CS-552

CYBER FORENSICS

Transport-level and Web Security (SSL / TLS, SSH)

Web Security Considerations

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

The following characteristics of Web usage suggest the need for tailored security tools:

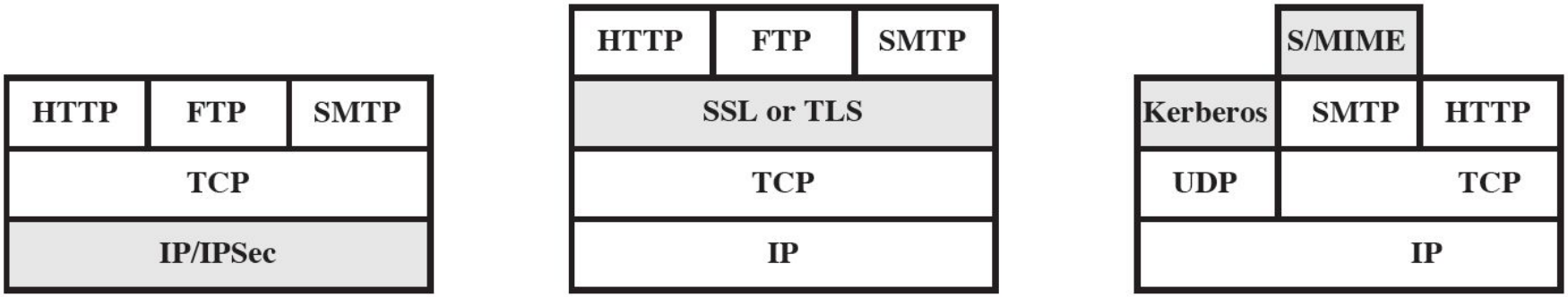
- Web servers are relatively easy to configure and manage
- Web content is increasingly easy to develop
- The underlying software is extraordinarily complex
  - May hide many potential security flaws
- A Web server can be exploited as a launching pad into the corporation's or agency's entire computer complex
- Casual and untrained (in security matters) users are common clien
  Web-based services
  - Such users are not necessarily aware of the security risks that
    have the tools or knowledge to take effective cou

	Threats	Consequences	Countermeasures
Integrity	•Modification of user data •Trojan horse browser •Modification of memory •Modification of message traffic in transit	•Loss of information •Compromise of machine •Vulnerabilty to all other threats	Cryptographic checksums
Confidentiality	•Eavesdropping on the net •Theft of info from server •Theft of data from client •Info about network configuration •Info about which client talks to server	•Loss of information •Loss of privacy	Encryption, Web proxies
Denial of Service	•Killing of user threads •Flooding machine with bogus requests •Filling up disk or memory •Isolating machine by DNS attacks	•Disruptive •Annoying •Prevent user from getting work done	Difficult to prevent
Authentication	•Impersonation of legitimate users •Data forgery	•Misrepresentation of user •Belief that false information is valid	Cryptographic techniques

Table 17.1 A Comparison of Threats on the Web

Where to provide security?

Long-lasting discussion, no ultimate answer



have seen

this lecture

have seen
and will see

SSL (Secure Socket Layer)

originally developed by Netscape

version 3 designed with public input

subsequently Internet standardization effort started at IETF

- TLS (Transport Layer Security) working group established
- TLS can be viewed as SSL v3.1 and compatible with SSL v3

TLS Protocol Stack

- makes use of TCP (reliable end to end data transfer)

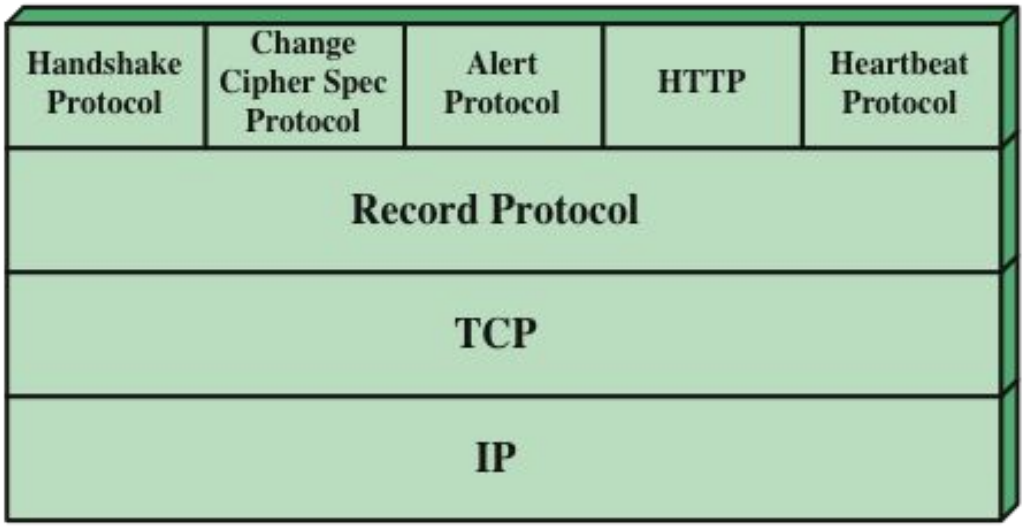
any upper layer protocol

adds security features

- reliable and secure end to end data transfer

TLS is not a single protocol

- two-layers of protocols



Two TLS concepts

TLS session

- an association between client and server
- define a set of cryptographic parameters created by the *Handshake Protocol*
- may be shared by multiple *TLS connections*

TLS connection

- a transient, peer-to-peer, secure communication link
- associated with (derived from) a TLS session
- A session is used several times to create connections

Session vs. Connection

- *Sessions* are used to avoid expensive negotiation for crypto parameters for each *connection*

Both are characterized by several parameters

- that define a *session state* or *connection state*

Session state parameters

Session identifier
- chosen by server

Peer certificate
- certificate of the peer entity (server's if the entity is client, client's if the entity is server)
- may be null (which is the likely case for server)

Compression method (to be deprecated in TLS v1.3)
- algorithm used for compression

Cipher Spec
- bulk data encryption algorithm (AES, etc.) - may be null (rarely)
- hash algorithm used in MAC calculation (MD5 or SHA-1 or SHA-2)

Master Secret
- 48-bytes *secret* shared between client and server

Is resumable
- a flag that is set if the session can be used later for new connections

Connection State Parameters

Random numbers
- server and client exchange
- used as nonces during key exchange

MAC secret
- secret key used for MAC operations
- Separate for server and client

conventional encryption keys
- Separate for server and client

initialization vector
- if CBC mode is used

TLS Record Protocol

serves to TLS connections

- uses connection parameters

provides confidentiality and integrity

also fragments (into 2^{14} bytes chunks)

optionally compresses data (in practice, no compression and this is deprecated in the last version of TLS)

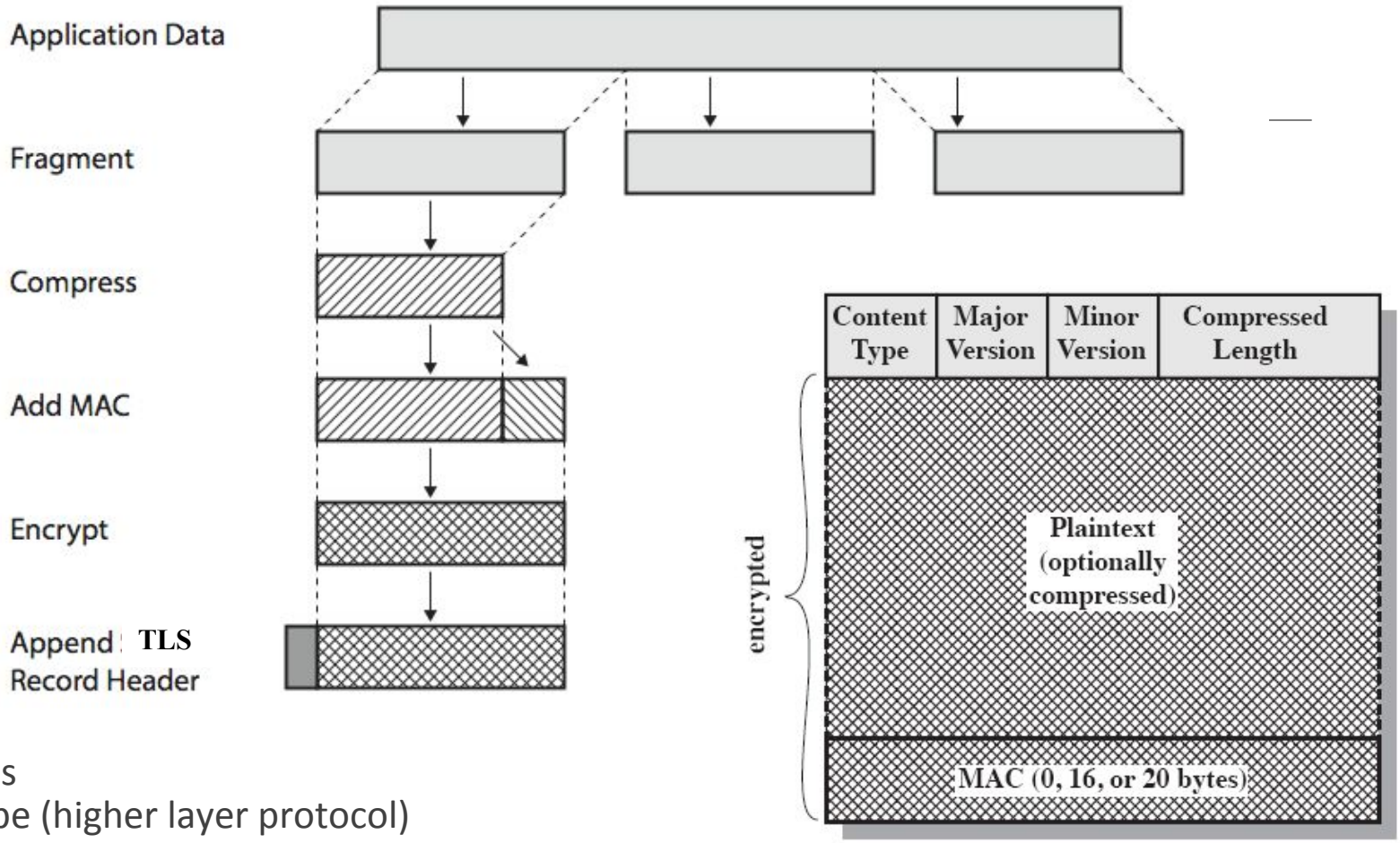
confidentiality

- AES, IDEA, DES, 3DES, RC4, etc.

message integrity

- using HMAC with shared secret key
- In SSL, the method was similar to HMAC but pads were concatenated rather than XORed

TLS Record Protocol



header fields
- content type (higher layer protocol)
  - change_cipher_spec, alert, handshake, applicatio...
- version
- compressed length (or plaintext length if no compression) of the fragment

Change Cipher Spec Protocol

very simple protocol

the new state established by the handshake protocol is a *pending* state

- that is, it is not yet valid

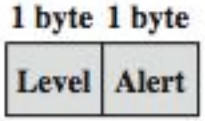
change cipher spec protocol (actually a single command exchanged between client and server) makes this pending state the current one

- connection parameters change after that

will see its use in the handshake protocol

Record Protocol Payload

Alert Protocol



Record Protocol Payload

conveys TLS-alerts to peer entity

secured using the record protocol (if any)

- and with current connection state parameters

each message is two bytes

- one byte for level (severity)
  - warning (connection may resume) or fatal (connection is terminated)
- one byte for the alert code
  - unexpected message, bad record MAC, decompression failure
  - handshake failure (no common ground), illegal parameters (inconsistent or unrecognized parameters)
  - no certificate, bad certificate, unsupported certificate, certificate revoked, certificate expired, certificate unknown

Handshake Protocol

~~The most complex part of TLS~~

Allows server and client:

- to authenticate each other
- to negotiate encryption and MAC algorithms
- to create cryptographic keys to be used
- in general, to establish a *session* and then a *connection*

handshake is done before any data is transmitted

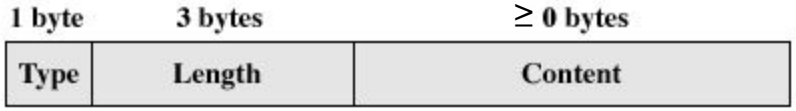
- so cannot assume a secure record protocol

Handshake Protocol

a series of messages in 4 phases

1. ~~Establish Security Capabilities~~
2. Server Authentication and Key Exchange
3. Client Authentication and Key Exchange
4. Finish

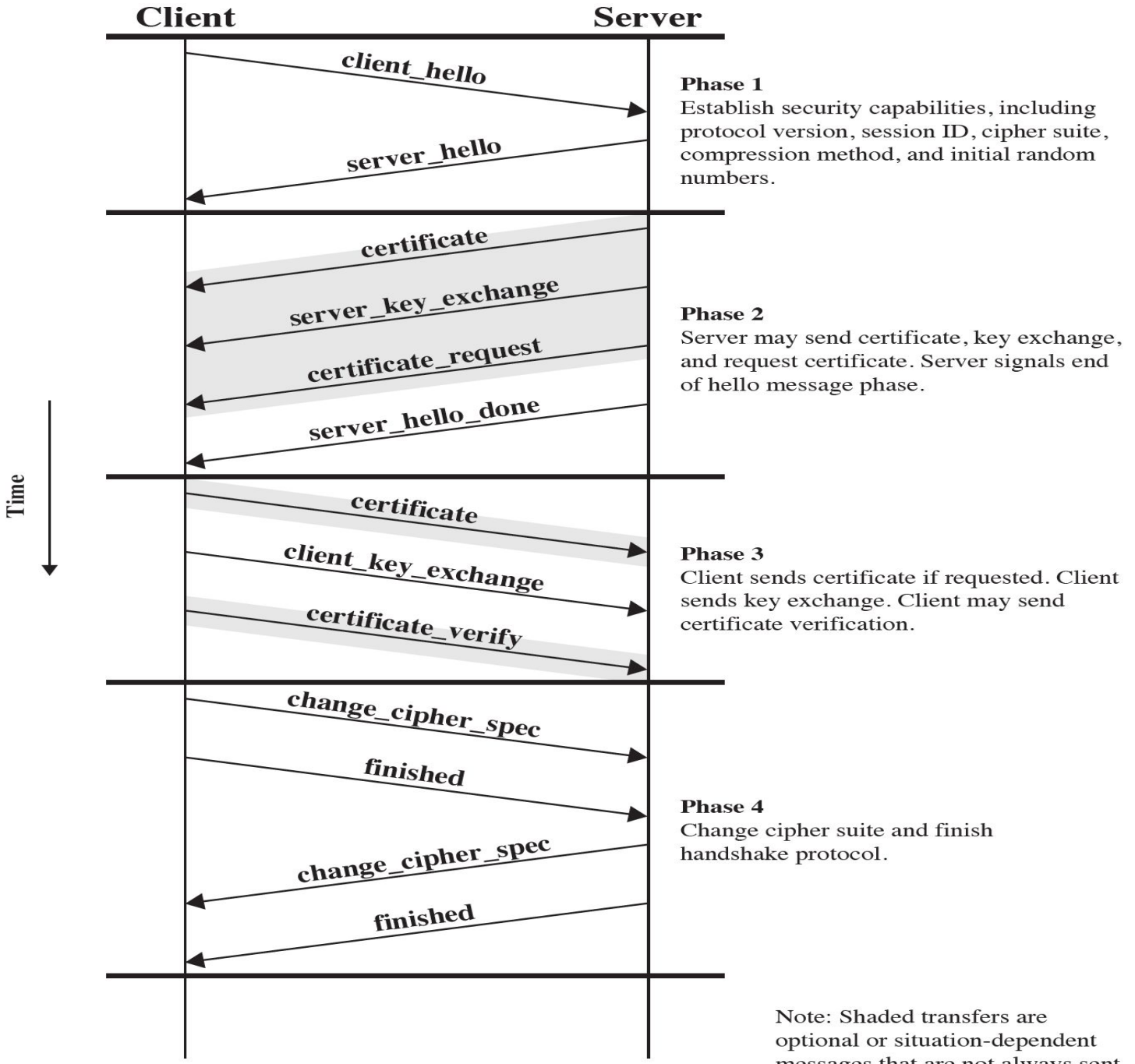
Handshake message format



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

Handshake Protocol



Handshake Phase 1 – Establish Security Capabilities

Client Hello (a list of client's preferences)

- version: highest TLS version supported by client
- client's random
  - also includes a timestamp
  - are used during key exchange to prevent replay attacks
- session ID
  - nonzero means client wants to use an existing session state for a new connection state (abbreviated handshake); zero means new connection on a new session
- compression methods supported by client
- Cipher Suite
  - a list that contains the combination of crypto algorithms supported by the client in the order of preference
  - each entry is a key exchange algorithm and a cipher spec

Handshake Phase 1 – Establish Security Capabilities

Server Hello (response to client's requests)

- version: version proposed by client if also supported by server; otherwise highest supported by server
- server's random
  - same structure as client's, but independent of client's random
- session ID
  - if client offered one and it is also supported by server, then the same ID (abbreviated handshake)
  - otherwise a new ID assigned by server
- compression methods chosen from the client's list
- Cipher Suite selected from the client's list

Key exchange methods

How the conventional encryption and HMAC keys are exchanged?

- actually first *pre-master secret* is exchanged using public key crypto
- *master secret* is derived from pre-master secret
  - At this point *session* is created
- other keys are derived from the master secret
  - At this point *connection* is created

Key exchange methods – cont'd

Rephrase question: how is the pre-master secret exchanged?

- RSA
  - server provides its RSA certificate (that has the server's public key), client encrypts the pre-master secret using server's public key and sends it
- Fixed Diffie-Hellman (DH)
  - Server and client DH public keys are fixed and sent in certificates
- Ephemeral DH
  - server and client certificates contain RSA or DSS public keys
  - server creates DH parameters (used one-time) and signs by its private key. For client, see later.
- Anonymous DH
  - no certificates, no authentication, just send out DH parameters
  - vulnerable to man-in-the-middle-attacks

Some Cipher Specs Fields

Cipher algorithm and key size

- RC4, RC2, DES, 3DES, DES40 (40-bit DES), IDEA, AES, etc.

Hash algorithm for HMAC

- MD5 or SHA-1 or others

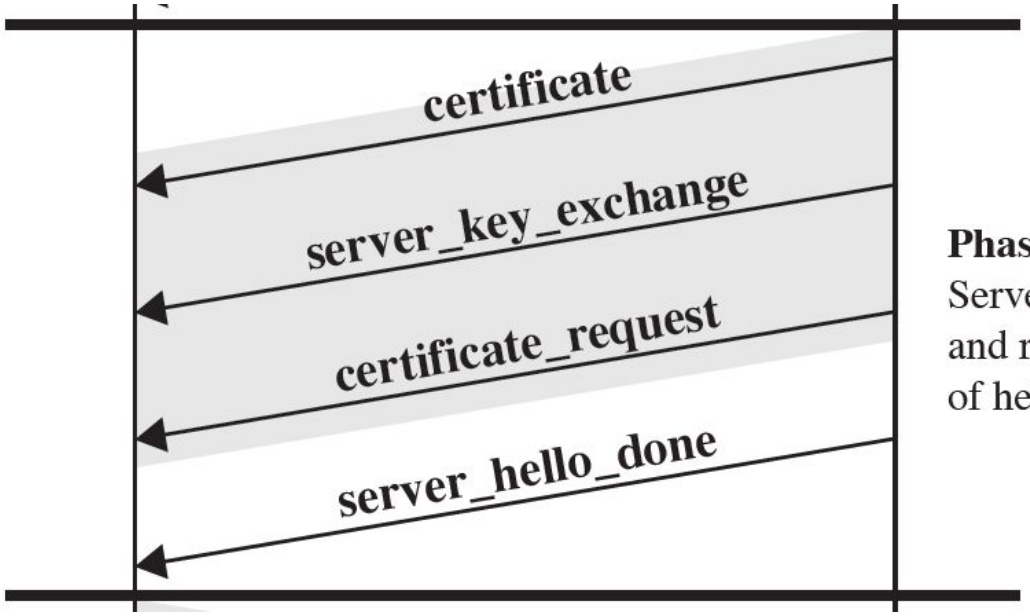
Cipher type

- stream or block

IV size

- size of the init. vector for CBC mode

Handshake Phase 2: Server Auth. and Key Exchange



Phase 2
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.

Certificate is needed except for anon-DH (anon-DH is not used most of the time in anyway)

- Certificate is the basis for server authentication
- if fixed DH, then certificate contains enough information for key exchange (so server key exchange message is not needed)

Handshake Phase 2: Server Auth. and Key Exchange

Server Key Exchange

- not needed for
  - fixed DH and RSA key exchange
- message content depends on the key exchange method agreed
  - Anon-DH
    - message contains public DH parameters and server's DH public key
  - Ephemeral DH
    - same as anon-DH plus a signature on them
- Signatures contain random values (that are exchanged in hello phase) to resist against replay attacks

Handshake Phase 2: Server Auth. and Key Exchange

Certificate Request Message

- although not common in practice, server may request client to send its certificate
  - to authenticate the client
- two fields: certificate type and acceptable CAs
  - a list of them
- Certificate types
  - fixed DH (certificate may be signed with RSA or DSS)
  - signature only
    - Certificate may contain RSA or DSS key
    - Used for client authentication (see Phase 3)
    - Also used for signing ephemeral DH parameters

Server Hello Done message

- server is finished and will wait for client's response

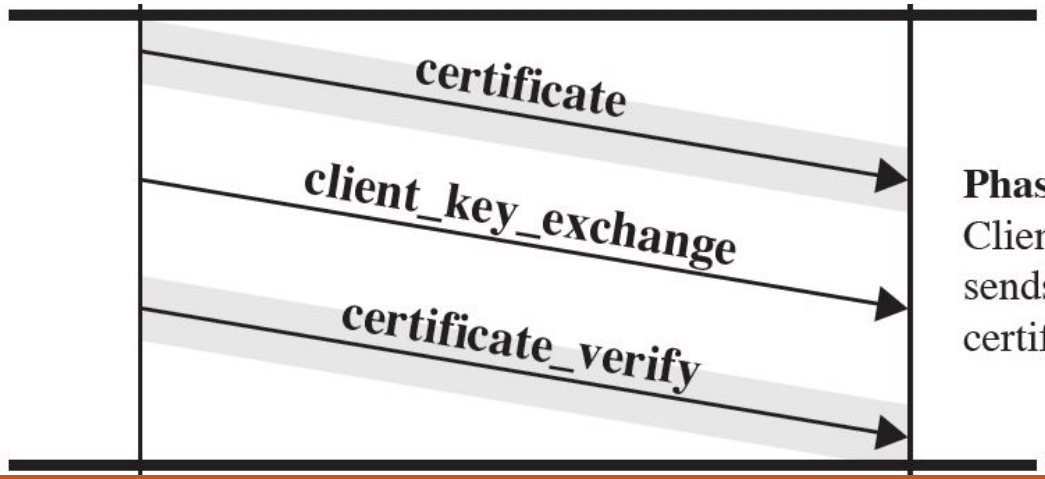
Handshake Phase 3: Client Auth. and Key Exchange

Upon receipt of server hello done

- client checks the server certificate and server hello parameters
- after that client starts sending its own messages

Client's Certificate

- is sent if requested and available



Phase 3
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.

Handshake Phase 3: Client Auth. and Key Exchange

Client Key exchange message

- content depends on the key exchange method agreed
- RSA
  - *48-byte pre-master secret* is encrypted using server's RSA key (obtained at phase 2)
- fixed-DH
  - client DH parameters are in client certificate, so key exchange message is null
- Anon or ephemeral DH
  - Client DH parameters and DH public key are sent
  - no signature even for ephemeral DH
- no client authentication and authenticated key exchange so far (only key exchange)

Handshake Phase 3: Client Auth. and Key Exchange

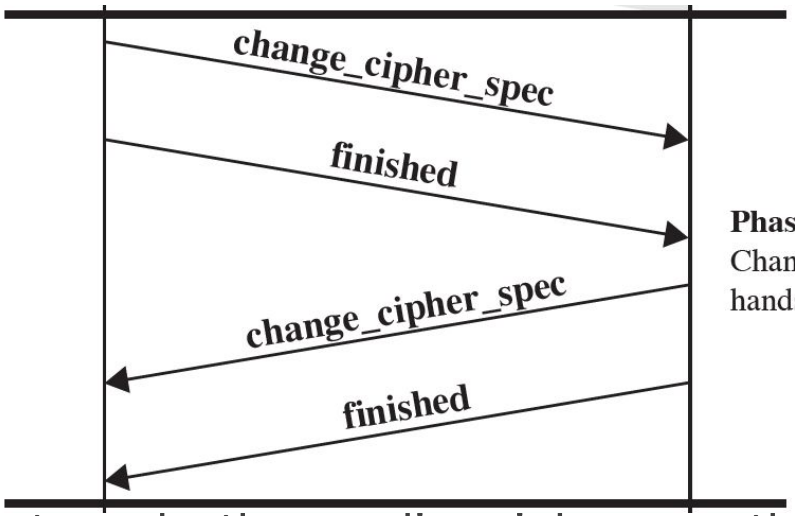
Certificate Verify message

- client is not authenticated via client key exchange message
  - anyone could send the key exchange message
- the method for authentication is via the *certificate verify* message
  - client shows ownership of private key that corresponds to the public key in client certificate by signing a hash that contains the master secret and handshake messages
  - except for fixed DH that does not contain a signature key
- what about authentication for fixed DH case?
  - no real authentication but the attacker cannot produce the pre-master and master secrets since it does not know the DH private key (so there is a kind of an implied auth.)

Handshake Phase 4: Finish

- At the end of Phase 3, both client and server can calculate keys

Phase 4 is wrap-up phase



Phase 4
Change cipher suite and finish handshake protocol.

(

- to make the pending cipher spec the current one

Handshake Phase 4: Finish

Finished message

- a MAC on exchanged handshake messages using the master secret
- to verify that handshake is successful and both parties have the same master secret
- client's *finished* is verified by server and vice versa
- the connection state of the record protocol that encrypts and MACs the finished message is the new one
  - so this is also verification of all the keys that are recently created

Master Secret Creation

Master Secret

- 48-byte value generated for a session

two stage creation

- pre-master secret is exchanged during handshake
  - if RSA, client creates, encrypts and sends; server decrypts
  - if DH, both calculates the same secret which is the pre-master secret
- master secret is calculated using pre-master secret and random nonces exchanged during handshake
  - by using a PRF (Pseudo Random Function)
  - Several levels of HMACs until 48 bytes have been generated

Generation of cryptographic parameters

Encryption keys, MAC secrets, IV are to be generated from a key block obtained from the master secret

- again using a PRF based on several levels of HMACs
  - master secret is the key
  - random nonces are also used
- variable length output, until enough output has been generated.

Abbreviated Handshake

We have said that a particular TLS session can serve to different TLS connections

- Abbreviated handshake is the mechanism for that

During phase-1 if both parties agree on using an existing session ID, then that means they run abbreviated handshake

- that session's *master secret* (which is already stored) and new random nonces exchanged during phase-1 are directly used for the generation of the keys
  - Thus, phase 2 and 3 are skipped (that is why it is abbreviated)
  - Phase 4 is the same as full handshake

Heartbeat Protocol

Relatively a new extension of TLS

- In 2012 (RFC 6250)

In the context of computer networks, a heartbeat is a periodic signal or messages to probe the availability of the other party.

Heartbeat protocol runs on top of TLS protocol (part of upper layer of TLS)

- Simple periodic request-response pairs sent with a random payload (same payload in both request and response)
  - Helps request-response matching if used over UDP, since UDP is connectionless

Two purposes of HeartBeat Protocol

- Sender assures that the recipient is still alive
- Even if the real communication is idle, it generates traffic such that the firewalls, which do not tolerate idle times, do not cause connection closures

SSL/TLS Attacks

The attacks can be grouped into four general categories:

- Attacks on the handshake protocol
- Attacks on the record and application data protocols
- Attacks on the PKI
- Other attacks

- The constant back-and-forth between threats and countermeasures determines the evolution of Internet-based protocols

TLSv1.3

Primary aim is to improve the security of TLS

Significant changes from version 1.2 are:

- TLSv1.3 removes support for a number of options and functions
  - Deleted items include:
    - Compression
    - Ciphers that do not offer authenticated encryption
    - Static RSA and DH key exchange
    - 32-bit timestamp as part of the Random parameter in the client_hello message
    - Renegotiation
    - Change Cipher Spec Protocol
    - RC4
    - Use of MD5 and SHA-224 hashes with signatures
- TLSv1.3 uses Diffie-Hellman or Elleptic Curve Diffie-Hellman for key exchange and does not permit RSA
- TLSv1.3 allows for a “1 round trip time” handshake by changing the order of message sent with establishing a secure connection

HTTPS (HTTP over SSL)

Refers to the combination of HTTP and SSL to implement secure communication between a Web browser and a Web server

The HTTPS capability is built into all modern Web browsers

A user of a Web browser will see URL addresses that begin with https:// rather than http://

If HTTPS is specified, port 443 is used, which invokes SSL

Documented in RFC 2818, *HTTP Over TLS*

- There is no fundamental change in using HTTP over either SSL or TLS and both implementations are referred to as HTTPS

When HTTPS is used, the following elements of the communication are encrypted:

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

Connection Initiation

For HTTPS, the agent acting as the HTTP client also acts as the TLS client

The client initiates a connection to the server on the appropriate port and then sends the TLS ClientHello to begin the TLS handshake

When the TLS handshake has finished, the client may then initiate the first HTTP request

All HTTP data is to be sent as TLS application data

There are three levels of awareness of a connection in HTTPS:

At the HTTP level, an HTTP client requests a connection to an HTTP server by sending a connection request to the next lowest layer

- Typically the next lowest layer is TCP, but is may also be TLS/SSL

At the level of TLS, a session is established between a TLS client and a TLS server

- This session can support one or more connections at any time

A TLS request to establish a connection begins with the establishment of a TCP connection between the TCP entity on the client side and the TCP entity on the server side

Connection Closure

An HTTP client or server can indicate the closing of a connection by including the line `Connection: close` in an HTTP record

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

TLS implementations must initiate an exchange of closure alerts before closing a connection

- A TLS implementation may, after sending a closure alert, close the connection without waiting for the peer to send its closure alert, generating an “incomplete close”

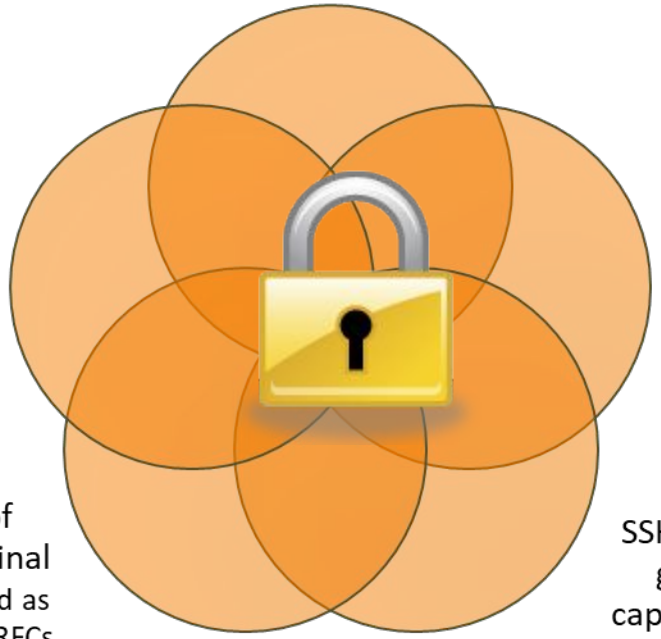
An unannounced TCP closure could be evidence of some sort of attack so the HTTPS client should issue some sort of security warning when this occurs

Secure Shell (SSH)

A protocol for secure network communications designed to be relatively simple and inexpensive to implement

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

- Has become the method of choice for remote login and X tunneling
- Is rapidly becoming one of the most pervasive applications for encryption technology outside of embedded systems



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

SSH2 fixes a number of security flaws in the original scheme and is documented as a proposed standard in IETF RFCs 4250 through 4256

SSH also provides a more general client/server capability and can be used for such network functions as file transfer and e-mail

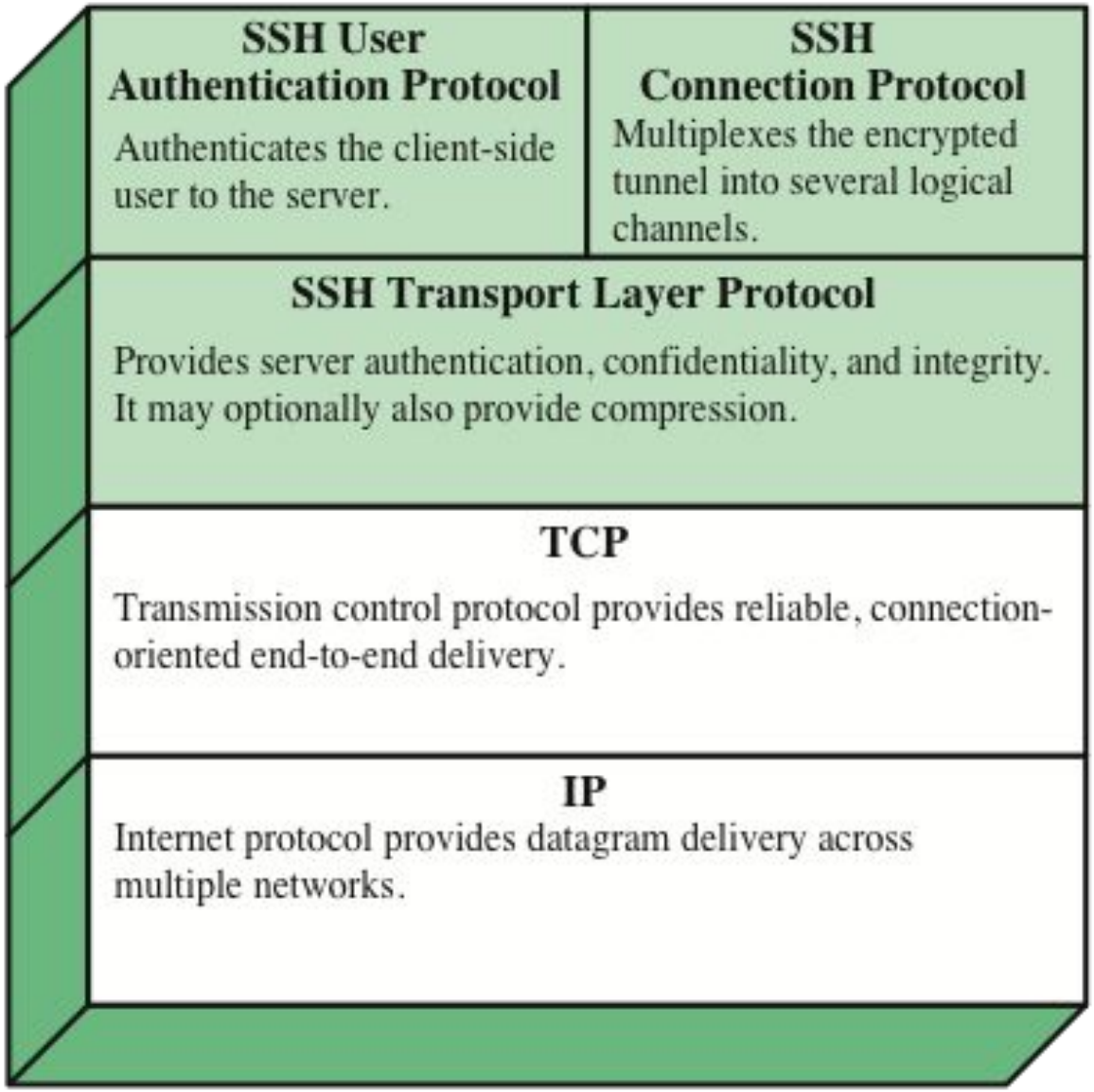


Figure 17.8 SSH Protocol Stack

Transport Layer Protocol

- 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
- The server host key is used during key exchange to authenticate the identity of the host
- RFC 4251 dictates two alternative trust models:
  - The client has a local database that associates each host name with the corresponding public host key
  - 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

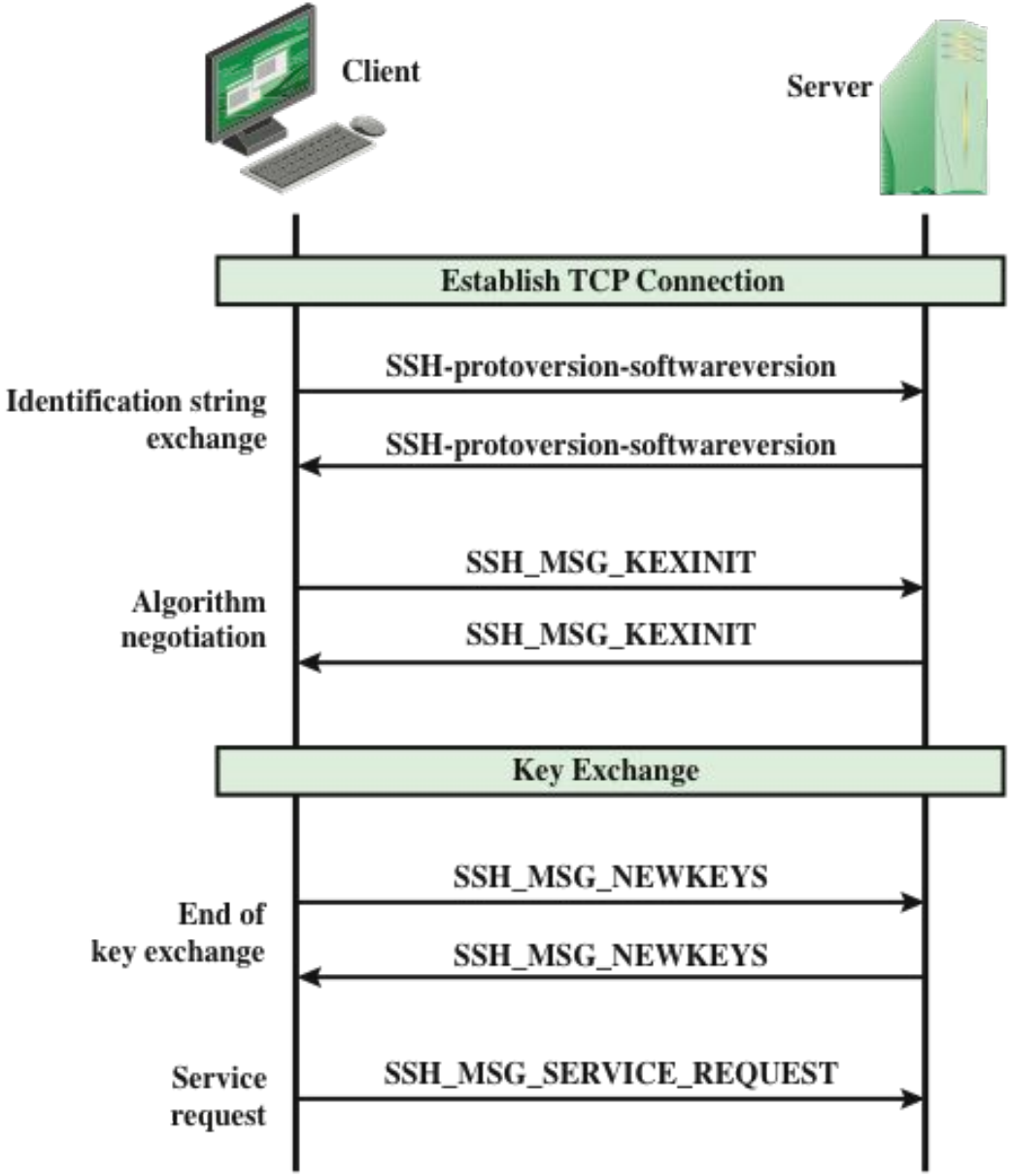


Figure 17.9 SSH Transport Layer Protocol Packet Exchanges

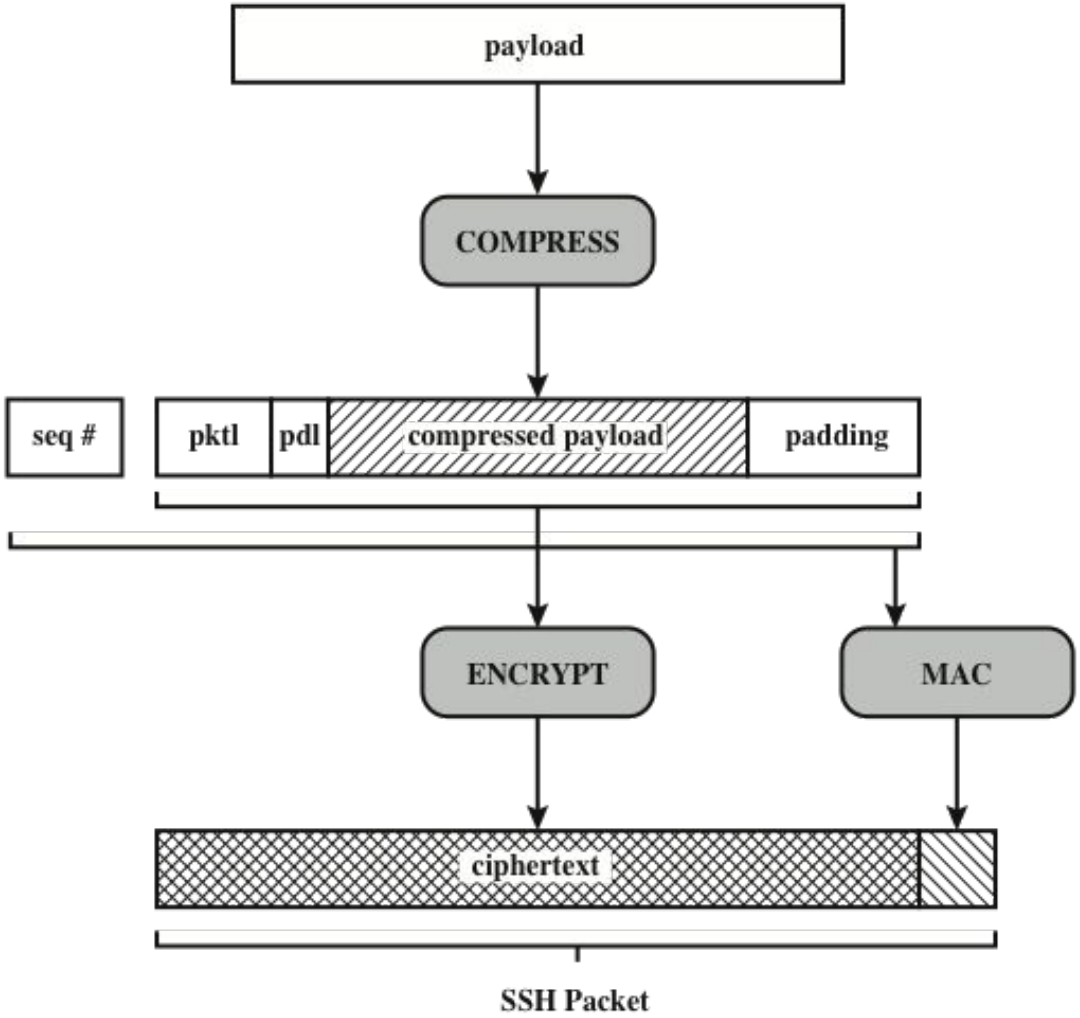


Figure 17.10 SSH Transport Layer Protocol Packet Formation

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-MD5; digest length = key length = 16
hmac-md5-96	First 96 bits of HMAC-MD5; digest length = 12; key length = 16

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

* = Required
** = Recommended

Table 17.3

SSH

Transport

Layer

Cryptographic

Algorithms

Authentication Methods

Publickey

- 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
- This method works by having the client send a signature created with the private key of the client host
- Rather than directly verifying the user's identity, the SSH server verifies the identity of the client host

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

- The 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 are supported using separate channels
- Either side may open a channel
- For each channel, each side associates a unique channel number
- 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

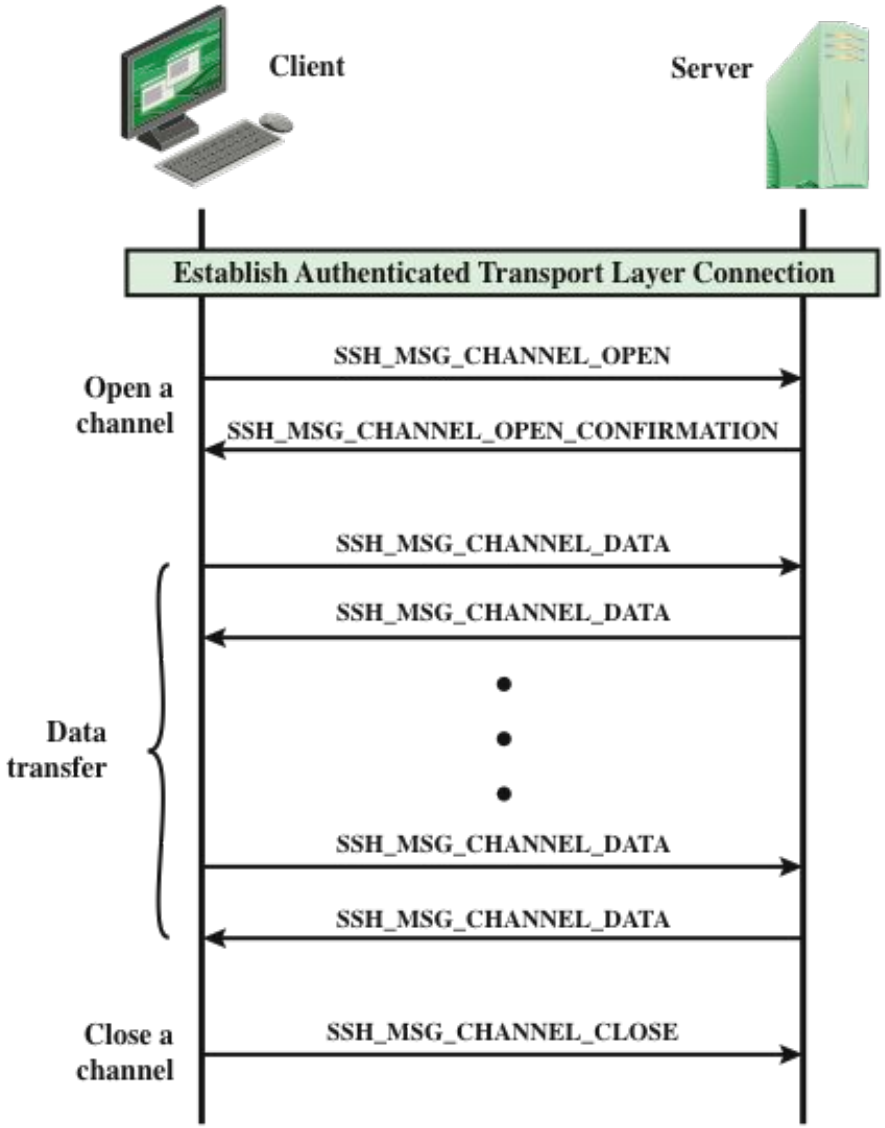


Figure 17.11 Example SSH Connection Protocol Message Exchange

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

X11

- Refers to the X Window System, a computer software system and network 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

- Remote port forwarding

Direct-tcpip

- Local port forwarding

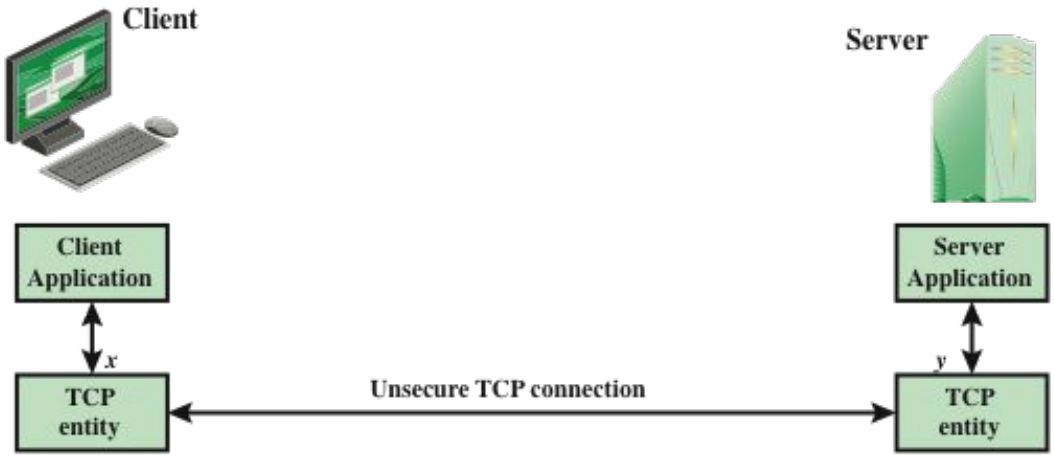
Port Forwarding

One of the most useful features of SSH

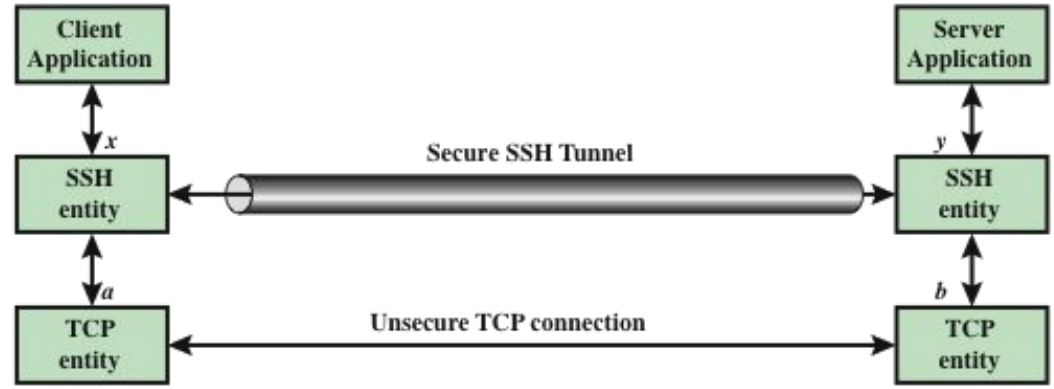
Provides the ability to convert any insecure TCP connection into a secure SSH connection (also referred to as SSH tunneling)

Incoming TCP traffic is delivered to the appropriate application on the basis of the port number (a port is an identifier of a user of TCP)

An application may employ multiple port numbers



(a) Connection via TCP

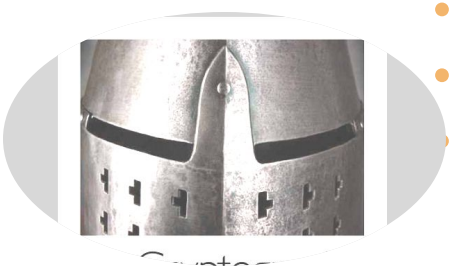


(b) Connection via SSH Tunnel

Figure 17.12 SSH Transport Layer Packet Exchanges

Summary

- Web security considerations
  - Web security threats
  - Web traffic security approaches
- Secure sockets layer
  - SSL architecture
  - SSL record protocol
  - Change cipher spec protocol
  - Alert protocol
  - Handshake protocol
  - Cryptographic computations
  - Heartbeat protocol
  - SSL/TLS attacks
  - TLSv1.3



- Secure shell (SSH)
- Transport layer protocol
- User authentication protocol
- Communication protocol

- HTTPS
- Connection initiation
- Connection closure