Manolis (Emmanouil Vasilomanolakis)

network security: TLS & Certificates

Course plan

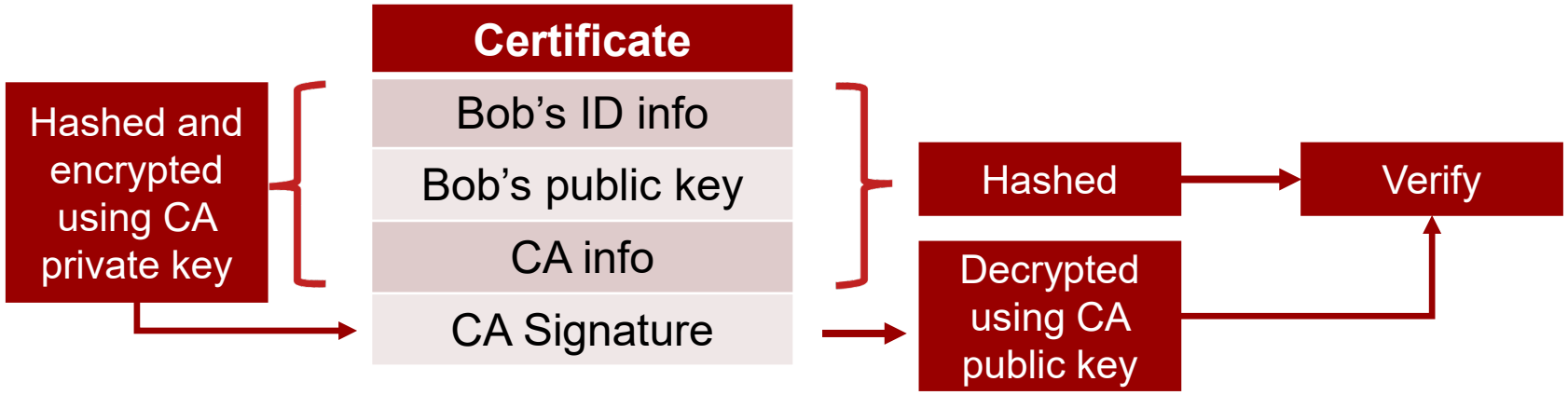
- Lecture 1: Intro (**Manolis**, **Carsten**) 30.01
- Lecture 2: **Crypto** essentials (**Carsten**) 06.02
- Lecture 3: **Authentication** (**Manolis**), lab bootcamp (TAs) 13.02
- Lecture 4: **TLS** (**Manolis**) 20.02
- Lecture 5: **Threat detection** (**Manolis**) 27.02
- Lecture 6: Hacking Lab day (**TAs**) – blue team 05.03
- Lecture 7: **IoT security** (**Manolis**) 12.03
- Lecture 8: **WIFI security** (**Manolis**) 19.03
- Lecture 9: **Private communication** (**Carsten**) 02.04
- Lecture 10: **When everything fails** (**Manolis**) 09.04
- Lecture 11: Hacking Lab day (**TAs**) – red team 16.04
- Lecture 12: Guest lecture (OT security, **Ludwig**) 23.04
- Lecture 13: **Exam preps** (**Carsten, Manolis**) 30.04

Outline

- **Introduction**
- **Certificates**
- **TLS**
  - What it can offer
  - TLS protocol
  - TLS resumption
- **Beyond HTTP**
- **Attacks on TLS**
- **Lab exercises**

Certificates

- A **certificate** is a document that links someone's identity to their public key
  - Certificates are issued and signed by trusted **Certification Authorities** (CA)
    - A CA is a third party that **the community trusts** (e.g. government agency)
  - **X.509** is the universally accepted standardised format for certificates



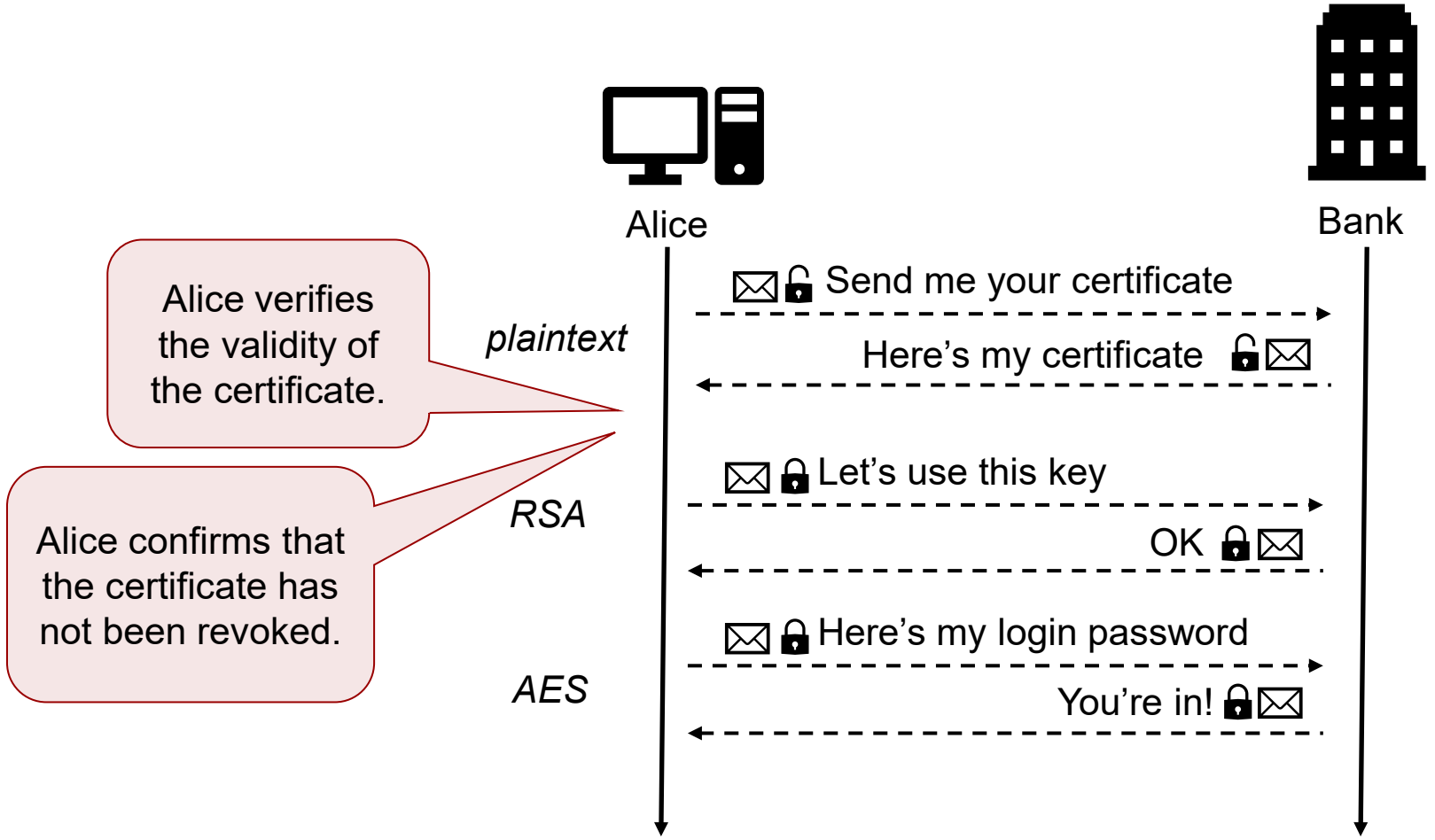
- Knowing the **public key of the CA** is sufficient to verify all the certificates that it issues
- No party except for the CA can **modify** the certificate without being detected

Certificate Lifetime

- Certificates have a **period of validity** (like credit cards)
  - "Not before" and "not after" fields
  - If expired then the certificate is invalid
    - A new certificate is issued with new public key

- Revocation of Certificates
  - Revocation of non-expired certificates may be needed if…
    - Private key gets compromised
    - Vital user info changes
    - The CA gets compromised
  - A CA keeps a **Certificate Revocation List (CRL)**
    - Signed with the CA's private key

Public-Key Certificates

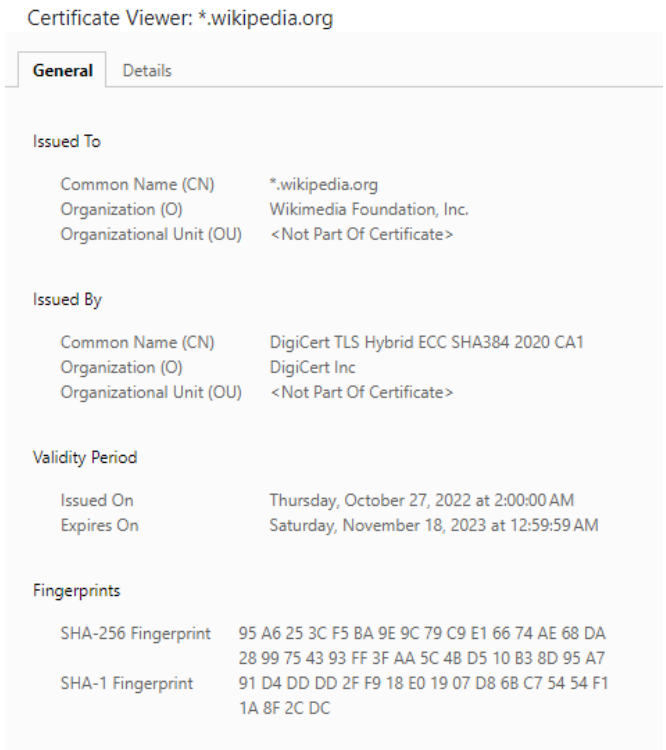


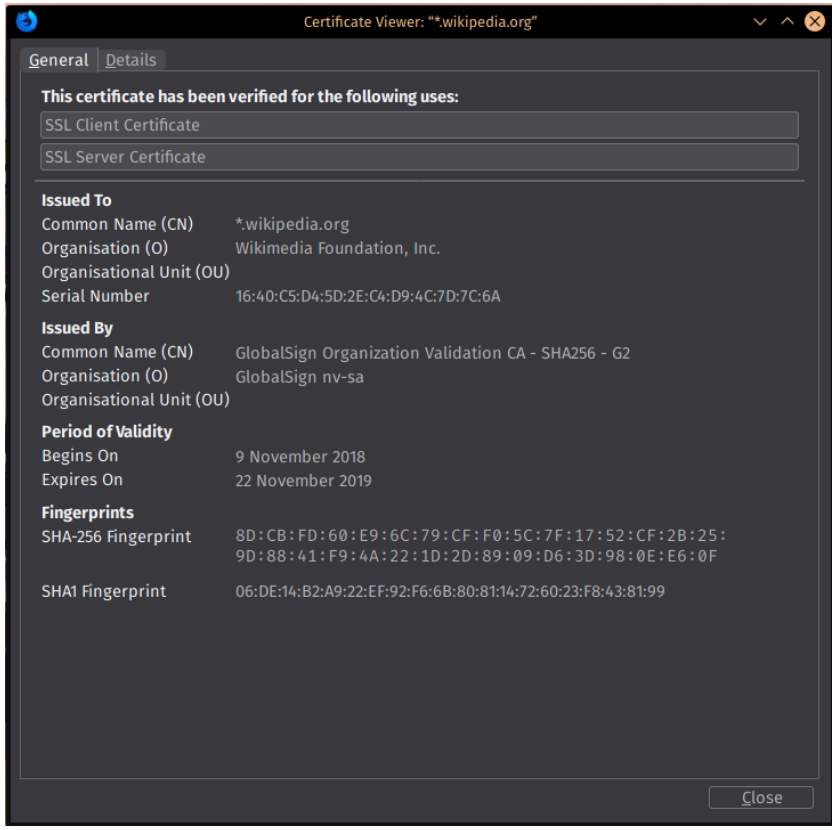
Certificates

- A piece of information that proves the identity of a public-key's owner.
  - Signed and delivered securely by a trusted third party entity called a Certificate Authority (CA)
- A Certificate contains
  - CA's identity
  - Owner's identity
  - Owner's **public-key**
  - Certificate expiry date
  - CA's signature of that certificate
  - …
- With a certificate instead of a public key
  - A recipient can verify a few things about the issuer

x.509 certificates

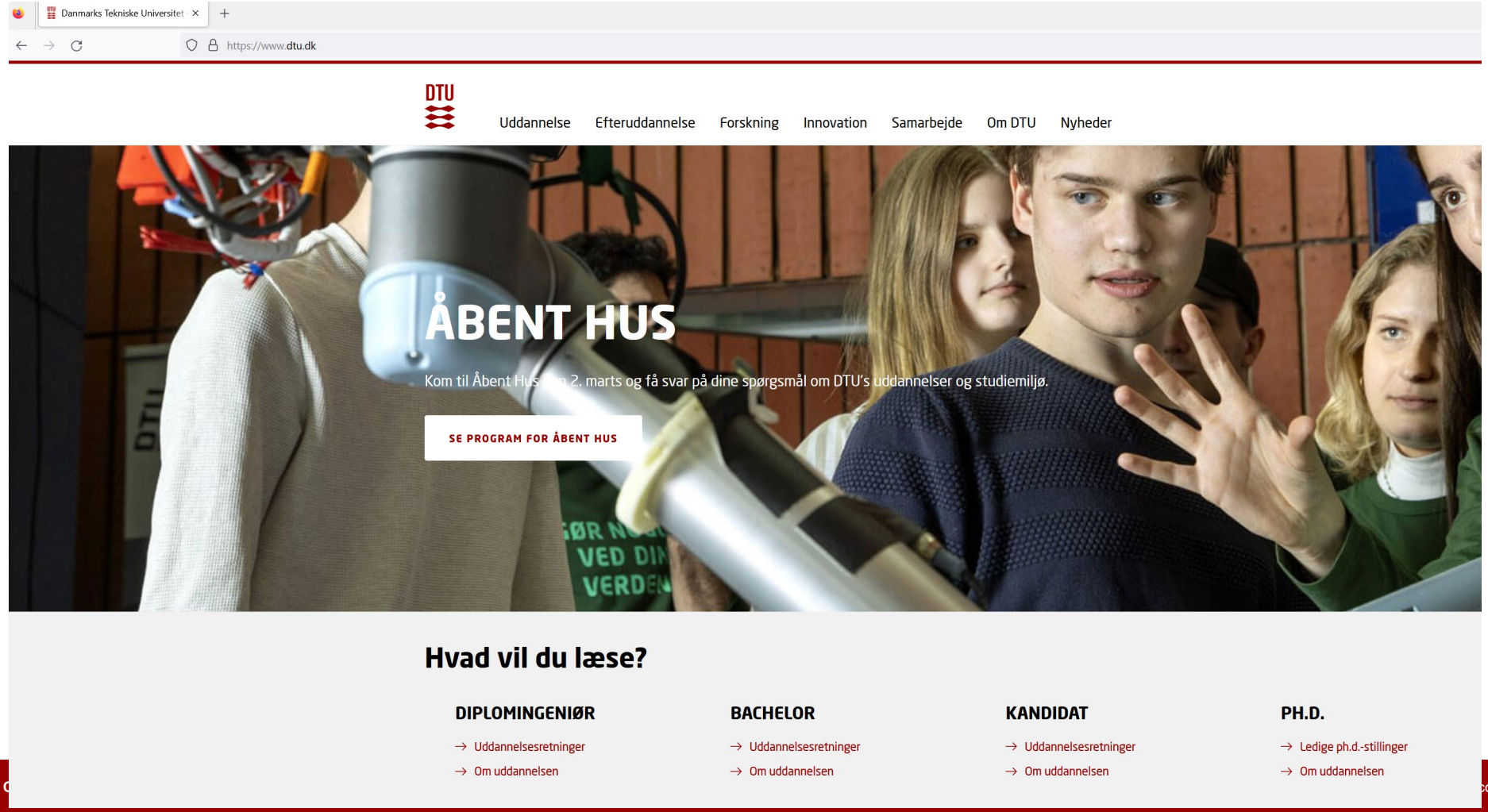
- A standard for defining the format of a public key certificate
- The structure of an X.509 v3 digital certificate is as follows:
  - Certificate
    - Version Number
    - Serial Number
    - Signature Algorithm ID
    - Issuer Name
    - Validity period
      - Not Before
      - Not After
    - Subject name
    - Subject Public Key Info
      - Public Key Algorithm
      - Subject Public Key
    - Issuer Unique Identifier (optional)
    - Subject Unique Identifier (optional)
    - Extensions (optional)
      - ...
  - Certificate Signature Algorithm
  - Certificate Signature





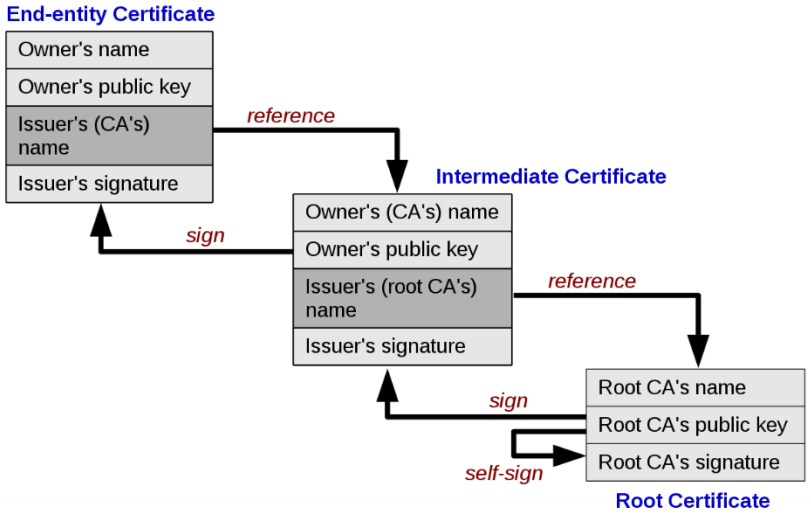
What is this website?

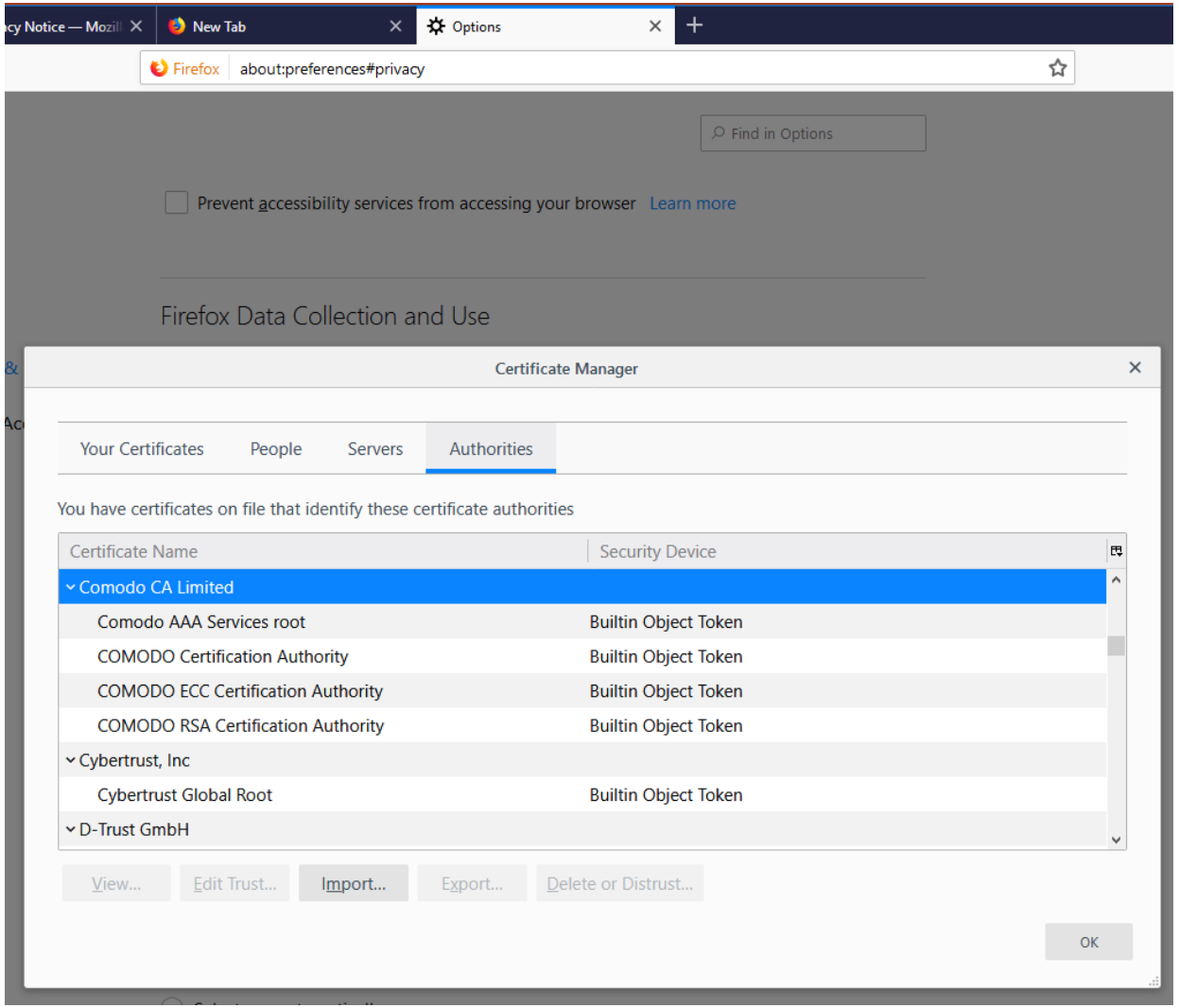
- How do you know?



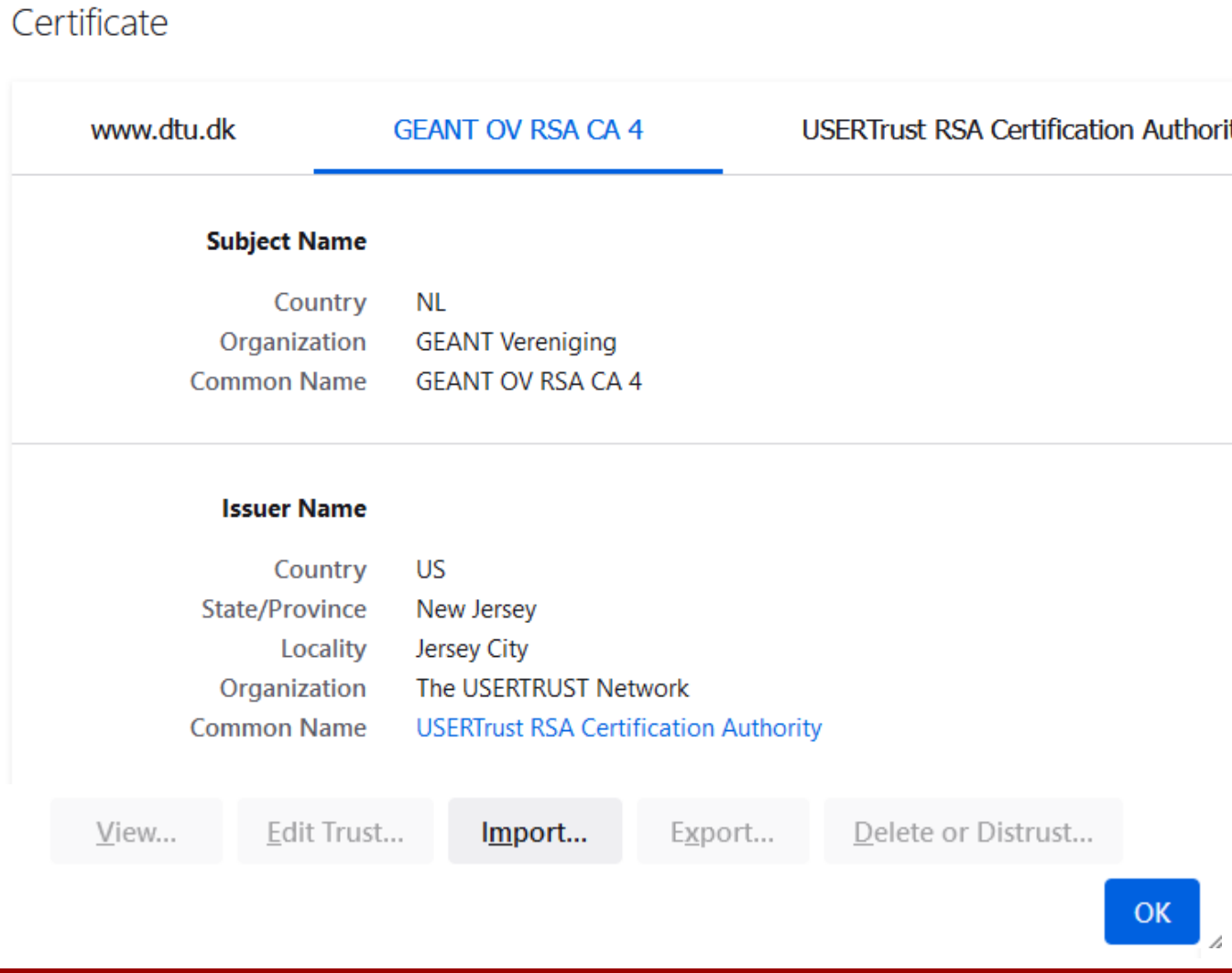
Root certificates

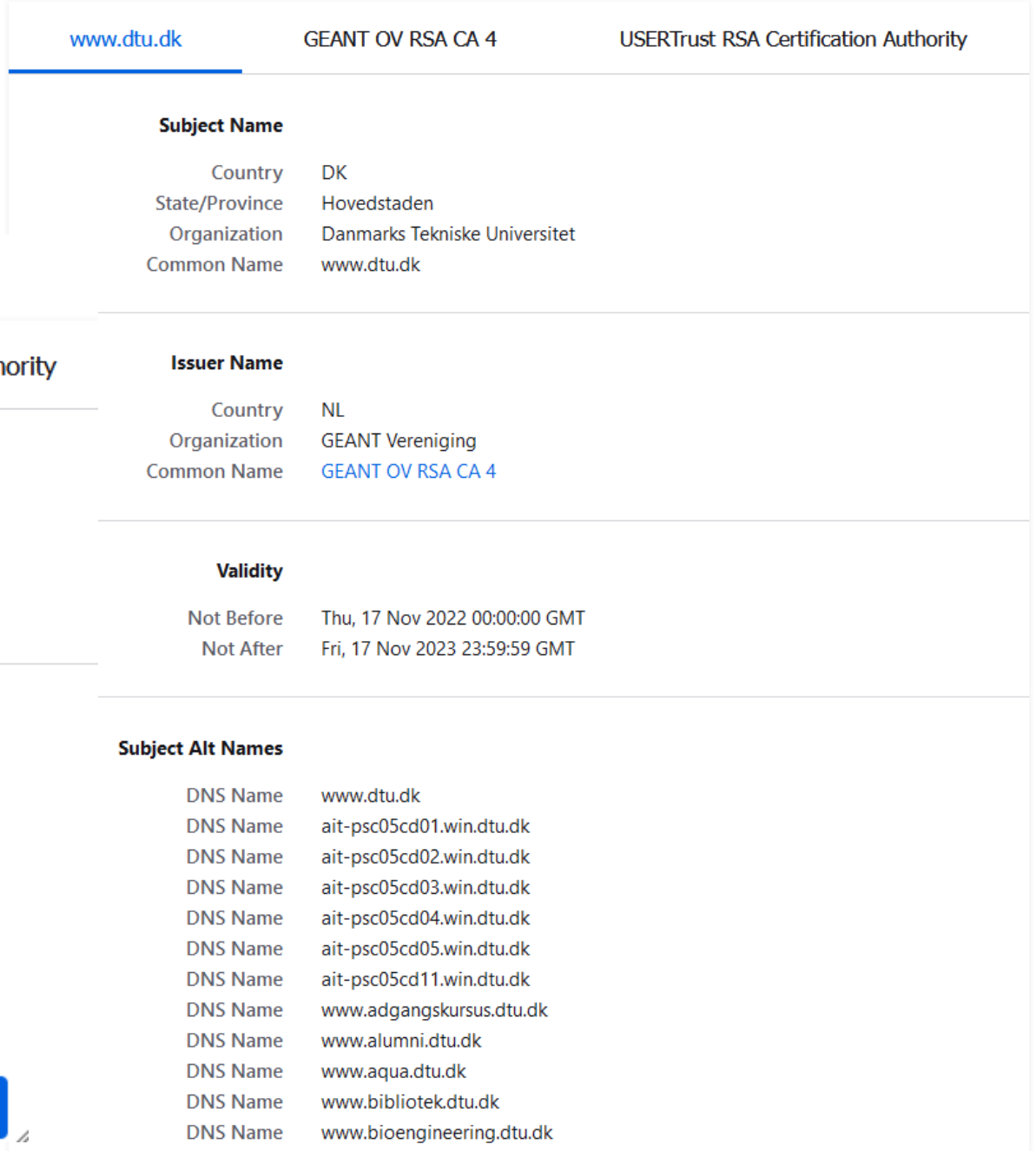
- Root/trusted certificates are scary





Dtu.dk certificate



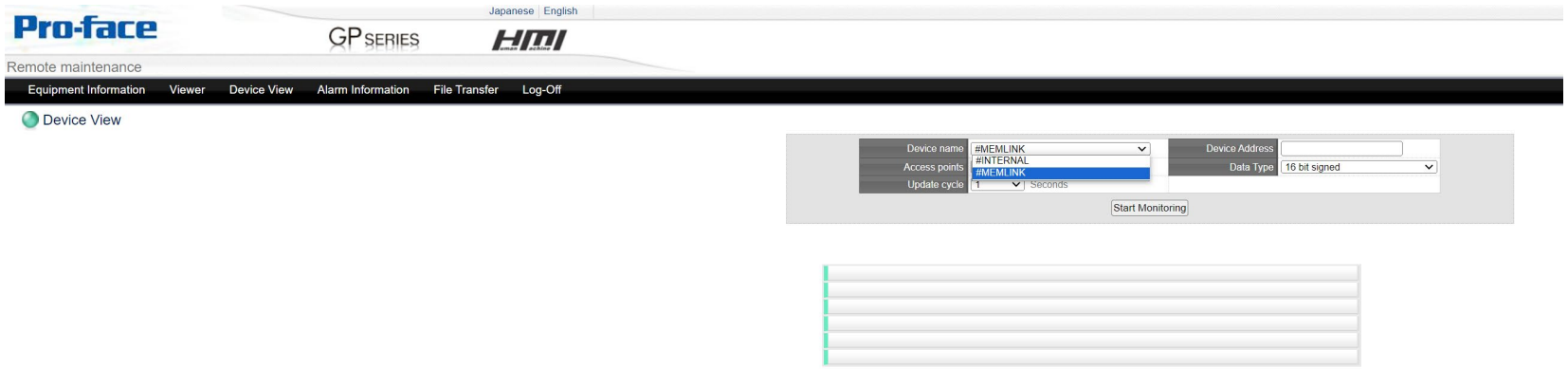


Certificates and expiration…

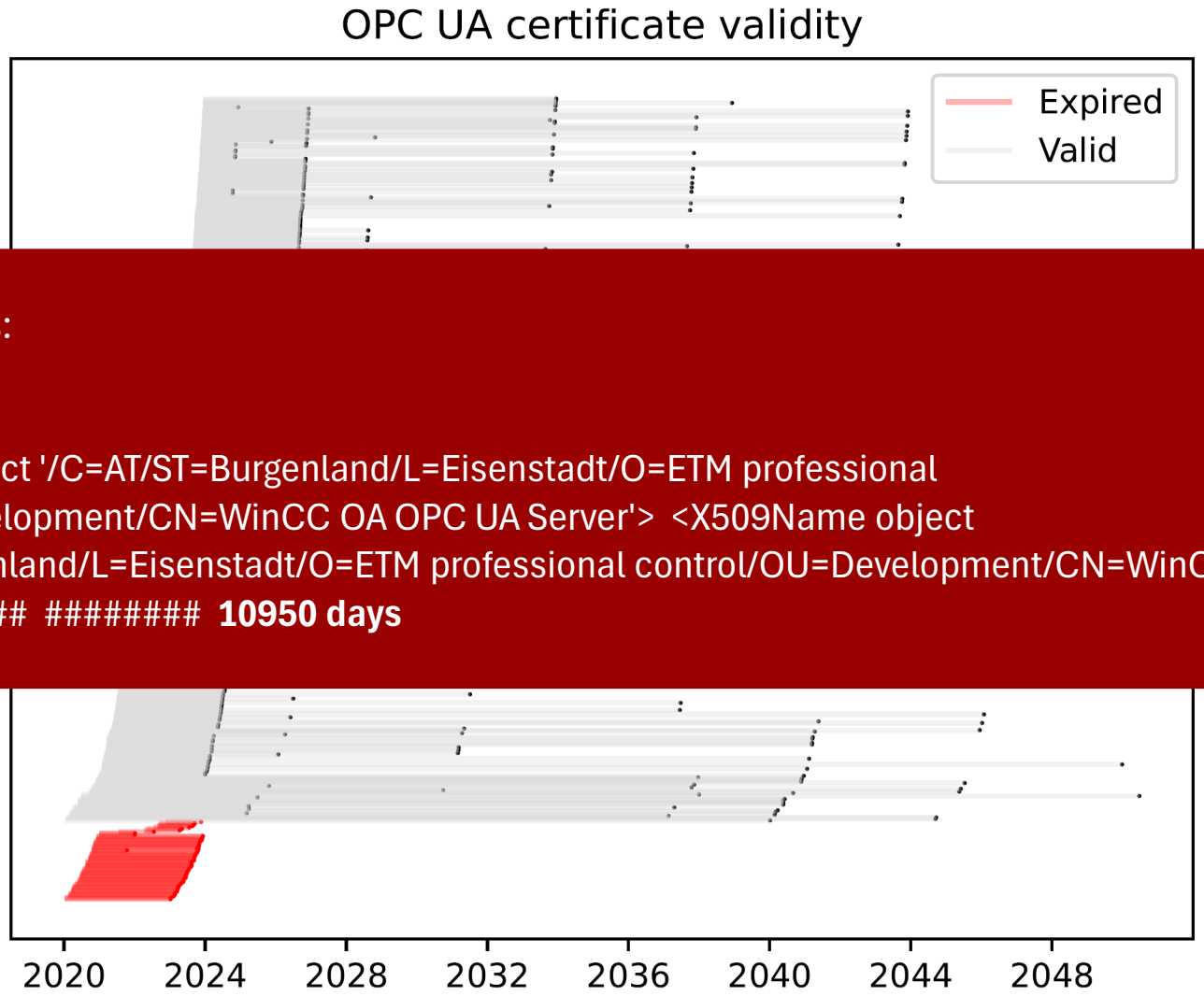
- Expiring certificates are a pain
- It is very tempting to create **certificates** that will **last forever**
- **BAD** practice!

An example of bad certificate practice

- We have been running an experiment:
  - Scan the whole Internet (IPv4 address space)
  - Focus on an industrial automation protocol: OPC UA (Unified Architecture)
    - Port 4840
  - Commonly used for device-to-device communication



Certificates and expiration…



And the **winner** is:

<X509Name object '/C=AT/ST=Burgenland/L=Eisenstadt/O=ETM professional control/OU=Development/CN=WinCC OA OPC UA Server'> <X509Name object '/C=AT/ST=Burgenland/L=Eisenstadt/O=ETM professional control/OU=Development/CN=WinCC OA OPC UA Server'> ######## ######## **10950 days**

CA attacks

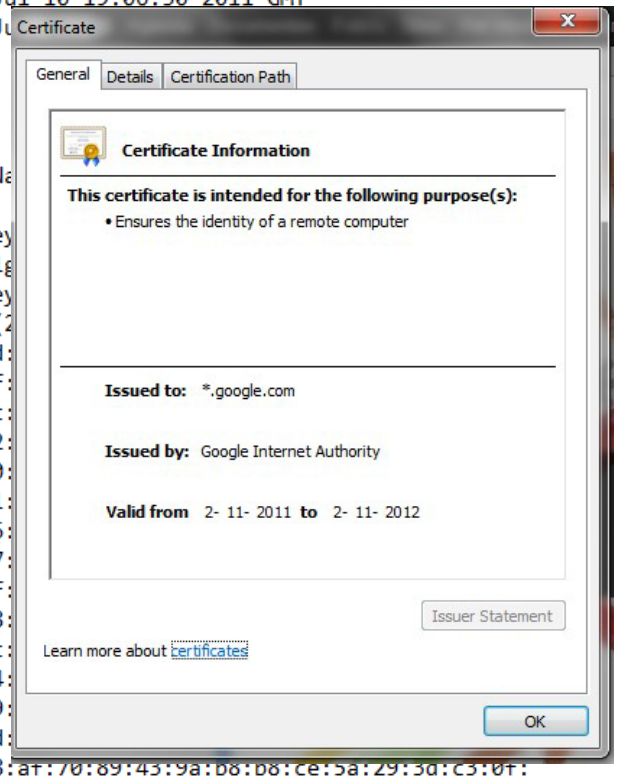
- There is no cryptography to protect us from a CA compromise
- Only option: stringent auditing and scrupulous OpSec
- Not a theoretical threat
- Nightmare scenario: CA compromise that remains not publicly known.
- Example: DigiNotar
  - Summer 2011: Hacker gains access to DigiNotar's network
  - July 10: 531 rogue certificates are issued
  - July 19: DigiNotar detects the breach, but doesn't go public
  - End of August: Google notices a man-in-the-middle attack in Iran with one of the rogue certificates. Trust in DigiNotar certificates is revoked by all major browsers
  - Further info: podcast "Darknet Diaries", Episode 3

DigiNotar attack (2011)

- Wildcard certificate for Google
  - Mitm attacks in Iran
  - Searches and email communication
  - Claims that people even died because of this

- Detected from Chrome users
  - Public key pinning: Google hard-coded the fingerprints for its own sites' encryption keys into Chrome, and told the browser to ignore contrary information from certificate authorities

```
Certificate:
Data:
    Version: 3 (0x2)
    Serial Number:
        05:e2:e6:a4:cd:09:ea:54:d6:65:b0:75:fe:22:a2:56
    Signature Algorithm: sha1WithRSAEncryption
    Issuer:
        emailAddress               = info@diginotar.nl
        commonName                 = DigiNotar Public CA 2025
        organizationName           = DigiNotar
        countryName                = NL
    Validity
        Not Before: Jul 10 19:06:30 2011 GMT
        Not After : Ju
    Subject:
        commonName
        serialNumber
        localityName
        organizationNa
        countryName
    Subject Public Key
        Public Key Alg
        RSA Public Key
            Modulus (2
                00:cd:
                24:3f:
                0d:ec:
                79:e2:
                9e:90:
                cc:d1:
                88:b6:
                6e:f7:
                2d:1f:
                a1:28:
                26:ec:
                aa:64:
                2a:79:
                9f:2d:
                f7:c8:af:70:89:43:9a:b8:b8:ce:5a:29:3d:c3:0f:
                93:de:57:37:f8:ad:f2:4a:40:d8:02:4d:68:88:05:
                cf:57:71:61:14:ba:cc:f0:02:c9:e6:83:b7:b6:10:
                94:5d
            Exponent: 65537 (0x10001)
    X509v3 extensions:
        Authority Information Access:
            OCSP - URI:http://validation.diginotar.nl
```



An attack on a CA

- Comodo 2016 Optical Character Recognition (OCR) attack

1. Register "altelekom.at"

That's Telekom Austria :$
one of the largest telecom providers in Austria

2. Ask Comodo for an TLS certificate for "a1telekom.at"

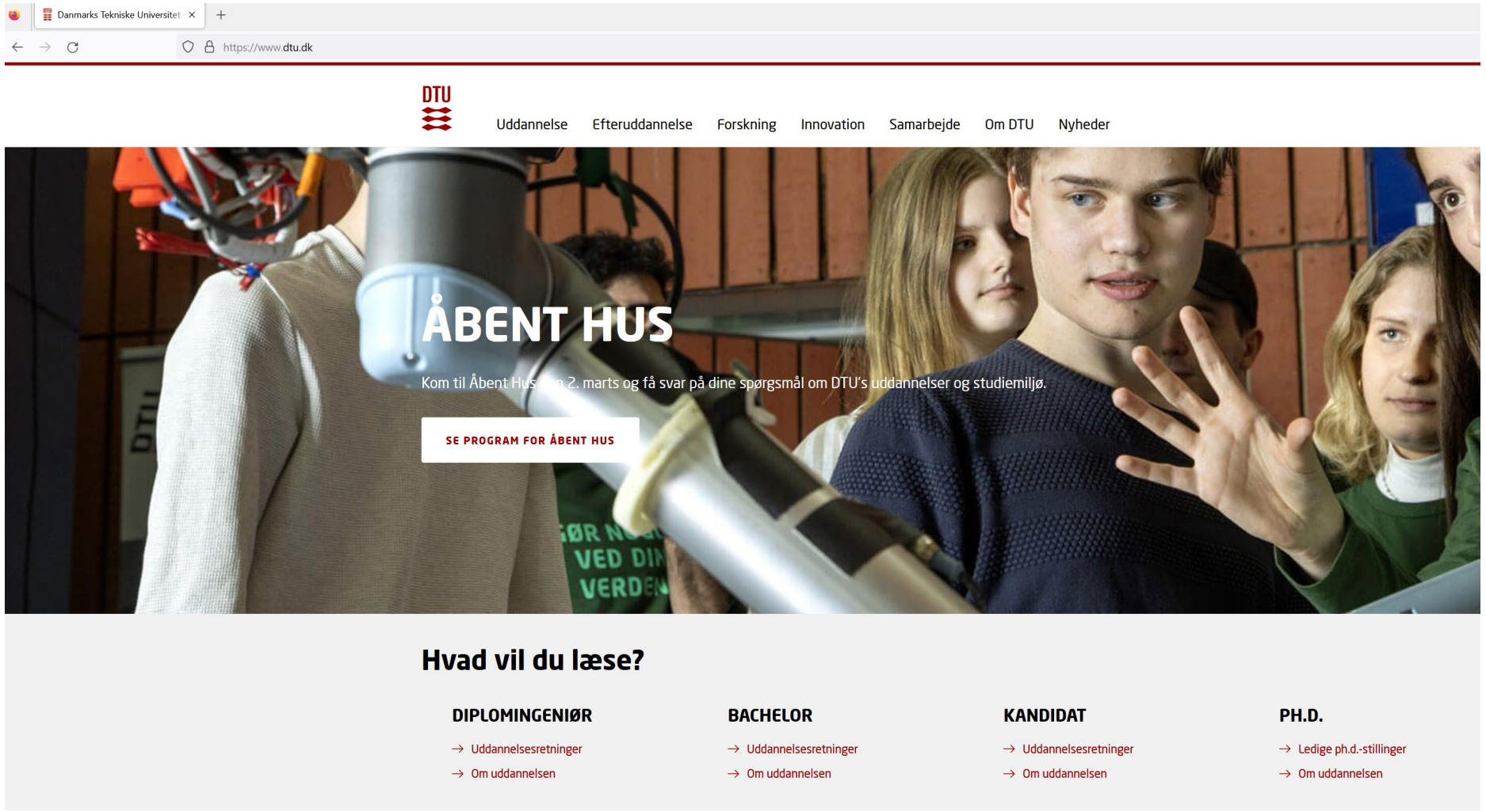
COMODO
Creating Trust Online®

3. Comodo uses OCR to read received documents

4. Shit happens ☺
OCR reads "a1telekom.at"

5. Here is your TLS certificate for "a1telekom.at"!

And why do we need the s in https?



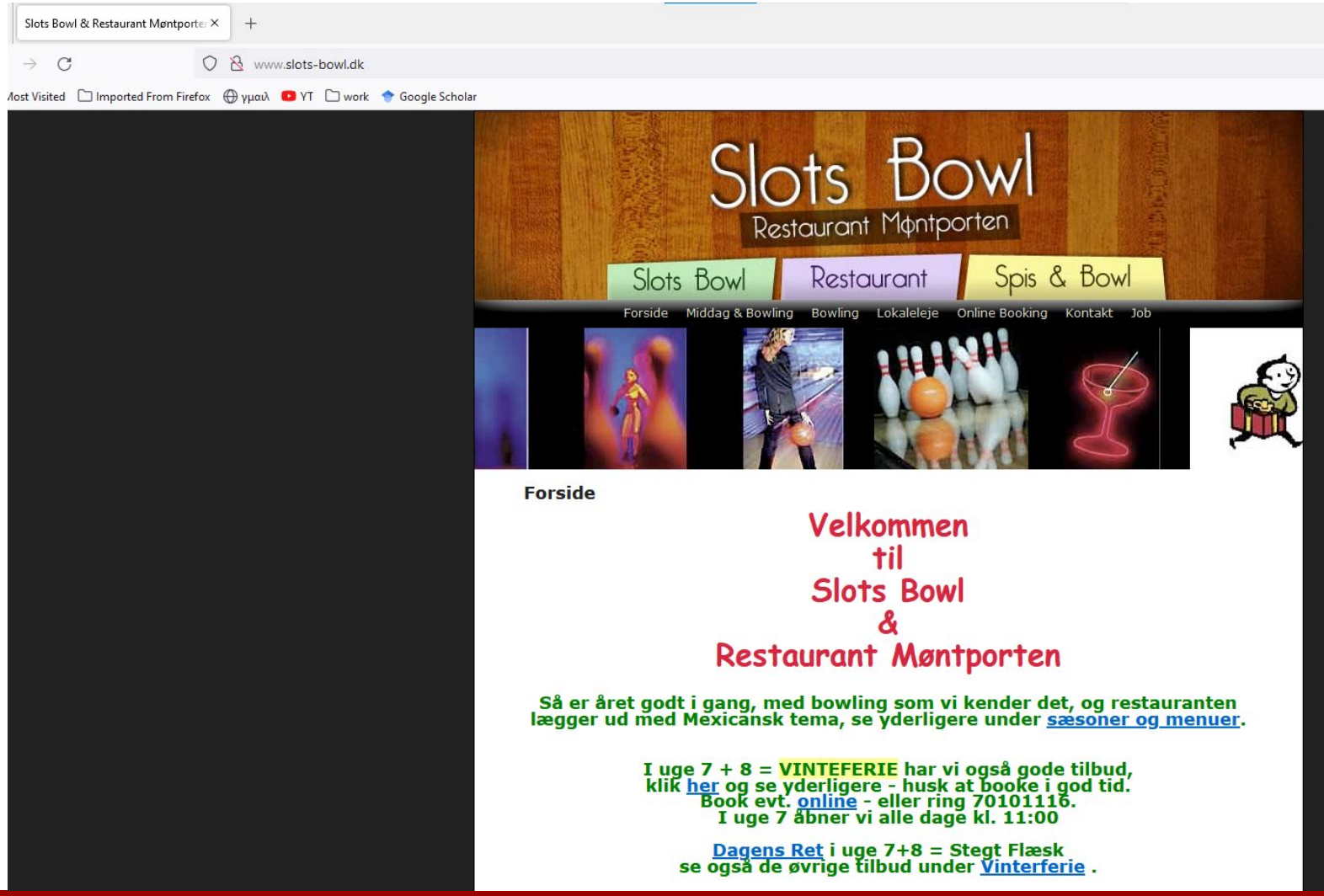
Why is it always bowling websites?

Video: anatomy of an IoT attack

Among others the hacker exploited a (very old) bowling website

Let’s go back in time

- Meet Slots bowl
- Bowling place in Hillerod
- Website is a time machine
  - –90s aesthetics
  - –But also, **no TLS**

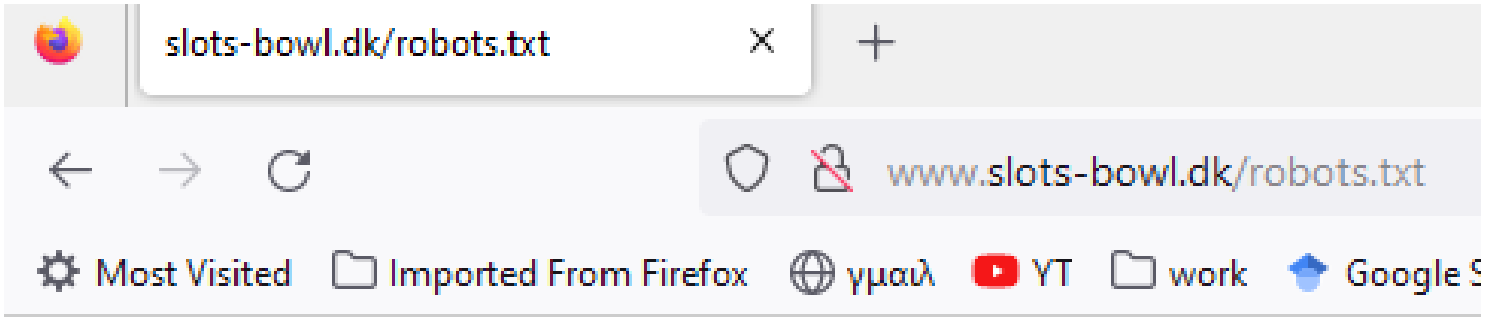


Let’s go back in time

– Booking is done via TLS (that’s good)
– BUT how about the rest of the website?

Let’s go back in time (don’t try this at home)

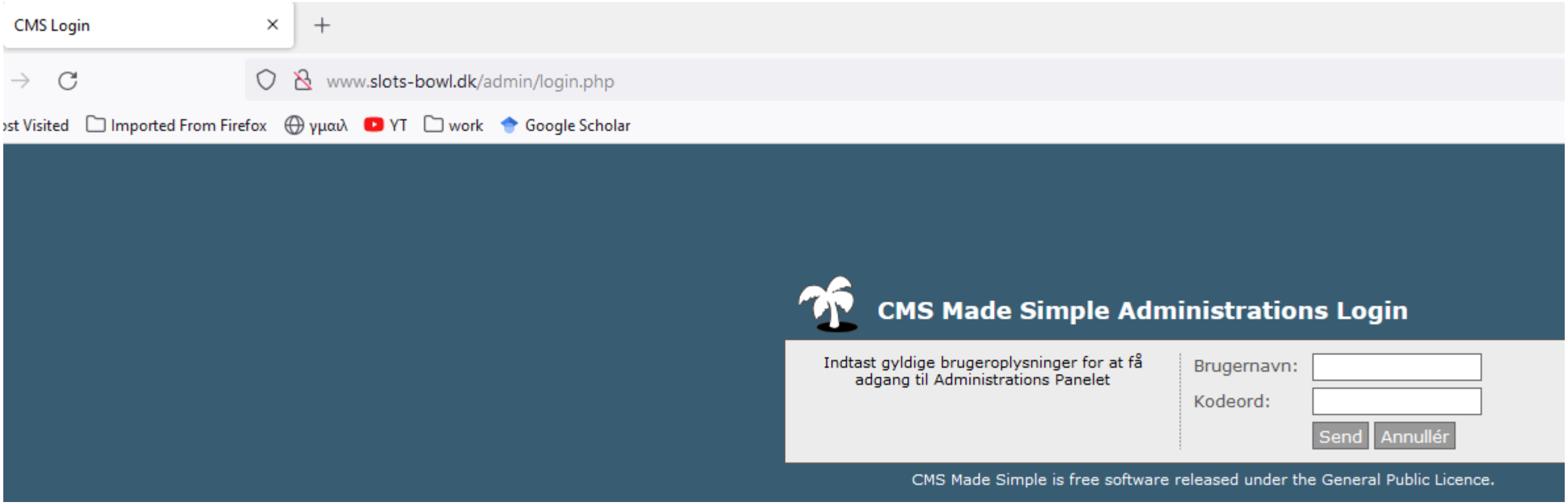
– Booking is done via TLS (that’s good)
– BUT how about the rest of the website?



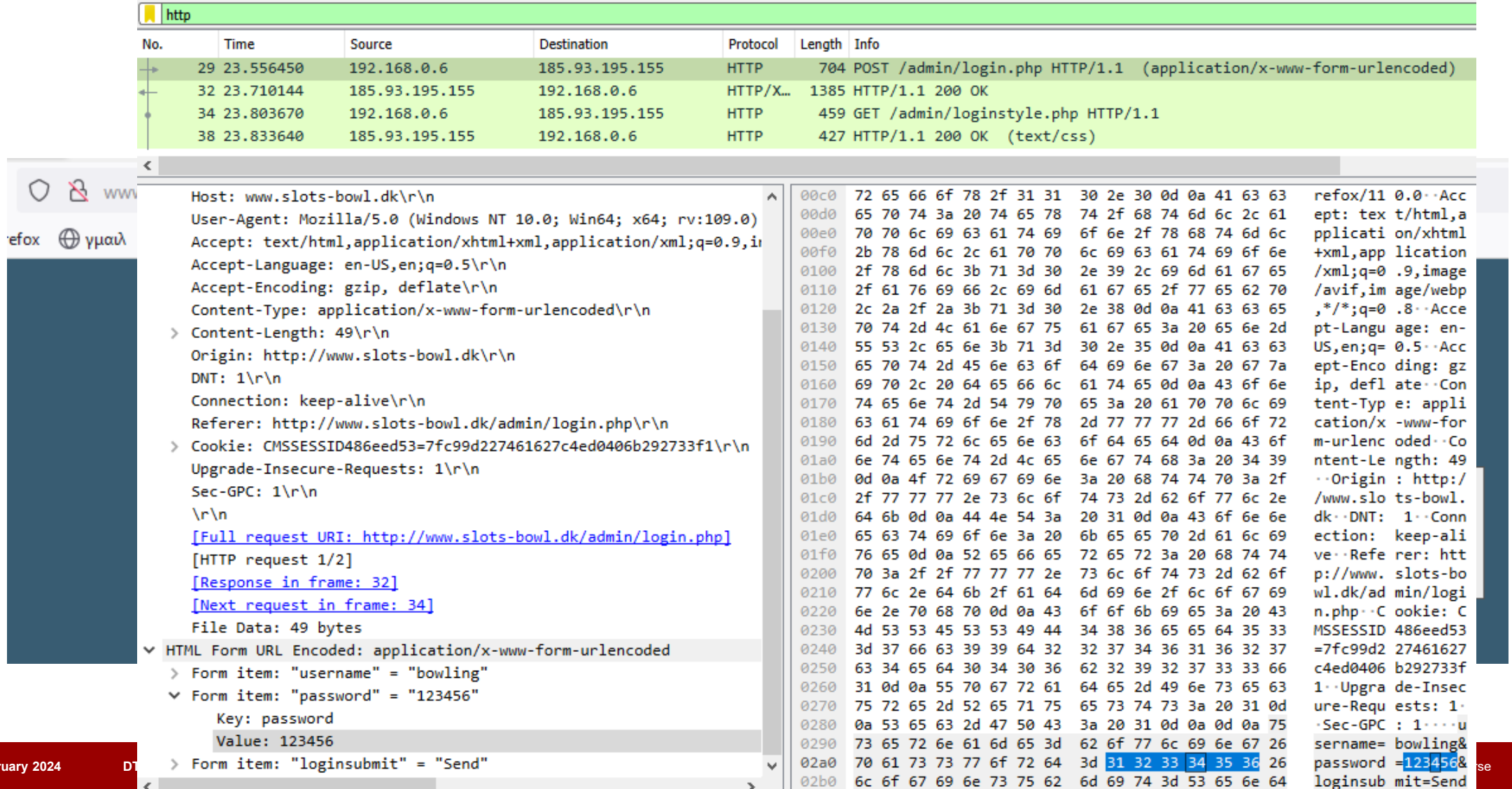

```
User-agent: *
Disallow: /admin/
Disallow: /contrib/
Disallow: /doc/
Disallow: /lib/
Disallow: /modules/
Disallow: /plugins/
Disallow: /scripts/
Disallow: /tmp/
```

Let’s go back in time (don’t try this at home)

– Booking is done via TLS (that’s good)
– BUT how about the rest of the website?



Let’s go back in time (don’t try this at home)



Outline

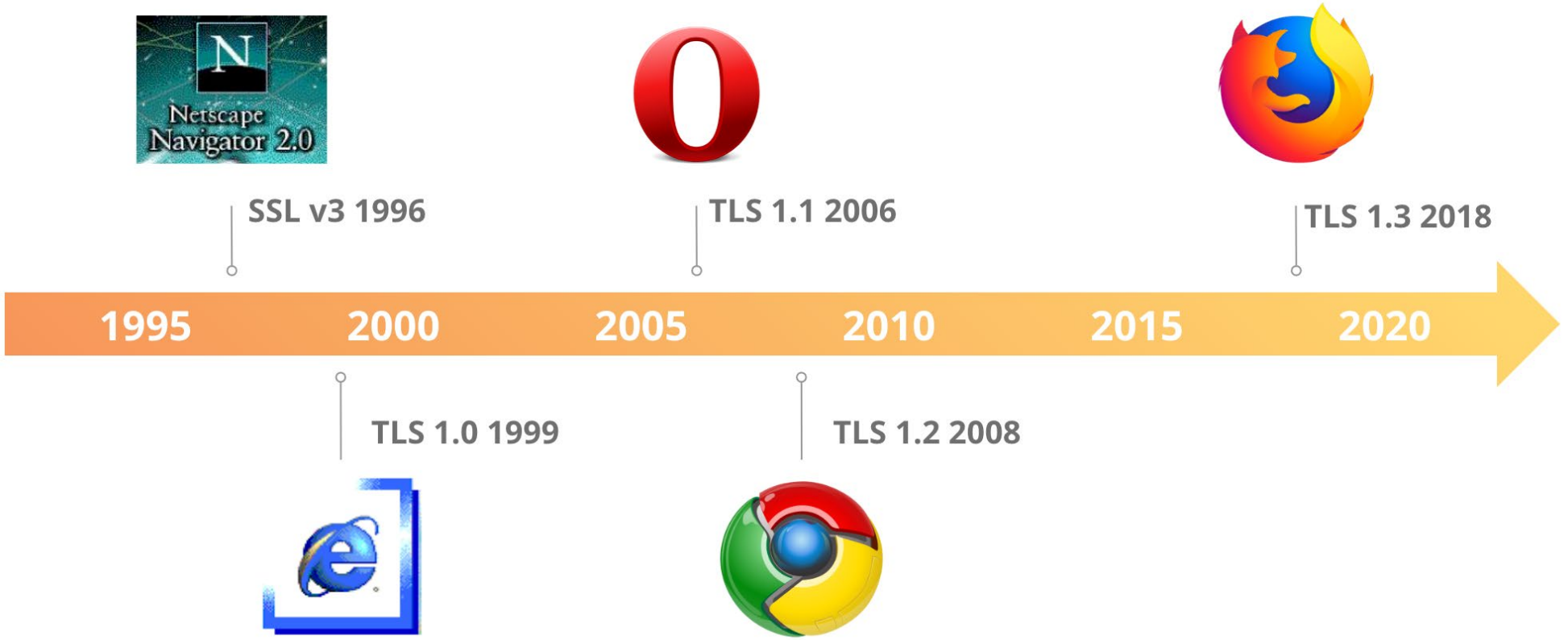
- Introduction
- Certificates
- **TLS**
  - What it can offer
  - TLS protocol
  - TLS resumption
- **Beyond HTTP**
- **Attacks on TLS**
- **Lab exercise**

On the ambiguity of names

- SSL: Secure Sockets Layer
- TLS: Transport Layer Security
- TLS is a **newer version** of SSL
  - For political/historical reasons SSL is still used/mentioned

Protocol	Published	Status
SSL 1.0	Unpublished	Unpublished
SSL 2.0	1995	Deprecated in 2011 (RFC 6176)
SSL 3.0	1996	Deprecated in 2015 (RFC 7568)
TLS 1.0	1999	Deprecation planned in 2020[11]
TLS 1.1	2006	Deprecation planned in 2020[11]
TLS 1.2	2008	
TLS 1.3	2018	

On the ambiguity of names



What does it do?

- **Confidentiality**
  - Encryption of a session (e.g., via DH-RSA)
- **Authentication** (via certificates)
  - Is valid for the same domain as the one being accessed
  - Has been issued by a trusted CA (Certificate Authority)
  - Is valid and not passed its expiration date
- Data **integrity**
- Forward secrecy
  - (most of the time; depends on protocol version and settings)

What it doesn’t do

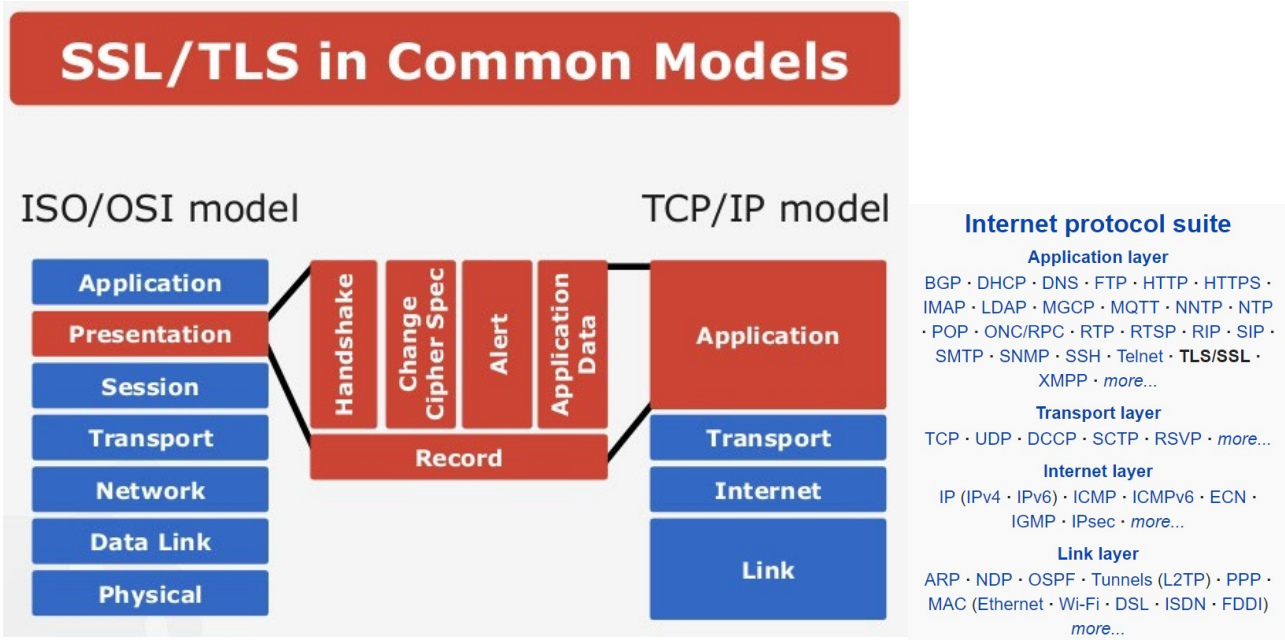
- **Privacy/Anonymity**
  - Everyone knows that A communicates with B
  - They don’t know the content of the communication
- It doesn’t make a website “**safe**”
  - Anyone can create an SSL/TLS certificate
- It doesn’t mean that the data saved on the server is encrypted too
- (also sometimes protocol ≠ implementation ;))

Outline

- **Introduction**
- **Certificates**
- **TLS**
  - What it can offer
  - TLS protocol
  - TLS resumption
- **Beyond HTTP**
- **Attacks on TLS**
- **Lab exercise**

TLS at a glance

- A protocol offering various security properties



Internet protocol suite

Application layer

BGP · DHCP · DNS · FTP · HTTP · HTTPS · IMAP · LDAP · MGCP · MQTT · NNTP · NTP · POP · ONC/RPC · RTP · RTSP · RIP · SIP · SMTP · SNMP · SSH · Telnet · **TLS/SSL** · XMPP · *more...*

Transport layer

TCP · UDP · DCCP · SCTP · RSVP · *more...*

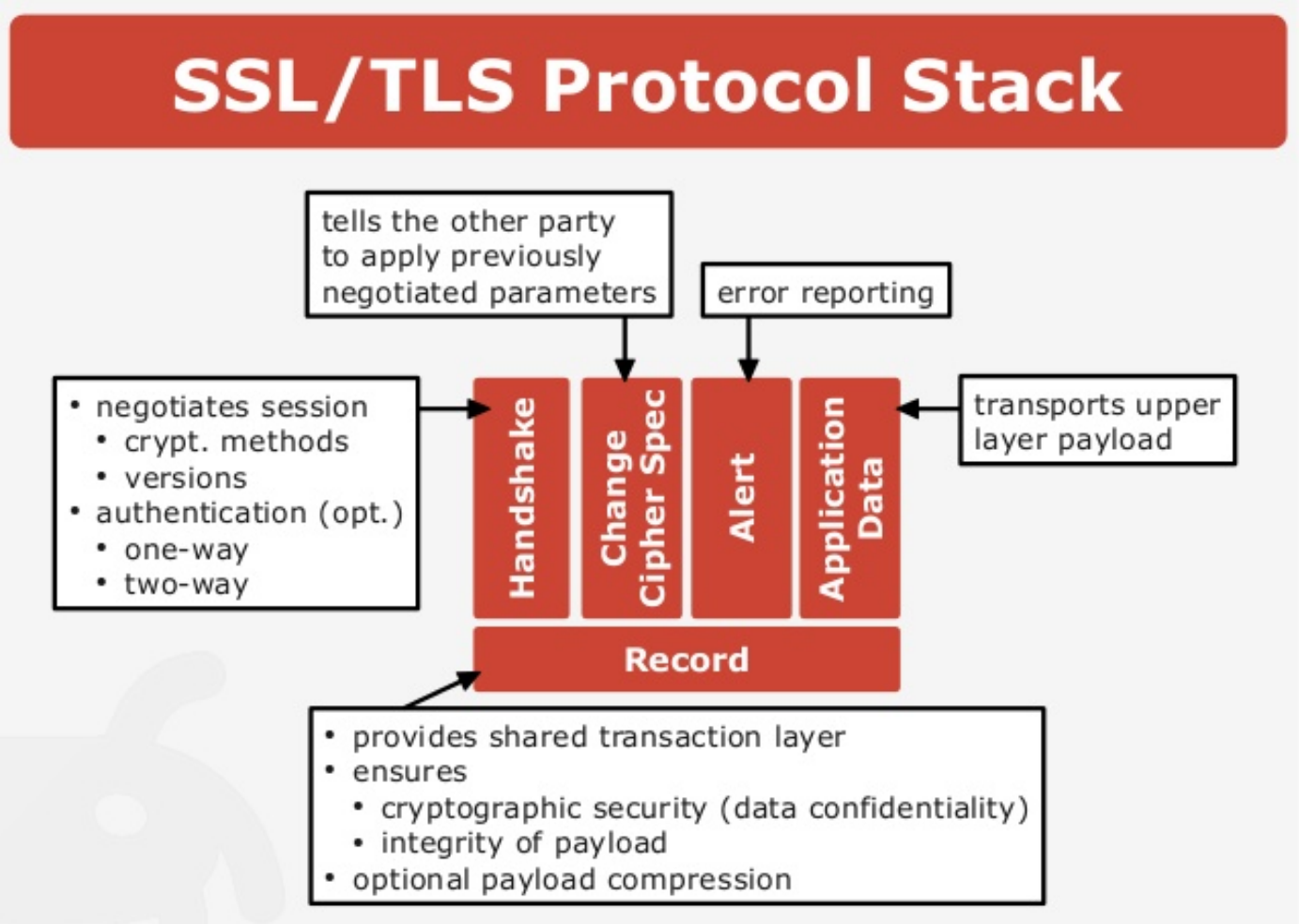
Internet layer

IP (IPv4 · IPv6) · ICMP · ICMPv6 · ECN · IGMP · IPsec · *more...*

Link layer

ARP · NDP · OSPF · Tunnels (L2TP) · PPP · MAC (Ethernet · Wi-Fi · DSL · ISDN · FDDI) *more...*

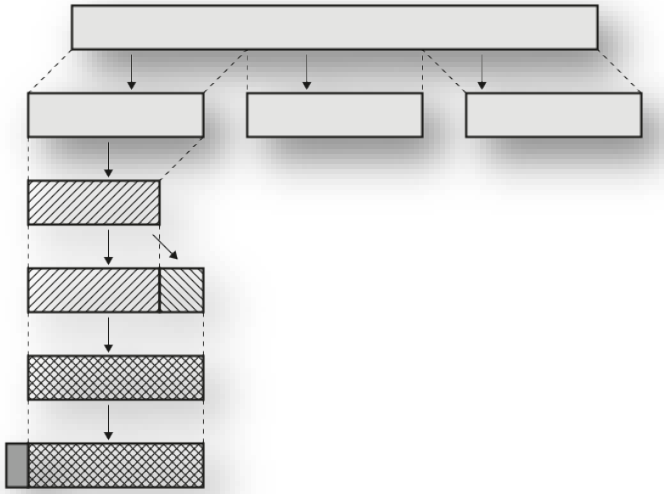
TLS at a glance



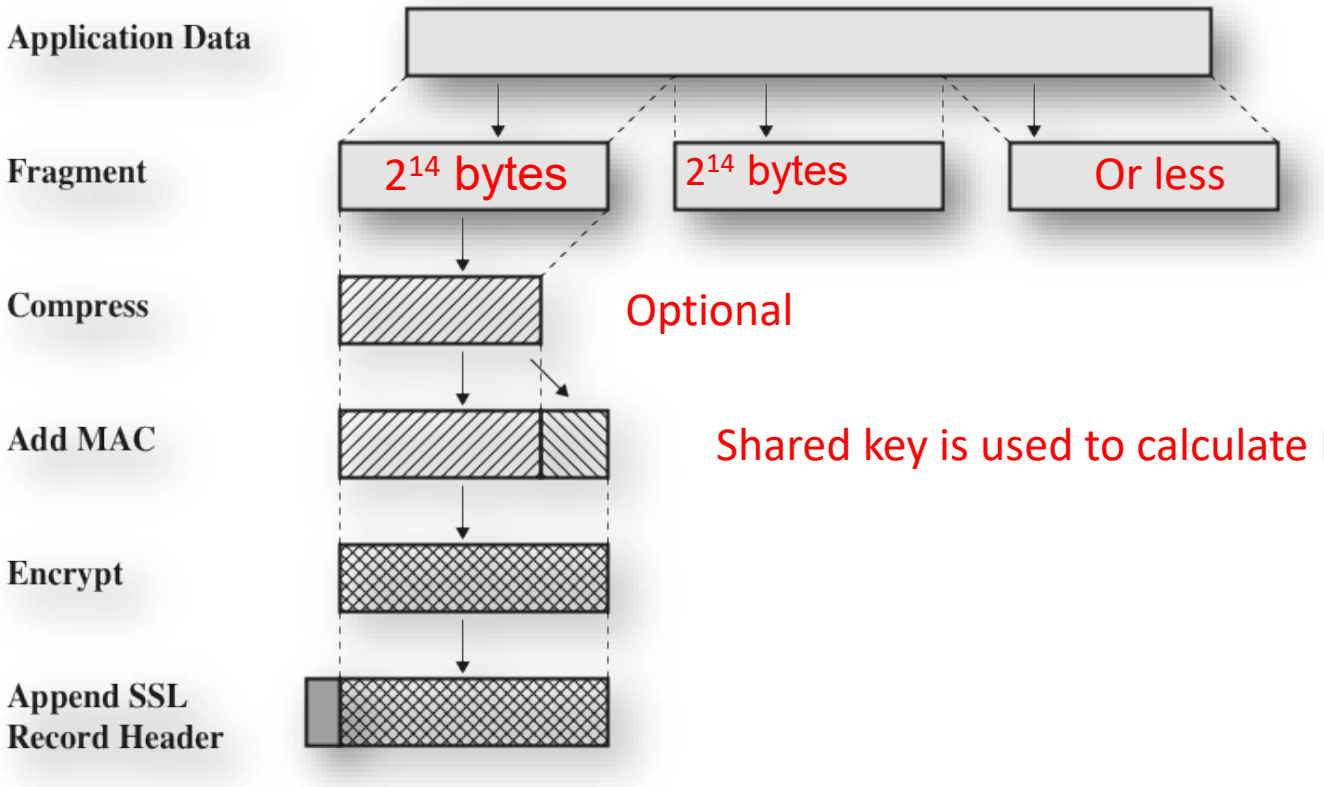
TLS Record Protocol

- The Record Protocol **takes messages to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies a MAC, encrypts, and transmits the result**

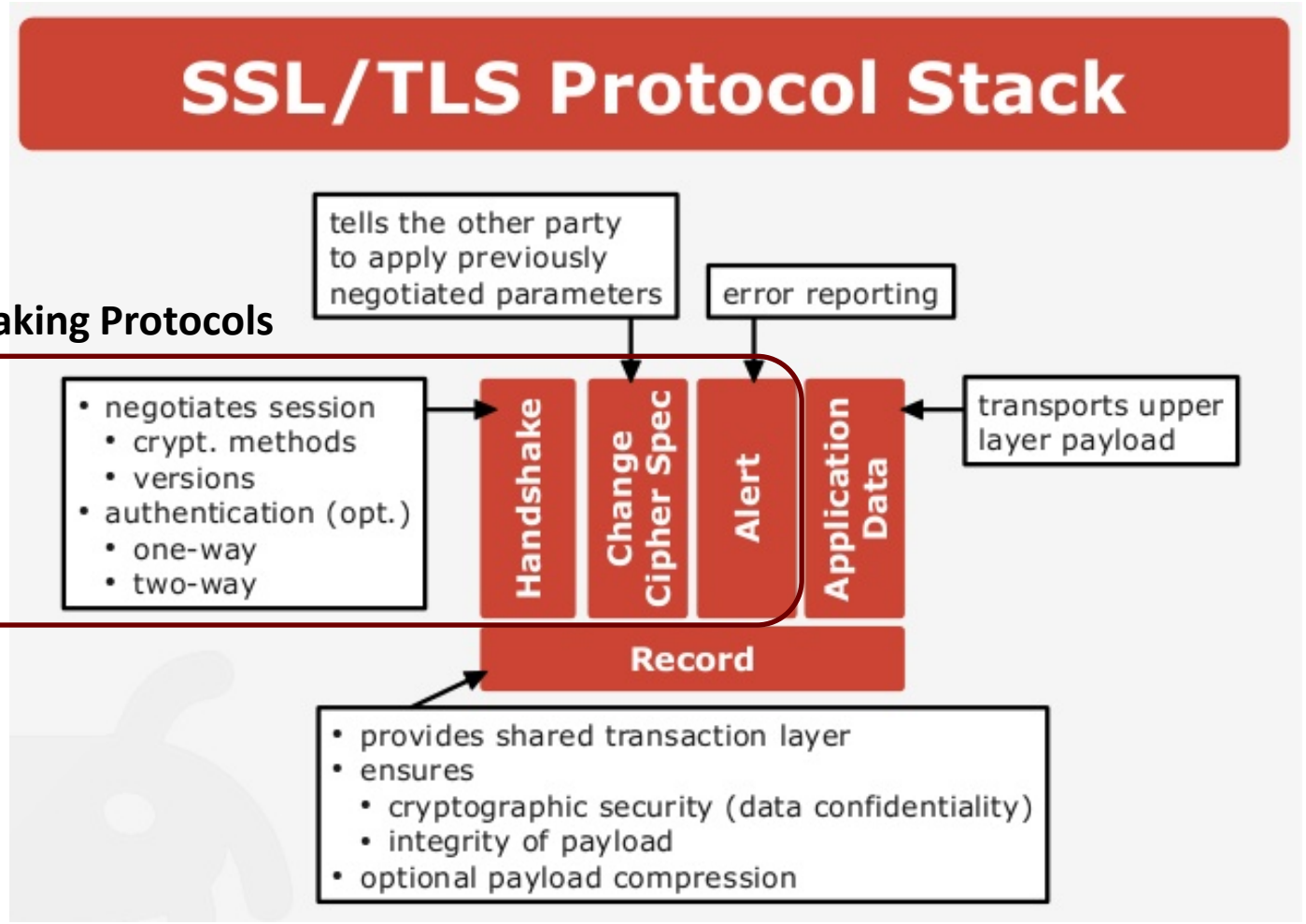
- (For the receiver's end: received data is verified, decrypted, reassembled, and then delivered to higher-level clients)



TLS Record Layer



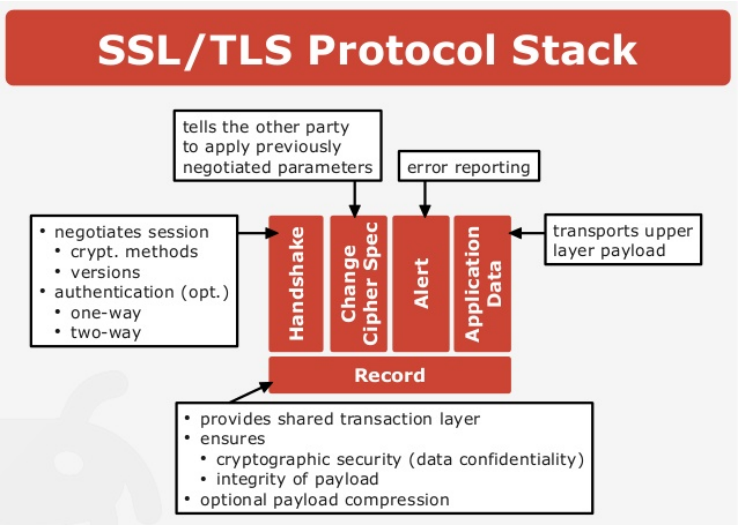
TLS at a glance



TLS Higher Layer sub-protocol

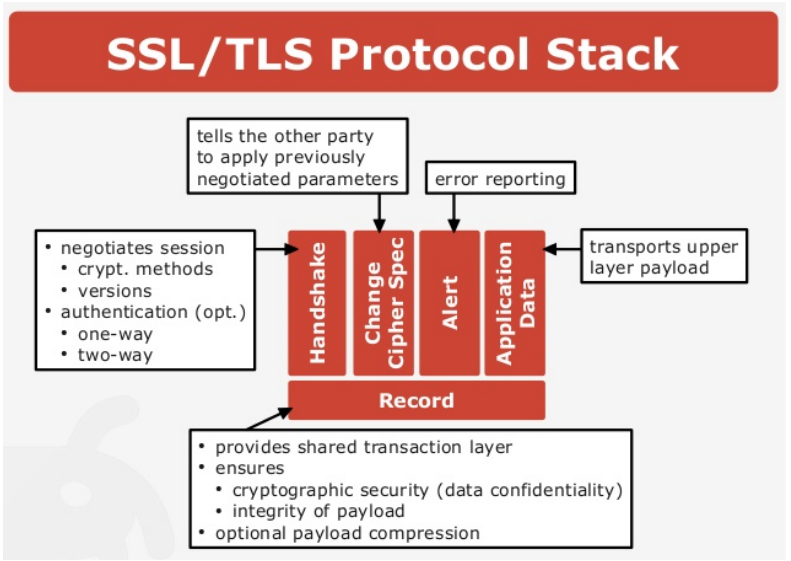
- **Change Cipher Spec**
  - Notify the receiving party that subsequent records will be protected under the just-negotiated CipherSpec and keys
  - Only one byte long, and signals the change in communications protocol by having a value of ‘1’
  - Exists only in TLS versions <=1.2

- **Alert** –Sends errors, problems or warnings about the connection. This layer is formed with two fields:
  - **Severity Level**: value 1 (warning) or 2 (fatal error; discontinue session)
  - **Alert Description**: indicates the specific error that caused the Alert Message to be sent from a party. This field is one byte, mapped to one of twelve specific numbers (e.g., handshakeFailure, BadCertificate, CertificateRevoked, etc.)



TLS Handshake

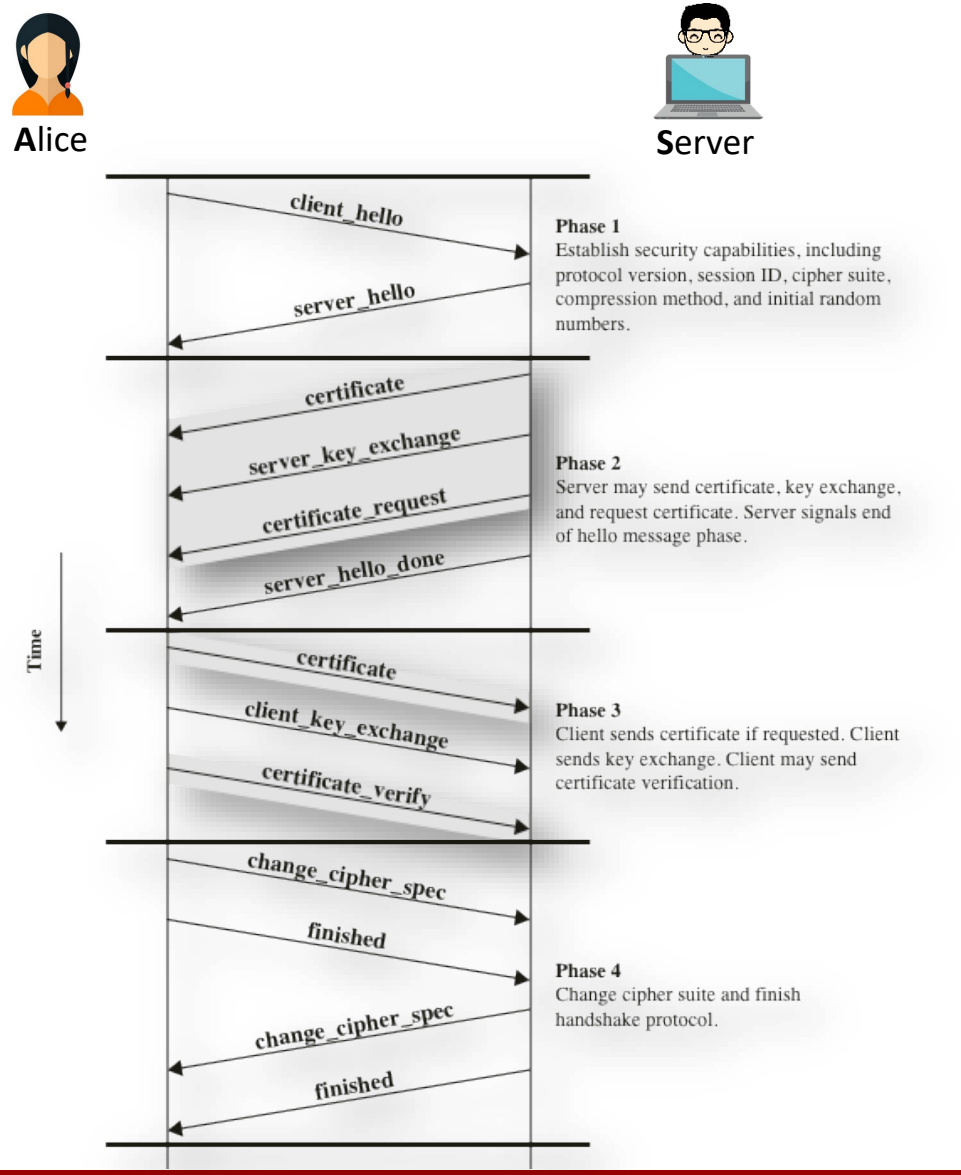
- Allows server/client to authenticate each other and negotiate an encryption and MAC algorithm and cryptographic keys
- Used before any application data is transmitted
- Each message has three fields
  - Type (1 byte) – Indicates one of the 10 messages (see next slide)
  - Length (3 bytes) – the length of the message in bytes
  - Content (> 0 bytes) – the parameters associated with this message



TLS Handshake Protocol Message Types

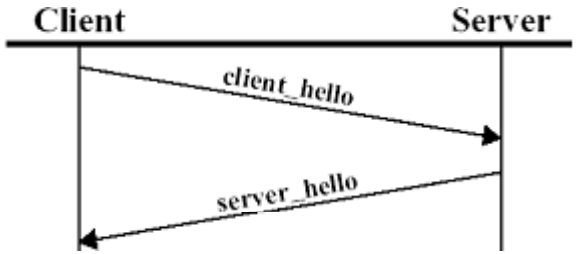
Message Type	Parameters
hello_request	null
client_hello	version, random, session id, cipher suite, compression method
server_hello	version, random, session id, cipher suite, compression method
certificate	chain of X.509v3 certificates
server_key_exchange	parameters, signature
certificate_request	type, authorities
server_done	null
certificate_verify	signature
client_key_exchange	parameters, signature
finished	hash value

Handshake Protocol Action (TLS <=1.2)



Phase 1 – Establish Security Capabilities

- Used to initiate a logical connection and to establish the security capabilities that will be associated with it
  - Exchange is initiated by the client with following parameters:
    - Version
    - Random
    - Session-ID
    - CipherSuite
    - Compression Method



- First element in the CipherSuite is Key exchange method
  - RSA, Fixed Diffie-Hellman, Ephemeral Diffie-Hellman, Anonymous Diffie-Hellman, Fortezza

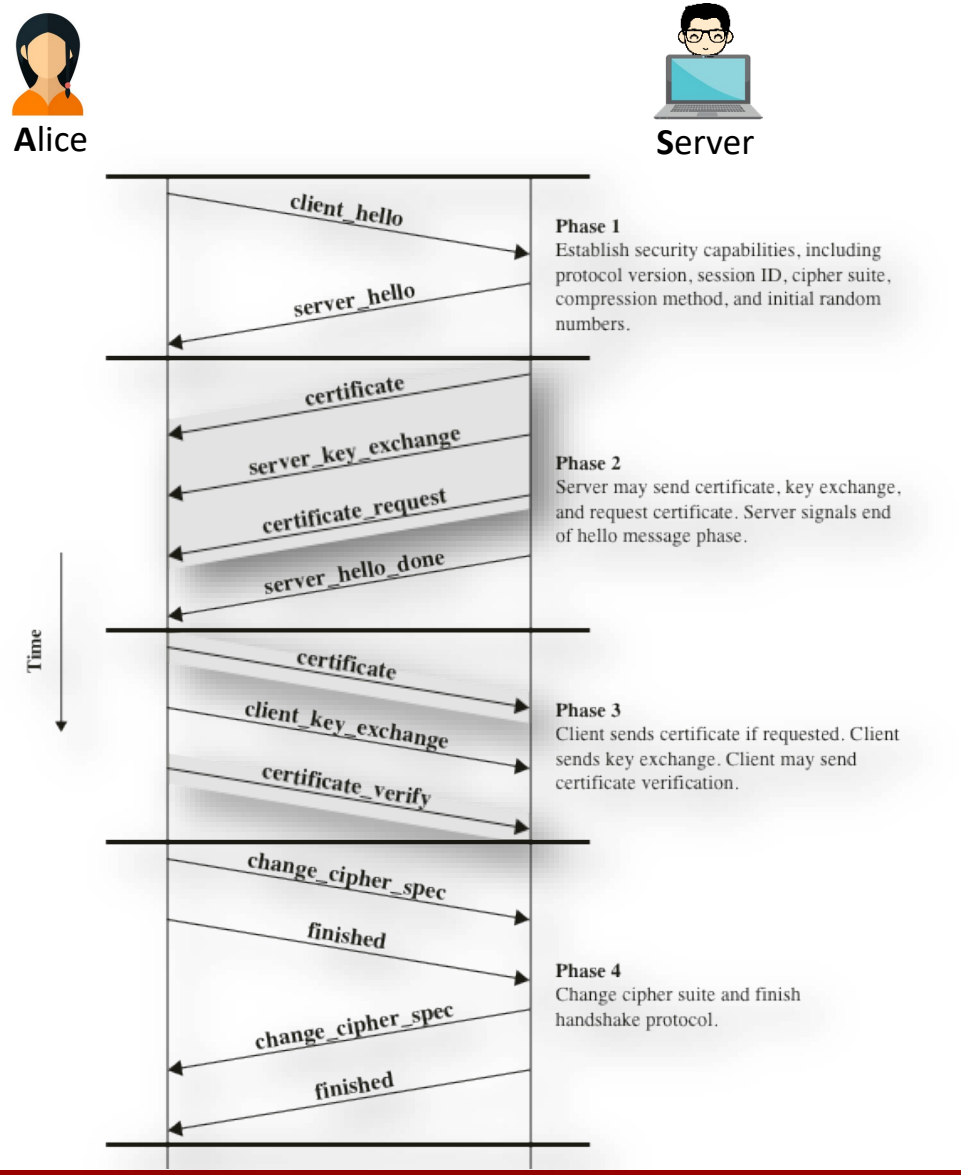
Note on DH versions

- **Anonymous Diffie-Hellman:** classic Diffie-Hellman, i.e., without authentication
  - Because the keys used in the exchange are not authenticated, the protocol is susceptible to Man-in-the-Middle attacks.

- **Fixed Diffie-Hellman** embeds the server's public parameter in the certificate, and the CA then signs the certificate. That is, the certificate contains the Diffie-Hellman public-key parameters, and those parameters never change.

- **Ephemeral Diffie-Hellman** uses temporary, public keys. Each instance or run of the protocol uses a different public key. The authenticity of the server's temporary key can be verified by checking the signature on the key.
  - Because the public keys are temporary, a compromise of the server's long-term signing key does not jeopardize the privacy of past sessions (**Perfect Forward Secrecy**)

Handshake Protocol Action (TLS <=1.2)

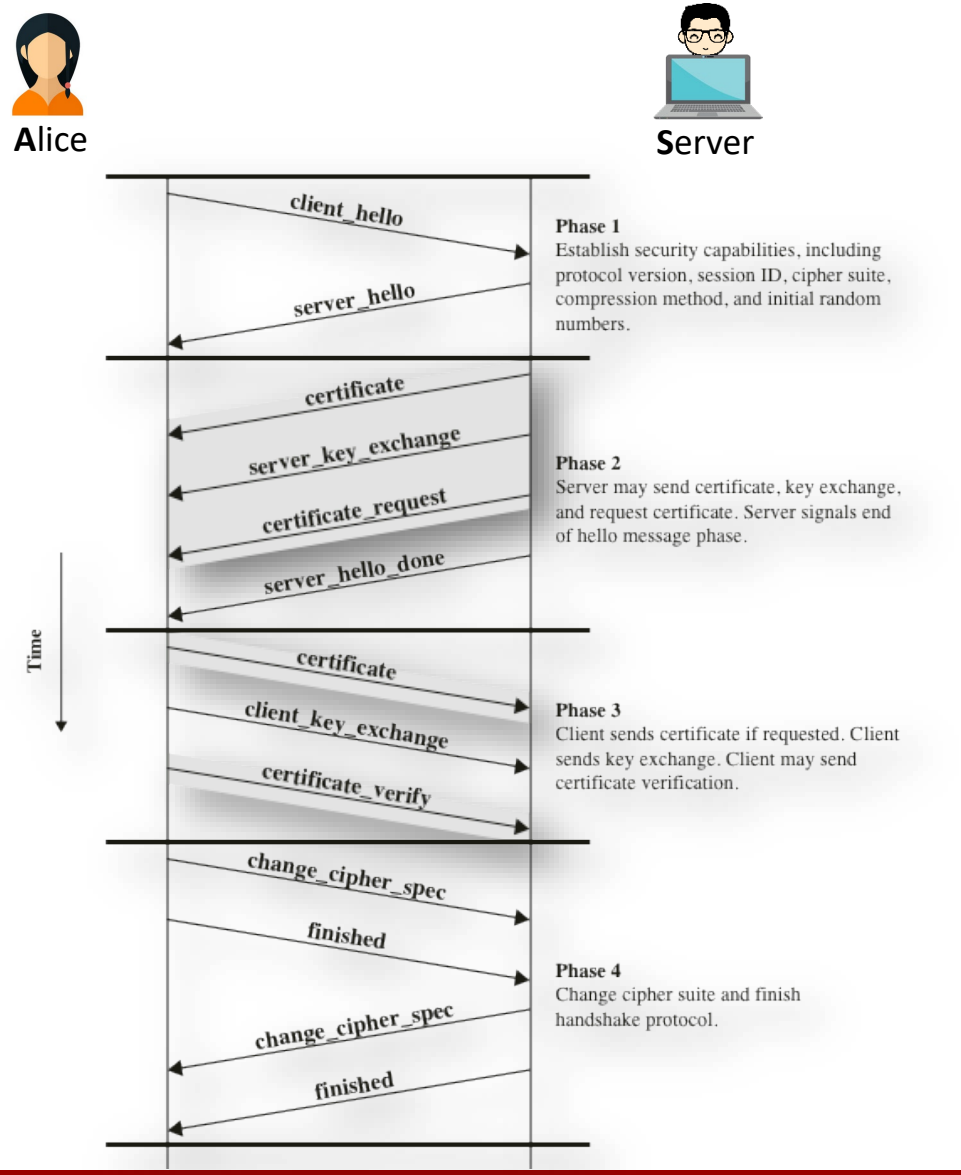


Phase 2– Server Authentication and Key Exchange

- Server begins this phase by sending its **certificate** if it needs to be authenticated
  - Message contains one chain of certificates
- Next Message “server_key_exchange message”
  - The message is not required if the server has sent a certificate with fixed Diffie-Hellman Parameters or RSA key exchange is to be used
- Certificate_request message
  - Includes two parameters – certificate_type and certificate_authorities

- Final message “server_done message”
  - Sent by the server to indicate the end of the server hello and associated messages
- Server waits for a client response

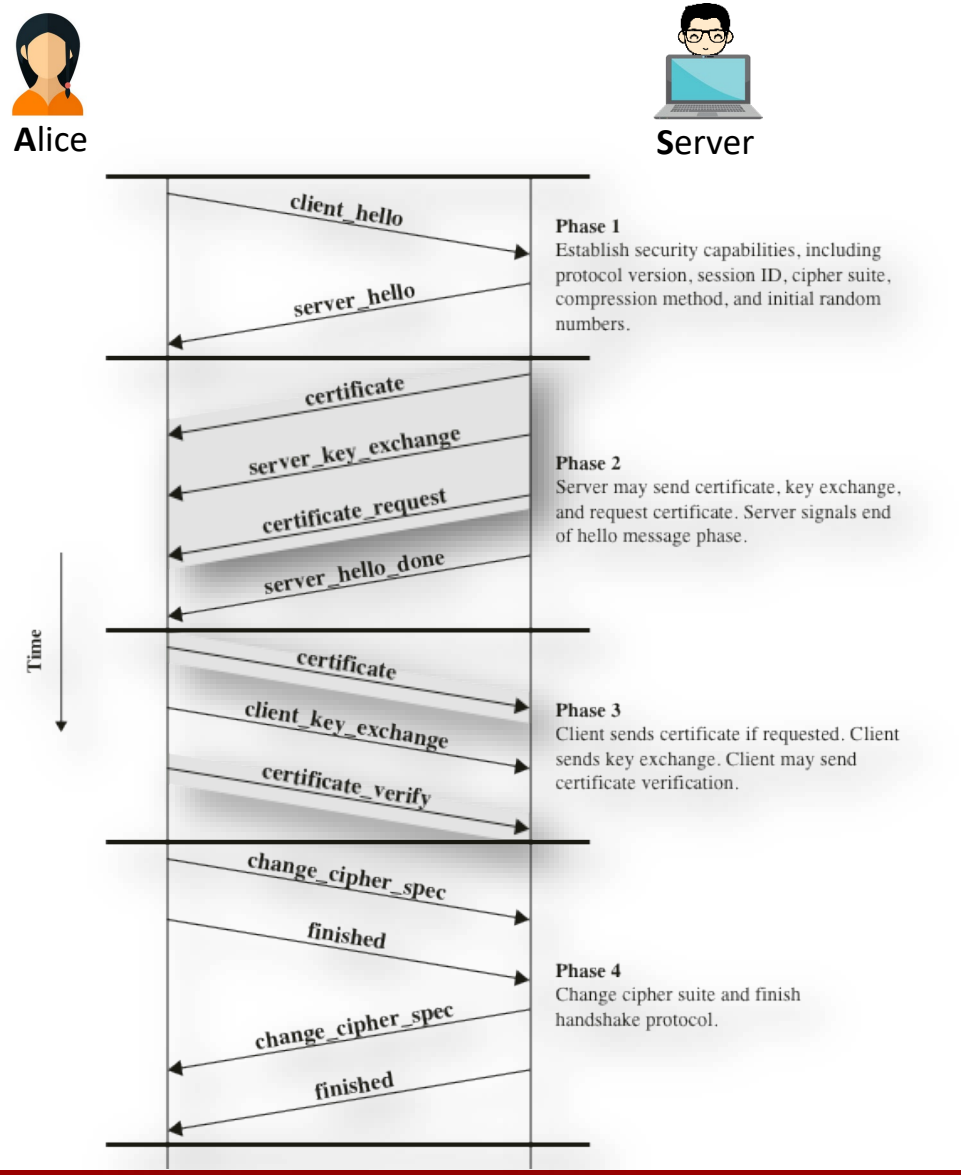
Handshake Protocol Action (TLS <=1.2)



Phase 3– Client Authentication and Key Exchange

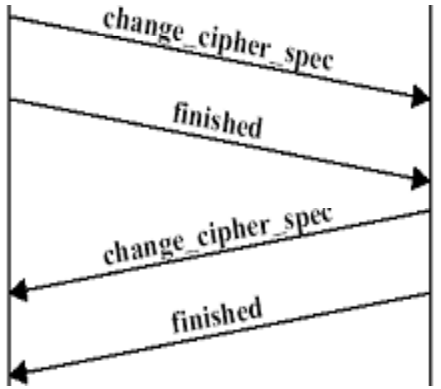
- Client verifies that the server provided a valid certificate
- Checks that the parameters are acceptable
- If requested by the server client sends its own ***certificate***
- Next message “client_key_exchange message”
  - Contents of message depends on the type of the key exchange
- Final message “certificate_verify message”
  - Provide verification of a client certificate

Handshake Protocol Action (TLS <=1.2)



Phase 4– Finish

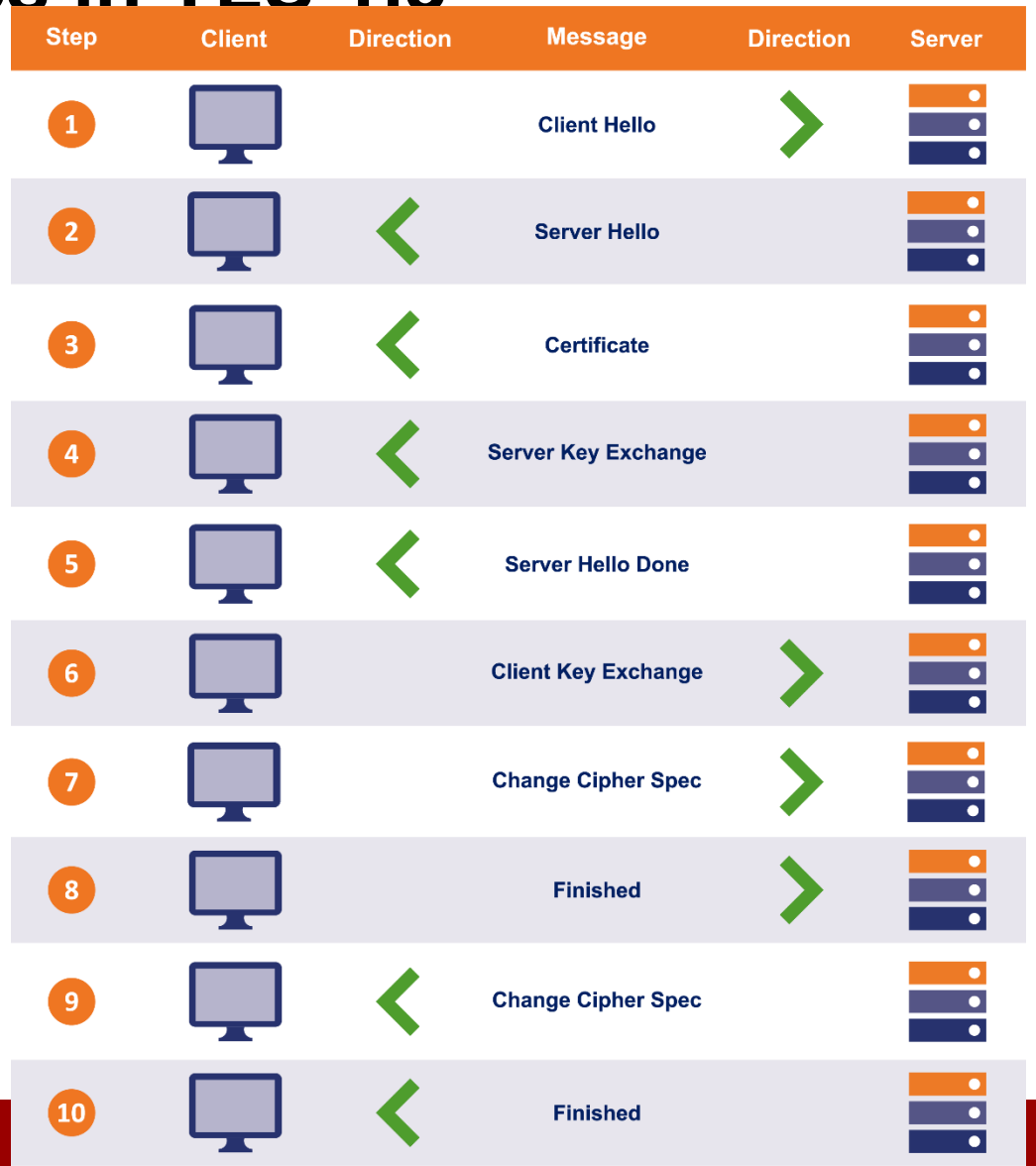
- This phase completes the set up
- Client send a change_cipher_spec message
  - Copies the pending CipherSpec into the current CipherSpec
- Client sends the finished message under the new algorithms, keys, and secrets
  - Verifies that that the key exchange and authentication process was successful
- Similar response from the server side
  - The client and the server can begin to exchange application-layer data



Heartbeat Protocol

- Periodic message exchange to **keep connection alive**
  - Used in popular TLS library implementations
  - problem: no bound checks on heartbeat messages

Differences in TLS 1.3



TLS 1.2 (and earlier versions)

Differences in TLS 1.3

TLS 1.3

TLS 1.3 Handshake

Alice

Server

-Alice Hello
-Supported cipher suite
-Key share

-Server Hello
-Chosen ciphers
-Key share
-Certificate & Signature
-Finished

-Finished
-HTTP GET

-HTTP answer

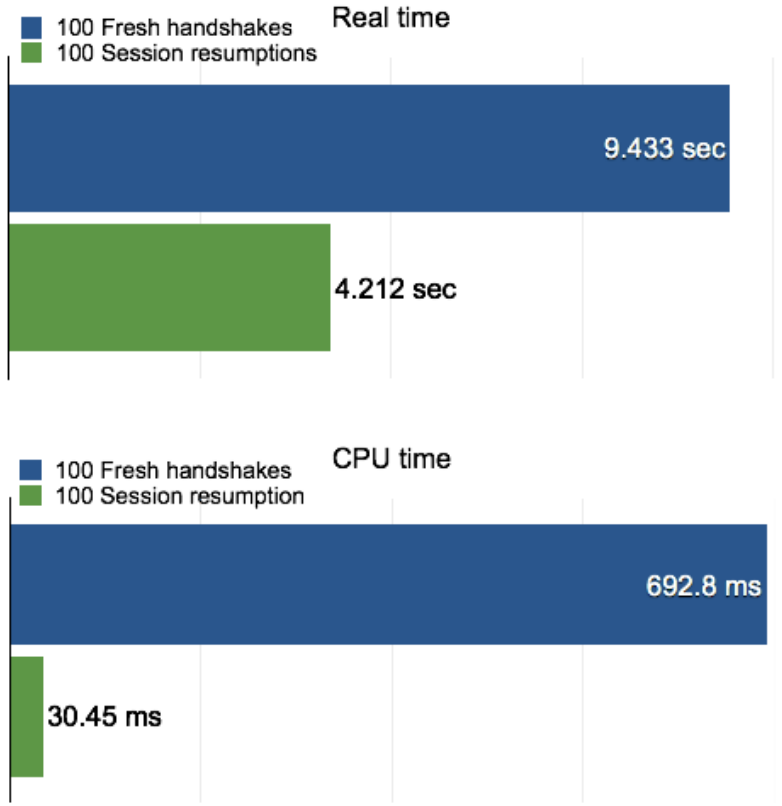
Outline

- **Introduction**
- **Certificates**
- **TLS**
  - What it can offer
  - TLS protocol
  - TLS resumption
- **Beyond HTTP**
- **Attacks on TLS**
- **Lab exercise**

TLS resumption

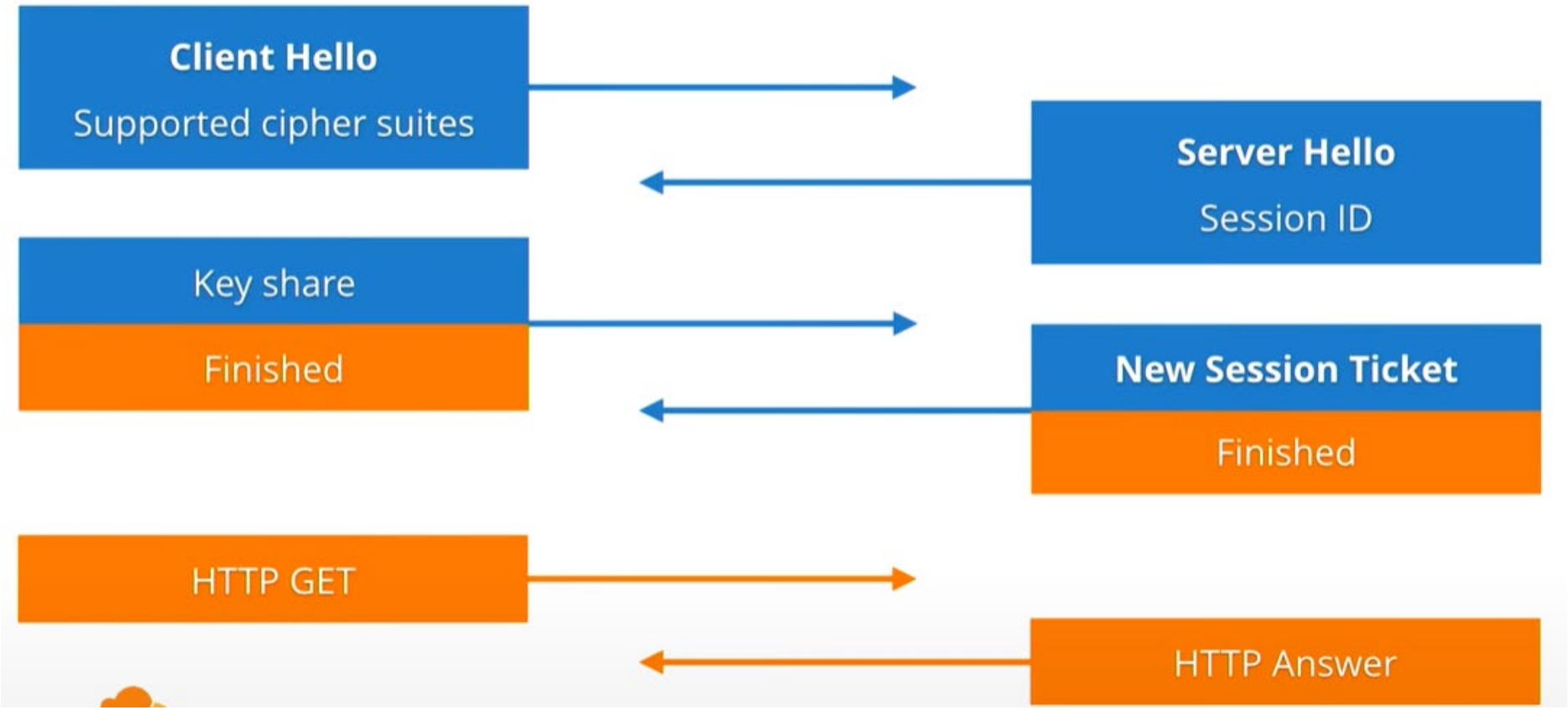
Most TLS connections are not new

- If a client has visited a server before then a TLS resumption will occur
- Much faster
- TLS 1.2 and 1.3 do things differently



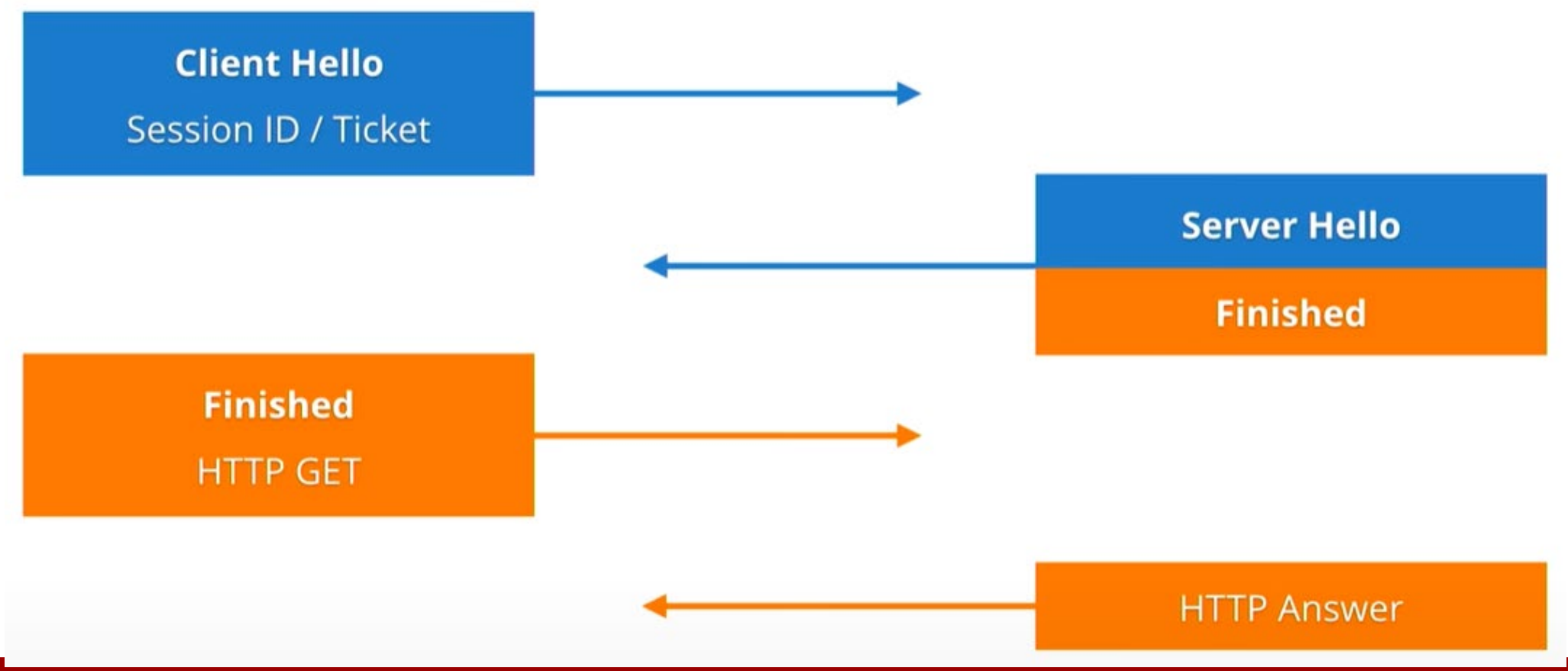
TLS 1.2 resumption

- Reminder: full TLS 1.2 (and introducing session ticket)



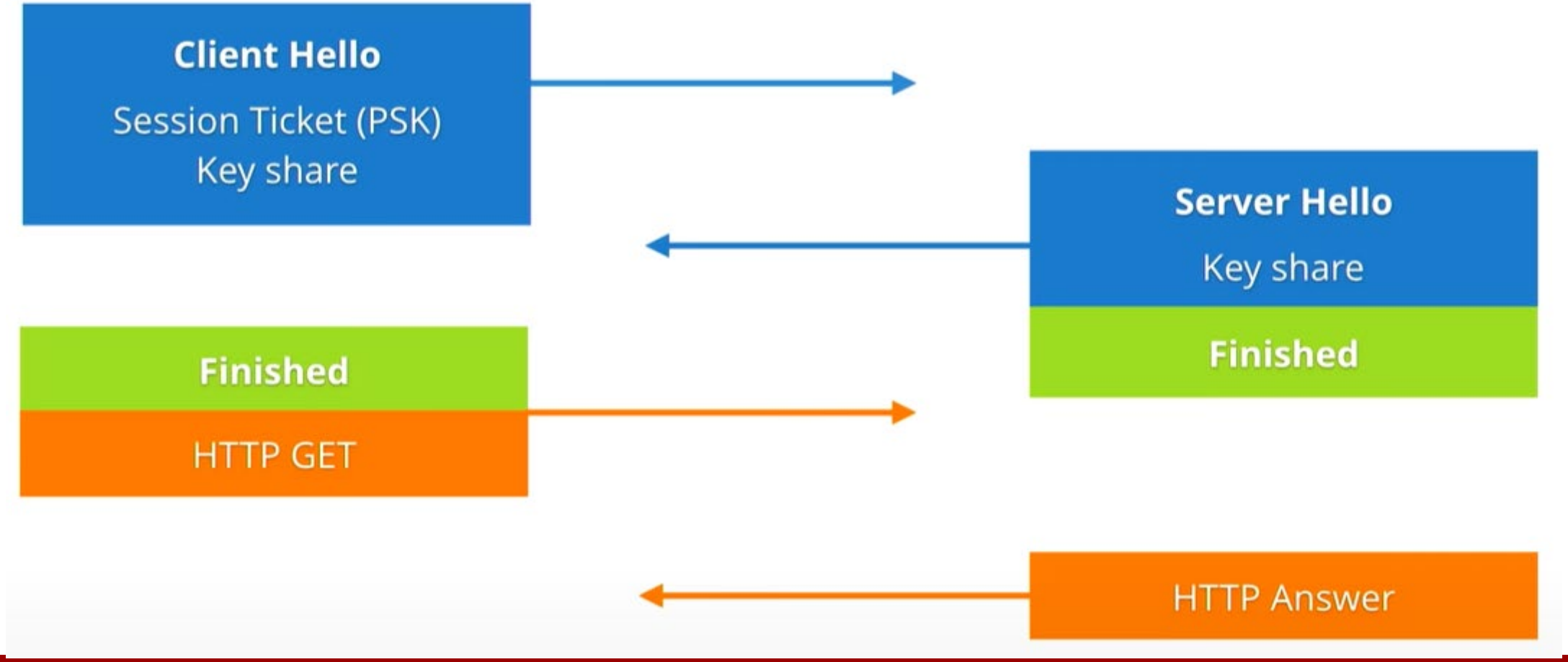
TLS 1.2 resumption

- TLS 1.2 with resumption is 1-RTT

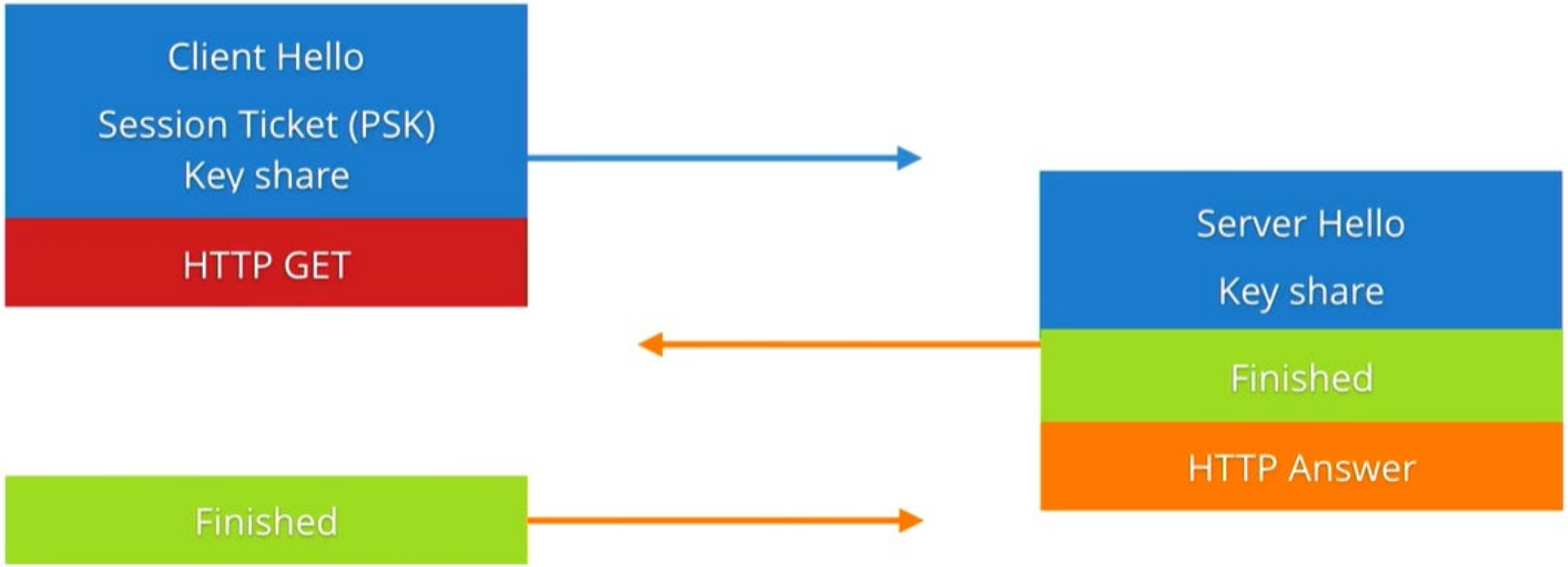


TLS 1.3 resumption 0-RTT

- Reminder of TLS 1.3 (and introducing PSK)

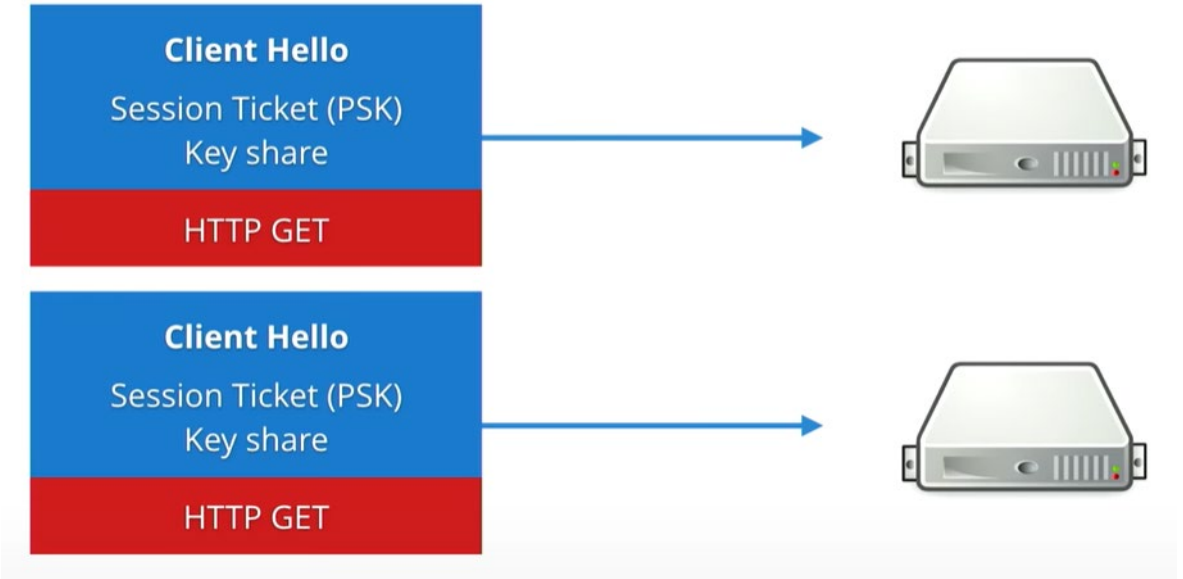


TLS 1.3 resumption 0-RTT



0-RTT limitations

- No forward secrecy on early data
- Early data replay attack on 0-RTT



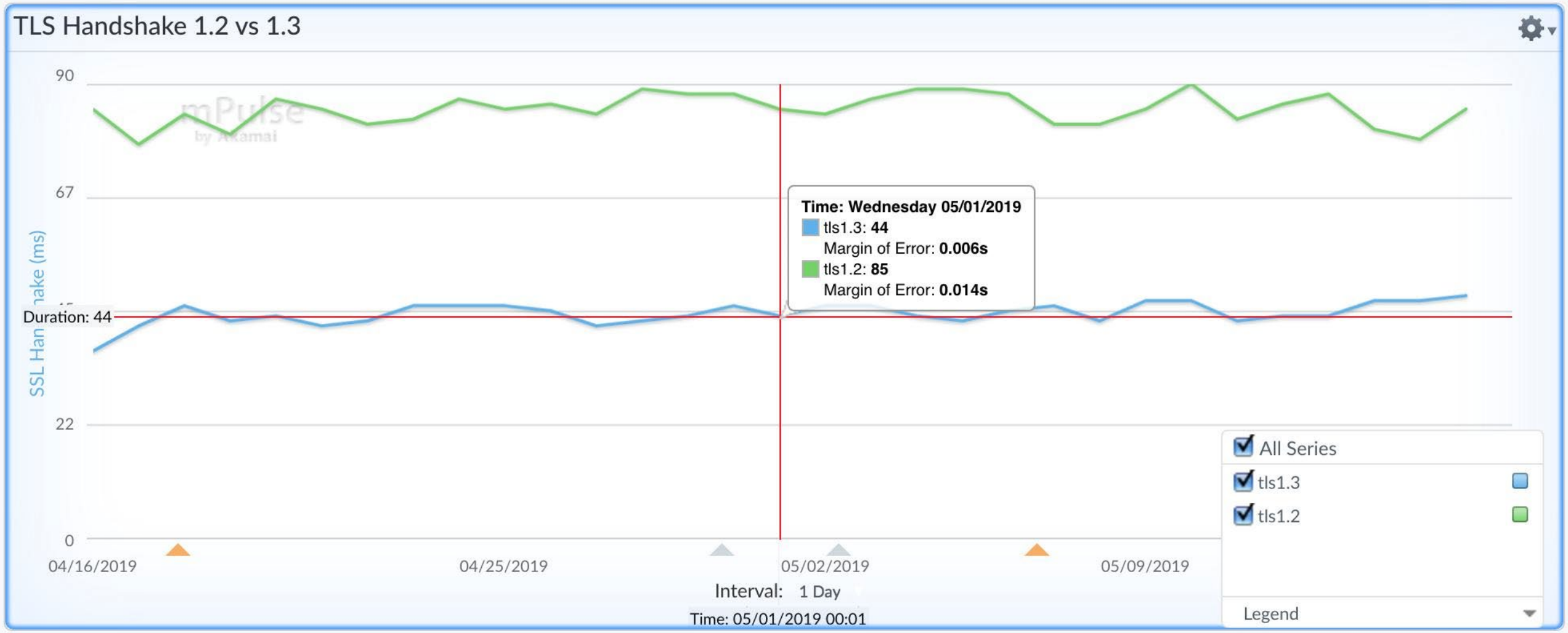
Forward secrecy comparison

- **TLS 1.2**
  - Certificate compromise: **always but only with ECDHE mode**
  - Ticket key compromise : **never**
- **TLS 1.3**
  - Certificate compromise : **always**
  - Ticket key compromise : **always except 0-RTT early data**

Algorithm	SSL 2.0	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3
RSA	Yes	Yes	Yes	Yes	Yes	No
DH-RSA	No	Yes	Yes	Yes	Yes	No
DHE-RSA (forward secrecy)	No	Yes	Yes	Yes	Yes	Yes
ECDH-RSA	No	No	Yes	Yes	Yes	No
ECDHE-RSA (forward secrecy)	No	No	Yes	Yes	Yes	Yes
DH-DSS	No	Yes	Yes	Yes	Yes	No
DHE-DSS (forward secrecy)	No	Yes	Yes	Yes	Yes	No[54]
ECDH-ECDSA	No	No	Yes	Yes	Yes	No
ECDHE-ECDSA (forward secrecy)	No	No	Yes	Yes	Yes	Yes
ECDH-EdDSA	No	No	Yes	Yes	Yes	No
ECDHE-EdDSA (forward secrecy)[55]	No	No	Yes	Yes	Yes	Yes
PSK	No	No	Yes	Yes	Yes	
PSK-RSA	No	No	Yes	Yes	Yes	
DHE-PSK (forward secrecy)	No	No	Yes	Yes	Yes	Yes
ECDHE-PSK (forward secrecy)	No	No	Yes	Yes	Yes	Yes
SRP	No	No	Yes	Yes	Yes	
SRP-DSS	No	No	Yes	Yes	Yes	
SRP-RSA	No	No	Yes	Yes	Yes	
Kerberos	No	No	Yes	Yes	Yes	
DH-ANON (insecure)	No	Yes	Yes	Yes	Yes	
ECDH-ANON (insecure)	No	No	Yes	Yes	Yes	
GOST R 34.10-94 / 34.10-2001[56]	No	No	Yes	Yes	Yes	

TLS 1.2 vs TLS 1.3

- Much faster!
- 0-RTT via resumption
- Old ciphers removed
- AEAD ciphers
- Version negotiation removed
- Forward Secrecy


TLS Handshake 1.2 vs 1.3
90
67
22
0
SSL Handshake (ms)
Duration: 44
Time: Wednesday 05/01/2019
tls1.3: 44
Margin of Error: 0.006s
tls1.2: 85
Margin of Error: 0.014s
All Series
tls1.3
tls1.2
Legend
04/16/2019
04/25/2019
05/02/2019
05/09/2019
Interval: 1 Day
Time: 05/01/2019 00:01

Permitted encryption algorithms

Cipher security against publicly known feasible attacks

Cipher			Protocol version						Status
Type	Algorithm	Nominal strength (bits)	SSL 2.0	SSL 3.0 [n 1][n 2][n 3][n 4]	TLS 1.0 [n 1][n 3]	TLS 1.1 [n 1]	TLS 1.2 [n 1]	TLS 1.3	
Block cipher with mode of operation	AES GCM[46][n 5]	256, 128	N/A	N/A	N/A	N/A	Secure	Secure	Defined for TLS 1.2 in RFCs
	AES CCM[47][n 5]		N/A	N/A	N/A	N/A	Secure	Secure	
	AES CBC[n 6]		N/A	N/A	Depends on mitigations	Depends on mitigations	Depends on mitigations	N/A	
	Camellia GCM[48][n 5]	256, 128	N/A	N/A	N/A	N/A	Secure	N/A	
	Camellia CBC[49][n 6]		N/A	N/A	Depends on mitigations	Depends on mitigations	Depends on mitigations	N/A	
	ARIA GCM[50][n 5]	256, 128	N/A	N/A	N/A	N/A	Secure	N/A	
	ARIA CBC[50][n 6]		N/A	N/A	Depends on mitigations	Depends on mitigations	Depends on mitigations	N/A	
	SEED CBC[51][n 6]	128	N/A	N/A	Depends on mitigations	Depends on mitigations	Depends on mitigations	N/A	
	3DES EDE CBC[n 6][n 7]	112[n 8]	Insecure	Insecure	Insecure	Insecure	Insecure	N/A	
	GOST 28147-89 CNT[45][n 7]	256	N/A	N/A	Insecure	Insecure	Insecure	N/A	Defined in RFC 4357
	IDEA CBC[n 6][n 7][n 9]	128	Insecure	Insecure	Insecure	Insecure	N/A	N/A	Removed from TLS 1.2
	DES CBC[n 6][n 7][n 9]	56	Insecure	Insecure	Insecure	Insecure	N/A	N/A	
		40[n 10]	Insecure	Insecure	Insecure	N/A	N/A	N/A	Forbidden in TLS 1.1 and later
	RC2 CBC[n 6][n 7]	40[n 10]	Insecure	Insecure	Insecure	N/A	N/A	N/A	
Stream cipher	ChaCha20-Poly1305[56][n 5]	256	N/A	N/A	N/A	N/A	Secure	Secure	Defined for TLS 1.2 in RFCs
	RC4[n 11]	128	Insecure	Insecure	Insecure	Insecure	Insecure	N/A	Prohibited in all versions of TLS by RFC 7465
		40[n 10]	Insecure	Insecure	Insecure	N/A	N/A	N/A	
None	Null[n 12]	–	N/A	Insecure	Insecure	Insecure	Insecure	N/A	Defined for TLS 1.2 in RFCs

Permitted key-exchange & authentication algorithms

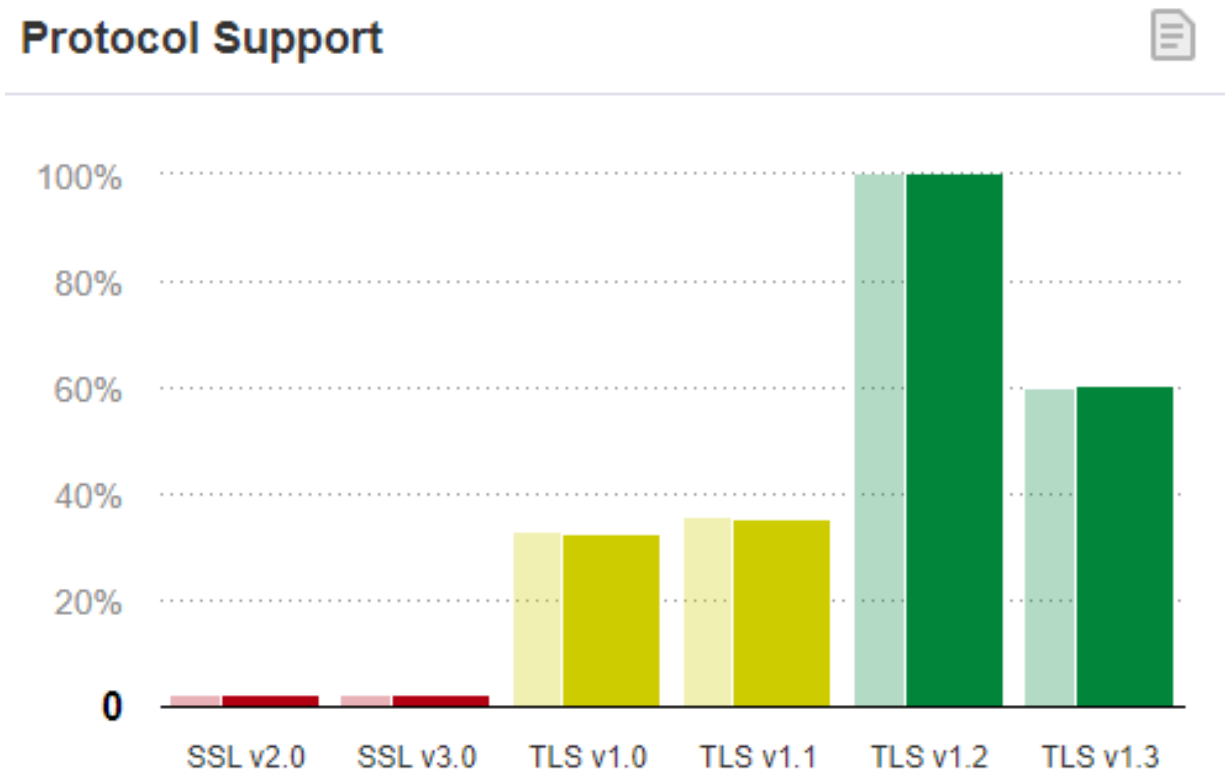
Key exchange/agreement and authentication

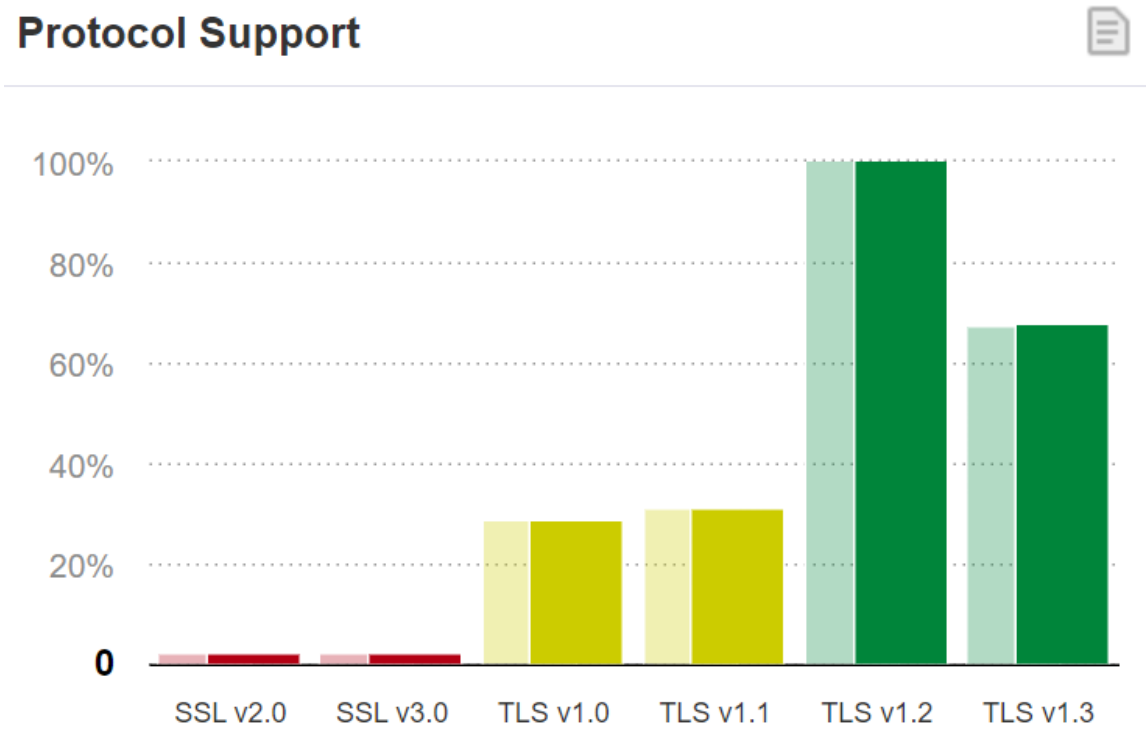
Algorithm	SSL 2.0	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	Status
RSA	Yes	Yes	Yes	Yes	Yes	No	
DH-RSA	No	Yes	Yes	Yes	Yes	No	
DHE-RSA (forward secrecy)	No	Yes	Yes	Yes	Yes	Yes	
ECDH-RSA	No	No	Yes	Yes	Yes	No	
ECDHE-RSA (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
DH-DSS	No	Yes	Yes	Yes	Yes	No	
DHE-DSS (forward secrecy)	No	Yes	Yes	Yes	Yes	No[44]	
ECDH-ECDSA	No	No	Yes	Yes	Yes	No	
ECDHE-ECDSA (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
PSK	No	No	Yes	Yes	Yes		Defined for TLS 1.2 in RFCs
PSK-RSA	No	No	Yes	Yes	Yes		
DHE-PSK (forward secrecy)	No	No	Yes	Yes	Yes		
ECDHE-PSK (forward secrecy)	No	No	Yes	Yes	Yes		
SRP	No	No	Yes	Yes	Yes		
SRP-DSS	No	No	Yes	Yes	Yes		
SRP-RSA	No	No	Yes	Yes	Yes		
Kerberos	No	No	Yes	Yes	Yes		
DH-ANON (insecure)	No	Yes	Yes	Yes	Yes		
ECDH-ANON (insecure)	No	No	Yes	Yes	Yes		
GOST R 34.10-94 / 34.10-2001[45]	No	No	Yes	Yes	Yes		Proposed in RFC drafts

Permitted integrity algorithms

	Data integrity						
Algorithm	SSL 2.0	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	Status
HMAC-MD5	Yes	Yes	Yes	Yes	Yes	No	Defined for TLS 1.2 in RFCs
HMAC-SHA1	No	Yes	Yes	Yes	Yes	No	
HMAC-SHA256/384	No	No	No	No	Yes	No	
AEAD	No	No	No	No	Yes	Yes	
GOST 28147-89 IMIT[45]	No	No	Yes	Yes	Yes		Proposed in RFC drafts
GOST R 34.11-94[45]	No	No	Yes	Yes	Yes		

Protocol support





Source: https://www.ssllabs.com/ssl-pulse/

Outline

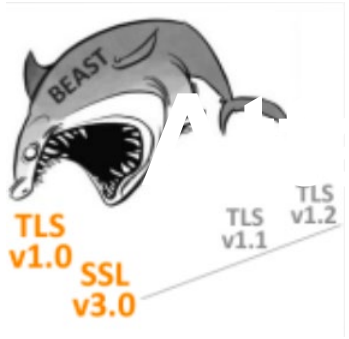
- **Introduction**
- **Certificates**
- **TLS**
  - What it can offer
  - TLS protocol
  - TLS resumption
- **Beyond HTTP**
- **Attacks on TLS**
- **Lab exercise**

Other TLS based Applications

- DTLS: Datagram Transport Layer Security (DTLS)
  –Provides security for **UDP datagrams**
- SMTPS: Simple Mail Transfer Protocol (SMTP) over TLS
- IMAPS: Internet Message Access Protocol (IMAP) over TLS
- POP3S: Post Office Protocol 3 (POP3) over TLS
- DoT: Domain Name System (DNS) over TLS
- FTPS: File Transfer Protocol (FTP) over TLS
- IRCS: Internet Relay Chat (IRC) over TLS
- …

Outline

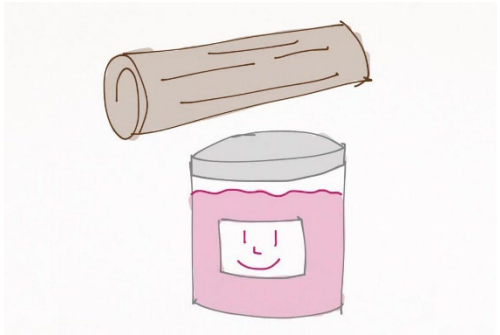
- **Introduction**
- **Certificates**
- **TLS**
  - What it can offer
  - TLS protocol
  - TLS resumption
- **Beyond HTTP**
- **Attacks on TLS**
- **Lab exercise**

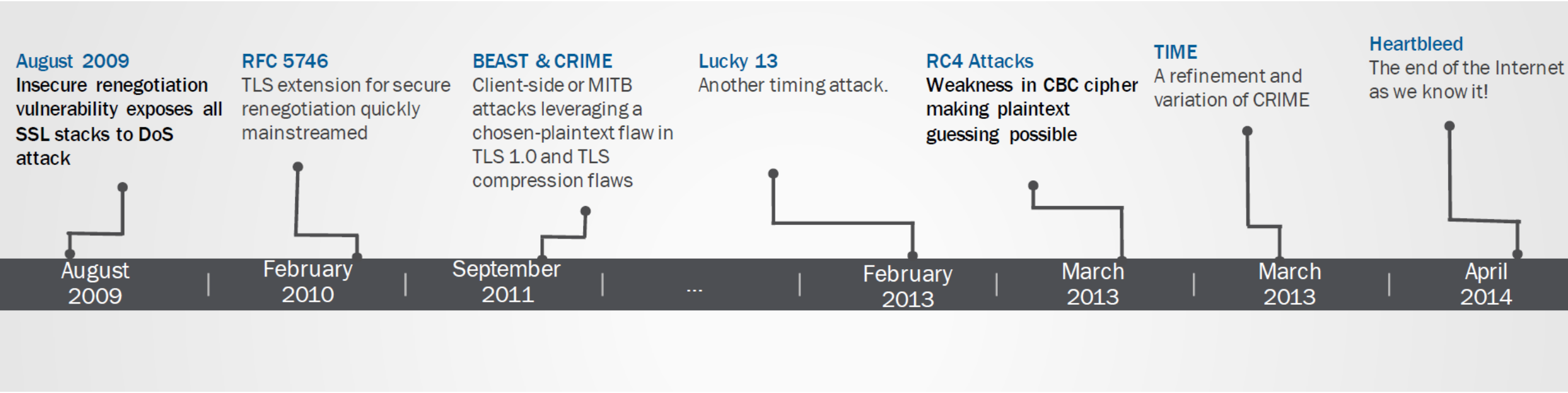

BEAST
TLS v1.0
SSL v3.0
TLS v1.1
TLS v1.2


ssl/tls
SSL/TLS Vulnerability


vunerability
TLS
warning
HACK


LUCKY
13




August 2009
Insecure renegotiation vulnerability exposes all SSL stacks to DoS attack
August 2009
RFC 5746
TLS extension for secure renegotiation quickly mainstreamed
February 2010
BEAST & CRIME
Client-side or MITB attacks leveraging a chosen-plaintext flaw in TLS 1.0 and TLS compression flaws
September 2011
Lucky 13
Another timing attack.
February 2013
RC4 Attacks
Weakness in CBC cipher making plaintext guessing possible
March 2013
TIME
A refinement and variation of CRIME
March 2013
Heartbleed
The end of the Internet as we know it!
April 2014
...

Attacks on TLS

Not many that work against TLS 1.2 or 1.3:

- Cipher-specific attacks (e.g., RC4)
- Compression attacks (e.g., CRIME, TIME, BREACH)
- Downgrade attacks (e.g., POODLE, FREAK)
- Padding Oracle attacks (e.g., Lucky13)
- Implementation attacks (e.g., Heartbleed)

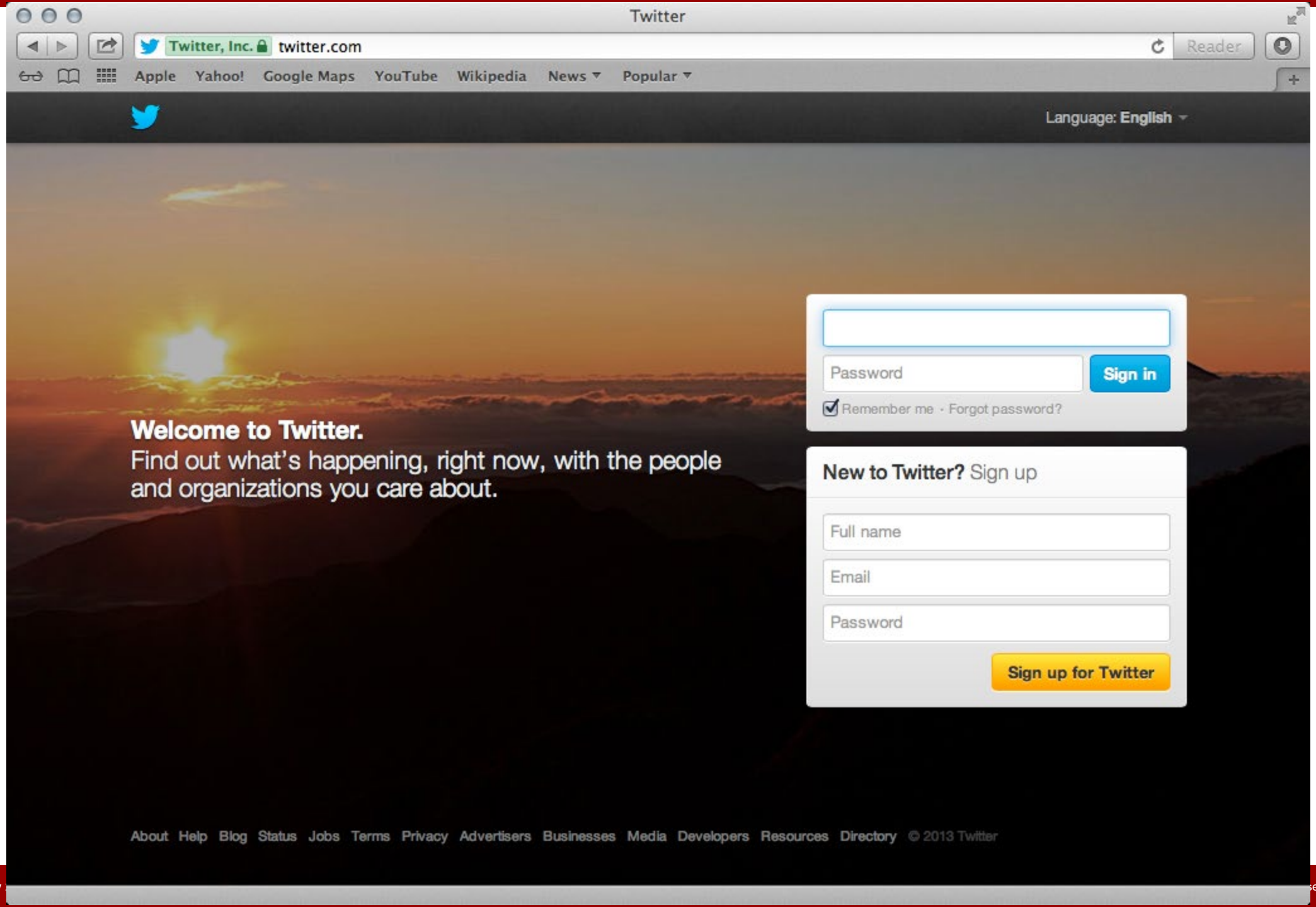
The most used practical attack: sslstrip

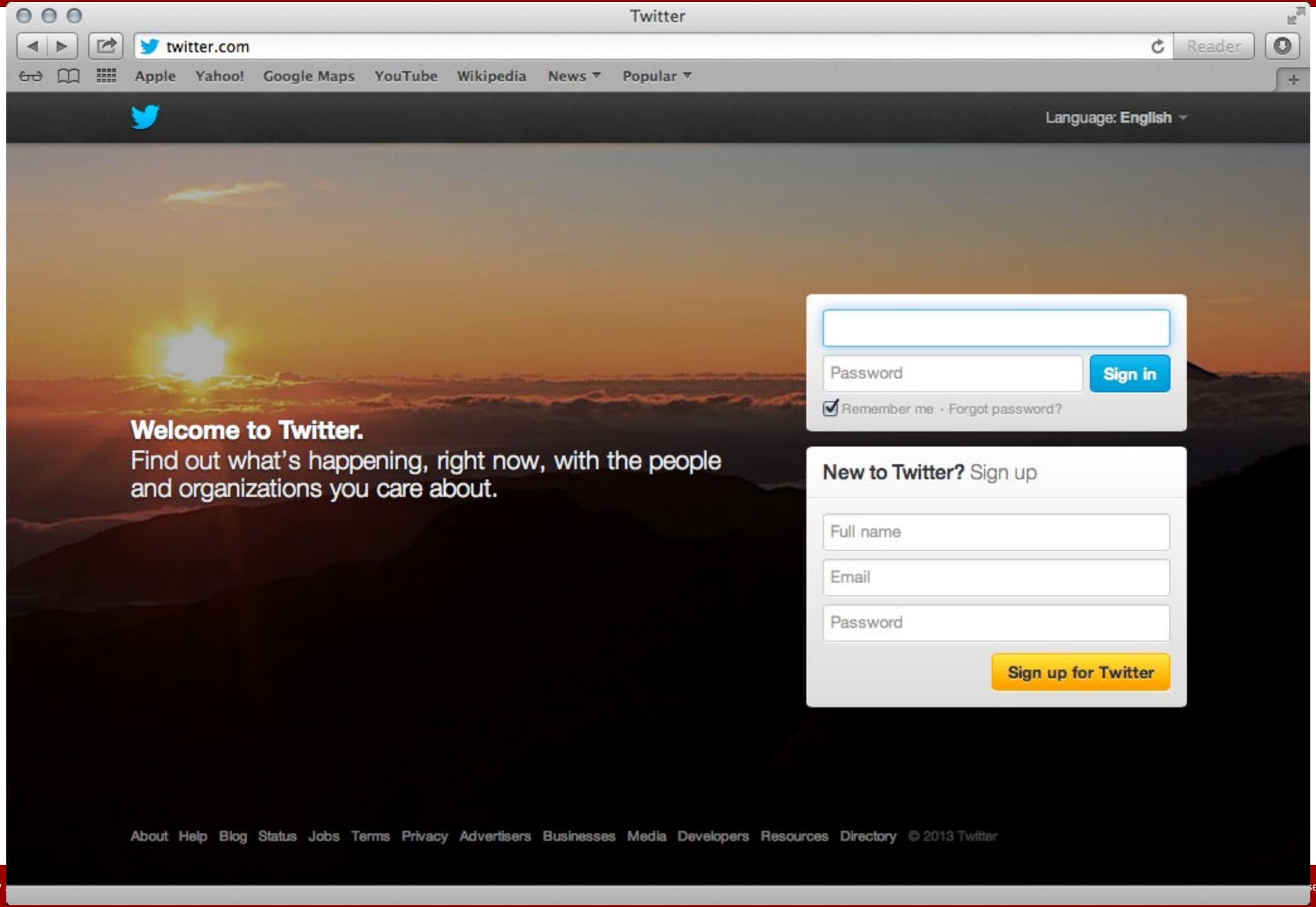
- Original presentation: Moxie Marlinspike, Black Hat 2009
  https://www.youtube.com/watch?v=MFoI6IMbZ7Y

- sslstrip is a tool that
  - transparently hijacks HTTP traffic on a network
  - watch for HTTPS links and redirects, and then map those links into look-alike HTTP links or homograph-similar HTTPS links
  - also supports modes for supplying a favicon which looks like a lock icon, selective logging, and session denial.

The most used (practical) attack

- Original presentation: https://www.youtube.com/watch?v=MFol6IMbZ7Y


Twitter
Twitter, Inc.
twitter.com
Reader
Apple
Yahoo!
Google Maps
YouTube
Wikipedia
News
Popular
Language: English
Welcome to Twitter.
Find out what's happening, right now, with the people and organizations you care about.
Password
Sign in
Remember me · Forgot password?
New to Twitter? Sign up
Full name
Email
Password
Sign up for Twitter
About Help Blog Status Jobs Terms Privacy Advertisers Businesses Media Developers Resources Directory © 2013 Twitter


Twitter
twitter.com
Reader
Apple
Yahoo!
Google Maps
YouTube
Wikipedia
News
Popular
Language: English
Password
Sign in
Remember me · Forgot password?
Welcome to Twitter.
Find out what's happening, right now, with the people and organizations you care about.
New to Twitter? Sign up
Full name
Email
Password
Sign up for Twitter
About Help Blog Status Jobs Terms Privacy Advertisers Businesses Media Developers Resources Directory © 2013 Twitter

Is it still possible? (usually not)

- Depends on
  - –Website/server
  - –User
  - –Browser

- Mitigation: **HTTP Strict Transport Security**
  - –Policy, RFC 6797
  - –Web-servers demand browsers to HTTPS only

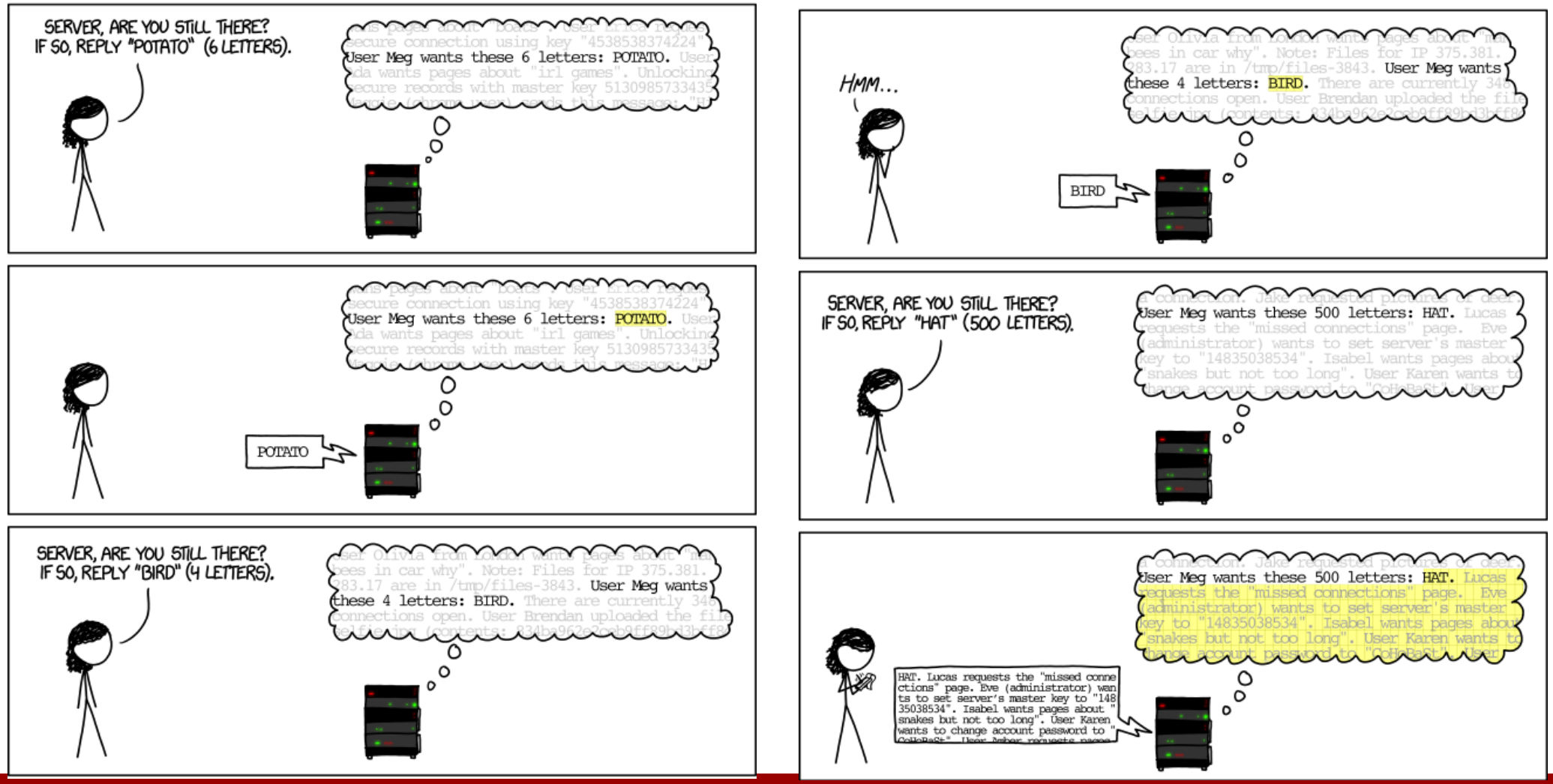
HTTP Strict Transport Security

- Limitations
  - Initial request still vulnerable
    - The protection only applies after a user has visited the site at least once
  - TLS attacks: e.g., Beast, Crime, etc. not mitigated
  - DNS spoofing can circumvent the policy

Heartbleed at a glance

- 01.04.**14**
- Implementation-specific bug of **OpenSSL**
- **Buffer-over-read** type of vulnerability
- Improper input validation (due to a missing bounds check) in the implementation of the TLS heartbeat extension
- **Huge** impact (amazon, Github, Wikipedia, Tumblr, Reddit, etc.)

xkdc Heartbleed Explanation



Heartbleed at a glance

Some more attacks

- **Poodle attack** (downgrade attack):

https://www.youtube.com/watch?v=C8ks8WLoZto

- Blackhat Asia 15, **Bar-Mitzva Attack** (RC4-based)

https://www.youtube.com/watch?v=AHkUCD_EYok

- **FREAK attack** (RSA downgrade attack)

https://en.wikipedia.org/wiki/FREAK

Some more attacks

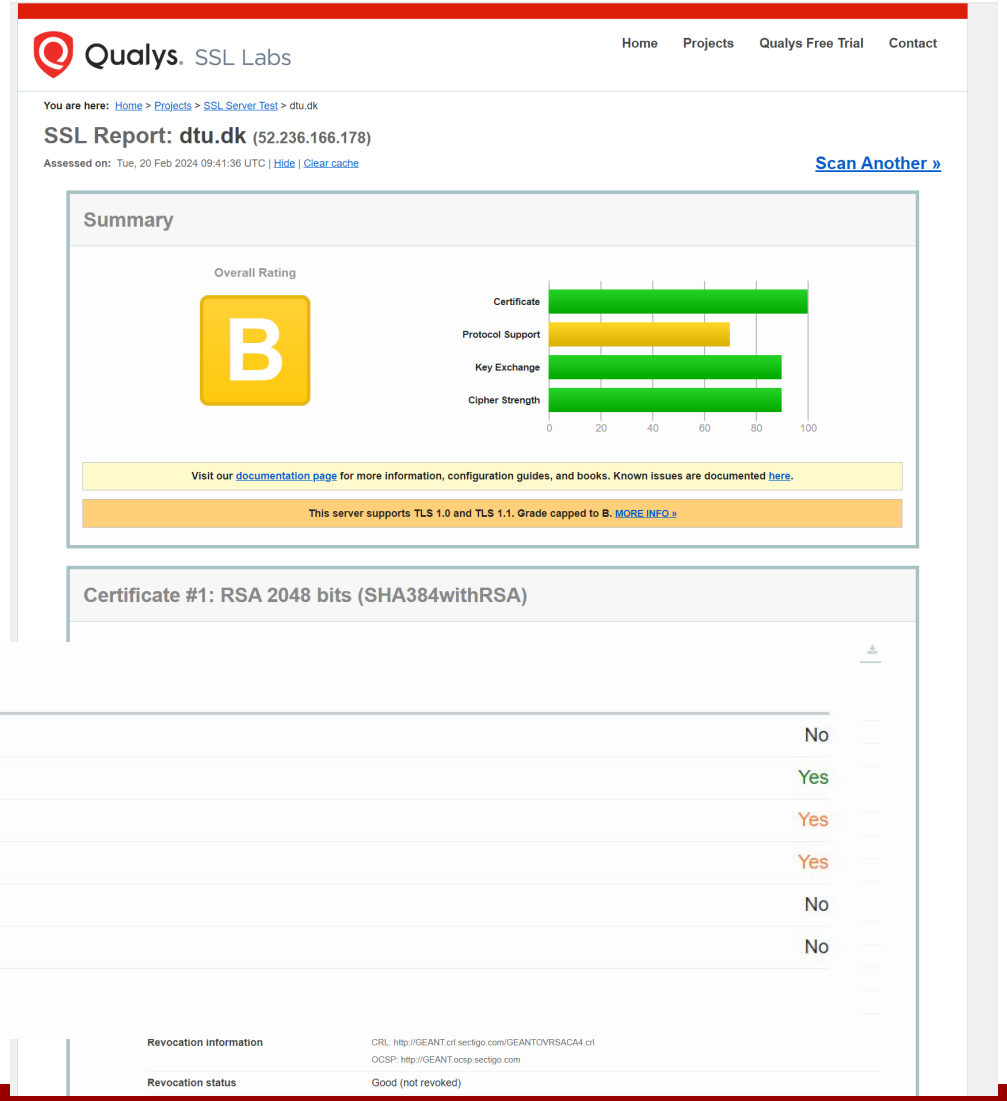
The 9 Lives of Bleichenbacher's CAT: New Cache ATtacks on TLS Implementations

See: https://eyalro.net/project/cat.html

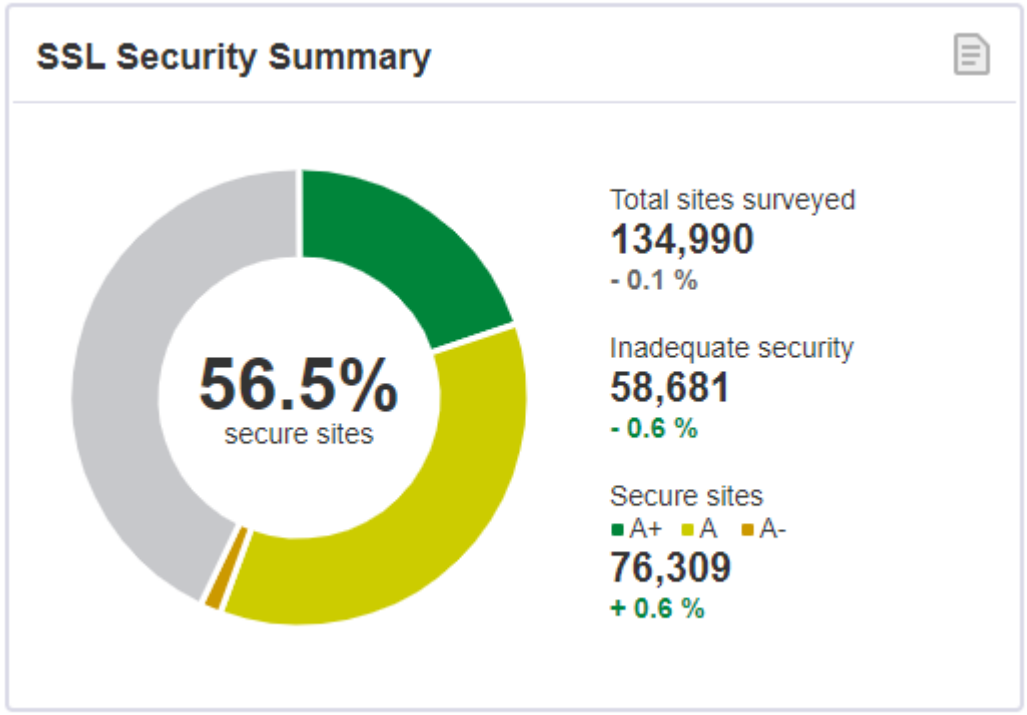
Works against TLS1.3 implementations too (by downgrading and attacking RSA)

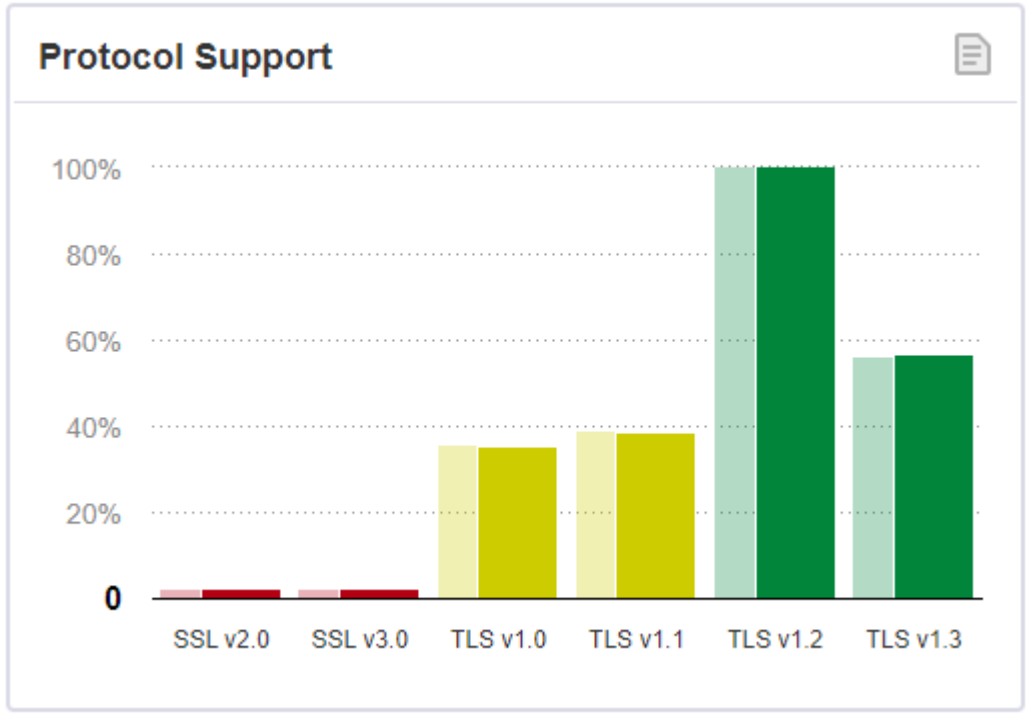
Testing TLS implementations

- https://www.ssllabs.com offers a very nice scanning tool
- Also https://www.ssllabs.com/ssl-pulse/ provides some global statistics about TLS usage worldwide



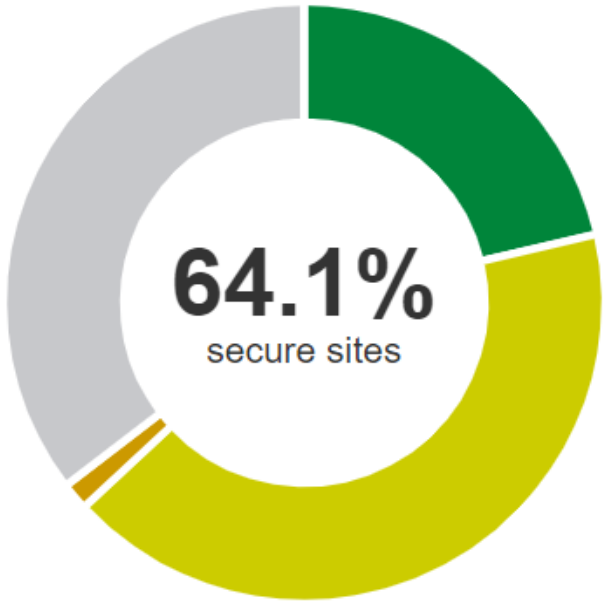
From (2023) https://www.ssllabs.com/ssl-pulse





From (2024) https://www.ssllabs.com/ssl-pulse

SSL Security Summary

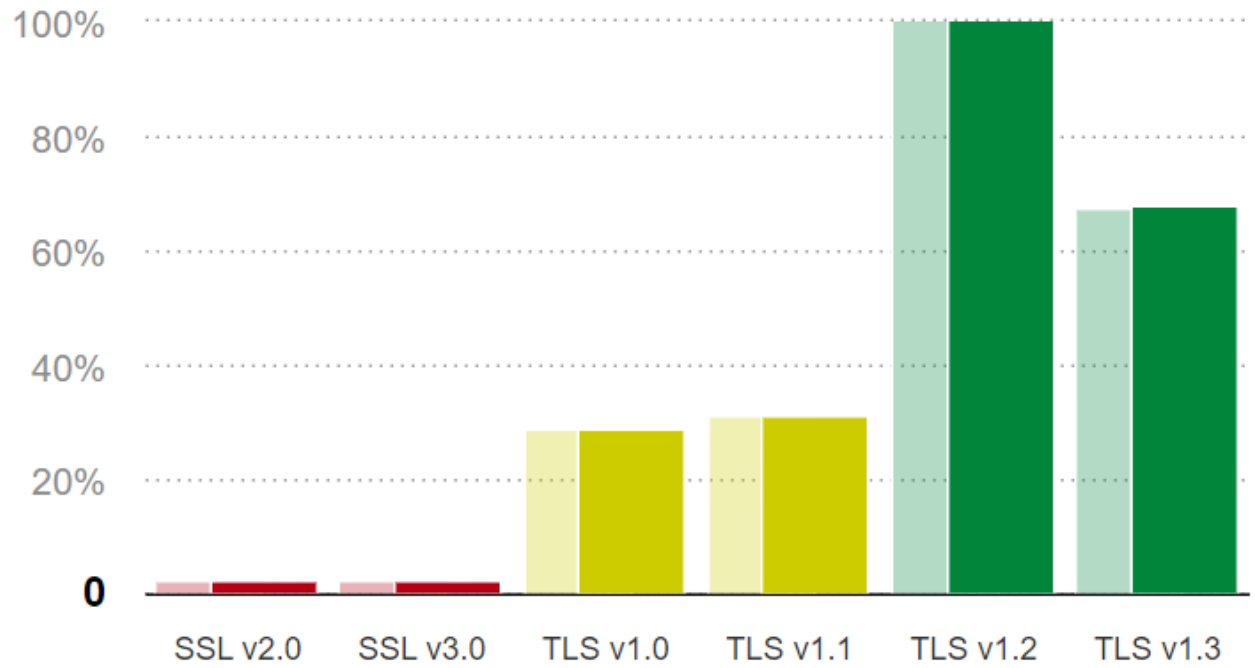


Total sites surveyed
134,595
- 0.4 %

Inadequate security
48,330
- 0.6 %

Secure sites
■ A+ ■ A ■ A-
86,265
+ 0.6 %

Protocol Support



Outline

- **Introduction**
- **Certificates**
- **TLS**
  - What it can offer
  - TLS protocol
  - TLS resumption
- **Beyond HTTP**
- **Attacks on TLS**
- **Lab exercise**

Conclusion

- **Transport Layer Security**
  - –One of THE most important security protocols in use
  - –Various layers/sub-protocols
    - Handshake is the most complicated one
  - –Remember that most connections are TLS Resumptions
  - –Used for many, many different things
    - HTTPS, email (kinda), TOR, voip, etc.

Further reading

- **Deploying TLS 1.3: the great, the good and the bad (33c3) from Cloudflare**
  - https://www.youtube.com/watch?v=0opakLwtPWk

- **Certificate transparency**
  - https://certificate.transparency.dev/

Lab exercises

- Get hands-on experience with **self-signed certificates** using openssl

- Get hands-on experience with **setting up a test HTTPS** Web server using openssl

- Replicate an example of **HTTP** and **Wireshark**

- Play with sslstrip (optional)