UNIT- V

KEY MANAGEMENT AND DISTRIBUTION

POORNIM
COLLEGE OF ENGINEER

LECTURE NOTES

Campus: ..... Course: ..... Class/Section: ..... Date: ....

Name of Faculty: ..... Name of Subject: ..... Code: ...

Date (Prep.): ..... Date (Del.): .....Unit No.:.....Lect. No: .....

OBJECTIVE: To be written before taking the lecture (Pl. write in bullet points the main topics/concepts etc., w will be taught in this lecture)

Transport layer Security
HTTPS
SSH.

IMPORTANT & RELEVANT QUESTIONS:

FEED BACK QUESTIONS (AFTER 20 MINUTES):

OUTCOME OF THE DELIVERED LECTURE: To be written after taking the lecture (Pl. write in bu

Transport Layer Security – TLS

- It is designed to provide security at the transport layer. TLS is derived from a security protocol called SSL.
- TLS ensures that no third party may eavesdrop or tampers with any message.
- It uses a pseudo random function to create master secret.
- It uses the HMAC as record protocol.
- In alert protocol it uses – decryption failed, record overflow, unknown CA, Access denied, Decode error, export restriction, protocol version etc fields.

Working of TLS –

The client connect to server, the client will be something. The client sends no. of specification –

1. version of SSL/TLS
2. which cipher suites, compression method it wants to use.

The server checks what is the highest TLS version that is supported by them both, picks a cipher suite from one of the clients option and optionally picks a compression method.

Both the server and client can now compute the key for symmetric encryption. The handshake is finished and the two hosts can communicate securely. To close a connection by finishing.

HTTPS - Hypertext Transfer Protocol Secure -

- Secure extension or version of HTTP.
- protocol is mainly used for providing security to the data sent between a website and a web browser.
- It is widely used on internet and used for secure communication.
- protocol uses 443 port number for communicating the data.
- It is supported by various web browsers.
- It allows users to create a secured encrypted connection and helps them to protect their information from being stolen.

SECURE SHELL (SSH)

Secure Shell (SSH) is a protocol for secure network communications designed to be relatively simple and inexpensive to implement. The initial version, SSH1 was focused on providing a secure remote logon facility to replace TELNET and other remote logon schemes that provided no security. SSH also provides a more general client/server capability and can be used for such network functions as file transfer and e-
mail. A new version, SSH2, fixes a number of security flaws in the original scheme. S SH2 is documented as a proposed standard in IETF RFCs 4250 through 4256.

SSH client and server applications are widely available for most operating sys- tems. It has become the method of choice for remote login and X tunneling and is rapidly becoming one of the most pervasive applications for encryption technology outside of embedded systems.

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

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

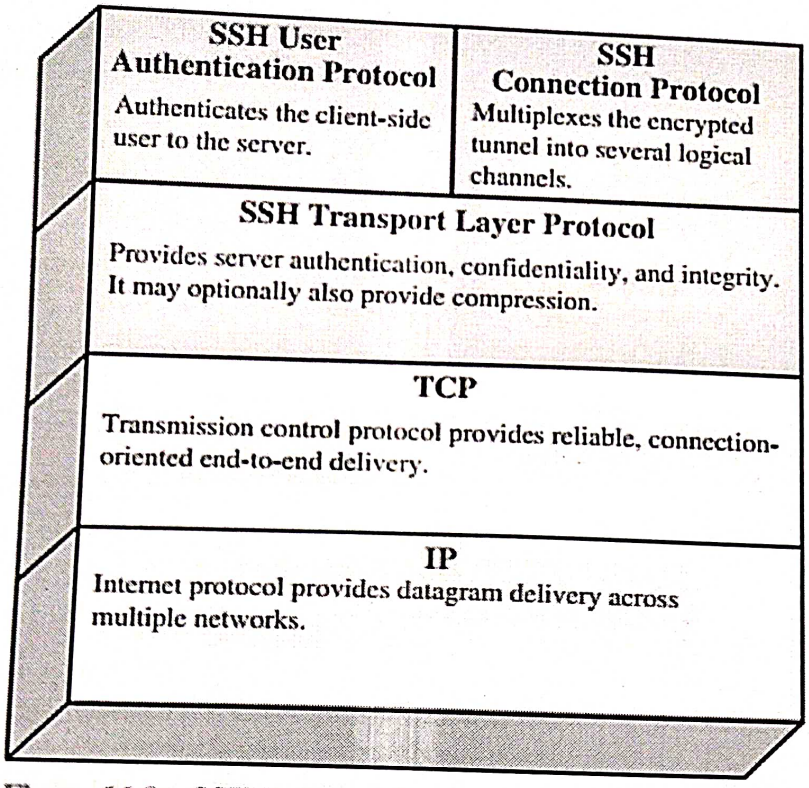


Figure 16.8 SSH Protocol Stack

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

Transport Layer Protocol

HOST KEYS Server authentication occurs at the transport layer, based on the server possessing a public/private key pair. A server may have multiple host keys using multiple different asymmetric encryption algorithms. Multiple hosts may share the same host key. In any case, the server host key is used during key exchange to authenticate the identity of the host. For this to be possible, the client must have a priori knowledge of the server's public host key. RFC 4251 dictates two alternative trust models that can be used:

1. The client has a local database that associates each host name (as typed by the user) with the corresponding public host key. This method

requires no centrally administered infrastructure and no third-party coordination. The downside is that the database of name-to-key associations may become burdensome to maintain.

2. The host name-to-key association is certified by a trusted certification authority (CA). The client only knows the CA root key and can verify the validity of all host keys certified by accepted CAs. This alternative eases the maintenance problem, since ideally, only a single CA key needs to be securely stored on the client. On the other hand, each host key must be appropriately certified by a central authority before authorization is possible.

PACKET EXCHANGE Figure 16.9 illustrates the sequence of events in the SSH Transport Layer Protocol. First, the client establishes a TCP connection to the server. This is done via the TCP protocol and is not part of the Transport Layer Protocol. Once the connection is established, the client and server exchange data, referred to as packets, in the data field of a TCP segment. Each packet is in the following format (Figure 16.10).

- **Packet length:** Length of the packet in bytes, not including the packet length and MAC fields.
- **Padding length:** Length of the random padding field.
- **Payload:** Useful contents of the packet. Prior to algorithm negotiation, this field is uncompressed. If compression is negotiated, then in subsequent packets, this field is compressed.
- **Random padding:** Once an encryption algorithm has been negotiated, this field is added. It contains random bytes of padding so that that total length of the packet (excluding the MAC field) is a multiple of the cipher block size, or 8 bytes for a stream cipher.
- **Message authentication code (MAC):** If message authentication has been negotiated, this field contains the MAC value. The MAC value is computed over the entire packet plus a sequence number, excluding the MAC field. The sequence number is an implicit 32-bit packet sequence that is initialized to

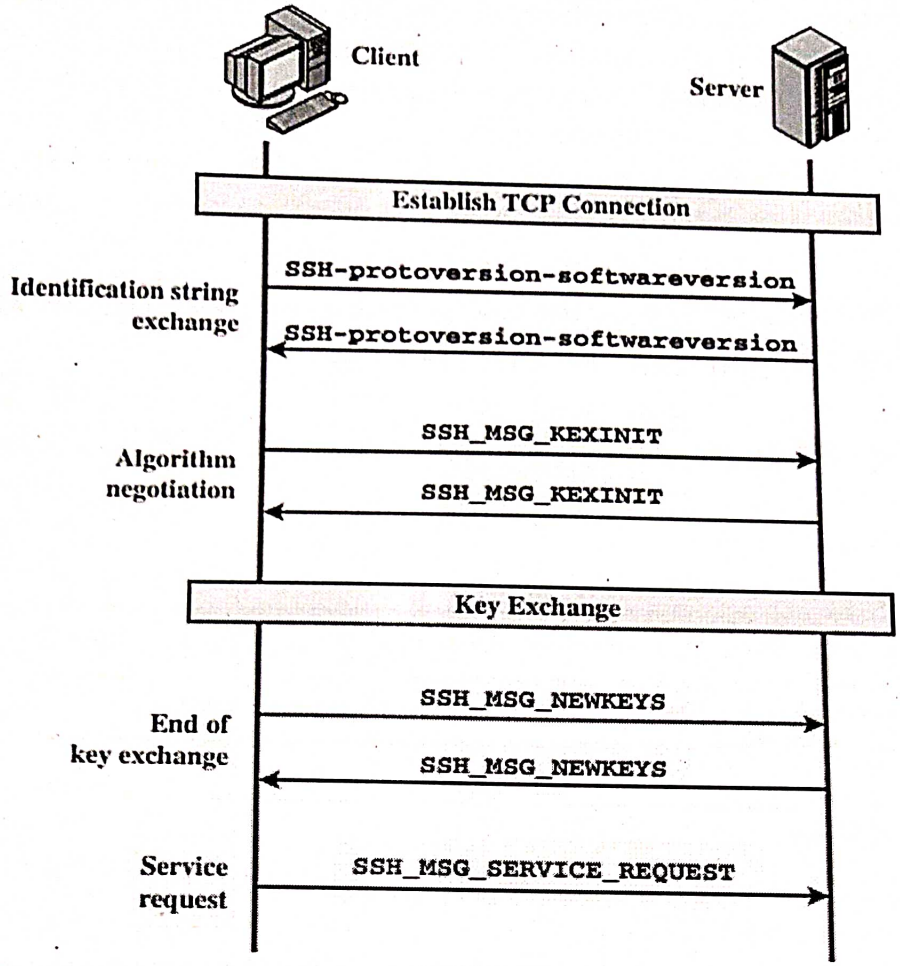


Figure 16.9 SSH Transport Layer Protocol Packet Exchanges

zero for the first packet and incremented for every packet. The sequence number is not included in the packet sent over the TCP connection.

Once an encryption algorithm has been negotiated, the entire packet (excluding the MAC field) is encrypted after the MAC value is calculated.

The SSH Transport Layer packet exchange consists of a sequence of steps (Figure 16.9). The first step, the **identification string exchange**, begins with the client sending a packet with an identification string of the form:

SSH-protoversion-softwareversion SP comments CR LF

where SP, CR, and LFare space character, carriage return, and line feed, respectively. An example of a valid string is SSH-2.0-billsSSH_3.6.3q3<CR><LF>. The server responds with its own identification string. These strings are used in the Diffie- Hellman key exchange.

Next comes **algorithm negotiation**. Each side sends an SSH_MSG_KEXINITcontaining lists of supported algorithms in the order of preference to the sender. There is one list for each type of cryptographic algorithm. The algorithms include key exchange, encryption, MAC algorithm, and compression algorithm. Table 16.3 shows the allowable options for encryption, MAC, and compression. For each category, the algorithm chosen is the first algorithm on the client's list that is also supported by the server.

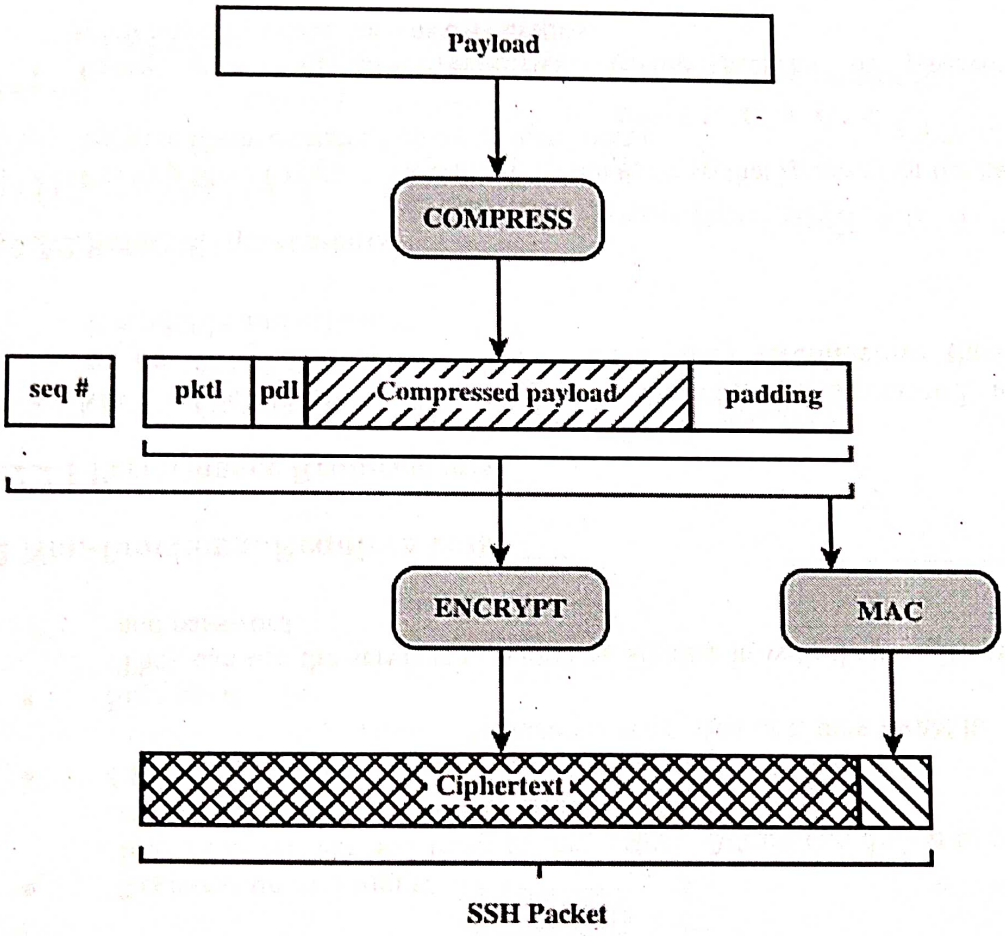


Figure 16.10 SSH Transport Layer Protocol Packet Formation

The next step is **key exchange**. The specification allows for alternative methods of key exchange, but at present, only two versions of Diffie-Hellman key exchange are specified. Both versions are defined in RFC 2409 and require only one packet in each direction. The following steps are involved in the exchange. In this, C is the client; S is the server; p is a large safe prime; g is a generator for a subgroup of GF(p); q is the order of the subgroup; V_S is S's identification string; V_C is C's identification string;

K_S is S's public host key; I_C is C's SSH_MSG_KEXINIT message and I_S is S's SSH_MSG_KEXINIT message that have been exchanged before this part begins. The values of p, g, and q are known to both client and server as a result of the algorithm selection negotiation. The hash function hash() is also decided during algorithm negotiation.

1. C generates a random number $x (1 < x < q)$ and computes $e = g^x \bmod p$. C sends e to S.

2. S generates a random number $y (0 < y < q)$ and computes $f = g^y \bmod p$. S receives e. It computes $K = e^y \bmod p$, $H = \text{hash}(V\_C \| V\_S \| I\_C \| I\_S \| K\_S \| e \| f \| K)$, and signature s on H with its private host key. S sends $(K\_S \| f \| s)$ to C. The signing operation may involve a second hashing operation.

Table 16.3 SSH Transport Layer Cryptographic Algorithms

Cipher	
3des-cbc*	Three-key 3DES in CBC mode
blowfish-cbc	Blowfish in CBC mode
twofish256-cbc	Twofish in CBC mode with a 256-bit key
twofish192-cbc	Twofish with a 192-bit key
twofish128-cbc	Twofish with a 128-bit key
aes256-cbc	AES in CBC mode with a 256-bit key
aes192-cbc	AES with a 192-bit key
aes128-cbc**	AES with a 128-bit key
Serpent256-cbc	Serpent in CBC mode with a 256-bit key
Serpent192-cbc	Serpent with a 192-bit key
Serpent128-cbc	Serpent with a 128-bit key
arcfour	RC4 with a 128-bit key
cast128-cbc	CAST-128 in CBC mode

MAC algorithm	
hmac-sha1*	HMAC-SHA1; digest length = key length = 20
hmac-sha1-96**	First 96 bits of HMAC-SHA1; digest length = 12; key length = 20
hmac-md5	HMAC-SHA1; digest length = key length = 16
hmac-md5-96	First 96 bits of HMAC-SHA1; digest length = 12; key length = 16

Compression algorithm	
none*	No compression
zlib	Defined in RFC 1950 and RFC 1951

* = Required

** = Recommended

1. C verifies that K_S really is the host key for S (e.g., using certificates or a local database). C is also allowed to accept the key without verification; however, doing so will render the protocol insecure against active attacks (but may be desirable for practical reasons in the short term in many environments). C then computes $K = f_x \bmod p$, $H = \text{hash}(V\_C \| V\_S \| I\_C \| I\_S \| K\_S \| e \| f \| K)$, and verifies the signature s on H.

As a result of these steps, the two sides now share a master key K. In addition, the server has been authenticated to the client, because the server has used its pri- vate

key to sign its half of the Diffie-Hellman exchange. Finally, the hash value H serves as a session identifier for this connection. Once computed, the sessio n identifier is not changed, even if the key exchange is performed again for this connection to obtain fresh keys.

The **end of key exchange** is signaled by the exchange of SSH_MSG_NEWKEYS pac kets. At this point, both sides may start using the keys generated from K, as discussed subsequently.

The final step is **service request**. The client sends an SSH_MSG_ SERVICE_REQUEST packet to request either the User Authentication or the Connection Protocol. Subsequent to this, all d ata is exchanged as the payload of an SSH Transport Layer packet, protected by encryption and MAC.

KEY GENERATION The keys used for encryption and MAC (and any needed IVs) are generated from the shared secret key K, the hash value from the key exchange H, and the session identifier, which is equal to H unless there has been a subsequent key exchange after the initial key exchange. The values are computed as follows.

- Initial IV client to server: HASH(K || H || "A" || session_id)
- Initial IV server to client: HASH(K || H || "B" || session_id)
- Encryption key client to server: HASH(K || H || "C" || session_id)
- Encryption key server to client: HASH(K || H || "D" || session_id)
- Integrity key client to server: HASH(K || H || "E" || session_id)
- Integrity key server to client: HASH(K || H || "F" || session_id)

where HASH()is the hash function determined during algorithm negotiation.

User Authentication Protocol

The User Authentication Protocol provides the means by which the client is authenticated to the server.

MESSAGE TYPES AND FORMATS Three types of messages are always used in the User Authentication Protocol. Authentication requests from the client have the format:

byte	SSH_MSG_USERAUTH_REQUEST (50)		
string	user name string	service name string	method name

... method specific fields

where user name is the authorization identity the client is claiming, service name is the facility to which the client is requesting access (typically the SSH Connection Protocol), and method name is the authentication method being used in this request. The first byte has decimal value 50, which is interpreted as SSH_MSG_USERAUTH_REQUEST.

If the server either (1) rejects the authentication request or (2) accepts the request but requires one or more additional authentication methods, the server sends a message with the format:

byte	SSH_MSG_USERAUTH_FAILURE (51)
name-list	authentications that can continue
boolean	partial success

where the name-list is a list of methods that may productively continue the dialog. If the server accepts authentication, it sends a single byte message: SSH_MSG_ USERAUTH_SUCCESS (52).

MESSAGE EXCHANGE The message exchange involves the following steps.

1. The client sends a SSH_MSG_USERAUTH_REQUEST with a requested method of none.
2. The server checks to determine if the user name is valid. If not, the server returns SSH_MSG_USERAUTH_FAILURE with the partial success value of false. If the user name is valid, the server proceeds to step 3.
3. The server returns SSH_MSG_USERAUTH_FAILURE with a list of one or more authentication methods to be used.
4. The client selects one of the acceptable authentication methods and sends a SSH_MSG_USERAUTH_REQUEST with that method name and the required method-specific fields. At this point, there may be a sequence of exchanges to perform the method.
5. If the authentication succeeds and more authentication methods are required, the server proceeds to step 3, using a partial success value of true. If the authentication fails, the server proceeds to step 3, using a partial success value of false.

6. When all required authentication methods succeed, the server sends a SSH_MSG_USERAUTH_SUCCESSmessage, and the Authentication Protocol is over.

AUTHENTICATION METHODS The server may require one or more of the following authentication methods.

- **publickey:**The details of this method depend on the public-key algorithm chosen. In essence, the client sends a message to the server that contains the client's public key, with the message signed by the client's private key. When the server receives this message, it checks whether the supplied key is acceptable for authentication and, if so, it checks whether the signature is correct.
- **password:** The client sends a message containing a plaintext password, which is protected by encryption by the Transport Layer Protocol.
- **hostbased:**Authentication is performed on the client's host rather than the client itself. Thus, a host that supports multiple clients would provide authentication for all its clients. This method works by having the client send a signature created with the private key of the client host. Thus, rather than directly verifying the user's identity, the SSH server verifies the identity of the client host—and then believes the host when it says the user has already authenti- cated on the client side.

Connection Protocol

The SSH Connection Protocol runs on top of the SSH Transport Layer Protocol and assumes that a secure authentication connection is in use.2 That secure authentication connection, referred to as a **tunnel**, is used by the Connection Protocol to multiplex a number of logical channels.

CHANNEL MECHANISM All types of communication using SSH, such as a terminal session, are supported using separate channels. Either side may open a channel. For each channel, each side associates a unique channel number, which need not be the same on both ends. Channels are flow controlled using a window mechanism. No data may be sent to a channel until a message is received to indicate that window space is available.

The life of a channel progresses through three stages: opening a channel, data transfer, and closing a channel.

When either side wishes to **open a new channel**, it allocates a local number for the channel and then sends a message of the form:

byte	SSH_MSG_CHANNEL_OPEN
string	channel type
uint32	sender channel
uint32	initial window size
uint32	maximum packet size
....	channel type specific data follows

where uint32 means unsigned 32-bit integer. The channel type identifies the application for this channel, as described subsequently. The sender channel is the local channel number. The initial window size specifies how many bytes of channel data can be sent to the sender of this message without adjusting the window. The maximum packet size specifies the maximum size of an individual data packet that can be sent to the sender. For example, one might want to use smaller packets for interactive connections to get better interactive response on slow links.

If the remote side is able to open the channel, it returns a SSH_MSG_ CHANNEL_OPEN_CONFIRMATION message, which includes the sender channel number, the recipient channel number, and window and packet size values for incoming traffic. Otherwise, the remote side returns a SSH_MSG_CHANNEL_ OPEN_FAILUREmessage with a reason code indicating the reason for failure.

Once a channel is open, **data transfer** is performed using a SSH_MSG_ CHANNEL_DATAmessage, which includes the recipient channel number and a block of data. These messages, in both directions, may continue as long as the channel is open.

When either side wishes to **close a channel,** it sends a SSH_MSG_ CHANNEL_CLOSEmessage, which includes the recipient channel number.

Figure 16.11 provides an example of Connection Protocol Message Exchange.

CHANNEL TYPES Four channel types are recognized in the SSH Connection Protocol specification.

session: The remote execution of a program. The program may be a shell, an application such as file transfer or e-mail, a system command, or some built-in

subsystem. Once a session channel is opened, subsequent requests are used to start the remote program.

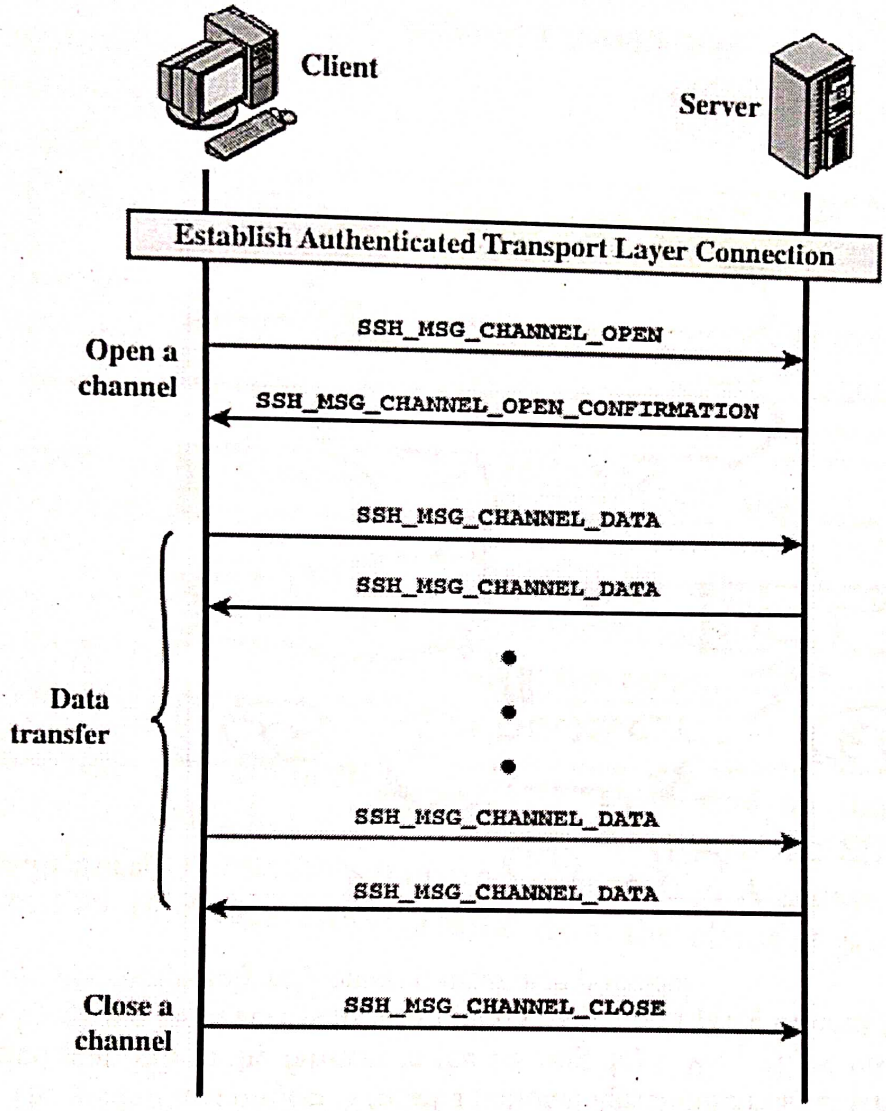


Figure 16.11 Example SSH Connection Protocol Message Exchange

- **x11:** This refers to the X Window System, a computer software system and n et- work protocol that provides a graphical user interface (GUI) for networked computers. X allows applications to run on a network server but to be displayed on a desktop machine.
- **forwarded-tcpip:** This is remote port forwarding, as explained in the next sub- section.
- **direct-tcpip:** This is local port forwarding, as explained in the next subsection.

PORT FORWARDING One of the most useful features of SSH is port forwarding. In essence, port forwarding provides the ability to convert any insecure TCP connection into a secure SSH connection. This is also referred to as SSH tunneling. We need to know what a port is in this context. A **port** is an identifier of a user of TCP. So, any application that runs on top of TCP has a port number. Incoming TCP traffic is delivered to the appropriate application on the basis of the port number. An application may employ multiple port numbers. For example, for the Simple Mail Transfer Protocol (SMTP), the server side generally listens on port 25, so an incoming SMTP request uses TCP and addresses the data to destination port 25. TCP recognizes that this is the SMTP server address and routes the data to the SMTP server application.

Figure 16.12 illustrates the basic concept behind port forwarding. We have a client application that is identified by port number x and a server application identified by port number y. At some point, the client application invokes the local TCP entity and requests a connection to the remote server on port y. The local TCP entity negotiates a TCP connection with the remote TCP entity, such that the connection links local port x to remote port y.

To secure this connection, SSH is configured so that the SSH Transport Layer Protocol establishes a TCP connection between the SSH client and server entities with TCP port numbers a and b, respectively. A secure SSH tunnel is established over this TCP connection. Traffic from the client at port x is redirected to the loc

al.

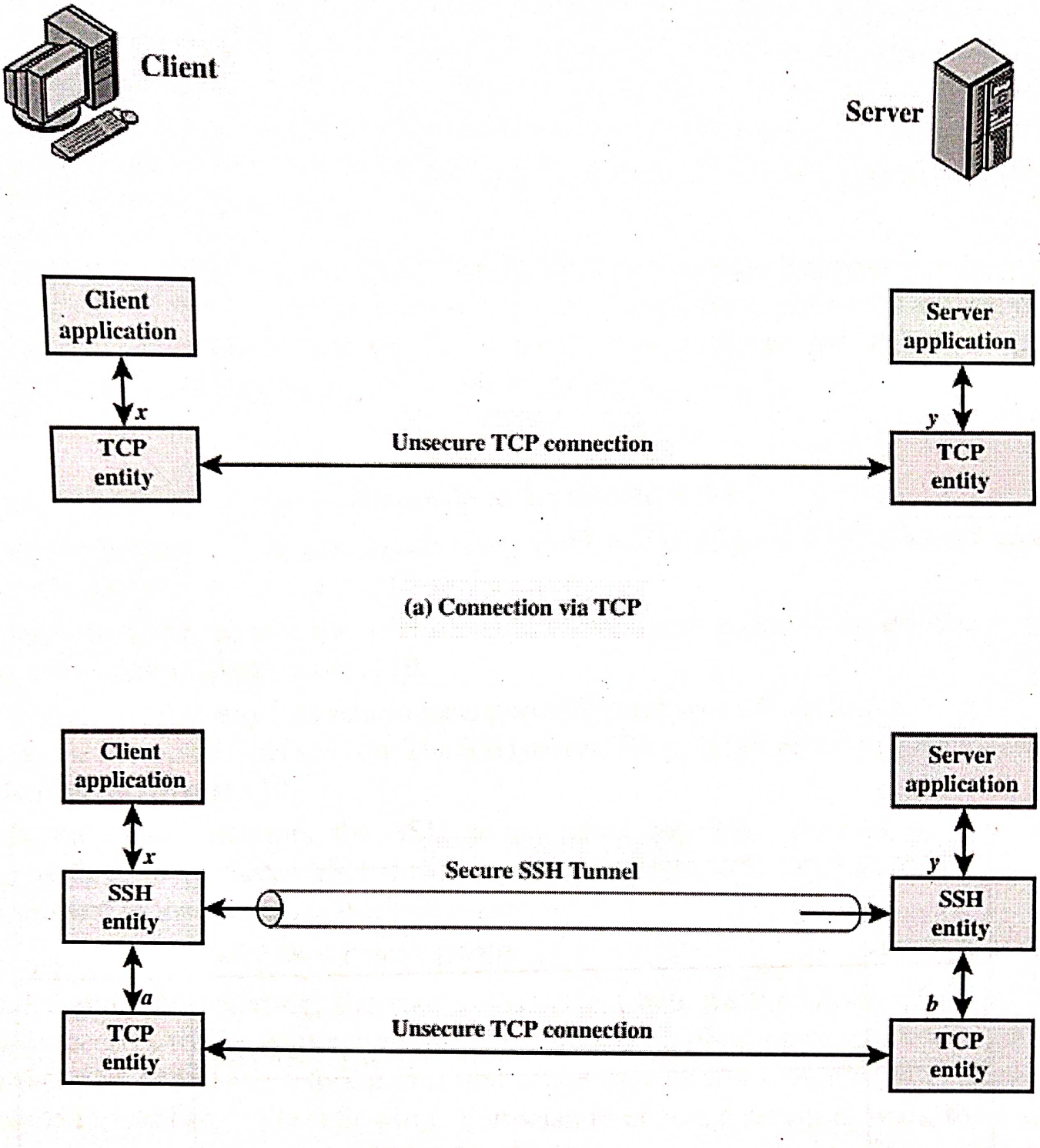


Figure 16.12 SSH Transport Layer Packet Exchanges

SSH entity and travels through the tunnel where the remote SSH entity delivers the data to the server application on port *y*. Traffic in the other direction is similarly redirected.

SSH supports two types of port forwarding: local forwarding and remote forwarding. **Local forwarding** allows the client to set up a "hijacker" process. This will intercept selected application-level traffic and redirect it from an unsecured TCP connection to a secure SSH tunnel. SSH is configured to listen on selected ports. SSH grabs all traffic using a selected port and sends it through an SSH tunnel. On the other end, the SSH server sends the incoming traffic to the destination port dictated by the client application.

The following example should help clarify local forwarding. Suppose you have an e-mail client on your desktop and use it to get e-mail from your mail server via the Post Office Protocol (POP). The assigned port number for POP3 is port 110. We can secure this traffic in the following way:

1. The SSH client sets up a connection to the remote server.
2. Select an unused local port number, say 9999, and configure SSH to accept traffic from this port destined for port 110 on the server.
3. The SSH client informs the SSH server to create a connection to the destination, in this case mailserver port 110.
4. The client takes any bits sent to local port 9999 and sends them to the server inside the encrypted SSH session. The SSH server decrypts the incoming bits and send s the plaintext to port 110.
5. In the other direction, the SSH server takes any bits received on port 110 and sends them inside the SSH session back to the client, who decrypts and sends them to the process connected to port 9999.

With **remote forwarding**, the user's SSH client acts on the server's behalf. The client receives traffic with a given destination port number, places the traffic on the correct port and sends it to the destination the user chooses. A typical example of remote forwarding is the following. You wish to access a server at work from your home computer. Because the work server is behind a firewall, it will not accept an SSH request from your home computer. However, from work you can set up an SSH tunnel using remote forwarding. This involves the follow- ing steps.

1. From the work computer, set up an SSH connection to your home computer. The firewall will allow this, because it is a protected outgoing connection.

2. Configure the SSH server to listen on a local port, say 22, and to deliver data across the SSH connection addressed to remote port, say 2222.

3. You can now go to your home computer, and configure SSH to accept traffic on port 2222.

4. You now have an SSH tunnel that can be used for remote logon to the work server.

POORNIMA
COLLEGE OF ENGINEERING

LECTURE NOTES

Campus: ..... Course: ..... Class/Section: ..... Date: .....

Name of Faculty: ..... Name of Subject: ..... Code: .....

Date (Prep.): ..... Date (Del.): .....Unit No./Topic:.....Lect. No: .....

OBJECTIVE: To be written before taking the lecture (Pl. write in bullet points the main topics/concepts etc., which will be taught in this lecture)

Web Security Threats and Approaches

IMPORTANT & RELEVANT QUESTIONS:

FEED BACK QUESTIONS (AFTER 20 MINUTES):

OUTCOME OF THE DELIVERED LECTURE: To be written after taking the lecture (Pl. write in bullet points about students' feedback on this lecture, level of understanding of this lecture by students etc.)

IP-Security

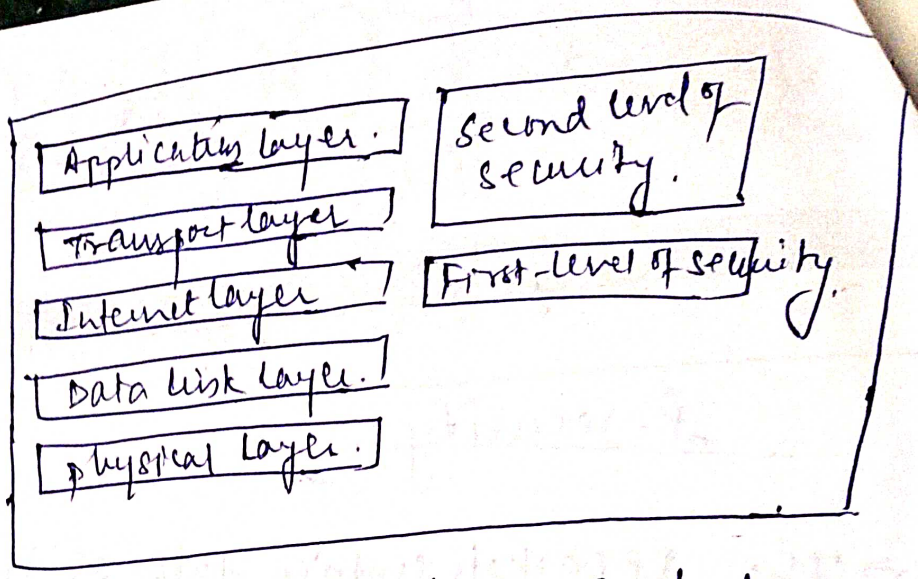
⇒ The IP packets contain data in the plain text form. That is any one watching IP packets pass by can access them, read their contents and even change them.

→ There is a high level of security (such as SSL, SHTTP, PGP, S/MIME, & SET etc) to prevent such attacks, These protocols provide enhance security.

→ If we able to provide security to IP-packets then we need not rely on higher security and It can serve as additional security measures.

→ So we have 2 levels of security.

1) First offer security at the IP-level itself.
2) Continue implementing high level security whenever required.



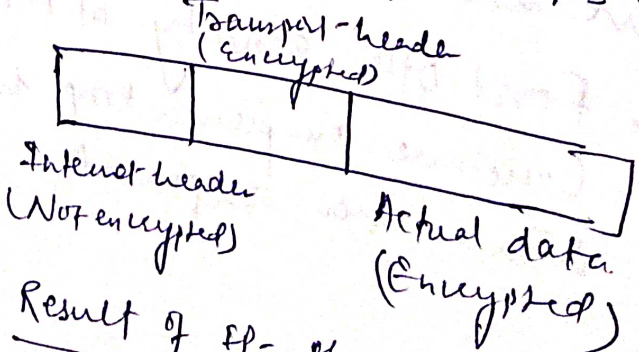
Security at Internet layer and above layer.

→ IPV4 may support these features, but IPV6 must support them.

→ The overall idea of IPSec is to encrypt and seal the transport and application layer data during transmission. It also offers integrity protection for the Internet layer.

→ Internet header itself is not encrypted, it header used to transmit to intermediate routers.

→ The logical format of a message after IP-Sec. processing



→ Result of IP- processing.

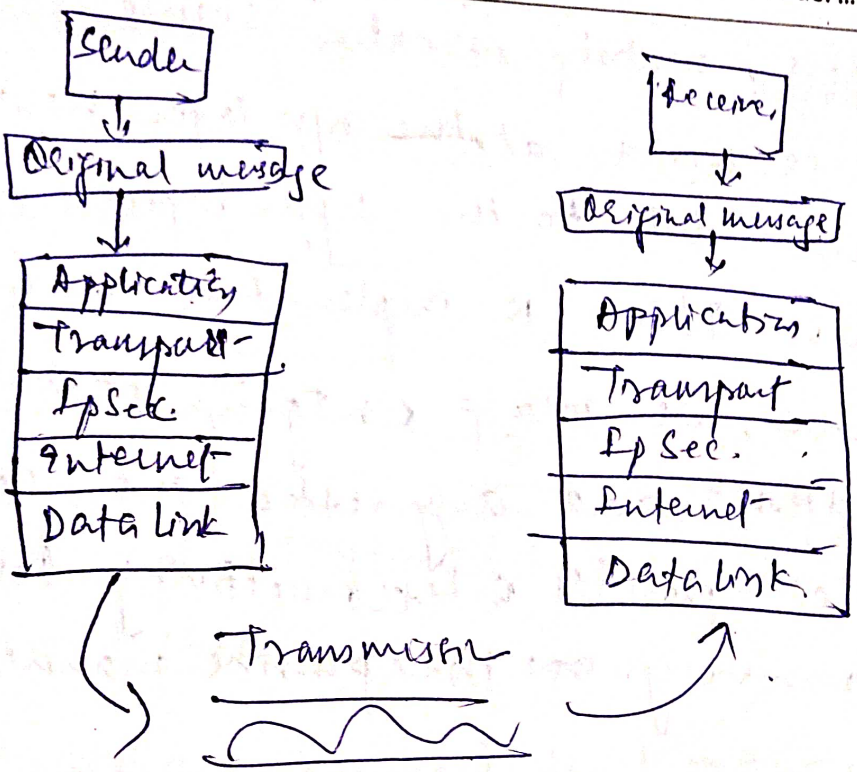


Fig:- Conceptual IPSec positioning in the TCP/IP protocol stack

2) Applications and Advantages of IP-sec.

- Applications:

1) Secure remote Internet access – secure connection between local to remote desktops. through ISP).

2) Secure branch office connectivity – needn't required costly leased line. can connect through IP-sec. enabled hpns. to connect securly all branches in cities.

3. Set-up communication with other organisations.

→ Following are the main advantages of IPSec

- IPSec is transparent to the end users. There is no need for an user training, key issuance & revocation.
- When IP sec is configured to work with a firewall it becomes the only entry-exit point for all traffic; making it extra secure.
- IPSec works at the n/w layer, Hence no change are needed to the upper layer.
- When IPSec is implemented in a firewall & router, all incoming & outgoing packets are protected. It doesn't add any extra overhead.
- IP-Sec provides interconnectivity between branches in a very ~~ex~~ inexpensive manner.

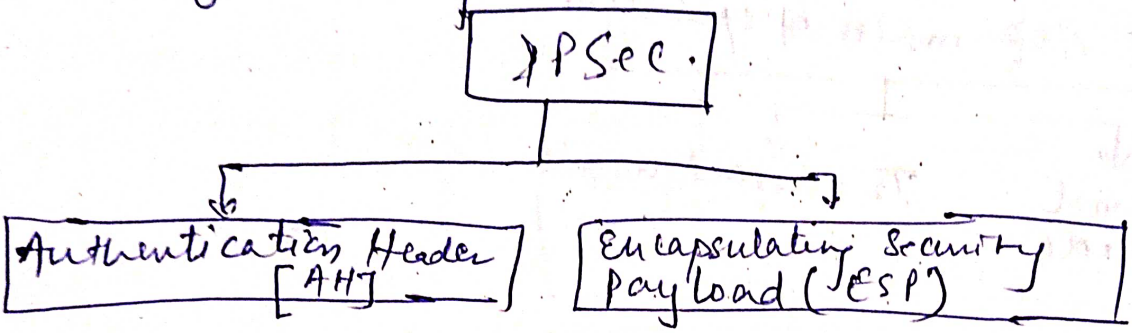
IPSec Protocols:

IP packet < IP header
actual data.

→ IP-sec features are implemented in the form of additional IP-headers (called as extension headers). ~~addi~~ extention IP headers follow the standard IP headers.

→ IPSec 2 services < Authentication
Confidentiality.

to implement these 2 services, IPSec defines two IP extention headers one for authentication & another for confidentiality.



IPSec. Protocols.

→ These 2 protocols are required for the following purposes.

1) The Authentication Header (AH) protocol, provides authentication, integrity and an Optional anti-replay service. The AH is simply inserted between the IP header and any subsequent packet contents. No changes required to the data contents packet. The security resides completely inside the contents of the AH.

→ Encapsulating Security Payload (ESP) protocol: provides data confidentiality. It also defines a new header to be inserted into the IP packet. ESP processing also includes the transformation of the protected data into an unreadable, encrypted format.

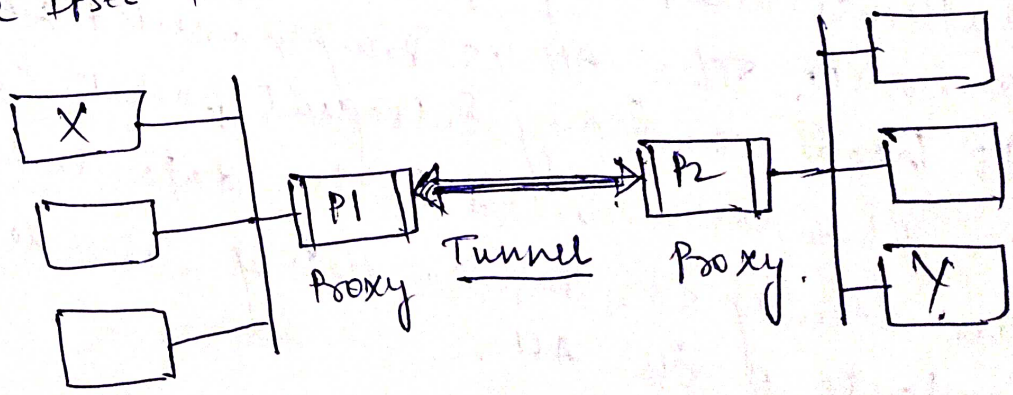
Modes of operation

AH & ESP modes of operation

- Tunnel mode
- Transport mode

1. Tunnel mode :-

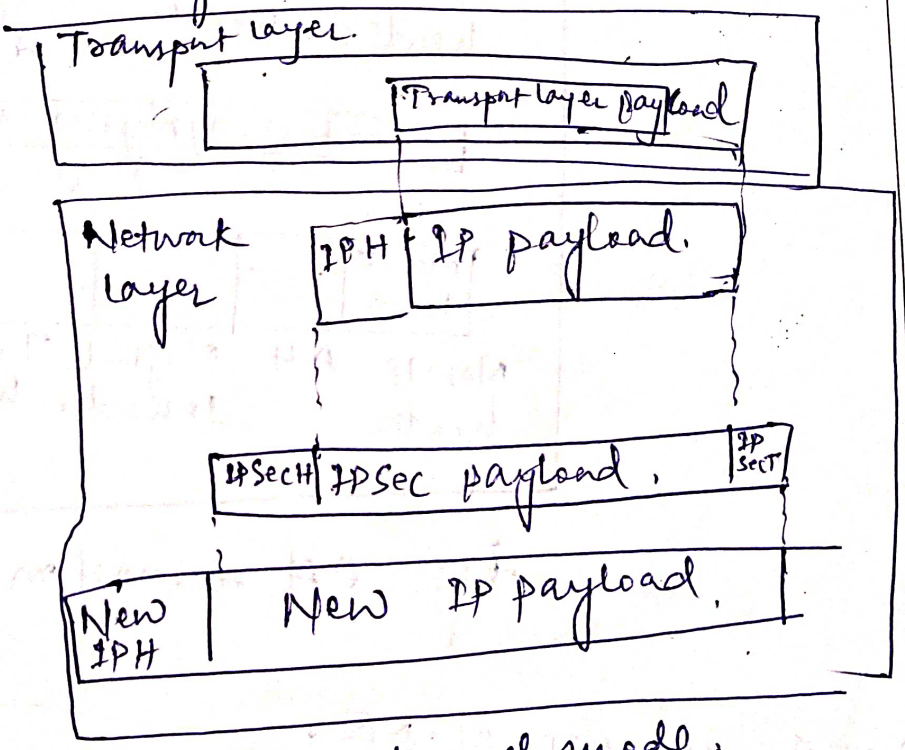
In tunnel mode an encrypted tunnel is established between two hosts. Suppose X and Y are two hosts, wanting to communicate with each other using the IPsec tunnel mode.



Concept of tunnel mode.

Campus: ..... Course: .....
Name of Faculty: .....
Class/Section: .....
Name of Subject: .....
Date: .....
Code: .....

→ In the tunnel mode, IP sec protects the entire datagram. It takes an IP datagram (including the IP header), adds the IPSec header and trailer and encrypts the whole thing. It then adds new IP header to this encrypted datagram.



IP Sec. tunnel mode.

→ In tunnel mode, the new IP header has information different from that is there in the original IP header. The tunnel mode used between two routers, a host and router.

→ It is not used between 2 hosts.

⇒ AH tunnel tunnel mode:-

In the tunnel mode, entire original IP packet is authenticated and the AH is inserted between the original IP header and a new outer IP header.

→ The inner IP header contains the original source and destination IP addresses, whereas the outer IP address contains possibly different IP addresses.

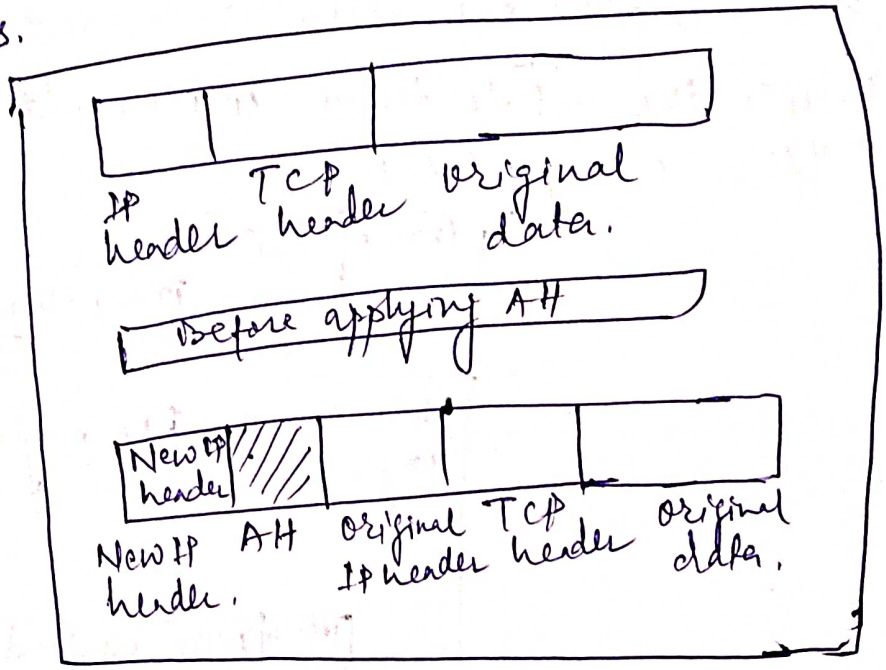


Fig. AH tunnel mode.

2) TRANSPORT MODE:-

In contrast, the transport mode does not hide the actual source and destination addresses. They are visible in plain text, while in transmit. In the transport mode, IPSec. takes the transport layer payload adds IPSec header and trailer, encrypts the whole thing and then adds the IP header. Thus, the IP header is not encrypted.

Campus: ..... Course: ..... Class/Section: ..... Date: .....
Name of Faculty: ..... Name of Subject: ..... Code: .....

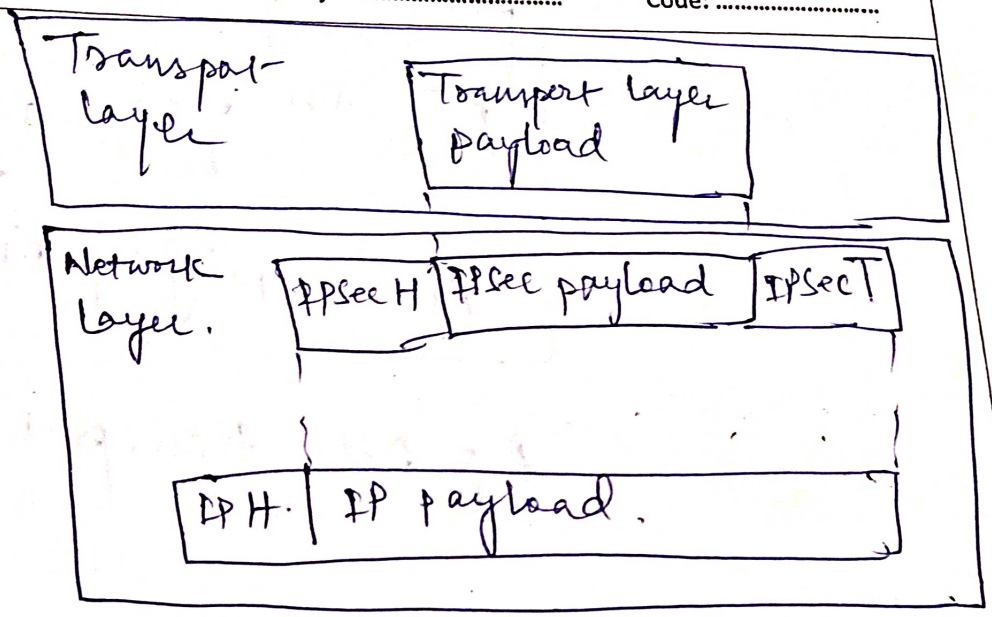
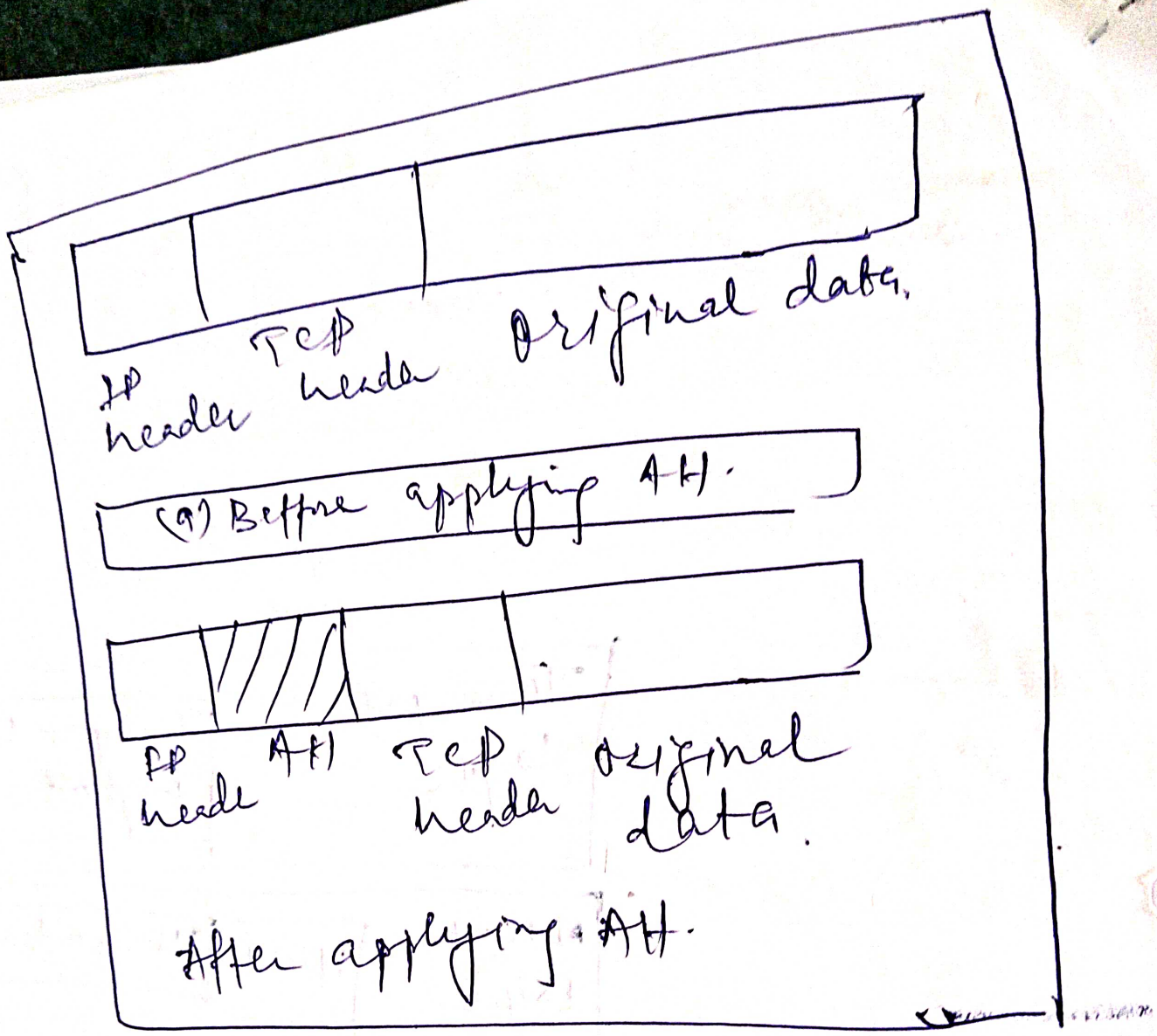


fig. IPsec transport mode.

→ The transport mode is useful when we are interested in a ~~but~~ not uses IPsec to authenticate and / or encrypt the transport layer payload and only the receiver verifies it.

AH transport mode:

In this mode the position of authentication header (AH) is placed between original IP header and original TCP header of the IP packet.



Ag AH transport mode.

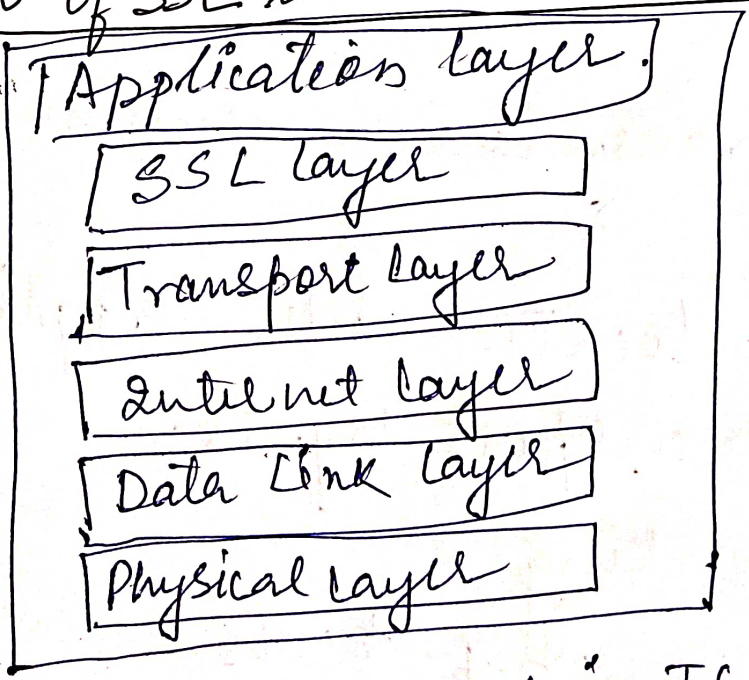
Secure Socket Layer (SSL)

PAGE NO.

It is an internet protocol for secure exchange of information between a web browser and a web server.
It provides two basic security services -

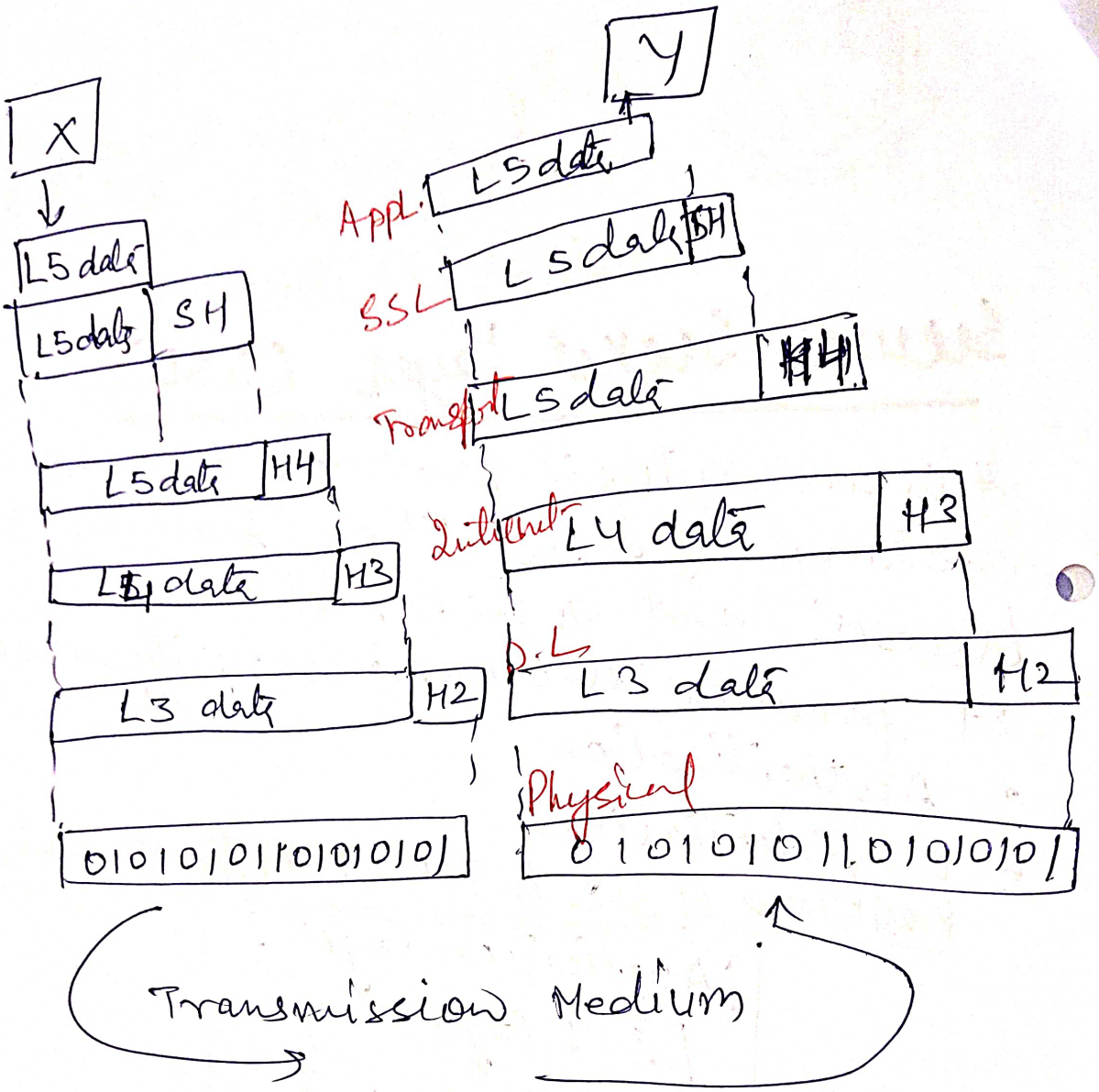
- Authentication
- confidentiality

Position of SSL in TCP/IP Protocol Suite -



Position of SSL in TCP/IP

SSL can be conceptually considered as an additional layer in the TCP/IP protocol suite.



SSL is located b/w application and transport layers.

SSL working –

SSL has 3 sub-protocols –

↳ Handshake protocol
↳ Record protocol
↳ Alert protocol.

} 3 constitute the working of SSL.

Handshake protocol -

PAGE NO.

Type	Length	Content
1byte	3bytes	1 or more bytes

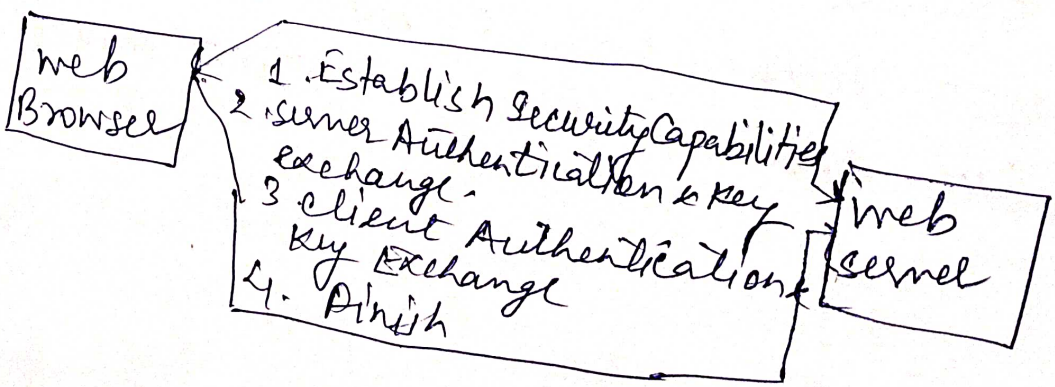
format for handshake protocol

- It is used by the client and server to communicate using an SSL enable connection.
- It consists of a series of messages between the client and server.
- Similar to how Alice and Bob would first shake hands with each other with hello before they start conversing.

Message type	Parameters
Hello request	None
client hello	version, sessionid, cipher suite
server hello	"
certificate	X.509 V3 certificate
server key exchange	Parameters, Signature
certificate request	Type, Authorities
server hellodone	None
certificate verify	Signature
client key Exchange	Parameters, Signature
Finished	Hash value.

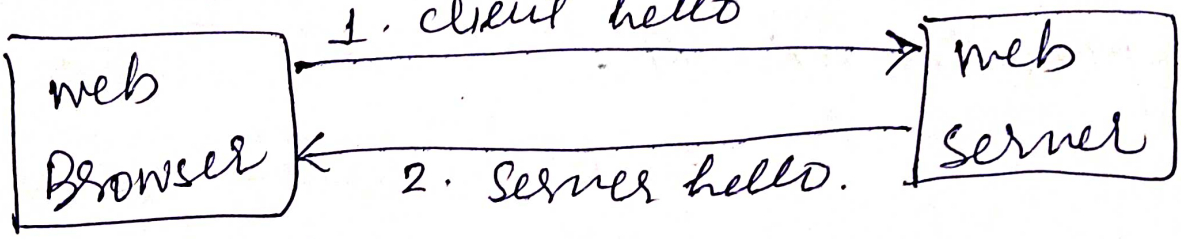
Handshake protocol has four phases -

1. Establish security capabilities
2. Server authentication and key Exchange.
3. client authentication and key exchange.
4. Finish



Phase-1 · Establish security capabilities -

It is used to initiate a logical connection and establish the security capabilites associated



with that connection.

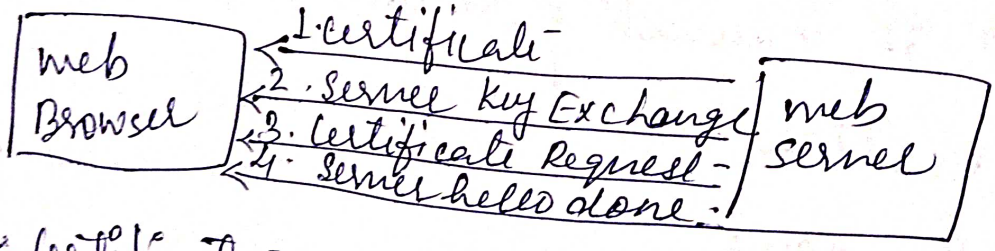
client hello message from client to server and it consists of following parameters -

- version
- Random
- session id
- Cipher suite
- compression Method.

client sends hello message to the server. and waits for server's response. Accordingly, server sends back a server hello message to the client. this contains -

- version
- Random
- session id
- cipher suite
- compression Method.

Phase 2 - Server Authentication and Key Exchange - Server initiates this 2 phase of the SSL handshake and is the sole sender of all messages in this phase.



→ Certificate - server sends its digital certificate and entire chain leading up to root CA to the client. Server certificate is mandatory in all situations, except if the key is being agreed upon Diffie-Hellman.

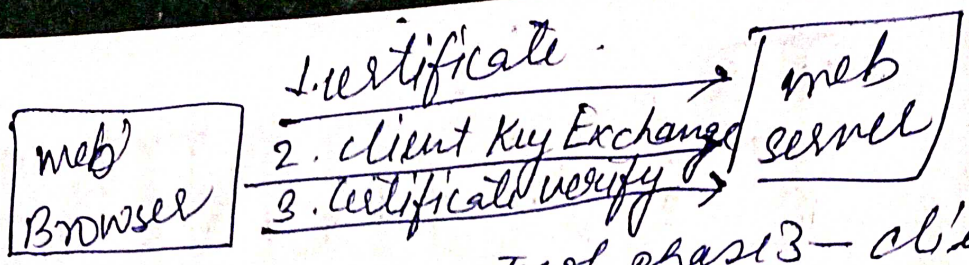
- Server Key Exchange - It is optional. It is used only if the server does not send it digital certificate to the client in step 1. Server sends its public key to client.

- Certificate Request - Server can request for client's digital certificate. The client authentication in SSL is optional and the server may not always except the client to be authenticated.

- Server hello done - indicates to the client that its portion of hello message is complete.

Phase 3 - Client Authentication and Key Exchange -

The client initiates this third phase of the SSL handshake and is the sole sender of all messages in this phase.



SSL handshake protocol phase 3 – client authentication & Key Exchange.

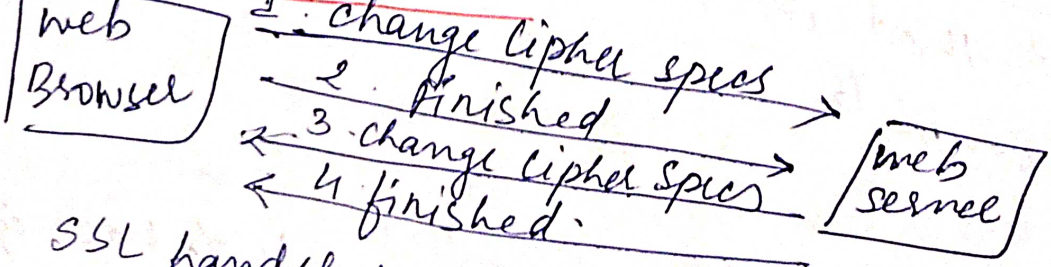
This phase contains 3 steps – These steps are – certificate, client-Key Exchange and certificate verify.

– certificate – optional. It is performed only if the server had requested for the client's digital certificate.

– client Key Exchange – allows the client to send information to the server but in opposite direction.

– certificate verify – It is necessary only if the server had demanded client-Authentication.

Phase 4 – finish –

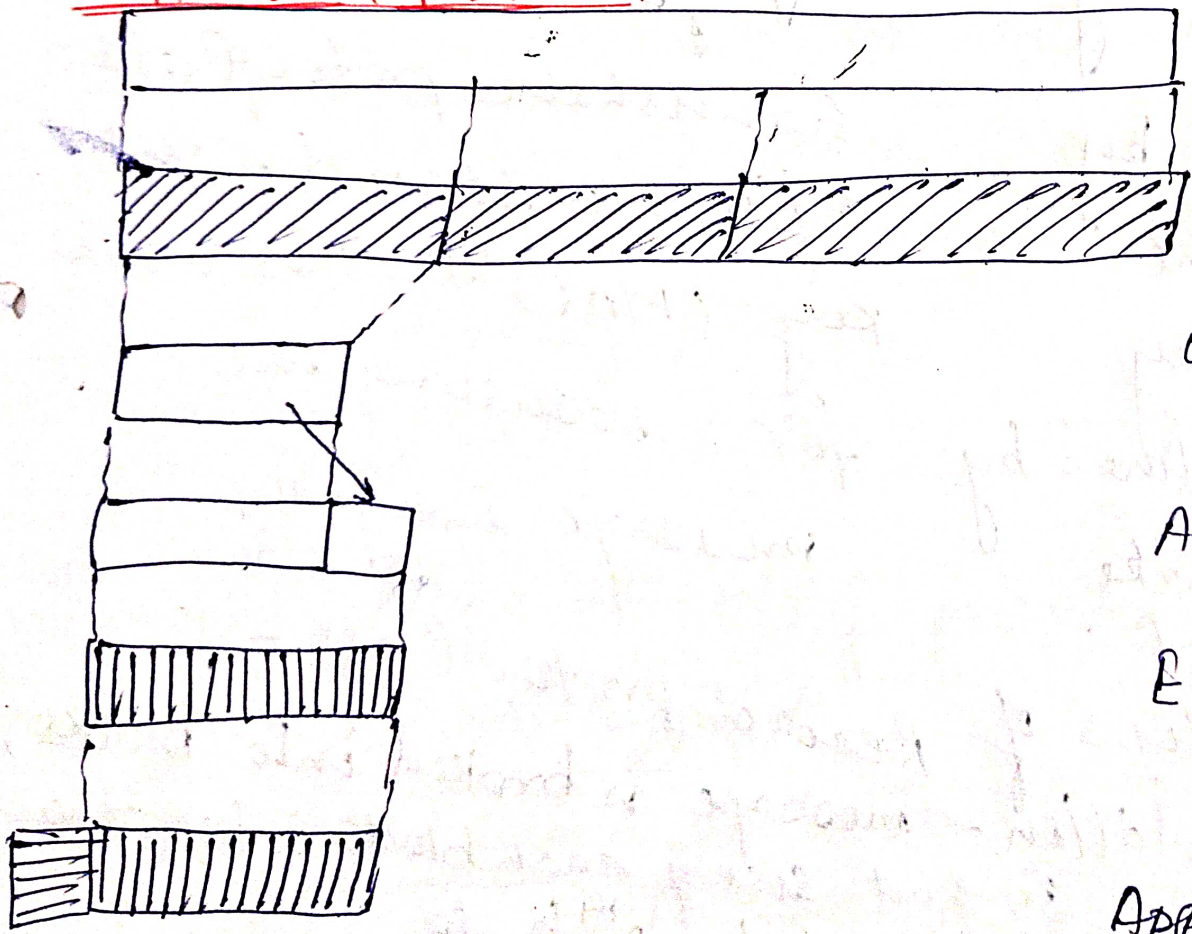


SSL handshake protocol, phase 4. finished

The client initiates this fourth phase of SSL which the server ends. It contains four steps -

- first two messages are from client change cipher spec and finished.
- 2 two messages are from server responds back - change cipher spec, finished.

2 Record Protocol -



SSL record protocol.

SSL comes into picture after a successful handshake is computed b/w the client & server. i.e., after the client and server have optionally authenticated each other and have decided what algorithms to use for secure information exchange.

SSL provides ~~& services~~ -

Confidentiality
↓
achieved by using the secret key that is defined by handshake protocol.

Integrity
↓
handshake protocol also defines a shared secret key (MAC) that is used for assuring the message integrity.

The steps of Record protocol are -

- Fragmentation - message is broken into blocks, so that size of each block is less than or equal to 2^{14} bytes.
- Compression - optional.
- Additional MAC - use shared key ~~to~~ for HMAC.

SSL comes into picture after a successful handshake is computed b/w the client & server. i.e., after the client and server have optionally authenticated each other and have decided what algorithms to use for secure information exchange.

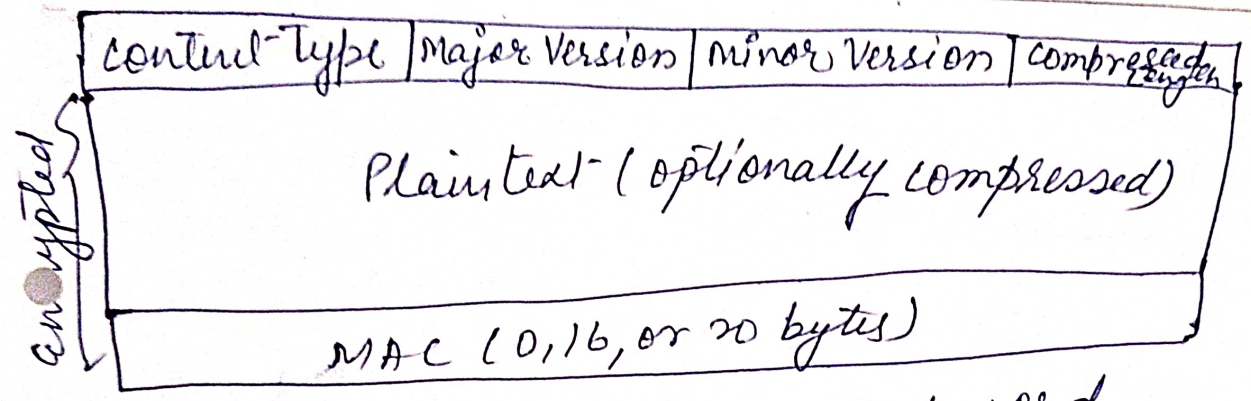
SSL provides 2 services -

Confidentiality
↓
achieved by using the secret key that is defined by handshake protocol.

Integrity
↓
handshake protocol also defines a shared secret key (MAC) that is used for assuring the message integrity.

The steps of Record protocol are -

- Fragmentation - message is broken into blocks, so that size of each block is less than or equal to 2^{14} bytes.
- Compression - optional.
- Additional MAC - use shared key for HMAC.



Final output after SSL record protocol operations.

3- Alert protocol - when the client or the server detects an error, the detecting party sends an alert message to other party.

Severity	Cause
Byte 1	Byte 2

* Encryption — o/p of previous step is encrypted.

← Append header
- content-type (8 bit)
- Major version (8 bit)
- Minor version (8 bit)
- compressed length (16 bit)

If the error is fatal, both the parties immediately close the SSL connection.