

SSL and TLS

Kjell Jørgen Hole
NoWires Research Group
Department of Informatics
University of Bergen

last changed October 27, 2011

KJhole.com

Outline

- Introduction to
 - SSL (Secure Sockets Layer) protocol
 - TLS (Transport Layer Security) protocol
- Operation
- Details

Introduction

KJhole.com

SSL history

- 1994: Netscape completes SSL 1.0 design
- 1995: Netscape Navigator with SSL 2.0 ships
- 1996: SSL 3.0 published

TLS History

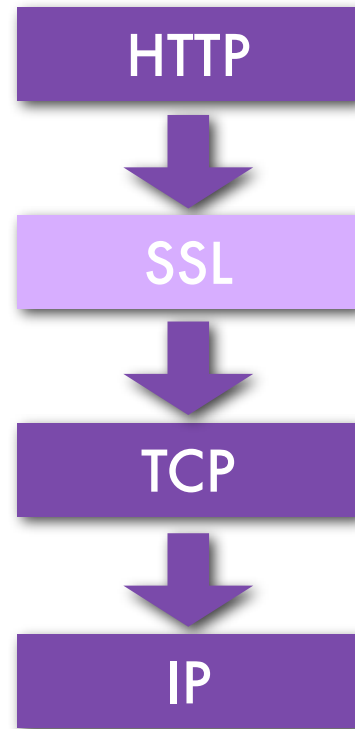
- 1996: IETF (Internet Engineering Task Force) renames SSL to **TLS** (Transport Layer Security)
- 1999: TLS 1.0 published (also denoted SSL 3.1)
- 2006: TLS 1.1 published (SSL 3.2)
- 2008: TLS 1.2 published (SSL 3.3)

Provided services

- Two-way authentication using public-key certificates
- Confidentiality (privacy)
- Data integrity

Separate security protocol

- The original designers of SSL decided to create a separate protocol layer
- The introduction of a separate protocol allows SSL to support applications other than HTTP



7

Protocol limitations

- TLS must be implemented on top of a reliable transport protocol such as TCP
- TLS does **not** provide non-repudiation
- A PKI is needed to generate and distribute certificates

8

TLS operation

TLS for client-server apps KJhole.com

- TLS is a protocol for client-server applications:
 1. the client initiates the communication and suggests security parameters
 2. the server selects from the client's options

TLS functionality



- Two-way authentication:
 1. client and server exchange certificates
 2. each party verifies that the other party has the private key corresponding to the public key in the certificate
- Message integrity and encryption

There also exists a TLS mode where only the server has a public-key certificate

11

12 steps (1)



- **Step 1.** Client sends **ClientHello** message proposing TLS options
- **Step 2.** Server responds with **ServerHello** selecting the TLS options
- **Step 3.** Server sends its public-key certificate in **Certificate** message

12

12 steps (2)



- **Step 4.** Server sends a `CertificateRequest` message to indicate that it wants to authenticate the client
- **Step 5.** Server concludes its part of the negotiation with `ServerHelloDone` message
- **Step 6.** Client sends its public-key certificate in a `Certificate` message

13

12 steps (3)

- **Step 7.** Client sends session key information (encrypted with the server's public key) in a `ClientKeyExchange` message
- **Step 8.** Client sends a `CertificateVerify` message, which signs important information about the session using the client's private key; the server uses the public key from the client's certificate to verify the client's identity

14

12 steps (4)



- **Step 9.** Client sends **ChangeCipherSpec** message to activate the negotiated options for all future messages it will send
- **Step 10.** Client sends a **Finished** message to let the server check the newly activated options

15

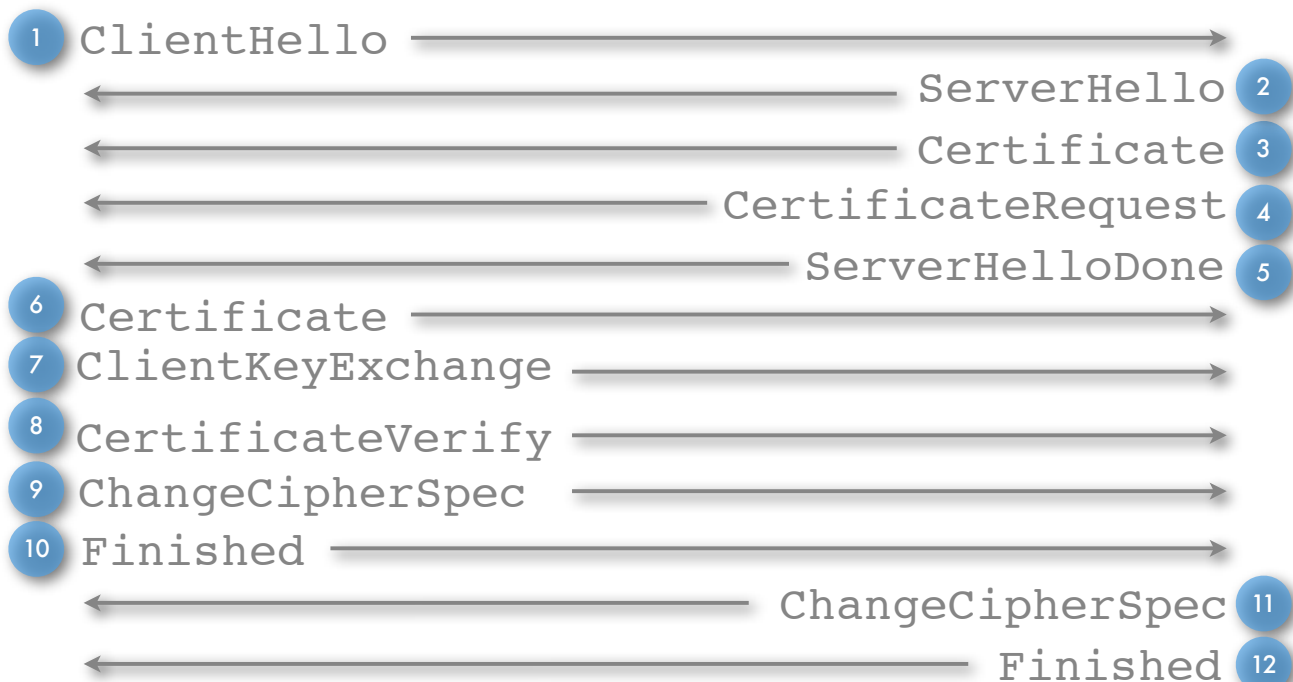
12 steps (5)



- **Step 11.** Server sends a **ChangeCipherSpec** message to activate the negotiated options for all future messages it will send
- **Step 12.** Server sends a **Finished** message to let the client check the newly activated options

16

TLS messages



17

1. ClientHello

- The **ClientHello** message starts the TLS communication

Field	Use
Version	highest version of the TLS protocol that the client can support (≤ 3.3)
RandomNumber	28-byte random number used to seed the cryptographic calculations
SessionID	identifies a specific TLS session (empty in this example)
CipherSuites	a list of cryptographic algorithms and key sizes that the client can support
CompressionMethods	data compression methods that the client can support

18

Version field

- Assume that a client sends a version 3.3 `ClientHello` to a server that only supports version 3.1 of TLS
- The server may respond with version 3.1 messages that it expects the client to understand
- The client can decide to continue with TLS version 3.1, or it can abandon the attempt

RandomNumber field

- The 28 bytes must be a cryptographically secure random number
- The TLS specification suggests that 4 bytes consisting of the time and date are added to the 28 random bytes to ensure that a random number is never used twice

2. ServerHello

- When the server receives a `ClientHello` message, responds with a `ServerHello`

Field	Use
Version	identifies the version of the TLS protocol to be used for this communication
RandomNumber	28-byte random number used to seed the cryptographic calculations
SessionID	identifies a specific TLS session, contains a value if possible to reuse session
CipherSuite	the cryptographic algorithms and key sizes to be used for this communication
CompressionMethod	the single data compression method to be used for this communication

21

CipherSuite field

- The server must choose a single cipher suite and compression method from among those listed by the client in its `ClientHello` message

22

3,6 Certificate

- The **Certificate** message from the server (client) contains a certificate chain that begins with the server's (client's) public-key certificate and ends with the certificate authority's root certificate
- The client (server) must verify the signature, validity, and revocation status
- It is also important to verify that the certificate really identifies the correct party

23

4. CertificateRequest

- The **CertificateRequest** message from the server asks the client to provide a public-key certificate
- The server will only send this message if it expects the client to have a certificate

24

CertificateRequest

- The request lists various types of certificates that the server will accept
- the certificates are differentiated by the signature algorithm used

Field	Use
CertificateTypes	a list of certificate types acceptable to the server
DistinguishedNames	a list of distinguished names of CAs acceptable to the server

5. ServerHelloDone

- The message tells the client that the server has finished its initial negotiation messages
- the message itself contains no other information

7. ClientKeyExchange

- The `ClientKeyExchange` message provides the server with key information from the client
- The key information is for the symmetric encryption that both parties will use during the session
- The information is encrypted with the public key in the server's certificate

27

ClientKeyExchange

- Allows the client to verify that the server has the private key corresponding to the public key in the certificate
 - if this is not the case, then the server will not be able to obtain the key needed for the symmetric encryption
- Note that encryption with the server's public key protects against a fake server

28

8. CertificateVerify

- The client uses the `CertificateVerify` message to prove that it possesses the private key corresponding to the public key in the certificate
- message contains a digitally signed cryptographic hash of information available to both client and server
- The `CertificateVerify` message relies on cryptographic values in `ClientKeyExchange`

29

9,11 ChangeCipherSpec

- Message indicates that the security services should be invoked
 - symmetric encryption algorithm
 - message integrity algorithm
 - specific key material for algorithms
 - different for each communication direction

10,12 Finished

- The **Finished** message allows both the client and the server to verify that the negotiation has been successful and that security has not been compromised
- each message contains a cryptographic hash of the negotiated information
- if the receiving party cannot successfully decrypt and verify this message, then something went wrong

31

Ending communication

- Each party sends a **ClosureAlert** message to the other party
- Important that this procedure is used to avoid truncation attacks

Client**Server**

32

Two more messages

- In addition to the messages introduced previously, there exist two other messages:
 - **Alert**, informs the other party of possible security breach or communication failure
 - **ApplicationData**, information that the client and server exchange, which is encrypted, authenticated, and verified by TLS

TLS details

TCP

- TLS depends on a lower-level reliable transport protocol, e.g., TCP
- TCP will typically combine several TLS messages into a single TCP segment

MAC

- TLS uses a keyed Message Authentication Code (MAC) to protect message integrity:
 - HMAC using the Secure Hash Algorithm (SHA-256) hash function (256 output bits)
- MAC is appended to message
- MAC value is encrypted

Secret material (1)

- The security of TLS depends on generating secret (key) material known only to the communicating client and server
- A Pseudo-Random Number Generator (PRNG) is used to generate this material
 - the PRNG must be cryptographically secure

37

Secret material (2)

- The TLS client utilizes the PRNG to generate the 28-byte random number, or **nonce**, in the `ClientHello` message
 - nonce is sent in the clear to the server
- The TLS client also uses the PRNG to generate a 48-byte **premaster secret** in the `ClientKey Exchange` message
 - premaster secret is sent encrypted to server

38

Secret material (3)

- The SSL server utilizes the PRNG to generate the 28-byte random number, or **nonce**, in the **ServerHello** message
 - nonce is sent in the clear to the client
- Client and server use the two nonces and premaster secret to generate a **master secret**
- The master secret is the source of all secret (key) material

39

Final remarks

- TLS can provide strong authentication and communication security over the Internet
- The protocol is in widespread use in web browsing, email, and VoIP
- Claim: the usability of TLS is inadequate because users do not understand the function of certificates

40

RFC 5246

- T. Dierks and E. Rescorla, [The TLS Protocol Version 1.2](http://tools.ietf.org/html/rfc5246), Aug. 2008;
<http://tools.ietf.org/html/rfc5246>