

Introduction to cryptology (GBIN8U16)



A few things about TLS

Pierre Karpman

`pierre.karpman@univ-grenoble-alpes.fr`

`https://www-ljk.imag.fr/membres/Pierre.Karpman/tea.html`

2019-04-17

Back to the start: set up a secure channel

A client C wants to securely communicate with a server S :

- ▶ S should prove to C that it is the right server
 - ▶ Using a public-key digital signature (e.g. RSA-PSS)
- ▶ C and S should exchange a shared secret
 - ▶ Using asymmetric key exchange (e.g. DH)
- ▶ C and S may use a shared secret to communicate
 - ▶ Using an authenticated symmetric encryption scheme (e.g. AES-CBC + HMAC-SHA-256)

TLS: *Transport Security Layer*

- ▶ Former SSL (*Secure Socket Layer*): 95-99
- ▶ Latest version: 1.2 since 2008
- ▶ TLS 1.3 is nearly ready!
- ▶ Quite a complex protocol
 - ▶ Mixes crypto, networking, implementation aspects
 - ▶ Cf. e.g. the RFCs; Wikipedia's article

TLS in a small nutshell

TLS uses:

- ▶ *A handshake protocol*
 - ▶ To set up the shared secret
- ▶ *A record protocol*
 - ▶ To further exchange data

It also relies on a certification authority (CA)

- ▶ To help trusting the servers, if one needs that

A brief handshake

Goal of the handshake:

- ▶ (Perform the key exchange; possibly prove S 's identity; possibly (rarely) prove C 's identity)
- ▶ Negotiate the protocol's version
- ▶ Negotiate the algorithms to be (later) used

In a borrowed picture

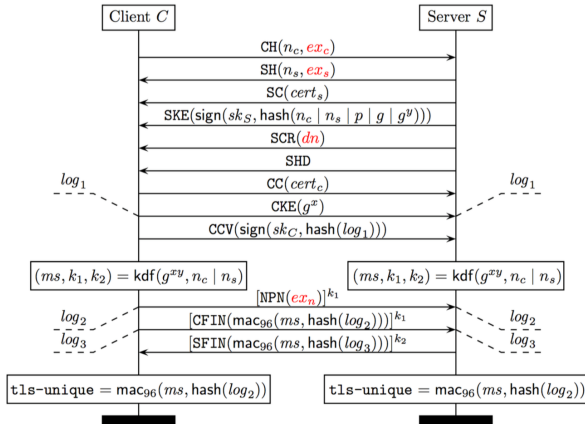


Figure: A mutually-authenticated DHE handshake, from (Bhargavan & Leurent, 2016)

Some comments

- ▶ The server's key exchange parameters are signed
 - ▶ Shows that it knows its secret key
 - ▶ Prevents tampering
- ▶ The exchange is concluded by two-ways encrypted MACs of the transcript
 - ▶ Allows to check that all secrets are indeed shared
- ▶ `tls-unique` may be used to uniquely (err... not really) identify the exchange
 - ▶ May be used later at the application level

What about certificates now?

X.509 Certificates: \approx signed public keys; specify among others:

- ▶ A serial number
- ▶ The algorithm used to sign the certificate
- ▶ Identities
 - ▶ Of the issuer (e.g. *Let's Encrypt*, typically a *Certification Authority*)
 - ▶ Of the subject (e.g. `secure.iacr.org`)
- ▶ Validity dates
- ▶ The subject's public key (for a specified algorithm)
- ▶ *Whether the subject is a Certification Authority*

Certification Authorities

CAs:

- ▶ Are trusted (by your browser)
- ▶ Authenticate third parties
 - 1 Establish that a user S is who it claims to be
 - 2 Establish that it knows a public/secret key pair
 - 3 Agree to sign a certificate with these information
 - ▶ A client trusting the CA may now trust S 's certificate
- ▶ May delegate trust to third parties
 - ▶ Leading to certification chains: "Root" CA \rightarrow (Intermediary CA)* \rightarrow End subject
 - ▶ (A CA may (not) be restricted in the length of chains it can issue)
- ▶ (If a CA is malicious/compromised, then things can turn *bad*)

Who signs what

Depending on the context, certificates may e.g.:

- ▶ Altogether not be signed by a CA
 - ▶ Instead being self-signed: prevents tampering in e.g. TLS handshakes; one has to already trust the issuer
 - ▶ Only for a small-scale context; quite brittle
- ▶ Signed by a free CA
 - ▶ E.g. <https://letsencrypt.org/>. Quite recent; nice!
- ▶ Signed by a commercial/organisational CA (e.g. DigiCert/TERENA)

Finer-grain management: certificate “pinning”

An issue with the CA approach:

- ▶ There are *many* CAs
 - ▶ 100+ Root CAs, that can further delegate
- ▶ CAs could issue fake certificates
 - ▶ If compromised; if acting maliciously
 - ▶ Happened in practice (e.g. DigiNotar in 2011)

A remediation strategy: certificate/public key pinning:

- ▶ Services/websites declare (e.g. to a browser developer) which specific CA issued their certificate
- ▶ Upon connection, valid certificates from other CAs are rejected
- ▶ (But hard to deploy for everyone; scalability issues; browsers (say) need to be trusted?)

Alternative approach: certificate transparency

Cf. <https://www.certificate-transparency.org>: create a giant trusted log of certificates

- ▶ CAs, users may submit certificates to an append-only log
- ▶ Publicly record misuse/attacks
- ▶ Double-check the authenticity of a (doubtful) certificate
- ▶ (Kind of a heavy mechanism?)

~> Key distribution is a really hard problem!

What about attacks now?



TLS is:

- ▶ Widely used; useful
- ▶ Pretty complex
- ▶ Mixes many cryptographic algorithms
- ▶ Makes people feel safe

⇒ A very good real-world attack target

- ▶ Implementation-based (not crypto)
- ▶ Crypto-based (crypto)
- ▶ A selective overview of both kind:
<https://mitls.org/pages/attacks>

Three quick case studies

Let's have an overview of attacks on:

- ▶ The CA infrastructure
- ▶ The handshake protocol
- ▶ The record protocol

Quick case study 1: Fake CA thru MD5 collisions

MD5 quick facts:

- ▶ A 128-bit hash function from '92 (Rivest)
- ▶ Serious weaknesses found in '93 (den Boer & Bosselaers)
- ▶ Very efficient practical collision attacks in '05 (Wang & Yu)
- ▶ Efficient practical *chosen-prefix* collisions in '07 (Stevens & al.)
- ▶ Still pretty popular after that... ← Cryptographers are very bad at communication

Identical v. Chosen-prefix collisions

- ▶ An *identical-prefix* collision for a hash function is a collision of the form $m = p||c||s$, $m' = p||c'||s$
 - ▶ p , s may be chosen; c , c' are given by the attack
- ▶ A *chosen-prefix* collision is of the form $m = p||c||s$, $m' = p'||c'||s$
 - ▶ p , p' , s may be chosen; c , c' are given by the attack
- ▶ A generic attack is chosen-prefix by default
- ▶ *Cryptographic attacks* (w/ cost $< 2^{n/2}$, n the hash size) tend to be easier if identical-prefix

Chosen-prefix collision and fake CAs

- ▶ A once popular signing algorithm for certificates: RSA-MD5
 - ▶ Attack strategy: Ask a CA to sign an innocent-looking certificate cert
 - ▶ Prepare a colliding certificate cert'
 - ▶ The CA “also signed” cert'
- ▶ How's that useful?
 - ▶ No CA in their right mind would let a λ user become an intermediary CA
 - ▶ So make cert' be an intermediate CA certificate and wreak havoc on the internet
 - ▶ (Should now be detected/prevented through pinning, CT)

“Rogue CAs”

Exploiting hash collisions to create fake CAs works in practice (Stevens & al., 2009)

- ▶ Used a fast(er) chosen-prefix collision attack for MD5
- ▶ Fully done in the wild
- ▶ Further exploited predictability of certificates' serial numbers
- ▶ (Maybe using MD5 is not such a great idea?)

Colliding certificates structures

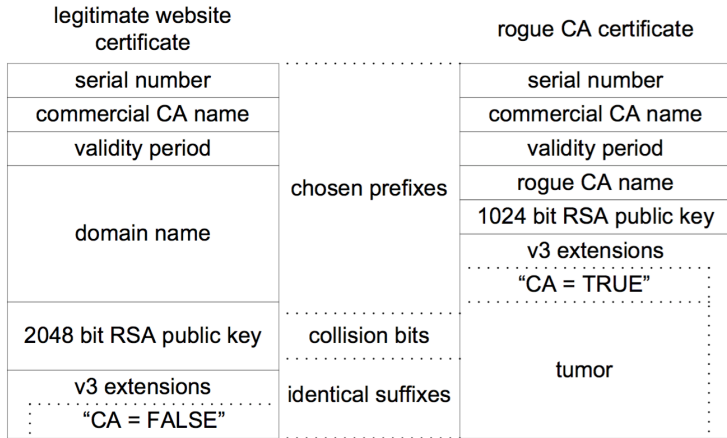


Figure: From (Stevens & al., 2009)

(MD5 CP collisions beyond TLS)

The strategy can be applied to other signing settings; it was also used to propagate the FLAME malware

- ▶ Detected in 2012, active since 2007?
 - ▶ (Most likely) targeted the Iranian nuclear program
- ▶ Passed as a malicious “Windows update”
- ▶ With a valid signature, obtained through a collision

Quick case study 2: “Logjam” weak DH attack

- ▶ Some of the algorithms that may be used w/ TLS are *weak*
 - ▶ E.g. the “export” suite from the 90’s
 - ▶ Include 512-bit groups for Diffie-Hellman (over finite fields)
 - ▶ For which a dlog can be computed within minutes (after two weeks of precomputation)
 - ▶ (And also symmetric encryption w/ 40-bit keys)
- ▶ These are open for negotiation during a TLS handshake
- ▶ Well-configured client do not ask for weak crypto
- ▶ But some servers may offer it
 - ▶ Weak crypto is better than no crypto?

An active attack strategy

Objective: impersonate the server to the client

- ▶ Intercept a client's message to the server, tamper it to ask for weak DH parameters, forward to the server
- ▶ Intercept the server's answer, tamper it to hide the bogus weak request, forward to the client
- ▶ Forward the server's DH parameters to the client
- ▶ Compute the dlog of the server's group element; derive the shared secret; authenticate the bogus transcript

This attack (and variants) have been implemented in practice (Bhargavan & al., 2015). It jointly exploited (among others) that (at the time):

- ▶ Some servers still implement weak crypto
- ▶ Some clients fail to reject weak DH groups (unlike e.g. weak block ciphers)
- ▶ Individual “export-grade” discrete logarithms can be computed quite fast
- ▶ Some clients are fine with waiting for that much time

Quick case study 3: The “BEAST” scenario

Some “theoretical” attacks on some encryption schemes are well-known:

- ▶ On weak ciphers[†]
 - ▶ E.g. RC4
- ▶ On bad implementations/strategies*
 - ▶ E.g. bad MAC-then-Encrypt checks
- ▶ On improper usage[†]
 - ▶ E.g. encrypting too much w/o changing the key

But these (†) attacks may have strong requirements, e.g.:

- ▶ Large data volume
 - ▶ E.g. $\approx 2^{32}$ blocks
- ▶ Partial knowledge of the messages
 - ▶ \approx Known-plaintext attacks

With “weak” results, e.g.:

- ▶ Do not result in key recovery
- ▶ Only allow to learn limited information
 - ▶ E.g. the XOR of two messages

So are these really a threat?

The target: Authentication Cookies

Cookies:

- ▶ Long-term data associated with an HTTP service, stored by a client's browser

Authentication Cookies:

- ▶ Cookies storing information that identifies/authenticates a user
- ▶ Useful to log in “automatically” on a web account
- ▶ Can be exported to other browsers
- ▶ Perfect target for a partial-plaintext-recovery attack!

A Cookie-harvesting strategy

An attacker (†-type) is happy if able to:

- ▶ Capture the network traffic of the target user
- ▶ Trigger many encryptions of the same target cookie
- ▶ (Potentially) know partial information about the data surrounding the cookie

The last two points are enabled by (Duong & Rizzo, 2011):

- ▶ Tricking the target user into visiting a malicious webpage
- ▶ Having the page request (e.g. using Javascript code) many connections to the cookie-using URL
 - ▶ Will (hopefully) be encrypted with a defective mechanism
 - ▶ Will attach the cookie as part of the query

Some Cookie-retrieval settings

RC4 biases (AlFardan & al., 2013):

- ▶ RC4 is a weak stream cipher with many keystream biases
- ▶ Lends itself well to *broadcast attacks*
 - ▶ Encrypt an unknown plaintext many times with different keys
 - ▶ Given the biases, guess its most probable value
- ▶ So just broadcast a cookie

64-bit block ciphers, e.g. (Bhargavan & Leurent, 2016):

- ▶ Use the generic collision attack on CBC encryption
- ▶ Require some known information in the plaintext
 - ▶ But network protocols typically provide that
- ▶ Find & exploit collisions between known data and unknown cookie

That was the last lecture...

- See you on 2019-05-02 for the exam

This is the end

