isec
INFORMATION SECURITY
lectures by Ferucio Laurentiu Tiplea

Web Security

Secure Socket Layer and Transport Layer Security

Prof.dr. Ferucio Laurențiu Țiplea
Fall 2022

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

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

Outline

Secure Socket Layer (SSL)

Secure Sockets Layer (SSL)

SSL, developed by Netscape Communications, 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 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

Version 3.0 of SSL was produced by Paul Kocher, working with Netscape engineers, when Taher ElGamal was Chief Scientist at Netscape Communications:

> *"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

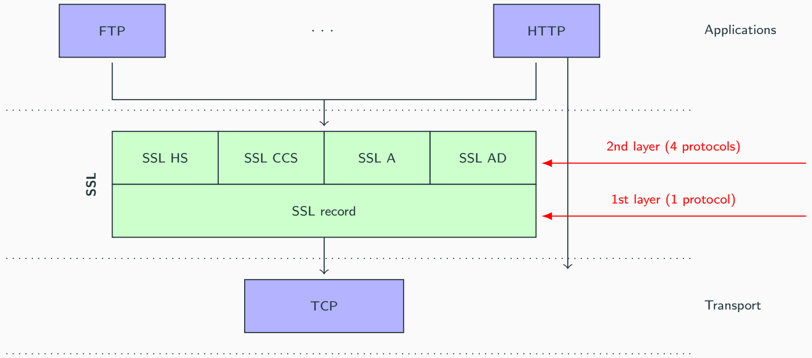
Figure 1: Paul Kocher (left) and Taher ElGamal (right) awarded the 2019 Marconi prize for their contributions to the security of communications.

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:

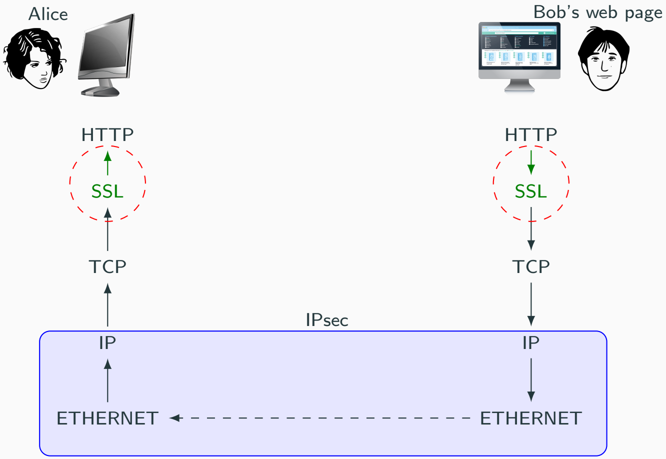
1. Primary goal: to provide a private channel between communicating applications to ensure:
    - Privacy of data;
    - Authentication of the parties;
    - Integrity;
2. SSL does not provide non-repudiation;
3. SSL is socket-oriented: all or none of the data that is sent or received from a socket are 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



Transport Layer Security (TLS)

Transport Layer Security (TLS)

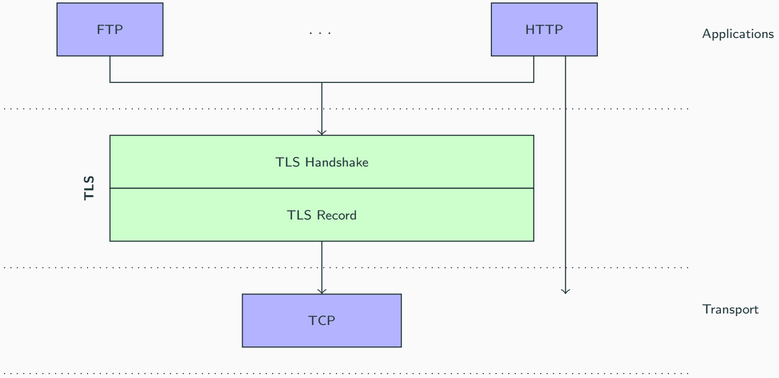
- 1996: The IETF Transport Layer Security (TLS) working group was created to unify and standardize the SSL 3.0 protocol and similar variants such as PCT/STLP. The final result would be called TLS;
- TLS Standards:
  - TLS 1.0 – 1999 (RFC 2246);
  - TLS 1.1 – 2006 (RFC 4346);
  - TLS 1.2 – 2008 (RFC 5246);
  - TLS 1.3 – 2018 (RFC 8446);
- TLS has a similar structure to SSL but added significant changes from standard to standard:
  - Removed broken features (e.g., compression);
  - Improved the crypto architecture of the protocol: HMAC, AEAD, HKDF, etc.;
  - Improved latency (e.g., handshake in 1-RTT or even 0-RTT);
  - Improved security and privacy (e.g., forward secrecy).

TLS 1.3

- TLS 1.3 – work started on April 2014 (draft-00) and completed on Aug 10, 2018 (draft-29);
- Already in Firefox, Chrome, Safari, Opera, Cloudflare, Google, Facebook, OpenSSL,etc.;
- Vibrant research topic: more than 40 papers have been dedicated to the understanding, analysis, and implementation of this standard:
  - HMAC-based key derivation functions;
  - SIGMA = SIGma + MAc;
  - Symbolic verification in Tamarin;
  - Computational verification.

TLS in the TCP/IP protocol stack

Major components of TLS 1.3:



Sessions and connections

- Session
  - Association between two communicating peers;
  - Defines a set of cryptographic parameters which can be shared among multiple connections;
  - Created by the handshake protocol;
  - Primarily used to avoid expensive negotiation of new security parameters for each connection;
- 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.

TLS handshake protocol

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

- Agree on a protocol version;
- Negotiate cryptographic algorithms;
- Authenticate each other;
- Establish shared secret keying material.

Once the handshake is complete, the peers use keying material to derive the application data traffic keys.

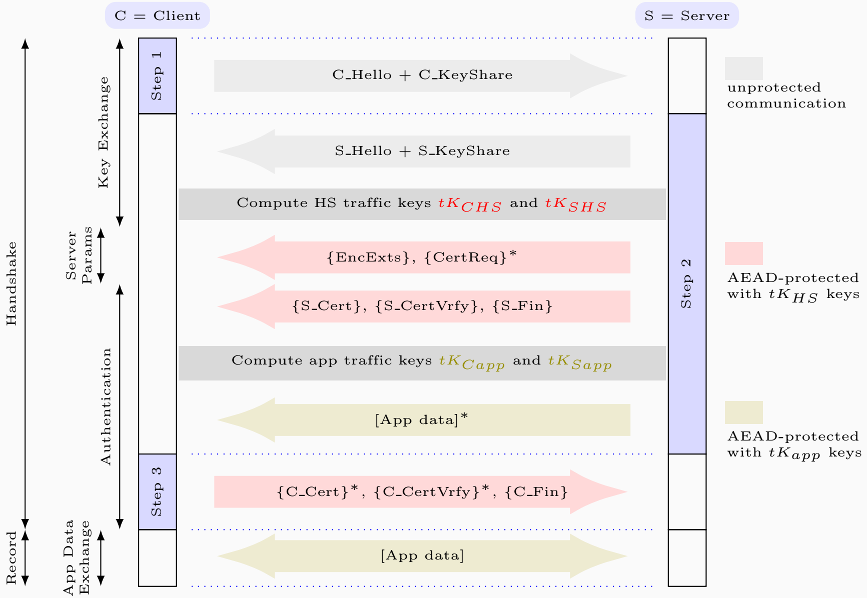
TLS handshake protocol

TLS supports two basic key exchange modes, one of which has two variants:

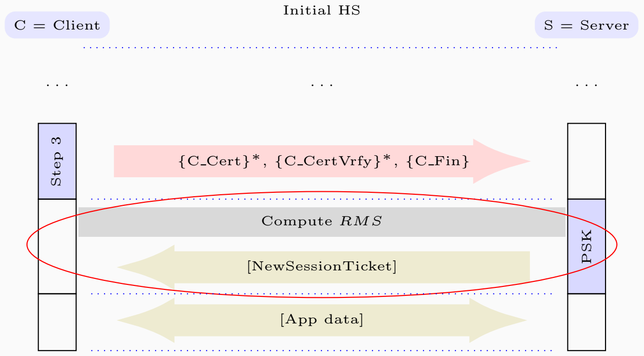
1. (EC)DHE mode. This is the standard handshake mode that does not use pre-shared secrets;
2. Pre-shared key (PSK) based mode. This is based on pre-shared keys. It may come in two variants:
   2.1 PSK-only, which does not provide forward secrecy;
   2.2 PSK with (EC)DHE, which provides forward secrecy.

When clients and servers share a PSK (either obtained externally or via a previous handshake), TLS 1.3 allows clients to send data on the first flight (early data). The client uses the PSK to authenticate the server and to encrypt the early data.

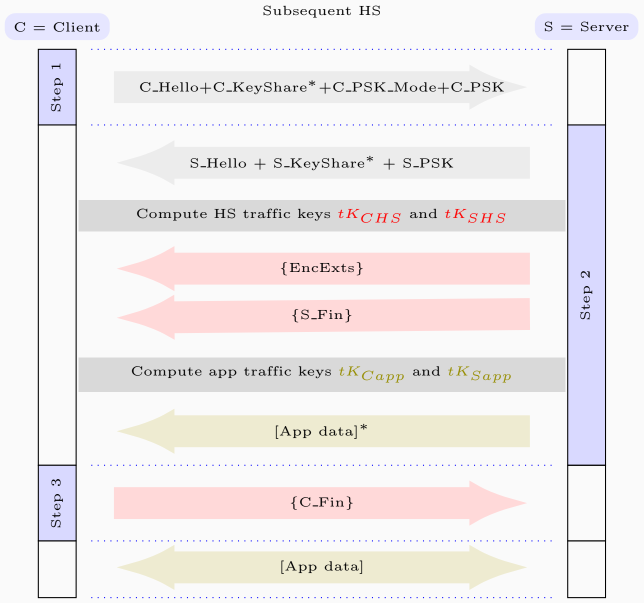
Basic full handshake


C = Client
S = Server
Step 1
C_Hello + C_KeyShare
S_Hello + S_KeyShare
Compute HS traffic keys tK_{CHS} and tK_{SHS}
{EncExts}, {CertReq}*
{S_Cert}, {S_CertVrfy}, {S_Fin}
Compute app traffic keys tK_{Capp} and tK_{Sapp}
[App data]*
Step 2
Step 3
{C_Cert}*, {C_CertVrfy}*, {C_Fin}
[App data]
Handshake
Record
Key Exchange
Server Params
Authentication
App Data Exchange
unprotected communication
AEAD-protected with tK_{HS} keys
AEAD-protected with tK_{app} keys

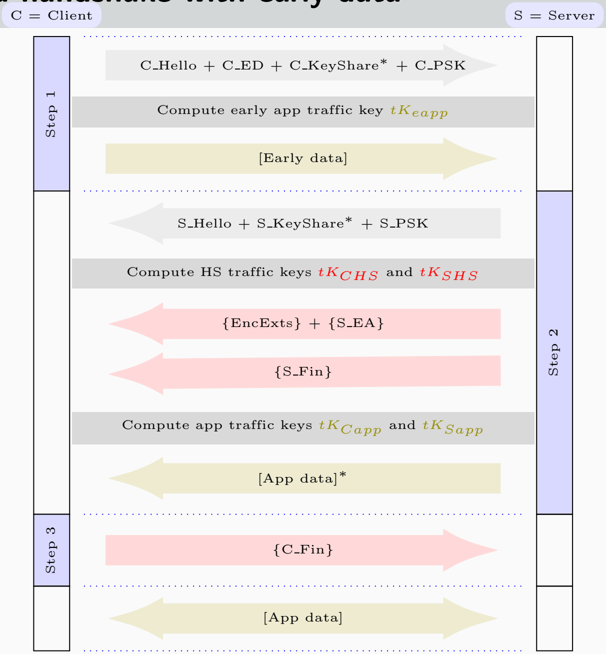
Pre-shared keys



PSK-based handshake


Subsequent HS
C = Client
S = Server
Step 1
C_Hello+C_KeyShare*+C_PSK_Mode+C_PSK
Step 2
S_Hello + S_KeyShare* + S_PSK
Compute HS traffic keys tK_{CHS} and tK_{SHS}
{EncExts}
{S_Fin}
Compute app traffic keys tK_{Capp} and tK_{Sapp}
[App data]*
Step 3
{C_Fin}
[App data]

PSK-based handshake with early data


C = Client
S = Server
Step 1
C_Hello + C_ED + C_KeyShare* + C_PSK
Compute early app traffic key tK_{eapp}
[Early data]
S_Hello + S_KeyShare* + S_PSK
Step 2
Compute HS traffic keys tK_{CHS} and tK_{SHS}
{EncExts} + {S_EA}
{S_Fin}
Compute app traffic keys tK_{Capp} and tK_{Sapp}
[App data]*
Step 3
{C_Fin}
[App data]

HMAC-based key derivation

TLS 1.3 uses a HMAC-based key derivation function (HKDF) (RFC 5869) that, starting with an initial keying material (IKM) generates an output keying material (OKM) following the two steps extract-and-expand paradigm:

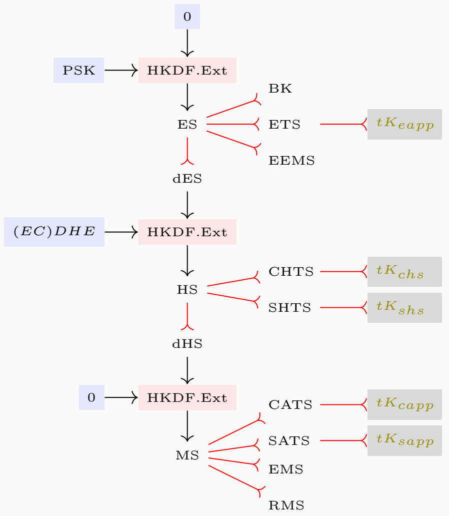
1. In the first stage, it takes IKM and "extracts" from it a fixed-length pseudorandom key K. Formally,

$$K \leftarrow HKDF.Ext(salt, IKM)$$

   (salt = practice jargon for "random non-secret quantity");
2. In the second stage, based on some optional context and application-specific information, "expands" the key K into OKM of some length L

$$OKM \leftarrow HKDF.Exp(K, info, L)$$

Key derivation flow



Note: The red "arrows" means application of the HKDF.Exp function. For instance, ETS=HKDF.Exp(ES,X,Y), for some X and Y

ES = Early Secret
BK = Binder Key
ETS = Early Traffic Secret
tK_{eapp} = Early App Traffc Key
EEMS = Early Exporter Master Secret
dES = Derived Early Secret

HS = Handshake Secret
CHTS = Client Handshake Traffic Secret
tK_{chs} = Client Handshake Traffic Key
SHTS = Server Handshake Traffic Secret
tK_{shs} = Server Handshake Traffic Key
dHS = Derived Handshake Secret

MS = Master Secret
CATS = Client App Traffic Secret
tK_{capp} = Client App Traffic Key
SATS = Server App Traffic Secret
tK_{sapp} = Server App Traffic Key
EMS = Exporter Master Secret
RMS = Resumption Master Secret

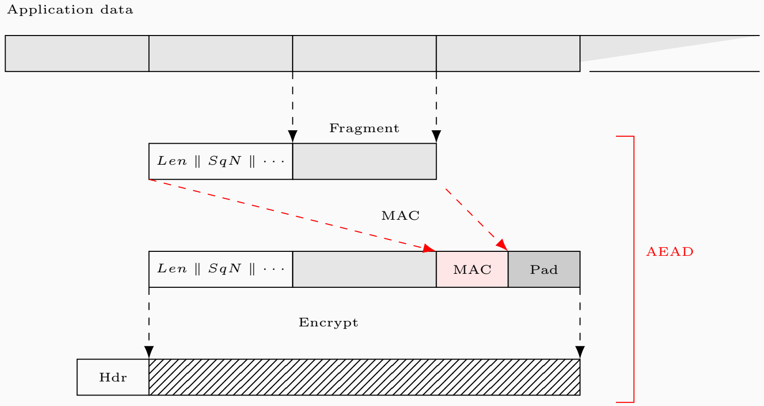
Handshake properties (1)

1. Establishing the same session keys on both sides of the handshake, provided that it completes successfully on each endpoint;
2. Secrecy of the session keys: The shared session keys should be known only to the communicating parties and not to the attacker;
3. Peer authentication: Server side of the channel is always authenticated; the client side is optionally authenticated;
4. Uniqueness of the session keys: Any two distinct handshakes should produce distinct, unrelated session keys. Individual session keys produced by a handshake should also be distinct and independent;
5. Downgrade protection: The cryptographic parameters should be the same on both sides and should be the same as if the peers had been communicating in the absence of an attack;

Handshake properties (2)

6. Forward secret with respect to long-term keys: protect the secrecy of past sessions so that a session stays secret going forward:;
7. Key compromise impersonation resistance: In a mutually authenticated connection with certificates, compromising the long-term secret of one actor should not break that actor's authentication of their peer in the given connection. For example, if a client's signature key is compromised, it should not be possible to impersonate arbitrary servers to that client in subsequent handshakes;
8. Protection of endpoint identities: The server's identity (certificate) should be protected against passive attackers. The client's identity should be protected against both passive and active attackers.

The record protocol



Application data
Fragment
$Len \parallel SqN \parallel \cdots$
MAC
$Len \parallel SqN \parallel \cdots$
MAC
Pad
Encrypt
Hdr
AEAD

Alert messages

- Alert messages convey closure information and errors;
- Alert messages are encrypted in accordance with the current connection state;
- In contrast to the previous versions, the severity of the alert in TLS 1.3 is implicit in the type of alert being sent;
- There are two types of alerts:
  - Closure alerts:
    - close_notify: notifies the recipient that the sender will not send any more messages on this connection;
    - user_canceled: notifies the recipient that the sender is canceling the handshake for some reason unrelated to a protocol failure;
  - Error alerts indicate abortive closure of the connection. Examples: unexpected_message, bad_record_mac, handshake_failure, etc.

Applications

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: 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;
- TLS based VPN.

Security of TLS

Security of TLS

1. The study of TLS 1.3 protocol security is a hot research topic with paramount practical importance;
2. Many research articles and doctoral theses have been written in this direction;
3. Special sessions at conferences and workshops focused on cryptography and information security have been dedicated to TLS 1.3;
4. The two primary directions are the study of TLS 1.3 security in the:
   - 4.1 Symbolic model: Horvat (2015); Cremers et al. (2017); Hoyland (2018); van der Merwe (2018); Scott (2018);
   - 4.2 Computational model: Jager et al. (2012); Giesen et al. (2013); Kohlar et al. (2013); Dowling et al. (2021); Aviram et al. (2021); Bhargavan et al. (2022); Davis et al. (2022).

References

Aviram, N., Gellert, K., and Jager, T. (2021). Session resumption protocols and efficient forward security for TLS 1.3 0-RTT. *Journal of Cryptology*, 34.

Bhargavan, K., Cheval, V., and Wood, C. (2022). Handshake privacy for TLS 1.3. Research report, Inria Paris; Cloudflare.

Cremers, C., Horvat, M., Hoyland, J., Scott, S., and van der Merwe, T. (2017). A comprehensive symbolic analysis of TLS 1.3. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1773–1788.

Davis, H., Diemert, D., Günther, F., and Jager, T. (2022). On the concrete security of TLS 1.3 PSK mode. In Dunkelman, O. and Dziembowski, S., editors, *Advances in Cryptology – EUROCRYPT 2022*, pages 876–906, Cham. Springer International Publishing.

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.

Horvat, M. (2015). Formal analysis of modern security protocols in current standards, *p*h.d. thesis.

Hoyland, J. (2018). An analysis of TLS 1.3 and its use in composite protocols, *p*h.d. thesis.

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.

Scott, S. (2018). The design and analysis of real-world cryptographic protocols, *p*h.d. thesis.

van der Merwe, T. (2018). An analysis of the Transport Layer Security protocol, *p*h.d. thesis.