Lecture 17: Transport Layer Security Protocol Part 1

COSC362 Data and Network Security

Book 1: Chapter 17 – Book 2: Chapter 22

Spring Semester, 2021

Motivation

- TLS is the most widely used security protocol.
- TLS is used to secure communications with banks, online shops, email providers, etc.
- TLS uses most of the mainstream cryptographic algorithms.
- TLS is a very complex protocol.
- TLS has been subject of many attacks, and subsequent repairs.

Outline

Outline

History and Overview

TLS Record Protocol

TLS Handshake Protocol

History

- 1994: Netscape Communications developed Secure Sockets Layer (SSL) 2.0:
  - It should no longer be used!
- 1995: Netscape released SSL 3.0:
  - It should no longer be used!
- 1999: RFC 2246 issued by IETF, documenting Transport Layer Security (TLS) 1.0, similar to SSL 3.0:
  - It should no longer be used!
- 2006: RFC 4346 documenting TLS 1.1:
  - Fixing problems with non-random IVs and exploitation of padding error messages.
- 2008: RFC 5246 documenting TLS 1.2:
  - Allowing the use of standard authenticated encryption rather than separating encryption and MAC.
- 2018: RFC 8446 documenting TLS 1.3:
  - Separating key agreement and authentication algorithms for cipher suites.

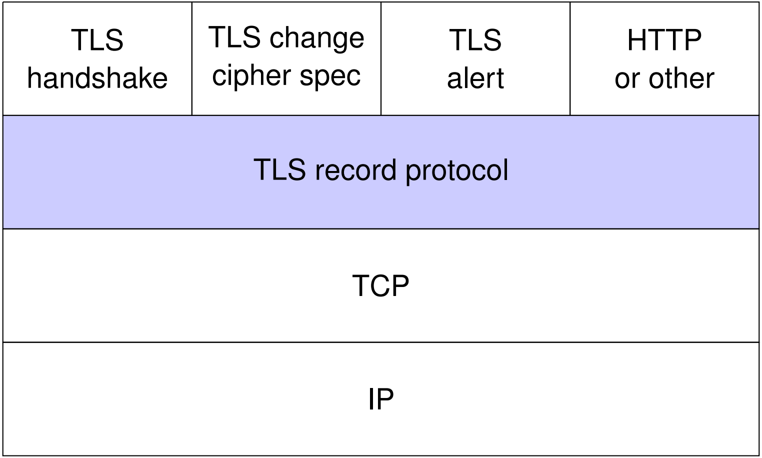
Applications

- Cryptographic services protocol based upon PKI and commonly used on the Internet.
- Often used to allow browsers to establish secure sessions with Web servers.
- Many other application areas.
- TLS runs primarily over TCP:
  - Variant DTLS runs over datagram protocols.

Architecture Overview

- Designed to secure reliable end-to-end services over TCP.
- 3 higher level protocols:
  - *TLS handshake protocol* to set up sessions.
  - *TLS alert protocol* to signal events, such as failures.
  - *TLS change cipher spec protocol* to change the cryptographic algorithms.
- TLS record protocol provides basic services to various higher level protocols.

Protocol Stack

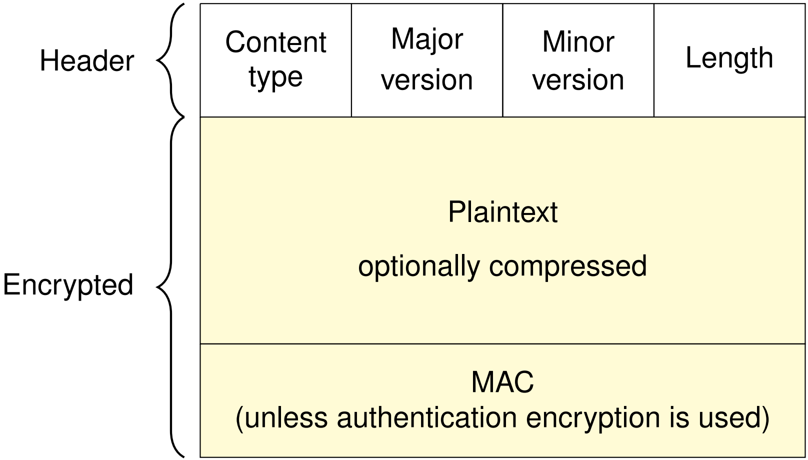


Outline

Overview

- TLS connection services:
  - Message confidentiality: ensuring that the message contents cannot be read in transit.
  - Message integrity: ensuring that the receiver can detect if a message is modified in transmission.
- Services possibly provided by a symmetric encryption algorithm and a MAC.
- From TLS 1.2, services provided with authenticated encryption modes (CCM, GCM).
- Handshake protocol establishes symmetric session keys to use with these mechanisms.

Format


Header
Encrypted
Content type
Major version
Minor version
Length
Plaintext
optionally compressed
MAC
(unless authentication encryption is used)

Header

- Content type:
  - change-cipher-spec
  - alert
  - handshake
  - application-data
- Protocol version:
  - Major version: 3 for TLS
  - Minor version:
    - 1 for TLS 1.0
    - 2 for TLS 1.1
    - 3 for TLS 1.2
    - 4 for TLS 1.3
- Length: of the data, in octets.

Operation

- Fragmentation: each application layer message is fragmented into blocks of 2^{14} bytes or less.
- Compression:
  - Default compression algorithm is null in TLS 1.2 (thus optionally applied).
  - Removed in TLS 1.3.
- Authenticated data: consisting of the (compressed) data, header and an implicit record sequence number.
- Plaintext: compressed data and MAC (if present).
- Session keys: computed during handshake protocol, for either MAC and encryption algorithms, or authenticated encryption algorithm.
- Specification: encryption and MAC algorithms are specified in the negotiated cipher suite.

MAC Algorithm

- HMAC in all TLS versions using a negotiated hash function.
- SHA-2 allowed only from TLS 1.2.
- MD5 and SHA-1 discarded from TLS 1.3.

Encryption Algorithm

- Either a negotiated block cipher in CBC mode or a stream cipher.
- Most common block cipher is AES.
- 3DES and RC4 discarded in TLS 1.3.
- For block ciphers, padding is applied after MAC to make a multiple of the cipher block size.

Authenticated Encryption Algorithm

- Allowed instead of encryption and MAC from TLS 1.2.
- Only AES with either CCM or GCM modes in TLS 1.3.
- Authenticated additional data in the header and implicit record sequence number.

Outline

Purposes

- Negotiating the TLS version and cryptographic algorithms to be used.
- Establishing a shared session key for use in the record protocol.
- Authenticating the server, and optionally authenticating the client.
- Completing the session establishment.
- Variations with:
  - RSA
  - Diffie-Hellman
  - Pre-shared keys
  - Mutual authentication
  - Server-only (unilateral) authentication
- Simplified in TLS 1.3 (see later).

Phases

- Phase 1: initiating the logical connection and establishing its security capabilities.
- Phases 2 and 3: performing key exchange.
  - Messages and their contents depend on the handshake variant negotiated in Phase 1.
- Phase 4: completing the setting up of a secure connection.

Cipher Suites

- Specifying:
  - Public key algorithms used for key establishment.
  - Symmetric algorithms used for providing authenticated encryption and key computation.
- Over 300 standardised cipher suites:
  - Many are weak.
  - Many have been discarded in TLS 1.3.
- Big change in TLS 1.3:
  - All supported cipher suites must be Authenticated Encryption with Associated Data (AEAD).

Cipher Suite Example

TLS_RSA_WITH_3DES_EDE_CBC_SHA[1]

- Mandatory in TLS 1.0 and 1.1.
- RSA used for key exchange, to encrypt a secret chosen by the client.
- 3DES (Encrypt Decrypt Encrypt) in CBC mode used for encryption, for data confidentiality.
- SHA-1 used for HMAC, for data integrity.

[1] https://ciphersuite.info/cs/TLS_RSA_WITH_3DES_EDE_CBC_SHA/

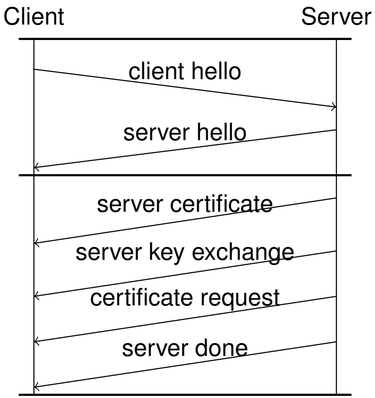
Handshake Algorithms

Algorithm	Description	TLS versions
DHE-DSS	DHE with Digital Signature Standard	1.2
DHE-RSA	Ephemeral Diffie-Hellman with RSA signatures	1.2 and 1.3
ECDHE-RSA	Elliptic curve DHE with RSA signatures	1.2 and 1.3
ECDHE-ECDSA	Elliptic curve DHE with elliptic curve Digital Signature Algorithm	1.2 and 1.3

Record Algorithms

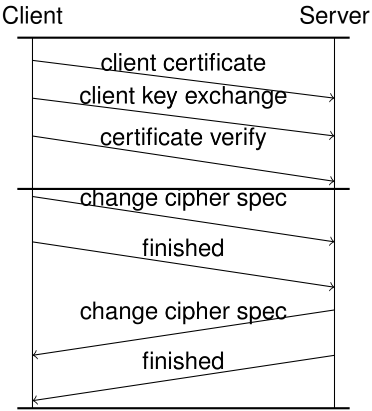
Algorithm	Description	TLS versions
AES-CBC-SHA256	AES in CBC mode with HMAC from SHA256	1.2
AES-GCM	AES with GCM mode	1.2 and 1.3
CHACHA20 -POLY1305	ChaCha stream cipher with Poly1305 MAC	1.2 and 1.3

Phases 1 and 2



- Phase 1: client and server negotiate version, cipher suite and compression, and exchange nonces.
- Phase 2: server sends certificate and key exchange message (if needed).

Phases 3 and 4



- Phase 3: client sends certificate and key exchange message.
- Phase 4: client and server start secure communications. Finished messages include a check value (pseudorandom function) of all the previous messages.

Handshake Messages

- Client hello:
  - Stating the highest TLS version available.
  - Advertising cipher suites available to the client.
  - Sending the client's nonce N_C.
- Server hello:
  - Returning the selected version and cipher suite.
  - Sending the server's nonce N_S.
- Server key exchange: server's inputs to key exchange.
- Client key exchange: client's inputs to key exchange.
- Change cipher suite: switching to newly negotiated cipher suite for record layer.

Ephemeral Diffie-Hellman Handshake Variant

- Server key exchange: inputs are the Diffie-Hellman generator and group parameters, along with the server's ephemeral Diffie-Hellman value, all signed by the server.
- Client key exchange: inputs are client's ephemeral Diffie-Hellman value:
  - Optionally signed by the client if the client's certificate is used.
- Pre-master secret *pms* is the shared Diffie-Hellman secret (from key agreement).

RSA Handshake Variant

- Server key exchange: not required.
- Client key exchange: key transport of pre-master secret *pms*:
  - The client randomly selects the pre-master secret *pms*.
  - The client encrypts *pms* with the server's public key and sends the ciphertext to the server.
  - The server decrypts using its secret key to recover *pms*.

Session Key Generation

- Master secret *ms* defined using the pre-master secret *pms* (and a pseudorandom function):

$$ms = PRF(pms, \text{“master secret”}, N_C \| N_S)$$

- Key material generated as much as required by agreed cipher suite:

$$k = PRF(ms, \text{“key expansion”}, N_S \| N_C)$$

- Independent session keys are partitioned from *k* in each direction:
  - A write key and a read key on each side.
- Depending on the agreed cipher suite, key material includes:
  - Encryption key
  - MAC key
  - IV

Pseudorandom Function

- PRF built from HMAC with a specified hash function.
  - TLS 1.0 and 1.1: based on a combination of MD5 and SHA-1.
  - TLS 1.2: based on SHA-2.
- Example in TLS 1.2:

$$
\begin{aligned}
PRF(key, label, nonce) = \quad & HMAC(key, A(1) \| label \| nonce) \| \\
& HMAC(key, A(2) \| label \| nonce) \| \\
& \ldots
\end{aligned}
$$

where $A(0) = nonce$, $A(i) = HMAC(key, A(i-1))$.
HMAC uses a specified SHA-2 variant (e.g. SHA256) as its hash function.

Other Handshake Variants

- Diffie-Hellman: client and server use static/fixed Diffie-Hellman with certified keys:
  - When the client does not have a certificate (usual on the Internet), she uses an ephemeral Diffie-Hellman key.
- Anonymous Diffie-Hellman: the ephemeral Diffie-Hellman keys are not signed at all:
  - It only protects against passive eavesdropping.

Alert Protocol

- Handling connection by sending an alert message of various degrees of severity.
- Types:
  - Warning alerts
  - `close_notify` alerts
  - Fatal alerts
- Improperly handling alert messages leads to truncation attacks.

Forward Secrecy

- Reminder: compromise of a long-term key should not lead to compromise of session keys established before the long-term key is compromised.
- Diffie-Hellman key exchange achieves forward secrecy:
  - Exchange is authenticated using signatures from the long-term keys.
  - Diffie-Hellman-based cipher suites provide forward secrecy.
- RSA-based handshake does not offer forward secrecy but is currently used in many cipher suites:
  - TLS 1.3 does not allow static RSA.

Summary

- TLS consists of 3 protocols:
  - Handshake protocol
  - Record layer protocol
  - Alert protocol
- New 1.3 version has been rolled out as understanding of cryptography and potential attacks increase.
- TLS assumes reliable message delivery, provided by TCP.

Some time left?

Yes, then we keep on with Lecture 18!