isec
INFORMATION SECURITY
lectures by Ferucio Laurentiu Tiplea

Web Security

Secure Socket Layer and Transport Layer Security

Prof.dr. Ferucio Laurențiu Țiplea

Fall 2021

Department of Computer Science
“Alexandru Ioan Cuza” University of Iași
Iași 700506, Romania

e-mail: `ferucio.tiplea@uaic.ro`

Outline

- Secure Socket Layer (SSL)
  - Overview
  - Sessions and connections
  - Key exchange: the handshake protocol
  - SSL record protocol
  - SSL change cipher spec protocol
  - SSL alert protocol
  - SSL application data protocol
- Transport Layer Security (TLS)
- SSL and TLS applications
- Security of TLS

Secure Socket Layer (SSL)

Secure Sockets Layer (SSL)

SSL, followed by TLS, is a protocol whose primary purpose is to establish a private communication channel between two applications to ensure:

- Privacy of data;
- Authentication of the parties;
- Integrity.

A bit of history:

- SSL was developed by Netscape Communication Corporation when Taher ElGamal was a chief scientist from 1995 to 1998;
- SSL 1.0 was never released because of serious security flaws;
- SSL 2.0, released in Feb 1995, was shortly discovered to contain security and usability flaws;
- Version 3.0 released in Nov 1996 is a complete reconstruction of the SSL protocol, being the basis for further developments.

SSL 3.0

Taher ElGamal is often referred to as the father of SSL.

Version 3.0 of SSL was produced by Paul Kocher, working with Netscape engineers:

> *"Taher had a clear vision for SSL 3.0, but had many other things on his platter as well, so he made arrangements for me and Phil Karlton to do the main work of designing the protocol."*
>
> *Paul Kocher*

SSL is a significant contribution to the practice of cryptography:

> *"The entire Internet population benefits from the work of Kocher and ElGamal every day."*
>
> *Dr. Vinton Cerf, Chair of the Marconi Society*

Guglielmo Marconi (1874-1937), the inventor of radio, shared the 1909 Nobel Prize in Physiscs with Karl Braun.

The 2019 Marconi prize

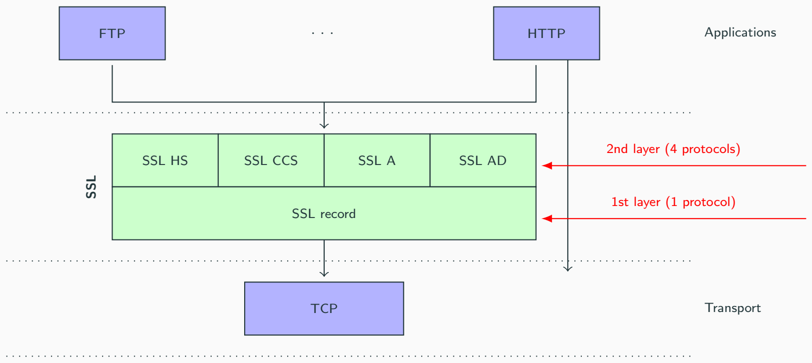
Figure 1: Paul Kocher (left) and Taher ElGamal (right) awarded the 2019 Marconi prize for their contributions to the security of communications.

SSL in a nutshell

Even though SSL 3.0 has been implemented and used since 1996, the first complete document published appeared only in Aug 2011 (Freier et al. (2011)). Some points of reference on it:

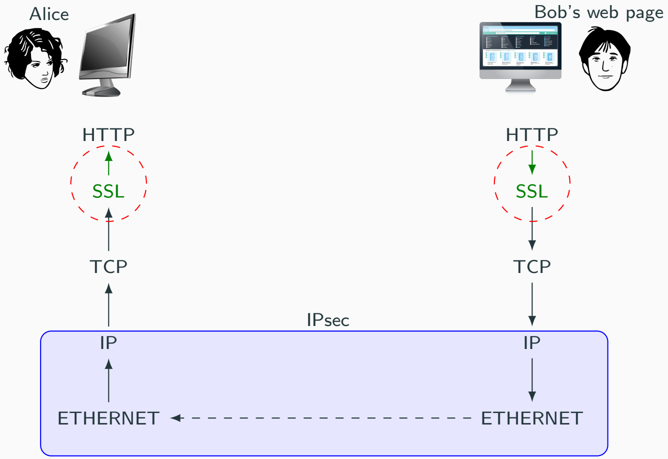
1. Primary goal: to provide a private channel between communicating applications to ensure:
    - Privacy of data;
    - Authentication of the parties;
    - Integrity;
2. SSL does not provide non-repudiation;
3. SSL is socket-oriented: all or none of the data that is sent or received from a socket are protected in the same way, i.e., there is no way to sign individual pieces of data;
4. SSL is application independent.

SSL in the TCP/IP protocol stack



SSL provides security services to any TCP-based application protocol (e.g., HTTP, FTP etc.).

Usage scenario



SSL sessions and connections

- SSL session
    - Association between two communicating peers;
    - Defines a set of cryptographic parameters which can be shared among multiple connections;
    - Created by the SSL handshake protocol;
    - Primarily used to avoid expensive negotiation of new security parameters for each connection;
- SSL connection
    - Transport (in the OSI layering model definition) that provides a suitable type of service;
    - Each connection is associated with one session.

Multiple simultaneous sessions between a pair of parties may coexist.

Session states

SSL sessions are stateful. Each peer maintains two states:

1. read, for receive;
2. write, for send.

Logically, each state is represented twice, as current and pending:

- Assume B is sending messages to A:
    - Peer A: read state is pending (A is receiving messages) and write state is current;
    - Peer B: read state is current and write state is pending (B is sending messages);
- Assume B signaled that the transmission had ended:
    - Peer A: pending read goes to current read;
    - Peer B: pending write goes to current write.

Session state parameters

- Session id – a byte sequence chosen by the server to identify an active or resumable session state;
- Peer certificate – an X509.v3 certificate of the peer (this element may be null);
- Compression method – the algorithm used to compress data prior to encryption;
- CipherSpec – defines the data encryption algorithm (such as null, DES etc.), a hash algorithm (such as MD5 or SHA-1), and cryptographic attributes (such as the hash size);
- Master secret – a 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.

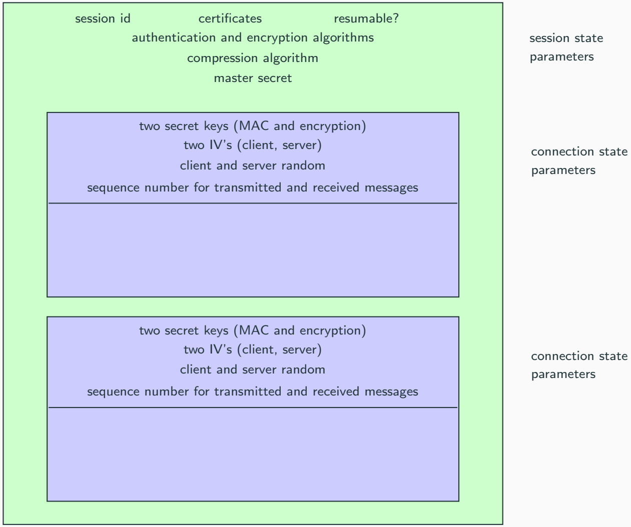
A vector of values for these parameters is usually called a session state.

Connection state parameters

- Client/server random value N_C/N_S – a byte sequence chosen by the client/server for each connection;
- Client/server write MAC secret K^C_{MAC}/K^S_{MAC} – secret key used in MAC operations on data sent by client/server;
- Client/server write key K_C/K_S – secret key used by client/server to encrypt and by server/client to decrypt;
- Initialization vector IV – used in CBC mode;
- Sequence numbers – each party maintains separate sequence numbers for transmitted and received messages for each connection.

A vector of values for these parameters is usually called a connection state.

Pictorial view of state and connection parameters


session id
certificates
resumable?
authentication and encryption algorithms
compression algorithm
master secret
session state parameters
two secret keys (MAC and encryption)
two IV's (client, server)
client and server random
sequence number for transmitted and received messages
connection state parameters
two secret keys (MAC and encryption)
two IV's (client, server)
client and server random
sequence number for transmitted and received messages
connection state parameters

SSL handshake protocol

The handshake protocol is the most complex protocol of SSL. It allows parties to:

- Agree on a protocol version;
- Negotiate a cryptographic suite and a compression method;
- Exchange keys;
- Authenticate each other.

It comes in two main forms:

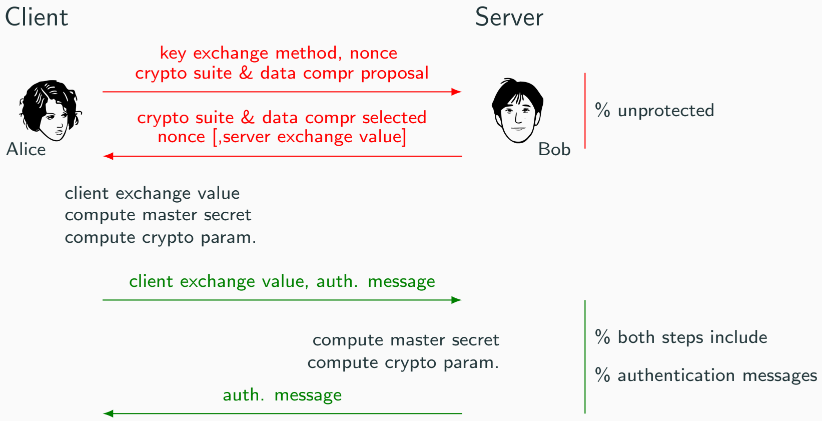
- To establish a new session;
- To resume a session.

Key exchange in SSL and the pre-master secret

The SSL 3.0 protocol implements the following key exchange methods:

1. RSA: the client generates a pre-master secret that is encrypted with the server's public key, and sends it to the server;
2. DH: the pre-master secret is the DH value obtained from the client's and server's DH public parameters. There are three variants:
   - Fixed DH: the server must have a certificate which should include his DH public parameters. The client provides its DH public parameters either in a certificate or in a key exchange message;
   - Ephemeral (temporary, one-time) DH: DH public parameters are exchanged and signed using sender's private RSA or DSS key. Certificates are needed to authenticate the public keys;
   - Anonymous DH: this is DH with no authentication;
3. Fortezza: this is the Fortezza scheme (based on the Skipjack cipher).

The handshake protocol illustrated


Client
Server
key exchange method, nonce
crypto suite & data compr proposal
Alice
Bob
crypto suite & data compr selected
nonce [,server exchange value]
% unprotected
client exchange value
compute master secret
compute crypto param.
client exchange value, auth. message
compute master secret
compute crypto param.
auth. message
% both steps include
% authentication messages

Master secret generation

Both peers compute a master secret as follows:

$$\begin{aligned} master\_secret \quad = \quad & MD5(pre\text{-}master\_secret \parallel sha(A, pre\text{-}master\_secret))|| \\ & MD5(pre\text{-}master\_secret \parallel sha(B, pre\text{-}master\_secret))|| \\ & MD5(pre\text{-}master\_secret \parallel sha(C, pre\text{-}master\_secret)) \end{aligned}$$

where:

- $sha(A) = SHA(`0x41' \parallel pre\text{-}master\_secret \parallel N_C \parallel N_S)$;
- $sha(B) = SHA(`0x4242' \parallel pre\text{-}master\_secret \parallel N_C \parallel N_S)$;
- $sha(C) = SHA(`0x434343' \parallel pre\text{-}master\_secret \parallel N_C \parallel N_S)$;
- $N_C =$ client_hello_random (client nonce);
- $N_S =$ server_hello_random (server nonce).

Generation of cryptographic parameters

The master secret is used to generate a key bloc

$$\begin{aligned} key\_bloc &= MD5(master\_secret \parallel sha(A, master\_secret))|| \\ &\quad MD5(master\_secret \parallel sha(B, master\_secret))|| \\ &\quad MD5(master\_secret \parallel sha(C, master\_secret))|| \\ &\quad \ldots \end{aligned}$$

from which the cryptographic parameters are sliced in order:

- client/server write MAC;
- client/server write key;
- client/server write IV.

In some cases, the client/server write key and the client/server write IV may be subject to some additional processing.

The handshake protocol for a new session

1. $C \to S$: *ClientHello*
2. $S \to C$: *ServerHello*, $[Certificate,][ServerKeyExchange,][CertificateRequest,]$ *ServerHelloDone*
3. $C \to S$: $[Certificate,]$ *ClientKeyExchange*, $[CertificateVerify,]$ *ChangeCipherSpec*, *Finished*
4. $S \to C$: *ChangeCipherSpec*, *Finished*

The handshake protocol for resuming a session

The simplified version below of the SSL protocol is to be used when the client and server decide to resume a previous session or duplicate an existing one, instead of negotiating new security parameters:

$$\begin{array}{lll}
1. & C \rightarrow S: & \mathit{ClientHello} \\
2. & S \rightarrow C: & \mathit{ServerHello}, \\
 & & \mathit{ChangeCipherSpec}, \\
 & & \mathit{Finished} \\
3. & C \rightarrow S: & \mathit{ChangeCipherSpec}, \\
 & & \mathit{Finished}
\end{array}$$

The handshake protocol: the first step

ClientHello message includes:

- Highest SSL version understood by the client;
- Random value generated by client for use in the master secret generation;
- Session ID (it is 0 if the client is starting a new session);
- A key exchange method (RSA, DH, Fortezza);
- Cipher suite list (in order of preference). A cipher suite consists of:
    - A cipher algorithm (e.g., 3DES) and a cipher type (stream or block);
    - A MAC algorithm (MD5 or SHA-1) and a hash value;
    - A value for "is exportable" (true or false);
    - Key material (used in generating the write keys);
    - The size of IV;
- Data compression methods (supported by the client).

The handshake protocol: the second step

ServerHello message includes: version number, time information, session ID, chiper suite, compression method, random value.

Certificate message includes a server certificate (if the server is required to be authenticated by the key exchange method).

ServerKeyExchange message consists of the elements provided by server according to the key exchange method.

CertificateRequest message requests a client certificate if the client is required to be authenticated.

ServerHelloDone message indicates the end of the server hello and associated messages.

The handshake protocol: the third step

ClientKeyExchange message consists of the elements provided by client according to the key exchange method.

CertificateVerify message provides explicit verification of the client certificate. It contains a signature with the client private key over hash values. The hash values are computed as follows:

$$h(master\_secret \parallel pad\_2 \parallel h(handshake\_messages \parallel master\_secret \parallel pad\_1))$$

where h is MD5 or SHA-1 and

$$pad\_1 = \begin{cases} (0x36)^{48}, & for\ MD5 \\ (0x36)^{40}, & for\ SHA\text{-}1 \end{cases} \qquad pad\_2 = \begin{cases} (0x5c)^{48}, & for\ MD5 \\ (0x5c)^{40}, & for\ SHA\text{-}1 \end{cases}$$

ChangeCipherSpec – see next slides.

The handshake protocol: the “Finished” message

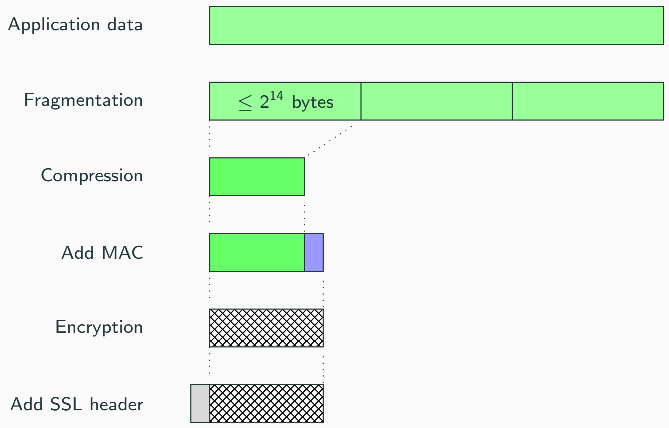
The Finished message indicates that the negotiation part is completed and authenticates the cryptographic parameters established by peers. It includes an MD5 hash value (16 bytes) followed by a SHA-1 hash value (20 bytes). The structure of hash values is similar to the one on the previous slide:

$$h(master\_secret \parallel pad\_2 \parallel \\ h(handshake\_messages \parallel sender \parallel master\_secret \parallel pad\_1))$$

where:

- h is MD5 or SHA-1;
- *handshake_messages* is all of the data from all handshake messages up to but not including this message;
- *sender* is $0x434c4e54$ for client, and $0x53525652$ for server.

SSL Record Protocol



Received data are processed in reverse order.

Computing the MAC and encrypting

The MAC is computed as follows:

$$h(K \parallel pad\_2 \parallel \\ h(K \parallel pad\_1 \parallel seq\_number \parallel type \parallel length \parallel fragment))$$

where:

- h is MD5 sau SHA-1;
- K is the MAC key;
- *seq_number* is the sequence number for this message.

Encryption:

- If a stream cipher is used (RC4) then no padding or IV are needed;
- If a block cipher is used (RC2, DES, 3DES, IDEA, Skipjack), then a padding is needed, as well as an IV for the CBC mode.

SSL Change Cipher Spec Protocol

- The SSL change cipher spec message consists of a single one byte message with the value 1;
- The purpose of the message is to cause the pending state to be copied into the current state, which updates the cipher suite to be used on this connection.

SSL Alert Protocol

- Alert messages convey the severity of the message and the description of the alert;
- Alert level of fatal leads to immediate termination of the connection;
- Closure alerts : notify the recipient that the sender will not send any more messages on this connection;
- Error alerts : unexpected_message, bad_record_mac, decompression_failure, handshake_failure etc.

SSL Application Data Protocol

- Allows the communicating peers to exchange application data;
- Takes application data and feeds it into the SSL record protocol.

Transport Layer Security (TLS)

Transport Layer Security (TLS)

- When SSL was standardized by the IEFT, it was renamed to TLS;
- TLS has an identical structure to SSL;
- TLS sessions and connections are as in SSL;
- Session and connection states in TLS are basically the same;
- Major change: generation of master secret (new PRF);
- Major change: use of HMAC;
- Major change: use of SHA-2;
- Major change: explicit use of IV in the CBC mode;
- Major change: new cipher suites (such as AES-based).

From SSL 3.0 to TLS 1.0

Some differences between SSL 3.0 and TLS 1.0 (Allen and Dierks (1999)):

- TLS 1.0 very close to and backward-compatible with SSL 3.0;
- Cipher suites: removes 3 suites based on FORTEZZA, and any TLS compliant application MUST implement the suite

  TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA

- Generation of master secret is based on a new PRF (it will be described later);
- MAC construction mechanism modified into an HMAC (the MAC in SSL 3.0 is similar to the HMAC construction but it is not exactly the same).

From TLS 1.0 to TLS 1.1

Some differences between TLS 1.0 and TLS 1.1 (Dierks and Rescorla (2006)):

- The implicit Initialization Vector (IV) is replaced with an explicit IV to protect against CBC attacks;
- The cipher suites have been changed considerable.

From TLS 1.1 to TLS 1.2

Some differences between TLS 1.1 and TLS 1.2 (Rescorla and Dierks (2008)):

- SHA-256 is the default digest method (the combined use of MD5 and SHA-1 has been removed);
- Several new cipher suites use SHA-256;
- A new, simpler but more secure, PRF;
- TLS_RSA_WITH_AES_128_CBC_SHA is now the mandatory to implement cipher suite;
- Added HMAC-SHA256 cipher suites;
- Removed IDEA and DES cipher suites, they are now deprecated;
- Support for the SSL 2.0 backward-compatible is now optional only.

TLS PRF 1.2

$$
\begin{aligned}
P\_hash(secret, seed) \quad = \quad & HMAC\_hash(secret, A(1) \parallel seed) \parallel \\
& HMAC\_hash(secret, A(2) \parallel seed) \parallel \\
& HMAC\_hash(secret, A(3) \parallel seed) \parallel \\
& \ldots
\end{aligned}
$$

where

$$
\begin{aligned}
A(0) \quad &= \quad seed \\
A(i) \quad &= \quad HMAC\_hash(secret, A(i-1)) \quad \forall i > 0
\end{aligned}
$$

Then,

- $PRF(secret, label, seed) = P\_hash(secret, label \parallel seed)$
- $master\_secret = PRF(pre\_master\_secret, \text{“}master\ secret\text{''}, N_C \parallel N_S)$
- $key\_block = PRF(master\_secret, \text{“}key\ expansion\text{''}, N_C \parallel N_S)$

From TLS 1.2 to TLS 1.3

Some differences between TLS 1.2 and TLS 1.3 (Rescorla (2018)):

- Removes obsolete algorithms and ciphers: RC4 stream cipher, RSA key transport, SHA-1 hash function, CBC mode ciphers, MD5 algorithm, various Diffie-Hellman groups, EXPORT-strength ciphers, DES, 3DES;
- Introduces a brand new handshake:
  - Removes the RSA method and keeps the Ephemeral Diffie-Hellman method;
  - This reduces the time it takes to encrypt a connection;
  - TLS 1.2 requires two round-trips to complete the TLS handshake, but TLS 1.3 needs only one round-trip;
  - As a result, it cuts down the encryption time to half.

From TLS 1.2 to TLS 1.3

More differences between TLS 1.2 and TLS 1.3:

- Forward secrecy: protect the secrecy of past sessions so that a session stays secret going forward:
  - TLS 1.2 – an adversary who discovers a server's private key could use it to decrypt earlier network traffic;
  - TLS 1.3 – uses the Ephemeral Diffie-Hellman key exchange protocol, which generates a one-time key that is used only for the current session. At the end of the session, the key is discarded;
- New feature that cuts down the encryption time: Zero Round Trip Time Resumption (0-RTT). When a user re-visits a site in a short time, 0-RTT makes the connection almost instantaneous.

SSL and TLS applications

SSL and TLS applications

- HTTPS (HTTP + SSL/TLS + TCP);
- FTPS (FTP + TLS);
- Software update programs: in more recent versions of Windows, Windows Update is a custom app secured by TLS. Many other online software update programs, such as the getPlus program used by Adobe, use TLS connections for security;
- Client security with TLS/SSL: client side certificates can be used with TLS to prove the identity of the client to the server, and vice-versa. This is called "two-way TLS" and requires the client and server both provide certificates to each other;
- Server-to-Server security with TLS: many server-to-server connections offer TLS as an option;
- Google Chrome included TLS 1.3 since Oct 2018, and Google Cloud since June 2020.

Security of TLS

Security of TLS

Aim: study security of TLS considering the unmodified handshake in a model without idealized assumptions (i.e., random oracle):

- Security analysis of the handshake protocol in TLS 1.3 (Dowling et al. (2021));
- Security of TLS-DHE in the standard model (Jager et al. (2012));
- Security of TLS-DH and TLS-RSA in the standard model (Kohlar et al. (2013));
- Security of TLS renegotiation (Giesen et al. (2013)).

References

Allen, C. and Dierks, T. (1999). The TLS Protocol Version 1.0. RFC 2246.

Dierks, T. and Rescorla, E. (2006). The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346.

Dowling, B., Fischlin, M., Günther, F., and Stebila, D. (2021). A cryptographic analysis of the TLS 1.3 handshake protocol. *Journal of Cryptology*, 34.

Freier, A. O., Karlton, P., and Kocher, P. C. (2011). The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101.

Giesen, F., Kohlar, F., and Stebila, D. (2013). On the security of tls renegotiation. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, CCS'13, page 387–398, New York, NY, USA. Association for Computing Machinery.

Jager, T., Kohlar, F., Schäge, S., and Schwenk, J. (2012). On the security of TLS-DHE in the standard model. In Safavi-Naini, R. and Canetti, R., editors, *Advances in Cryptology – CRYPTO 2012*, pages 273–293, Berlin, Heidelberg. Springer Berlin Heidelberg.

Kohlar, F., Schäge, S., and Schwenk, J. (2013). On the security of TLS-DH and TLS-RSA in the standard model. *IACR Cryptol. ePrint Arch.*, 2013:367.

Rescorla, E. (2018). The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446.

Rescorla, E. and Dierks, T. (2008). The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246.