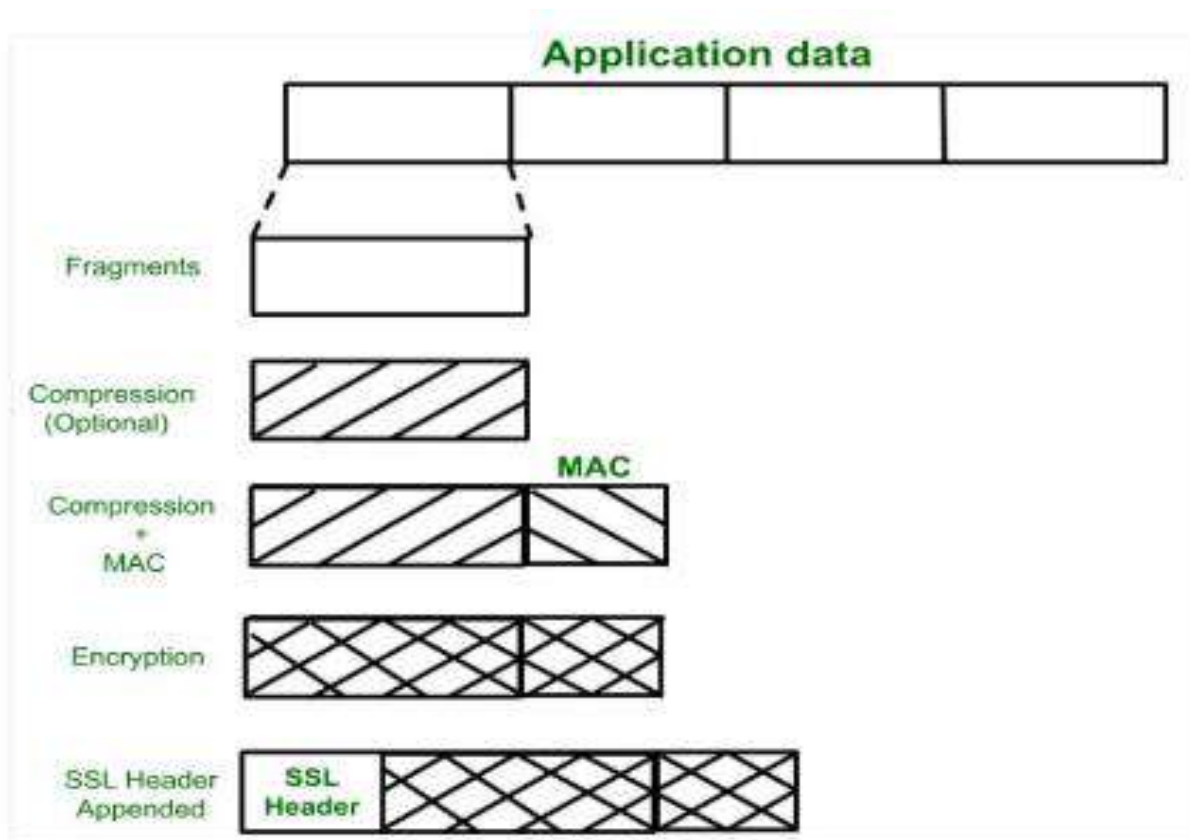
Fragments : A small part broken off or detached(separate).

MAC (Message Authentication Code) : a message authentication code (MAC), sometimes known as a tag, is a short piece of information used for authenticating a message. In other words, to confirm that the message came from the stated sender (its authenticity) and has not been changed.

SHA (Secure Hash Protocol) : The SSH protocol was developed by SSH communication security Ltd to safely communicate with the remote machine. Secure communication provides a strong password authentication and encrypted communication with a public key over an insecure channel.

MD5 (Message Digest) : The message digest is a hashing algorithm used to protect data when files are conveyed via insecure channels.

SSL Record Protocol Operation :



Handshake Protocol:

Handshake Protocol is used to establish sessions. This protocol allows the client and server to authenticate each other by sending a series of messages to each other. Handshake protocol uses four phases to complete its cycle.

Phase-1: In Phase-1 both Client and Server send hello-packets to each other. In this IP session, cipher suite and protocol version are exchanged for security purposes.

Phase-2: Server sends his certificate and Server-key-exchange. The server ends phase-2 by sending the Server-hello-end packet.

Phase-3: In this phase, Client replies to the server by sending his certificate and Client-exchange-key.

Phase-4: In Phase-4 Change-cipher suite occurred and after this Handshake Protocol ends.

Change-cipher Protocol:

This protocol uses the SSL record protocol. Unless Handshake Protocol is completed, the SSL record Output will be in a pending state. After handshake protocol, the Pending state is converted into the current state.

Change-cipher protocol consists of a single message which is 1 byte in length and can have only one value. This protocol's purpose is to cause the pending state to be copied into the current state.

1 Byte

Alert Protocol:

This protocol is used to convey SSL-related alerts to the peer entity. Each message in this protocol contains 2 bytes.



Silent Features of Secure Socket Layer:

- The advantage of this approach is that the service can be tailored to the specific needs of the given application.
- Secure Socket Layer was originated by Netscape.
- SSL is designed to make use of TCP to provide reliable end-to-end secure service.
- This is a two-layered protocol.

Handshake Protocol(TCP 3-Way Handshake Process) :

Definition:

- A more complex handshaking protocol might allow the sender to ask the receiver if it is ready to receive or for the receiver to reply with a

negative acknowledgement meaning "I did not receive your last message correctly, please resend it" (e.g., if the data was corrupted en route).

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

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

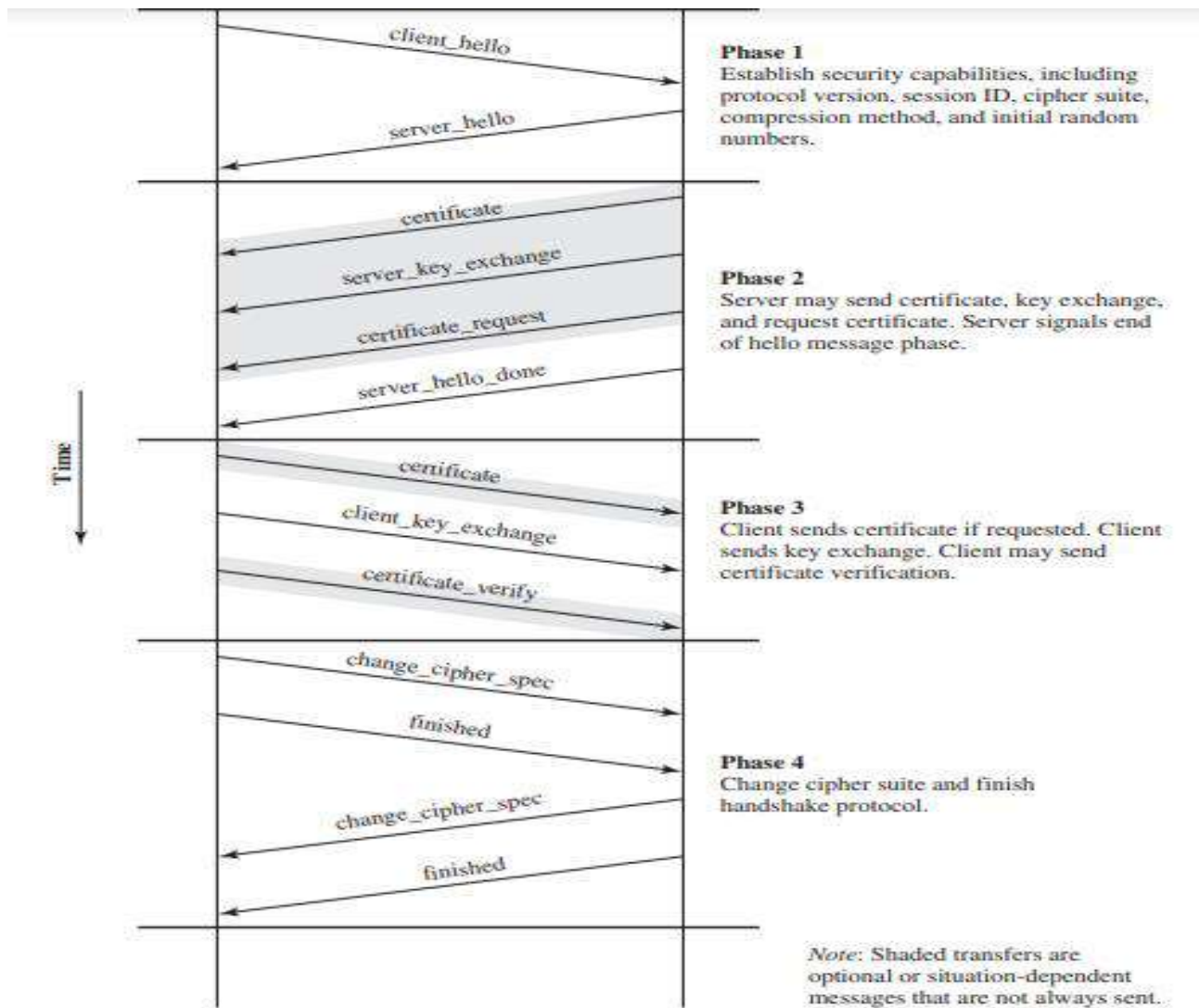


Figure 16.6 Handshake Protocol Action

✓ PHASE 1:

- ✓ **ESTABLISH SECURITY CAPABILITIES** This phase is used to initiate a logical connection and to establish the security capabilities that will be associated with it. The exchange is initiated by the client, which sends a *client_hello* message with the following parameters:
 - ✓ **Version:** The highest SSL version understood by the client.
 - ✓ **Random:** A client-generated random structure consisting of a 32-bit timestamp and 28 bytes generated by a secure random number generator. These values serve as nonces and are used during key exchange to prevent replay attacks

- ✓ Session ID: A variable-length session identifier. A nonzero value indicates that the client wishes to update the parameters of an existing connection or 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

PHASE 2. SERVER AUTHENTICATION AND KEY EXCHANGE

- ✓ The server begins this phase by sending its certificate if it needs to be authenticated
- ✓ The certificate message is required for any agreed-on key exchange method except anonymous Diffie-Hellman.
- ✓ A server_key_exchange message may be sent if it is required.

It is not required in two instances:

- ✓ (1) The server has sent a certificate with fixed Diffie-Hellman parameters or
- ✓ (2) a RSA key exchange is to be used.

PHASE 3. CLIENT AUTHENTICATION AND KEY EXCHANGE

- ✓ Upon receipt of the server_done message, the client should verify that the server provided a valid certificate (if required) and check that the server_hello parameters are acceptable. If all is satisfactory, the client sends one or more messages back to the server. If the server has requested a certificate, the client begins this phase by sending a certificate message. If no suitable certificate is available, the client sends a no_certificate alert instead

- ✓ PHASE 4. FINISH This phase completes the setting up of a secure connection.
- ✓ The client sends a change_cipher_spec message and copies the pending CipherSpec into the current CipherSpec.
- ✓ Note that this message is not considered part of the Handshake Protocol but is sent using the Change Cipher Spec Protocol. The client then immediately sends the finished message under the new algorithms, keys, and secrets.

- ✓ The finished message verifies that the key exchange and authentication processes were successful. The content of the finished message is the concatenation of two hash values
- ✓ **Change-cipher Protocol**: This protocol uses the SSL record protocol. Unless Handshake Protocol is completed, the SSL record Output will be in a pending state. After handshake protocol, the Pending state is converted into the current state. Change-cipher protocol consists of a single message which is 1 byte in length and can have only one value.
- ✓ 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

The following key exchange methods are supported.

- RSA
- Fixed Diffie-Hellman
- Ephemeral Diffie-Hellman
- Anonymous Diffie-Hellman
- Fortezza

- **RSA:** The secret key is encrypted with the receiver's RSA public key. A public-key certificate for the receiver's key must be made available.
- **Fixed Diffie-Hellman:** This is a Diffie-Hellman key exchange in which the server's certificate contains the Diffie-Hellman public parameters signed by the certificate authority (CA). That is, the public-key certificate contains the Diffie-Hellman public-key parameters. The client provides its Diffie-Hellman public-key parameters either in a certificate, if client authentication is required, or in a key exchange message. This method results in a fixed secret key between two peers based on the Diffie-Hellman calculation using the fixed public keys.
- **Ephemeral Diffie-Hellman:** This technique is used to create ephemeral (temporary, one-time) secret keys. In this case, the Diffie-Hellman public keys are exchanged, signed using the sender's private RSA or DSS key. The receiver can use the corresponding public key to verify the signature. Certificates are used to authenticate the public keys. This would appear to be the most secure of the three Diffie-Hellman options, because it results in a temporary, authenticated key.
- **Anonymous Diffie-Hellman:** The base Diffie-Hellman algorithm is used with no authentication. That is, each side sends its public Diffie-Hellman parameters to the other with no authentication. This approach is vulnerable to man-in-the-middle attacks, in which the attacker conducts anonymous Diffie-Hellman with both parties.
- **Fortezza:** The technique defined for the Fortezza scheme.