UniteImaSapienza
Università degli Studi di Roma

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Secure Socket Layer

SSL / TLS

Diagrams courtesy of W.Stallings' chapter on «Web Security»

F. Parisi Presicce

UniteImaSapienza.it

Security Threat Classifications

One way to classify Web security threats in terms of the **type of the threat**:

- Passive threats
- Active threats

Another way to classify Web security threats in terms of the **location of the threat**:

- Web server
- Web browser
- Network traffic (between browser and server)

Web Security Approaches

Number of possible ways, similar in provided services

Differ in

- Scope of Applicability
- Location within the TCP/IP Protocol Stack

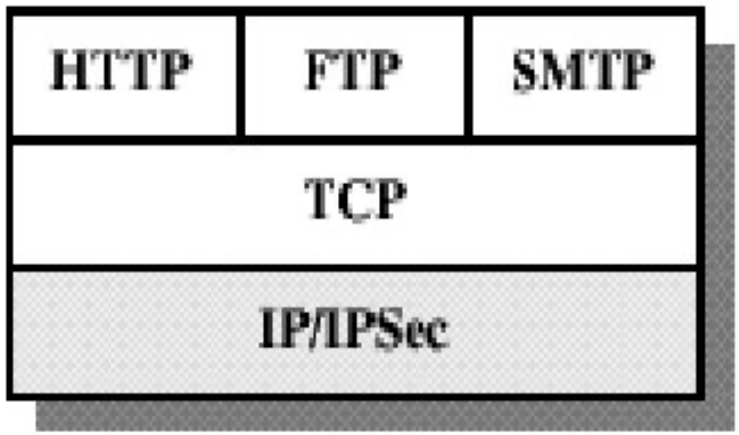
Web security can be provided at any of these networking layers or levels:

- Network level
- Transport level
- Application level

Web Security: Network Level

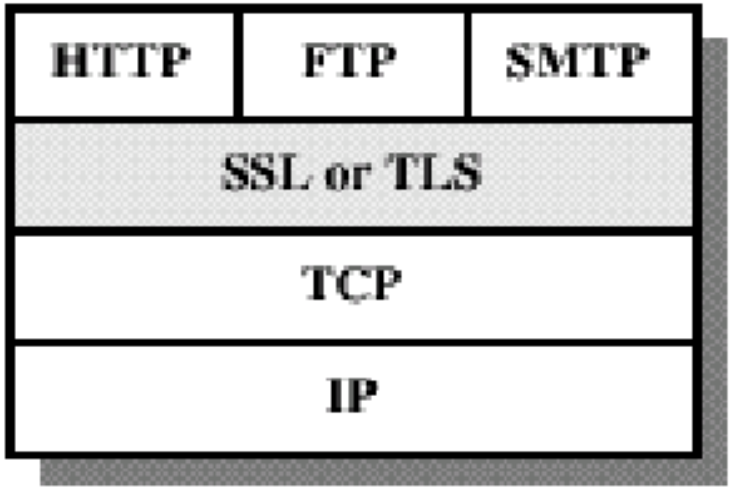
- Provide security using IPSec

- Advantages:
  - Transparent to users and applications
  - Filtering : only selected traffic need incur the overhead of IPSec processing



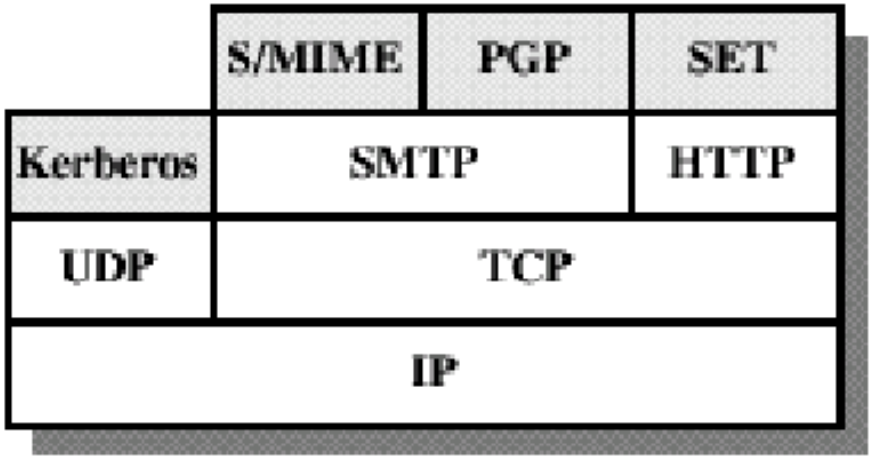
Web Security: Transport Level

- Implemented above TCP layer
- Secure Sockets Layer (SSL) and successor Transport Layer Security (TLS)
- Advantages:
  - Could be transparent to Applications
  - Part of most Browsers



Web Security: Application Level

- Application specific services embedded in application
- Secure Electronic Transaction (SET), S/MIME, PGP
- Advantage
  - Service tailored to specific needs of application



Introduction to SSL

- The overall goal of the Secure Sockets Layer (SSL) protocol was to provide privacy and reliability between two communicating applications.
  - SSL was developed by Netscape.
  - Evolved through an unreleased v1 (1994), flawed-but-useful v2
  - The last version of the SSL protocol was Version 3 (V3), specification released March 1996.
  - replaced by standard TLS 1.0 in Jan 1999 (often referred to as SSL3.1 because of minor tweaks)
  - most recent TLS 1.3 draft 23 as of March 2018

Goals 1

- The goals of SSL Protocol v3.0, in order of their priority, are:
  - **Cryptographic security:** SSL should be used to establish a secure connection between two parties.
  - **Interoperability:** Independent programmers should be able to develop applications utilizing SSL 3.0 that will then be able to successfully exchange cryptographic parameters without knowledge of one another's code.

Goals 2

- Extensibility: SSL seeks to provide a framework into which new public key and bulk encryption methods can be incorporated as necessary.
  - This will also accomplish two sub-goals:
    - To prevent the need to create a new protocol.
    - To avoid the need to implement an entire new security library.

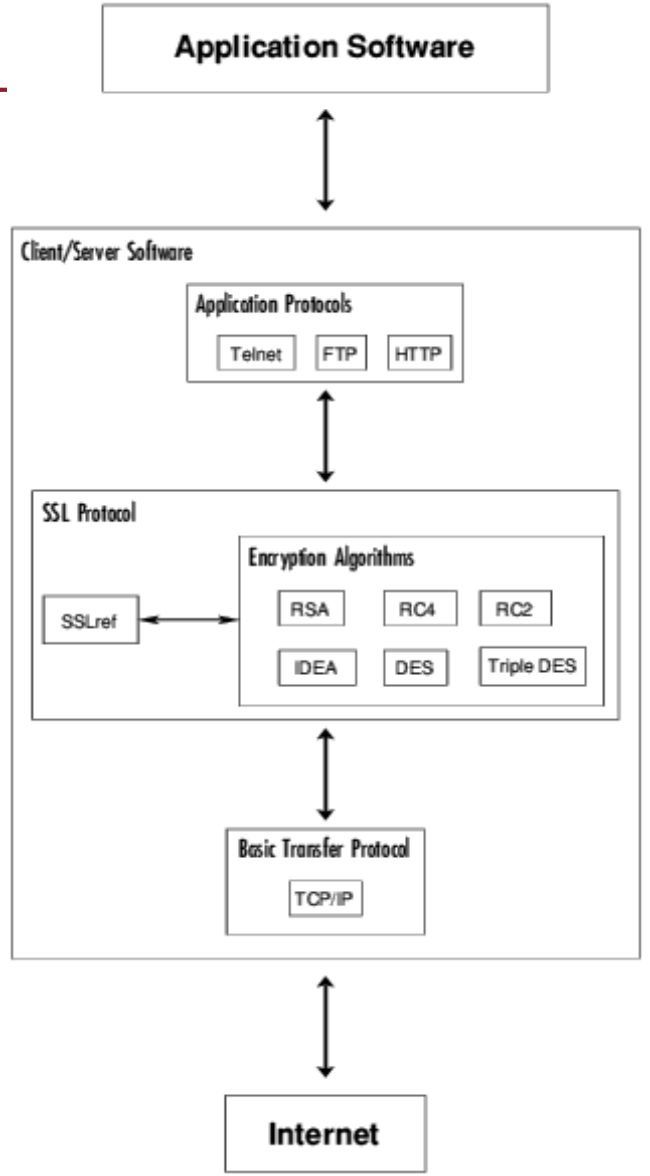
- Relative efficiency: Cryptographic operations tend to be highly CPU intensive, particularly public key operations.

SSL Services

- peer entity authentication
- data confidentiality
- data authentication and integrity
- compression/decompression
- generation/distribution of session keys
  - integrated into protocol
- security parameter negotiation
- NOT non-repudiation
- NO protection against traffic analysis attacks

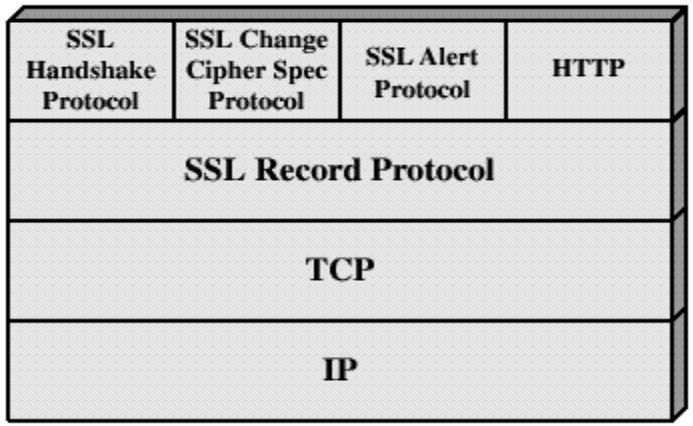
Protocol Architecture

- SSL can run under application protocols such as:
  - HTTP
  - FTP
  - TELNET
- SSL normally uses TCP/IP as its basic transport protocol.
- SSL uses X.509 certificates for authentication
- SSL uses RSA as its public-key cipher.
- SSL can use any one of RC4-128, RC2-128, DES, Triple DES or IDEA as its bulk symmetric cipher.



Protocol Architecture

- SSL is not a single protocol but rather two layers of protocols:
  - The SSL Record Protocol which provides the basic security services to higher layer protocols.
  - Three higher-layer protocols are defined as part of SSL:
    - SSL Handshake
    - SSL Change Cipher Spec
    - SSL Alert
- These three higher-level protocols are used in the management of SSL exchanges.
- SSL is designed to make use of TCP to provide a reliable end-to-end secure service.



SSL Communication

- The Secure Sockets Layer (SSL) protocol defines two roles for entities in the network:
  - One entity is always a client.
  - The other entity is a server.
- SSL requires that each entity behaves very differently:
  - The client is the system that initiates the secure communication.
  - The server responds to the client's request.

- SSL clients and servers communicate by exchanging SSL messages.
- The most basic operation that an SSL client and server can perform is to establish a channel for encrypted communication.

SSL Architecture

- SSL session
  - an association between client and server
  - created by the Handshake Protocol
  - defines a set of cryptographic parameters
  - may be shared by multiple SSL connections
- SSL connection
  - a transient, peer-to-peer, communications link
  - associated with one SSL session

Handshake protocol either establishes new session and connection or uses existing session for new connection

SSL Session Parameters

- SSL session negotiated by handshake protocol
  - session ID
    - chosen by server
  - X.509 public-key certificate of peer
    - possibly null
  - compression algorithm
  - cipher spec
    - encryption algorithm
    - message digest algorithm
  - master secret
    - 48 byte shared secret
  - is_resumable flag
    - can be used to initiate new connections

SSL Connection State

connection end: client or server

Characterized by

- client and server chosen random: 32 bytes each
- keys generated from master secret, client/server random
  - client_write_MAC_secret server_write_MAC_secret
  - client_write_key server_write_key
  - client_write_IV server_write_IV
  - cipher state: initially IV, subsequently next feedback block
- sequence number: set to 0 (max 2^{64}-1) at each ChangeCipherSpec message

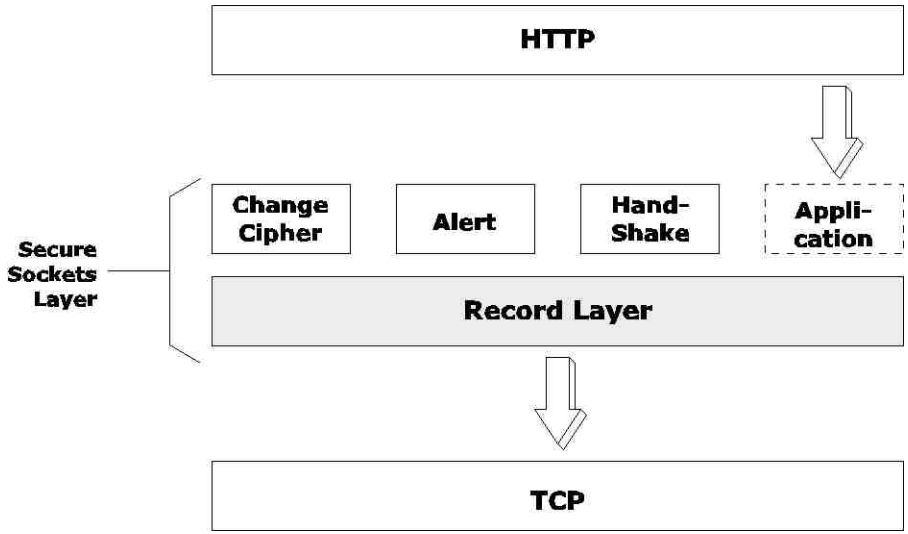
SSL Connection State

- 4 parts to state
  - current read state (security info for receiving)
  - current write state (security info for sending)
  - pending read state
  - pending write state
- handshake protocol
  - initially current state is empty
  - either pending state can be made current and reinitialized to empty by (receiving for read, sending for write) ChangeCipherSpec message

SSL Record Layer Protocol

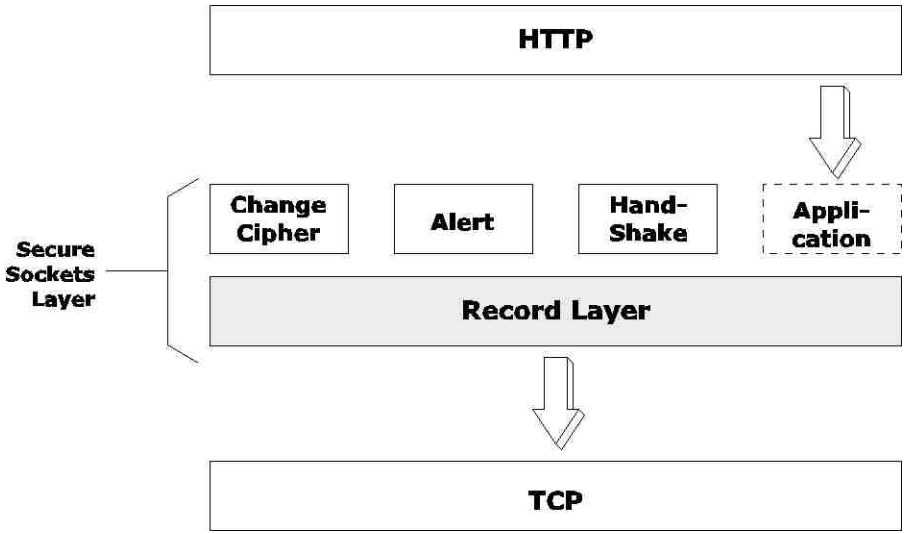
- SSL uses its Record Layer Protocol to encapsulate all messages.

- It provides a common format to frame the following message types:
  - Alert
  - ChangeCipherSpec
  - Handshake
  - Application protocol messages



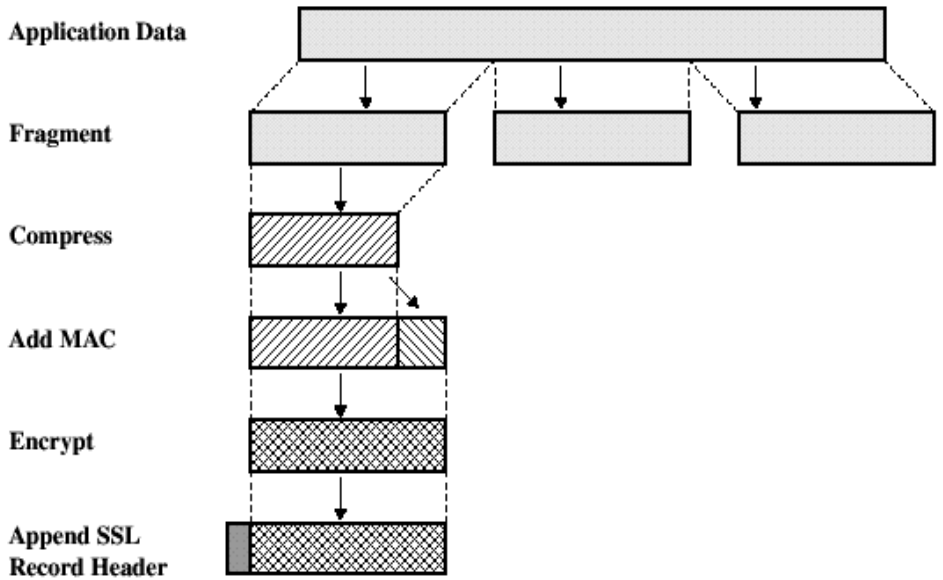
SSL Record Layer Protocol

- The Record Layer formatting consists of 5 bytes that precede other protocol message.
  - If message integrity is active, a message authentication code is placed at the end of the message.
  - If encryption is active, this layer is also responsible for the encryption process.



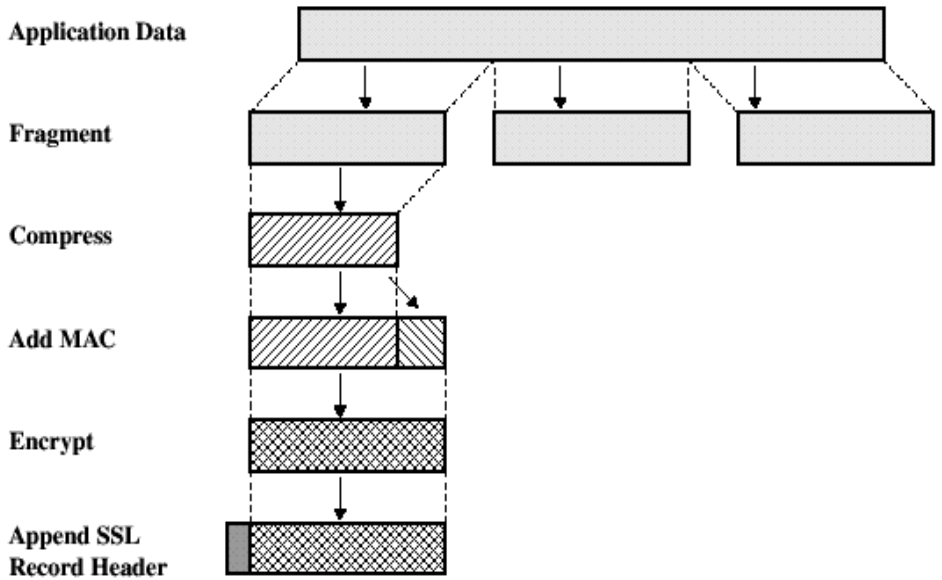
SSL Record Layer Protocol

- The Record Layer Protocol takes an application message and performs the following operations:
  - Fragments the data into manageable blocks.
  - Optionally, compresses the data.
    - Default is no compression.



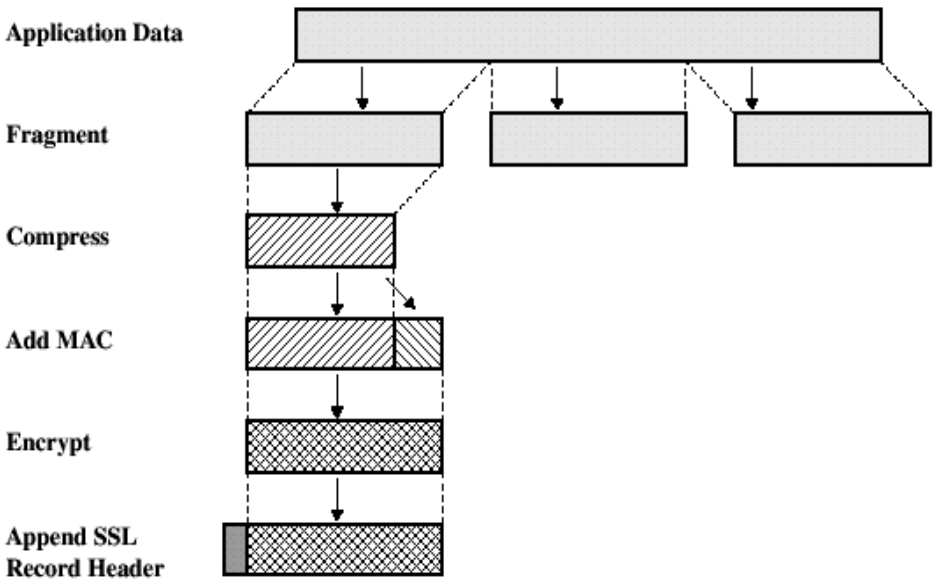
SSL Record Layer Protocol

- Adds a message authentication code (MAC).
- Encrypts the data plus MAC using symmetric encryption.
- Prepends a header.
- Transmits the unit in a TCP segment.

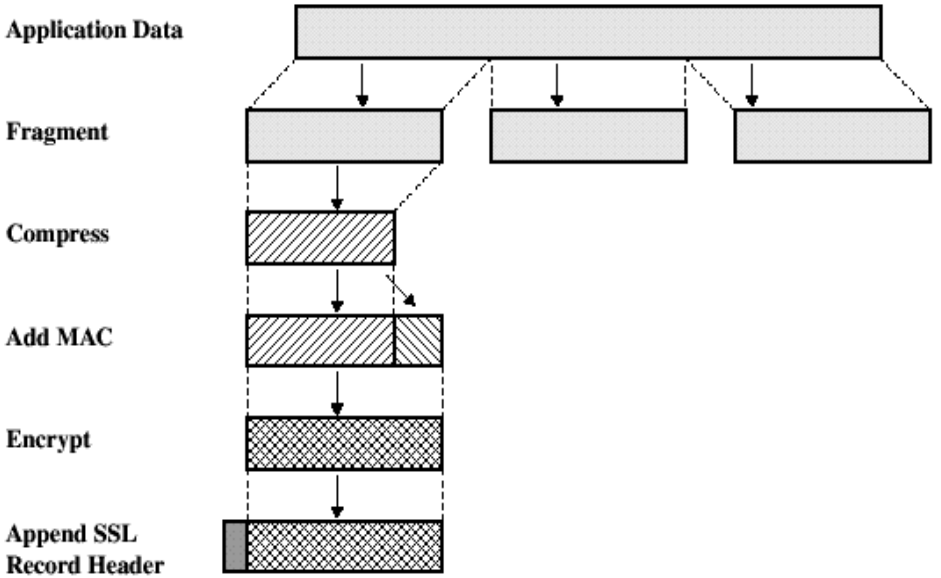


SSL Record Layer Protocol

- Some values:
  - Fragment size: (no more than) 2^{14} bytes, or 16,384 bytes.
  - Compression: This operation may not increase content length by more than 1024 bytes.
    - However, it should shrink it!



SSL Record Layer Protocol



- Message authentication code is calculated over the data using a shared secret key as follows:

  hash(MAC_write_secret || pad_2 ||
  hash(MAC_write_secret || pad_1 || seq_num ||
  SSLCompressed.type || SSLCompressed.length ||
  SSLCompressed.fragment))

SSL Record Layer Protocol

where:

|| represents concatenation

MAC_write_secret represents the shared secret key.

hash represents the cryptographic hash algorithm (either MD5 or SHA-1).

pad_1 represents the byte 0x36 repeated 48 times (384 bits) for MD5 and 40 times for SHA-1.

pad_2 represents the byte 0x5C repeated 48 times for MD5 and 40 times for SHA-1.

SSL Record Layer Protocol

seq_num represents the sequence number for this fragment.

SSLCompressed.type represents the higher level protocol used to process this message.

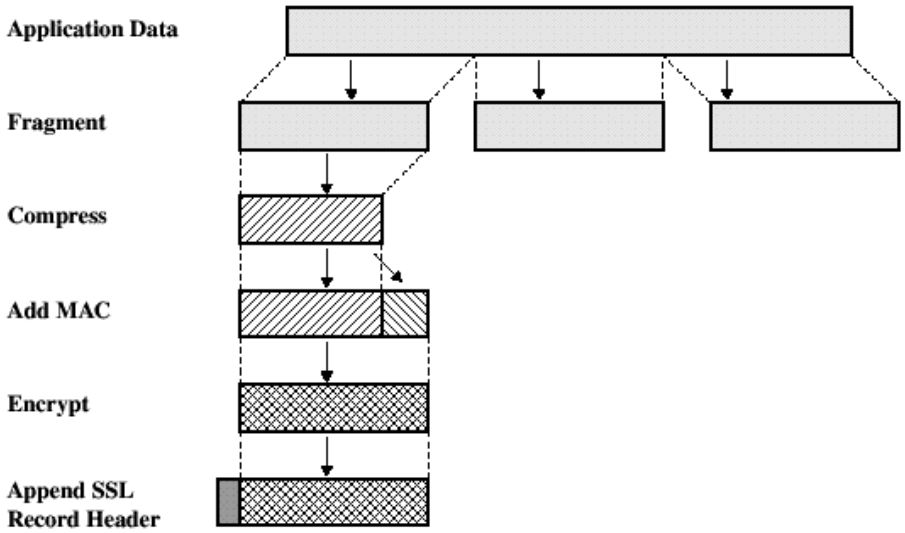
SSLCompressed.length represents the length of the compressed fragment.

SSLCompressed.fragment represents the compressed (or plaintext) fragment.

SSL Record Layer Protocol

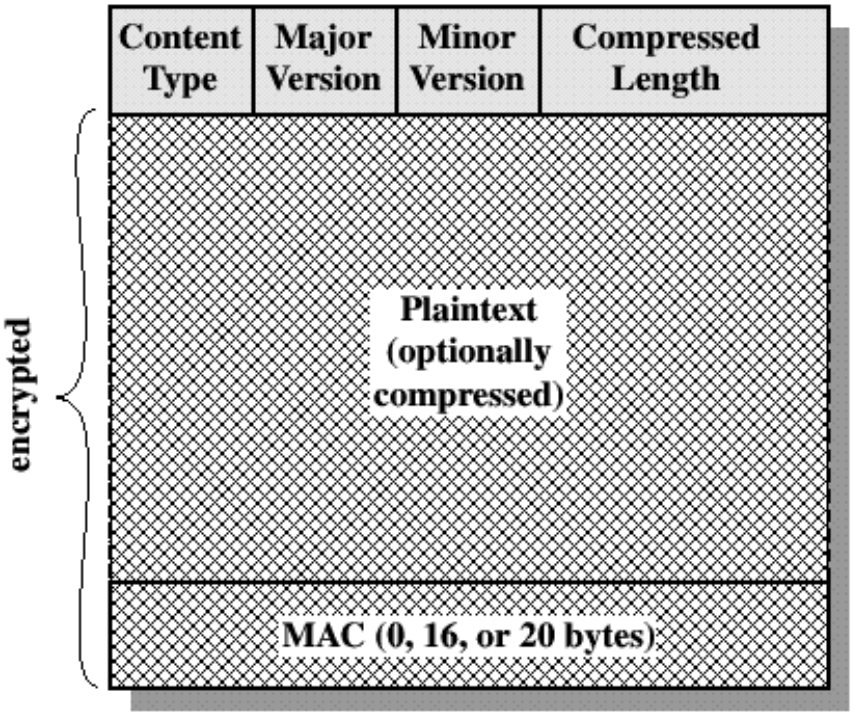
- Encryption: After encrypting the plaintext (or, compressed plaintext) message plus the MAC, the overall message size should not be more than 2^{14} + 2048 bytes

- Valid encryption algorithms:
  - IDEA, DES, DES-40, 3DES, RC2-40, Fortezza
  - RC4-40 & RC4-128



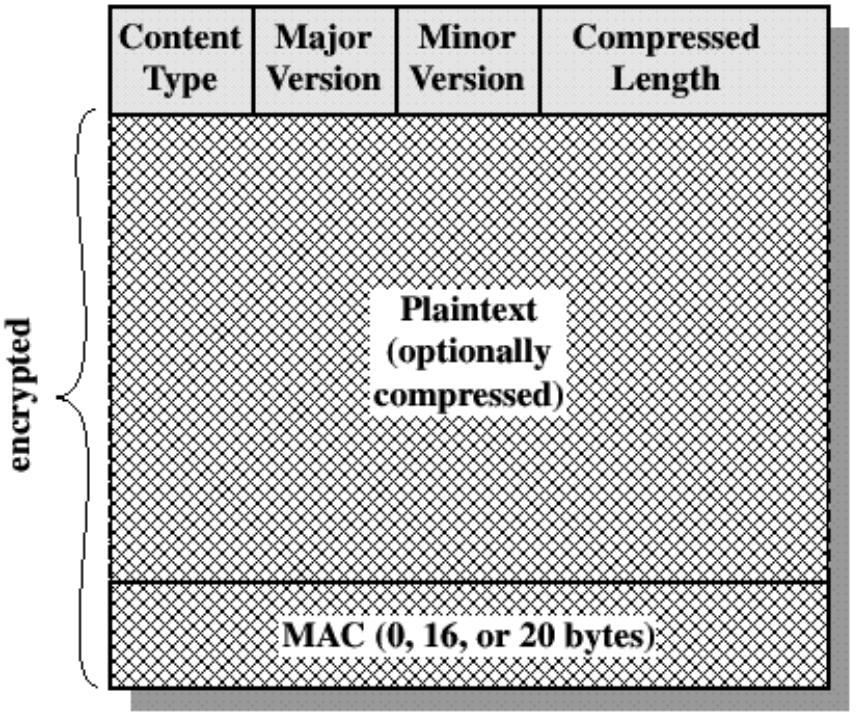
SSL Record Layer Protocol: header

- The pre-pended header consists of the following fields:
  - Content Type: An 8-bit field to define the higher-layer protocol encapsulated.
  - The content types defined are:
    - 20: ChangeCipherSpec
    - 21: Alert
    - 22: Handshake
    - 23: Application



SSL Record Layer Protocol: header

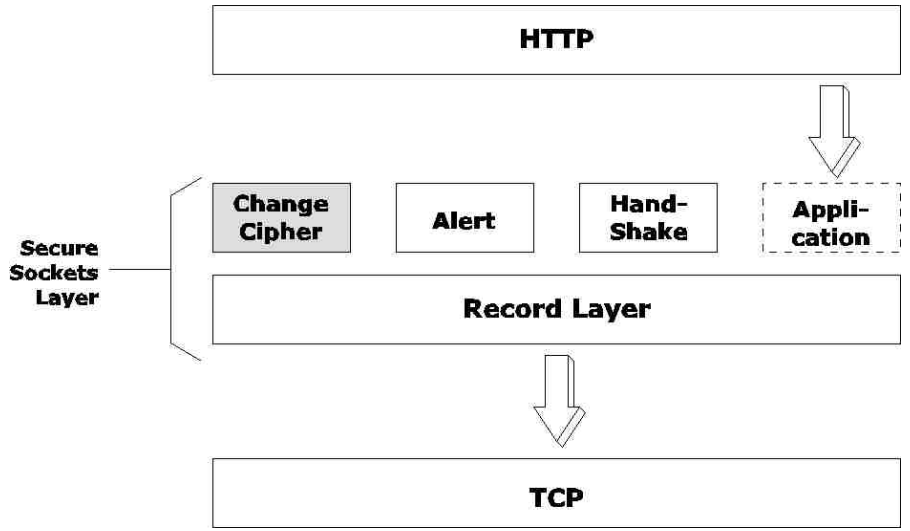
- Major Version: An 8-bit field which indicates the major version of SSL is in use (e.g., 3).
- Minor Version: An 8-bit field which indicates the minor version of SSL is in use (e.g., 0).
- Compressed Length: A 16-bit field which indicates the length of the compressed (plaintext) fragment.

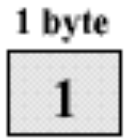


SSL Change Cipher Spec Protocol

- The ChangeCipherSpec Protocol is simplest possible protocol since it has only one message.
  - It consists of a single byte with a value of 1.

- This message causes a pending state to be copied into the current state which updates the cipher suite to be used on the connection.



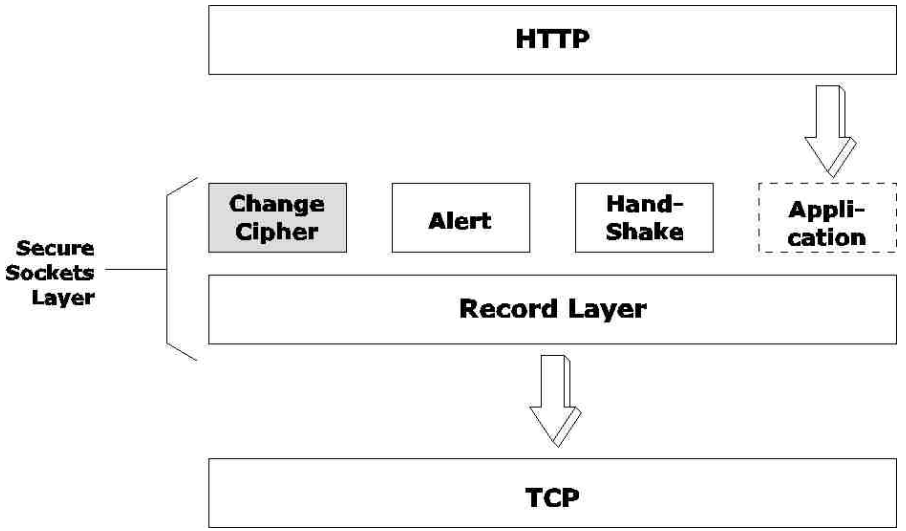


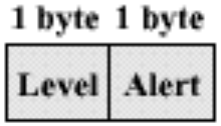
SSL Alert Protocol

- The Alert Protocol is used to signal an error, or caution, condition to the other party in the communication.

- Two bytes
- The first byte takes either of the following two values: :
  - “1” indicates a warning.
  - “2” indicates a fatal error.

- Fatal errors terminate the connection.





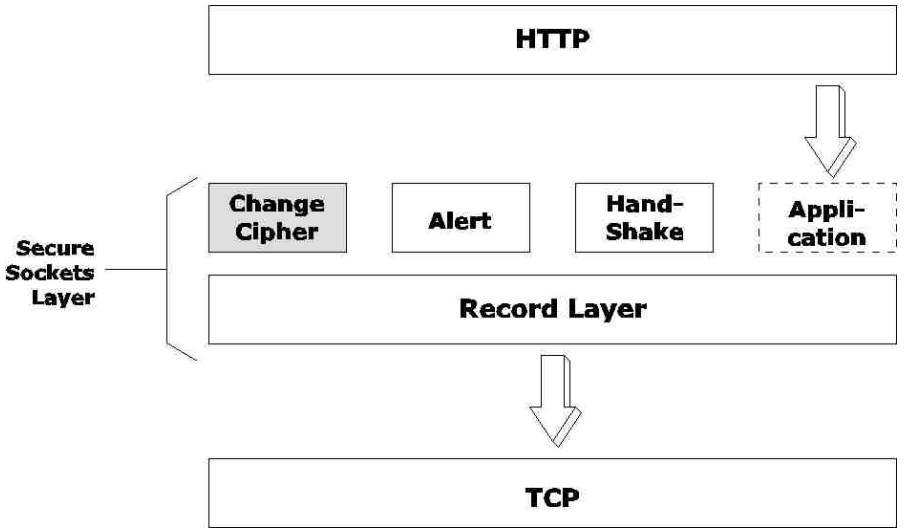
SSL Alert Protocol

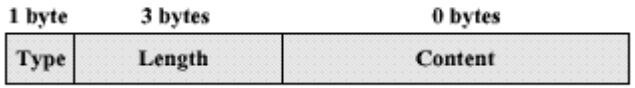
```
Warning or fatal (*)

        close_notify(0),
    *   unexpected_message(10),
    *   bad_record_mac(20),
        decryption_failed(21),
        record_overflow(22),
    *   decompression_failure(30),
    *   handshake_failure(40),
        bad_certificate(42),
        unsupported_certificate(43),
        certificate_revoked(44),
        certificate_expired(45),
        certificate_unknown(46),
    *   illegal_parameter(47),
        unknown_ca(48),
        access_denied(49),
        decode_error(50),
        decrypt_error(51),
        export_restriction(60),
        protocol_version(70),
        insufficient_security(71),
        internal_error(80),
        user_canceled(90),
        no_renegotiation(100)
```

SSL Handshake Protocol

- The most complex part of SSL is the Handshake Protocol.
- provides means for client and server to:
  - Authenticate each other.
  - Negotiate an encryption and MAC algorithm.
  - Negotiate the secret key to be used.
- This protocol consists of a series of messages. Each message consists of three fields:
  - Type: A 8-bit field indicating the type of message (1 of 10).
  - Length: A 3-byte field-length field.
  - Content: >=0-byte field for message parameters.





SSL Handshake Protocol

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 X509 V3 certificates
server_key_exchange	Parameters, signature
certificate_request	Certificate type, Supp. Signature algs, CAs
server_hello_done	Null
certificate_verify	Signature
client_key_exchange	Parameters, signature
finished	Hash value

- The table shows the SSL messages with the appropriate parameters that are used with those messages.

Message Exchange

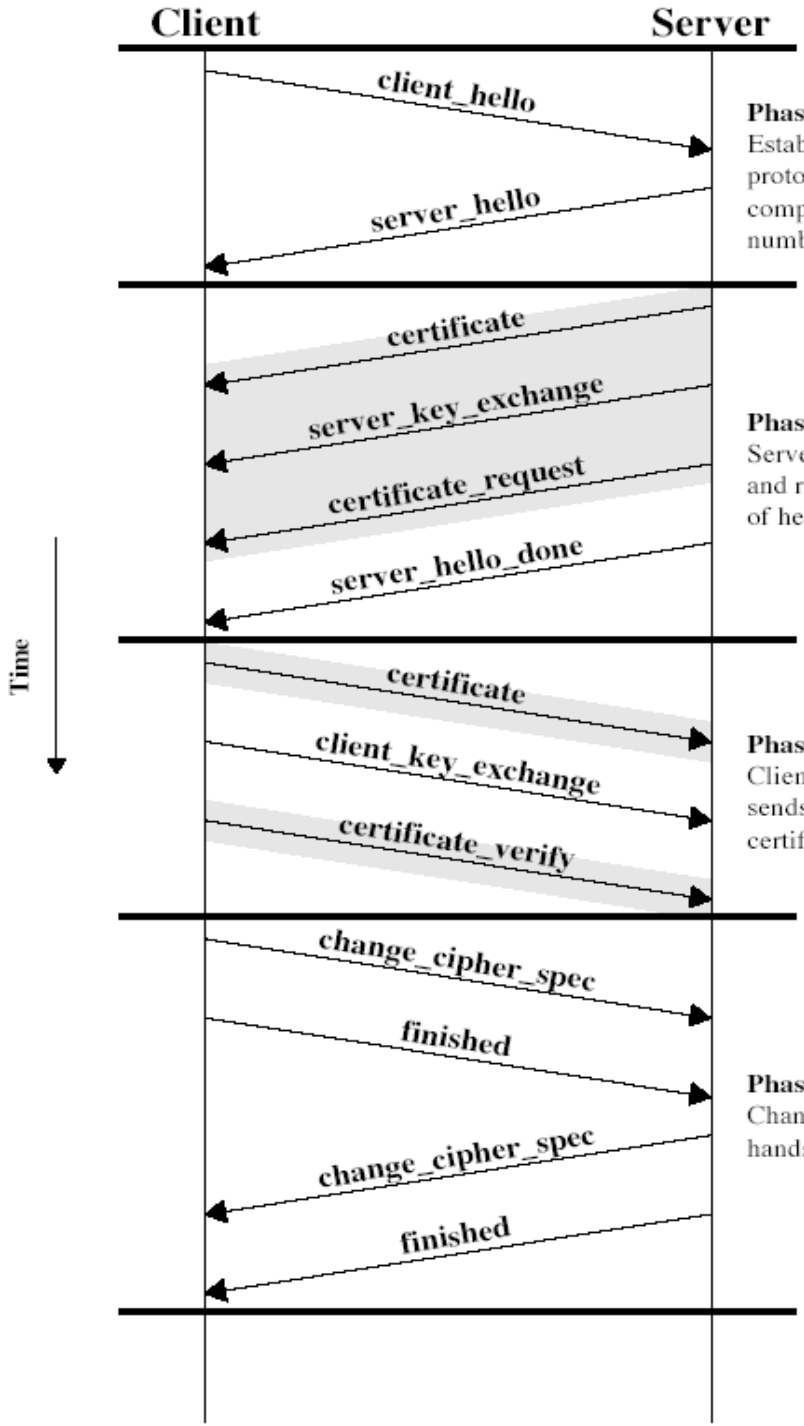
- In SSL, the message exchange process is used to:
  - Authenticate the server.
  - Authenticate the client.
  - Select a cipher.
  - Exchange a key.
  - Transfer data.

- All messages during handshaking and after, are sent over the SSL Record Protocol layer.

SSL Handshake Protocol

- **Phase 1:**
  - Establish security capabilities
- **Phase 2:**
  - Server authentication and key exchange
- **Phase 3:**
  - Client authentication and key exchange
- **Phase 4:**
  - Finish

SSL Handshake Protocol



Phase 1
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.

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

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

Phase 4
Change cipher suite and finish handshake protocol.

Note: Shaded transfers are optional or situation-dependent messages that are not always sent.

SSL Handshake Protocol

- *hello_request* (not shown) can be sent anytime from server to client to request client to start handshake protocol to renegotiate session when convenient
- can be ignored by client
  - if already negotiating a session
  - Do not want to renegotiate a session
    - client may respond with a *no_renegotiation* alert

PHASE 1: establish security capabilities

- The client-hello message sends the server some challenge-data and a list of ciphers which the client can support. The challenge-data is used to authenticate the server later on.
- client_hello parameters
  - 4 byte timestamp, 28 byte random value
  - session ID:
    - non-zero for new connection on existing session
    - zero for new connection on new session
  - client version: highest version
  - cipher_suite list: ordered list
  - compression list: ordered list

PHASE 1: establish security capabilities

- cipher suite
  - key exchange method
    - RSA: requires receiver's public-key certificates
    - Fixed DH: requires both sides to have public-key certificates
    - Ephemeral DH: signed ephemeral keys are exchanged, need signature keys and public-key certificates on both sides
    - Anonymous DH: no authentication of DH keys, susceptible to man-in-the-middle attack
    - Fortezza: Fortezza key exchange (ignore from now on)

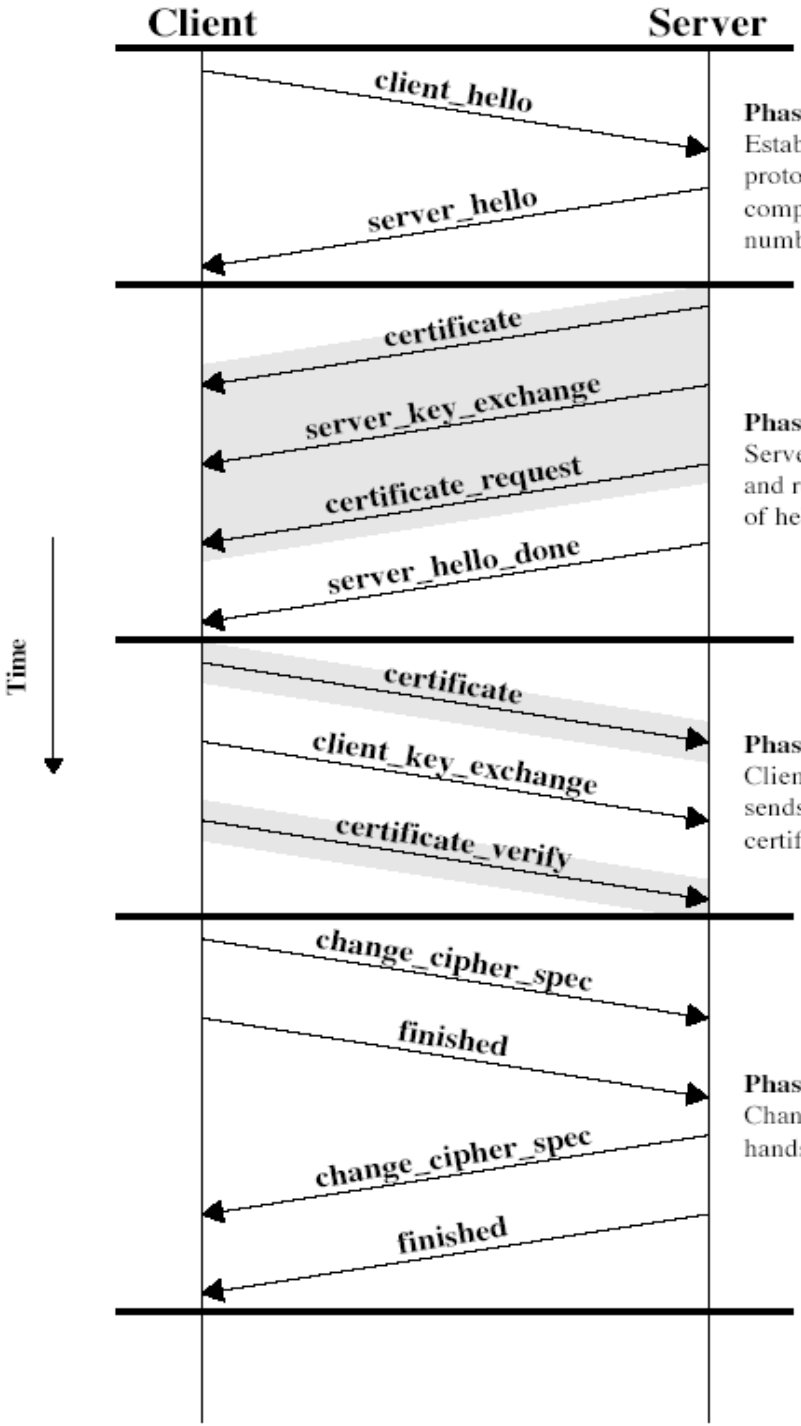
PHASE 1: establish security capabilities

- cipher suite
  - cipher spec
    - CipherAlgorithm: RC4, RC2, DES, 3DES, DES40, IDEA, Fortezza
    - MACAlgorithm: MD5 or SHA-1
    - CipherType: stream or block
    - IsExportable: true or false
    - HashSize: 0, 16 (for MD5) or 20 (for SHA-1) bytes
    - Key Material: used to generate write keys
    - IV Size: size of IV for CBC

PHASE 1: establish security capabilities

- The server-hello message returns a connection-id, a server certificate and a modified list of ciphers which the client and server can both support.
- server_hello parameters
  - 32 byte random value
  - session ID:
    - new or reuse
  - version
    - lower of client suggested and highest supported
  - cipher_suite list: single choice
  - compression list: single choice

SSL Handshake Protocol



Phase 1
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.

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

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

Phase 4
Change cipher suite and finish handshake protocol.

Note: Shaded transfers are optional or situation-dependent messages that are not always sent.

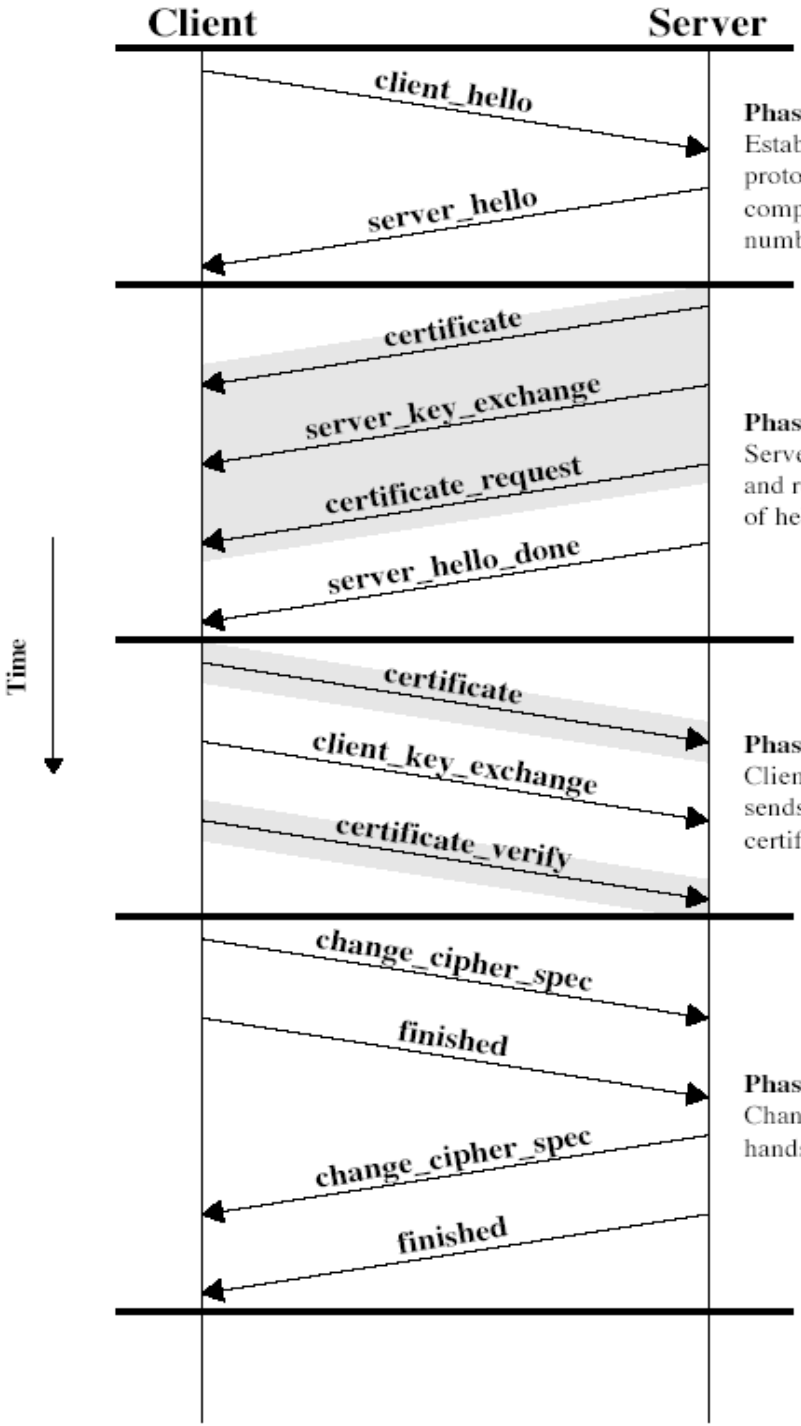
PHASE 2: server authentication & key exchange

- Certificate message
  - used by the client to obtain the servers public key and verify the identity of the server using any certification authority certificates it has.
  - server's X.509v3 certificate followed by optional chain of certificates
  - required for RSA, Fixed DH, Ephemeral DH but not for Anonymous DH
- Server Key Exchange message
  - not needed for RSA, Fixed DH
  - needed for Anonymous DH, Ephemeral DH
  - needed for RSA where server has signature-only key
    - server sends temporary RSA public encryption key to client

PHASE 2: server authentication & key exchange

- Server Key Exchange message
  - signed by the server
  - signature is on hash of
    - ClientHello.random, ServerHello.random
    - Server Key Exchange parameters
- Certificate Request message
  - request a certificate from client
  - specifies Certificate Type and Certificate Authorities
    - certificate type specifies public-key algorithm and use
- Server Done message
  - ends phase 2, always required, no parameters

SSL Handshake Protocol



Phase 1
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.

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

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

Phase 4
Change cipher suite and finish handshake protocol.

Note: Shaded transfers are optional or situation-dependent messages that are not always sent.

PHASE 3: client authentication & key exchange

- Certificate message
  - sent if server has requested certificate and client has appropriate certificate
    - otherwise send *no_certificate* alert
- Client Key Exchange message
  - content depends on type of key exchange (see next slide)
- Certificate Verify message
  - can be optionally sent following a client certificate with signing capability
  - signs hash of master secret (established by key exchange) and all handshake messages so far
  - provides evidence of possessing private key corresponding to certificate

PHASE 3: client authentication & key exchange

- Client Key Exchange message
  - RSA
    - client generates 48-byte pre-master secret, encrypts with server's RSA public key (from server certificate or temporary key from Server Key Exchange message)
  - Ephemeral or Anonymous DH
    - client's public DH value
  - Fixed DH
    - null, public key previously sent in Certificate Message

POST PHASE 3: crypto computation

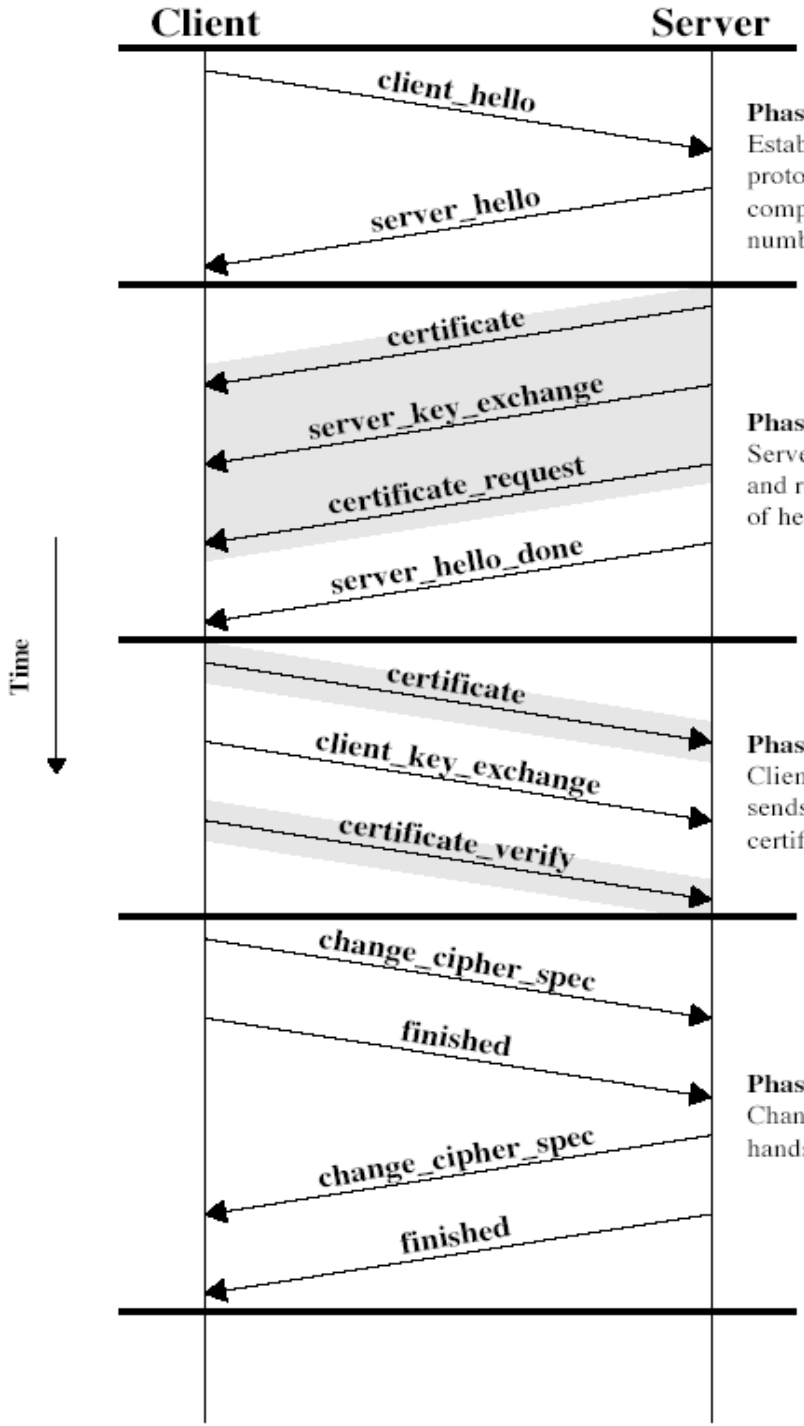
- 48 byte pre master secret
  - RSA
    - generated by client
    - sent encrypted either by server's public RSA from certificate or by temp. RSA from server_key_exchange message to server
  - DH
    - both sides compute the same value
    - each side uses its own private value and the other sides public value

```
master_secret = PRF(pre_master_secret, "master secret",
ClientHello.random + ServerHello.random) [0..47];
```

pre_master_secret: 48 bytes

PRF as sequences and nestings of MD5 and SHA

SSL Handshake Protocol



Phase 1
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.

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

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

Phase 4
Change cipher suite and finish handshake protocol.

Note: Shaded transfers are optional or situation-dependent messages that are not always sent.

PHASE 4: finish

- Change Cipher Spec message
  - not considered part of handshake protocol but in some sense is part of it
  - 1 byte message protected by current state
  - copies pending state to current state
    - sender copies write pending state to write current state
    - receiver copies read pending state to read current state
  - immediately send finished message under new current state
- Finished message
  - sent under new algorithms and keys
  - content is hash of all previous messages and master secret

PHASE 4: finished message

```
verify_data
    PRF(master_secret, finished_label, MD5(handshake_messages)+
    SHA-1(handshake_messages)) [0..11];

finished_label
    For Finished messages sent by the client, the string "client
    finished". For Finished messages sent by the server, the
    string "server finished".

handshake_messages
    All of the data from all handshake messages up to but not
    including this message. This is only data visible at the
    handshake layer and does not include record layer headers.
```

SSL Summary

- **Handshake protocol**: complicated
  - embodies key exchange & authentication
  - 10 message types
- **Record protocol**: straightforward
  - fragment, compress, MAC, encrypt
- **Change Cipher Spec protocol**: straightforward
  - single 1 byte message with value 1
  - could be considered part of handshake protocol
- **Alert protocol**: straightforward
  - 2 byte messages
    - 1 byte alert level- fatal or warning; 1 byte alert code

UPDATES

TLS 1.1 (SSL 3.2)

defined in RFC 4346 in Apr 2006, an update from TLS version 1.0.

- Added protection against Cipher block chaining (CBC) attacks.
- The implicit Initialization Vector (IV) replaced with an explicit IV
- Deprecated as of June 2019.

TLS 1.2 (SSL 3.3)

defined in RFC 5246 in Aug 2008, based on earlier TLS 1.1 spec. Major differences:

- The MD5-SHA-1 combination in the pseudorandom function (PRF), in the Finished message hash, and in the digitally-signed element replaced with SHA-256, with option to use cipher-suite specified *.

UPDATES

TLS 1.2 (SSL 3.3) (cont.)

- Enhance client's and server's ability to specify which hash and signature algorithms they will accept (37 new cipher suites).
- Expansion of support for authenticated encryption ciphers
- TLS Extensions definition and Advanced Encryption Standard CipherSuites were added.
- Updated 3/2011 to forbid backward compatibility with SSL2.0
- Recommended version since 2008

TLS 1.3 draft as of March 2016

- https://tools.ietf.org/html/draft-ietf-tls-tls13-12
  - Deprecate SHA-1 with signatures and DH
  - Remove ChangeCipherSpec and support for compression

UPDATES

TLS 1.3 consensus August 2018

- https://tools.ietf.org/html/rfc8446
  - Server side always authenticated, client side optional
  - All handshake messages after ServerHello encrypted with keys established with ECDHE
  - Removed
    - RSA for key exchange
    - RC4, 3DES for encryption
    - MD5, SHA-224 hashes with signatures
    - Cipher mode AES-CBC
  - HMAC-based (SHA-256 or -384) Hashed-Key Derivation Function (HKDF)
  - Support for 0-RTT if client and server share PSK