

Lecture 10 – Transport Layer Security (TLS)

Putting it all together

Bruno Grenet



<https://membres-ljk.imag.fr/Bruno.Grenet/IntroCrypto.html>

Introduction to cryptology
Université Grenoble Alpes – IM²AG
M1 INFO, MOSIG & AM

What is TLS?

A cryptographic protocol to provide secure communication over a network

Features

- ▶ Data encryption
- ▶ Server and optional client authentication
- ▶ Integrity checking

Usage

- ▶ On top of some reliable transport protocol
- ▶ Browsing (`https`), file transfer (`ftps`), email (`smtps`), VoIP (`xmpp`), ...

e.g TCP

Two-stage protocol

Handshake: negotiation of cryptographic parameters, key exchange

Record-layer: authentication and encryption of the communication

A very brief history of TLS

The SSL family

- ▶ Produced by Netscape Communications (Tahar Elgamal)

SSL 1.0: Unpublished

SSL 2.0: 1995 – deprecated in 2011

SSL 3.0: 1996 – deprecated in 2015

many security flaws

The TLS family

- ▶ Standardized by the Internet Engineering Task Force

TLS 1.0: 1999 – deprecated in 2021

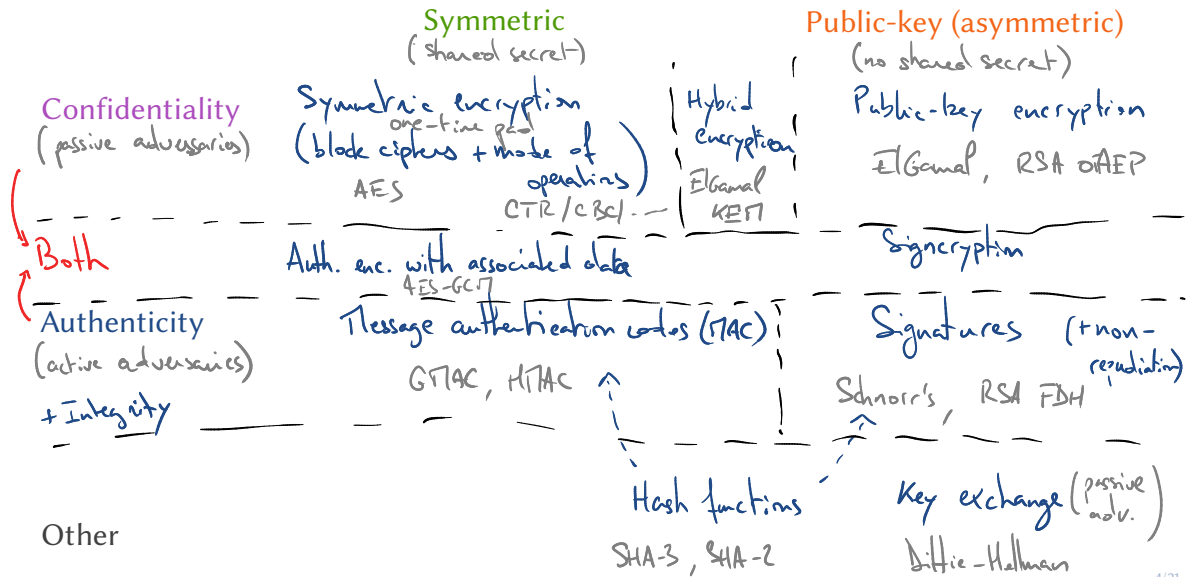
TLS 1.1: 2006 – deprecated in 2021

TLS 1.2: 2008 – in use

TLS 1.3: 2018 – in use

our focus

Quick summary of previous lectures



Contents

1. TLS Handshake protocol

2. TLS record-layer protocol

3. Some attacks

TLS Handshake protocol: goals

Establish a secure session between the client and the server

Agree on the algorithms/protocols

- ▶ Version of the protocol
- ▶ Which algorithms to use
- ▶ Key size

Authentication

- ▶ Authenticate the server using certificate authorities
- ▶ (possibly) Authenticate the client

Key exchange

- ▶ Set-up keys for future encryption / authentication

TLS Handshake protocol: the ingredients

Starting point: Key exchange

- ▶ The Client and Server must agree on shared keys for subsequent communication
- ▶ Use of Diffie-Hellman Key Exchange protocol

“What-ifs?”

What if an adversary intercepts the messages from the Server?

→ the Server signs its messages with its private key

What if the public key is not really the Server's public key?

→ the Server provides a certificate from a Certificate Authority

What if an adversary *replays* the exchange?

→ use random nonces to make replays impractical

What if some messages were modified in transit?

→ use a MAC

What if the Server and Client do not use the same algorithms?

→ include their descriptions in the messages

TLS Handshake protocol: overview

Client initial data: a set of CAs public keys $\{pk_1, \dots, pk_n\}$

Server initial data: a pair (pk_S, sk_S) and a certificate $\text{cert}_{i \rightarrow S}$ from a CA

1. **Client** sends a message with one or several sets of parameters:

- ▶ a group $G = \langle g \rangle$, a symmetric encryption scheme Enc and hash function H
- ▶ a group element $h = g^x \in G$ and a random *nonce* $N_C \leftarrow \{0, 1\}^n$

2. **Server:** upon reception, chooses a set of parameters (G, Enc, H) and

- ▶ computes a shared secret $K = g^{xy}$ and keys k'_S, k'_C, k_S, k_C using a *key derivation function*

- ▶ sends a message made of

↳ based on H

- ▶ the choice of parameters
- ▶ the group element g^y and a random *nonce* $N_S \leftarrow \{0, 1\}^n$
- ▶ $\text{Enc}_{k'_S}(pk_S, \text{cert}_{i \rightarrow S}, \sigma)$ where σ is a signature (with sk_S) of the message

3. **Client:** upon reception,

- ▶ computes the shared secret K and the keys k'_S, k'_C, k_S and k_C
- ▶ decrypts $pk_S, \text{cert}_{i \rightarrow S}$ and σ and checks whether pk_S and σ are valid
- ▶ computes and sends a MAC of all exchanged messages, using k'_C

4. **Server:** checks the MAC

Shared final data: keys k_S and k_C + algorithms

*Diffie-Hellman
key exchange*

Sets of parameters

Groups

- ▶ One of several predefined groups
- ▶ Includes elliptic curves and subgroups of finite fields

Symmetric encryption

- ▶ Requires Authenticated Encryption with Associated Data
- ▶ Includes GCM with AES-128

Lectures 2 and 5

Hash function

- ▶ Used for key derivation function and HMAC
- ▶ Includes SHA-256

Lecture 4

The key derivation function

Goal

From a secret K , deduce one or several keys

- ▶ The secret may not have the right format
- ▶ The secret may not be uniform in a suitable set

Example of HKDF

Input: a secret K , optional salt s , optional info i , output length L

1. $t \leftarrow \text{HMAC}(s, K)$ *extract stage*
2. $z_0 \leftarrow$ empty string
3. for $j = 1$ to L :
4. $z_j \leftarrow \text{HMAC}(t, z_{j-1} \| i \| \text{0x}\langle j \rangle)$ *expand stage*
5. Return $z_1 \| \dots \| z_L$

Security intuition for TLS

Authentication

- ▶ Using the certificate, pk_S is guaranteed to be the correct public key
- ▶ If σ is valid, the client must be communicating with the intended server
- ▶ Protection against replay attack: use of the random nonce

Integrity

- ▶ The server signs all messages of the Diffie-Hellman key-exchange
 - ▶ the values were not modified in transit
 - ▶ protection against a person-in-the-middle attack

Confidentiality

- ▶ Based on Diffie-Hellman key exchange security

Beware!

- ▶ This is far from a formal proof!

Contents

1. TLS Handshake protocol

2. TLS record-layer protocol

3. Some attacks

TLS record-layer protocol: goal

Ensure confidentiality, integrity and authenticity of the communications

Context

- ▶ The Client and Server share two keys k_S and k_C
- ▶ They agreed on a set of cryptographic algorithms

Tools

- ▶ Symmetric encryption
- ▶ Message authentication codes

→ Authenticated Encryption with Associated Data

The tool: Nonce-based Authenticated Encryption with Associated Data

Construction

Encryption: $c \leftarrow E_k(m, d, N)$ where

- ▶ m is the message
- ▶ d is the associated data
- ▶ N is a *nonce*

encrypted
clear

Decryption: $D_k(c, d, N)$ returns

- ▶ either m , if c is correct, and d, N are unchanged
- ▶ or « reject »

Properties

Correction if for all k and m , $D_k(E_k(m, d, N), d, N) = m$

Security if whenever *no nonce is used more than once*,

- ▶ ciphertexts are *indistinguishable*
- ▶ a valid ciphertext is hard to *forge*

IND-CPA security
ciphertext integrity

TLS record-layer protocol: overview

Shared data

- ▶ two keys k_C and k_S
- ▶ a *sequence number* n , initialized to 0

Data sending

- ▶ Data is split into blocks of 2^{14} bytes
 - ▶ Each block is sent separately
 - ▶ For each block, nonce-based AEAD encryption with inputs:
 - ▶ k : k_C or k_S , depending on the sender
 - ▶ m : the block to be sent
 - ▶ d : empty associated data
 - ▶ N : $n \oplus IV$ where $IV = IV_C$ or IV_S is random
- obtained in the handshake

End of session

- ▶ Not part of the TLS protocol
- ▶ Delegated to the application layer

Contents

1. TLS Handshake protocol

2. TLS record-layer protocol

3. Some attacks

Attack on the CA infrastructure

Stevens *et al.*, 2009

Reminder: role of certificates

- ▶ A Certificate Authority signs a certificate $\text{cert}_{CA \rightarrow S}$ for the Server's public key pk_S
- ▶ The Client can accept the public key as valid
- ▶ A *fake certificate* allows an adversary to impersonate the Server

A bad signature algorithm: RSA-MD5

- ▶ Signature algorithm (simplified):
 1. Hash the value using MD5 $\rightarrow H(m)$
 2. Compute $\sigma = H(m)^d \bmod N$
- ▶ Collision attack on MD5 $\rightarrow$ forge of signatures

Fake certificates

- ▶ The adversary asks a CA to sign a certificate $\text{cert}_{CA \rightarrow S}$
- ▶ The adversary finds a collision $\rightarrow \text{cert}'$ with the same signature
- ▶ Difficulty: cert' should be a certificate $\rightarrow$ *chosen-prefix collision*

Attack on the handshake protocol

Logjam attack, Adrian *et al.*, 2015

Context

- ▶ TLS gives the choice between different algorithms / key sizes
- ▶ Some of them are too weak
- ▶ Example: 512-bit subgroups of finite fields for Diffie-Hellman key exchange
 - ▶ Discrete logarithm within a few minutes (after two weeks of precomputation)

An active attack strategy

1. Intercept the Client's message to the Server
 - ▶ Tamper it to ask for weak DH parameters
 - ▶ Forward to the Server
2. Intercept the answer from the Server
 - ▶ Hide the weak request
 - ▶ Forward the Server's DH parameters
3. Compute a discrete logarithm → get the shared secret

Logjam attack in practice

Real-life experiment

- ▶ Attack implemented in practice
- ▶ Tested on the top 1 million domains → 8.4% of them were vulnerable
- ▶ Not only for HTTPS, but also SMTP+STARTTLS, POP3S, IMAPS

Reasons for success

- ▶ Some servers still implement weak cryptography
- ▶ Some clients fail to reject weak DH groups
- ▶ Efficient discrete logarithm computations
- ▶ Some clients are fine with waiting

Discrete logarithms with precomputation: Number Field Sieve (NFS)

Precomputation: build a database of discrete logarithms

Computation: *descent* to compute a targeted discrete logarithm

- ▶ *Offline* precomputation, *shared* for all subsequent computations

Attack on the record-layer protocol

BEAST attack, Duong & Rizzo, 2011

The theoretical vulnerability (Rogaway, 2002)

- ▶ Symmetric encryption using CBC mode of operation
 - ▶ $m_1 || m_2 || \dots || m_t \rightarrow c_0 || c_1 || \dots || c_t$
 - ▶ c_0 is a *random* IV
 - ▶ $c_i \leftarrow E_k(m_i \oplus c_{i-1})$ for $i > 0$
- ▶ Attack if IV is not uniform

Lecture 3, slide 11/20

Exploitation of the vulnerability

- ▶ In TLS 1.0, use of CBC with a predictable IV $\rightarrow$ last block of previous record
- ▶ The attack focuses on authentication cookies
- ▶ (Partially) Chosen Plaintext Attack

It is that easy?

- ▶ Requires code injection for instance
- $\rightarrow$ Attacks mix cryptography techniques with security techniques

Conclusion

TLS mixes everything we have seen!

- ▶ Symmetric encryption through Authenticated Encryption with Associated Data
- ▶ MAC: direct use and through AEAD
- ▶ Diffie-Hellman key exchange
- ▶ Signatures
- ▶ Public-Key encryption through KEM → in a variant called KEMTLS

Protocols are hard to design

- ▶ Many attacks on different aspects of TLS
- ▶ Every tiny vulnerability will probably be exploited

For more: Cybersecurity master!

- ▶ More advanced cryptographic primitives and concepts
- ▶ More details on security architectures
- ▶ Other aspects of cybersecurity