NTNU | Norwegian University of Science and Technology

TRANSPORT LAYER SECURITY

TTM4135 - Lecture 13

Tjerand Silde

19.02.2026

Motivation

- Transport Layer Security (TLS) is a cryptographic protocol designed to provide communication security over a computer network.
- TLS is probably the most widely used security protocol today.
- TLS is used to secure communications with banks, hospitals, online shops, email providers,newspapers, and much more.
- TLS uses most of the mainstream cryptographic algorithms we covered.
- TLS is a very complex protocol and has been the subject of many attacks and subsequent repairs over the years. The newest version is TLS 1.3.

Contents

SSL/TLS History

- 1994: Netscape Communications developed Secure Sockets Layer (SSL) 2.0. This version should no longer be used due to the many flaws.
- 1995: Netscape released SSL 3.0. This version should also not be used.
- RFC 2246 issued in 1999 by IETF document TLS 1.0, similar to SSL 3.0.
- TLS 1.1 was specified in 2006 in RFC 4346. This version fixes issues with non-random IVs and padding error messages, but no major changes.
- TLS 1.2 specified in 2008 in RFC 5246. Allows authenticated encryption.
- The changes are usually vulnerability fixes and additional algorithms.

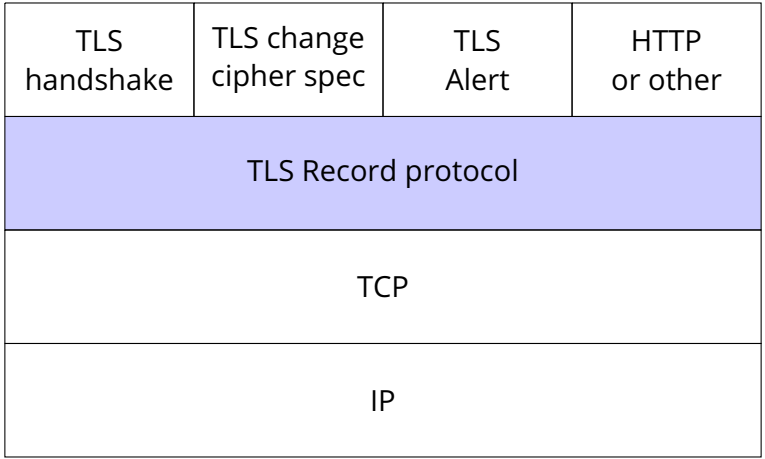
TLS 1.2 vs. TLS 1.3

- TLS 1.3 was standardized in 2018 in RFC 8446 with significant differences from TLS 1.2, removing algorithms and parameter sets as well as updating and changing several sub-protocols to improve efficiency and security.
- TLS 1.2 is still widely supported, so we focus on **TLS 1.2 in this lecture**.
- We focus on TLS 1.3 in the next lecture and explain the main differences.

TLS: Architecture Overview

- TLS is not a single protocol but two layers of protocols (see next slide).
- The TLS higher level consists of three protocols:
  - TLS handshake protocol to set up new communication sessions.
  - TLS alert protocol to signal events such as warnings and failures.
  - TLS change cipher protocol to change the cryptographic algorithms.
- The TLS record protocol provides services to the higher level protocols.

TLS: Protocol Stack



TLS Alert and TLS Change Cipher Spec Protocols

- The Alert protocol handles connections by sending an “alert” message of various degrees of severity. Three types of alerts:
  - warning alerts
  - `close_notify` alerts
  - fatal alerts
- Improper handling of alerts can lead to man-in-the-middle attacks.
- The change cipher spec protocol is normally used after the handshake protocol to indicate commencement of secure traffic going forward.

TLS Ciphersuites

- TLS ciphersuites specify the public key algorithms used in the handshake protocol and the symmetric algorithms used in the record protocol.
- Over the years, over 300 ciphersuites has been standardized, many of which are weak and should no longer be used. Full list is held by IANA.
- Example ciphersuite: TLS_RSA_WITH_3DES_EDE_CBC_SHA
  - key exchange with RSA to encrypt a secret chosen by the client
  - triple DES (Encrypt-Decrypt-Encrypt) in CBC mode for encryption
  - SHA-1 HMAC for data integrity

HMAC construction – Reminder

Let H be any iterated cryptographic hash function. Then define:

$$\mathrm{HMAC}(M, K) = \mathrm{H}\big((K \oplus \mathrm{opad}) \parallel \mathrm{H}((K \oplus \mathrm{ipad}) \parallel M)\big)$$

where

- M: message to be authenticated
- K: key padded with zeros to the block size of H
- opad: `0x5c5c5c...5c`
- ipad: `0x363636...36`
- $\parallel$ denotes concatenation of bit strings
- HMAC is secure (unforgeable) if H is collision resistant or if H is a pseudorandom function.

HMAC – Reminder

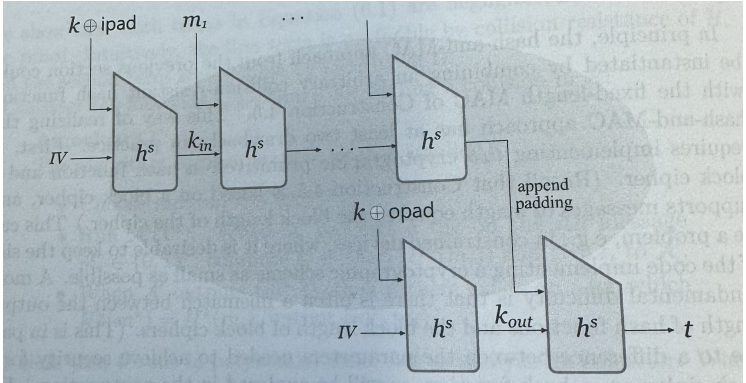


FIGURE 6.2: HMAC, pictorially.

Common TLS 1.2 Ciphersuites

Handshake algorithms (all use signed Diffie–Hellman)

DHE-RSA Ephemeral Diffie-Hellman with RSA signatures

ECDHE-RSA Elliptic curve DHE with RSA signatures

DHE-DSS DHE with DSS signatures

Record algorithms

AES-GCM AES authenticated encryption with GCM mode

AES-CBC-SHA256 AES in CBC mode with HMAC from SHA256

CHACHA20-POLY1305 ChaCha stream cipher with Poly1305 MAC

Contents

Record Protocol Overview

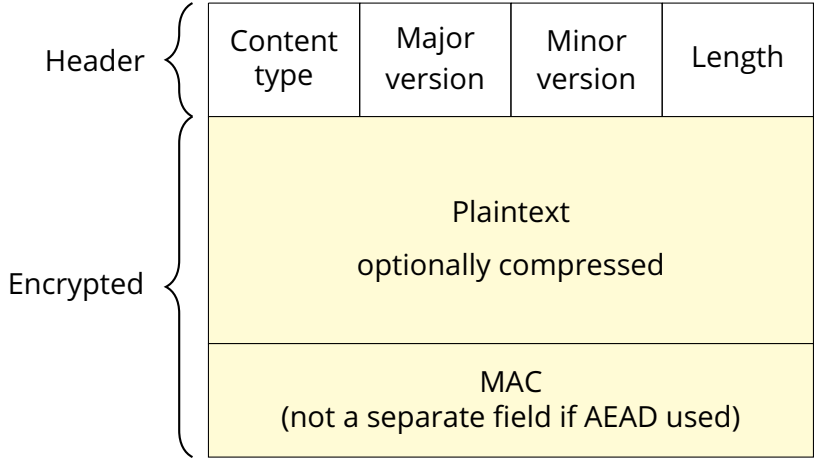
The record protocol provides two services for higher-layer protocols:

Message confidentiality: Ensure that the message contents cannot be read in transit.

Message integrity: Ensure that the receiver can detect if a message is modified in transit.

- These services can be provided by using symmetric encryption and a MAC.
- From TLS 1.2, these services are often provided with authenticated encryption with associated data (AEAD) modes such as CCM or GCM.
- The handshake protocol establishes symmetric session keys for use here.

Record Protocol Format



TLS: Record Protocol Header

- Content Type: Defined content types are
  - change-cipher-spec
  - alert
  - handshake
  - application-data
- Protocol Version
  - Major Version: 3 for TLS
  - Minor Version: 1 for TLS v1.0; 2 for TLS v1.1; 3 for TLS v1.2
- Length: length in bytes of the data

Record Protocol Operation

- Fragmentation: Each application layer message is fragmented into blocks of 16 kilobytes or less.
- Compression: Optionally applied – default compression algorithm is null.
- Authenticated data: consists of the (compressed) data, the header, and an implicit record sequence number.
- Plaintext: Compressed data and the MAC, if present.
- Session keys for the MAC and encryption algorithms, or AEAD algorithm, are established during the handshake protocol.
- The encryption and MAC algorithms are specified in the *ciphersuite*.

Record Protocol Cryptographic Algorithms

MAC: HMAC with negotiated hash function. SHA-2 allowed from TLS 1.2.

Encryption: Either a negotiated block cipher in CBC mode or a stream cipher (AES, 3DES; RC4 originally supported). Padding is applied after the MAC for block ciphers.

AEAD: Allowed instead of encryption+MAC in TLS 1.2 (e.g., AES-CCM or AES-GCM). Additional authenticated data is the header together with an implicit record sequence number.

Contents

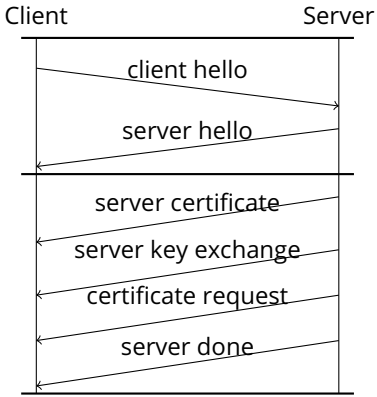
Handshake Protocol Purpose

- Negotiate the TLS version and cryptographic algorithms.
- Establish a shared session key for the record protocol.
- Authenticate the server via certificates and CAs.
- Optionally authenticate the client (this is not common).
- Complete the session establishment to enable communication.
- Variations of the handshake:
  - RSA variant (supported but not recommended)
  - Diffie-Hellman variant (recommended)
  - Pre-shared key variant (from previous sessions)
  - Mutual authentication or server-only authentication

Handshake Protocol - Four Phases

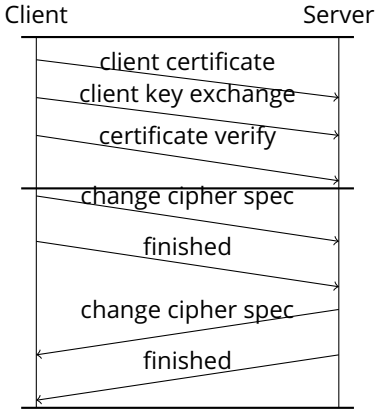
- Phase 1: Initiate the logical connection and establish security capabilities.
- Phases 2 and 3: Perform key exchange; messages depend on the variant that is negotiated in phase 1.
- Phase 4: Complete setup of a secure connection.

Handshake Protocol - Phases 1 and 2



- Phase 1: Client and server negotiate version, ciphersuite and compression, exchange nonces.
- Phase 2: Server sends certificate and key exchange message.

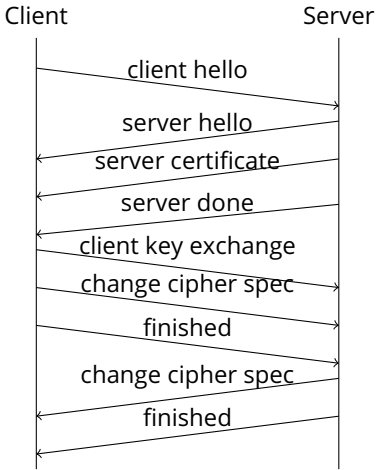
Handshake Protocol - Phases 3 and 4



- Phase 3: Client sends certificate and key exchange message.
- Phase 4: Client and server start secure communications.

Finished messages include a check value (pseudo-random function) of all the previous messages.

RSA-Based Handshake Protocol



- Simplest variation with server-only authentication and server public key suitable for RSA encryption.
- On completion of Phase 1, assume that RSA-based key exchange has been selected.

Main Handshake Messages

Client hello States highest TLS version available, advertises ciphersuites, sends client nonce N_C.

Server hello Returns the selected version and ciphersuite and sends server nonce N_S.

Server key exchange Server's input to key exchange.

Client key exchange Client input to key exchange.

Change-cipher-spec Switch to newly negotiated ciphersuite for record layer.

Ephemeral Diffie–Hellman Handshake Variant

Server key exchange Diffie–Hellman generator and group parameters and server ephemeral Diffie–Hellman value, all signed by the server.

Client key exchange Client ephemeral Diffie–Hellman value (optionally signed if client certificate is used).

- Pre-master secret *pms* is the shared Diffie–Hellman secret.
- Provides forward secrecy and therefore *recommended* today.

RSA Handshake Variant

Server key exchange Not used (but the RSA public key is available).

Client key exchange Key transport of pre-master secret *pms*:

- Client selects random pre-master secret *pms*.
- Client encrypts *pms* with server's public key and sends the ciphertext to server.
- Server decrypts using its private key to recover *pms*.

No forward secrecy and *not recommended* today.

Generating Session Keys

- The master secret ms is defined using the premaster secret pms as:

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

- As much keying material is generated as required by the ciphersuite using:

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

- Independent session keys are partitioned from k in each direction (a write key and a read key on each side), and may be updated frequently.
- Depending on the ciphersuite, keying material may include the following: encryption key, MAC key, IV.

The Pseudorandom Function PRF

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

- Example (TLS 1.2):

$$\begin{aligned} PRF(K, \text{label}, r) = {} & HMAC(K, A(1) \parallel \text{label} \parallel r) \\ & \parallel \; HMAC(K, A(2) \parallel \text{label} \parallel r) \\ & \parallel \; \ldots \end{aligned}$$

where $A(0) = r$, $A(i) = HMAC(K, A(i-1))$ and HMAC uses a specified SHA-2 variant (typically SHA-256); r is a seed.

Other Handshake Variants

Diffie–Hellman (DH) Parties use **static** DH with certified keys – if the client lacks a certificate (typical on the Internet), the client uses an ephemeral DH key.

Anonymous DH (DH_Anon) Ephemeral DH keys are not signed at all; protects only against passive eavesdropping.

These handshake methods are possible but not recommended today.

Forward Secrecy

- Forward secrecy: compromise of long-term keys should not compromise past session keys.
- Typically achieved with Diffie–Hellman key exchange authenticated with signatures from long-term keys.
- RSA-based handshake in TLS does not achieve forward secrecy but is still seen in some ciphersuites.
- Ephemeral (EC)DH ciphersuites provide forward secrecy.

SSL and TLS Limitations

- Higher layers should not assume TLS will always negotiate the strongest possible connection since this depends on both parties involved.
- A man-in-the-middle can attempt to force downgrade to weaker methods.
- Example: the adversary may block the secure port, or force negotiation of an unauthenticated connection (this is prevented in TLS 1.3).

TLS Protocol Summary

- TLS includes two main protocols: the Handshake Protocol (using asymmetric algorithms to negotiate keys and authenticate users) and the Record Layer Protocol (using symmetric algorithms for communication).
- New versions of TLS have been rolled out as the understanding of cryptographic security and attacks have increased over the years.
- Backward compatibility is a problem:
  - SSL 3.0 was deprecated as late as 2015.
  - TLS 1.0 is more than 20 years old and still widely supported.
- TLS assumes reliable delivery of messages (TCP, not UDP).

Contents

Attacks on TLS

- In recent years there have been many practical attacks on TLS.
- Many servers do not support the latest TLS versions and/or lack mitigations, so they will continue being vulnerable.
- SSL Pulse survey tracks real-world deployment of mitigations.
- There is a good coverage of attacks on Matt Green's blog.
- The following slides show a selection of attacks (there are more).

The CRIME and BREACH Attacks

- Side-channel leakage via **compression**: different inputs yield different compression, which reveals information about the plaintexts.
- CRIME targets TLS compression; BREACH targets HTTP compression.
- The idea was known already in 2002; the first practical demo in 2012.
- Common mitigation: disable TLS compression; disabling the HTTP compression would hurt the performance much more.

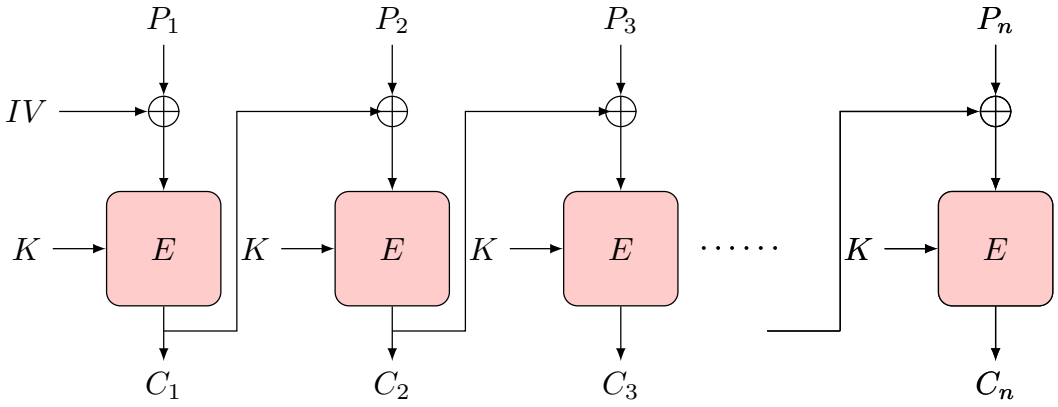
The Heartbleed Bug

- Implementation bug in OpenSSL (the heartbeat extension bounds check was missing, where the party would return a given length message).
- Leaks server memory, potentially including session and long-term keys.
- Discovered April 2014; required widespread key updates.
- Raised concerns about reliance on critical open-source components.

Cipher Block Chaining Mode – Reminder

- CBC chains blocks together. A random initialization vector IV is chosen and sent with the ciphertext blocks.
- *Encryption*:
  - $C_t = E(P_t \oplus C_{t-1}, K)$, where $C_0 = IV$ is sent as a first block.
  - P_t is XOR'd with C_{t-1} and encrypted with key K to produce C_t.
- *Decryption*:
  - $P_t = D(C_t, K) \oplus C_{t-1}$, where $C_0 = IV$.
  - C_t is decrypted with key K and XOR'd with C_{t-1} to produce P_t.
  - As in encryption, C_0 is the IV, used to retrieve the first block.

CBC Mode Encryption – Reminder



- IV and blocks $C_1, C_2, \ldots, C_n$ are sent.

The BEAST Attack

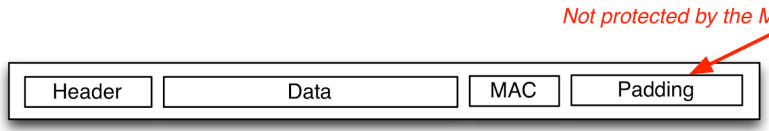
- BEAST exploits non-standard IV chaining in CBC mode – IVs are chained from previous ciphertexts in which the adversary may control.
- Allows attacker to recover plaintext byte by byte (but not the secret key).
- The weakness has been known since 2002; first practical demo in 2011.
- TLS 1.1 now mandates random IVs to avoid this attack going forward.
- Browsers mitigated this attack by splitting the plaintext to enforce randomized IV and additional MAC computations.
- The BEAST attack is no longer considered a realistic threat.

Encrypt-then-MAC vs. MAC-then-Encrypt

- Best practice is encrypt-then-MAC; TLS historically did MAC-then-encrypt.
- Upon encrypting a record:
  - Sender first applies a MAC to the plaintext.
  - Then adds up to 255 bytes of padding to reach a multiple of the block size.
  - Then CBC-encrypts the record.

The Padding Attacks

The critical point is that the padding is *not* protected by the MAC.



This enables a *padding oracle*: an attacker flips bits in the ciphertext and re-sends; if they can learn whether padding changed, they can adaptively decrypt the record in a chosen ciphertext attack.

TLS Error Padding Oracles and the POODLE Attack

- A padding oracle reveals whether the plaintext padding is correct.
- CBC mode can leak padding correctness due to error propagation.
- This was then applied to TLS in several different attacks.
- Main mitigation: uniform error response (making if indistinguishable whether it was a padding or a MAC error under failure).
- POODLE (2014): forces downgrade to SSL 3.0 then runs padding oracle.

TLS Timing Padding Oracle Attack

- Discovered by Nadhem AlFardan and Kenny Patterson ("Lucky 13").
- TL;DR: subtle timing bug in TLS decryption for CBC-mode ciphersuites can, potentially, enable decryption of sensitive data (passwords, cookies).
- Borderline practical in DTLS; largely theoretical for TLS in typical settings.

https://blog.cryptographyengineering.com/2013/02/04/attack-of-week-tls-timing-oracles
https://en.wikipedia.org/wiki/Lucky_Thirteen_attack

The Attack

- TLS designers were aware of padding oracle risks but opted for implementing mitigations rather than a redesign of the protocol.
- They removed detailed error messages that distinguished padding failures from MAC failures, hoping to avoid the attack.
- Researchers then used a *timing* attack instead:
  - Measure decryption time to infer padding validity vs MAC validity.
  - Many implementations checked padding before MAC and returned early.

The Attack

"[T]he best way to do this is to compute the MAC even if the padding is incorrect, and only then reject the packet. For instance, if the pad appears to be incorrect, the implementation might assume a zero-length pad and then compute the MAC."

- When padding check fails, the decryptor does not know how much padding to strip. Hence it does not know how much data to MAC.
- Recommended countermeasure: assume no padding and MAC the whole blob. Result: MAC computation can be longer when padding is damaged.

The Attack

"This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal."

- This was believed safe for years.
- The Lucky 13 attack showed it can be distinguished (under LAN conditions), with sub-microsecond timing.
- Advances in hardware and careful statistics enabled exploitation.

TLS Security Summary

- Attacks stem from implementation bugs, weak primitives, and protocol flaws. Attacks tend to get better over time.
- Backward compatibility enables downgrade attacks to older versions.
- Complexity is a major problem. TLS 1.3 removes many options and simplifies the handshake, and has not seen general attacks (yet).
- TLS 1.3 however adds new features (e.g., 0-RTT) with new challenges.

Questions?