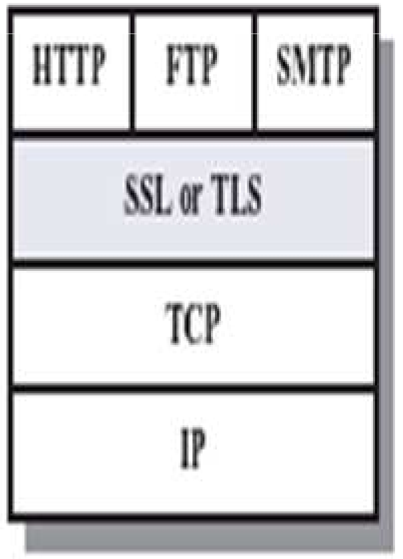
# Transport Layer Security

SSLv3 and TLSv1



(b) Transport Level

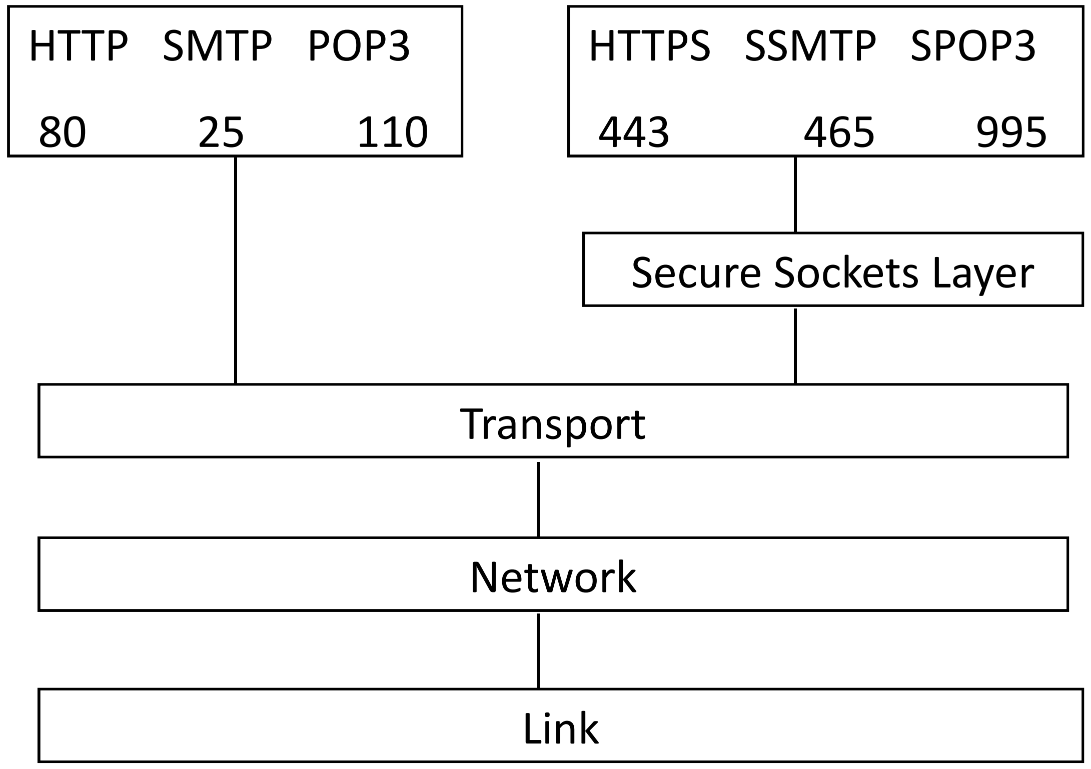
# Transport Layer Security Protocols

- Secure Sockets Layer version 3 (SSLv3)
  - Introduced by Netscape Communications Corporation
  - 1995
  - For transmitting private documents via the Internet
  - SSLeay – open source SSL implementation
  - Implements SSLv2 and SSLv3 and TLSv1 as of the release of SSLeay-0.9.0
  - SSLv3 was designed with public review and input from industry
- Transport Layer Security (TLS)
  - TLSv1 is very closely compatible with SSLv3
  - Provides communications privacy and data integrity
- Allow client/server applications to communicate in such a way that they prevent eavesdropping, tampering or message forgery

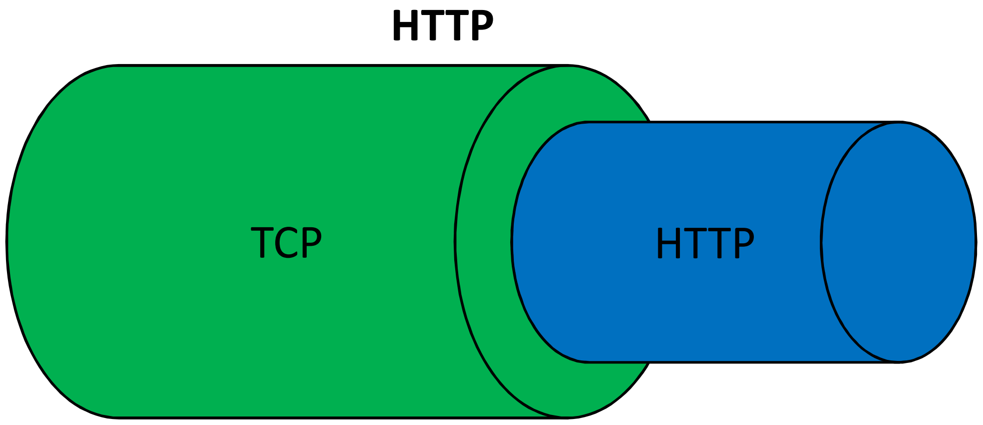
## Secure Socket Layer

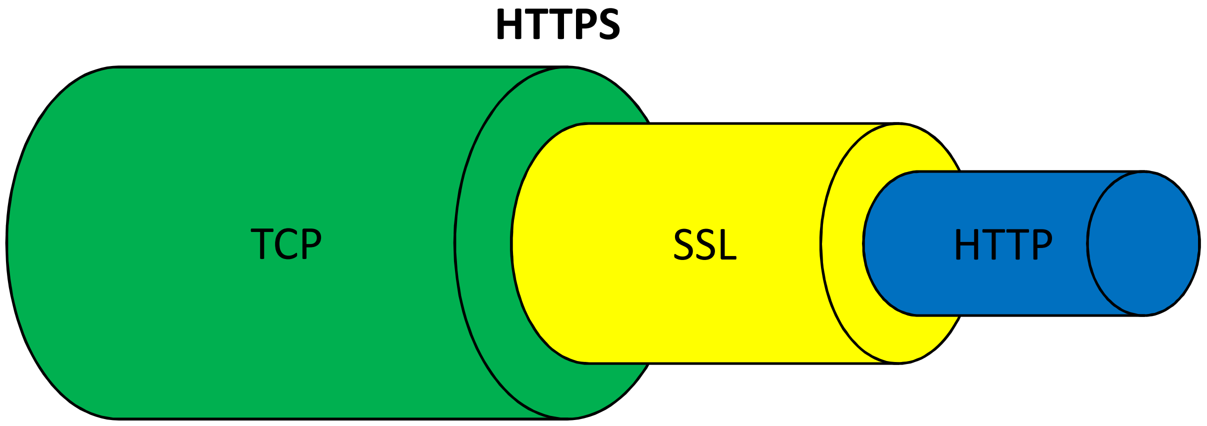
- The SSL Security protocol provides
  - Data encryption - to establish an encrypted connection
  - Server authentication - server to authenticate itself to client; the client to authenticate itself to the server
  - Message integrity
  - Provide privacy and reliability between two communicating applications
- SSL is built into all major browsers and web servers widely used on the Internet
- Websites use the protocol to obtain confidential user information, such as credit card numbers

# Where SSL Fits



## How SSL works

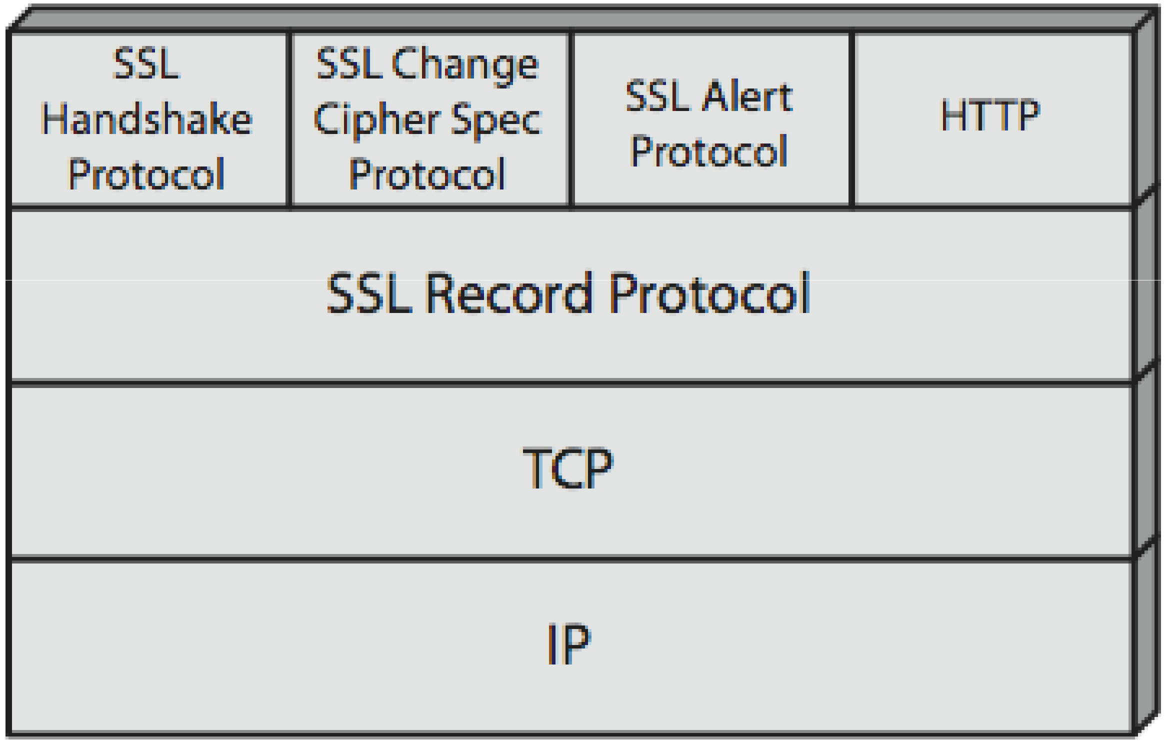




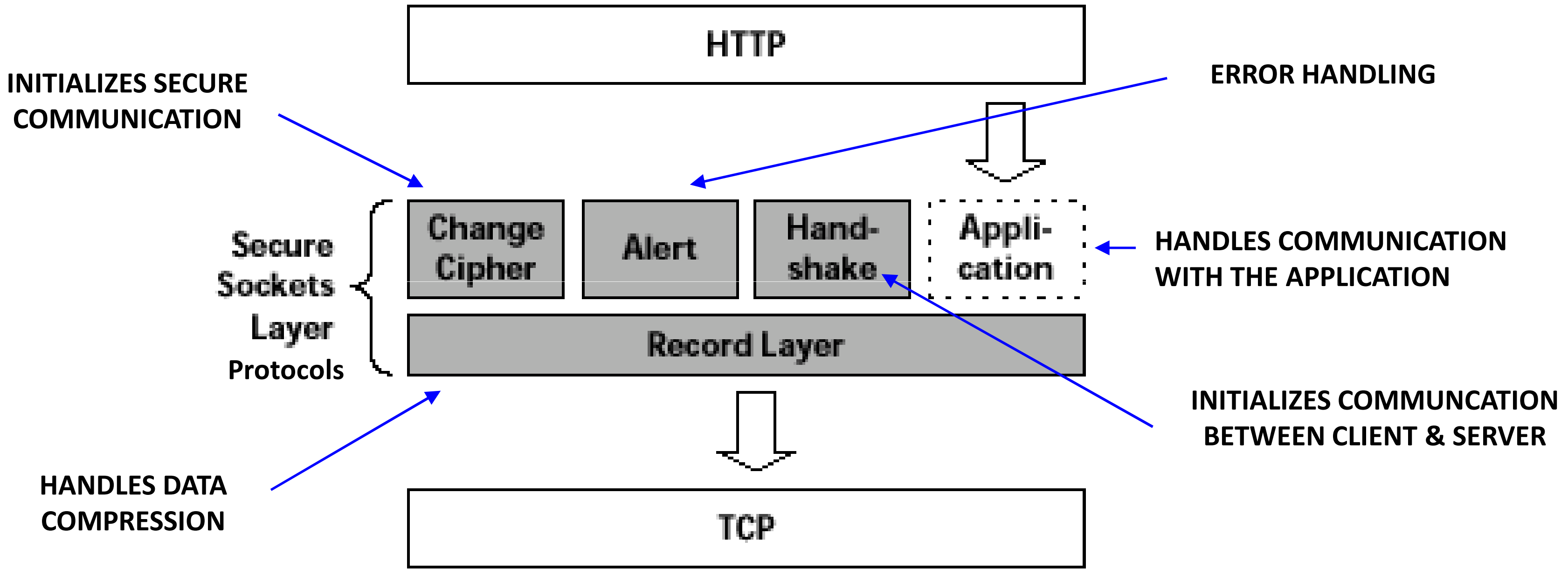
## SSL

- SSL sits on top of TCP
- Supports any application Layer Protocol
- TCP  provides a reliable end-to-end service.
- TCP & SSL together provides a reliable & secure end-to-end service.
- A higher-level protocol can layer on top of the SSL protocol transparently
  - For example, the HyperText Transfer Protocol (HTTP)
- HTTPS:  HTTP over SSL (or TLS)
  - Http on port 80
  - Https on port 443

## SSL Architecture



# SSL Architecure


HTTP
INITIALIZES SECURE COMMUNICATION
ERROR HANDLING
Secure Sockets Layer Protocols
Change Cipher
Alert
Hand-shake
Appli-cation
HANDLES COMMUNICATION WITH THE APPLICATION
Record Layer
INITIALIZES COMMUNCATION BETWEEN CLIENT & SERVER
HANDLES DATA COMPRESSION
TCP

# SSL Protocol Stack

1. The **first layer** is the higher layer - are used in the management of SSL exchanges

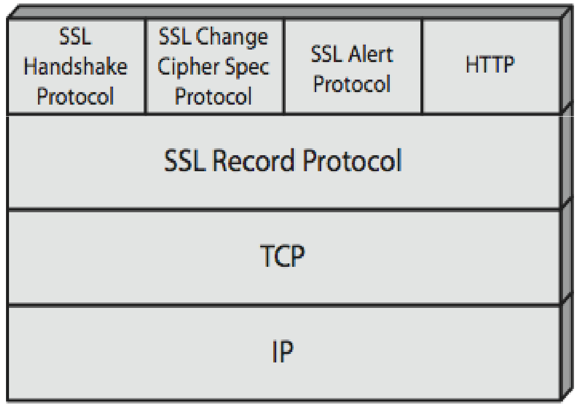
   Composed of S

   - SSL Handshake Protocol
   - SSL Change Cipher Spec Protocol
   - SSL Alert Protocol
   - HTTP

2. The **second layer** is the lower laye

   Composed of the

   - SSL Record Protocol
   - TCP
   - IP



# SSL Protocol (SSL V3.0)

- **Two layer Protocol**
  - **SSL Record Protocol**
    - Used for encapsulation of various higher level protocols
    - Upper-layer application message
      - **Transmitted** is fragmented into manageable blocks, optionally compressed, applies an MAC, encrypts, adds a header, and transmits the result to TCP segment
      - **Received** data is decrypted, verified, decompressed, reassembled, and then delivered to higher-level clients
    - The SSL Record Protocol is application protocol independent
  - **SSL Handshake Protocol**
    - Allows server and client to authenticate, negotiate encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data

## SSL Record Protocol Operation

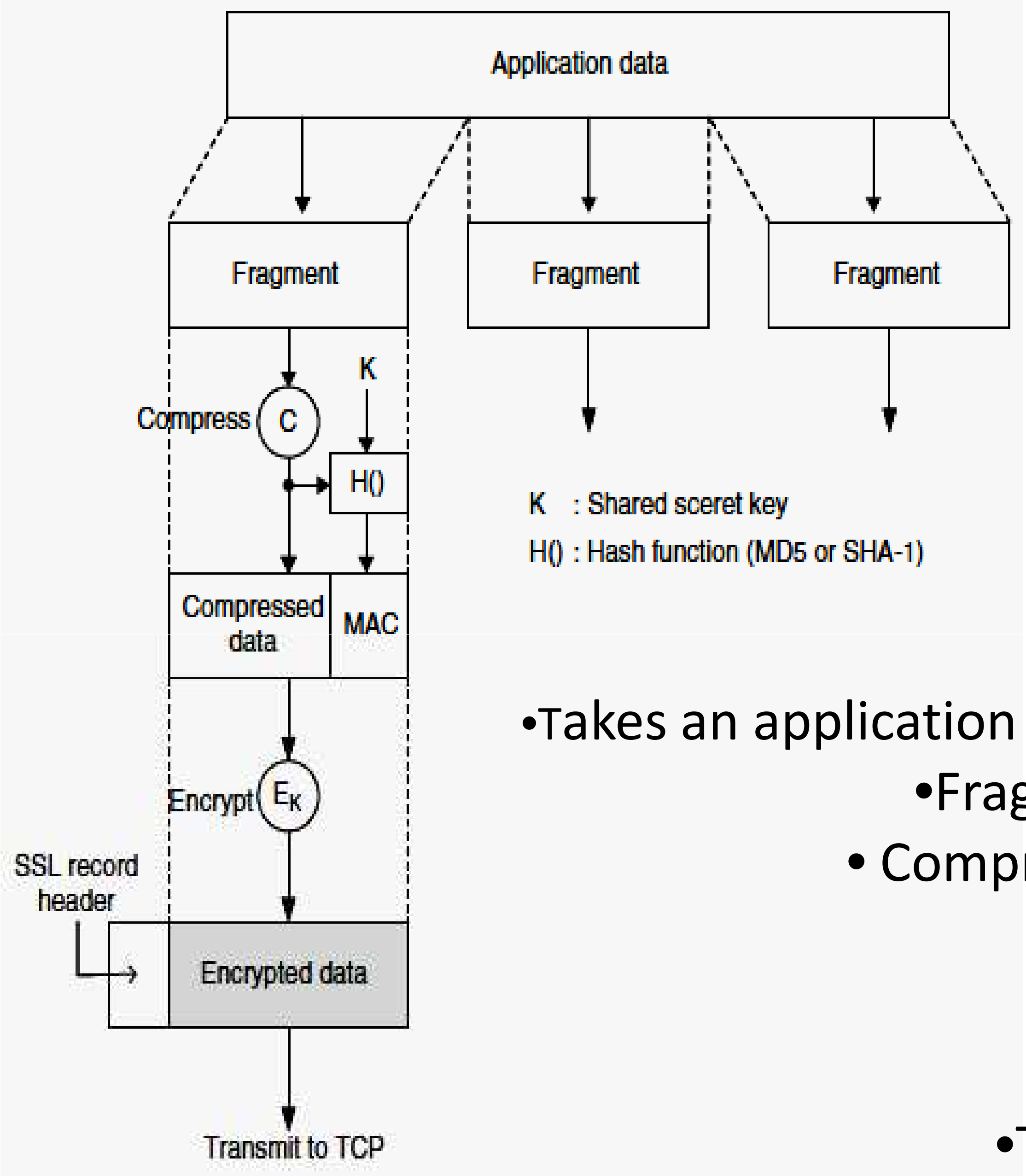


Figure 8.2 The overall operation of the SSL Record Protocol.

**Operations:**

- Takes an application message to be transmitted
- Fragments the data into blocks
- Compresses the data (optionally)
- Applies a MAC
- Encrypts
- Adds a header
- Transmits the resulting unit

## SSL Record Protocol Operation

**Fragmentation:**

- Fragments information from higher layer into blocks of $2^{14}$ bytes or less

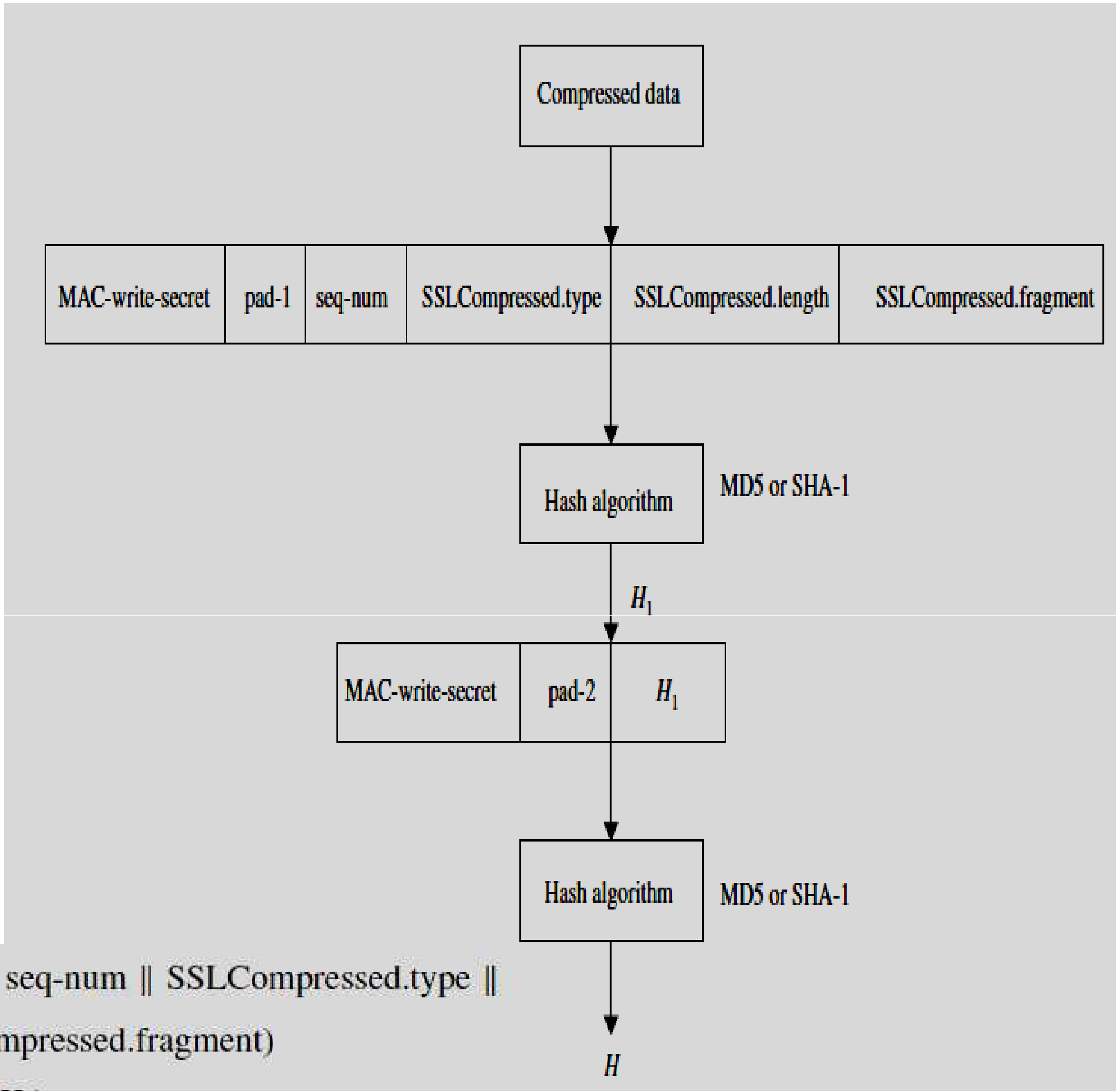
**Compression and decompression:**

- Compression algorithm defined in the current session state is used
- The compression algorithm translates an SSL Plaintext structure into an SSL Compressed structure
- The compression processing should ensure that an SSLPlaintext structure is identical after being compressed and decompressed
- Compression must be lossless and may not increase the current length by more than 1024 bytes
- Compression is essentially useful when encryption is applied
- Compression should be applied before encryption and MAC computation

## SSL Record Protocol Operation

**MAC:**

The MAC is computed before encryption



$$H_1 = \text{hash}(\text{MAC-write-secret} \parallel \text{pad-1} \parallel \text{seq-num} \parallel \text{SSLCompressed.type} \parallel \text{SSLCompressed.length} \parallel \text{SSLCompressed.fragment})$$

$$H = \text{hash}(\text{MAC-write-secret} \parallel \text{pad-2} \parallel H_1)$$

**Figure 8.3** Computation of MAC over the compressed data.

## SSL Record Protocol Operation

| | |
|---|---|
| MAC-write-secret: | Shared secret key |
| Hash ($H_1$ and $H$): | Cryptographic hash algorithm; either MD5 or SHA-1 |
| Pad-1: | The byte 0x36 (0011 0110) repeated 48 times (384 bits) for MD5 and 40 times (320 bits) for SHA-1 |
| Pad-2: | The byte 0x5C (0101 1100) repeated 48 times for MD5 and 40 times for SHA-1 |
| Seq-num: | The sequence number for this message |
| SSLCompressed.type: | The higher-level protocol used to process this fragment |
| SSLCompressed.length: | The length of the compressed fragment |
| SSLCompressed.fragment: | The compressed fragment (the plaintext fragment if not compressed) |
| \|\|: | concatenation symbol |

$$H_1 = \text{hash}(\text{MAC-write-secret} \,\|\, \text{pad-1} \,\|\, \text{seq-num} \,\|\, \text{SSLCompressed.type} \,\|\, \text{SSLCompressed.length} \,\|\, \text{SSLCompressed.fragment})$$

$$H = \text{hash}(\text{MAC-write-secret} \,\|\, \text{pad-2} \,\|\, H_1)$$

## SSL Record Protocol Operation

**Encryption:**

- The compressed message plus the MAC are encrypted using symmetric encryption
- The block ciphers being used as encryption algorithms are:
  - DES(56), Triple DES(168), IDEA(128), RC5(variable) and Fortezza(80)
  - Fortezza is a PCMCIA card that provides both encryption and digital signing.
  - Personal Computer Memory Card International Association (PCMCIA)

## SSL Record Protocol Operation

**Append SSL record header:**

- The final processing of the SSL Record Protocol
- The composed fields consist of:
  - Content type (8 bits): This field is the higher-layer protocol used to process the enclosed fragment
  - Major version (8 bits): This field indicates the major version of SSL in use. For SSLv3, the value is 3
  - Minor version (8 bits): This field indicates the minor version of SSL in use. For SSLv3, the value is 0
  - Compressed length (16 bits): This field indicates the length in bytes of the plaintext fragment or compressed fragment if compression is required

## SSL Record Protocol Format

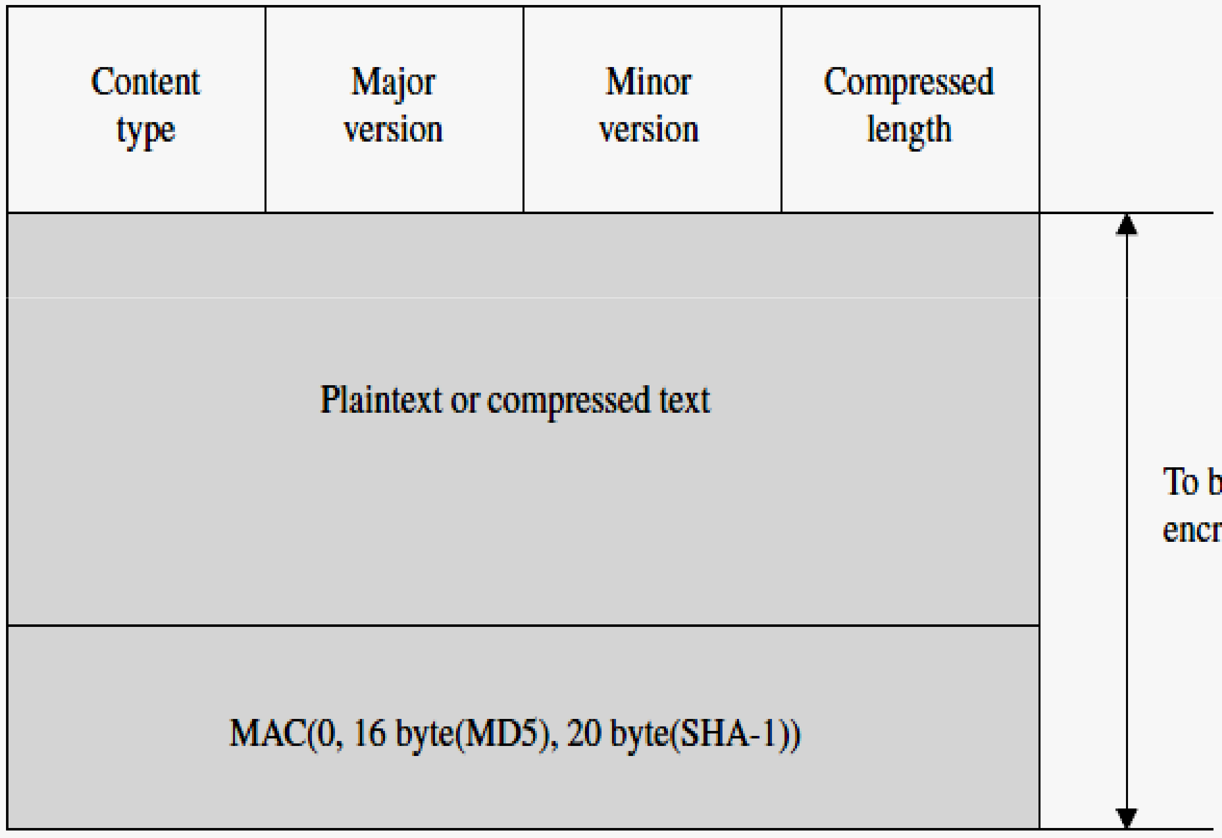


**Figure 8.4** SSL Record Protocol format.

## SSL – Session and Connection States

- **Connection States**
  - An SSL session may include multiple secure connections
  - Client may have multiple simultaneous sessions each with multiple connections

- **Session States**
  - It is an association between a client and a server
  - Created by the Handshake Protocol
  - Define a set of cryptographic security parameters that is shared among multiple connections
  - Helps in avoiding expensive negotiation of new security parameters for each connection

## SSL — Session and Connection States

- **Session States**
  - Coordinates the states of the client and server
  - Logical state : current operating state and pending state
    - When the client or server
      - **receives** a change cipher spec message, it copies the pending read state into the current read state
      - **sends** a change cipher spec message, it copies the pending write state into the current write state

- When the handshake negotiation is completed, the client and server exchange change cipher spec messages, and they then communicate using the newly agreed-upon cipher spec

## SSL-specific protocols

- Change Cipher Spec Protocol
- Alert Protocol
- Handshake Protocol

## SSL Change Cipher Spec Protocol

- **Simplest** of the three SSL-specific protocols
- Consists of **a single message**
- The message consists of a single byte of **value 1**
- Sent by both the **client and server**
- Notify the receiving party that subsequent records will be protected under the just-negotiated CipherSpec and keys
- Reception of this message causes **the pending state to be copied** into the current state, which updates the cipher suite to be used on this connection
- **Compressed & Encrypted** like all SSL data

## SSL Alert Protocol

- Conveys SSL-related alerts to peer entity
- Convey the **severity** of the message and a **description** of the alert
- SSL Alert messages are compressed & encrypted like all SSL data
- Alert messages consist of 2 bytes
- First byte - severity
  - Value may be **warning or fatal** - to convey the seriousness of the message
  - If the level is fatal, SSL immediately terminates the connection
  - Other connections in session may continue
  - **Session ID invalidated** to prevent failed session to open new connection
- Second byte - specific alert
  - Value - code that indicates the specific alert

•Uunexpected message
•Bad record mac
•Decompression failure
•Handshake failure
•Illegal parameter
•Close notify

•No certificate
•Bad certificate
•Unsupported certificate
•Certificate revoked
•Certificate expired
•Certificate unknown

## SSL Handshake Protocol

- Provides three services for SSL connections between the server and client.
- Allows the client/server
  - Agree on a protocol version to authenticate each other
  - Negotiate an encryption algorithm
  - Agree on cryptographic keys for protecting data

## SSL Handshake Protocol

- Series of messages exchanged between the client and server
- Called handshake messages
- Establish a logical connection between client and server
- Comprises a series of messages in 4 phases
  - Establish Security Capabilities
  - Server Authentication and Key Exchange
  - Client Authentication and Key Exchange
  - Change *CipherSpec* and Finish

## SSL Handshake Protocol

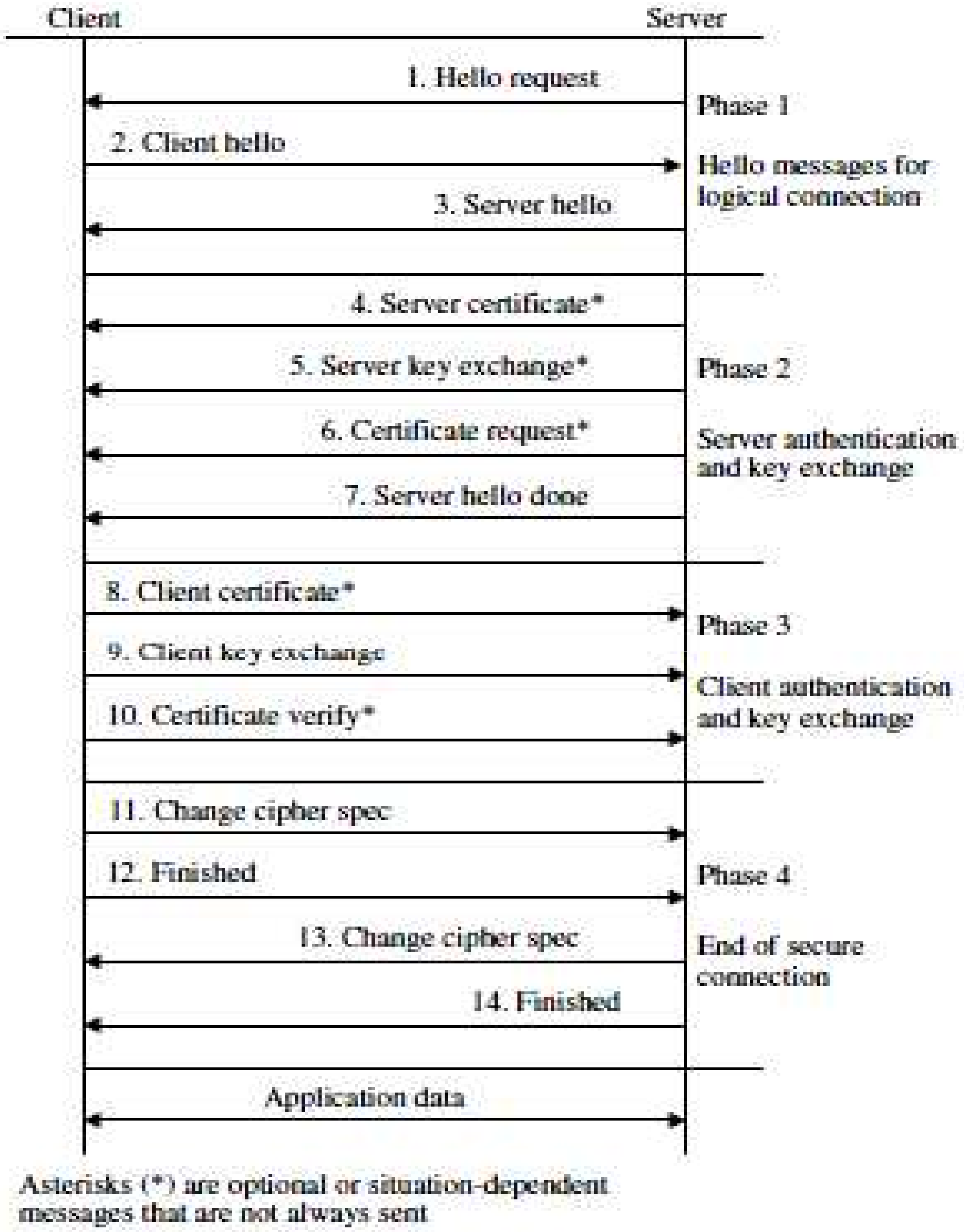
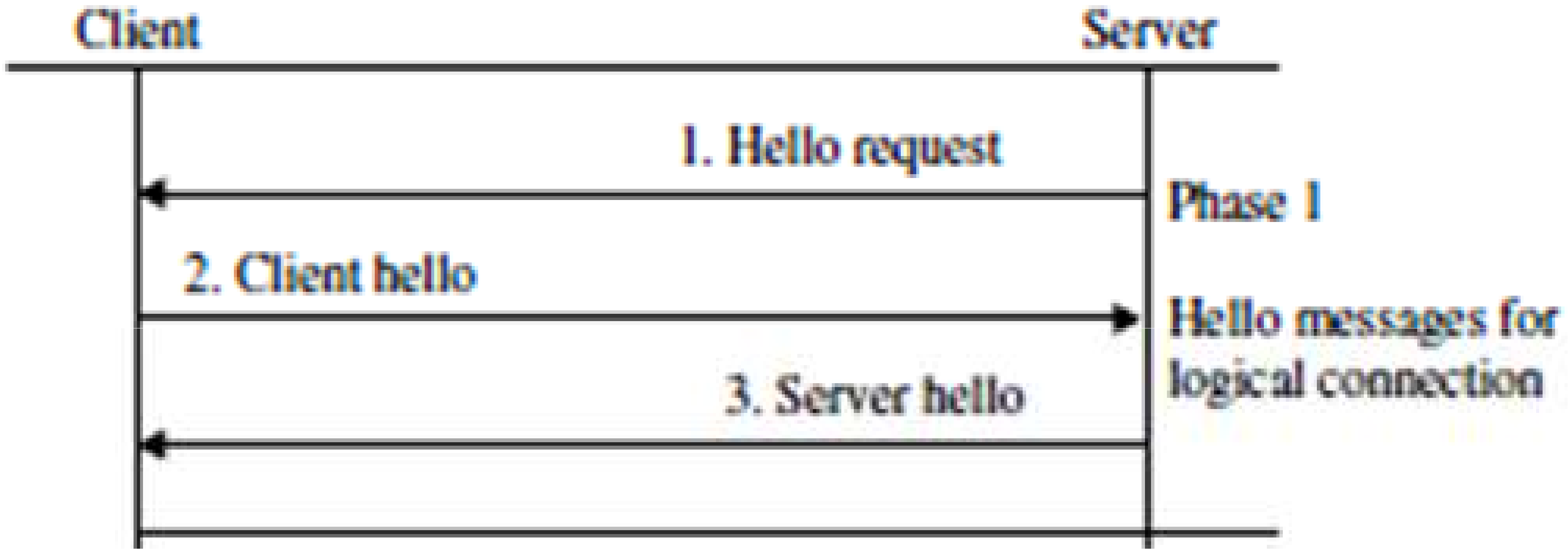


**Figure 8.5** SSL Handshake Protocol.

# SSL Handshake Protocol



## SSL Handshake Protocol

*Phase 1: Hello Messages for Logical Connection*

- Client sends a client hello message
- Server must respond with a server hello message
- Otherwise a fatal error will occur and the connection will fail
- These hello messages are used to establish security enhancement capabilities between client and server
- Hello messages establish the following attributes:
  - Protocol version
  - Random values (ClientHello.random and ServerHello.random),
  - Session ID
  - Cipher suite
  - Compression method

# SSLHP –Phase 1

**Hello messages**

1.Hello request

2. Client Hello

3. Server Hello

*1. Hello request:*

- *This message is sent by the server at any time*
- *May be ignored* by the client if the Handshake Protocol is already underway
- A client who receives a hello request while in a handshake negotiation state should simply ignore the message

# SSLHP –Phase 1

- **Hello messages**

  *2. Client hello:*

  - *Initiated by the client*
  - *A client sends a client hello* message using the session ID of the session to be resumed
  - The client sends a client hello message with the following parameters:
    - *Client version*
    - *Random*
    - *Session ID*
    - *Cipher suites*
    - *Compression method*
  - The server then checks its session cache for a match
  - If a match is found, the server will send a server hello message with the same session ID value

# SSLHP –Phase 1

- **Hello messages**

  *2. Client hello:*

  - *Parameters*
    - *Client version:*
      - » *Version of the SSL protocol in which the client wishes to* communicate during this session
      - » Most recent (highest-valued) version supported by the client
      - » The value of this version will be 3.0
    - *Random:*
      - » *28 bytes* secure random number generator
    - *Session ID:*
      - » *This is the identity of a session when the client wishes to use this connection*
      - » A nonzero value indicates
        - that the client wishes to update the parameters of an existing connection
        - create a new connection in this session
      - » A zero value indicates that the client wishes to establish a new connection in a new session
    - *Cipher suites:*
      - » *This is a list of the cryptographic options supported by the client,* with the client's first preference first
      - » The single cipher suite is an element of a list selected by the server from the list in ClientHello.cipher suites
    - *Compression method:*
      - » *This is a list of the compression methods supported by the* client, sorted by client preference.

## SSLHP –Phase 1

- **Hello messages**

  *3. Server hello:*

  - Response to a client hello message
  - Sent to client when server has found an acceptable set of algorithms in client hello
  - If it is unable to find such a match, it will respond with a handshake failure alert
  - The structure of this message consists of:
    - Server version
    - Random
    - Session ID
    - Cipher suite
    - Compression method

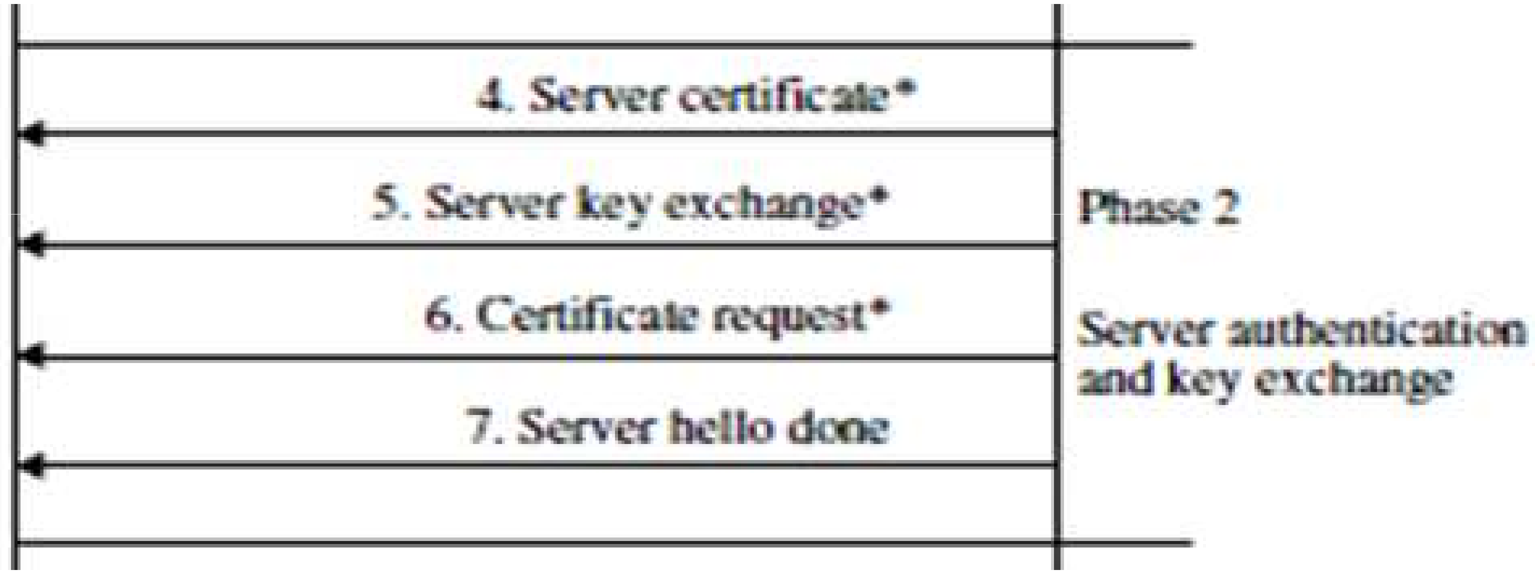
# SSLHP –Phase 1

- **Hello messages**

  *3. Server hello:*

  - *Server version:*
    - *This field will contain the lower-valued version suggested by the* client in the client hello and the highest-valued version supported by the server
    - The value of this version is 3.0.
  - *Random:*
    - *This structure is generated by the server and must be different from* ClientHello.random.
  - *Session ID:*
    - *Holds the Identity of the session corresponding to this* connection
    - If the ClientHello.session id is non-empty, the server will look in its session cache for a match
    - If a match is found and the server is willing to establish the new connection using the specified session state, the server will respond with the same value as was supplied by the client
    - This indicates a resumed session and dictates that the parties must proceed directly to the finished messages
  - *Cipher suite:*
    - *This is the single cipher suite selected by the server from the list in ClientHello.cipher suites*
    - For a resumed session, this field is the value from the state of the session being resumed.
  - *Compression method:*
    - *This is the single compression algorithm selected by the* server from the list in ClientHello.compression methods
    - For a resumed sessions, this field is the value from the resumed session state

## SSL Handshake Protocol



# SSLHP –Phase 2

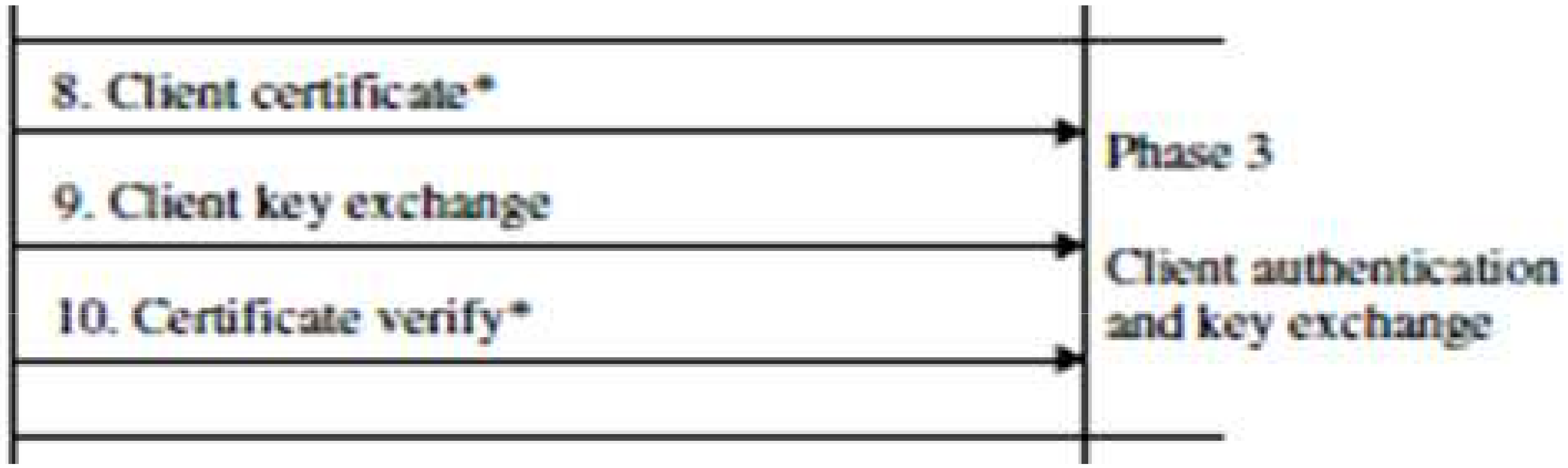
## *Phase 2: Server Authentication and Key Exchange*

- Server send
  - **Certificate**
  - **key exchange message**
  - May **request a certificate** from the client
  - server **hello done message**, indicating that the hello message phase of the handshake is complete
- The server will then wait for a client response
- If the server has sent a certificate request message, the client must send the certificate message

# SSLHP –Phase 2

- *Server certificate:*
  - *If the server is to be authenticated, it must send a certificate immediately* following the server hello message
  - The certificate type must be appropriate for the selected cipher suite's key exchange algorithm
  - Generally an X.509 v3 certificate
  - It must contain a key which matches the key exchange method
  - The signing algorithm for the certificate must be the same as the algorithm for the certificate key
- *Server key exchange message:*
  - *The server key exchange message is sent by the server* only when it is required
  - This message is not used if the server certificate contains Diffie–Hellman parameters, or RSA key exchange
- *Certificate request message:*
  - *server can optionally request a certificate* from the client, if appropriate for the selected cipher suite.
  - This message includes two parameters, certificate type and certificate authorities.
    - certificate types: This field is a list of the types of certificates requested, sorted in order of the server's preference.
    - certificate authorities: This is a list of the distinguished names of acceptable certificate authorities. These distinguished names may specify a desired distinguished name for a root CA or for a subordinate CA; thus, this message can be used to describe both known roots and a desired authorization space.
- *Server hello done message:*
  - *This message is sent by the server to indicate the end* of the server hello and associated messages
  - After sending this message, the server will wait for a client response
  - This message means that server has finished sending messages to support the key exchange, and the client can proceed with its phase of the key exchange
  - Upon receipt of the server hello done message, the client should verify that the server provided a valid certificate if required and check that the server hello parameters are acceptable. If all is satisfactory, the client sends one or more messages back to the server

## SSL Handshake Protocol



## SSLHP –Phase 3

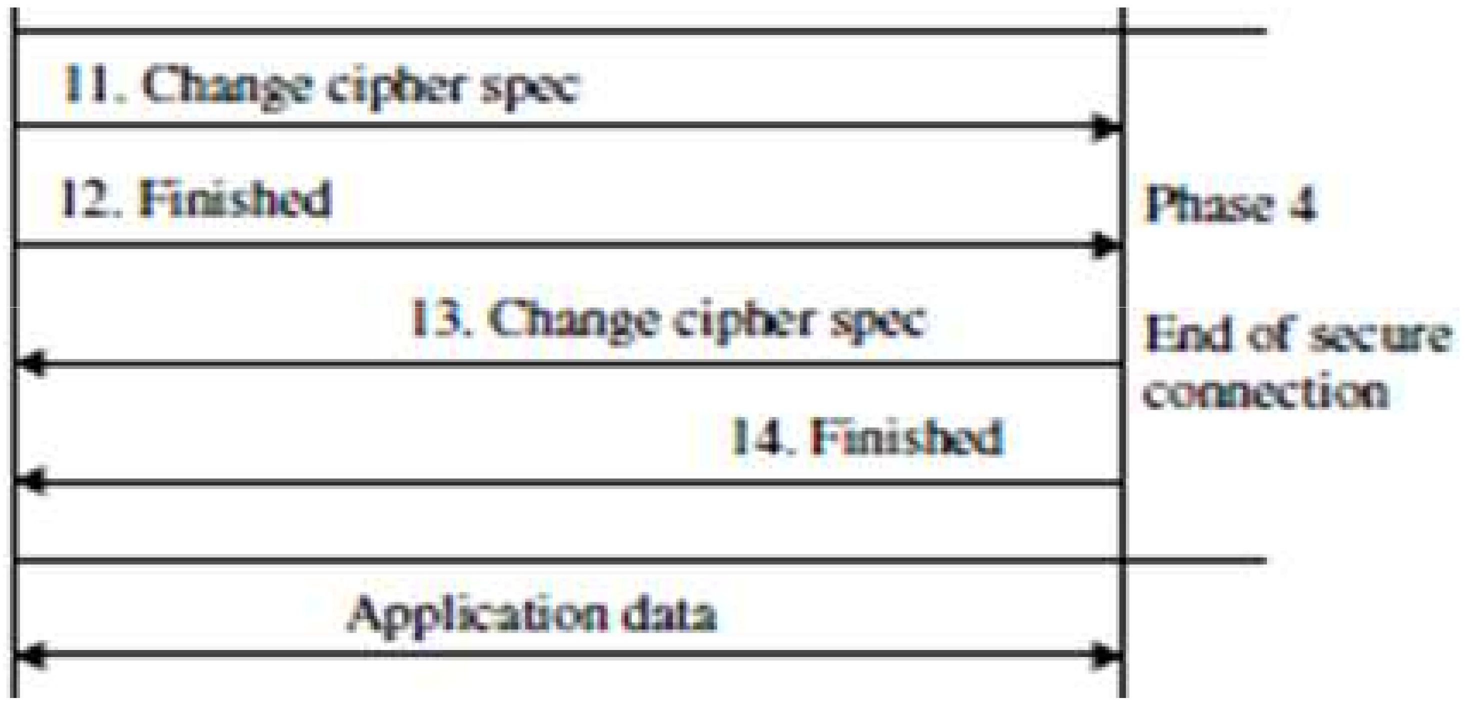
- *Phase 3: Client Authentication and Key Exchange*
  - If the server has sent a certificate request message
    - Client must send
      - Certificate message
      - Key exchange message
      - Certificate verify message:  if the client sent digitally signed certificate then a digitally signed certificate verify message is sent to explicitly verify the certificate

# SSLHP –Phase 3

- *Client certificate message:*
  - *First message the client can send after receiving* a server hello done message
  - Sent only when the server requests a certificate
  - if no suitable certificate is available then client sent certificate message containing no certificates
- *Client key exchange message:*
  - It will be the first message sent by the client after it receives the server hello done message (if client certificate is not requisted)
  - Premaster secret is set
  - RSA  or Diffie–Hellman
  - Certificate contained a Diffie–Hellman public key and parameters matched those specified by the server in its certificate; then this message contains no data
- Certificate verify message:
  - Follow the client key exchange message
  - Provide explicit verification of a client certificate
  - Used only when client certificate has signing capability
  - It is a Hash code based on the preceding messages
  - Structure is defined as follows:

```
struct{
      Signature signature;
} CertificateVerify;
CertificateVerify.signature.md5_hash
     MD5(master_secret||pad2||MD5(handshake-message||
          master_secret||pad1))
Certificate.signature.sha_hash
     SHA(master_secret||pad2||SHA(handshake-message||
          master_secret||pad1))
```

## SSL Handshake Protocol



# SSLHP –Phase 4

- *Phase 4: End of Secure Connection*
  - Client
    - Sent Change cipher spec message
    - Copies the pending CipherSpec into the current CipherSpec
    - Sent Finished message under the new algorithms, keys and secrets
  - Server in response
    - Send its own change cipher spec message
    - Transfer the pending CipherSpec to the current one
    - Send its finished message under the new CipherSpec
  - At this point, the handshake is complete and the client and server may begin to exchange application layer data

## SSLHP –Phase 4

- *Change cipher spec messages:*
  - Sent after the certificate verify message
  - *Client Sends a change cipher spec message and* copies the pending CipherSpec in the current CipherSpec
- *Finished message:*
  - *This is always sent immediately after a change cipher spec message*
  - Verify that the key exchange and authentication processes were successful
  - The content of the finished message is the concatenation of two hash values:

    MD5(master_secret||pad2||MD5(handshake_messages||Sender||
    master_secret||pad1))

    SHA(master_secret||pad2||SHA(handshake_messages||Sender||
    master_secret||pad1))

    - 'Sender' is a code that identifies that the sender
    - 'handshake messages' is code that identifies the data from all handshake messages up to but not including this message

## SSL Application Data

- Once hand shake is completed begin sending and receiving application data over the connections
- Application data treated as *transparent data*
- *It is carried by the Record Layer and is* fragmented, compressed and encrypted based on the current connection state