Security Protocols: the case of Secure Sockets Layer (SSL) and Transport Layer Security (TLS)

G. Bertoni
updated by L. Breveglieri
TCP / IP

- the Transmission Control Protocol (TCP) combined with the Internet Protocol (IP) governs the transportation and routing of data over the Internet
- on top of the TCP/IP it is possible to run applications as HTTP, IMAP, LDAP, …
- but TCP/IP does not provide any security mechanism
SSL / TLS

• one solution is to introduce a stack as the SSL protocol on top of TCP/IP

• SSL introduces some mechanisms in order to guarantee:
  – confidentiality
  – data integrity
  – and authentication of parties
SSL / TLS

- difference between SSL and TLS?
  - not so great, basically SSL v3 was proposed by Netscape, while TLS is a project from IETF for having a working group on the protocol
  - TLS has been built starting from SSL
  - TLS can downgrade its routine for being compatible to SSL3
- current versions: TLS v. 1.0, SSL v. 3.0
Protocol Architecture

• the protocol uses a set of cryptographic algorithms (or primitives), in order to guarantee different levels of security
• algorithms with lower security are available for low end devices, since they are computationally lighter
• it is also requested to have backward compatibility
• however, a server or a client can decide a minimum security level and deny communication if the counterpart is willing to use a low security algorithm
Secret Key

- the symmetric ciphers supported by SSL / TLS are the following:
  - DES and 3DES
  - RC2 and RC4
  - Skipjack
  - IDEA
  - AES

- hash functions
  - MD5 and SHA-1
Public Key

• the asymmetric ciphers supported by SSL / TLS are the following:
  – DSA and RSA are used for signature verification
  – RSA is used for key exchange
  – in addition, there are some algorithms recommended by the US government for internal use only
Start

• communication starts with the client sending its SSL / TLS version number, the cipher setting and some random data (ClientHello.random)

• the server answers with the SSL version number of the server, cipher setting, random data (ServerHello.random) and the certificate of the server
Certificate Check

- once received the server certificate, the client checks the validity of it
- this is done in four main steps:
  - check the validity period
  - check whether the issuing CA is in the list of the CAs that are trusted by the client
  - check the digital signature of the certificate
  - check whether the domain name included in the certificate matches with the actual name of the server
Pre-Master Secret

- if the four steps are passed, then the client creates the pre-master secret for the session
- the pre-master secret is sent to the server using public key encryption
- the server can optionally request client authentication
  - in this case the client has to sign some random data and send them along with the client certificate
  - the server checks the client certificate and the signature of the random data
Beginning of the Communication

• once the server has received the pre-master secret, both parties can compute the master secret
• then session keys are generated from the master secret
• each party sends a message informing the counterpart that all future communications will be encrypted
Master Secret Computation

• both client and server use the so-called Pseudo Random Function (PRF)
• PRF is:
  \[
  \text{PRF (secret, label, seed)} = \text{P\_MD5 (S1, label + seed)} \oplus \text{P\_SHA-1 (S2, label + seed)}
  \]
  where S1 and S2 are the first and second half of the pre-master secret
• P_hash is:
  \[
  \text{P\_hash (secret, seed)} = \text{HMAC\_hash (secret, A(1) + seed)} + \\
  \text{HMAC\_hash (secret, A(2) + seed)} + \\
  \text{HMAC\_hash (secret, A(3) + seed)} + ... (i times)
  \]
  where the A’s are constants
Master Secret Computation

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

- the requested length is 48 bytes

- from the master secret it is derived the key block, which is the set of session keys and initialisation vectors necessary for the block ciphers and HMAC
Key Block Computation

- the key block is obtained by applying once more the PRF
  \[
  \text{key\_block} = \text{PRF} (\text{SecurityParameters.master\_secret,}
  \text{"key expansion"}, \text{SecurityParameters.server\_random +}
  \text{SecurityParameters.client\_random})
  \]

- then the key\_block is partitioned as follows:
  \[
  \begin{align*}
  \text{client\_write\_MAC\_secret} & \ [\text{SecurityParameters.hash\_size}] \\
  \text{server\_write\_MAC\_secret} & \ [\text{SecurityParameters.hash\_size}] \\
  \text{client\_write\_key} & \ [\text{SecurityParameters.key\_material\_length}] \\
  \text{server\_write\_key} & \ [\text{SecurityParameters.key\_material\_length}] \\
  \text{client\_write\_IV} & \ [\text{SecurityParameters.IV\_size}] \\
  \text{server\_write\_IV} & \ [\text{SecurityParameters.IV\_size}]
  \end{align*}
  \]

- the length of the key block depends on the algorithms used
Usage

- server and client use the session keys for protecting privacy (write_key) and data integrity (write_MAC_secret)
- every session key is unidirectional:
  - client_write or server_write
Session and Connection

• if the parties assume that the master secret has not been compromised, it is possible to resume a session:
  – new ClientHello.random and ServerHello.random values are exchanged
  – then they are hashed with the master secret for generating a new key block
• sessions cannot be resumed unless both the client and server agree
• if either party suspects that the session may have been compromised, or that certificates may have expired or may have been revoked, it should force a new full handshake
• an upper limit of 24 hours is suggested for session lifetime