SSL

Prof. Ravi Sandhu
CONTEXT

- **Mid to late 90’s**
  - SSL 1.0 never released
  - SSL 2.0 flawed
  - SSL 3.0 complete redesign
  - TLS from Netscape to IETF

- **Competitors**
  - SET backed by credit card companies
  - S-HTTP (as opposed to https)
  - IPSEC backed by IETF committees
  - SSH for secure remote access to Unix hosts
CRYPTOGRAPHIC SERVICES

- Confidentiality
  - Encryption leaks profusely via side channels
- Authentication + Integrity
  - No point having one without the other
- Non-repudiation
  - Requires asymmetric cryptography
SYMMETRIC KEY ENCRYPTION

INSECURE CHANNEL

Plain-text → Encryption Algorithm E → Ciphertext → Decryption Algorithm D → Plain-text

Symmetric Key shared by A and B

CONFIDENTIAL AND AUTHENTICATED CHANNEL
SYMMETRIC KEY AUTHENTICATION

INSECURE CHANNEL
Plaintext + MAC

CONFIDENTIAL AND AUTHENTICATED CHANNEL
MAC: Message Authentication Code

© Ravi Sandhu
ASYMMETRIC KEY ENCRYPTION

INSECURE CHANNEL

Plain-text → Encryption Algorithm E → Ciphertext → Decryption Algorithm D → Plain-text

A

B's Public Key

B

B's Private Key

AUTHENTICATED CHANNEL
ASYMMETRIC KEY DIGITAL SIGNATURES

INSECURE CHANNEL
Plaintext + Signature

AUTHENTICATED CHANNEL

Plain-text

Signature Algorithm S

Verification Algorithm V

A's Private Key

A's Public Key

Yes/No
SPEED OF ASYMMETRIC KEY VERSUS SYMMETRIC KEY

- Asymmetric key runs 2-3 orders of magnitude slower than symmetric key
- This large difference in speed is likely to remain independent of technology advances
MESSAGE DIGEST

original message
no practical limit to size

message digest algorithm

message digest 160 bit

sign the message digest
not the message

easy

hard
CHALLENGE RESPONSE AUTHENTICATION

WORK STATION ──────NETWORK─────HOST

User ID

Challenge

Response
PUBLIC-KEY CERTIFICATES

- authenticated distribution of public-keys
- public-key encryption
  - sender needs public key of receiver
- public-key digital signatures
  - receiver needs public key of sender
- public-key key agreement
  - either one or both need the other’s public key
X.509v1 CERTIFICATE
authenticated distribution of public-keys

<table>
<thead>
<tr>
<th>VERSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERIAL NUMBER</td>
</tr>
<tr>
<td>SIGNATURE ALGORITHM</td>
</tr>
<tr>
<td>ISSUER</td>
</tr>
<tr>
<td>VALIDITY</td>
</tr>
<tr>
<td>SUBJECT</td>
</tr>
<tr>
<td>SUBJECT PUBLIC KEY INFO</td>
</tr>
<tr>
<td>SIGNATURE</td>
</tr>
</tbody>
</table>
X.509v1 CERTIFICATE

1
1234567891011121314
RSA+MD5, 512
C=US, S=VA, O=GMU, OU=ISE
9/9/99-1/1/1
C=US, S=VA, O=GMU, OU=ISE, CN=Ravi Sandhu
RSA, 1024, xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
SIGNATURE
## CRL FORMAT

<table>
<thead>
<tr>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGNATURE ALGORITHM</td>
</tr>
<tr>
<td>ISSUER</td>
</tr>
<tr>
<td>LAST UPDATE</td>
</tr>
<tr>
<td>NEXT UPDATE</td>
</tr>
<tr>
<td>REVOKED CERTIFICATES</td>
</tr>
<tr>
<td>SIGNATURE</td>
</tr>
<tr>
<td>SERIAL NUMBER</td>
</tr>
<tr>
<td>REVOCATION DATE</td>
</tr>
</tbody>
</table>
X.509 CERTIFICATES

- **X.509v1**
  - very basic
- **X.509v2**
  - adds unique identifiers to prevent against reuse of X.500 names
- **X.509v3**
  - adds many extensions
  - can be further extended
CERTIFICATE TRUST

- how to acquire public key of the issuer to verify signature
- whether or not to trust certificates signed by the issuer for this subject
SINGLE ROOT CA MODEL

Root CA

a b c d e f g h i j k l m n o p

User

Root CA
SINGLE ROOT CA
MULTIPLE RA’S MODEL
MULTIPLE ROOT CA’S MODEL

Root CA

User

User

User

© Ravi Sandhu
MULTIPLE ROOT CA’s PLUS INTERMEDIATE CA’s MODEL

ESTABLISHED BROWSER MODEL

© Ravi Sandhu
SECURE ELECTRONIC TRANSACTIONS (SET) CA HIERARCHY

Root

Brand

Brand

Brand

Geo-Political

Bank

Customer

Acquirer

Merchant
THE CERTIFICATE TRIANGLE

user

X.509 attribute certificate

attribute

SPKI certificate

X.509 identity certificate

public-key
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 ARCHITECTURE

<table>
<thead>
<tr>
<th>SSL Handshake Protocol</th>
<th>SSL Change Cipher Spec Protocol</th>
<th>SSL Alert Protocol</th>
<th>HTTP</th>
<th>Other Application Protocols</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>SSL Record Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
</tr>
<tr>
<td>IP</td>
</tr>
</tbody>
</table>
# APPLICATION PORTS

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>https</td>
<td>443</td>
</tr>
<tr>
<td>ssmtp</td>
<td>465</td>
</tr>
<tr>
<td>snntp</td>
<td>563</td>
</tr>
<tr>
<td>sldap</td>
<td>636</td>
</tr>
<tr>
<td>spop3</td>
<td>995</td>
</tr>
<tr>
<td>ftp-data</td>
<td>889</td>
</tr>
<tr>
<td>ftps</td>
<td>990</td>
</tr>
<tr>
<td>imaps</td>
<td>991</td>
</tr>
<tr>
<td>telnets</td>
<td>992</td>
</tr>
<tr>
<td>ircs</td>
<td>993</td>
</tr>
</tbody>
</table>
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 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
    - each session is created with one connection, but additional connections within the session can be further created
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

#### Phase 1
- **Client**: ClientHello
- **Server**: ServerHello

#### Phase 2
- **Client**: Certificate*
- **Server**: ServerKeyExchange*
- **Server**: CertificateRequest*

#### Phase 3
- **Client**: Certificate*
- **Client**: ClientKeyExchange
- **Client**: CertificateVerify*
- **Server**: [ChangeCipherSpec]
- **Client**: Finished
- **Server**: [ChangeCipherSpec]
- **Server**: ServerHelloDone

#### Phase 4
- **Client**: Application Data
- **Server**: Application Data

---

*Fig. 1 - Message flow for a full handshake*

* Indicates optional or situation-dependent messages that are not always sent.*
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.
SSS 2-WAY HANDSHAKE WITH RSA

<table>
<thead>
<tr>
<th>Phase 1</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td></td>
<td></td>
<td>ClientHello</td>
<td>--------&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><img src="image1.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Certificate*</td>
<td><img src="image2.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ServerKeyExchange*</td>
<td><img src="image3.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CertificateRequest*</td>
<td><img src="image4.png" alt="" /></td>
</tr>
</tbody>
</table>

| Phase 2          |                      |    | Certificate*               | ![Image](image5.png) |
|                  | ![Image](image6.png) |    | ServerHelloDone            | ![Image](image7.png) |
|                  | ![Image](image8.png) |    | Certificate*               | ![Image](image9.png) |
|                  | ![Image](image10.png) |    | ServerHelloDone            | ![Image](image11.png) |

| Phase 3          |                      |    | Certificate*               | ![Image](image12.png) |
|                  | ![Image](image13.png) |    | ServerHelloDone            | ![Image](image14.png) |
|                  | ![Image](image15.png) |    | Certificate*               | ![Image](image16.png) |
|                  | ![Image](image17.png) |    | ServerHelloDone            | ![Image](image18.png) |

| Phase 4          |                      |    | [ChangeCipherSpec]         | ![Image](image19.png) |
|                  | ![Image](image20.png) |    | Finished                   | ![Image](image21.png) |
|                  | ![Image](image22.png) |    | [ChangeCipherSpec]         | ![Image](image23.png) |
|                  | ![Image](image24.png) |    | Finished                   | ![Image](image25.png) |

| Record Protocol  |                      |    | Application Data           | ![Image](image26.png) |
|                  | ![Image](image27.png) |    | Application Data           | ![Image](image28.png) |

* 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

ClientHello --------> ServerHello

[ChangeCipherSpec] [ChangeCipherSpec]

Finished                      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

Phase 1

Client

ClientHello --------> Server

ServerHello

Certificate*

ServerKeyExchange*

CertificateRequest*

<-------- ServerHelloDone

Phase 2

Certificate*

ClientKeyExchange

CertificateVerify*

[ChangeCipherSpec]

Finished

--------->

[ChangeCipherSpec]

Finished

Phase 3

[ChangeCipherSpec]

Finished

<-------- Application Data

Phase 4

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: 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**

---

**Phase 1**
- **Client**
  - ClientHello
- **Server**
  - ServerHello

---

**Phase 2**
- **Server**
  - Certificate*
  - ServerKeyExchange*
  - CertificateRequest*
- **Client**
  - ServerHelloDone

---

**Phase 3**
- **Client**
  - Certificate*
  - ClientKeyExchange
  - CertificateVerify*
- **Server**
  - [ChangeCipherSpec]
  - Finished

---

**Phase 4**
- **Server**
  - [ChangeCipherSpec]
  - Finished
- **Client**
  - Application Data
- **Server**
  - Application Data

---

**Record 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

Phase 1

Client

ClientHello --------> Server

ServerHello

Certificate*
ServerKeyExchange*
CertificateRequest*

<-------- ServerHelloDone

Phase 2

Certificate*
ClientKeyExchange
CertificateVerify*

<--------

[ChangeCipherSpec]
Finished

Phase 3

[ChangeCipherSpec]
Finished

Phase 4

Application Data

<-------->

Application Data

Finished

Record 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

\[
\text{master\_secret} = \text{PRF}(\text{pre\_master\_secret}, \text{"master secret"}, \text{ClientHello.random + ServerHello.random})[0..47];
\]

\text{pre\_master\_secret: 48 bytes}

PRF is composed of a sequence and nesting of HMACs
SSL HANDSHAKE PROTOCOL

Phase 1
Client

ClientHello

Server

ServerHello

Certificate*
ServerKeyExchange*
CertificateRequest*

<-------- ServerHelloDone

Phase 2
Certificate*

Phase 3
ClientKeyExchange
CertificateVerify*

[ChangeCipherSpec]
Finished

Phase 4

<-------- [ChangeCipherSpec]

Finished

Application Data
<-------->
Application Data

Record 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
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_restiction(60),
protocol_version(70),
insufficient_security(71),
internal_error(80),
userCanceled(90),
no_renegotiation(100),
SSL ALERT MESSAGES

- always fatal
  - unexpected_message
  - bad_record_mac
  - decompression_failure
  - handshake_failure
  - illegal_parameter
SERVER-SIDE SSL (OR 1-WAY) HANDSHAKE WITH RSA

Handshake Protocol

Client

ClientHello

Server

ServerHello

Certificate

<-------- ServerHelloDone

ClientKeyExchange

[ChangeCipherSpec]

Finished

<-------- [ChangeCipherSpec]

<-------- Finished

Application Data

<-------- Application Data

Record Protocol
SERVER-SIDE MASQUARADING

Bob
Web browser
Server-side SSL
www.host.com
Web server

Ultratrust
Security
Services

www.host.com
SERVER-SIDE MASQUARADING

Bob
Web browser

www.host.com
Web server

Server-side SSL

Mallory’s Web server

BIMM Corporation

www.host.com

Ultratrust Security Services

www.host.com
SERVER-SIDE MASQUARADING

Bob
Web browser

www.host.com
Web server

BIMM Corporation
Ultratrust Security Services
www.host.com

Mallory’s Web server

Ultratrust Security Services
www.host.com
CLIENT-SIDE SSL (OR 2-WAY) HANDSHAKE WITH RSA

Handshake Protocol

Client

ClientHello

Certificate

CertificateRequest

ClientKeyExchange

CertificateVerify

[ChangeCipherSpec]

Finished

--------->

ServerHello

Certificate

CertificateRequest

ServerHelloDone

Server

[ChangeCipherSpec]

Finished

<-------->

Application Data

Record Protocol

<-------->

Application Data
MAN IN THE MIDDLE
MASQUARADING PREVENTED

Client Side SSL
end-to-end

Bob
Web browser

www.host.com
Web server

BIMM Corporation

Ultratrust Security Services

www.host.com

Bob

BIMM Corporation

Ultratrust Security Services

www.host.com

Bob

SSL

- Deployed in broken form
- Guardian of e-commerce
- World’s most successful crypto protocol