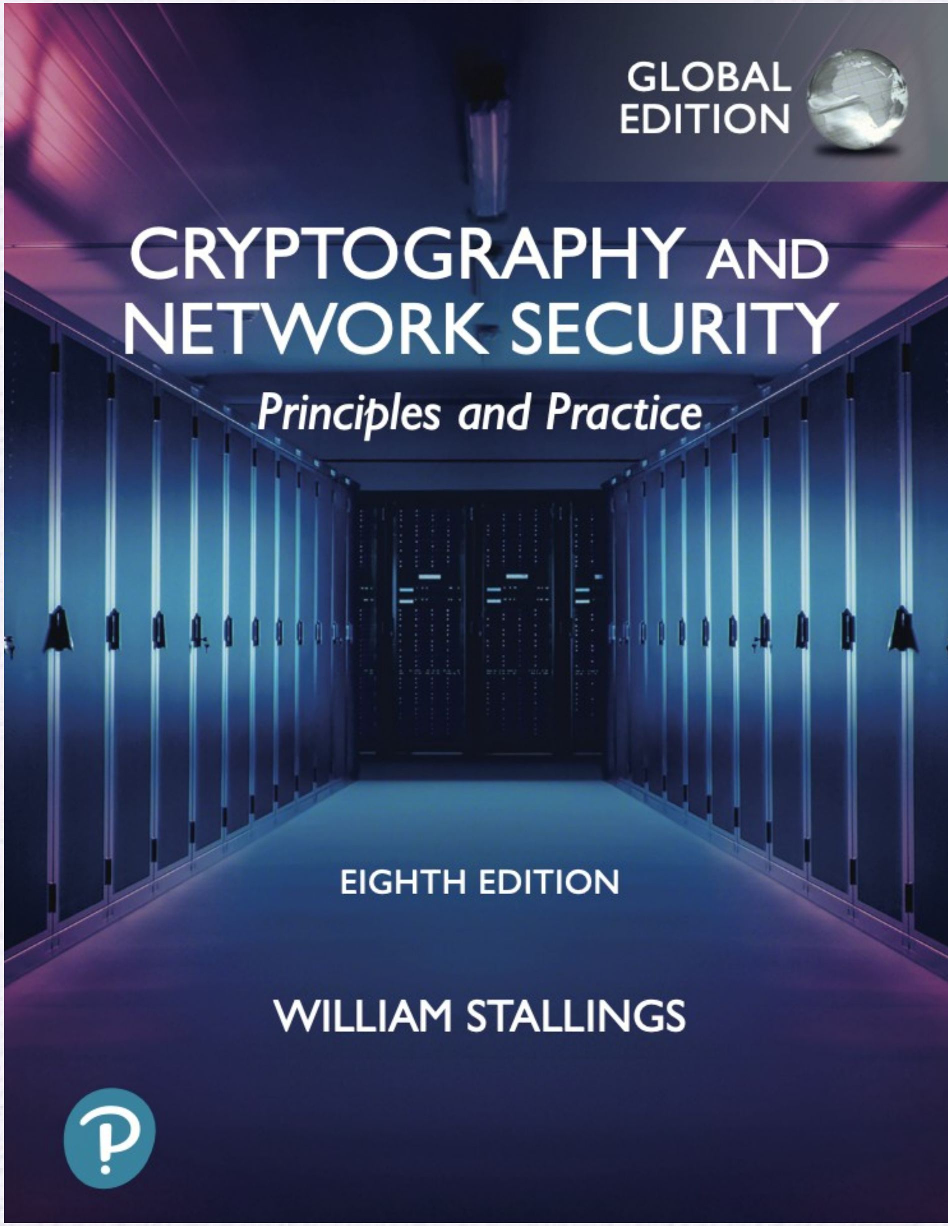
University of Nevada – Reno
Computer Science & Engineering Department

CS454/654 Reliability and Security of Computing Systems - Fall 2024

Lecture 23

Dr. Batyr Charyyev
bcharyyev.com



CHAPTER 17

TRANSPORT-LEVEL SECURITY

Web Security Considerations

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

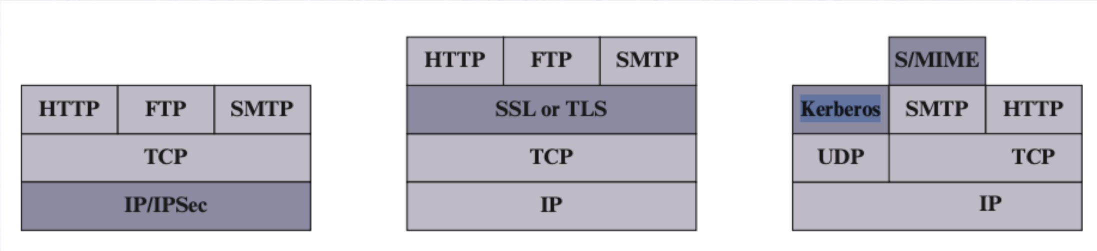
- Overall
  - Web servers are relatively easy to configure and manage.
  - Web content is increasingly easy to develop.

- However
  - Web server as an entry point to organization's internal network.
    - SQL injections, remote code execution.
  - Users are everyday users and do not have technical expertise.
    - Lack of security awareness.
      - Phishing attacks, weak passwords, insecure browsing.

- Way to group these threats
  - Passive and Active attacks.
    - Passive: monitoring network traffic between browser and server
    - Active: impersonating either browser, or server, altering message, etc.
  - In terms of the location of the threat: Web server, Web browser, and network traffic between browser and server.

Table 17.1 A Comparison of Threats on the Web

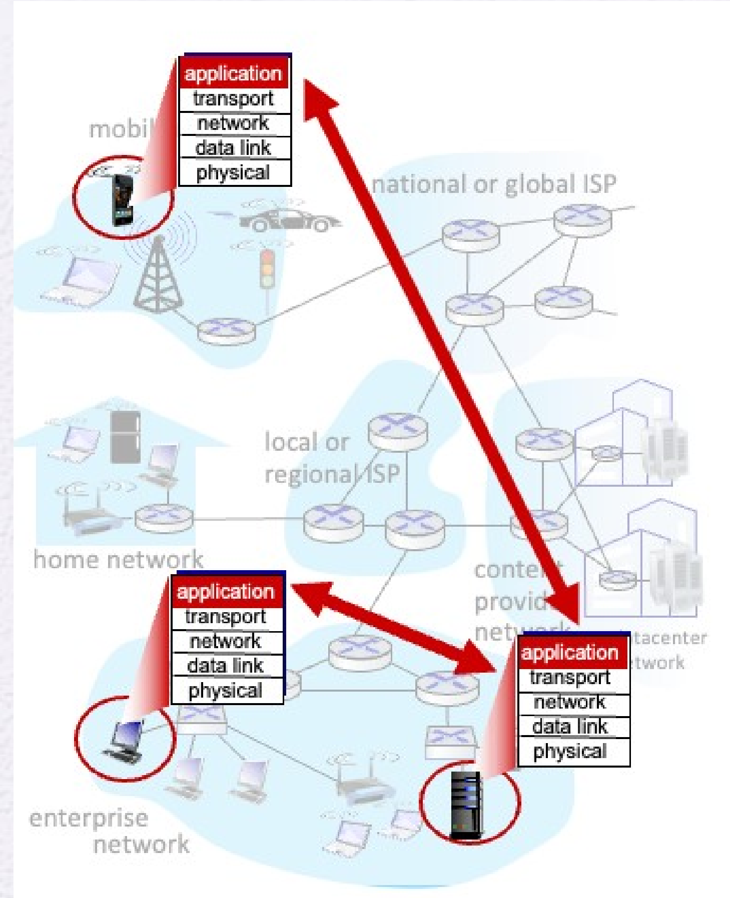
	Threats	Consequences	Countermeasures
Integrity	• Modification of user data • Trojan horse browser • Modification of memory • Modification of message traffic in transit	• Loss of information • Compromise of machine • Vulnerability to all other threats	Cryptographic checksums
Confidentiality	• Eavesdropping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

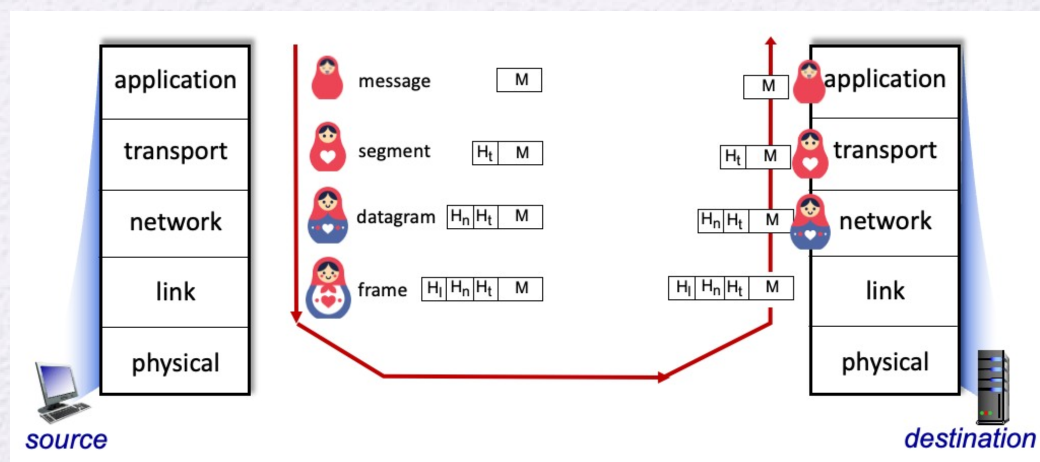


(a) Network level (b) Transport level (c) Application level

HTTPS
SSH: Secure Shell

SSL: Secure Sockets Layer SSL
TLS: Transport Layer Security





Transport Layer Security (TLS)

- TLS: Transport Layer Security protocol
- Evolved from SSL, and SSL is not used anymore
- TLS: RFC 5246
  - https://datatracker.ietf.org/doc/html/rfc5246
- Rely on TCP.
- Provides service to HTTP
  - Hypertext Transfer Protocol

```
Network Working Group                                          T. Dierks
Request for Comments: 5246                                   Independent
Obsoletes: 3268, 4346, 4366                                  E. Rescorla
Updates: 4492                                                RTFM, Inc.
Category: Standards Track                                    August 2008

              The Transport Layer Security (TLS) Protocol
                              Version 1.2

Status of This Memo

   This document specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements.  Please refer to the current edition of the "Internet
   Official Protocol Standards" (STD 1) for the standardization state
   and status of this protocol.  Distribution of this memo is unlimited.

Abstract

   This document specifies Version 1.2 of the Transport Layer Security
   (TLS) protocol.  The TLS protocol provides communications security
   over the Internet.  The protocol allows client/server applications to
   communicate in a way that is designed to prevent eavesdropping,
   tampering, or message forgery.

Table of Contents

   1. Introduction ....................................................4
      1.1. Requirements Terminology ...................................5
      1.2. Major Differences from TLS 1.1 .............................5
   2. Goals ...........................................................6
   3. Goals of This Document ..........................................7
   4. Presentation Language ...........................................7
      4.1. Basic Block Size ...........................................7
      4.2. Miscellaneous ..............................................8
      4.3. Vectors ....................................................8
      4.4. Numbers ....................................................9
      4.5. Enumerateds ................................................9
      4.6. Constructed Types .........................................10
           4.6.1. Variants ...........................................10
      4.7. Cryptographic Attributes ..................................12
      4.8. Constants .................................................14
   5. HMAC and the Pseudorandom Function .............................14
   6. The TLS Record Protocol ........................................15
      6.1. Connection States .........................................16
      6.2. Record Layer ..............................................19
           6.2.1. Fragmentation ......................................19
```

Transport Layer Security (TLS)

- Two important TLS concepts are the TLS session and the TLS connection

TLS session
- Established during handshake phase.
- Can be reused to establish new connection.
- Persist until explicitly terminated or timed out.
- Stores cryptographic parameters for connections.

TLS connection
- Single peer to peer communication. GET request to URL
- Uses parameters provided by the session, to enable secure transport of data.
- Temporary, typically lasts for the duration of data transfer, and can not be reused one closed.

A session state is defined by the following parameters:

Session identifier

An arbitrary byte sequence chosen by the server to identify an active or resumable session state

Peer certificate

An X509.v3 certificate of the peer. Used to verify the identity.

Compression method

The algorithm used to compress data prior to encryption

Cipher spec

Specifies the bulk data encryption algorithm and a hash algorithm used for MAC calculation; also defines cryptographic attributes such as the hash_size

Master secret

48-byte secret shared between the client and the server. Used to generate session keys.

Is resumable

A flag indicating whether the session can be used to initiate new connections

A connection state is defined by the following parameters:

Server and client random

- Byte sequences that are chosen by the server and client for each connection. Used for key generation and prevent replay attack.

Server write MAC secret

- The secret key used in MAC operations on data sent by the server

Client write MAC secret

- The secret key used in MAC operations on data sent by the client

Server write key

- The secret encryption key for data encrypted by the server and decrypted by the client

Client write key

- The symmetric encryption key for data encrypted by the client and decrypted by the server

Initialization vectors

- This field is first initialized by the TLS Handshake Protocol
- When a block cipher in Cipher Block Chaining mode is used, an initialization vector (IV) is maintained for each key

Sequence numbers

- Each party maintains separate sequence numbers for transmitted and received messages for each connection
- When a party sends or receives a change cipher spec message, the appropriate sequence number is set to zero
- Sequence numbers may not exceed $2^{64} - 1$ (64 bits)

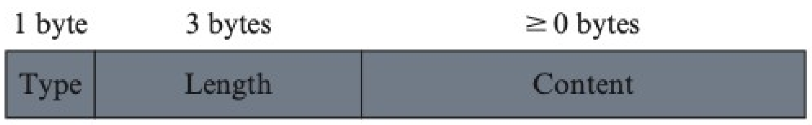
TLS Handshake

TLS messages Types

- Handshake
- Change_cipher_spec
- Application_data
- Alert

TLS Handshake

- 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 over TLS.
- All the messages exchanged in TLS Handshake is of this form.



(c) Handshake Protocol

TLS Handshake

(c) Handshake Protocol

- **Type** (1 byte): Type of the message (10 message types, listed in table).
- **Length** (3 bytes): The length of the message in bytes.
- **Content** (>= 0 bytes): The parameters associated with the message (for each message associated parameters are listed in table)

Table 17.2 TLS Handshake Protocol Message Types

Message Type	Parameters
`hello_request`	null
`client_hello`	version, random, session id, cipher suite, compression method
`server_hello`	version, random, session id, cipher suite, compression method
`certificate`	chain of X.509v3 certificates
`server_key_exchange`	parameters, signature
`certificate_request`	type, authorities
`server_done`	null
`certificate_verify`	signature
`client_key_exchange`	parameters, signature
`finished`	hash value

Phase – 1
Initiated by client, which sends client_hello message.

Inside client_hello

- **Version:** version of TLS (highest that client knows)
- **Random:** 28-bit timestamp, and random 28 bytes generated by random number generator. Used as Nonces.
- **Session ID:** non-zero value indicates client wants to update parameters of existing connection or create new connection on this session. Zero value indicates client wishes to create new connection or new session.
- **CipherSuite:** list of cryptographic algorithms supported by client, in decreasing order of preference.
- **Compression Method:** list of

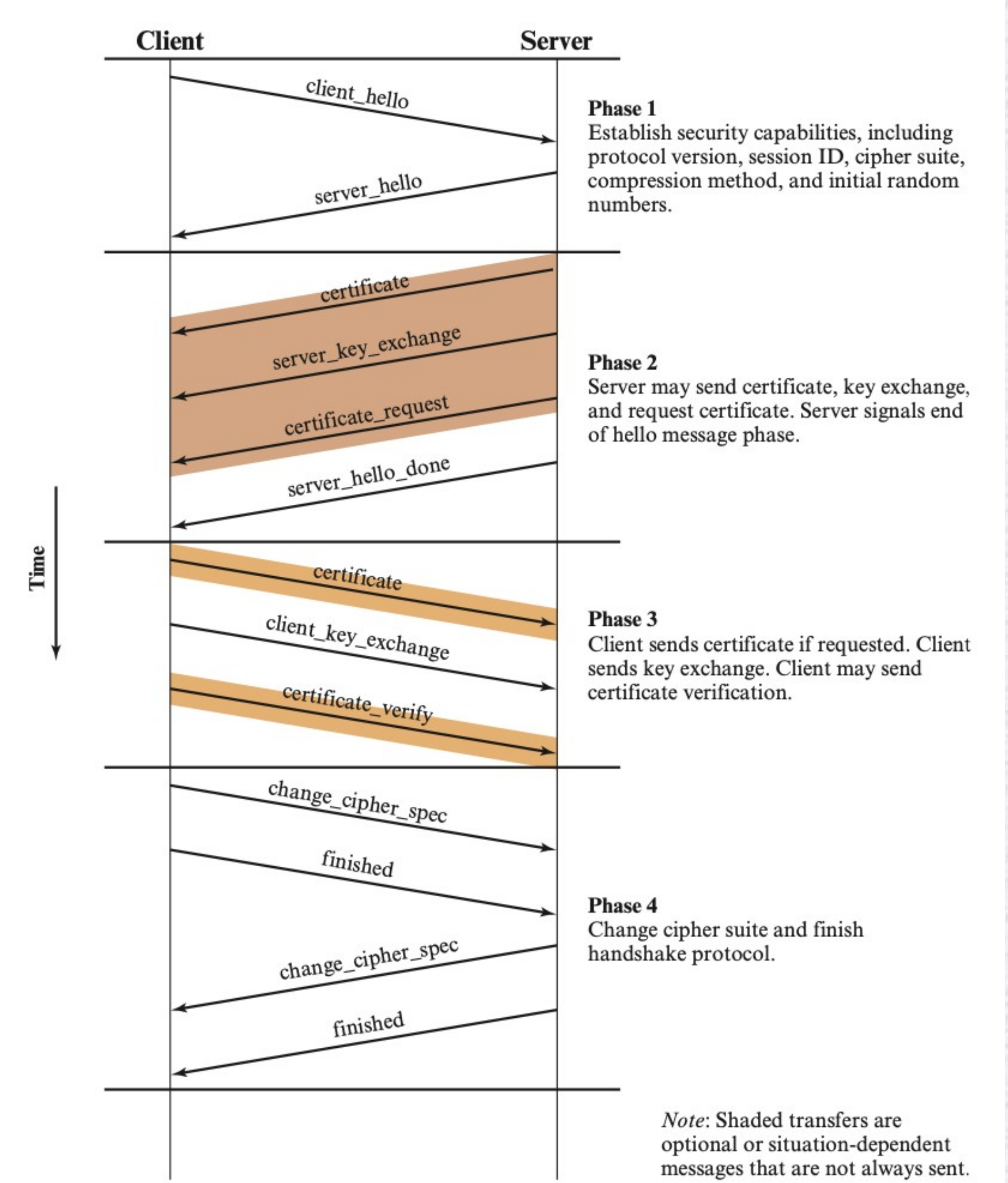


Figure 17.6 Handshake Protocol Action

Phase – 1
Client waits for **server_hello** message

server_hello message contains the same parameters as the client_hello message.

- Version: lowest (client version, server version).
- Random field: is independent of the client's Random field.
- If the SessionID field of the client was nonzero, the same value is used by the server; otherwise the server's SessionID field contains the value for a new session.
- The CipherSuite and Compression fields contain the single cipher suite and compression method selected

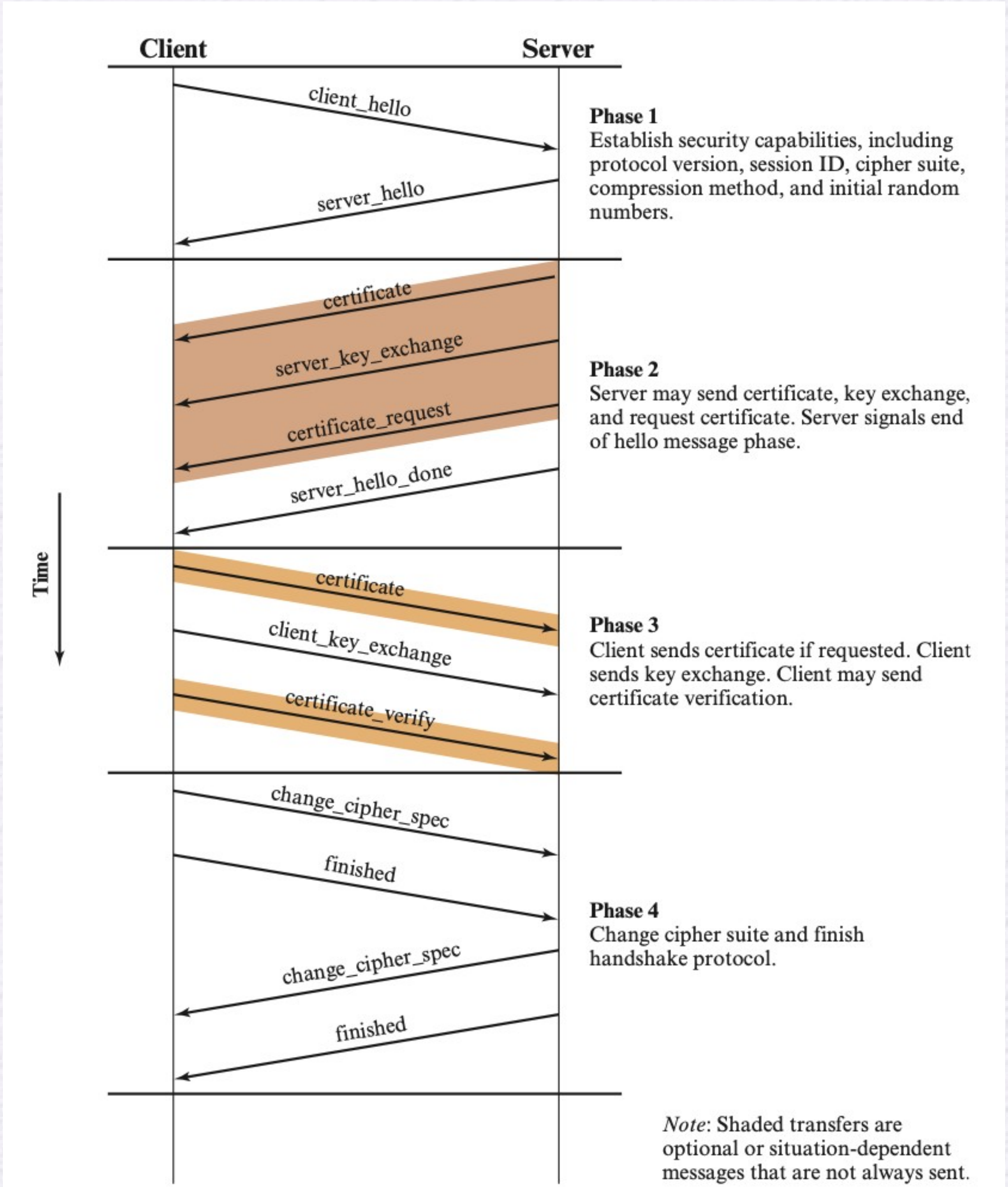


Figure 17.6 Handshake Protocol Action

Details on CipherSuite

- **Key exchange algorithms**
  - RSA, Fixed Diffie–Hellman, Ephemeral Diffie-Hellman, Anonymous Diffie–Hellman
- **Cipher Algorithms for data encryption**
  - DES, 3DES, DES40, RC4 etc.
- **MAC Algorithm for data integrity:** MD5, SHA-1
- **CipherType**: Stream or Block
- **HashSize**
  - MD5 produces a 128-bit hash (16 bytes)
  - SHA-1 produces a 160-bit hash (20 bytes)
- **Key Material**
  - Raw bytes used in generating the write keys (both encryption and MAC)
- **Initialization Vector Size**
  - Size of block cipher operators.

Phase – 2

Server initiates phase2 and send **certificate** message

certificate: This message is used by the server to authenticate itself to the client by sending its public key in the form of an X.509 certificate.

server_key_exchange: sent when required (depending on the key exchange method). It contains key exchange parameters.

- Anonymous Diffie-Helman
  - Prime number, public key, etc.

certificate_request: server can optionally request a certificate from the client for client authentication.

server_hello_ done: has no parameters, basically indicates the end of server's hello and

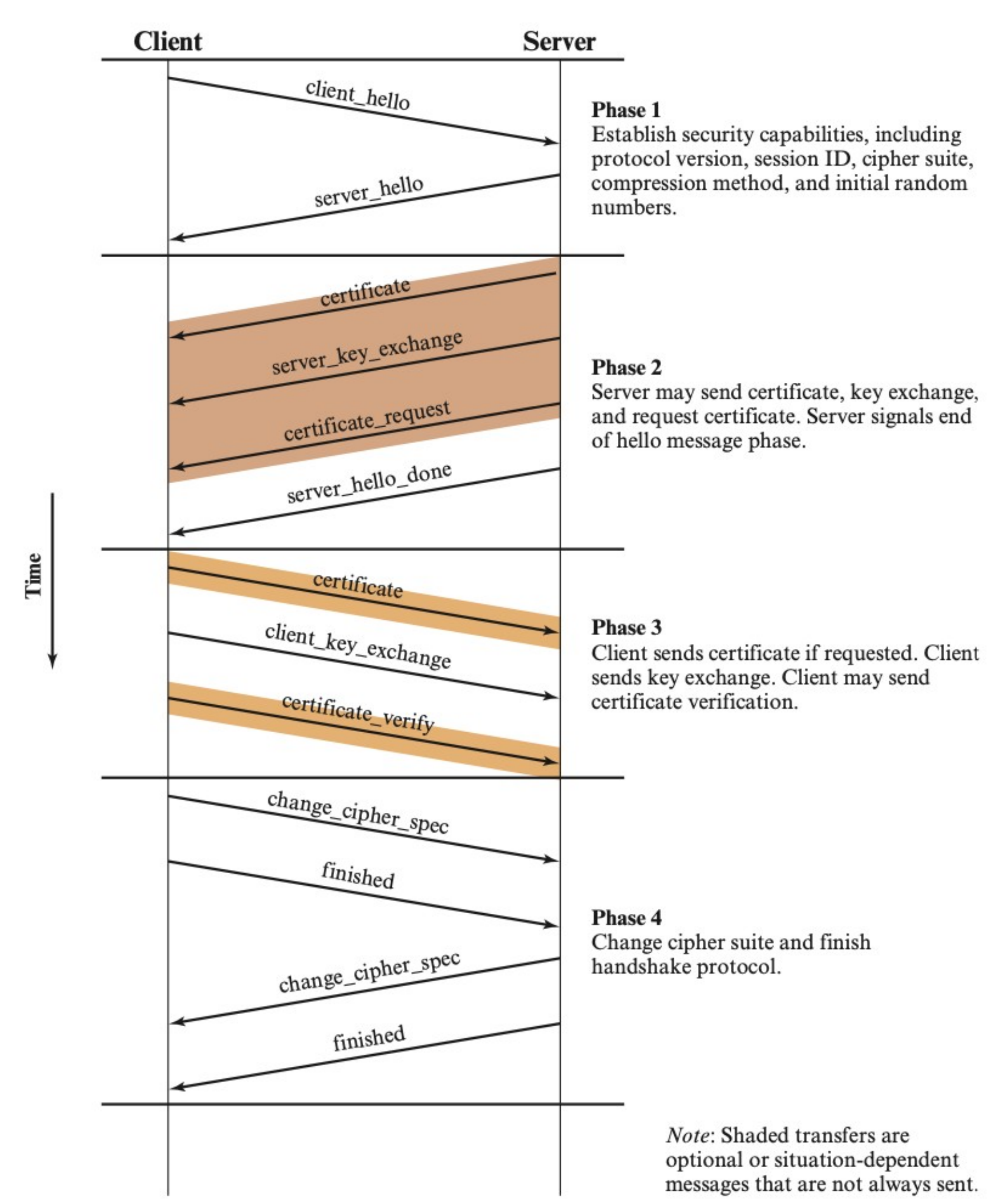


Figure 17.6 Handshake Protocol Action

Phase – 3

Client sends **certificate** if requested.
Can be used for client authentication.

Certificate Verify: contains signature (HMAC) of all handshake messages up to this point, ensuring integrity.

- Prevents Trudy to modifying the message (deleting strong encryption algorithms).
- Proves client possession of its private key.
    - ClientPrivateKey(HMAC(all_messages))=ClientPublicKey(HMAC(all_messages))
    - HMAC is whatever MAC algorithm client server agreed on cipher suite.

- **Client_Key_Exchange:** Similar to server_key_exchange sends key exchange parameters if needed.

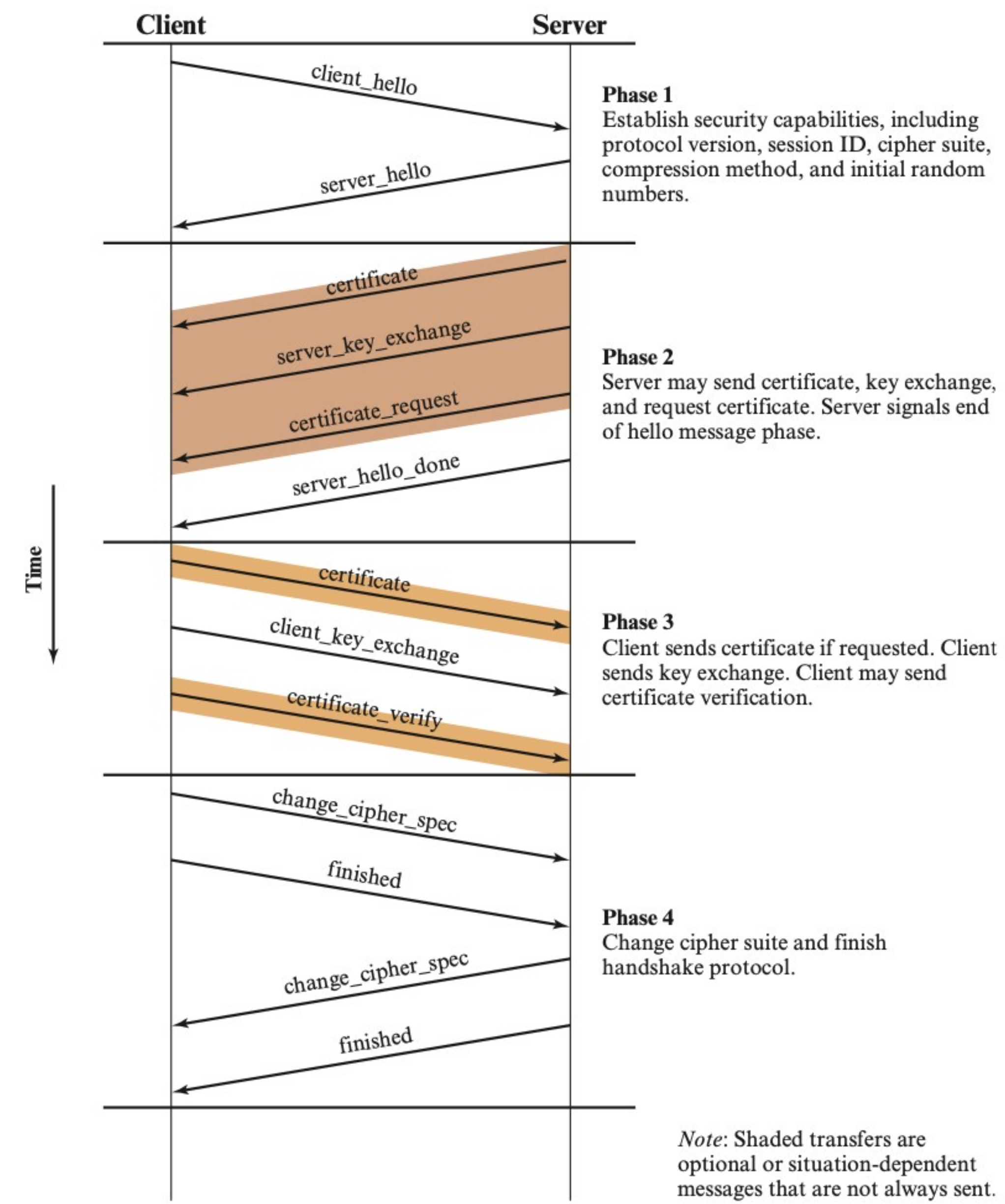


Figure 17.6 Handshake Protocol Action

Phase – 4

Change_cipher_spec: 1 byte message, if 0x01 indicates that session's negotiated cipher suite and keys are now active. All subsequent messages will be encrypted and protected using the negotiated cryptographic parameters.

Finished: both parties take hash of the all messages exchanged in handshake process. Then use agreed session key (Master Key) to compute HMAC of the hash.

- HMAC=agreed MAC algorithm (Master Key, all messages)

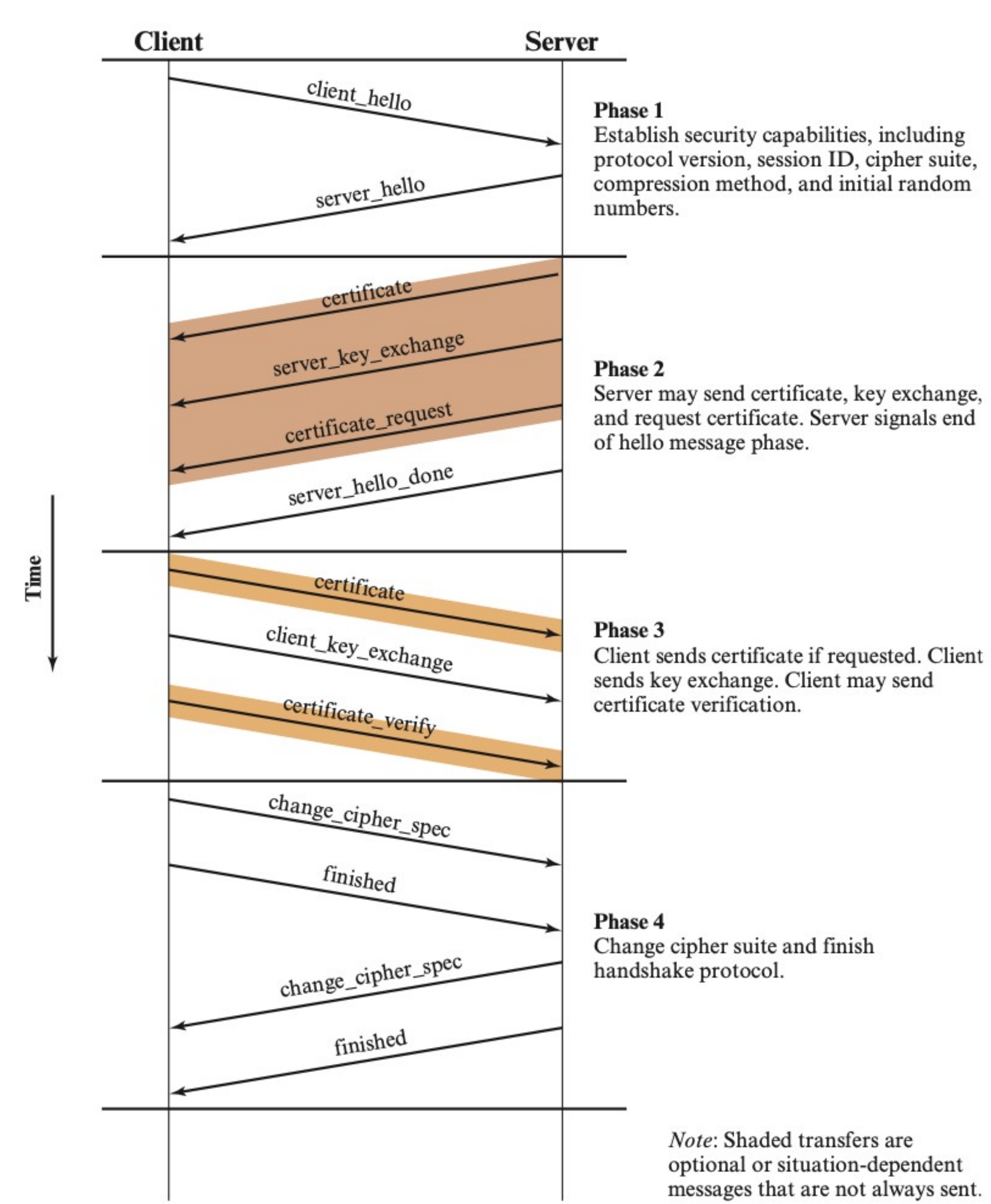


Figure 17.6 Handshake Protocol Action

In client_key_exchange and server_key_exchange messages parties agree on parameters of the key exchange algorithm. Using those parameters both client and server independently generate Master Key, **using Pseudo Random Function**.

Then using the Master Key both client and server generate following keys, again using the **Pseudo Random Function**.

K_c : encryption key for data sent from client to server

M_c : MAC key for data sent from client to server

K_s : encryption key for data sent from server to client

M_s : MAC key for data sent from server to client

The two encryption keys will be used to encrypt data; the two MAC keys will be used to verify the integrity of the data.

Pseudo Random Function.

er_secret = PRF(pre_master_secret, “master secret”, ClientHello.random || ServerHello .rar

pre_master_secret: exchanged as part of client_key_exchange and server_key_exchande. If key exchange algorithm is RSA it is value of 48 byte.

ClientHello.random and ServerHello .random: are random values exchanged in client_hello and server_hello.
|| means we combine them.

”master secret” is a just a label.

Seed is combination of

- ClientHello.random
- ServerHello .random
- Label

HMAC=agreed MAC algorithm (pre_master_secret, Seed)

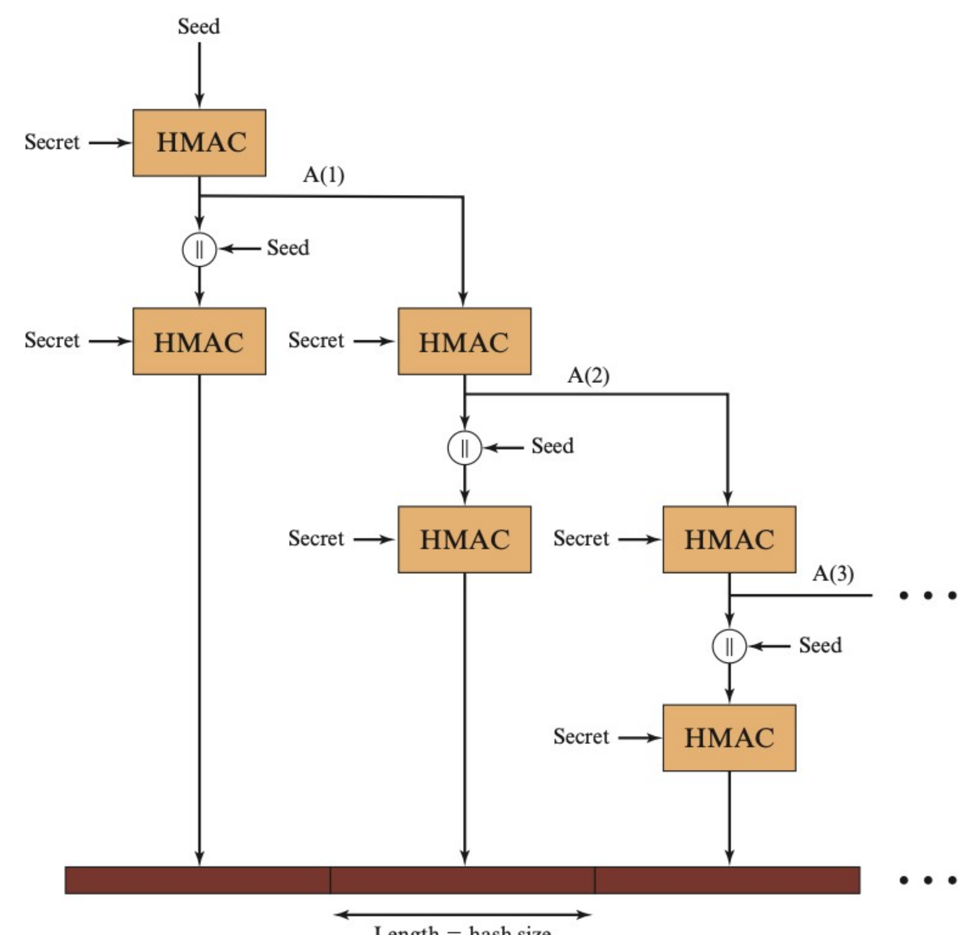


Figure 17.7 TLS Function P_hash(secret, seed)

Pseudo Random Function.

t_of_size_Total= PRF(master_secret, “key expansion”, ClientHello.random || ServerHello .ra

K_c and K_s both have size of X bytes
M_c and M_s both have size of Y bytes

Then we generate
secret_of_size_Total = 2*X+2*Y

And split it into X X Y Y which will correspond to Kc, Ks, Mc, Ms.

If as a result of last iteration in PRF we have let’s say 2*X+2*Y + Z bytes, the Z bytes are ignored.

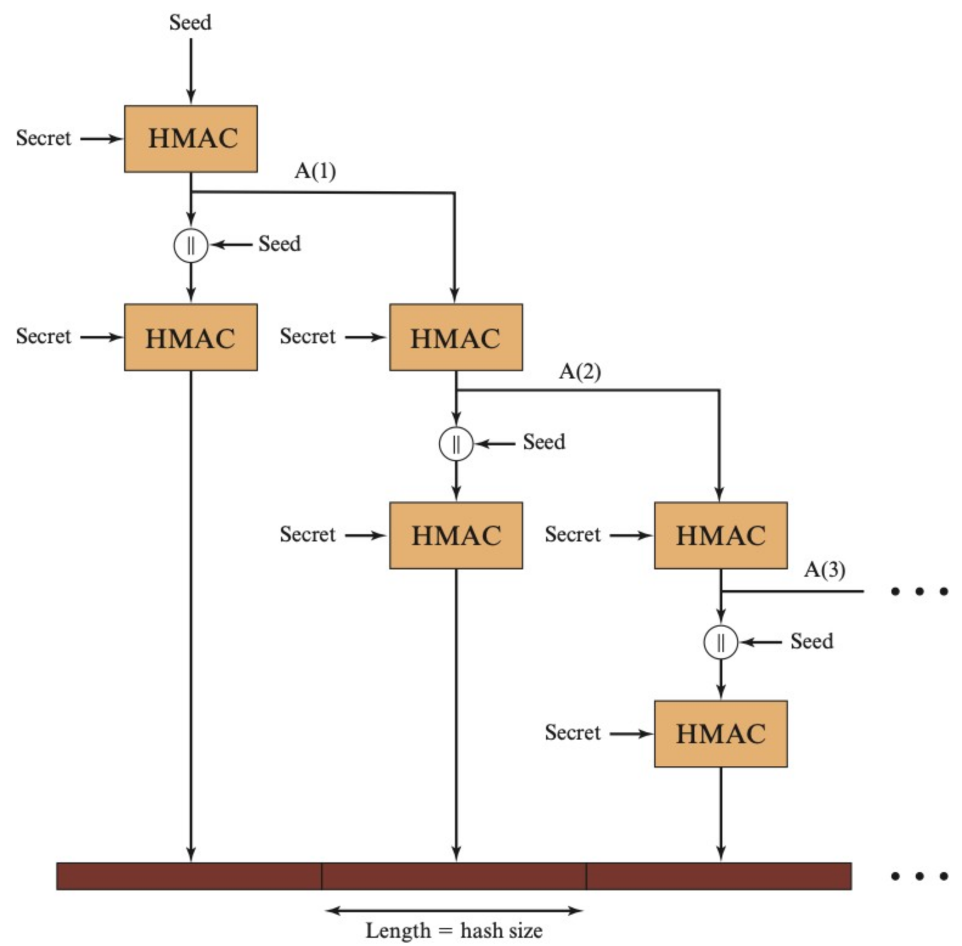


Figure 17.7 TLS Function `P_hash(secret, seed)`

TLS Record Protocol

TLS Record Protocol provides Confidentiality and Message Integrity using the 4 keys derived in TLS Handshake.

- Takes an application message to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies a MAC, encrypts, adds a header, and transmits the resulting unit in a TCP segment. Received data are decrypted, verified, decompressed, and reassembled before being delivered to higher-level users.

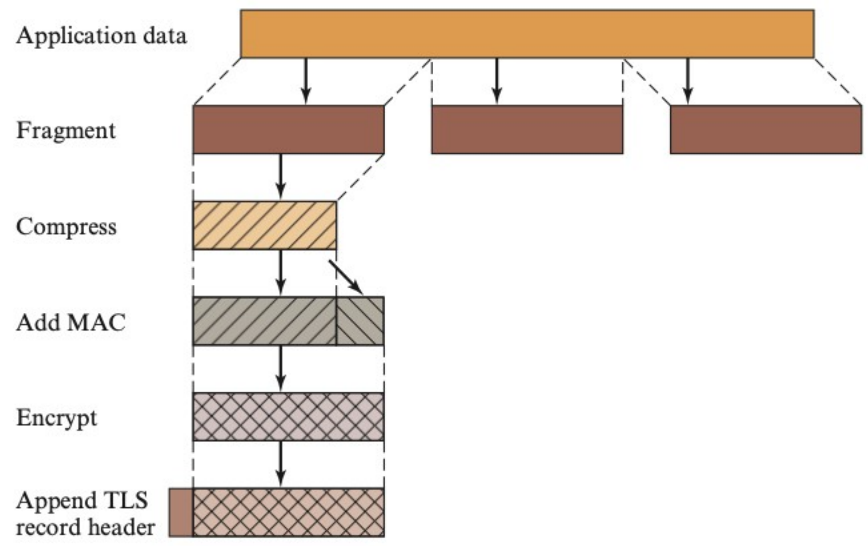


Figure 17.3 TLS Record Protocol Operation

TLS Record Protocol

HMAC

$$\mathrm{HMAC}_K(M) = \mathrm{H}[(K^+ \oplus \mathrm{opad}) \,\|\, \mathrm{H}[(K^+ \oplus \mathrm{ipad}) \,\|\, M]]$$

where

- H = embedded hash function (for TLS, either MD5 or SHA-1)
- M = message input to HMAC
- K^+ = secret key padded with zeros on the left so that the result is equal to the block length of the hash code (for MD5 and SHA-1, block length = 512 bits)
- ipad = 00110110 (36 in hexadecimal) repeated 64 times (512 bits)
- opad = 01011100 (5C in hexadecimal) repeated 64 times (512 bits)

For TLS, the MAC calculation encompasses the fields indicated in the following expression:

HMAC_hash(MAC_write_secret, seq_num || TLSCompressed.type || TLSCompressed.version || TLSCompressed.length || TLSCompressed.fragment)

Next, the compressed message plus the MAC are encrypted using symmetric encryption.

TLS Record Protocol

TLS Header

- **Content Type (8 bits)**
  - Handshake
  - Change_cipher_spec
  - Alert
  - Application_data

- **Major Version (8 bits):** Indicates major version of TLS in use (TLSv1.0, TLSv2.0, etc)

- **Minor Version (8 bits)**: Indicates minor version in use (TLSv1.1, TLSv1.2, etc).

- **Compressed Length (16 bits):** The length in bytes of the plaintext fragment (or compressed fragment if compression is used).

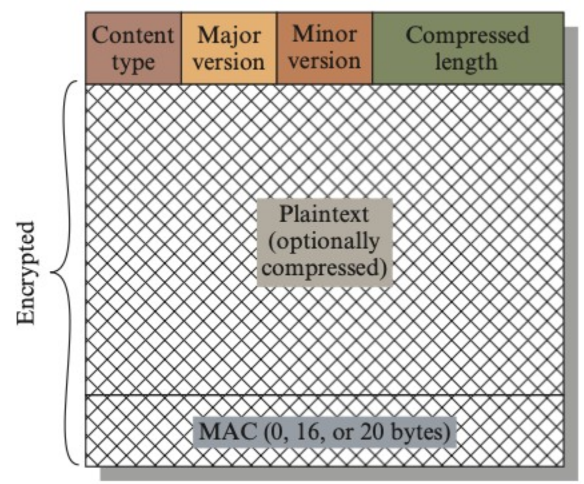


Figure 17.4 TLS Record Format

- **Content Type (8 bits)**
  - Handshake
  - Change_cipher_spec
  - Alert
  - Application_data

Change_cipher_spec: 1 byte message, if 0x01 indicates that session's negotiated cipher suite and keys are now active.

Application_data: indicates everything is working as expected and we are sending data to upper layer protocol (application layer protocol - HTTP)

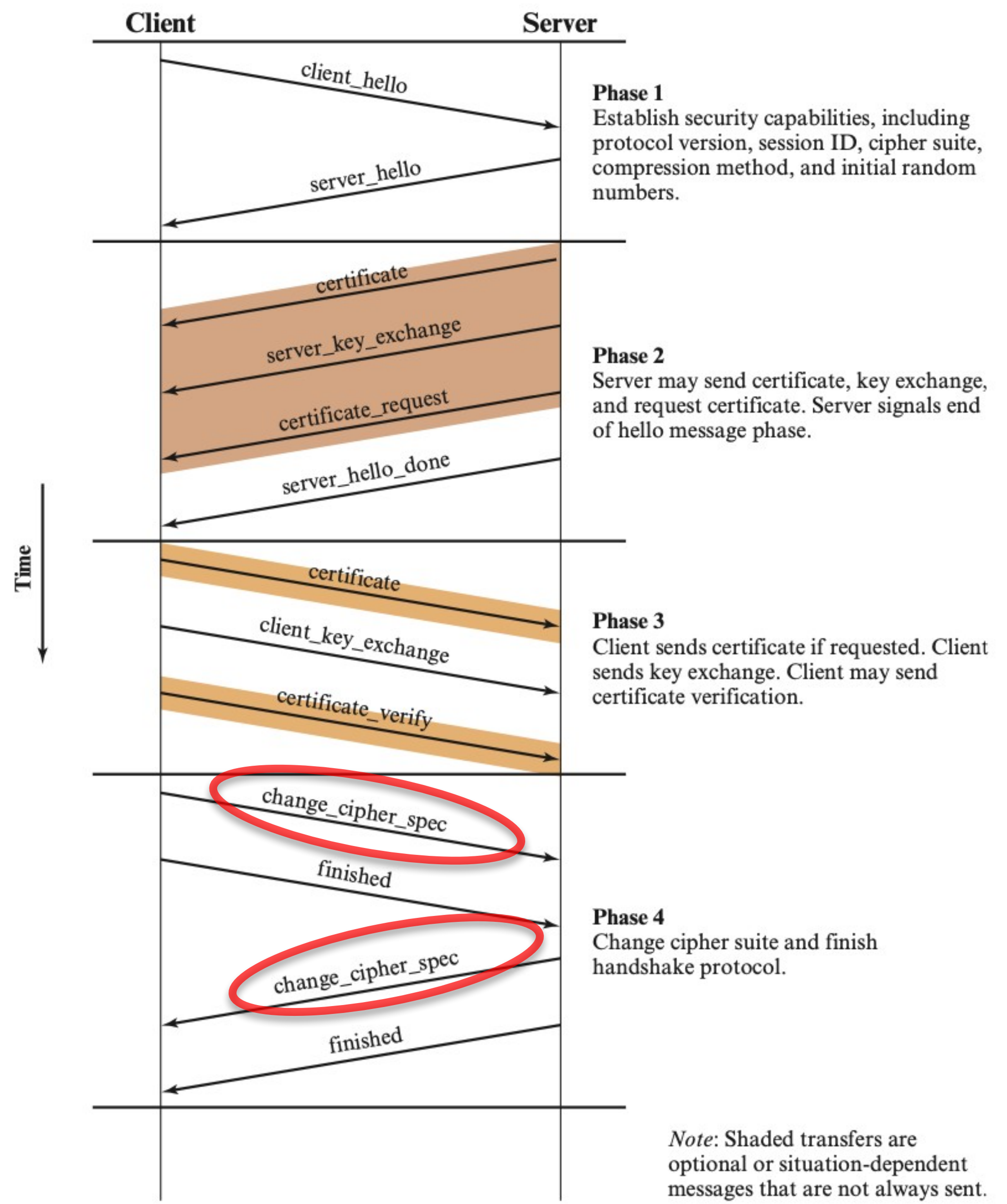
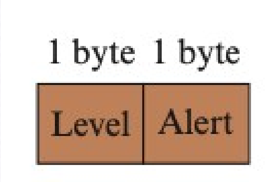


Figure 17.6 Handshake Protocol Action

- **Content Type (8 bits)**
  - Handshake
  - Change_cipher_spec
  - Alert
  - Application_data



- The Alert is used to convey TLS-related alerts to the peer entity.
- Each message in this protocol consists of two bytes.
  - The first byte is either warning (1) or fatal (2) to convey the severity of the message.
    - If the level is fatal, TLS immediately terminates the connection. Other connections on the same session may continue, but no new connections on this session may be established.
  - The second byte contains a code that indicates the specific alert.
    - Examples:
      - Fatal - **bad_record_mac** (an incorrect MAC was received)
      - Fatal - **handshake_failure** (sender was unable to negotiate an acceptable set of security parameters given the options available).
      - Warning - **unsupported_certificate** (the type of the received certificate is not supported).

- TLS: RFC 5246 https://datatracker.ietf.org/doc/html/rfc5246

SECURE SHELL (SSH)

SECURE SHELL (SSH)

- SSH is organized as **three protocols** that typically run on top of TCP
  - Transport Layer Protocol
  - User Authentication Protocol
  - Connection Protocol

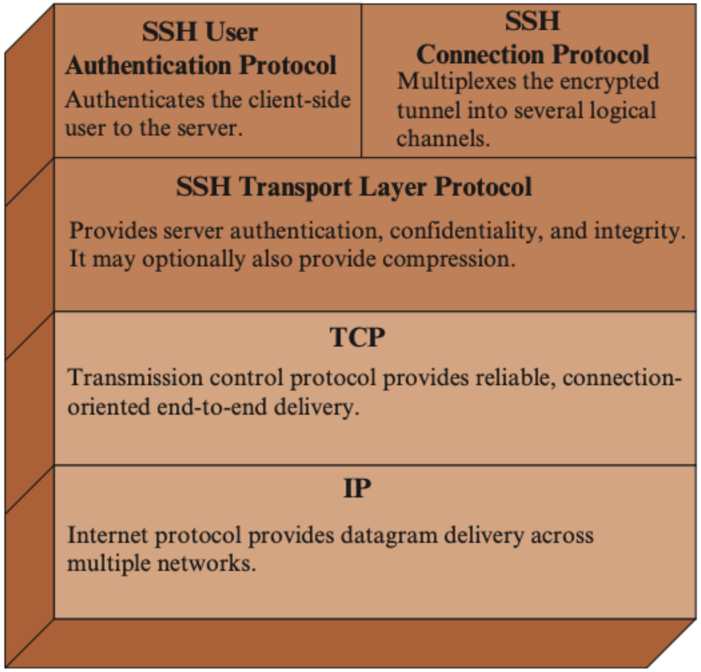


Figure 17.8 SSH Protocol Stack

SSH - Transport Layer Protocol

- Initially TCP connection is established which is separate from SSH Transport Layer Protocol.
- Then set of packets are exchanged
  - SSH-protoversion-softwareversion
  - SSH_MSG_KEXINIT
- Once the connection is established, the client and server exchange data, referred to as packets, in the data field of a TCP segment

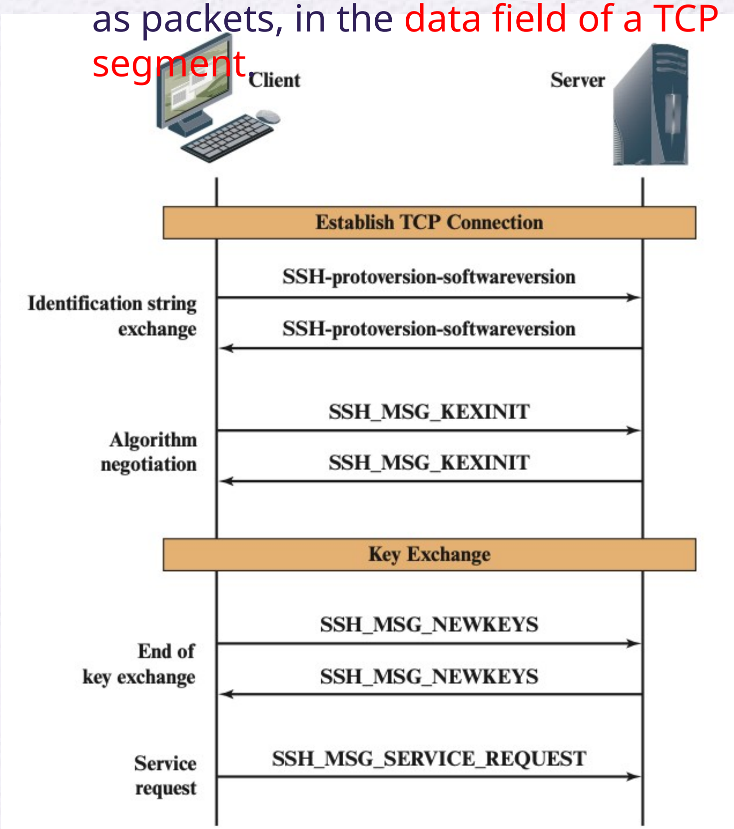


Figure 17.9 SSH Transport Layer Protocol Packet Exchanges

SSH - Transport Layer Protocol Packet Structure

- **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.
- **Random padding:** random bytes to make total length (excluding MAC) of the packet a multiply of cipher block size or 8 bytes for a stream cipher.
- **Message authentication code:** If message authentication has been negotiated, this field contains the MAC value. It is computed over the (unecrypted) entire packet plus a sequence number, excluding the MAC field.
- **Sequence number:** 32-bit packet sequence that is initialized to 0 for the 1st packet and incremented for every packet.

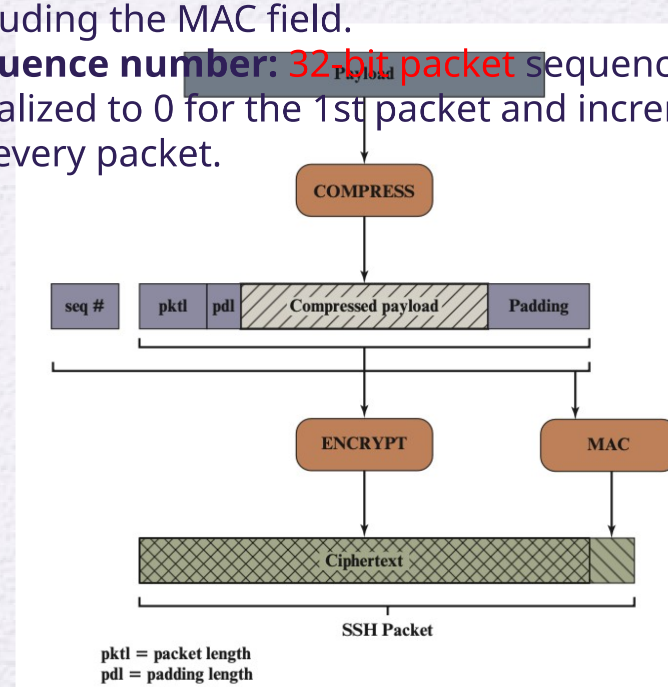


Figure 17.10 SSH Transport Layer Protocol Packet Formation

SSH - Transport Layer Protocol

- **Identification string exchange:** it is a string value of the form
  - SSH-protoversion-softwareversion SP comments CR LF
  - protoversion: is a SSH protocol version -> 1.5, 2.0, etc.
  - softwareversion: implementation of SSH -> OpenSSH, Sun_SSH, etc
  - SP: is a space character -> “ ”, 0x20
  - Comment: an optional information about platform -> Ubuntu_20.04, Debian
  - CR (Carriage Return) -> 0x0D, used to terminate line along with LF
  - LF (Line Feed) -> 0x0A, which signals end of the identification string.
  - **Example:** SSH-2.0-OpenSSH_8.6 Ubuntu-20.04<CR><LF>
- The **Identification string** might be different for client and server however overall, they should be compatible if they are not compatible then connection will fail. (update needed).

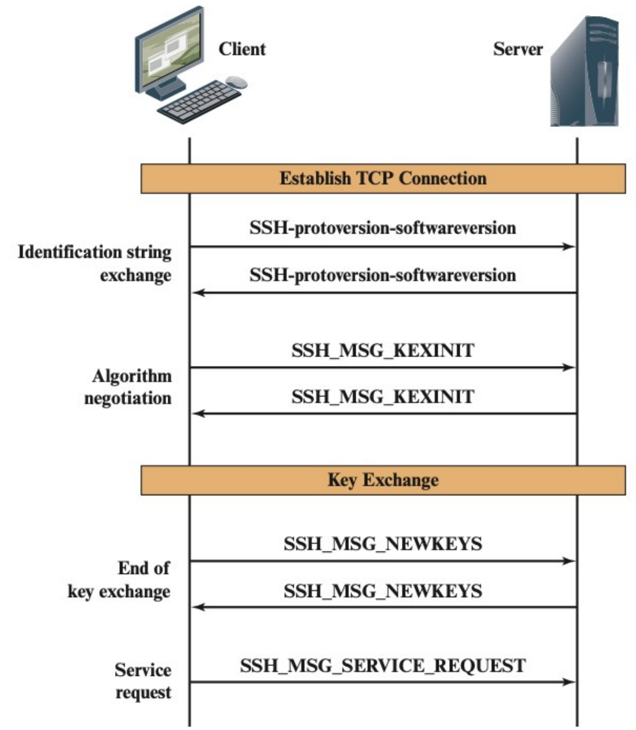


Figure 17.9 SSH Transport Layer Protocol Packet Exchanges

Algorithm negotiation: Each side sends an SSH_MSG_KEXINIT containing 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
  - Compression algorithm

Table 17.3 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-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

SSH - Transport Layer Protocol

- **Key exchange:** This step is different for each type of the key.
- It is used to establish shared secret key: **Master Key** between client and server. Book shows example with Diffie–Hellman.
  - p: large prime number - publicly known
  - g: a generator - publicly known
  - A- Alice's private key - private
  - B-Bob's private key - private
  - Alice computes her public key = A_public = g^A mod p
    - Sends it to Bob
    - Same is done by Bob
  - Alice and Bob calculate shared key => (A_public)^B mod p=(B_public)^A mod p

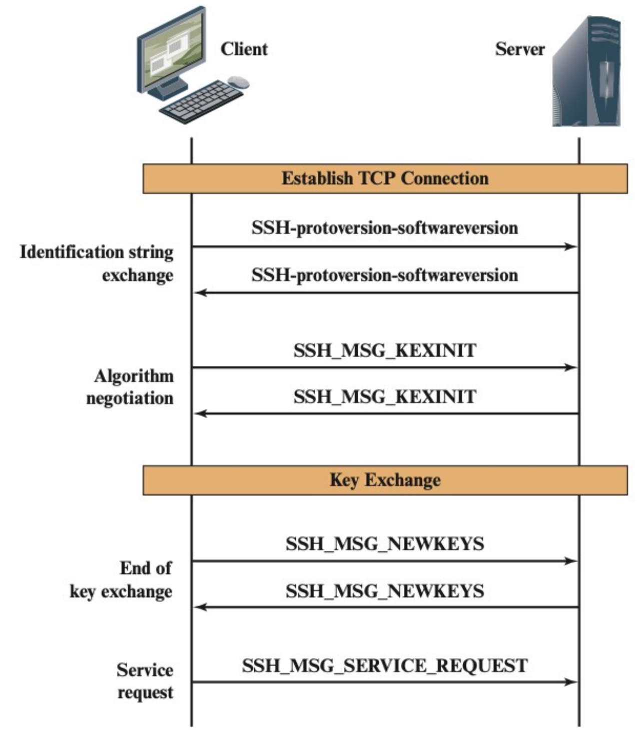


Figure 17.9 SSH Transport Layer Protocol Packet Exchanges

End of key exchange: Signaled with SSH_MSG_NEWKEYS

- At this point both parties can start using the shared key.

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 data is exchanged as the payload of an SSH Transport Layer packet, protected by encryption and MAC.

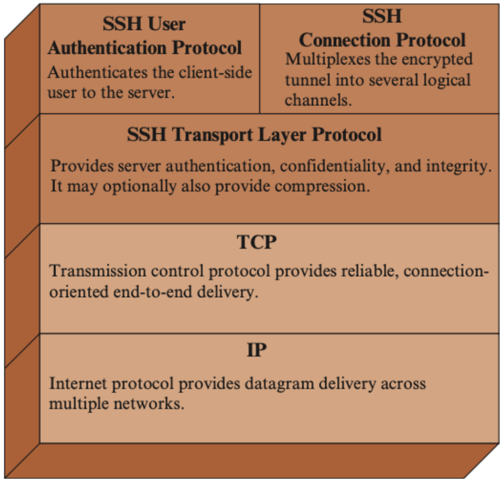


Figure 17.8 SSH Protocol Stack

SSH – User Authentication

- Three types of messages
  - Authentication request
  - Authentication failure
  - Authentication success

Message Exchange Steps

1. Initial Request: The client sends a SSH_MSG_USERAUTH_REQUEST with the method set to none.

- This is essentially a probe to check the validity of the user name.

2. Username Validation: If the user name is invalid, the server sends SSH_MSG_USERAUTH_FAILURE with no methods listed and partial success = false.

- If the user name is valid, the server responds with a SSH_MSG_USERAUTH_FAILURE listing acceptable authentication methods.

3. Authentication Method Selection: The client selects one of the listed methods and sends another SSH_MSG_USERAUTH_REQUEST with the chosen method and its required fields.

4. Method Execution: The server and client may exchange additional messages depending on the method (e.g., public key or password verification).

5. Partial Success or Final Success:

- If multiple methods are required: On success of one method, the server sends SSH_MSG_USERAUTH_FAILURE with partial success = true.

Authentication request

Byte SSH_MSG_USERAUTH_REQUEST (code 50 and value 0x32)
string user name
string service name
string method name
[method-specific fields]

Examples
user name: root, admin, bcharyyev
service name: ssh-connection, sftp
method name: password, public key

Authentication failure

Byte SSH_MSG_USERAUTH_FAILURE (code 51 and value 0x33)
string name-list
boolean partial success

Examples
name-list: password, public key
partial success: Indicates if any previous authentication attempt succeeded.

Authentication success

Byte SSH_MSG_USERAUTH_SUCCESS (code 52 and value 0x34)

SSH – Connection Protocol

- In **SSH Transport Layer Protocol** we established encryption, server authentication, and data integrity, in **SSH User Authentication Protocol** we authenticated client.

- Three packet types are used.
- Note that for given SSH session we can have multiple channels.
  - **Open Channel:** to open the channel between client and server
  - **Data Transfer:** to transfer the data
  - **Close Channel:** to close the channel

Message header format looks as follows

Byte	SSH_MSG_CHANNEL_OPEN
String	Channel type
Uint32	sender channel
Uint32	initial window size
Uint32	maximum packet size

Channel type: Can be session, x11, Forwarded-TCP/IP Channel, Direct-TCP/IP Channel.
- Session: remote shell, sftp

Sender Channel: Locally assigned channel number.

Initial Window Size: Used for flow control.

Max Packet Size: Maximum size of a single data packet.