The SSL and TLS protocols

-- architecture and services
-- SSL sessions and connections
-- SSL Record Protocol
-- SSL Handshake Protocol
-- changes in the TLS protocol

What is SSL?

- SSL – Secure Socket Layer
- it provides a secure transport connection between applications (e.g., a web server and a browser)
- SSL was developed by Netscape
- SSL version 3.0 has been implemented in many web browsers (e.g., Netscape Navigator and MS Internet Explorer) and web servers and widely used on the Internet
- SSL v3.0 was specified in an Internet Draft (1996)
- it evolved into RFC 2246 and was renamed to TLS v1.0 (Transport Layer Security)
- current version is TLS v1.1 (RFC 4346)
  - modifications to handle CBC attacks: explicit IV and bad_record_mac error message instead of decryption_failed

(c) Levente Buttyán (buttyan@crysys.hu)
SSL architecture

SSL components

- SSL Handshake Protocol
  - negotiation of security algorithms and parameters
  - key exchange
  - server authentication and optionally client authentication

- SSL Record Protocol
  - fragmentation
  - compression
  - message authentication and integrity protection
  - encryption

- SSL Alert Protocol
  - error messages (fatal alerts and warnings)

- SSL Change Cipher Spec Protocol
  - a single message that indicates the end of the SSL handshake
Sessions and connections

- an SSL session is an association between a client and a server
- sessions are stateful; the session state includes security algorithms and parameters
- a session may include multiple secure connections between the same client and server
- connections of the same session share the session state
- sessions are used to avoid expensive negotiation of new security parameters for each connection
- there may be multiple simultaneous sessions between the same two parties, but this feature is not used in practice

Session and connection states

- session state
  - session identifier
    - arbitrary byte sequence chosen by the server to identify the session
  - peer certificate
    - X509 certificate of the peer
    - may be null
  - compression method
  - cipher spec
    - bulk data encryption algorithm (e.g., null, DES, 3DES, …)
    - MAC algorithm (e.g., (HMAC-like with) MD5, SHA-1)
    - cryptographic attributes (e.g., hash size, IV size, …)
  - master secret
    - 48-byte secret shared between the client and the server
  - is resumable
    - a flag indicating whether the session can be used to initiate new connections
  - connection states
**Session and connection states cont’d**

- **connection state**
  - server and client random
    - random byte sequences chosen by the server and the client for every connection
  - server write MAC secret
    - secret key used in MAC operations on data sent by the server
  - client write MAC secret
    - secret key used in MAC operations on data sent by the client
  - server write key
    - secret encryption key for data encrypted by the server
  - client write key
    - secret encryption key for data encrypted by the client
  - initialization vectors
    - an IV is maintained for each encryption key if CBC mode is used
    - initialized by the SSL Handshake Protocol
    - final ciphertext block from each record is used as IV with the following record
  - sending and receiving sequence numbers
    - sequence numbers are 64 bits long
    - reset to zero after each Change Cipher Spec message

**State changes**

- **operating state**
  - currently used state
- **pending state**
  - state to be used
  - built using the current state
- **operating state ↔ pending state**
  - at the transmission and reception of a Change Cipher Spec message

---

Diagram:

- Party A (client or server)
- Party B (server or client)

The sending part of the pending state is copied into the sending part of the operating state.

The receiving part of the pending state is copied into the receiving part of the operating state.
SSL Record Protocol – processing overview

- fragmentation
- compression
- MAC computation
- padding
- encryption

→ SSL Record Protocol message:

```
+----------------+----------------+----------------+
<table>
<thead>
<tr>
<th>type</th>
<th>version</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>application data</td>
<td>(compressed fragment)</td>
<td></td>
</tr>
<tr>
<td>MAC</td>
<td>padding</td>
<td>p.len</td>
</tr>
</tbody>
</table>
```

**Header**

- **type**
  - the higher level protocol used to process the enclosed fragment
  - possible types:
    - change_cipher_spec
    - alert
    - handshake
    - application_data

- **version**
  - SSL version, currently 3.0

- **length**
  - length (in bytes) of the enclosed fragment or compressed fragment
  - max value is $2^{14} + 2048$
MAC

MAC = hash( MAC_wr_sec | pad_2 | 
    hash( MAC_wr_sec | pad_1 | seq_num | type | length | frag ))

- similar to HMAC but the pads are concatenated
- supported hash functions:
  - MD5
  - SHA-1
- pad_1 is 0x36 repeated 48 times (MD5) or 40 times (SHA-1)
- pad_2 is 0x5C repeated 48 times (MD5) or 40 times (SHA-1)

Encryption

- supported algorithms
  - block ciphers (in CBC mode)
    - RC2_40
    - DES_40
    - DES_56
    - 3DES_168
    - IDEA_128
    - Fortezza_80
  - stream ciphers
    - RC4_40
    - RC4_128
- if a block cipher is used, than padding is applied
  - last byte of the padding is the padding length
SSL Alert Protocol

- each alert message consists of 2 fields (bytes)
- first field (byte): "warning" or "fatal"
- second field (byte):
  - fatal
    - unexpected_message
    - bad_record_MAC
    - decryption_failure
    - handshake_failure
    - illegal_parameter
  - warning
    - close_notify
    - no_certificate
    - bad_certificate
    - unsupported_certificate
    - certificate_revoked
    - certificate_expired
    - certificate_unknown

- in case of a fatal alert
  - connection is terminated
  - session ID is invalidated → no new connection can be established within this session

SSL Handshake Protocol – overview

**Phase 1:** Negotiation of the session ID, key exchange algorithm, MAC algorithm, encryption algorithm, and exchange of initial random numbers

**Phase 2:** Server may send its certificate and key exchange message, and it may request the client to send a certificate. Server signals end of hello phase.

**Phase 3:** Client sends certificate if requested and may send an explicit certificate verification message. Client always sends its key exchange message.

**Phase 4:** Change cipher spec and finish handshake
Hello messages

- **client_hello**
  - **client_version**
    - the highest version supported by the client
  - **client_random**
    - current time (4 bytes) + pseudo random bytes (28 bytes)
  - **session_id**
    - empty if the client wants to create a new session, or
    - the session ID of an old session within which the client wants to create the new connection
  - **cipher_suites**
    - list of cryptographic options supported by the client ordered by preference
    - a cipher suite contains the specification of the
      - key exchange method, the encryption and the MAC algorithm
      - the algorithms implicitly specify the hash_size, IV_size, and key_material parameters (part of the Cipher Spec of the session state)
    - example: SSL_RSA_with_3DES_EDE_CBC_SHA
  - **compression_methods**
    - list of compression methods supported by the client

Hello messages cont’d

- **server_hello**
  - **server_version**
    - min( highest version supported by client, highest version supported by server )
  - **server_random**
    - current time + random bytes
    - random bytes must be independent of the client random
  - **session_id**
    - session ID chosen by the server
    - if the client wanted to resume an old session:
      - server checks if the session is resumable
      - if so, it responds with the session ID and the parties proceed to the finished messages
    - if the client wanted a new session
      - server generates a new session ID
  - **cipher_suite**
    - single cipher suite selected by the server from the list given by the client
  - **compression_method**
    - single compression method selected by the server
Supported key exchange methods

- **RSA based** (SSL_RSA_with...)
  - the secret key (pre-master secret) is encrypted with the server's public RSA key
  - the server's public key is made available to the client during the exchange
- **fixed Diffie-Hellman** (SSL_DH_RSA_with... or SSL_DH_DSS_with...)
  - the server has fix DH parameters contained in a certificate signed by a CA
  - the client may have fix DH parameters certified by a CA or it may send an unauthenticated one-time DH public value in the client_key_exchange message
- **ephemeral Diffie-Hellman** (SSL_DHE_RSA_with... or SSL_DHE_DSS_with...)
  - both the server and the client generate one-time DH parameters
  - the server signs its DH parameters with its private RSA or DSS key
  - the client may authenticate itself (if requested by the server) by signing the hash of the handshake messages with its private RSA or DSS key
- **anonymous Diffie-Hellman** (SSL_DH_anon_with...)
  - both the server and the client generate one-time DH parameters
  - they send their parameters to the peer without authentication
- **Fortezza**
  - Fortezza proprietary key exchange scheme

Server certificate and key exchange msgs

- **certificate**
  - required for every key exchange method except for anonymous DH
  - contains one or a chain of X.509 certificates (up to a known root CA)
  - may contain
    - public RSA key suitable for encryption, or
    - public RSA or DSS key suitable for signing only, or
    - fix DH parameters

- **server_key_exchange**
  - sent only if the certificate does not contain enough information to complete the key exchange (e.g., the certificate contains an RSA signing key only)
  - may contain
    - public RSA key (exponent and modulus), or
    - DH parameters (p, g, public DH value), or
    - Fortezza parameters
  - digitally signed
    - if DSS: SHA-1 hash of (client_random | server_random | server_params) is signed
    - if RSA: MD5 hash and SHA-1 hash of (client_random | server_random | server_params) are concatenated and encrypted with the private RSA key
### Cert request and server hello done msgs

- **certificate_request**
  - sent if the client needs to authenticate itself
  - specifies which type of certificate is requested (rsa_sign, dss_sign, rsa_fixed_dh, dss_fixed_dh, …)

- **server_hello_done**
  - sent to indicate that the server is finished its part of the key exchange
  - after sending this message the server waits for client response
  - the client should verify that the server provided a valid certificate and the server parameters are acceptable

### Client authentication and key exchange

- **certificate**
  - sent only if requested by the server
  - may contain
    - public RSA or DSS key suitable for signing only, or
    - fix DH parameters

- **client_key_exchange**
  - always sent (but it is empty if the key exchange method is fix DH)
  - may contain
    - RSA encrypted pre-master secret, or
    - client one-time public DH value, or
    - Fortezza key exchange parameters

- **certificate_verify**
  - sent only if the client sent an RSA or DSS certificate
  - provides client authentication
  - contains signed hash of all the previous handshake messages
    - if DSS: SHA-1 hash is signed
    - if RSA: MD5 and SHA-1 hash is concatenated and encrypted with the private key
      MD5( master_secret | pad_2 | MD5( handshake_messages | master_secret | pad_1 ) )
      SHA( master_secret | pad_2 | SHA( handshake_messages | master_secret | pad_1 ) )
## Finished messages

- **finished**
  - sent immediately after the change_cipher_spec message
  - used to authenticate all previous handshake messages
  - first message that uses the newly negotiated algorithms, keys, IVs, etc.
  - contains the MD5 and SHA-1 hash of all the previous handshake messages:
    \[
    \text{MD5}( \text{master_secret} | \text{pad}_2 | \text{MD5}( \text{handshake_messages} | \text{sender} | \text{master_secret} | \text{pad}_1 ) ) | \\
    \text{SHA}( \text{master_secret} | \text{pad}_2 | \text{SHA}( \text{handshake_messages} | \text{sender} | \text{master_secret} | \text{pad}_1 ) )
    \]
    where "sender" is a code that identifies that the sender is the client or the server (client: 0x434C4E54; server: 0x53525652)

## Cryptographic computations

- **pre-master secret**
  - if key exchange is RSA based:
    - generated by the client
    - sent to the server encrypted with the server's public RSA key
  - if key exchange is Diffie-Hellman based:
    - \( \text{pre_master_secret} = g^{xy} \mod p \)

- **master secret (48 bytes)**
  \[
  \text{master_secret} = \text{MD5}( \text{pre_master_secret} | \text{SHA}( \text{"A"} | \text{pre_master_secret} | \text{client_random} | \text{server_random}) ) | \\
  \text{MD5}( \text{pre_master_secret} | \text{SHA}( \text{"BB"} | \text{pre_master_secret} | \text{client_random} | \text{server_random}) ) | \\
  \text{MD5}( \text{pre_master_secret} | \text{SHA}( \text{"CCC"} | \text{pre_master_secret} | \text{client_random} | \text{server_random}) )
  \]

- **keys, MAC secrets, IVs**
  \[
  \text{MD5}( \text{master_secret} | \text{SHA}( \text{"A"} | \text{master_secret} | \text{client_random} | \text{server_random}) ) | \\
  \text{MD5}( \text{master_secret} | \text{SHA}( \text{"BB"} | \text{master_secret} | \text{client_random} | \text{server_random}) ) | \\
  \text{MD5}( \text{master_secret} | \text{SHA}( \text{"CCC"} | \text{master_secret} | \text{client_random} | \text{server_random}) ) | \ldots
  \]

**key block:**
- client write MAC sec
- server write MAC sec
- client write key
- server write key
- ...
**Miscellaneous changes in TLS**

- **version number**
  - for TLS 1.1 the version number is 3.2

- **cipher suites**
  - TLS doesn’t support Fortezza key exchange and Fortezza encryption

- **padding**
  - variable length padding is allowed (max 255 padding bytes)

- **MAC**
  - TLS uses the latest version of HMAC
  - the MAC covers the version field of the record header too

- **certificate_verify message**
  - in SSL, the hash contains the master_secret
  - in TLS, the hash is computed only over the handshake messages

- **more alert codes**

---

**New pseudorandom function (PRF)**

- \[ P_{\text{hash}}(\text{secret}, \text{seed}) = \text{HMAC}\_hash( \text{secret}, A(1) \mid \text{seed} ) \mid \]
  - \[ \text{HMAC}\_hash( \text{secret}, A(2) \mid \text{seed} ) \mid \]
  - \[ \text{HMAC}\_hash( \text{secret}, A(3) \mid \text{seed} ) \mid \ldots \]

  where
  \[
  A(0) = \text{seed} \\
  A(i) = \text{HMAC}\_hash(\text{secret}, A(i-1))
  \]

- \[ \text{PRF}(\text{secret}, \text{label}, \text{seed}) = \]
  - \[ P_{\text{MD5}}(\text{secret}\_left, \text{label} \mid \text{seed}) \oplus P_{\text{SHA}}(\text{secret}\_right, \text{label} \mid \text{seed}) \]
Usage of the new PRF

- **finished message**
  
  \[
  \text{PRF}( \text{master\_secret}, \\
  "\text{client finished}", \\
  \text{MD5(handshake\_messages)} | \text{SHA(handshake\_messages)} )
  \]

- **cryptographic computations**
  
  - pre-master secret is calculated in the same way as in SSL
  
  - master secret:
    
    \[
    \text{PRF}( \text{pre\_master\_secret}, \\
    "\text{master secret}", \\
    \text{client\_random | server\_random} )
    \]

  - key block:
    
    \[
    \text{PRF}( \text{master\_secret}, \\
    "\text{key expansion}", \\
    \text{server\_random | client\_random} )
    \]