Lecture 10 – Transport Layer Security (TLS)

Putting it all together

Crypto2526!en

Crypto2526!fr

Bruno Grenet

May 25.

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 *e.g* TCP
- Browsing (`https`), file transfer (`ftps`), email (`smtps`), VoIP (`xmpp`), …

Two-stage protocol

Handshake: negotiation of cryptographic parameters, key exchange

Record: 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SSL 1.0: Unpublished — many security flaws
SSL 2.0: 1995 – deprecated in 2011
SSL 3.0: 1996 – deprecated in 2015

The TLS family

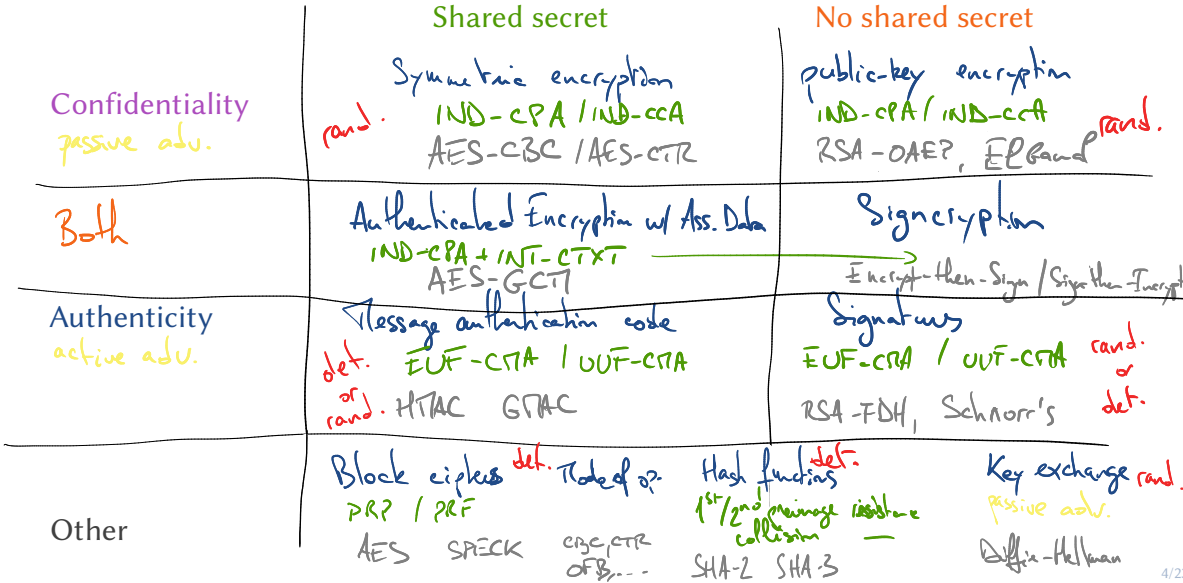
- Standardized by the Internet Engineering Task Force

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

Remark

- From TLS 1.0 to 1.2: add new features $\rightarrow$ increase in complexity
- TLS 1.3: rewrite, keeping only good parts $\rightarrow$ simpler and safer protocol

Quick summary of previous lectures



Contents

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

Key exchange

- Set-up keys for future encryption / authentication

Remark

- No authentication of the client
  - Delegated to the application
  - In the record protocol

password-based and/or using cookies

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 signing key

What if the verification key is not really the Server's verification 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: a simplified picture

Server

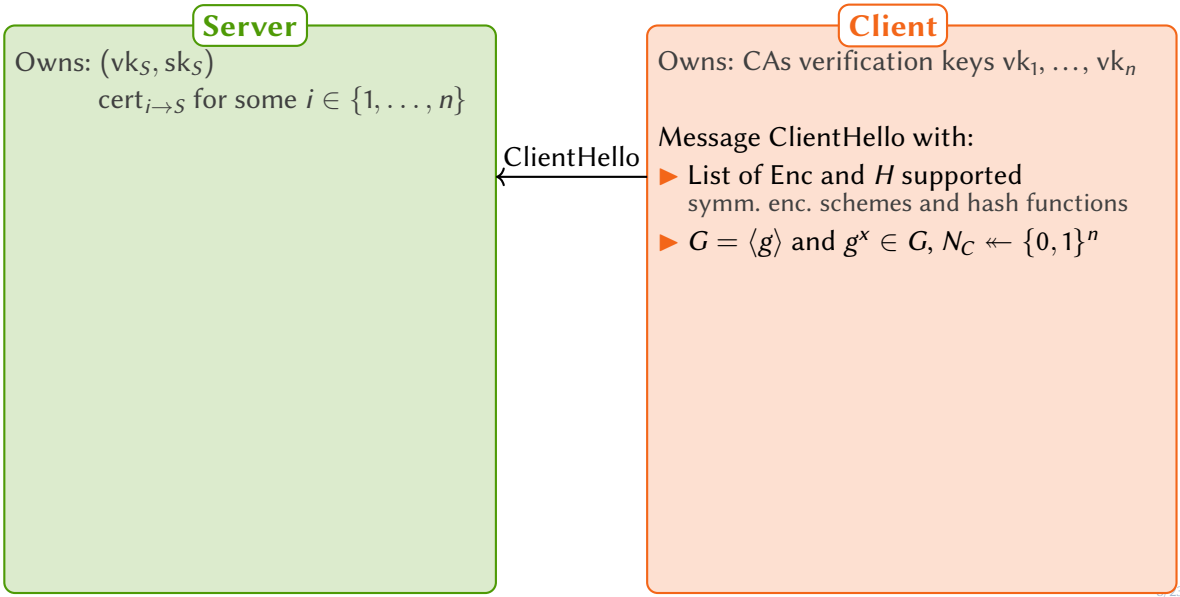
Owns: $(\mathsf{vk}_S, \mathsf{sk}_S)$

$\mathsf{cert}_{i \to S}$ for some $i \in \{1, \ldots, n\}$

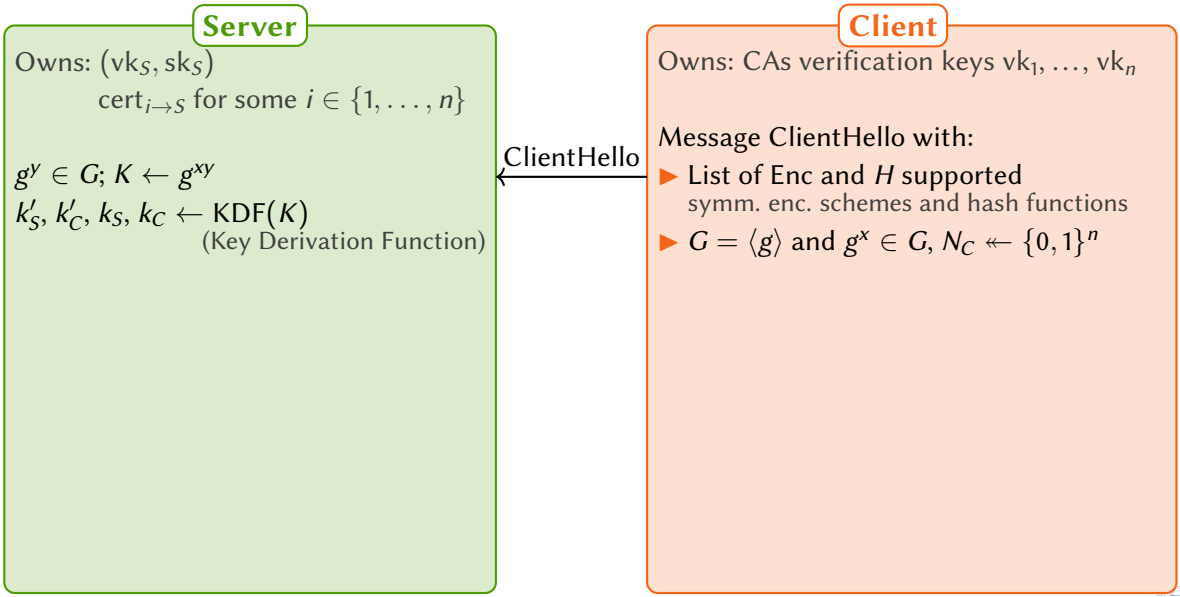
Client

Owns: CAs verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$

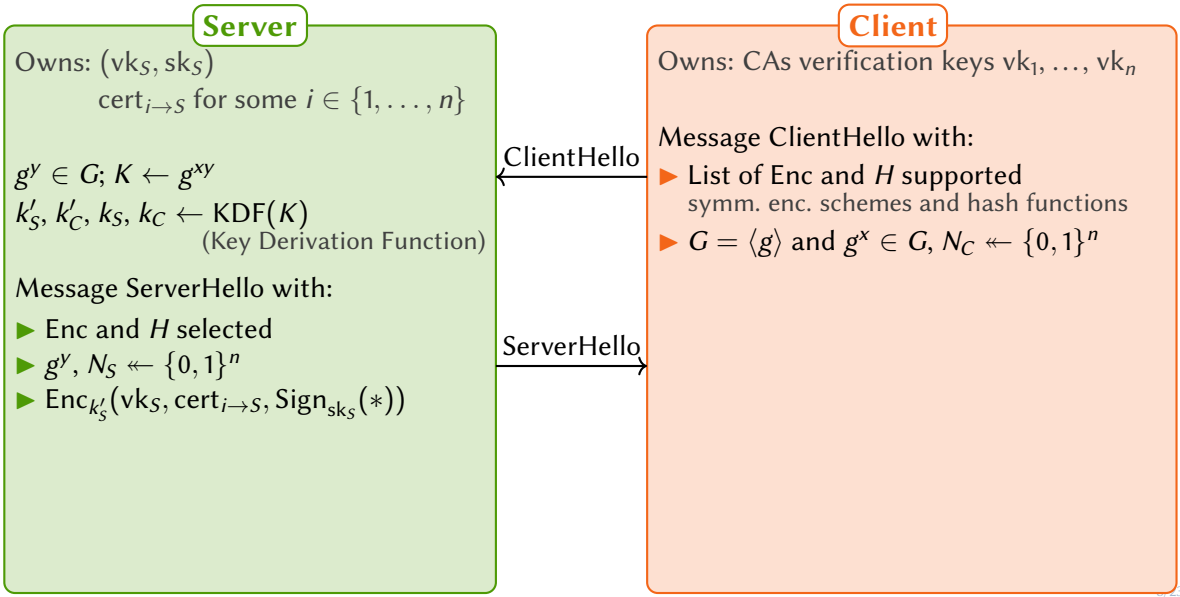
TLS Handshake protocol: a simplified picture



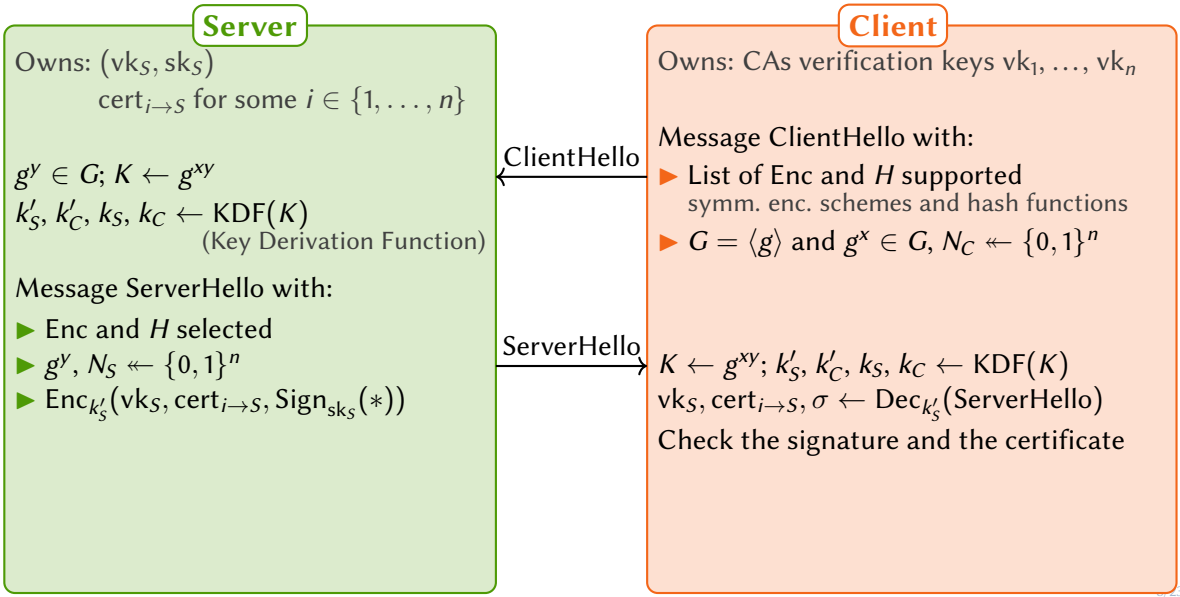
TLS Handshake protocol: a simplified picture



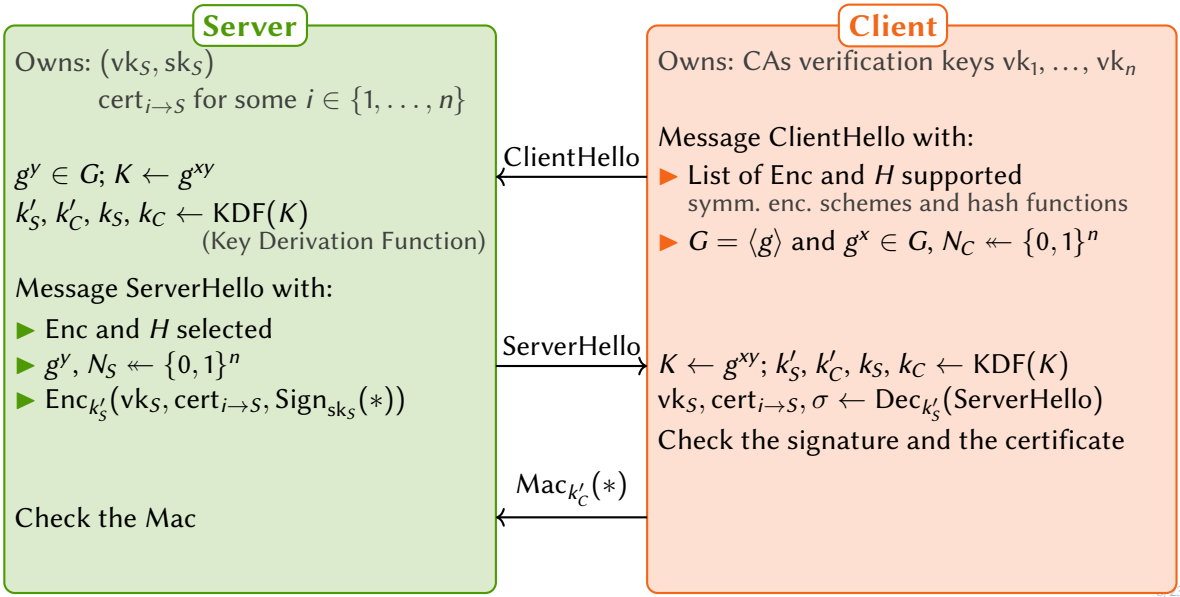
TLS Handshake protocol: a simplified picture



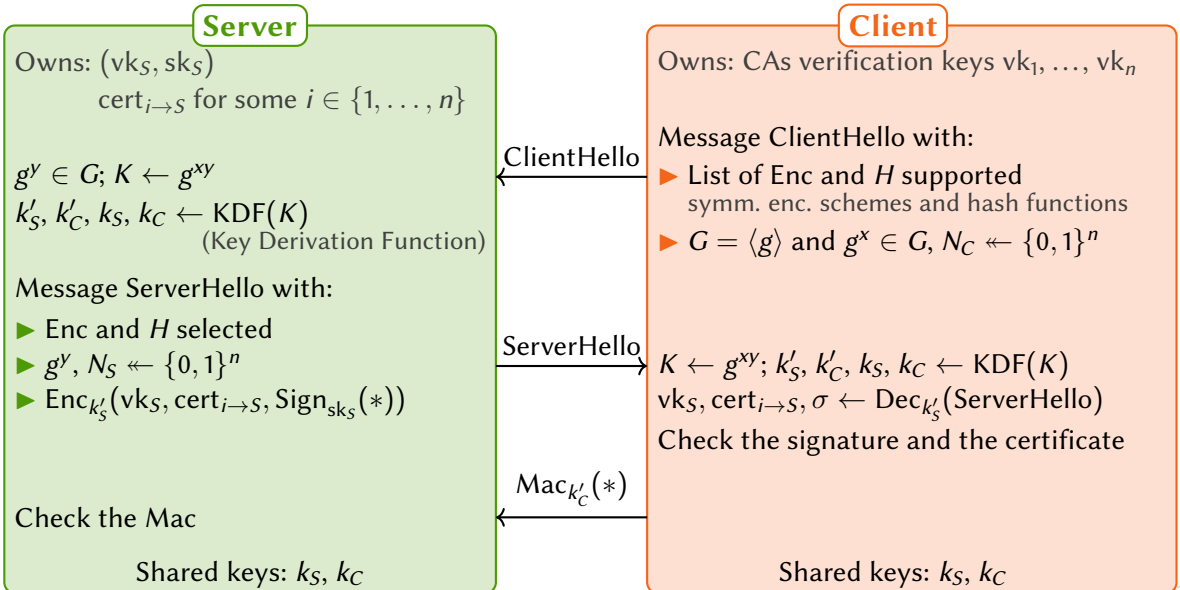
TLS Handshake protocol: a simplified picture



TLS Handshake protocol: a simplified picture



TLS Handshake protocol: a simplified picture



Choices of parameters

Possible groups

- Subgroups of $\mathbb{Z}/p\mathbb{Z}^\times$ — 5 possibilities, from 2048 to 8192 bits
- Elliptic curves — 5 possibilities, including Curve25519 and Curve448

Symmetric encryption

- Requires Authenticated Encryption with Associated Data
  - AES-GCM — Lectures 2 and 5
  - AES-CCM — slightly less efficient
  - ChaCha20 stream cipher + Poly1305 Mac — Lecture 5
- Key sizes:
  - either 128 bits — AES-GCM or AES-CCM
  - or 256 bits — AES-CCM or ChaCha20-Poly1305

Hash function

- Used for key derivation function and HMAC
- Either SHA-256 or SHA-284 (SHA-2 family) — 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. $t \leftarrow \mathrm{HMAC}(s, K)$ *extract* stage
2. $z_0 \leftarrow$ empty string
3. for $j = 1$ to L:
4. $\quad z_j \leftarrow \mathrm{HMAC}(t, z_{j-1} \| i \| 0\mathrm{x}\langle j \rangle)$ *expand* stage
5. Return $z_1 \| \cdots \| z_L$

Security intuition for TLS Handshake

Authentication

- $\text{cert}_{i \to S}$: vk_S is the correct verification key
- Valid signature: the client communicates with the intended server
- Random nonces: protection against replay attack

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
- The client computes a Mac of the transcript
  - No *downgrade* attack

Confidentiality

- Based on Diffie-Hellman key exchange security

Beware!

- This is **not** a proof!

Contents

TLS record 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

$\rightarrow$ Authenticated Encryption with Associated Data

Two additional securities

- *Zero padding*: pad the messages with a random number of zeroes — *traffic analysis*
- Limits on key usage: *e.g.* $2^{24.5}$ messages for AES-GCM — inherent weakness

The tool: Nonce-based Authenticated Encryption with Associated Data

Construction

Encryption: $c \leftarrow \mathrm{Enc}_k(m, d, N)$ where

- m is the message — encrypted
- d is the associated data — clear
- N is a *nonce*

Decryption: $\mathrm{Dec}_k(c, d, N)$ returns

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

Properties

Correction if for all k and m, $\mathrm{Dec}_k(\mathrm{Enc}_k(m, d, N), d, N) = m$

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

- ciphertexts are *indistinguishable* — IND-CPA security
- a valid ciphertext is hard to *forge* — ciphertext integrity

TLS record 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 obtained in the handshake derived from K

End of session

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

Contents

Attack on the CA infrastructure

Stevens *et al.*, 2009

Reminder: role of certificates

- A Certificate Authority signs a certificate $\text{cert}_{CA \to S}$ for the Server's key vk_S
- The Client can accept the verification 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 $\to H(m)$
  2. Compute $\sigma = H(m)^d \bmod N$
- Collision attack on MD5 $\to$ forge of signatures

Fake certificates

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

Chosen-prefix collisions

Definition

$H(p\|c\|s) = H(p'\|c'\|s)$ where

- p, p', s: chosen by the attacker
- c, c': result of the attack

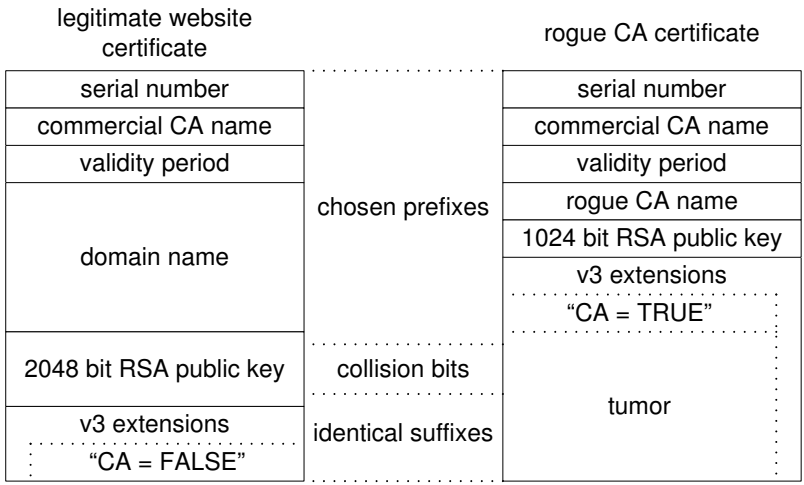
The case of MD5

- 1992: Design – 128 bit hash function — Rivest
- 1993: Collision attack on the compression function — den Boer & Bosselaers
- 2005: Practical collision attack on the hash function — Wang & Yu
- 2007: Practical chosen-prefix collisions → fake certificates — Stevens, Lenstra, de Weger
- 2009: *Short* chosen-prefix collisions → *rogue CA* certificates — Stevens *et al.*

Rogue Certificate Authority

- CA is legitimate if certified by another legitimate CA
- Fake certificate of legitimacy
- Can then issue many certificate

Example from Stevens *et al.* (2009)



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 $\rightarrow$ get the shared secret

Logjam attack in practice

Real-life experiment

- Attack implemented in practice
- Tested on the top 1 million domains $\rightarrow$ 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 protocol

BEAST attack, Duong & Rizzo, 2011

The theoretical vulnerability (Rogaway, 2002)

- Symmetric encryption using CBC mode of operation
  - $m_1 \| m_2 \| \cdots \| m_t \rightarrow c_0 \| c_1 \| \cdots \| c_t$
  - 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 predictible IV $\rightarrow$ last block of previous record
- The attack focuses on authentication cookies
- (Partially) Chosen Plaintext Attack

Is it 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 $\rightarrow$ 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