Security and Privacy

TLS, certificates and Trust

5.03.2019

Ph. Oechslin

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

TLS and HTTPS

Putting it all to work

com-402

TLS

- Transport Layer Security
  - previously known as SSL
- 'The primary goal of TLS is to provide a secure channel between two communicating peers' **IETF RFC 8846**
  - provides confidentiality, integrity and authentication
- Basic idea:
  - build your client-server app without security, add TLS, et voilà!
- History:
  - TLS 1.0, 1999, RFC 2246
  - TLS 1.1, 2006, RFC 4346
  - TLS 1.2, 2008, RFC 5246
  - TLS 1.3, 2019, RFC 8846 proposed standard

TLS building blocks

- The server is authenticated with a certificate
- It proves its identity by signing some information received from the client with its private key
- Client and server create a symmetric key using asymmetric crypto
  - Diffie-Hellman or EC Diffie Hellman, or key transfer (deprecated)
- They use a symmetric cipher to encrypt the data
- They use a HMAC to guarantee integrity

TLS Cipher Suites

- The algorithms to be used are specified in cipher suites:

`TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`

- protocol: `TLS` (TLS)
- key exchange: `ECDHE` (elliptic curve Diffie Hellman)
- signature algo to prove possession ot private key: `RSA`
- symmetric bloc cipher: `AES_128_GCM`
- hash function used for integrity (HMAC) and key derivation: `SHAH256`

Popular Cipher Suites (TLS 1.2)

```
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
TLS_DHE_DSS_WITH_AES_128_GCM_SHA256
TLS_DHE_DSS_WITH_AES_256_GCM_SHA384
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
```

Key exchange, Perfect Forward Secrecy

- 2 possiblities

1. Based on Diffie Helmann: (DHE, ECDHE)
2. RSA:
   - the client choses a random key,
   - encrypts it with the server's public key
   - sends it to the server

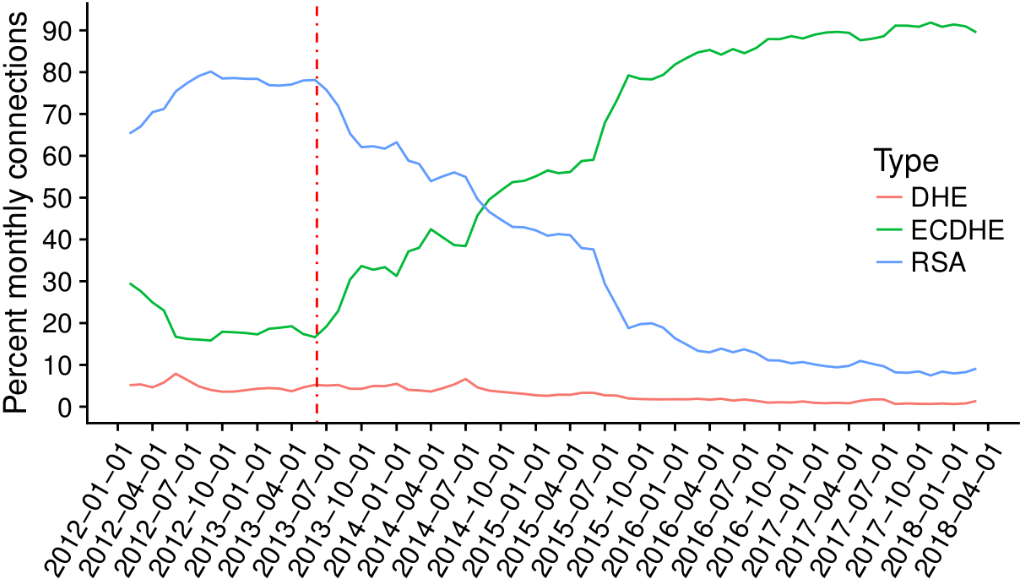
- Attack on RSA key exchange:
  - The attacker records the key exchange and all the traffic
  - Later he is able to steal the private key from the server
  - He can recover the symmetric key and decrypt all past traffic
- Diffie Hellman offers Perfect Forward Secrecy (PFS)
  - There is no way to recover the key in the future

Perfect Forward Secrecy

- At least since Ed. Snowden's revelations we know that governments
  - record a lot of traffic
  - exploit opportunities to steal private keys

➡ Diffie-Hellan requires a little more work from the client, but it should always be preferred over RSA key exchange

Perfect Forward Secrecy



source: **PKotzias**

TLS handshake

- According to RFC 5246

```
Client                                               Server

      ClientHello                  -------->
                                                ServerHello
                                               Certificate*
                                         ServerKeyExchange*
                                        CertificateRequest*
                                   <--------    ServerHelloDone
      Certificate*
      ClientKeyExchange
      CertificateVerify*
      [ChangeCipherSpec]
      Finished                     -------->
                                         [ChangeCipherSpec]
                                   <--------          Finished
      Application Data             <------->  Application Data
```

TLS handshake

Interesting facts:

- Cipher suites: The client gives a list of suites, the server chooses
- The server is authenticated by a certificate (almost always)
- The client can optionally be authenticated by a client certificate
- Session resumption: avoid doing a new key exchange
  - with session ID:
    - the client presents the id of the last session
    - if the server remembers the keys of the session, they can resume
  - with session ticket:
    - at the end of the handshake, the server gives a ticket to the client
    - it contains the encrypted session information
    - the client presents the ticket of the last session
    - if the server can decrypt the ticket, they can resume

TSL handshake

Interesting facts:

- Server Name Indication SNI
  - Often, several web servers are located on the same IP address
  - 24heures.ch and tagesanzeiger.ch are both on IP 151.252.10.121
  - They have different certificates (*.24heures.ch and *.tagesanzeiger.ch)
  - The browser uses the `Host` header to tell the server which website he means

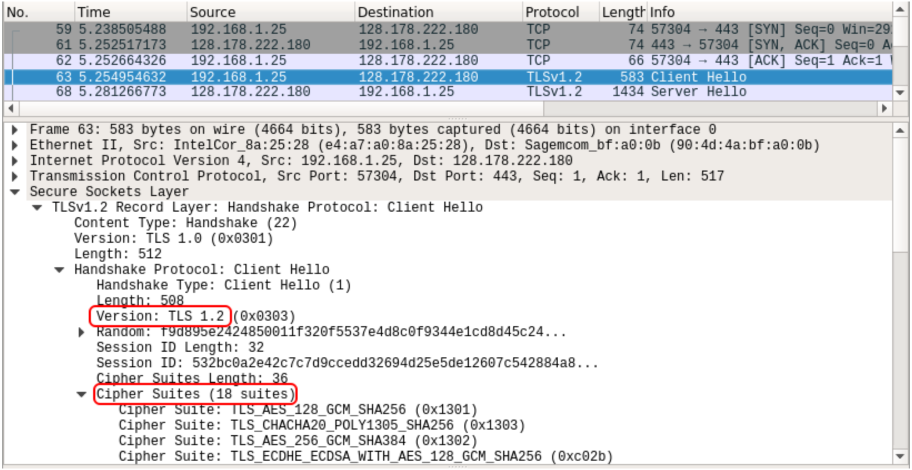
```
GET / HTTP/1.1
Host: www.24heures.ch
```

  - The host header is only sent after the handshake
  - How does the server know which certificate to send **?**

TSL handshake SNI

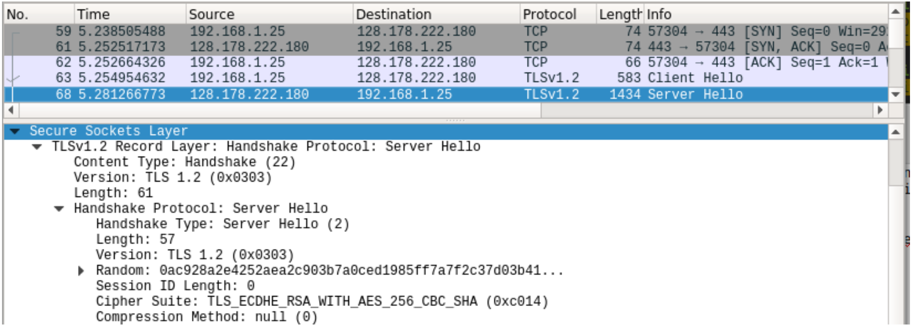
- Server Name Indication SNI
  - The SNI extension allows the broswer to tell the TLS layer for which hostname it is trying to establish a connection
  - The client adds this info to the Client Hello $\rightarrow$ the server knows which certificate to send
  - As this is not encrypted, an eavesdropper will know which site your are connecting to

Demo: client hello



- The client declares TLS version 1.2 (highest version),
- Itproposes 18 cipher suites

Demo: server hello



- The server chooses TLS 1.2
- The server chooses `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA`

⇒ we have perfect forward secrecy (PFS), good server!

Demo: server certificate

No.	Time	Source	Destination	Protocol	Length	Info
69	5.281370980	192.168.1.25	128.178.222.180	TCP	66	57304 → 443 [ACK] Seq=518 Ack=
70	5.281415243	128.178.222.180	192.168.1.25	TCP	98	443 → 57304 [PSH, ACK] Seq=136
71	5.281424124	192.168.1.25	128.178.222.180	TCP	66	57304 → 443 [ACK] Seq=518 Ack=
72	5.281440118	128.178.222.180	192.168.1.25	TLSv1.2	1430	Certificate [TCP segment of a
73	5.281448450	192.168.1.25	128.178.222.180	TCP	66	57304 → 443 [ACK] Seq=518 Ack=

```
▸ Frame 72: 1430 bytes on wire (11440 bits), 1430 bytes captured (11440 bits) on interface 0
▸ Ethernet II, Src: Sagemcom_bf:a0:0b (90:4d:4a:bf:a0:0b), Dst: IntelCor_8a:25:28 (e4:a7:a0:8a:25:28)
▸ Internet Protocol Version 4, Src: 128.178.222.180, Dst: 192.168.1.25
▸ Transmission Control Protocol, Src Port: 443, Dst Port: 57304, Seq: 1401, Ack: 518, Len: 1364
▸ [3 Reassembled TCP Segments (2639 bytes): #68(1302), #70(32), #72(1305)]
▾ Secure Sockets Layer
  ▾ TLSv1.2 Record Layer: Handshake Protocol: Certificate
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 2634
    ▾ Handshake Protocol: Certificate
        Handshake Type: Certificate (11)
        Length: 2630
        Certificates Length: 2627
      ▾ Certificates (2627 bytes)
          Certificate Length: 1261
        ▸ Certificate: 308204e9308203d1a003020102021462307 20f8dad716cfe... (id-at-commonName=*.epfl.ch,id-a…
          Certificate Length: 1360
        ▸ Certificate: 3082054c30820334a00302010202144898 2de2a92cb339e1... (id-at-commonName=QuoVadis Globa…
```

- The server sends it certificate
- …and also the intermediate certificate of QuoVadis
- So the client can follow the chain up to the QuoVadis root certificate

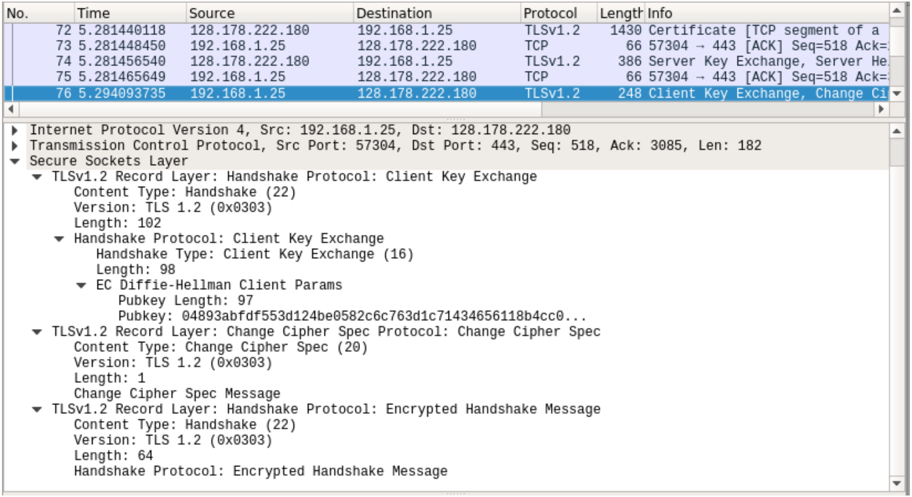
Demo: server key exchange

No.	Time	Source	Destination	Protocol	Length	Info
72	5.281440118	128.178.222.180	192.168.1.25	TLSv1.2	1430	Certificate [TCP segment of a
73	5.281448450	192.168.1.25	128.178.222.180	TCP	66	57304 → 443 [ACK] Seq=518 Ack=
74	5.281456540	128.178.222.180	192.168.1.25	TLSv1.2	386	Server Key Exchange, Server He
75	5.281465649	192.168.1.25	128.178.222.180	TCP	66	57304 → 443 [ACK] Seq=518 Ack=
76	5.294093735	192.168.1.25	128.178.222.180	TLSv1.2	248	Client Key Exchange, Change Ci

```
▶ Frame 74: 386 bytes on wire (3088 bits), 386 bytes captured (3088 bits) on interface 0
▶ Ethernet II, Src: Sagemcom_bf:a0:0b (90:4d:4a:bf:a0:0b), Dst: IntelCor_8a:25:28 (e4:a7:a0:8a:25:28)
▶ Internet Protocol Version 4, Src: 128.178.222.180, Dst: 192.168.1.25
▶ Transmission Control Protocol, Src Port: 443, Dst Port: 57304, Seq: 2765, Ack: 518, Len: 320
▶ [2 Reassembled TCP Segments (370 bytes): #72(59), #74(311)]
▼ Secure Sockets Layer
  ▼ TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 365
    ▼ Handshake Protocol: Server Key Exchange
        Handshake Type: Server Key Exchange (12)
        Length: 361
      ▼ EC Diffie-Hellman Server Params
          Curve Type: named_curve (0x03)
          Named Curve: secp384r1 (0x0018)
          Pubkey Length: 97
          Pubkey: 0487b1262fc8a2697c607fcaf98c712fe86a038f49acf5f5...
        ▶ Signature Algorithm: rsa_pkcs1_sha256 (0x0401)
          Signature Length: 256
          Signature: 9c20e8ab890f145ba4fdaf43f741334b8a09066addea62df...
▼ Secure Sockets Layer
  ▼ TLSv1.2 Record Layer: Handshake Protocol: Server Hello Done
      Content Type: Handshake (22)
```

- The server sends its part of the elliptic curve DH (pubkey)
- It signs to prove it knows the private key of the certificate
  - proves we are really talking to the holder of the certificate

Demo: client key exchange



- The client sends its part of the DH (pubkey)
- It sends 'Change Cipher Spec'
- It sends the 1st encrypted message (the Finished message)

Demo: server change cipher spec

No.	Time	Source	Destination	Protocol	Length	Info
72	5.281440118	128.178.222.180	192.168.1.25	TLSv1.2	1430	Certificate [TCP segment of a
73	5.281448450	192.168.1.25	128.178.222.180	TCP	66	57304 → 443 [ACK] Seq=518 Ack=
74	5.281456540	128.178.222.180	192.168.1.25	TLSv1.2	386	Server Key Exchange, Server He
75	5.281465649	192.168.1.25	128.178.222.180	TCP	66	57304 → 443 [ACK] Seq=518 Ack=
76	5.294093735	192.168.1.25	128.178.222.180	TLSv1.2	248	Client Key Exchange, Change Ci

```
▶ Internet Protocol Version 4, Src: 192.168.1.25, Dst: 128.178.222.180
▶ Transmission Control Protocol, Src Port: 57304, Dst Port: 443, Seq: 518, Ack: 3085, Len: 182
▼ Secure Sockets Layer
  ▼ TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 102
    ▼ Handshake Protocol: Client Key Exchange
        Handshake Type: Client Key Exchange (16)
        Length: 98
      ▼ EC Diffie-Hellman Client Params
          Pubkey Length: 97
          Pubkey: 04893abfdf553d124be0582c6c763d1c71434656118b4cc0...
  ▼ TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
      Content Type: Change Cipher Spec (20)
      Version: TLS 1.2 (0x0303)
      Length: 1
      Change Cipher Spec Message
  ▼ TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 64
      Handshake Protocol: Encrypted Handshake Message
```

- The server sends ‘Change Cipher Spec’
- It sends the 1st encrypted message (the Finished message)

Demo: wrap-up

- We've just applied all the crypto we saw today:
- Public key crypo:
  - The CA has signed the server's certificate (and its own root certificate)
  - The server signs the key exchange with RSA, to prove it holds the private key
  - Elliptic curve Diffie Hellman is used to exchange a symmetric key
- Symmetric crypto
  - AES block cipher is used in CBC mode for encryption
  - SHA hash is used for HMAC, for key derivations

Historical weaknesses of TLS

- Downgrade attacks (by a man-in-the-middle)
  - trick server into using an insecure version of TLS
  - trick server into using weak keys (Freak, 2014)
  - trick server into downgrading the DH parameters (Logjam, 2015)
- Poodle, 2014: downgrade TLS 1.0 to SSL3, then padding oracle attack
- Crime, Breach: exploit vulnerability when compression is used
- Lucky thriteen, 2013: break TLS 1.2 with padding oracle attack
- Bugs in implementations:
  - 2014: Heartbleed in OpenSSL: leaks random 64k bytes of server memory
  - 2017: Cloudbleed: bug in Cloudfalre HTML parsers allowed people to read data of other Cloudflare customers

News in TLS 1.3

Main differences:

- Old, unsafe cipher suites have been removed
- The remaining ones all use Authenticated Encryption with Associated Data
- Compression has been removed
- Handshake
  - can be shorter in some cases
  - is partially encrypted (SNI is no more in clear text)
- Key exchange is always forward secure

Implementing TLS

- Two major ways of implementing TLS

- Use a new name and port (HTTPS, LDAPS, IMAPS...):
  - HTTP on port 80 → HTTPS on 443
  - Start with a TLS handshake, security is mandatory
  - Not compatible with client that can't TLS

- Use the STARTTLS command on the standard protocol
  - e.g ESMTP (extended SMTP) on port 25
  - client types `STARTLS` if it wants to start a handshake
  - *Opportunistic* encryption, no guarantees
  - MITM can pretend `STARTTLS` is not supported

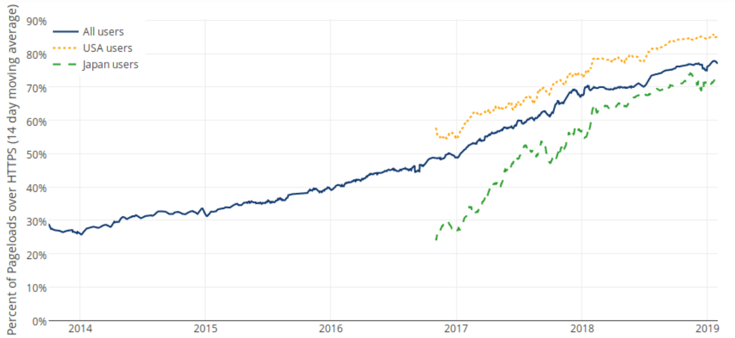
```
220 phil1.ethz.ch ESMTP Sat,
EHLO mail.example.com
250-phil1.ethz.ch Hello adsl
250-SIZE 52428800
250-8BITMIME
250-PIPELINING
250-STARTTLS
250 HELP
STARTTLS
220 TLS go ahead
...........;.FX..E.E..,..4..
...9.        ...3.....=.<.5./.
.
...........#...........
. ..........................
```

Deploying HTTPS in the Internet

TLS, certs, trust

Increasing Usage of HTTPS

- If all sites used HTTPS it would be better for Privacy and Security
- Since 2014, Google has started ranking HTTPS websites better than HTTP in search results
  - motivating the use of HTTPS
- Since July 2018, Google chrome labels HTTP web sites as not secure:



Let's encrypt, free certificates

- To be able to create certificates that are trusted by all browsers, you must undergo a costly certification
  - Prove that you protect your private keys
  - Prove that you diligently validate the identity of your customers
  - ...
- The Internet Security Research Group (not for profit) found enough sponsor to certify a fully automated CA that gives certificates for free!
- IT is called Let's Encrypt
- To obtain a certificate, you must place specific data
  - in a file on your web server, or
  - in a DNS entry of your domain.
- This can be fully automated: no excuse for not using TLS

Attacks on HTTPS and defense

TLS, certs, trust

SSL stripping

- A MITM makes you believe that the site uses HTTP, not HTTPS
  - when you type a URL, your browser first connects using HTTP
    - the server sends a redirect to HTTPS
    - the MITM doesn't show you the redirect, your browser continues to use HTTP
  - or, he replaces HTTPS with HTTP in the links of the pages that you visit

➡ You connect to the MITM with HTTP, he connects to the site with HTTPS

- You have no alert, as your browser doesn't know that you should be using HTTPS

HSTS

- HTTP Strict Transport Security (HSTS) **RFC 6707**
- The server sends an HTTP header indicating you must always use HTTPS

```
Strict-Transport-Security: max-age=63072000; includeSubDomains
```

- `max-age`: the browser will remember to connect directly by HTTPS for one year (63072000 seconds)
- `includeSubDomains`: this is true for all subdomains of this domain

HSTS preload list

- If the MITM intercepts your very first connection to the site, he can hide the redirection to HTTPS
  - the client will never see the HTSS header.

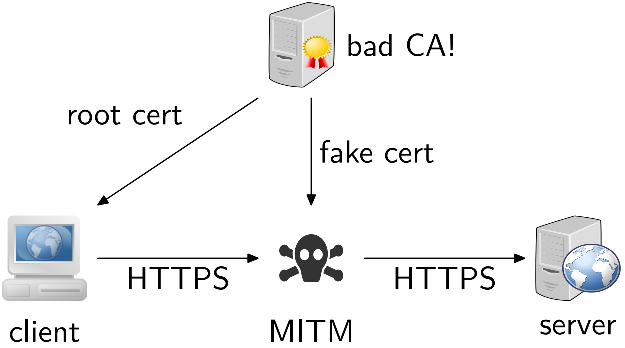
➡ You can request that your domain be added to the HSTS-preload list.

- All major browser have a copy the that list (**chromium.org**) and never connect by HTTP to the domais in the list.

Untrustworthy CAs

- A trusted CA (the root cert is in your browser) can give the MITM a trusted certificate in the name of the server

➡ The server can intercept the traffic without you knowing



Untrustworthy CAs

- Reasons for a CA to hand out fake certs
  - The CA has been hacked (2011 Comodo was hacked, certificates were generated for www.google.com, login.yahoo.com, login.skype.com ...)
    - the certificates were revoked
  - The CA is your company's CA.
    - Your company wants to intercept all traffic to detect malware.
    - They have inserted the company's root cert into your browser
  - The CA is 'experimenting' and not following the guidelines
    - in 2015 Symantec (owners of VersiSign, Thawte, Equifax, GeoTrust, RapidSSL) issued fake certificates for Google and other companies.
    - In 2018 Google blocked Symantec's root certificates in Chrome
    - Symantec sold its CA business to DigiCert
  - The government may have requested the certs in order to spy on its citizens (Syria, China, India, France, ...)

Protections against untrustworthy CAs

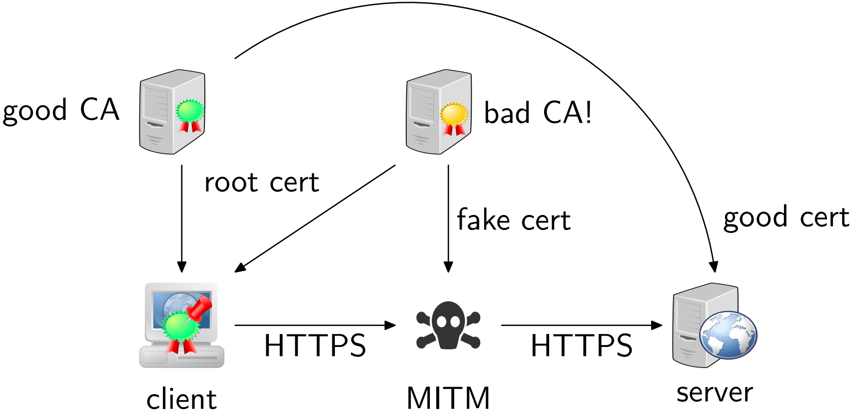
- Certificate pinning
  - client-side list of trusted certificates
- Certificate transparency
  - public list of certificates
- Dane
  - Certificates published in the DNS
- CAA
  - Official CA of domain published in DNS

Certificate pinning

- The developer of the client application (e.g. smartphone app), stores certificate of a trusted root CA, or intermediate CA in the client

- If the server shows a certificate which is not signed by this pin, it does not accept to connect.
- E.g. a mobile e-banking application only trusts certificates singed by an intermediate CA of the bank.
- There was a proposal to enable pinning in browsers with an HTTP header (HKPK) but it did not work out.

Certificate pinning



- The client knows that the server cert must be signed by the pinned cert from the good CA.

➡ It does not accept the fake cert, even if it is signed by a 'trusted' CA

DANE

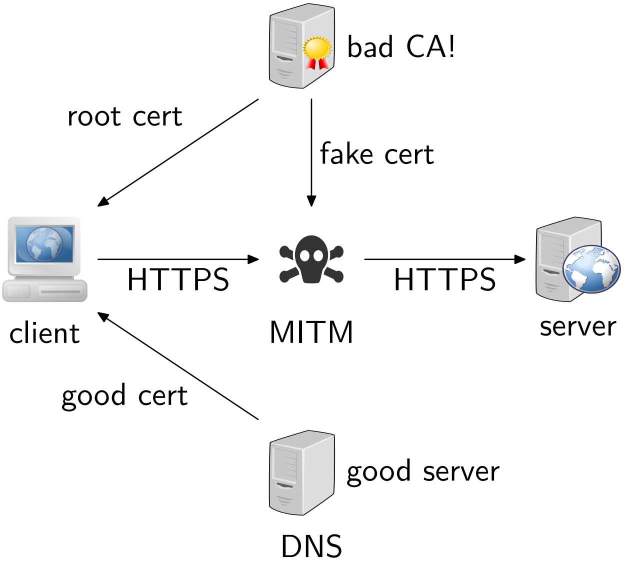
- DNS based Authentication of Named Entities uses the DNS instead or additionally to CAs

- DNS Sec must be activated for this to work

- DNSSec for hackers:
  - DNS servers sign all the responses they give
  - Their key is signed by the server above them in the hierarchy
    - e.g. epfl (.epfl.ch) $\rightarrow$ switch (.ch) $\rightarrow$ DNS root
  - The public key of the root is known to everybody

- When connecting to a server, a client can compare the certificate received from the server with the one in the DNS
  - if they differ, the connection is refused

DANE


bad CA!
root cert
fake cert
client
HTTPS
MITM
HTTPS
server
good cert
good server
DNS

DANE

- DANE is not very useful for HTTPS
  - most users do not have a working DNSSec client on their machine

- DANE is very useful for e-mail transfers between SMTP servers
  - an MITM could make you believe that the other server does not support STARTTLS
  - Also, if the mail server hosts many different domains, the domain of the server might not correspond to the domain of the e-mail recipient
    - how would you know if you can trust it ?
  - With DNSSec the sender is sure to connect to the correct server
  - With DANE the sender can query DNS to know if and with what cert the server does TLS
  - It is simpler to deploy DNSsec on all mail servers than on all web clients (DANE is simper for SMTP than for HTTPS)

DANE example

- Let's use DANE to ask for the certs used by `mailbox.org` on its HTTPS port (443)

```
$ dig _443._tcp.mailbox.org ANY

;_443._tcp.mailbox.org.           IN     ANY
_443._tcp.mailbox.org.      3290  IN     TLSA   3 1 1 29681D7841...
_443._tcp.mailbox.org.      3290  IN     TLSA   3 1 1 E41CC76330...
_443._tcp.mailbox.org.      3290  IN     TLSA   3 1 1 4758AF6F02...
_443._tcp.mailbox.org.      3290  IN     TLSA   3 1 1 51E89750F7...
_443._tcp.mailbox.org.      3290  IN     RRSIG  TLSA 7 4 3600
20190331082501 20190301074329 5719 mailbox.org. ZL3XdQnBQJjDjJTMVo...
_443._tcp.mailbox.org.      3290  IN     RRSIG  TLSA 10 4 3600
20190331082501 20190301074329 48028 mailbox.org. TtB1MHQ3keMKNppj...
```

- TLSA entries: SHA256 hash of certificate public key info
- RRSIG entries: DNSSec signatures

CAA

- Certification Authority Authorization (CAA) uses DNS to declare which CAs are allowed to deliver certificates for a domain
- It is used by the CAs to verify that a request for a certificate is legitimate
  - CAs will refuse to sell a cert if there is a CAA entry for the domain an they are not listed in that entry
  - Browser do not use CAA
- All CAs that are included in browsers already adhere to CAA
- Prevents attackers from fraudulently requesting certificates
- Does not prevent fraudulent creation of certificates by CAs

CAA example

- Let's check who is allowed to issue certs for `google.com`

```
$ dig google.com CAA

;  DiG 9.11.3-1ubuntu1.3-Ubuntu  google.com CAA
;; global options: +cmd
;google.com.                    IN      CAA
google.com.             86400   IN      CAA     0 issue "pki.goog"
```

- certificates for the domaine `google.com` can only be issued by the CA that has issued the cert for `pki.goog` (Google Internet Authority G3)

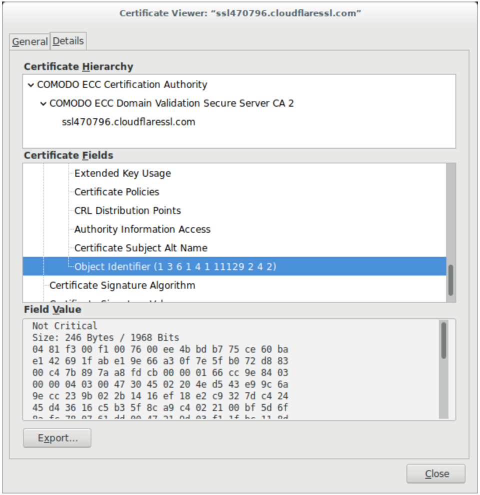
Certificate Transparency CT

Finally something that works...

- Public signed logs of certs used in the Internet
- CAs submit all the certs they generate
- Servers can verify if the logs contain certs that they did not request
  ➡ somebody else requested a cert for their domain!
- Clients can verify that a cert receive from a server is in the logs
- When a cert is added to a log, the log generates a Signed Certificate Timestamp (SCT)
  - the web server can give a copy of the SCT to the client
  - the client doesn't need to lookup the certificate in the log

➡ Very easy to detect fraudulous certificates

CT timestamp (SCT) example



- most browser don't know about the SCT field in certificates. They display its numeric identifier (1.3.6.1.4.1.11129.2.4.2)

Certificate Transparency Demo

- There is a set of public logs that you can query separately
  - you can find a list at **here**
- Or you can use the site **crt.sh** to search in the logs:

crt.sh Identity Search Group by Issuer

Criteria	Identity LIKE '%.epfl.ch'

crt.sh ID	Logged At ⇧	Not Before	Not After	Identity	Issuer Name
1170013914	2019-02-04	2019-02-04	2020-02-04	sbsstsrv2.epfl.ch	C=BM, O=QuoVadis Limited, CN=QuoVadis Global SSL ICA G2
1170013914	2019-02-04	2019-02-04	2020-02-04	sccsrv.epfl.ch	C=BM, O=QuoVadis Limited, CN=QuoVadis Global SSL ICA G2
1169704396	2019-02-04	2019-02-04	2021-02-04	objsuperv01.epfl.ch	C=BM, O=QuoVadis Limited, CN=QuoVadis Global SSL ICA G2
1169703462	2019-02-04	2019-02-04	2020-02-04	dsps-lhd.epfl.ch	C=BM, O=QuoVadis Limited, CN=QuoVadis Global SSL ICA G2
1171047336	2019-02-04	2019-02-04	2019-05-05	garzoni.dhlab.epfl.ch	C=US, O=Let's Encrypt, CN=Let's Encrypt Authority X3
1171047866	2019-02-04	2019-02-04	2019-05-05	garzoni-dev.dhlab.epfl.ch	C=US, O=Let's Encrypt, CN=Let's Encrypt Authority X3
1170361305	2019-02-02	2019-02-02	2019-05-03	dhlabsrv22.epfl.ch	C=US, O=Let's Encrypt, CN=Let's Encrypt Authority X3
1168006658	2019-02-02	2019-02-02	2019-05-03	dhlabsrv22.epfl.ch	C=US, O=Let's Encrypt, CN=Let's Encrypt Authority X3
1169735748	2019-02-02	2019-02-02	2019-05-03	croque.epfl.ch	C=US, O=Let's Encrypt, CN=Let's Encrypt Authority X3
1167700771	2019-02-02	2019-02-02	2019-05-03	croque.epfl.ch	C=US, O=Let's Encrypt, CN=Let's Encrypt Authority X3
1161705474	2019-01-31	2019-01-31	2020-01-31	stagcompanion.epfl.ch	C=BM, O=QuoVadis Limited, CN=QuoVadis Global SSL ICA G2
1161583417	2019-01-31	2019-01-31	2021-01-31	dacodeck.epfl.ch	C=BM, O=QuoVadis Limited, CN=QuoVadis Global SSL ICA G2
1161583417	2019-01-31	2019-01-31	2021-01-31	spssrv1.epfl.ch	C=BM, O=QuoVadis Limited, CN=QuoVadis Global SSL ICA G2
1156560390	2019-01-30	2019-01-30	2019-04-30	mail.parsa-new.epfl.ch	C=US, ST=TX, L=Houston, O="cPanel, Inc.", CN="cPanel, Inc. Certification Authori

source: **crt.sh**

Conclusions & Questions

Conclusions

- TLS authenticates the server (and possible the client) and protects confidentiality and integrity of data using symmetric and asymmetric crypto
- A Public Key Infrastructure distributes public keys using certificates
- This does not work on the Internet, because we can not trust 150 CAs
- HSTS, Certificate transparency and CAA protect against MITM and fraudoulus CAs
- Certificate pinning helps even more, but needs some manual setup
- For the moment, DANE is useful to turn opportunistic encryption in SMTP into trusted and secure encryption

Questions

- How can you find out all the cipher suites supported by a server ?
- Why is PFS important ?
- Why is DANE not useful without DNSsec ?
- If the government forces your CA to create a fake cert for spying on your users, who can help you the best:
  - CAA ?
  - Certifcate transparency ?
  - DANE ?

Questions

- The problem with TLS certificates is that we can not trust 150 CAs that are known to our browsers
  - Certificate Transparency is supposed to solve this issue using tens of certificate logs
  - Why can we trust tens of logs better than 150 CAs ?