INFS 766
Internet Security Protocols

Lecture 5
SSL

Prof. Ravi Sandhu

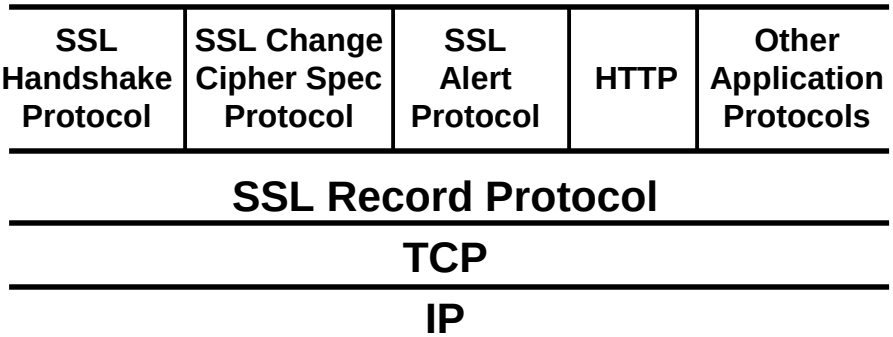
SECURE SOCKETS LAYER (SSL)

- **layered on top of TCP**
- **SSL versions 1.0, 2.0, 3.0, 3.1**
- **Netscape protocol**
- **later refitted as IETF standard TLS (Transport Layer Security)**
- **TLS 1.0 very close to SSL 3.1**

SECURE SOCKETS LAYER (SSL)

- **application protocol independent**
- **does not specify how application protocols add security with SSL**
  - **how to initiate SSL handshaking**
  - **how to interpret certificates**
- **left to designers of upper layer protocols to figure out**

SSL ARCHITECTURE



SSL ARCHITECTURE

- **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**

SSL/TLS DIFFERENCES

- **TLS uses HMAC, SSL uses a precursor**
- **TLS MAC covers compression version field in addition to what SSL MAC covers**
- **TLS defines additional alert codes**
- **other minor differences**
- **TLS has a mode to fall back to SSL**

SSL SERVICES

- peer entity authentication
- data confidentiality
- data authentication and integrity
- compression/decompression
- generation/distribution of session keys
  - integrated into protocol
- security parameter negotiation

SSL SESSIONS AND CONNECTIONS

- Every connection is associated with one session
- Session can be reused across multiple secure connections
- Handshake protocol
  - establishes new session and connection together
  - uses existing session for new connection

SSL SESSION

- **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**
- **client and server 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**
- **compression state**
- **cipher state: initially IV, subsequently next feedback block**
- **sequence number: starts at 0, max 2^{64}-1**

SSL CONNECTION STATE

- **4 parts to state**
  - **current read state**
  - **current write state**
  - **pending read state**
  - **pending write state**
- **handshake protocol**
  - **initially current state is empty**
  - **either pending state can be made current and reinitialized to empty**

SSL RECORD PROTOCOL

- **4 steps by sender (reversed by receiver)**
  - **Fragmentation**
  - **Compression**
  - **MAC**
  - **Encryption**

SSL RECORD PROTOCOL

- **each SSL record contains**
  - **content type: 8 bits, only 4 defined**
    - **change_cipher_spec**
    - **alert**
    - **handshake**
    - **application_data**
  - **protocol version number: 8 bits major, 8 bits minor**
  - **length: max 16K bytes (actually 2^{14}+2048)**
  - **data payload: optionally compressed and encrypted**
  - **message authentication code (MAC)**

SSL HANDSHAKE PROTOCOL

- **initially SSL session has null compression and cipher algorithms**
- **both are set by the handshake protocol at beginning of session**
- **handshake protocol may be repeated during the session**

SSL HANDSHAKE PROTOCOL

- **Type: 1 byte**
  - **10 message types defined**
- **length: 3 bytes**
- **content**

SSL HANDSHAKE PROTOCOL

```
Client                                               Server

ClientHello                  -------->
                                                ServerHello
                                               Certificate*
                                         ServerKeyExchange*
                                        CertificateRequest*
                             <--------      ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                          [ChangeCipherSpec]
                             <--------             Finished
Application Data             <------->     Application Data
```

Fig. 1 - Message flow for a full handshake

* Indicates optional or situation-dependent messages that are not always sent.

SSL HANDSHAKE PROTOCOL

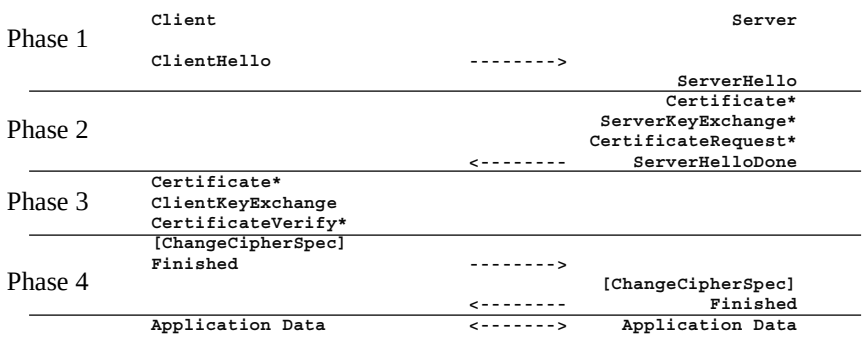


Fig. 1 - Message flow for a full handshake

* Indicates optional or situation-dependent messages that are not always sent.

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 1-WAY HANDSHAKE WITH RSA

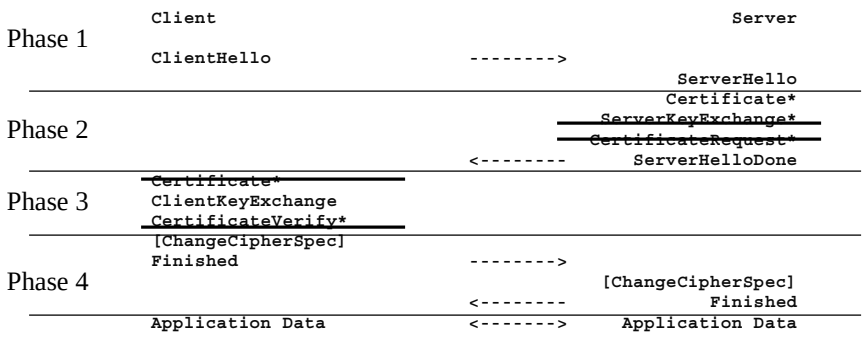


Fig. 1 - Message flow for a full handshake

* Indicates optional or situation-dependent messages that are not always sent.

SSL 2-WAY HANDSHAKE WITH RSA

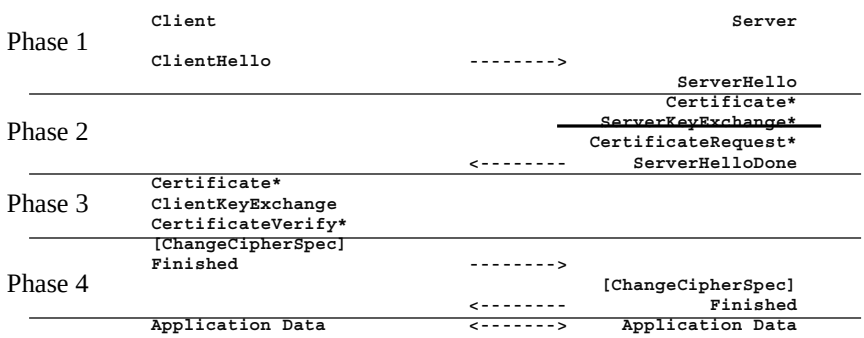


Fig. 1 - Message flow for a full handshake

* Indicates optional or situation-dependent messages that are not always sent.

SSL HANDSHAKE PROTOCOL

- **these 9 handshake messages must occur in order shown**
- **optional messages can be eliminated**
- **10th message explained later**
  - **hello_request message**
- **change_cipher_spec is a separate 1 message protocol**
  - **functionally it is just like a message in the handshake protocol**

SSL HANDSHAKE PROTOCOL

```
Client                                                Server

ClientHello                    -------->
                                                 ServerHello
                                               [ChangeCipherSpec]
                               <--------             Finished
[ChangeCipherSpec]
Finished                       -------->
Application Data               <------->     Application Data
```

Fig. 2 - Message flow for an abbreviated handshake

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**
  - **don't want to renegotiate a session**
    - **client may respond with a no_renegotiation alert**

SSL HANDSHAKE PROTOCOL

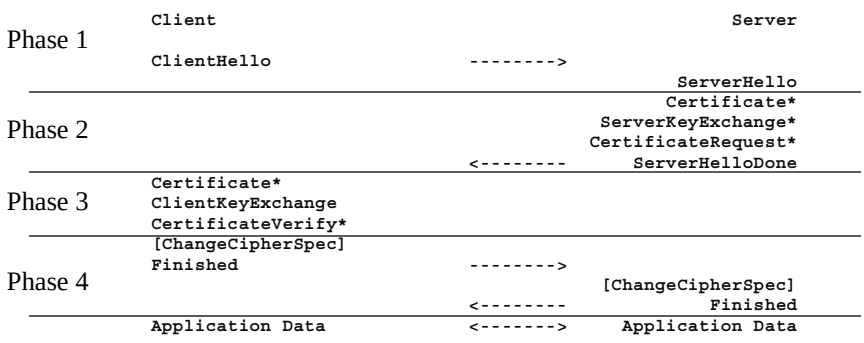


Fig. 1 - Message flow for a full handshake

* Indicates optional or situation-dependent messages that are not always sent.

SSL HANDSHAKE: PHASE 1
ESTABLISH SECURITY CAPABILITIES

- **client hello**
  - **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**

SSL HANDSHAKE: PHASE 1
ESTABLISH SECURITY CAPABILITIES

- **server hello**
  - **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: 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**
      we will ignore Fortezza from here on

SSL HANDSHAKE: 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 or 20 bytes**
    - **Key Material: used to generate write keys**
    - **IV Size: size of IV for CBC**

SSL HANDSHAKE PROTOCOL

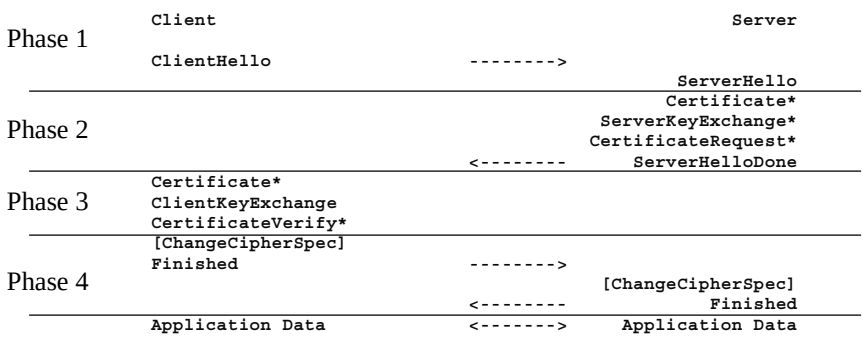


Fig. 1 - Message flow for a full handshake

* Indicates optional or situation-dependent messages that are not always sent.

SSL HANDSHAKE: PHASE 2

SERVER AUTHENTICATION & KEY EXCHANGE

- **Certificate message**
  - **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**

SSL HANDSHAKE: 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**

SSL HANDSHAKE PROTOCOL

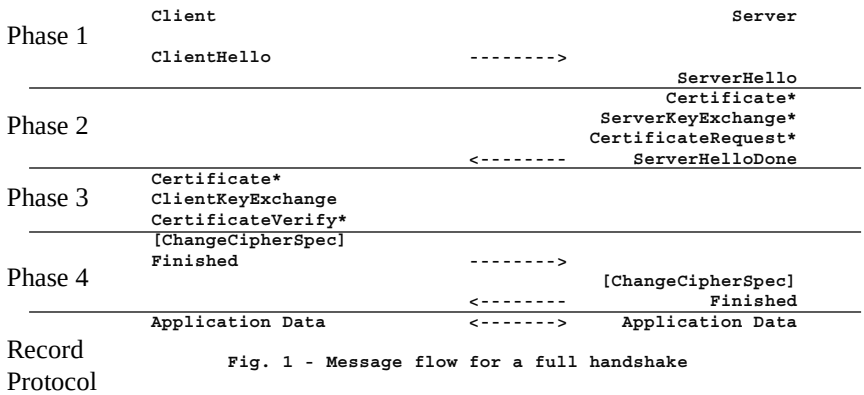


Fig. 1 - Message flow for a full handshake

* Indicates optional or situation-dependent messages that are not always sent.

SSL HANDSHAKE: PHASE 3
CLIENT AUTHENTICATION & KEY EXCHANGE

- **Certificate message**
  - **send 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**

SSL HANDSHAKE: 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**

SSL HANDSHAKE: POST PHASE 3
CRYPTOGRAPHIC COMPUTATION

- **48 byte pre master secret**
  - **RSA**
    - **generated by client**
    - **sent encrypted to server**
  - **DH**
    - **both sides compute the same value**
    - **each side uses its own private value and the other sides public value**

SSL HANDSHAKE: POST PHASE 3
CRYPTOGRAPHIC COMPUTATION

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

pre_master_secret: 48 bytes
```

PRF is composed of a sequence and nesting of HMACs

SSL HANDSHAKE PROTOCOL

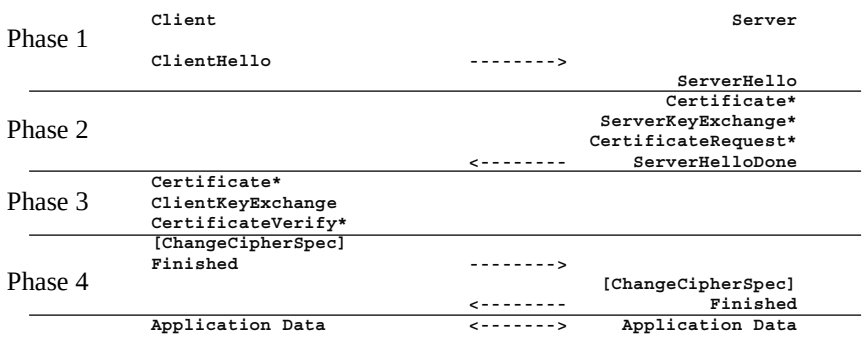


Fig. 1 - Message flow for a full handshake

* Indicates optional or situation-dependent messages that are not always sent.

SSL HANDSHAKE: PHASE 4
FINISH

- **Change Cipher Spec message**
  - **not considered part of handshake protocol but in some sense is part of it**
- **Finished message**
  - **sent under new algorithms and keys**
  - **content is hash of all previous messages and master secret**

SSL HANDSHAKE: PHASE 4
FINISH

- **Change Cipher Spec message**
  - **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**

SSL HANDSHAKE: PHASE 4
FINISH

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 ALERT PROTOCOL

- **2 byte alert messages**
  - **1 byte level**
    - **fatal or warning**
  - **1 byte**
    - **alert code**

SSL ALERT MESSAGES

```
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 ALERT MESSAGES

- **always fatal**
  - **unexpected_message**
  - **bad_record_mac**
  - **decompression_failure**
  - **handshake_failure**
  - **illegal_parameter**

APPLICATIONS AND SSL

- **use dedicated port numbers for every application that uses SSL**
  - **de facto what is happening**
- **use normal application port and negotiate security options as part of application protocol**
- **negotiate use of SSL during normal TCP/IP connection establishment**

APPLICATION PORTS OFFICIAL AND UNOFFICIAL

- **https 443**
- **ssmtp 465**
- **snntp 563**
- **sldap 636**
- **spop3 995**
- **ftp-data 889**
- **ftps 990**
- **imaps 991**
- **telnets 992**
- **ircs 993**