Secure Socket Layer

Agenda

- SSL Basics
- WTLS
- Security for Web Service

SSL Facts

- SSL was first developed by Netscape in 1994 and became an internet standard in 1996 (RFC 2246 – TLS V1.0)
- SSL is a cryptographic protocol to secure network across a *connection-oriented* layer
- Any program using TCP can be modified to use SSL connection

SSL Facts

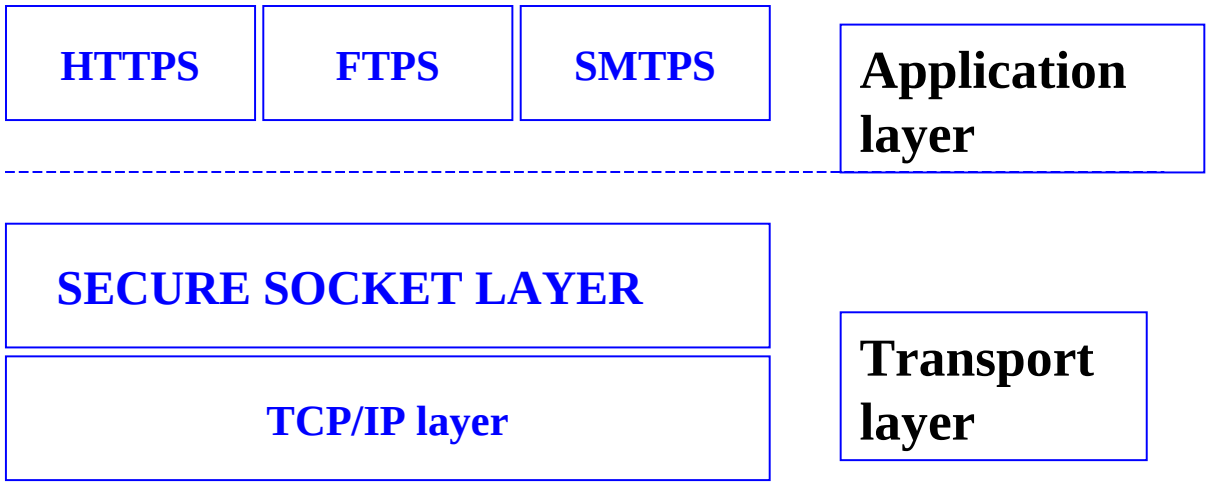
- SSL connection uses a dedicated TCP/IP socket(e.g. port 443 for https)
- SSL is flexible in choice of which symmetric encryption, message digest, and authentication can be used
- SSL provides built in data compression

SSL Usage

- Authenticate the server to the client
- Allow the client and server to select cryptographic algorithms, or ciphers, that they both support
- Optionally authenticate the client to the server
- Use public key encryption techniques to generate shared secret
- Establish an encrypted SSL connection

Secure Socket Layer

ssl is a secure protocol which runs above tcp/ip and allows users to encrypt data and authenticate servers/vendors identity securely.



SSL Stack

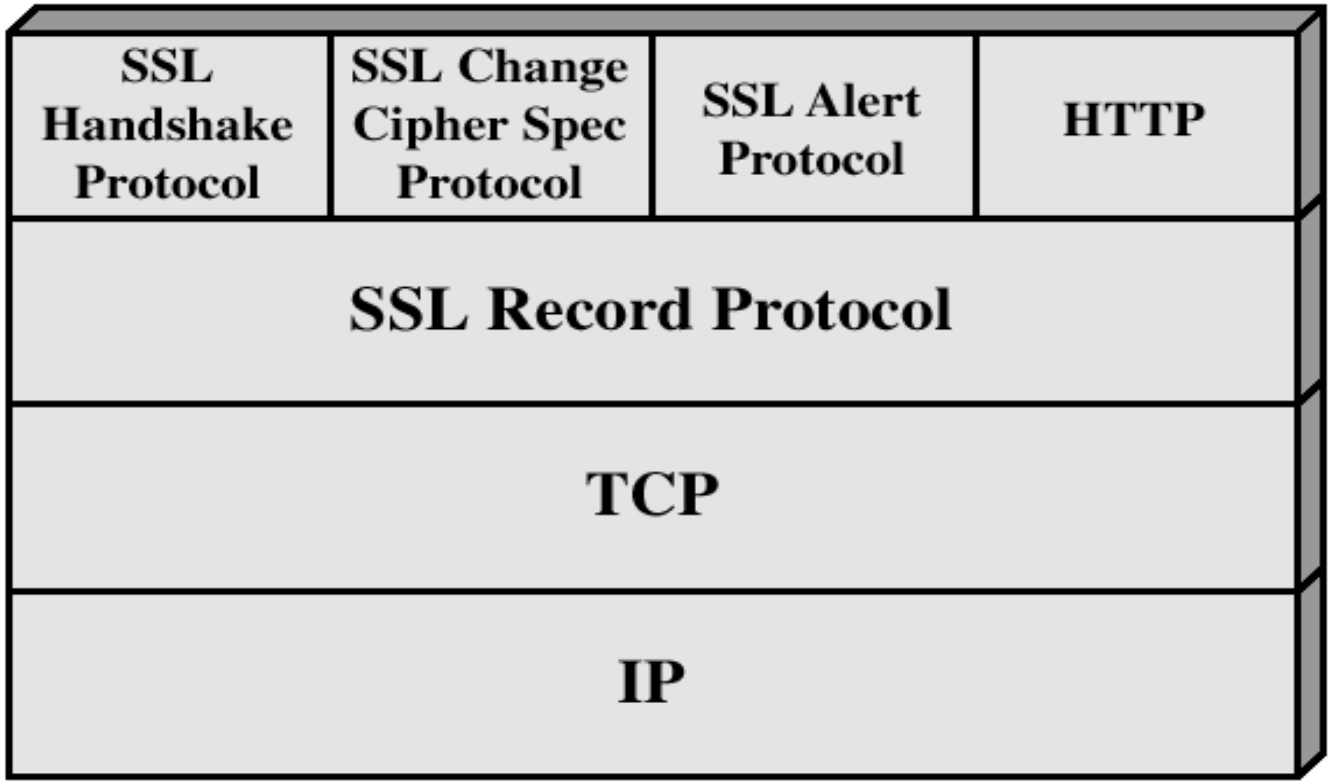
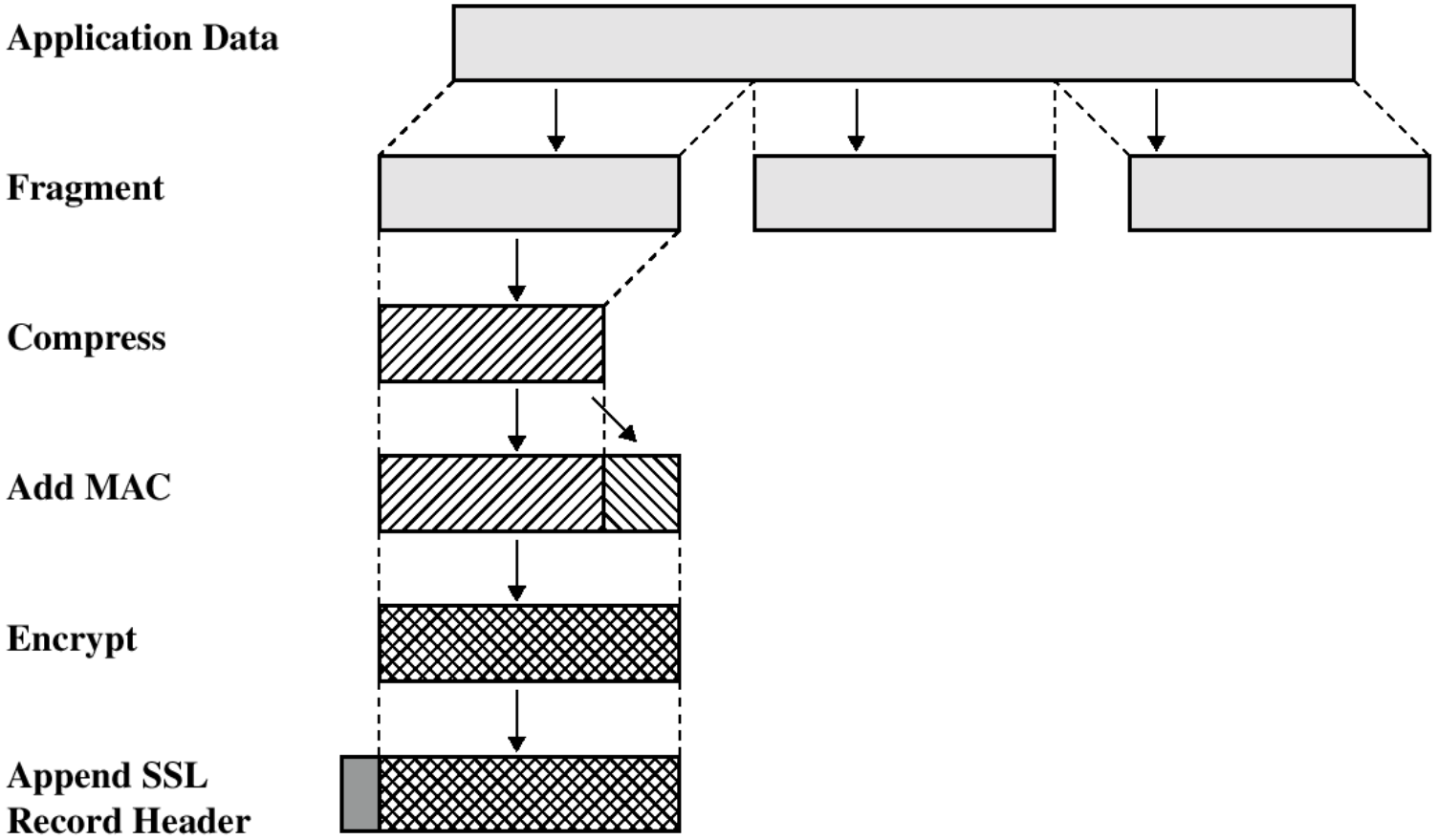
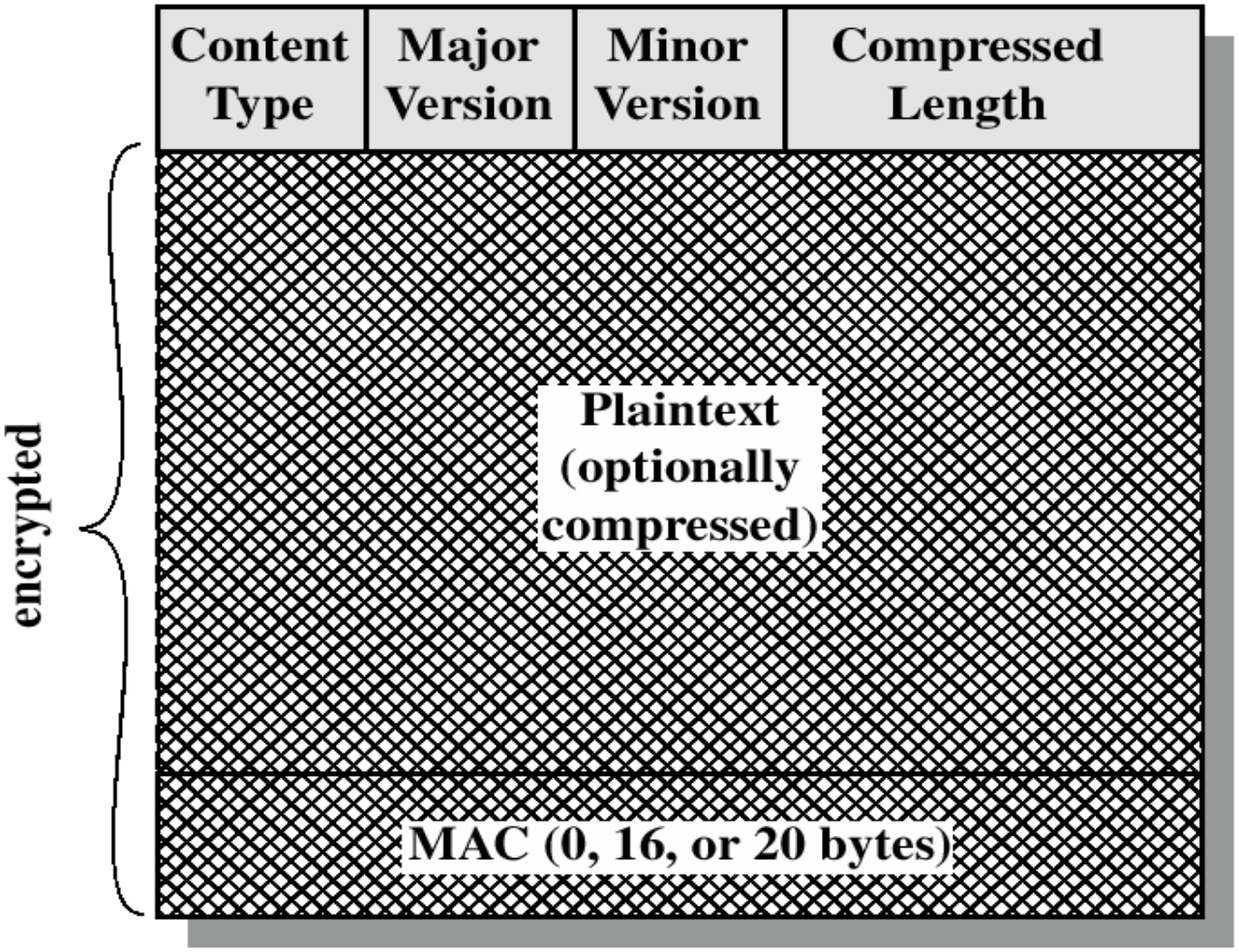


Figure 7.2 SSL Protocol Stack

SSL Record Protocol Operation



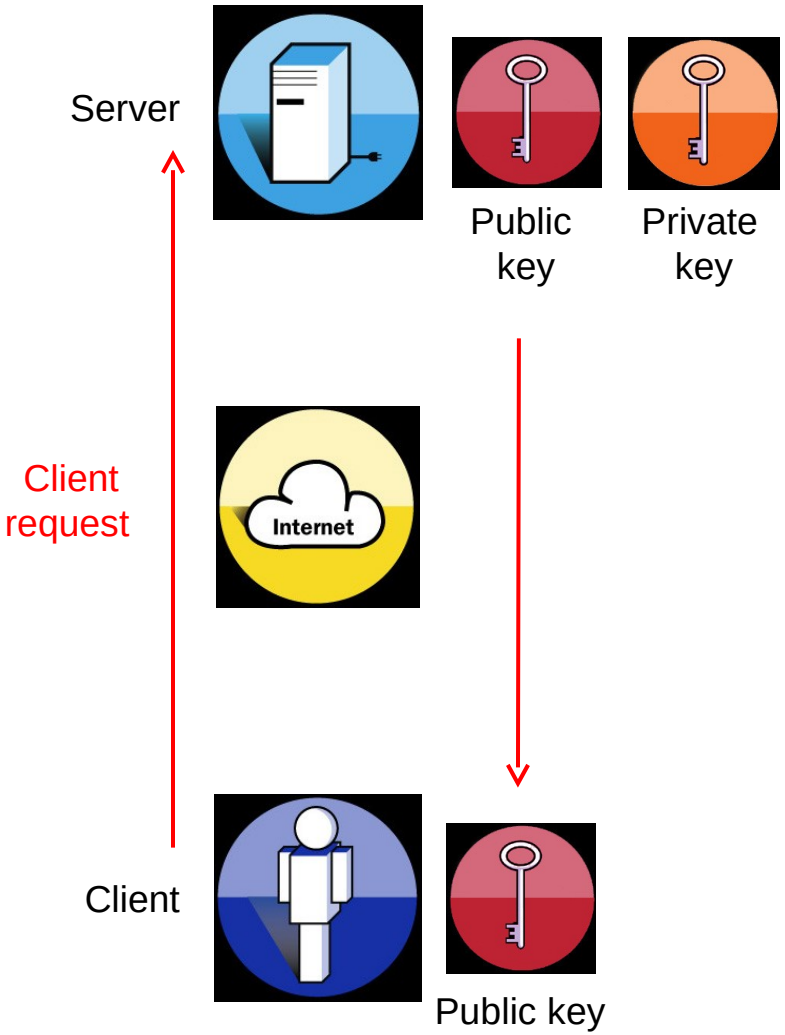
SSL Record Format



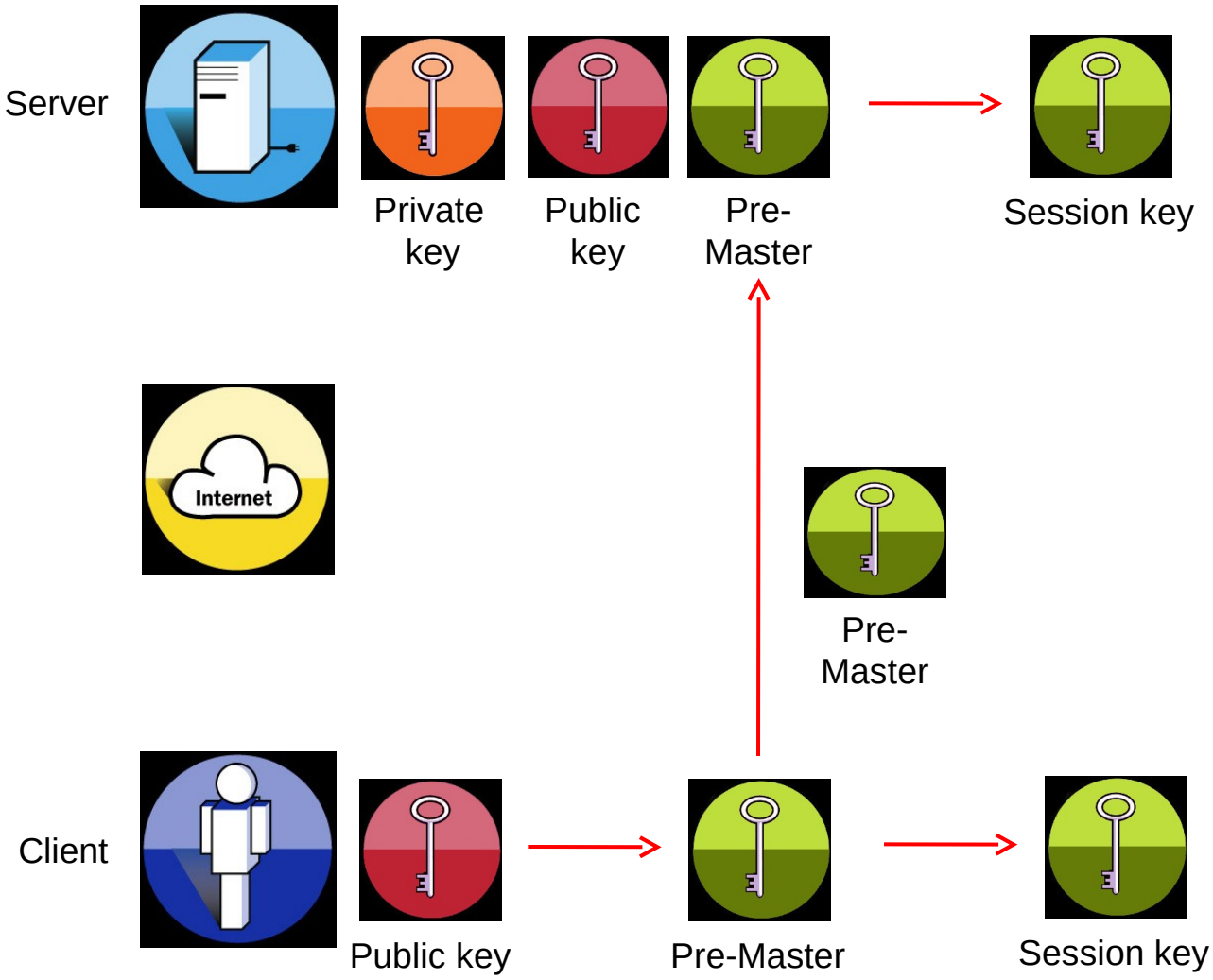
SSL Handshake

SSL handshake verifies the server and allows client and server to agree on an encryption set **before** any data is sent out

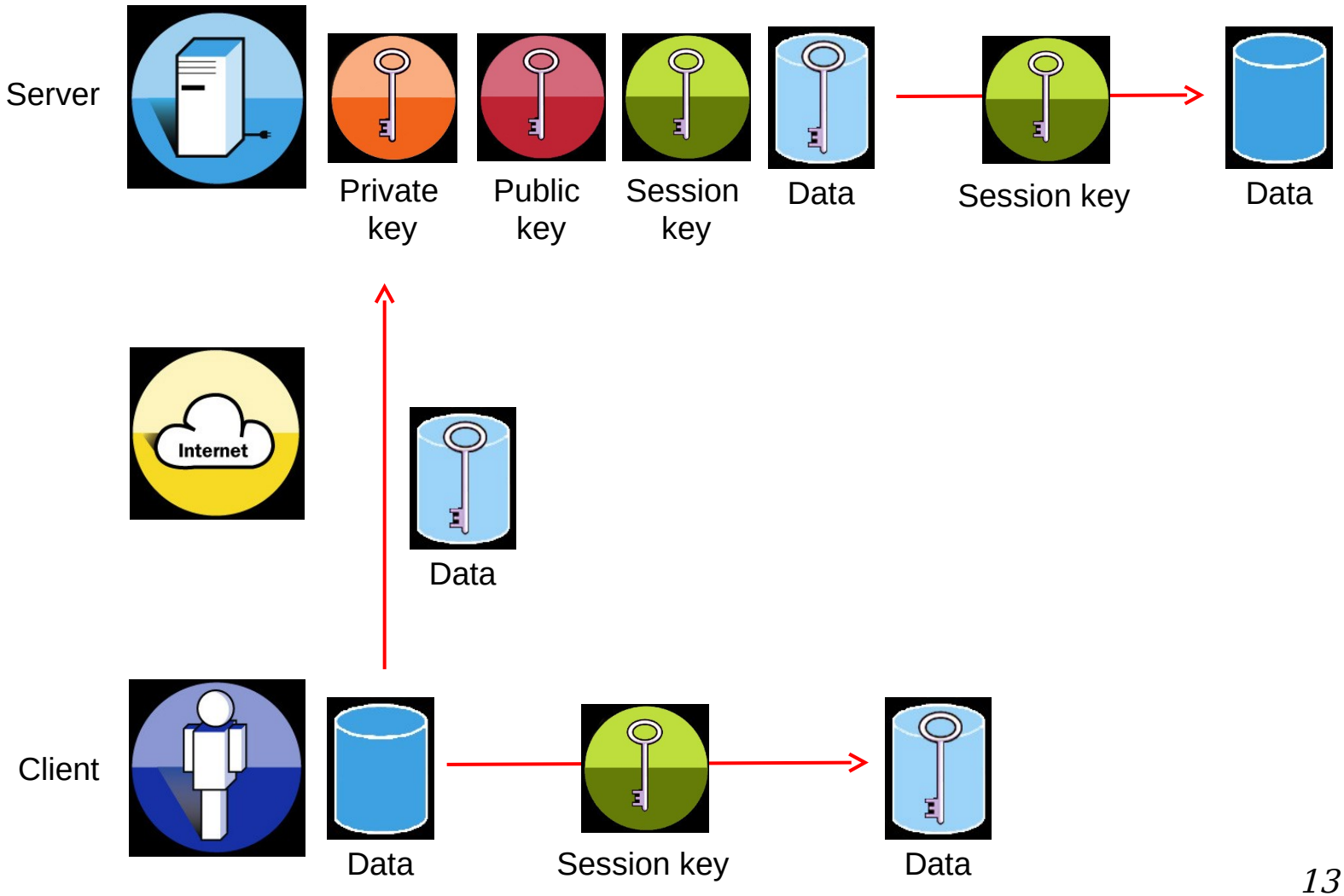
SSL Handshake



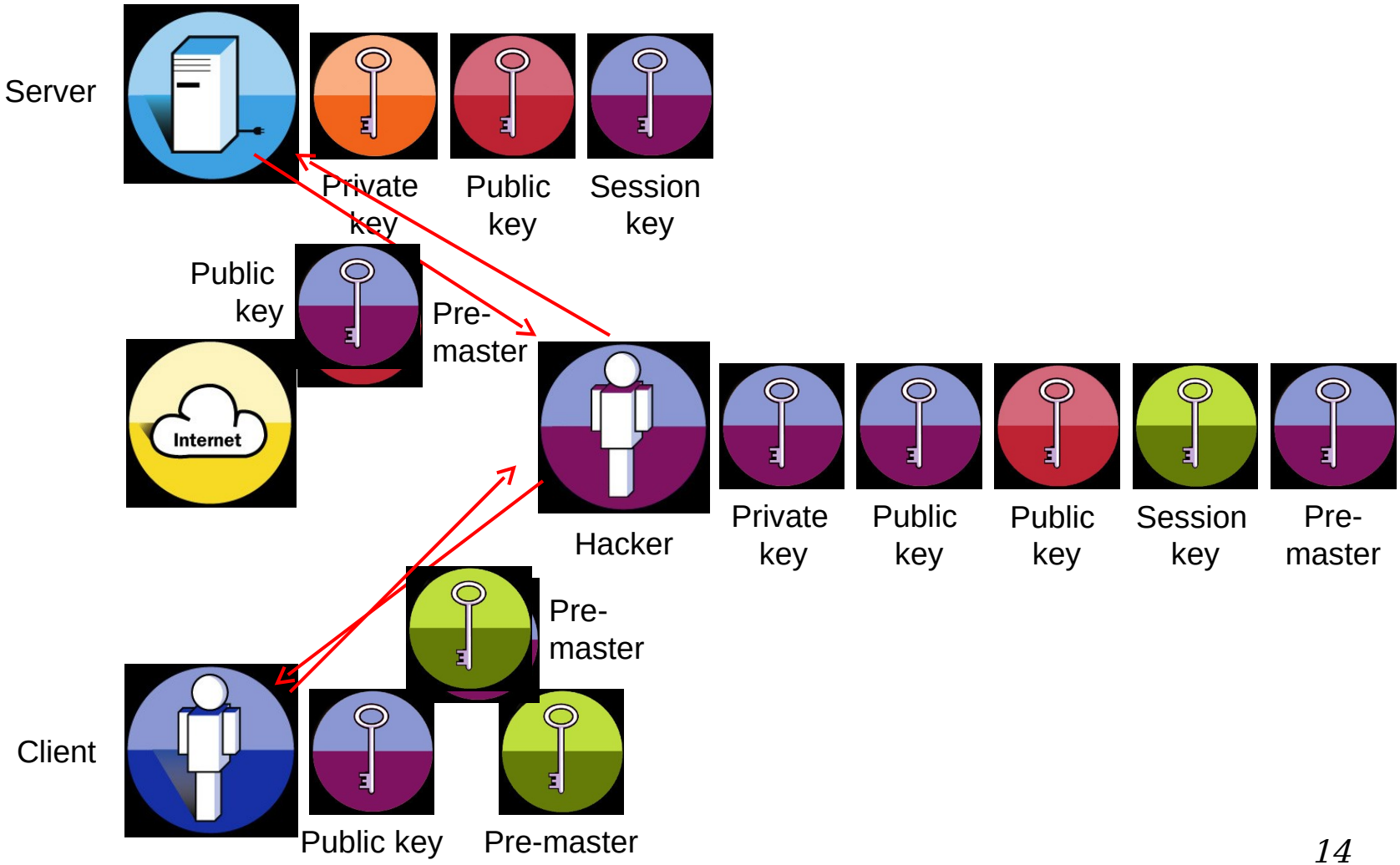
SSL Session Key



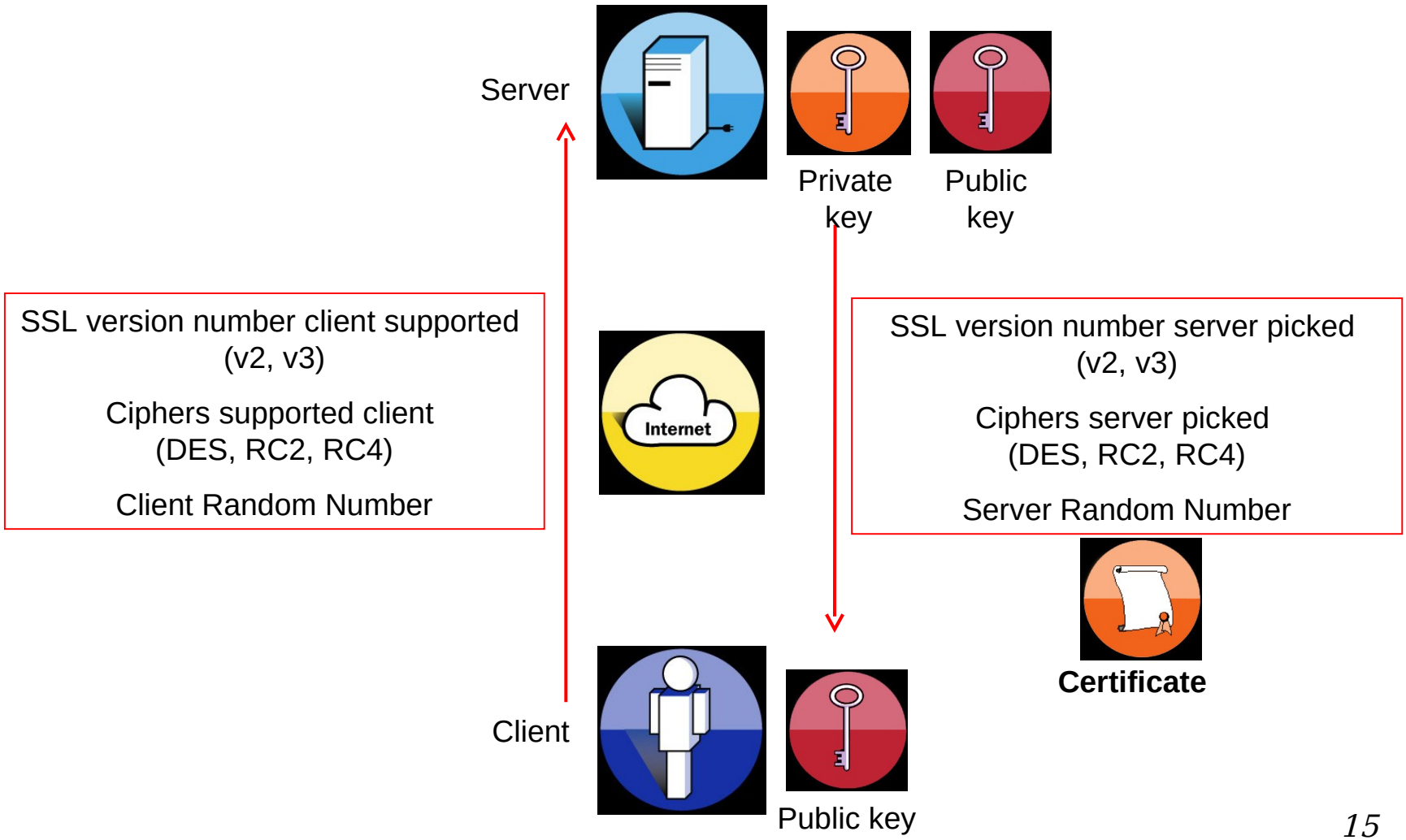
Secure Data on Network



Man-in-the-Middle Attack



Key exchange and certificate



Verify Certificate

Server

Private key

Public key

Certificate is Good and Valid

Server/vendor has been verified and authenticated

Client request

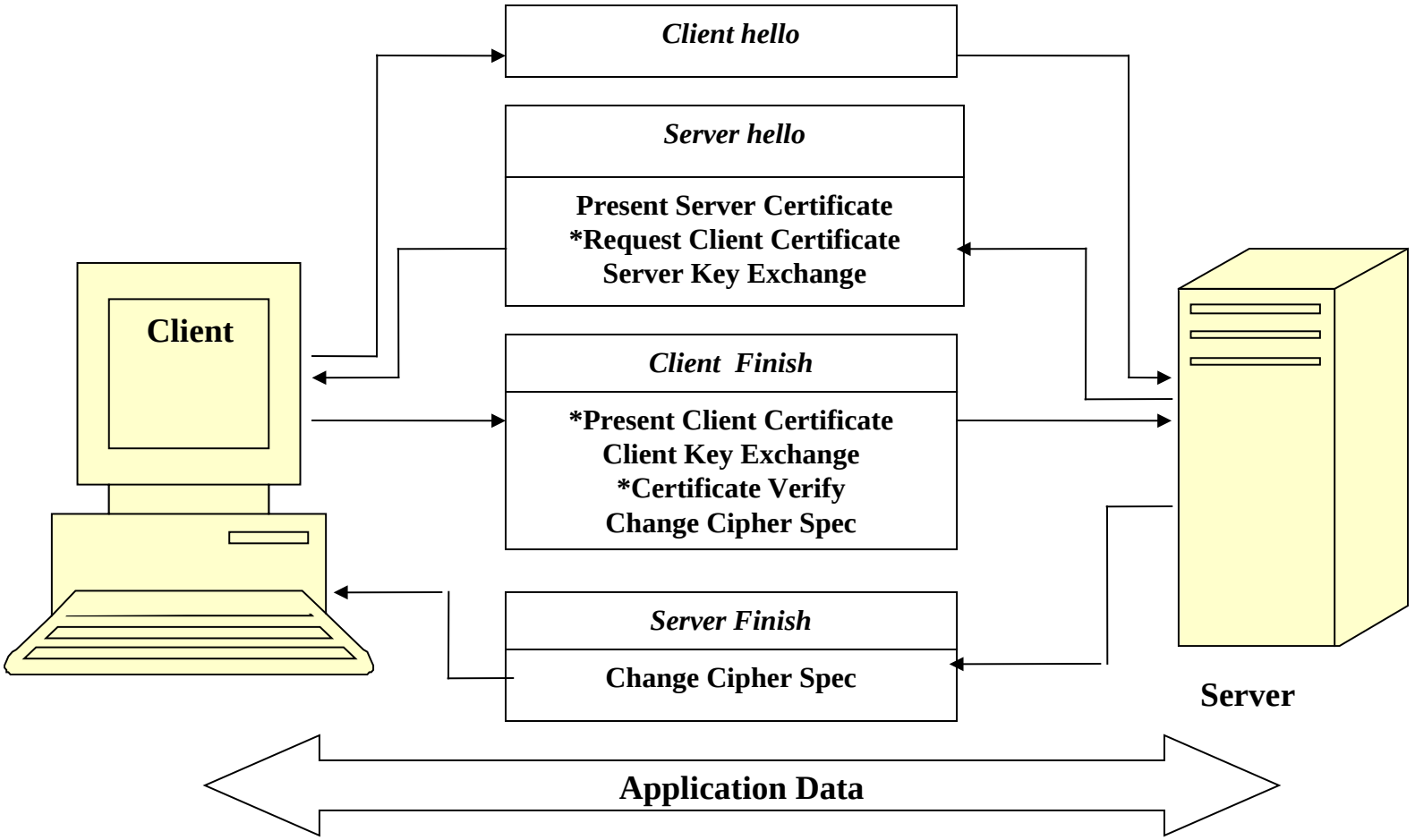
Certificate

Client has vendor's public key and can now encrypt pre-master to send to server/vendor

Client

Public key

SSL Handshake



Server Hello Request

- Notifies the client that they should send a client hello message to begin the negotiation process
- Sent by the server at any time
- After the server sends a request, it does not send another one until a handshake has been completed
- Client can choose to ignore them or send a Client Hello

Client Hello

- *Sent by the client*
  - *When first connecting to a server*
  - *In response to a hello request or on its own*

- *Contains*
  - *32 bytes random number created by a secure random number generator*
  - *Protocol version*
  - *Session ID*
  - *A list of supported ciphers*
  - *A list of compression methods*

Server Hello

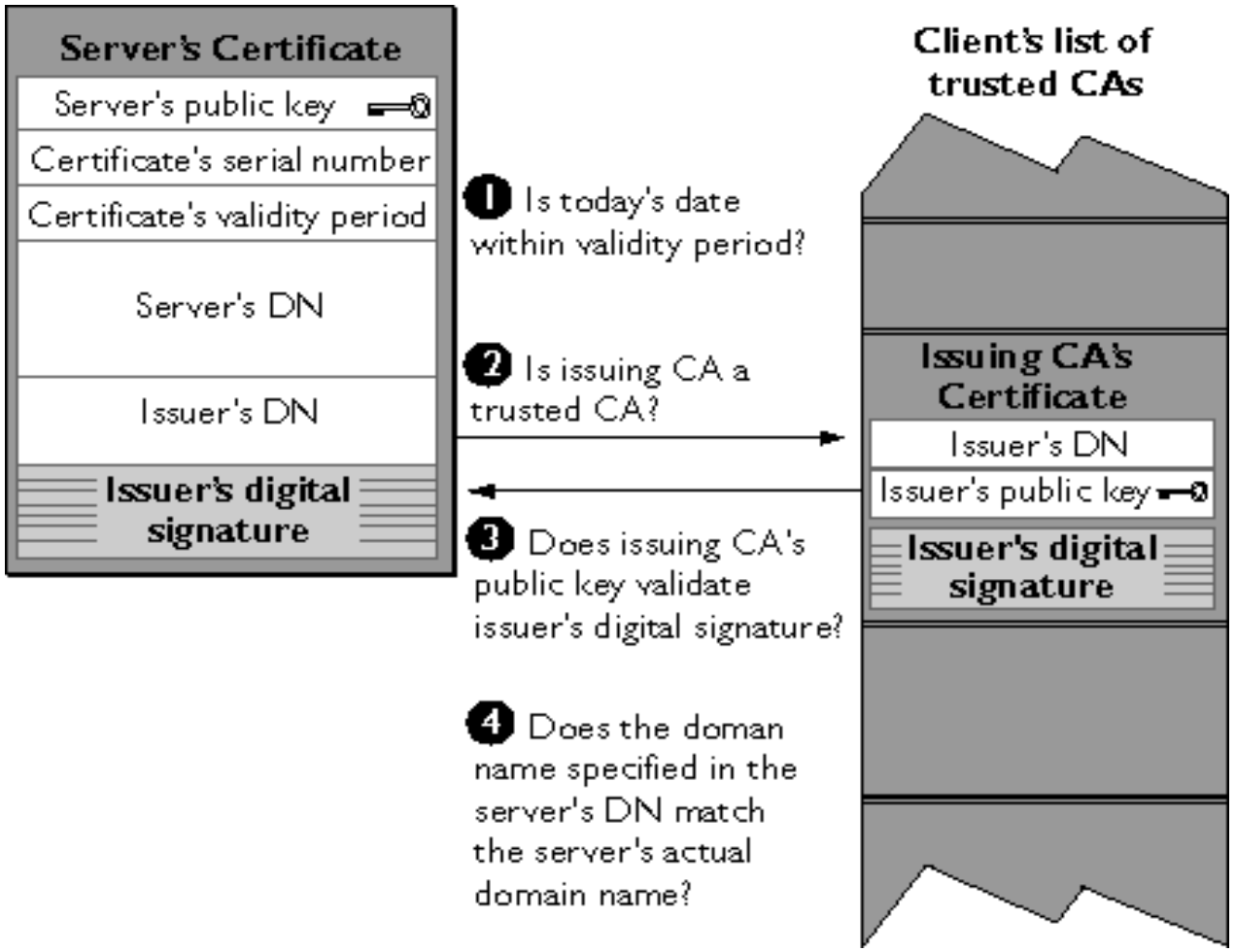
- Sent as response if client hello is accepted
  - *If not, a handshake failure alert is sent*

- Contains
  - *32 bytes random number created by a secure random number generator*
  - *Protocol version*
  - *Session ID*
  - *Cipher suite chosen*
  - *Compression method selected*

Server Certificates

- **Immediately following the server hello, the server sends its certificate**
  - *Generally an X.509.v3 certificate*
- **Server sends server hello done message**

Verify Server Certificate

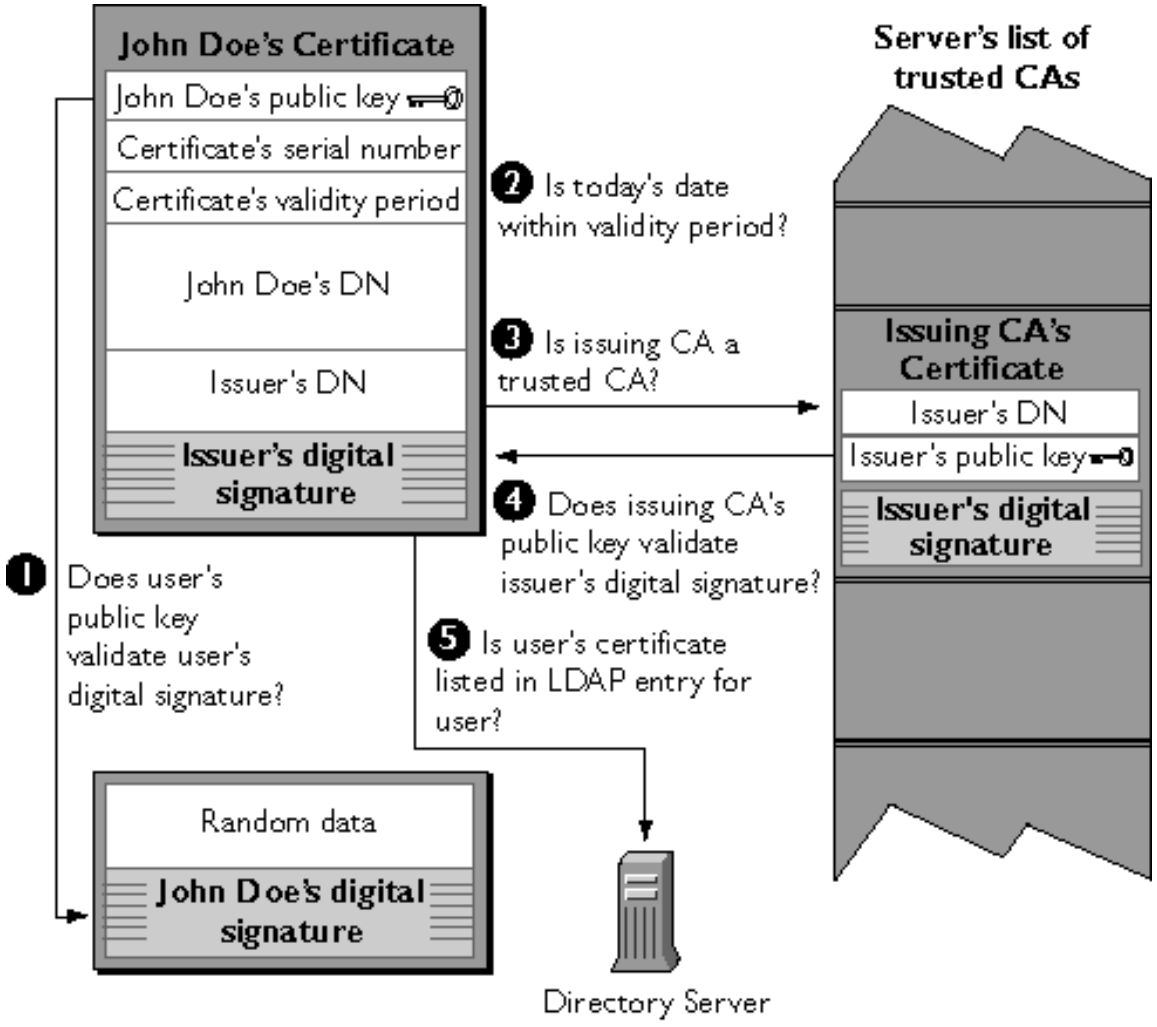


Client Certificate (*optional)*

Client only sends a certificate upon the receipt of a certificate request

- *Sends after receiving server hello done*
- *If the client does not have a suitable certificate, it sends a no certificate alert*
  - *Server will respond with a fatal handshake failure if a client certificate is necessary*

Verify Client Certificate



Key Exchange

- **Client sends 48-bytes pre-master, encrypted using server's public key, to the server**
- **Both server and client use the pre-master to generate the master secret**
- **A same session key is generated on both client and server side using the master secret**

Final Steps

- Client sends change_cipher_spec
- Client sends finished message
- Server sends change_cipher_spec
- Server sends finished message

SSL Architecture

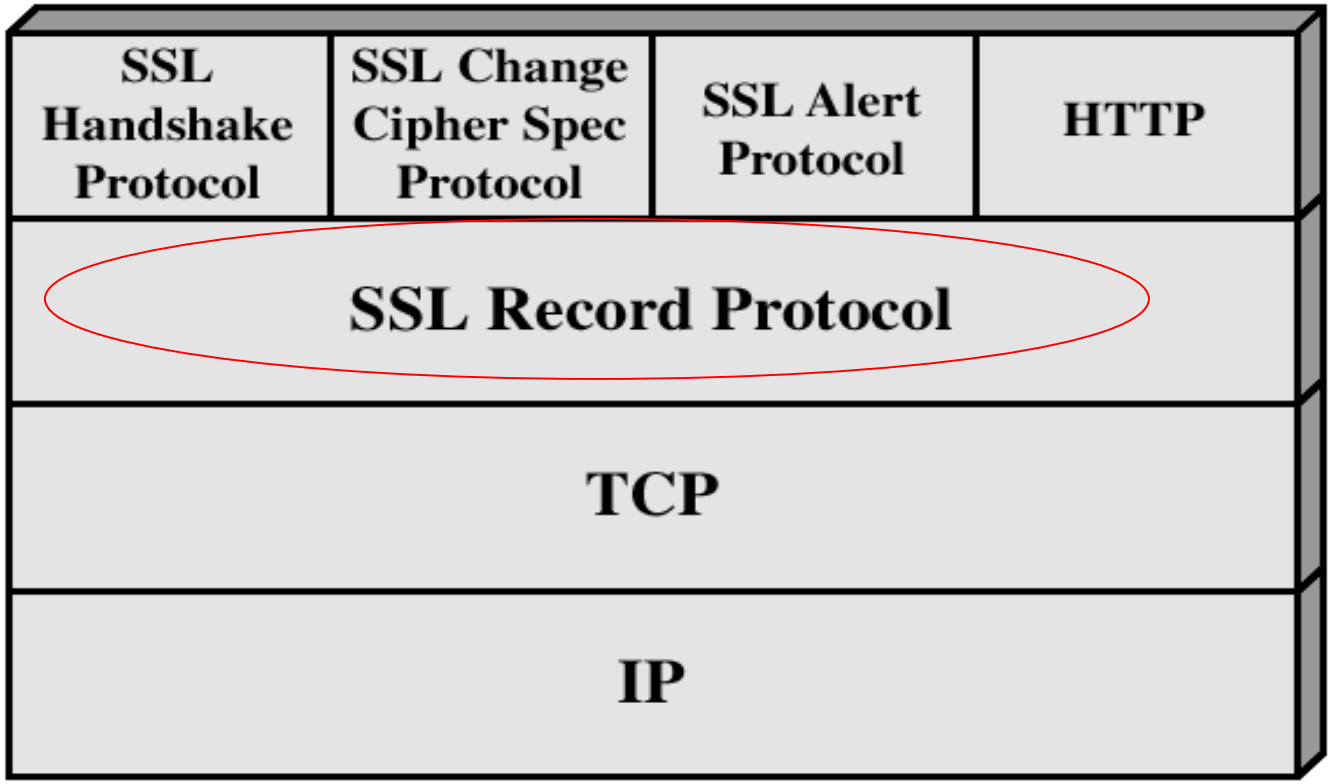


Figure 7.2 SSL Protocol Stack

Record Layer

- Compression and decompression
- A MAC is applied to each record using the MAC algorithm defined in the current cipher spec
- Encryption occurs after compression
- May need fragmentation

SSL Architecture

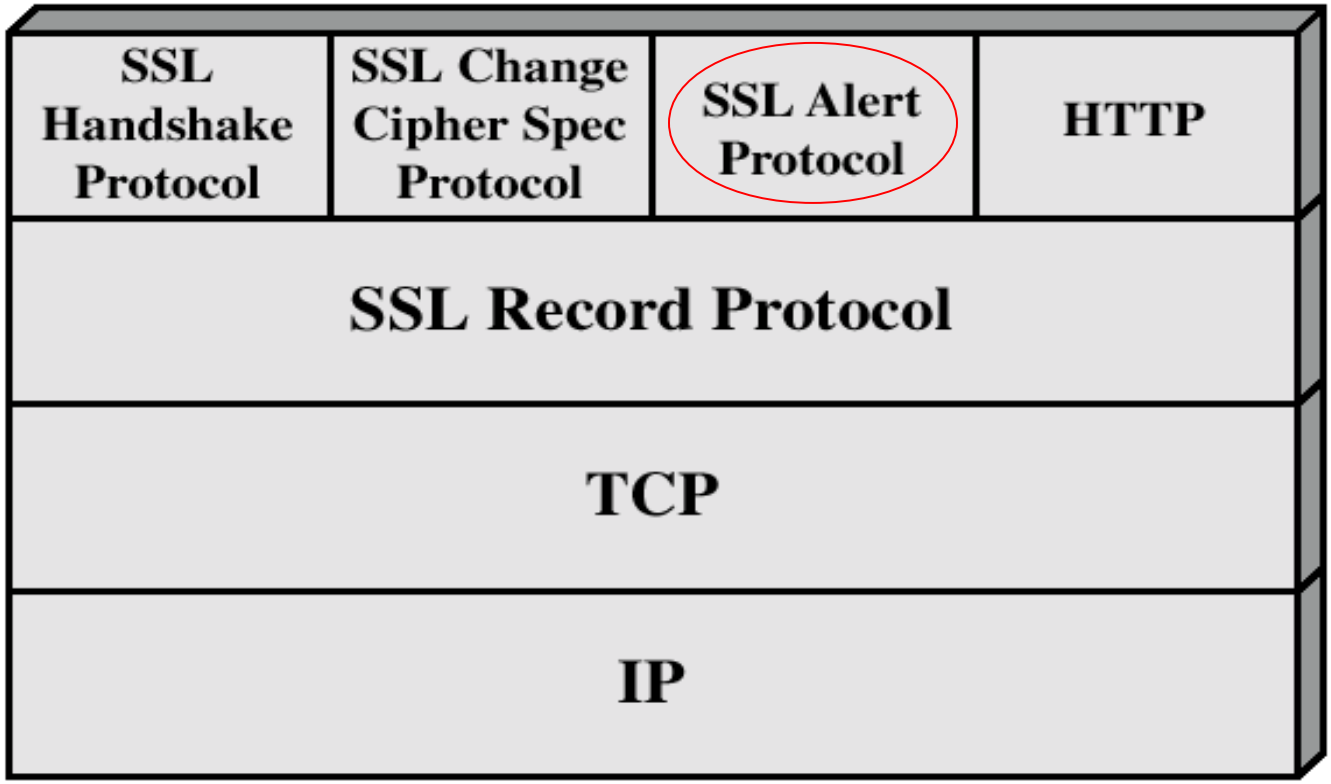


Figure 7.2 SSL Protocol Stack

Alert Layer

- **Explain severity of the message and a description**
  - ***fatal***
    - ***Immediate termination***
    - ***Other connections in session may continue***
    - ***Session ID invalidated to prevent failed session to open new sessions***
- **Alerts are compressed same as other data**

SSL Architecture

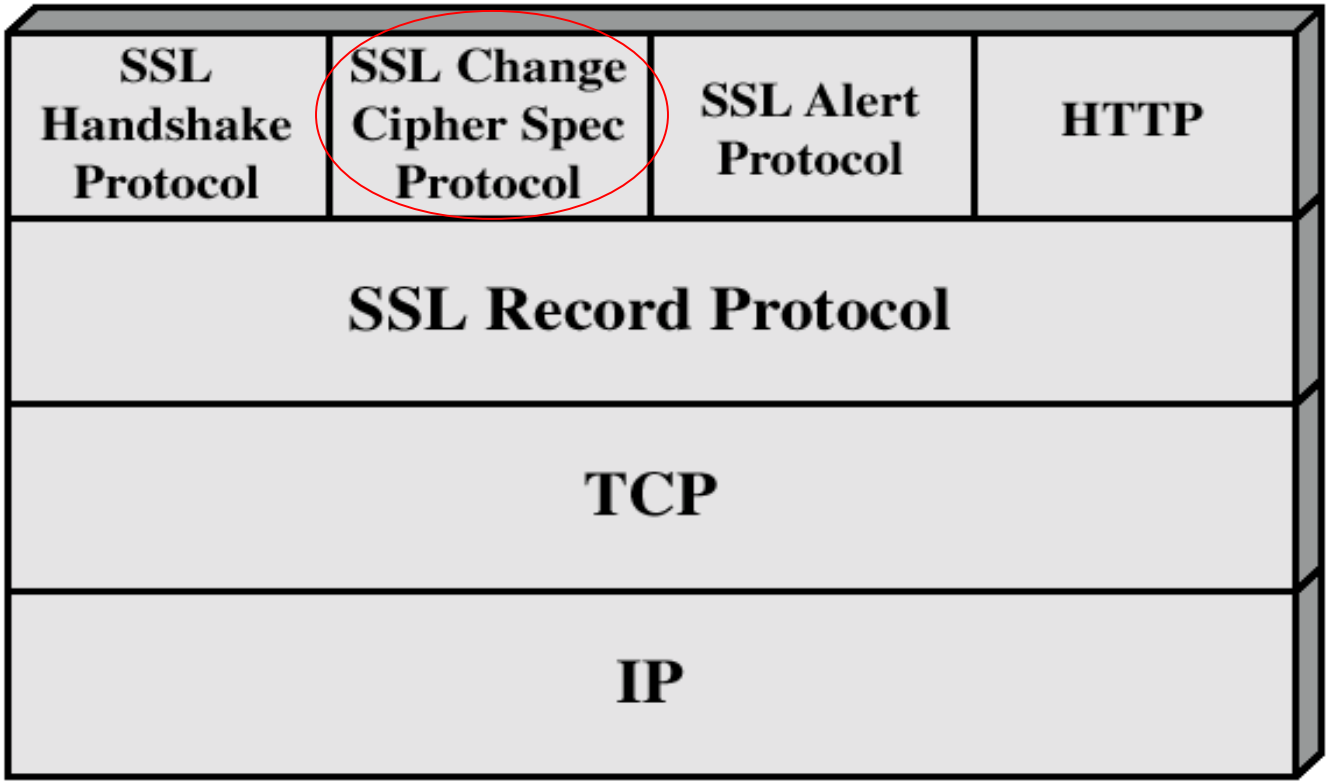


Figure 7.2 SSL Protocol Stack

Change Cipher Spec Protocol

- **Notify the other party to use the new cipher suite**
- **Before the Finished message**

Comparison of SSL V2.0 and V3.0

- SSL 2.0 is vulnerable to “man-in-the-middle” attack. The hello message can be modified to use 40 bits encryption. SSL 3.0 defends against this attack by having the last handshake message include a hash of all the previous handshake message

Comparison of SSL V2.0 and V3.0

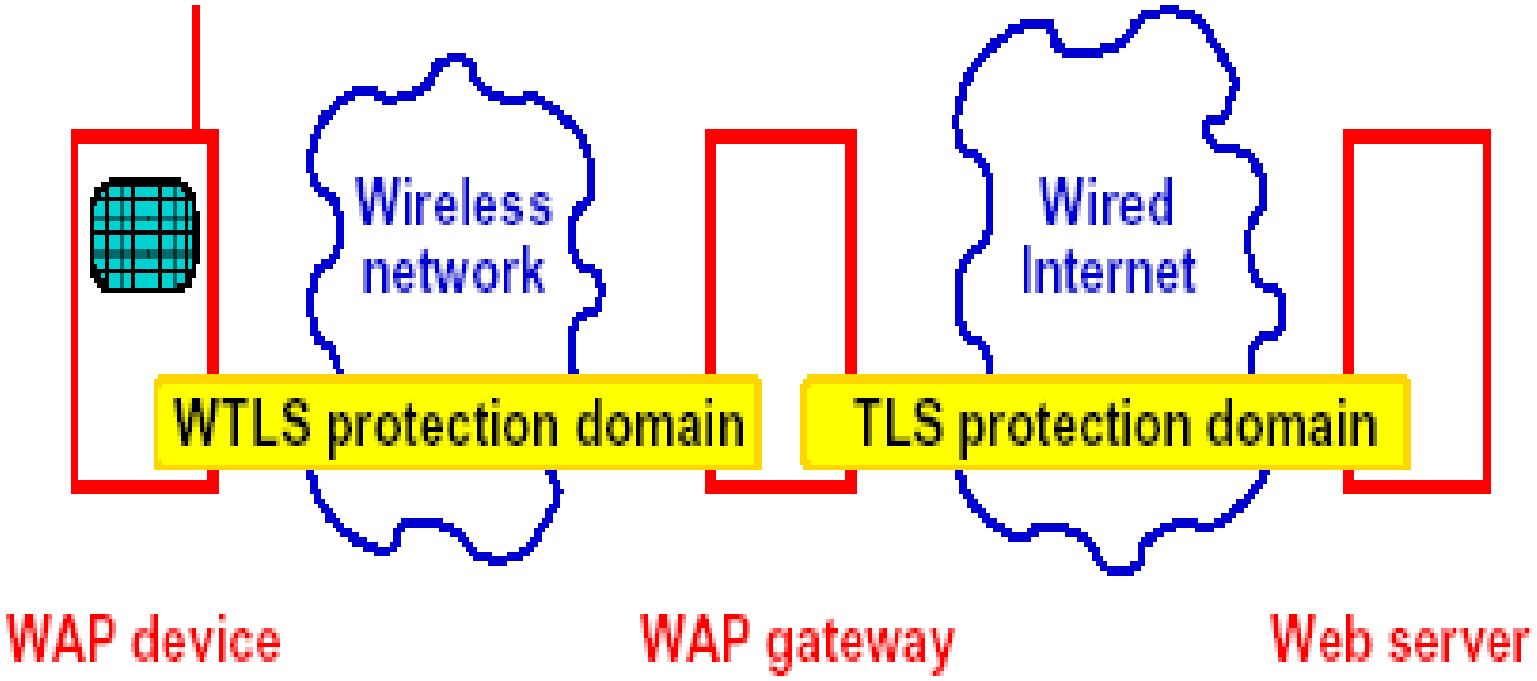
- SSL 2.0 uses a weak MAC construction
- In SSL 3.0, the Message Authentication Hash uses a full 128 bits of key material for Export cipher, while SSL 2.0 uses only 40 bits

Comparison of SSL V2.0 and V3.0

- SSL 2.0 only allows a handshake at the beginning of the connection. In 3.0, the client can initiate a handshake routine any time
- SSL 3.0 allows server and client to send chains of certificate
- SSL 3.0 has a generalized key exchange protocol. It allows Diffie-Hellman and Fortezza key exchange
- SSL 3.0 allows for record compression and decompression

Wireless Transport Layer Security

WTLS Overview



WTLS Facts

- Mainly used to secure data transport between wireless device and gateway
- Built on top of datagram (UDP) instead of TCP
- WTLS provides full, optimized and abbreviated handshake to reduce roundtrips in high-latency networks

WTLS Facts

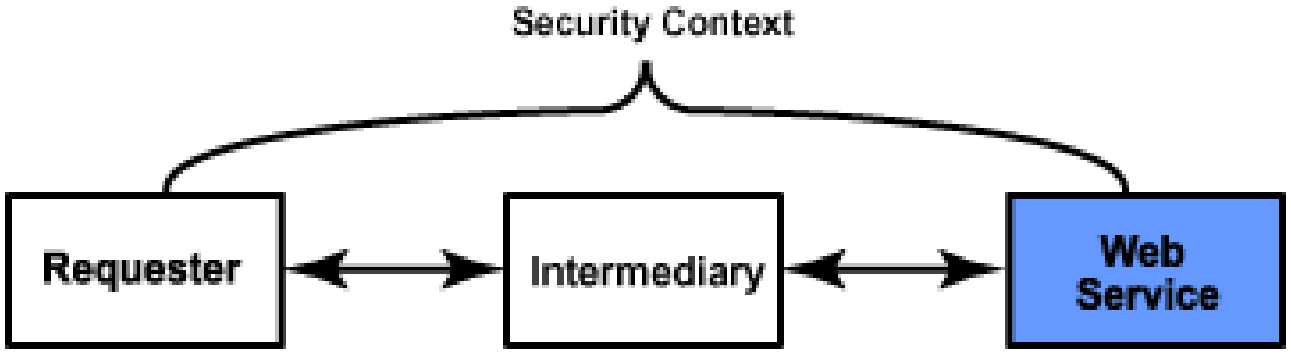
- WTLS uses different format of certificates, mainly WTLS certificate, X509v1 and 968. It also supports additional cipher suites, such as RC5, short hashes, ECC, etc;
- WTLS provides built-in key-refresh mechanism for renegotiation;
- WTLS can also set session resumable to continue on a previous session.

Web Service Security

Comparison of Traditional Web Application and Web Service

- Client-server system vs multi-party
- Simple protocol sets vs complicated protocol sets

Point-to-Point



End-to-End

Proposed Security Specification

Initial Specifications

- WS-Security
- WS-Policy
- WS-Trust
- WS-Privacy

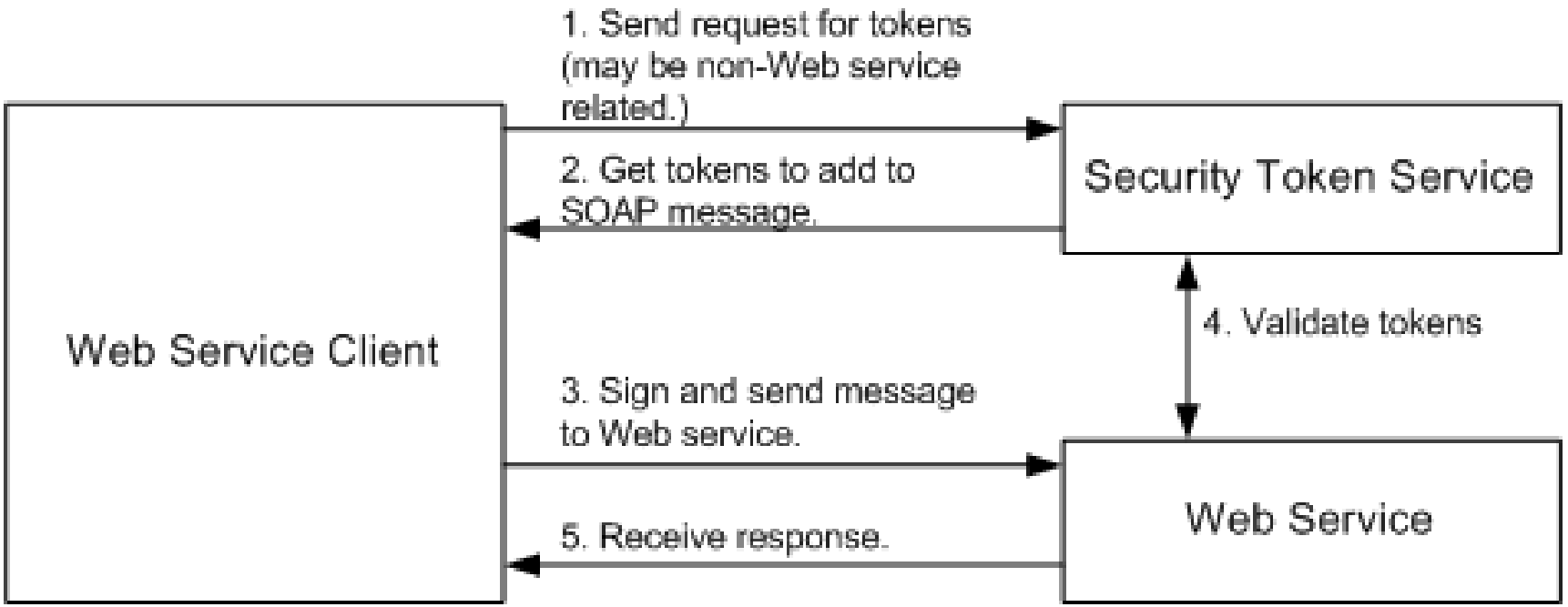
Follow-on Specifications

- WS-SecureConversation
- WS-Federation
- WS-Authorization

WS-Security

- A“ what” not “how”
- Security token is embedded inside SOAP headers
- Message integrity is provided by XML Signature and security tokens
- Message confidentiality is provided by XML Encryption with security tokens

WS-Security


1. Send request for tokens (may be non-Web service related.)
2. Get tokens to add to SOAP message.
Security Token Service
Web Service Client
4. Validate tokens
3. Sign and send message to Web service.
Web Service
5. Receive response.

Web Service Security