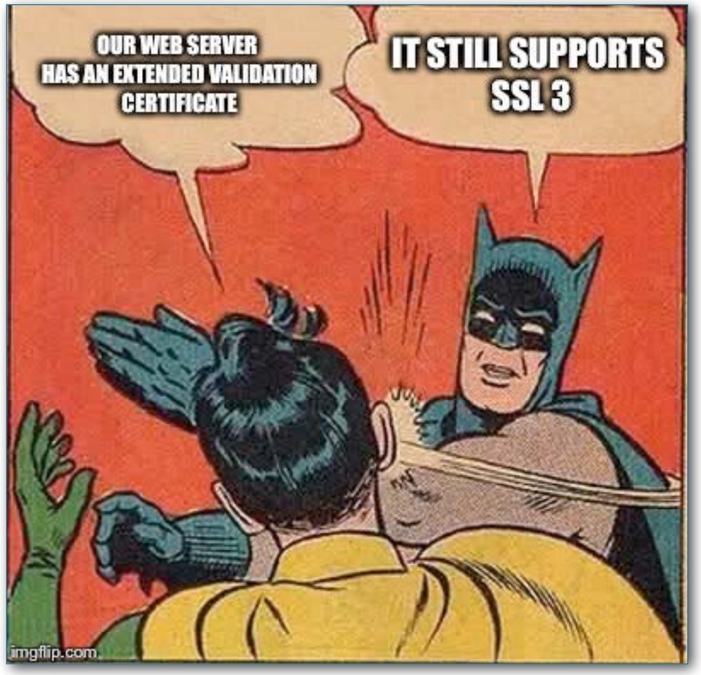
SSL/TLS troubleshooting with Wireshark

Sharkfest 2020 virtual pre-conference class

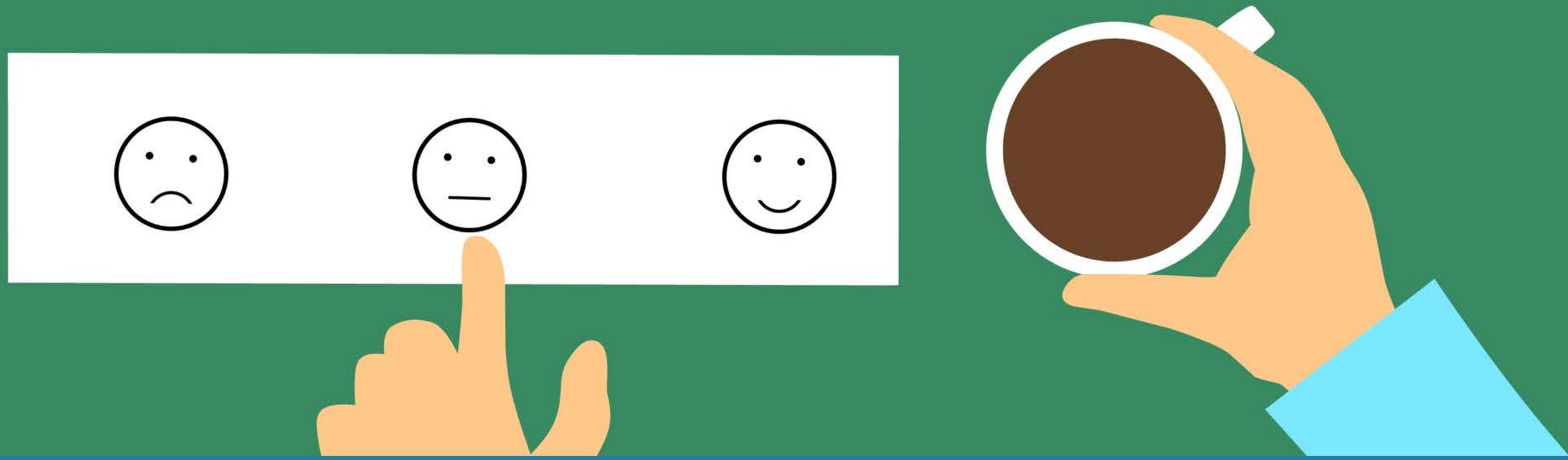
Sake Blok
Relational therapist for computer systems
sake.blok@SYN-bit.nl

SYN-bit
deep traffic analysis



Getting to know each other...

Time for a poll



Do you trust these companies?

whoami

- Relational therapist for computer systems
  - Solve {application,network,performance}-issues by looking at the communication between systems
- Member Wireshark core-team since 2007
- Started SYN-bit in 2010
  - Application and Network troubleshooting
  - Protocol and packet analysis
  - Training (Wireshark, TCP, TLS)

SYN-bit

deep traffic analysis

Application and network troubleshooting

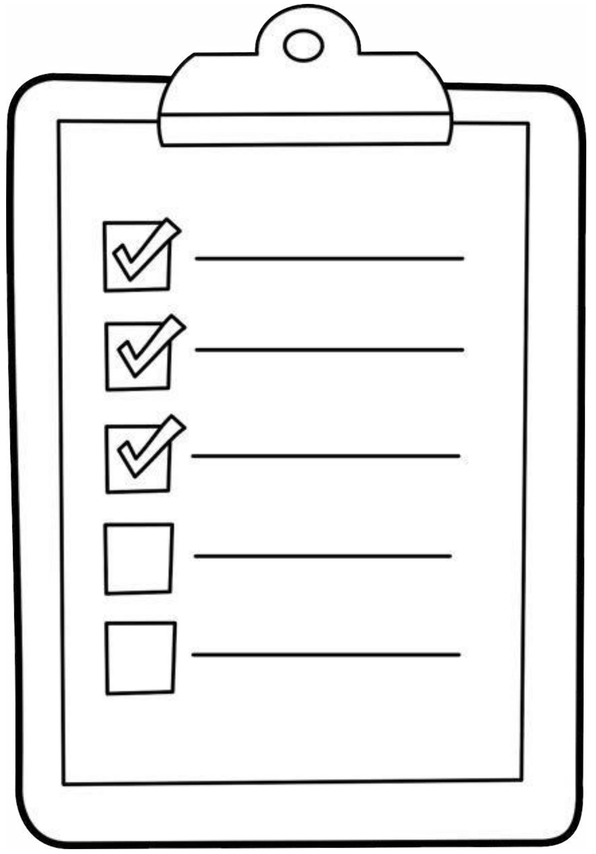
Protocol and packet analysis

Training (Wireshark, TCP, SSL)

www.SYN-bit.nl

Agenda

- TLS fundamentals
  - The need for TLS (and what happend to SSL?)
  - Cryptology 101 & PKI
  - Troubleshoot the TLS handshake (SSLv3 - TLSv1.2)
  - Troubleshoot the TLS handshake (TLSv1.3)
- Analysing Application Data
  - Without decryption
  - With decryption using the servers private key
  - With decryption using TLS session keys
- Creating configuration profiles for TLS analysis
- SSL/TLS vulnerabilities & privacy concerns
- Summary and Q&A



https://www.needpix.com/photo/download/94588/checklist-action-check-list-verification-tick-approved-okay-check-mark

Day planning(-ish)

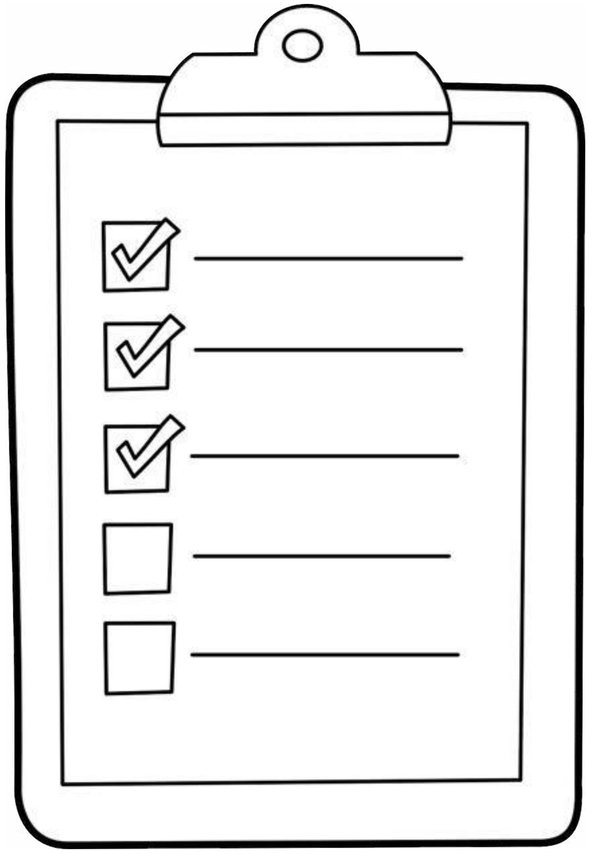
	San Francisco (PDT, -7)	New York (EDT, -4)	London (BST, +1)	Amsterdam (CEST, +2)	Perth (AWST, +8)	Sydney (AEDT, +11)	Wellington (NZDT, +13
Part I	8:00 - 11:00	11:00 - 14:00	16:00 - 19:00	17:00 - 20:00	23:00 - 2:00	2:00 - 5:00	4:00 - 7:00
Long Break (lunch/diner/snack/breakfast?)							
Part II	12:00 - 15:00	15:00 - 18:00	20:00 - 23:00	21:00 - 0:00	3:00 - 6:00	6:00 - 9:00	8:00 - 11:00
Short Break (tea/diner/breakfast/coffee?)							
Part III	15:30 - 17:00	18:30 - 20:00	23:30 - 1:00	0:30 - 2:00	6:30 - 8:00	9:30 - 11:00	11:30 - 13:00

Some Zoom Guidelines

- Please mute yourself
  - if I hear too much noise, I will just mute everyone :-)
- Share your video for a more personal touch
  - But feel free to stop video too of course
- Ask questions in the chat, I'll try to read it often
  - Please don't start chatting, then I can't see the questions
  - Use discord for chatting with each other, I'll read that later
- Raise your hand if you want to say something
  - And wait for me to unmute you please
- If I ask you a question by mentioning your name, please unmute yourself to answer
  - Please don't forget to mute yourself after the discussion :-)

Agenda

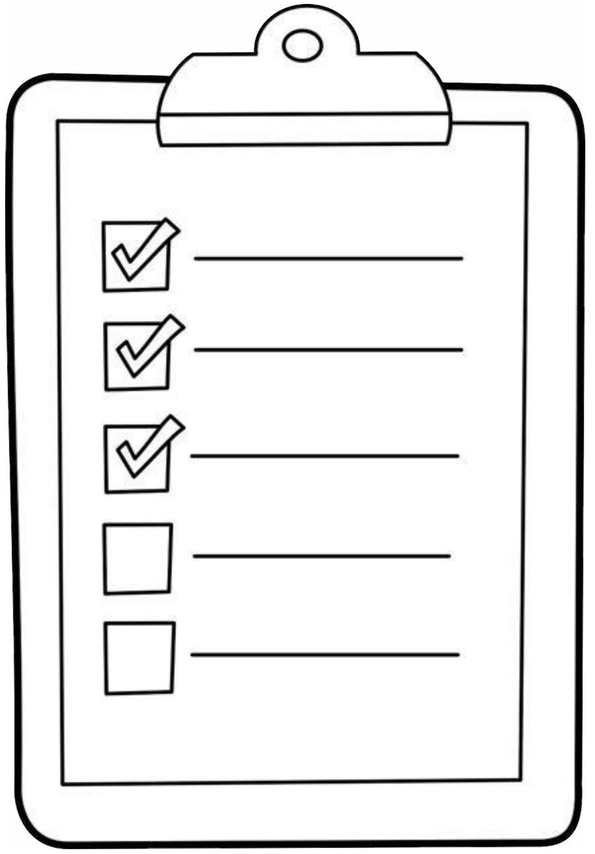
- **TLS fundamentals**
  - The need for TLS (and what happend to SSL?)
  - Cryptology 101 & PKI
  - Troubleshoot the TLS handshake (SSLv3 - TLSv1.2)
  - Troubleshoot the TLS handshake (TLSv1.3)
- Analysing Application Data
  - Without decryption
  - With decryption using the servers private key
  - With decryption using TLS session keys
- Creating configuration profiles for TLS analysis
- SSL/TLS vulnerabilities & privacy concerns
- Summary and Q&A



https://www.needpix.com/photo/download/94588/checklist-action-check-list-verification-tick-approved-okay-check-mark

Agenda

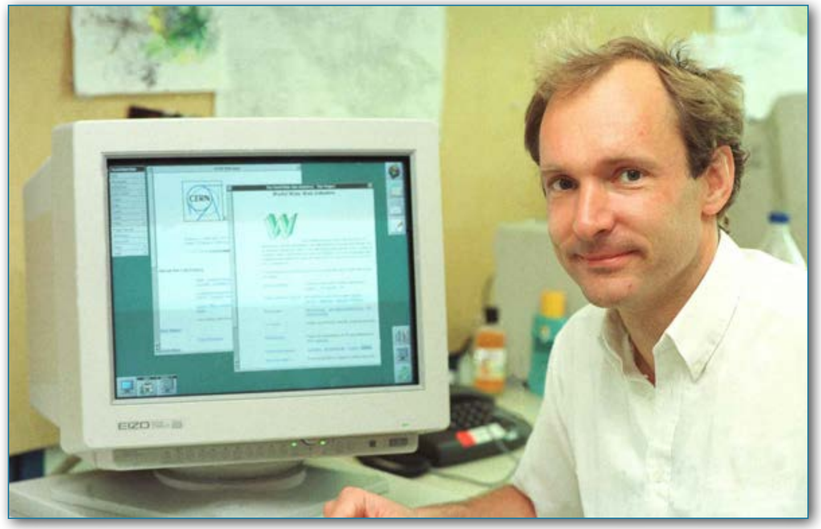
- TLS fundamentals
  - **The need for TLS (and what happend to SSL?)**
  - Cryptology 101 & PKI
  - Troubleshoot the TLS handshake (SSLv3 - TLSv1.2)
  - Troubleshoot the TLS handshake (TLSv1.3)
- Analysing Application Data
  - Without decryption
  - With decryption using the servers private key
  - With decryption using TLS session keys
- Creating configuration profiles for TLS analysis
- SSL/TLS vulnerabilities & privacy concerns
- Summary and Q&A



https://www.needpix.com/photo/download/94588/checklist-action-check-list-verification-tick-approved-okay-check-mark

Birth of HTTP and SSL/TLS

- 1989 invention of HTML, HTTP and URLs by Tim Berners-Lee (and team)
- Mid 90's, Internet made its way to the public
  - NCSA Mosaic (1993)
  - Netscape (1994)
  - Internet Explorer (1995)
- Start of e-commerce
  - but how to safely use credit card numbers?
- Secure Socket Layer & Transport Layer Security
  - Makes use of Public Key Infrastructure (PKI)



https://cds.cern.ch/record/39437

SSL/TLS History

- SSLv1 by Netscape (unreleased, 1994)
- SSLv2 by Netscape (v2-draft,1994)
  - deprecated in 2011 (RFC 6176)
- SSLv3 by Netscape (v3-draft, 1995)
  - deprecated in June 2015 (RFC 7568)
- TLSv1.0, IETF (RFC 2246, 1999)
  - Deprecated March 31, 2020
- TLSv1.1, IETF (RFC 4346, 2006)
  - Deprecated March 31, 2020
- **TLSv1.2, IETF (RFC 5246, 2008)**
- **TLSv1.3, IETF (RFC 8446, 2018)**

Public Key Infrastructure?

A public key infrastructure (PKI) is a **set of roles, policies, and procedures** needed to create, manage, distribute, use, store, and revoke **digital certificates** and manage **public-key encryption**.

The purpose of a PKI is to facilitate the **secure electronic transfer of information** for a range of network activities such as **e-commerce, internet banking and confidential email.**

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

Some methods of PKI certification

- Certificate Authorities
  - X.509
- Web of Trust
  - PGP, OpenPGP, GnuPG
- Simple Public Key Infrastructure
- Blockchain based

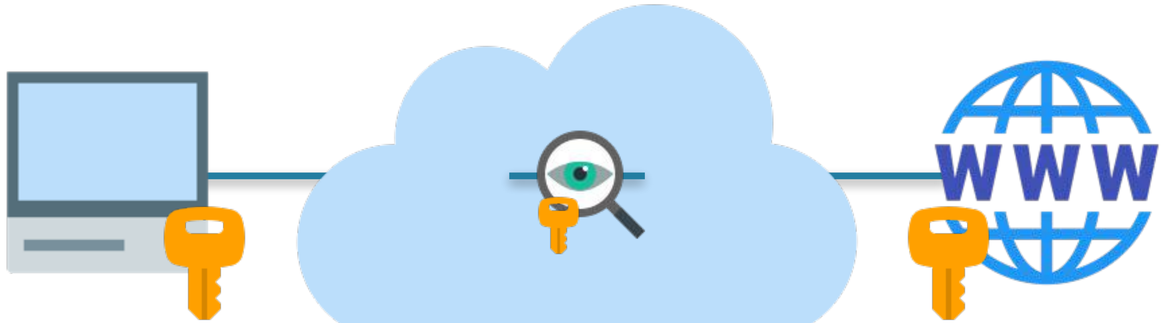
Methods of PKI certification

- **Certificate Authorities**
  - **X.509**
- Web of Trust
  - PGP, OpenPGP, GnuPG
- Simple Public Key Infrastructure
- Blockchain based

Why TLS and Certificates?

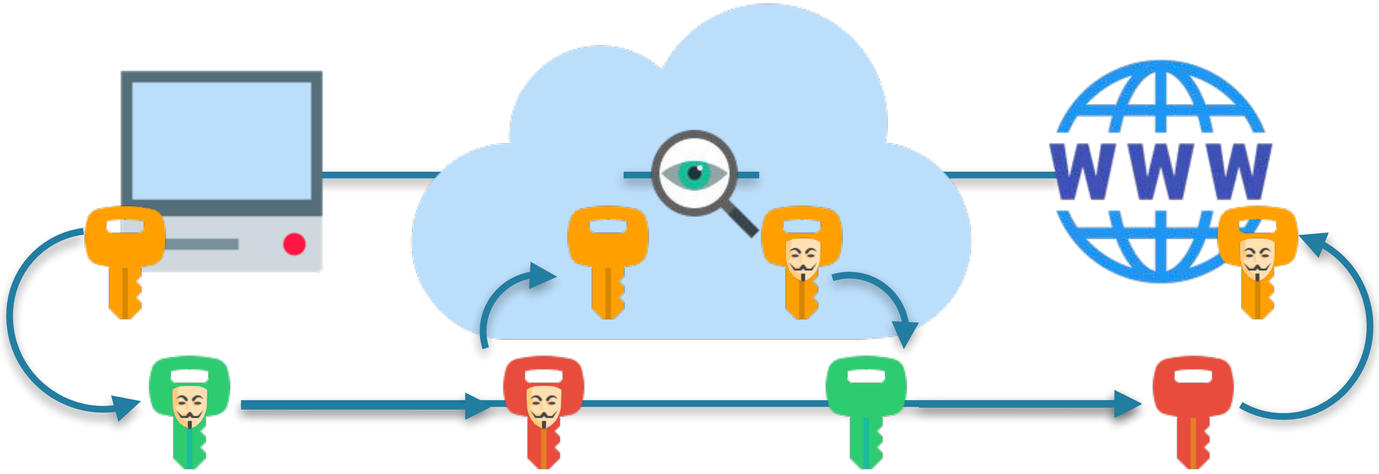
Step 1: symmetric keys

- Server creates a symmetric key
- Sends key to client through secure channel
- Encrypt traffic with symmetric key
- Keys need to be unique to prevent evesdropping
- **Need a solution for large scale key distribution**



Step 2: asymmetric keys

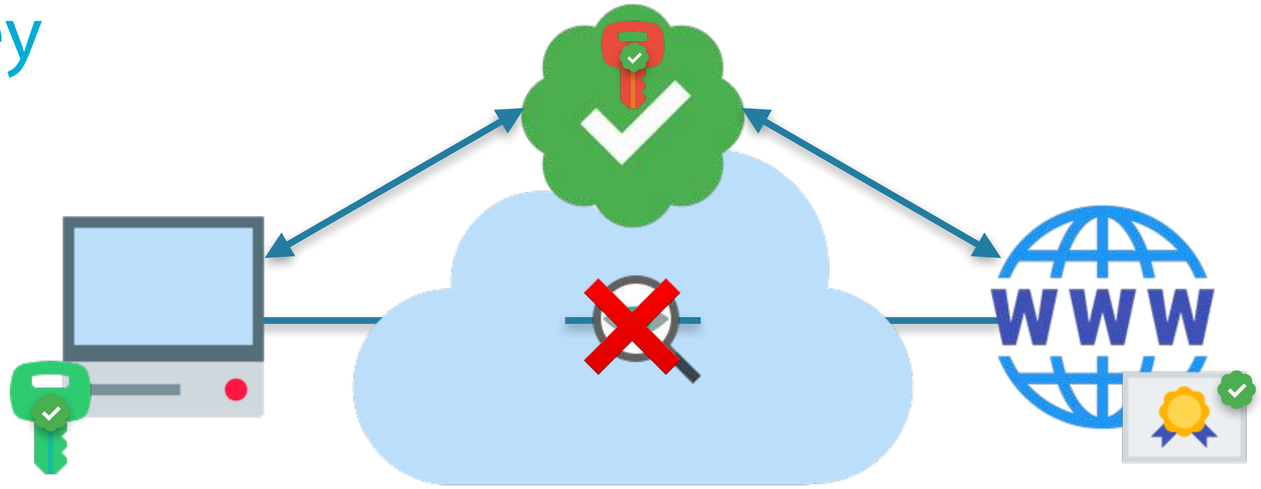
- Server creates public/private key-pair
- ...and sends public key to client
- Clients creates (symmetric) session key, encrypts with public key
- Client sends encrypted session key to server
- Server decrypts session key with private key
- **But how to prevent MITM?**



Step 3: certificates

- Agree on a trusted 3rd party (the certificate authority or CA)
- CA signs ID+PUB of the server with its private key => certificate
- Server sends certificate to client, client checks with public key of CA
- Now client knows who it is talking to and can trust the public key
- Client creates session key and encrypts with public key
- Client sends encrypted session key

✅ Confidentiality
✅ Integrity
✅ Authentication

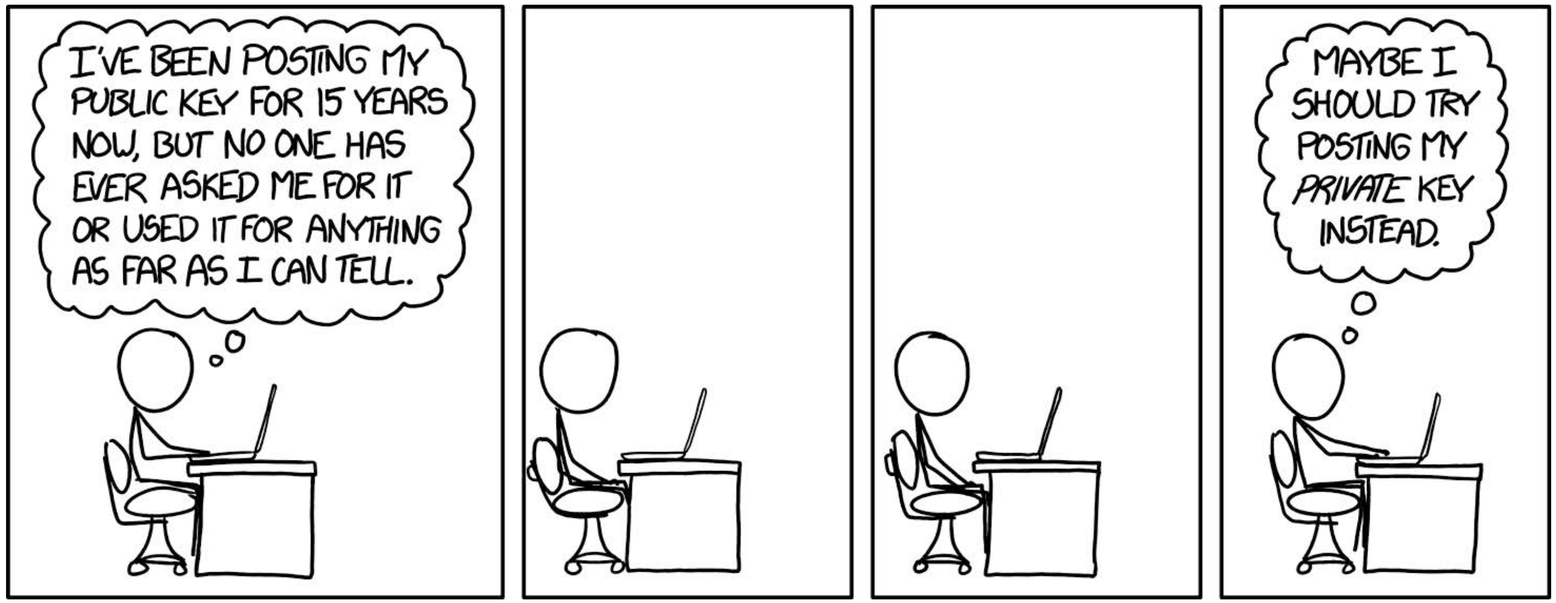


How can we accomplish these goals?

- Authentication
  - We want to know who we are talking to
  - By using: certificates and certificate authorities (the PKI)
- Confidentiality
  - We want to make sure nobody can read our messages
  - By using: symmetric encryption and decryption
- Integrity & non-repudiation
  - We want to be sure that our messages are not changed
  - By using: message digests and signing

Agenda

- TLS fundamentals
  - The need for TLS (and what happend to SSL?)
  - **Cryptology 101 & PKI**
  - Troubleshoot the TLS handsh
  - Troubleshoot the TLS handsh
- Analysing Application D
  - Without decryption
  - With decryption using the se
  - With decryption using TLS se
- Creating configuration p
- SSL/TLS vulnerabilities & privacy concerns
- Summary and Q&A



https://xkcd.com/1553/

History of cryptography

- It's not new!
  - 1900 BC: Non-standard hieroglyphs
  - 1500 BC: Recipe for pottery glaze
  - 400 BC to 200 AD: Communication between lovers
  - Ancient Greeks: Used by Spartan Military
- Techniques
  - Pen & paper (or simple mechanics)
  - Complex (electro)mechanical devices
  - Computing
- From governments to private persons
  - Big leap caused by e-commerce
- **Cryptoanalysis**

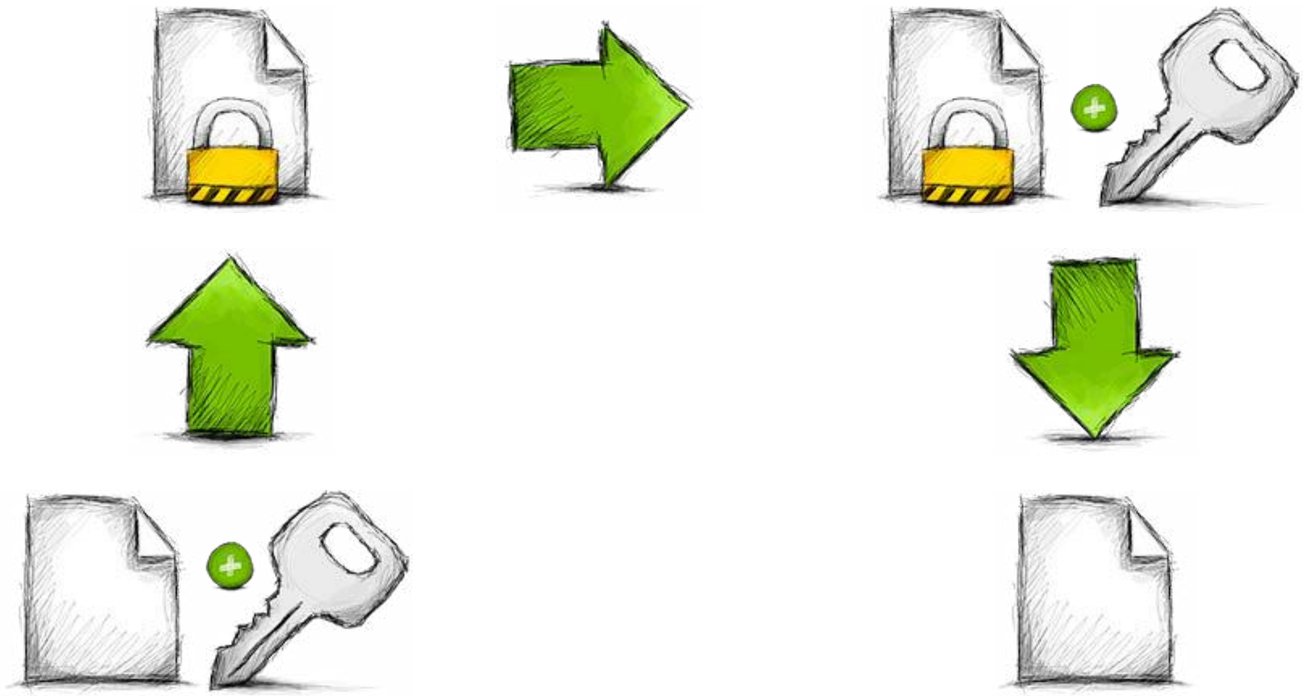
The weakest link

"Much of the German cipher traffic was encrypted on the Enigma machine. Used properly, the German military Enigma would have been virtually unbreakable; in practice, shortcomings in operation allowed it to be broken."

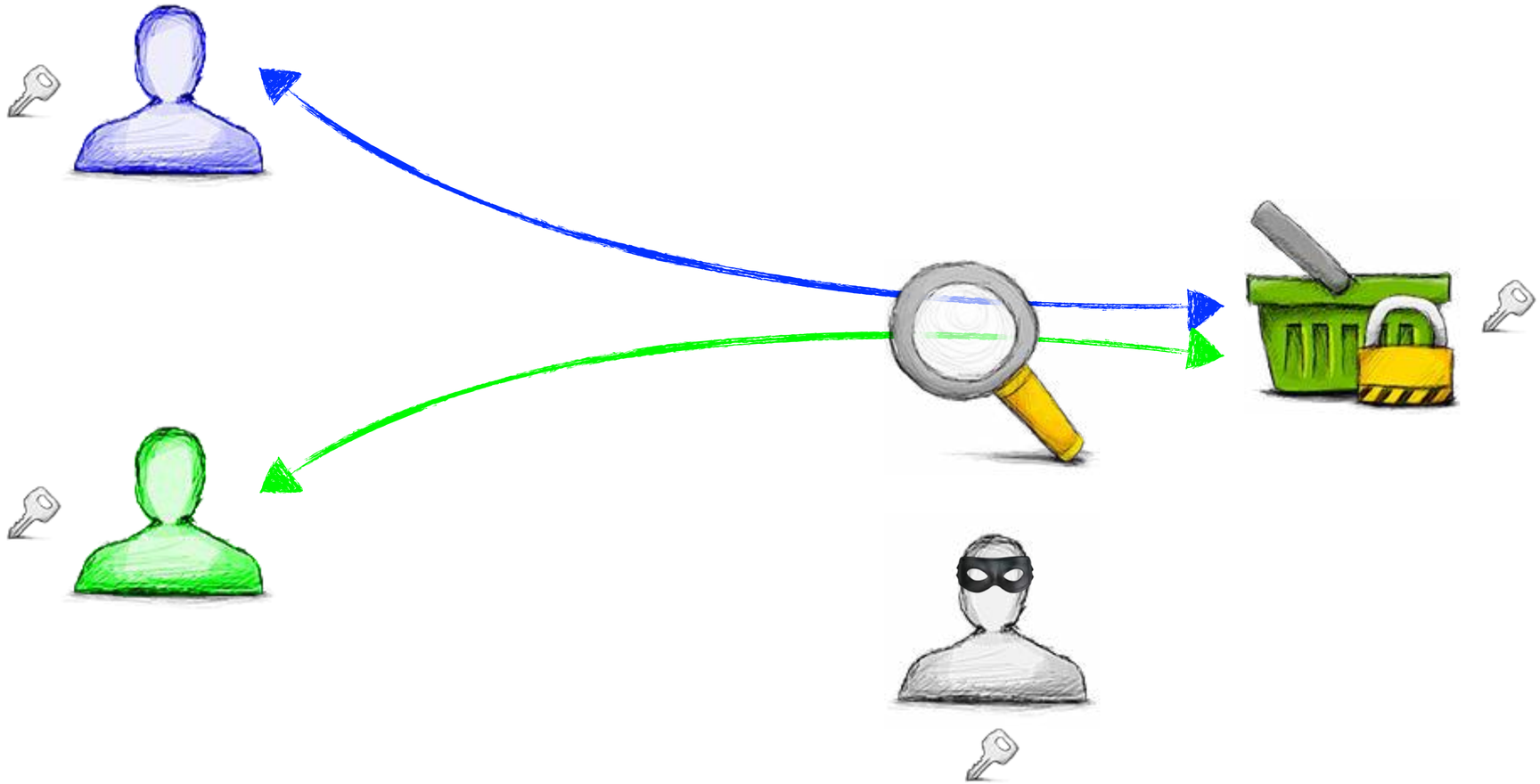
[https://en.wikipedia.org/wiki/Ultra]

Symmetrical Encryption

- Same key for encryption and decryption
- Computative "cheap"
- Short key lengths (typically 40-256 bits)
- ~~DES, 3DES, RC4~~, AESxxx

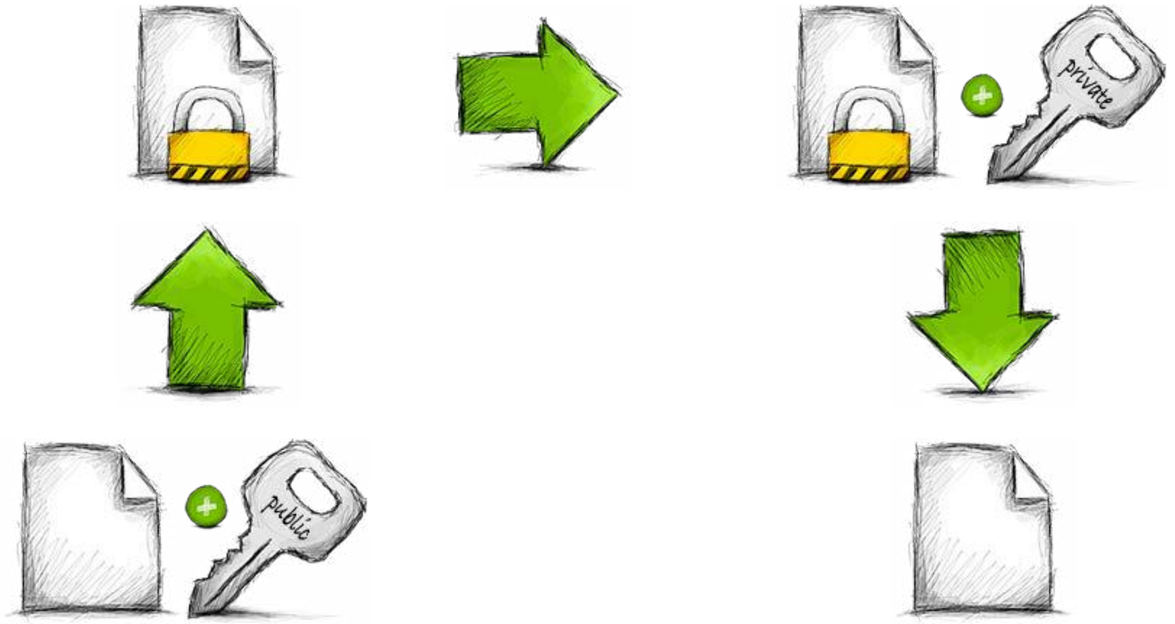


Symmetrical Session Key



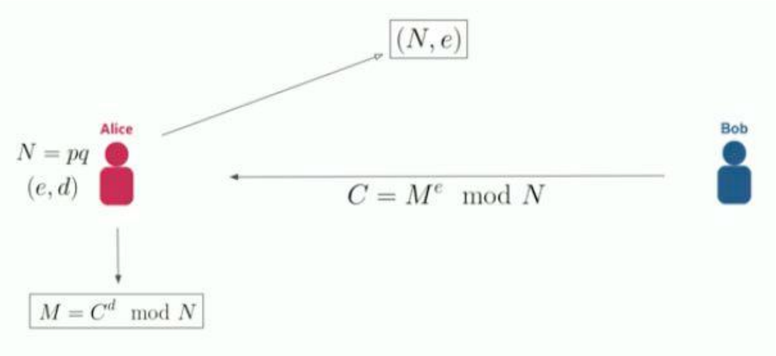
Asymmetrical Encryption

- One key for encryption, second key for decryption (both keys form a pair)
- Computative "expensive"
- Long keys (RSA: 512-4096 bits)
- RSA, DSA, ECC



RSA

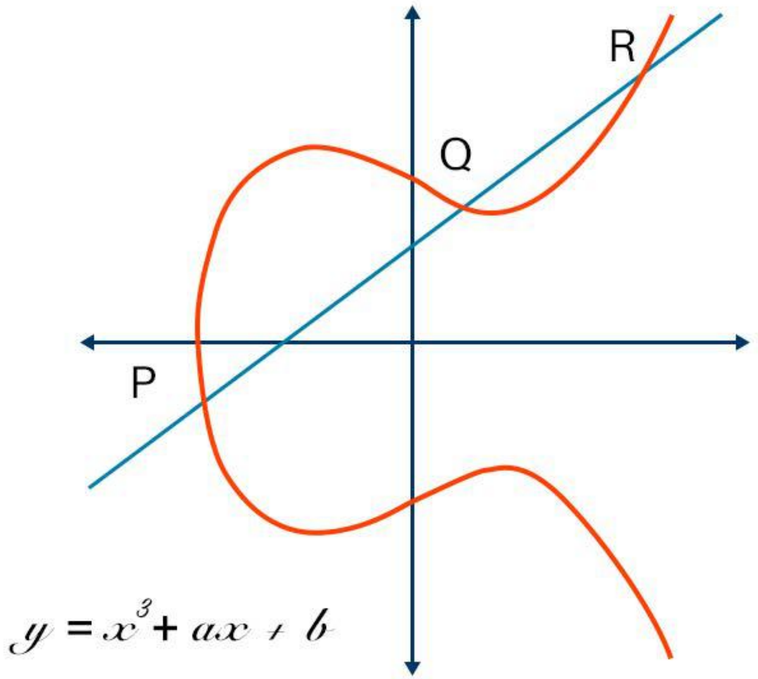
- Ron **R**ivest, Adi **S**hamir, and Leonard **A**dleman
- Publicly described the algorithm in 1977
- Public key is product of two large primes
- Keep primes private
- Relies on "factoring problem"
- In ~25 years(?), a quantum computer could break 2048-bit RSA encryption in ~8 hours???
- "Seriously, stop using RSA" (unless done right)
  - https://blog.trailofbits.com/2019/07/08/fuck-rsa/



https://blog.trailofbits.com/2019/07/08/fuck-rsa/

Elliptic-curve cryptography (ECC)

- suggested independently by Neal Koblitz and Victor S. Miller in 1985
- Based on the algebraic structure of elliptic curves over finite fields
- Wide in use since 2004/2005 (NSA Suite B)
- Good for mobile devices and IoT
  - Smaller key size, reducing storage and transmission
  - Less computing needed
- Can be broken with quantum computing (little easier even than RSA)

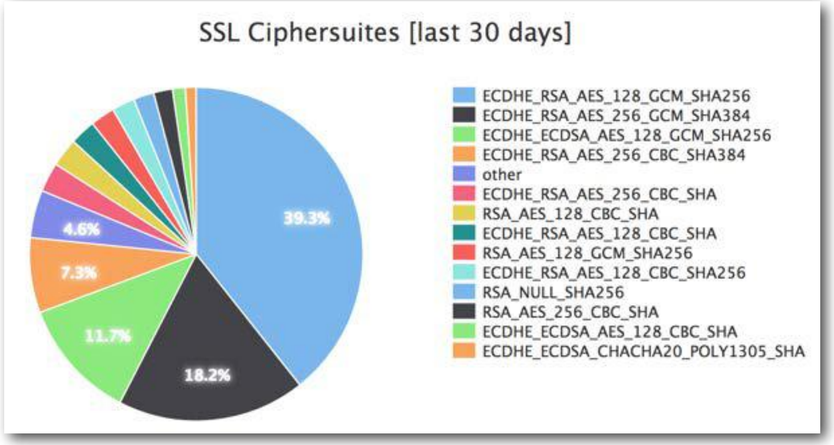


https://avinetworks.com/glossary/elliptic-curve-cryptography/

RSA versus ECC

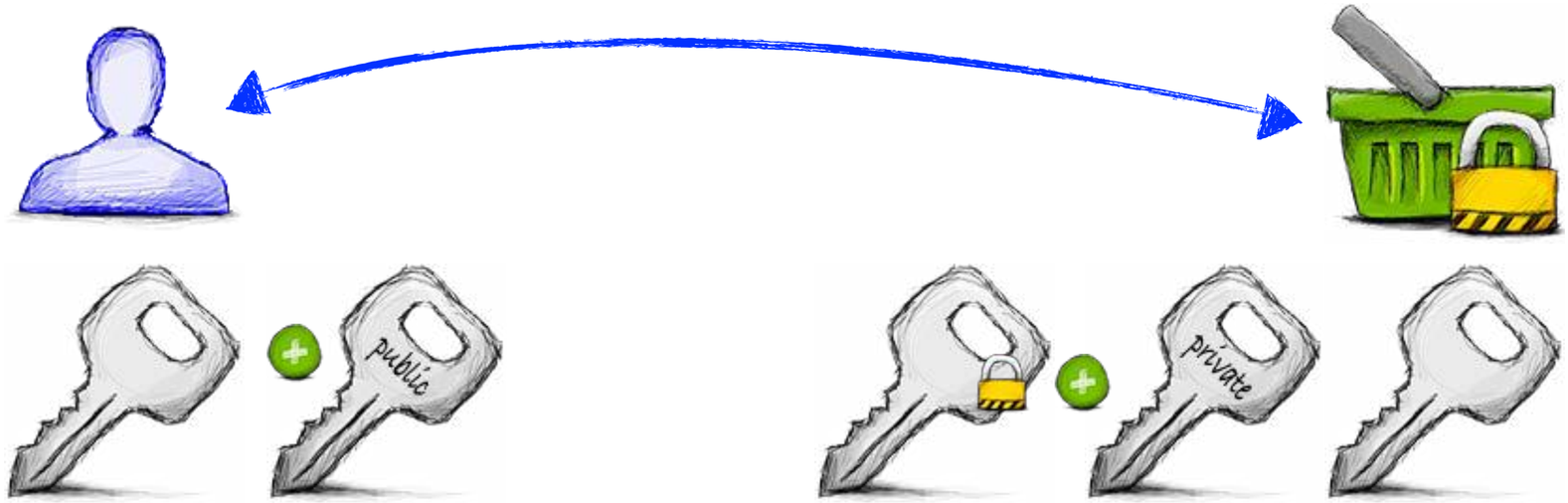
- Smaller key sizes
  - less CPU cycles needed
  - less memory needed
  - smaller PDU's (less traffic)
- Good for mobile devices
- Most TLS connections now use a cipher with ECDHE key generation
- Of those connections, most still use a certificate with a RSA public key

Symmetric	ECC	DH/DSA/RSA
80	163	1024
112	233	2048
128	283	3072
192	409	7680
256	571	15360

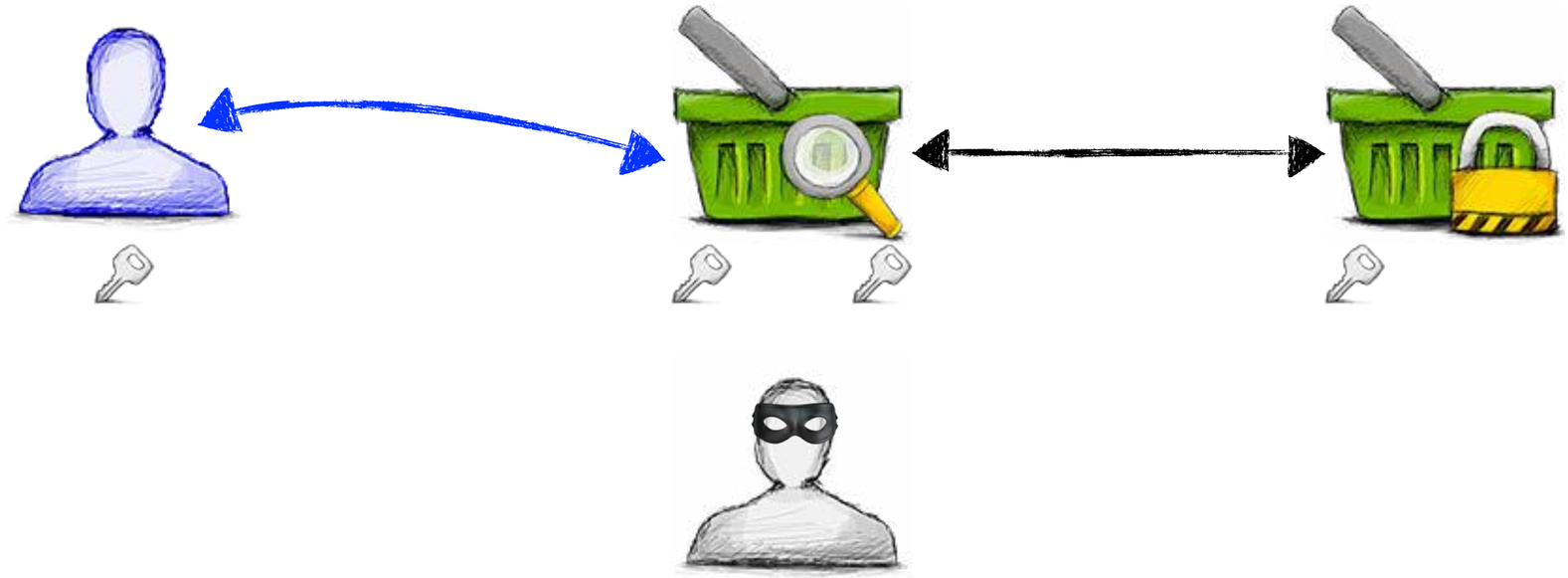


https://notary.icsi.berkeley.edu/ (from 2017)

Secure(?) Key Exchange



Man-in-the-Middle



How can we solve this?

Certificates

"In cryptography, a **public key certificate** (or identity certificate) is an **electronic document** which utilizes a **digital signature** to bind together **a public key** with an **identity**."

http://en.wikipedia.org/wiki/Digital_certificate

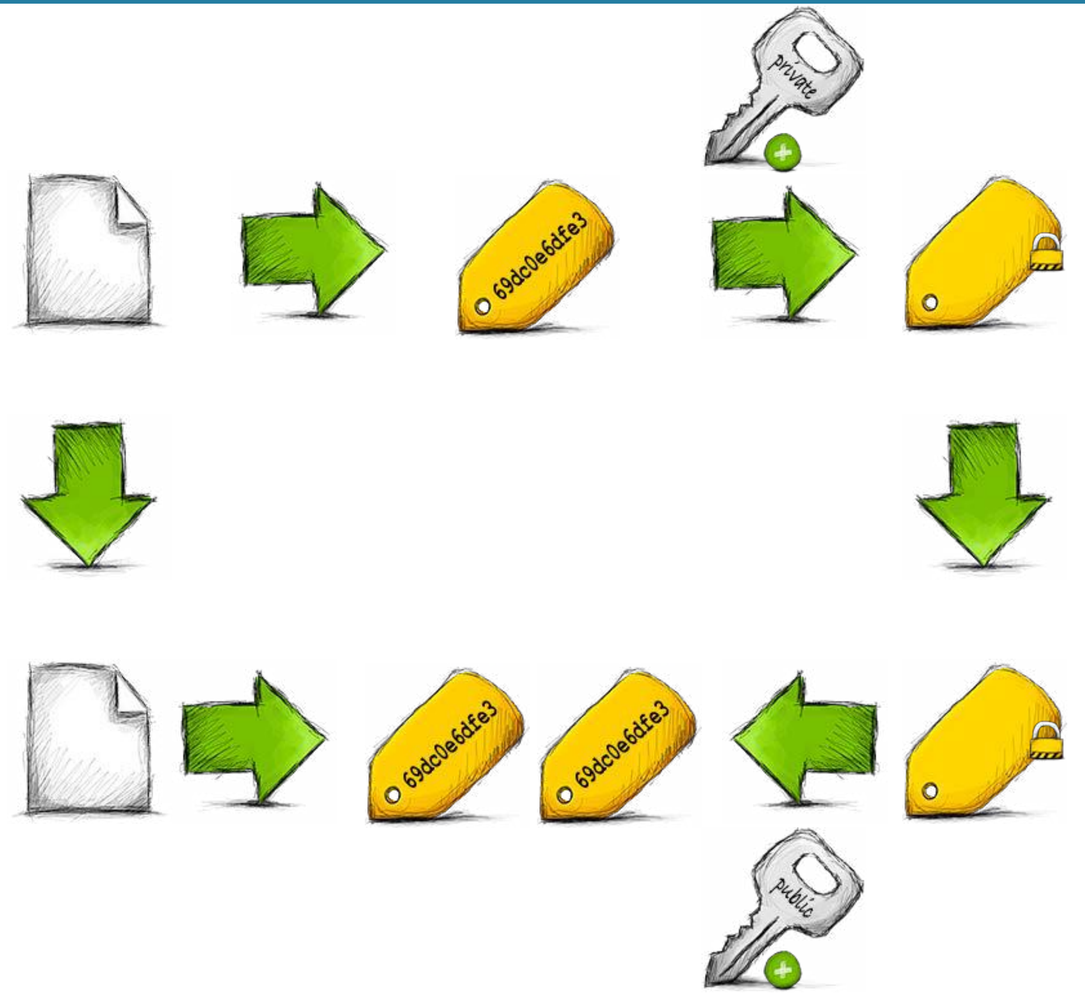
*But **what is signing**??? And **who** is doing it?*

Message Digest

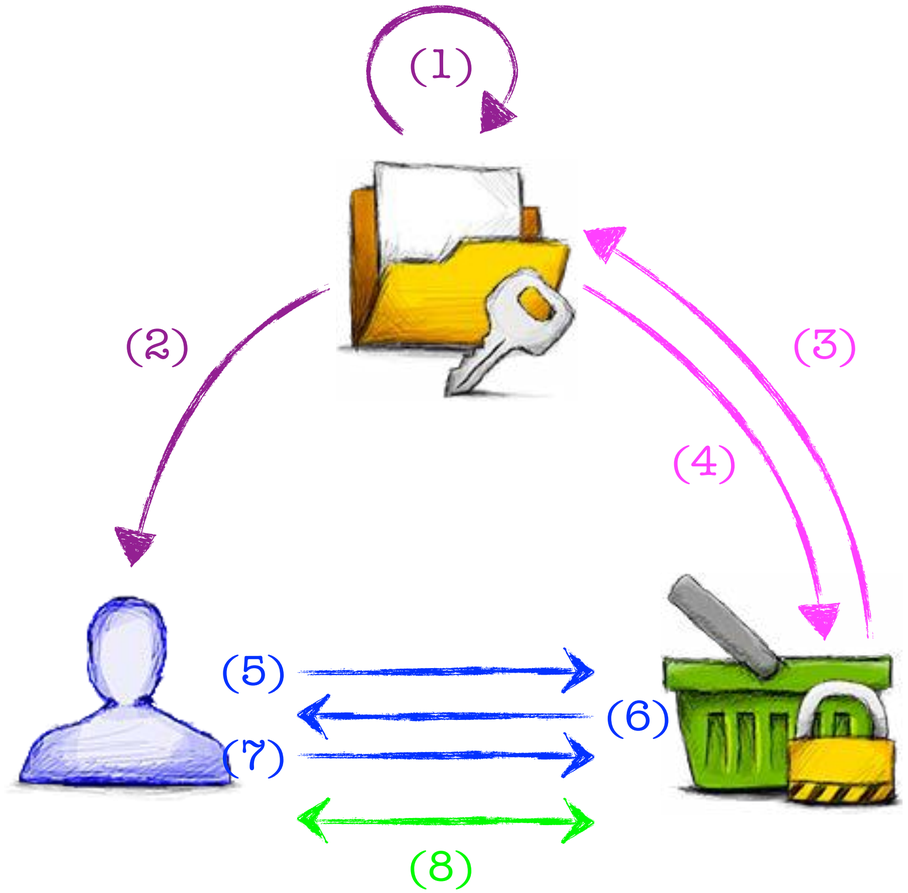
- Irreversible
  - original text not reproducible from the digest
- Collision-resistance
  - "Not possible" to create a message M'
    so that it has the same digest as message M
- ~~MD5, SHA-1~~, SHA-2, SHA-3

Signing

- Create digest of message
- Encrypt digest with private key
- Authenticity and sender of message can be checked with public key



Certificate Authority



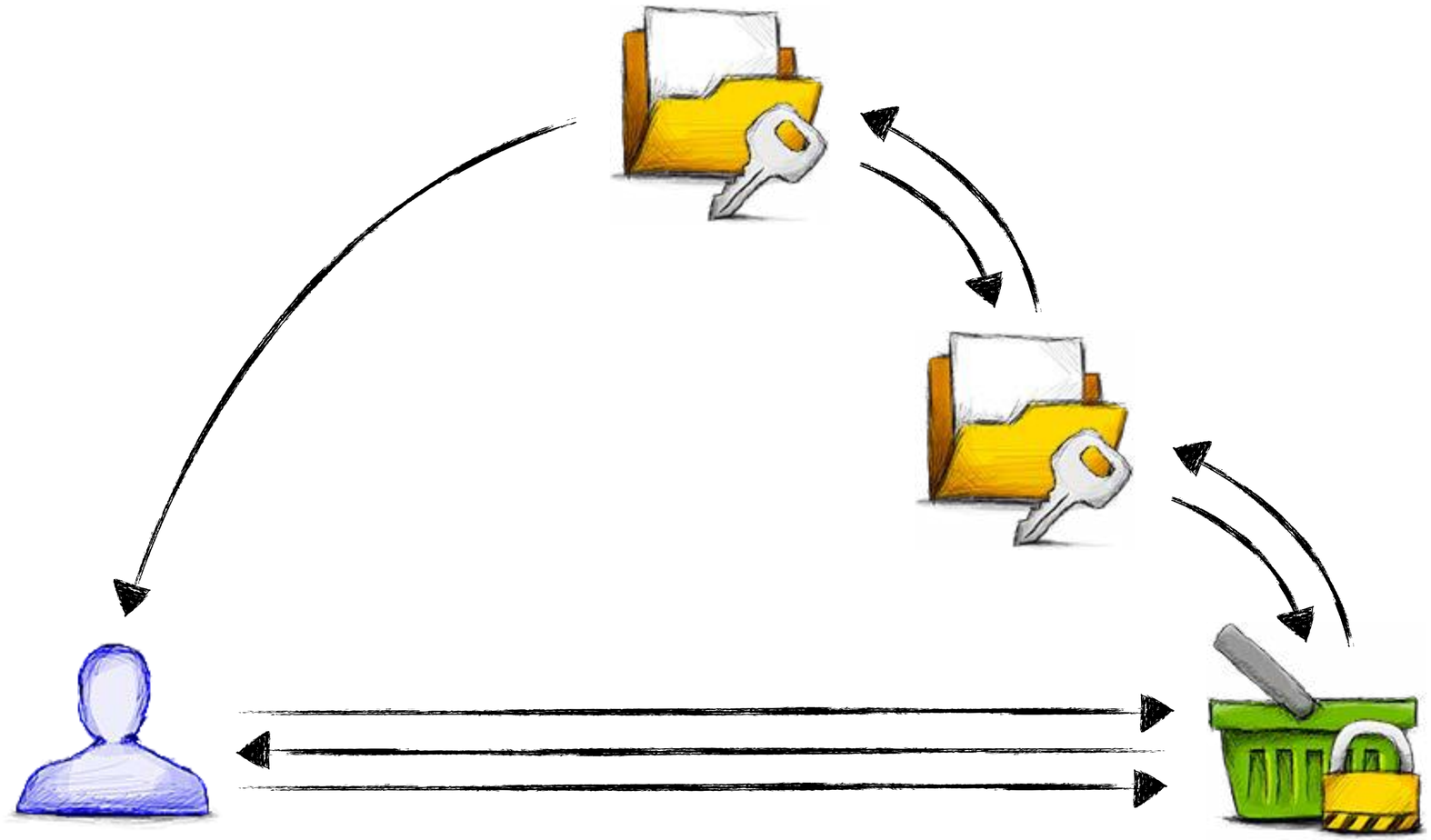
- Certificate Authority (CA) creates Identity & Keypair
- CA signs Identity & Public Key with Private Key to create its root-certificate (1)
- CA sends root-certificate to Clients **(in a secure way!?!)** (2)

- Server creates Identity & Keypair
- Server sends Identity & Public Key to Authority (3)
- **!!! CA Checks Identity !!!**
- CA Signs Identity & Public Key with Private Key
- CA sends back Certificate (4)

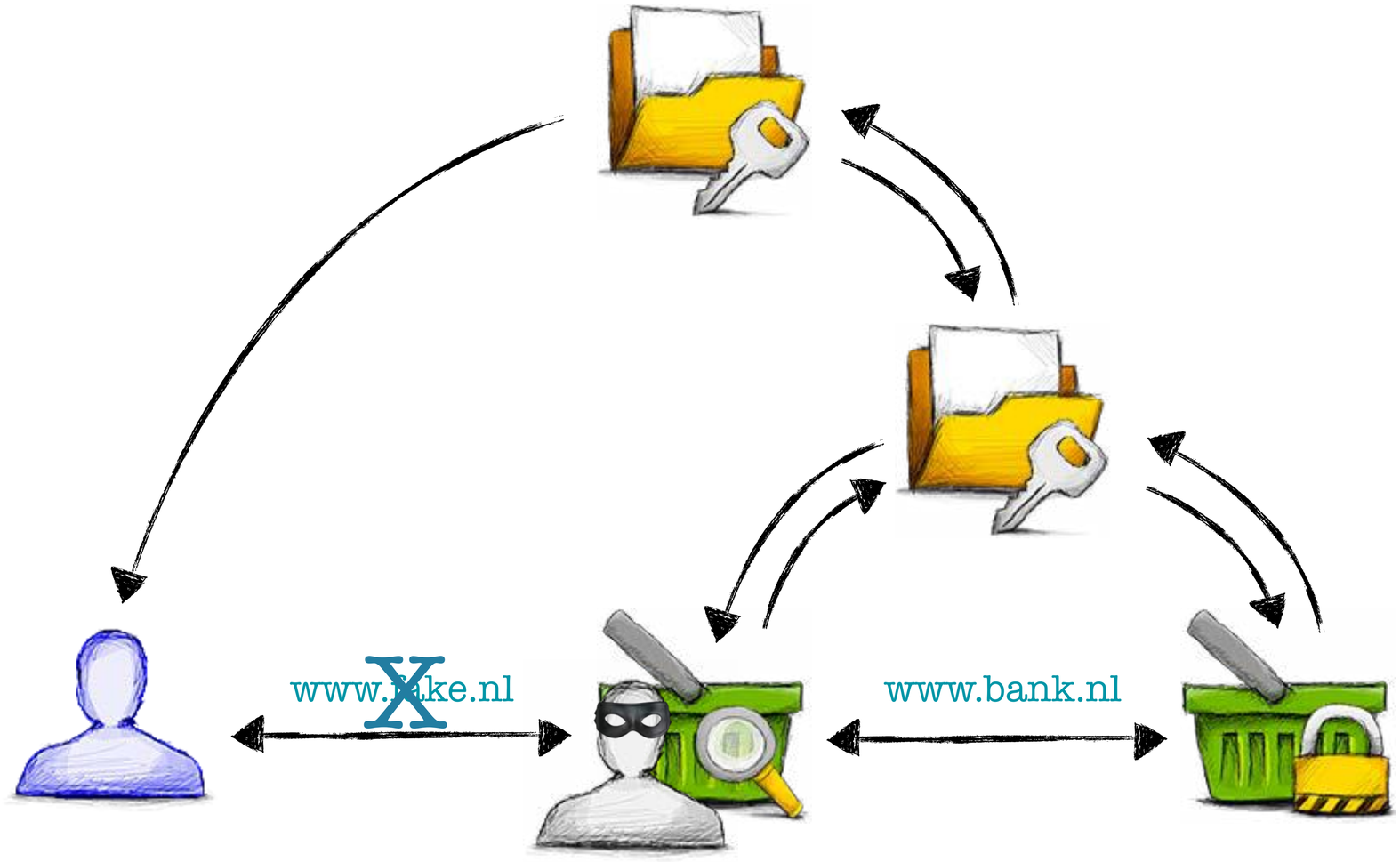
- Client connect to the Server (5)
- Server sends Certificate to Client (6)
- Client verifies Certificate with Public Key from the CA
- Client creates session keys
- Client encrypts session keys with the server Public Key
- Client sends encrypted keys to the server (7)
- Server decrypts session keys with its Private Key

- Client and server use session keys to exchange data

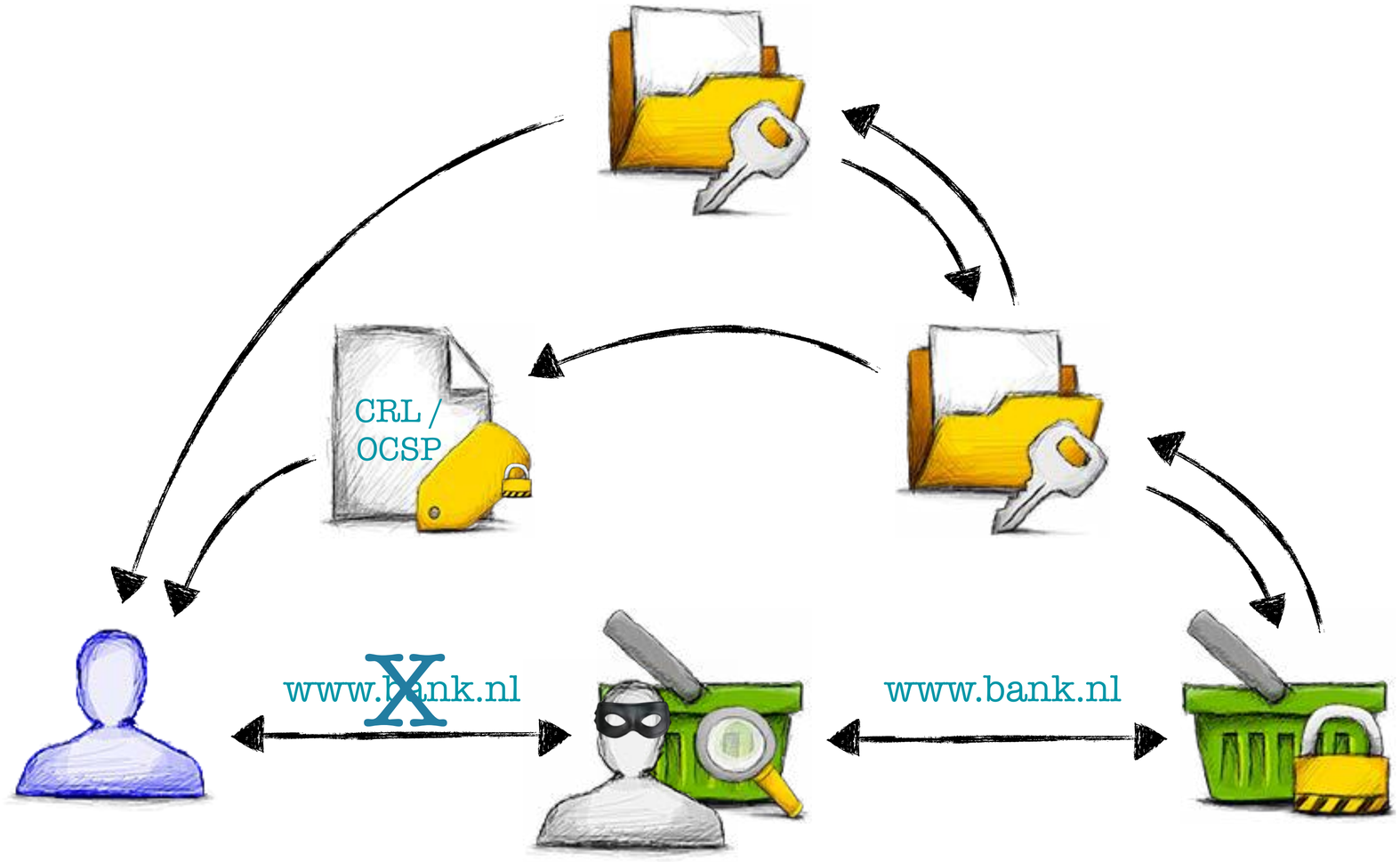
Intermediate CA's



Man-in-the-Middle II



Compromised Private Key



Certificate check on the client

- Check if the **ID** in the certificate matches what was requested

- Check if certificate chain is correct and ends in a **trusted CA**

- Check if the certificate is **valid**

- Check if the certificate has not been **revoked**

Certificate check on the client

- Check if the **ID** in the certificate matches what was requested

https://wrong.host.badssl.com/

- Check if certificate chain is correct and ends in a **trusted CA**

https://untrusted-root.badssl.com/

- Check if the certificate is **valid**

https://expired.badssl.com/

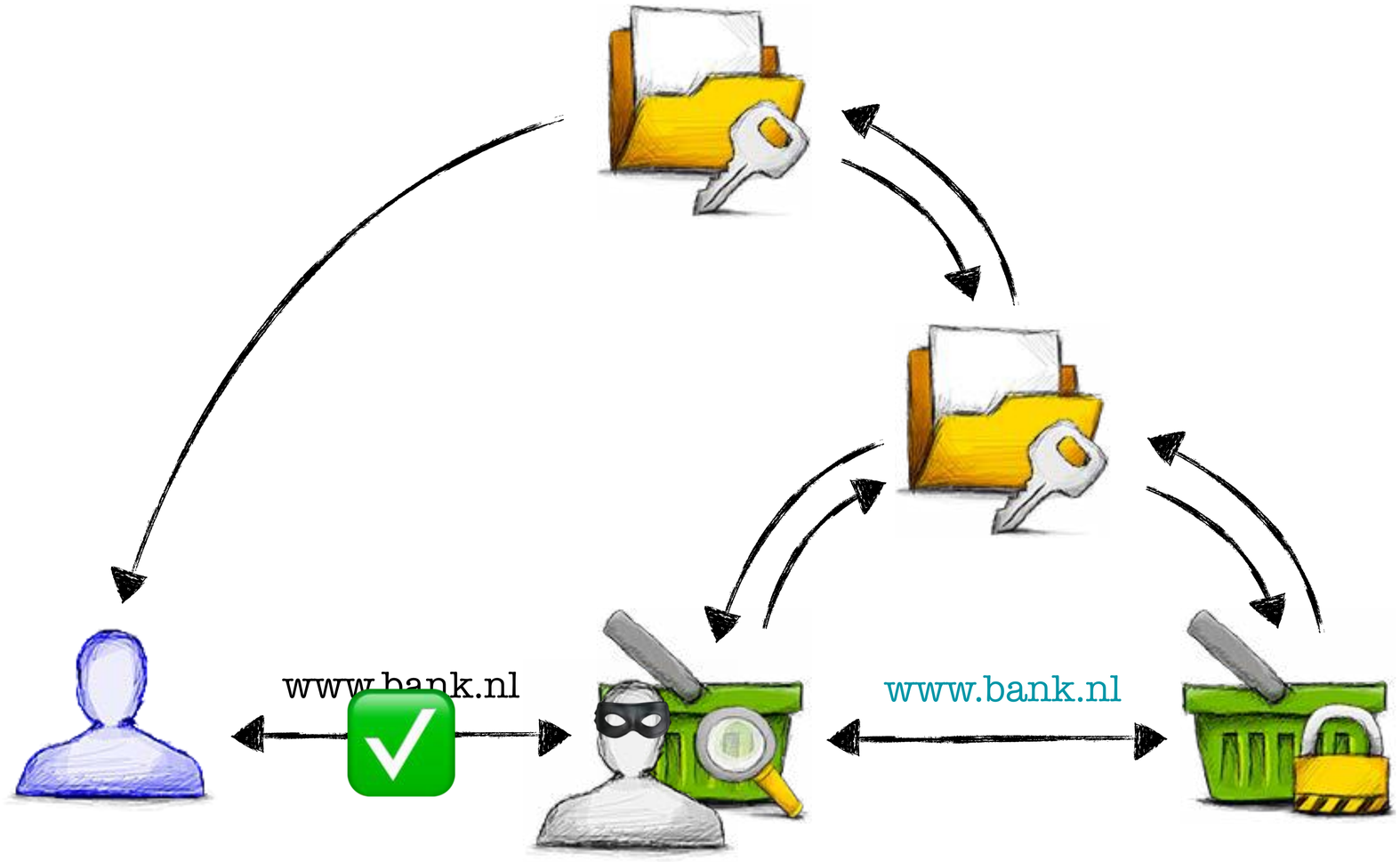
- Check if the certificate has not been **revoked**

https://revoked.badssl.com/

Certificate Revocation

- Certificate Revocation List (CRL)
  - Signed by CA
  - Can grow large, periodically retrieved
  - “deprecated”
- Online Certificate Status Protocol (OCSP)
  - Checked per connection
  - large load on CA
  - exposes user behaviour to CA
- OCSP Stapling
  - The server provides a time stamped, signed OCSP response
  - Must-staple extension

Compromised CA



Diginotar 2011

- 6-19 Jun: Hacker gained access to systems
- 10+19 Jul: Hacker created certificates
- 27 Jul: *.google.com certificate used for M-i-t-M attack in Iran
- 4 Aug: M-i-t-M attack now on large scale
- 27 Aug: Discovery of the falsified certificate
- 29 Aug: Revocation of *.google.com certificate
- 30 Aug: First removal of DigiNotar CA

Trust???

- Security measures not OK
- No virus scanners used on critical systems
- No separation of critical systems
- Weak passwords
- Unpatched software on public systems

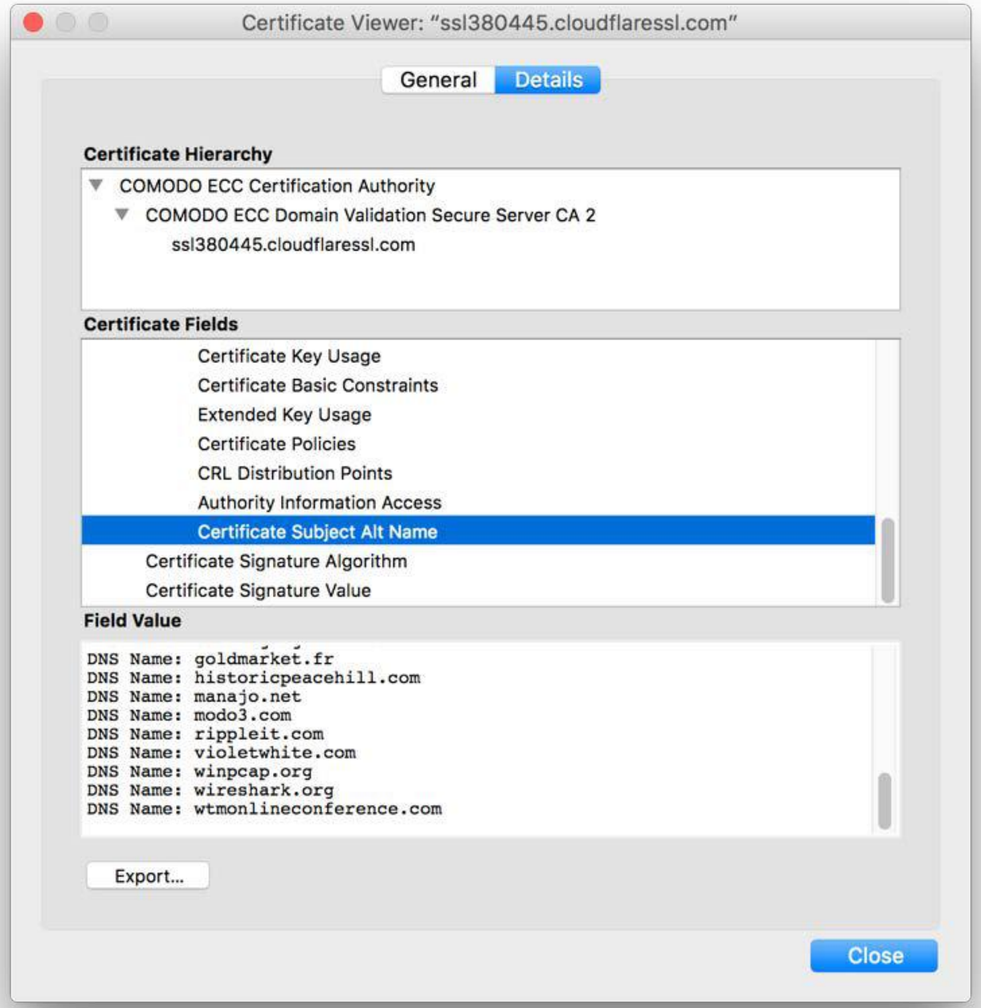
OSCP queries for fake Google Certificate

http://www.youtube.com/watch?v=wZsWoSxxwVY

- **No (open) communication about the hack**
- **Misleading information (PKIoverheid CA was indeed hacked contrary to earlier statements)**

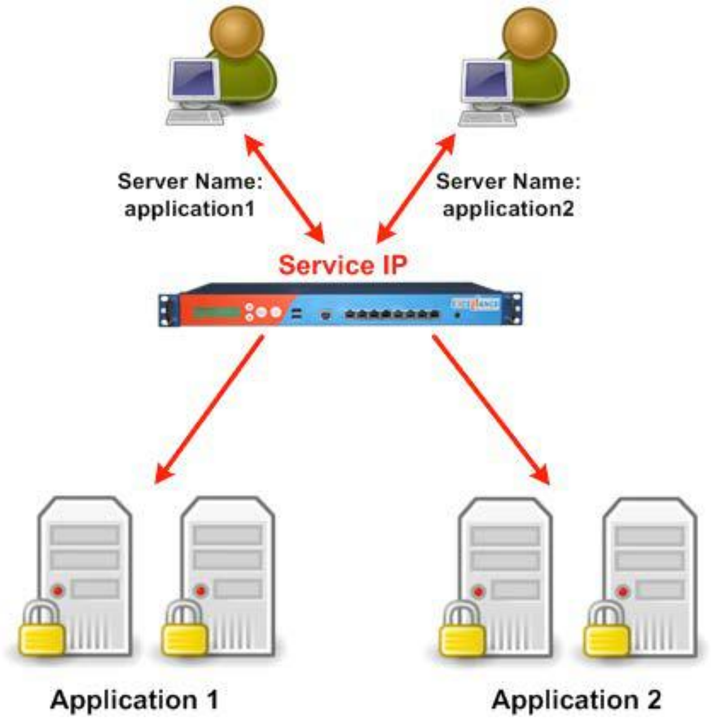
Certificate Identities (Common Name)

- Normal
  - www.example.com
- Wildcard
  - *.example.com
- SAN
  - www.example.com
  - mail.example.com
  - www.another.org



Server Name Indication (SNI)

- Multiple virtual hosts on one IP
- Host header (with FQDN) only available after SSLhandshake
- So, which certificate to use?
- Client provides servername in TLS extension
- Server (or ADC) can send specific certificate



Certificate types

- Self-singed
  - No identity verification (Of course you can trust me!!!)
- Domain Validated
  - Just ownership of the domain (automatically) checked
- Organization Validated (High Assurance)
  - Also (manual) organization check
- Extended Validation
  - Only audited CAs may provide EV certificates
  - Establish the legal identity as well as the operational and physical presence of website owner.
  - Establish that the applicant is the domain name owner or has exclusive control
  - Confirm the identity and authority of the individuals acting for the website owner
  - No wildcard certificates

Free Certificates

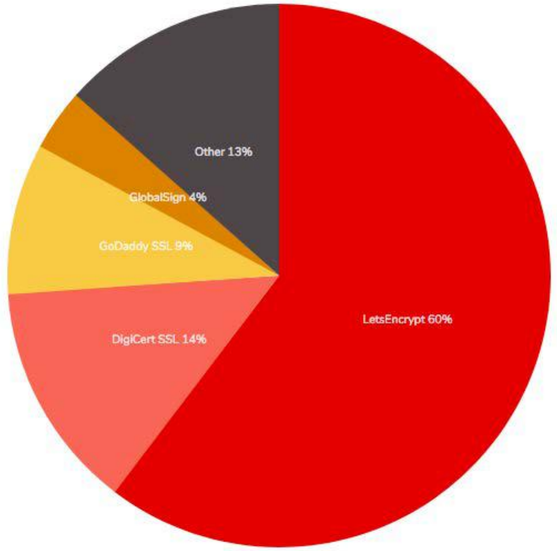
- Self-Signed certificates
- Create your own CA
- CAcert
  - 6 month expiration (24 with validation)
  - Community validation (face-to-face)
  - Root CA currently not trusted by many browsers
- Let's Encrypt
  - 90 day expiration
  - Automatic domain validation, no OV or EV certificates
  - No mail or code-signing certificates
- Some CA's offer free certificates for Open Source projects
- Free trials

FREE (AS IN) BEER.

Who do we trust the most?

Root Authority Usage Distribution in the Top 1 Million Sites

Statistics for websites using Root Authority providers

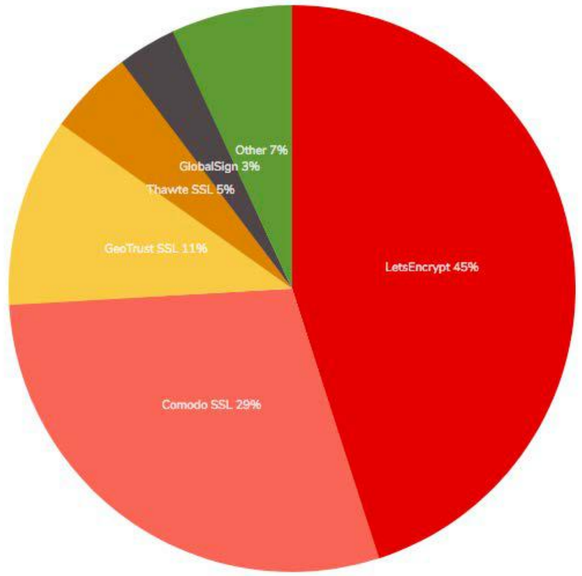


Top In Root Authority Usage Distribution in the Top 1 Million Sites

Technology	Websites	%
LetsEncrypt	401,152	40.12
DigiCert SSL	90,198	9.02
GoDaddy SSL	59,762	5.98
GlobalSign	24,434	2.44
GeoTrust SSL	18,144	1.81

Root Authority Usage Distribution in the Netherlands

Statistics for websites using Root Authority providers



Top In Root Authority Usage Distribution in the Netherlands

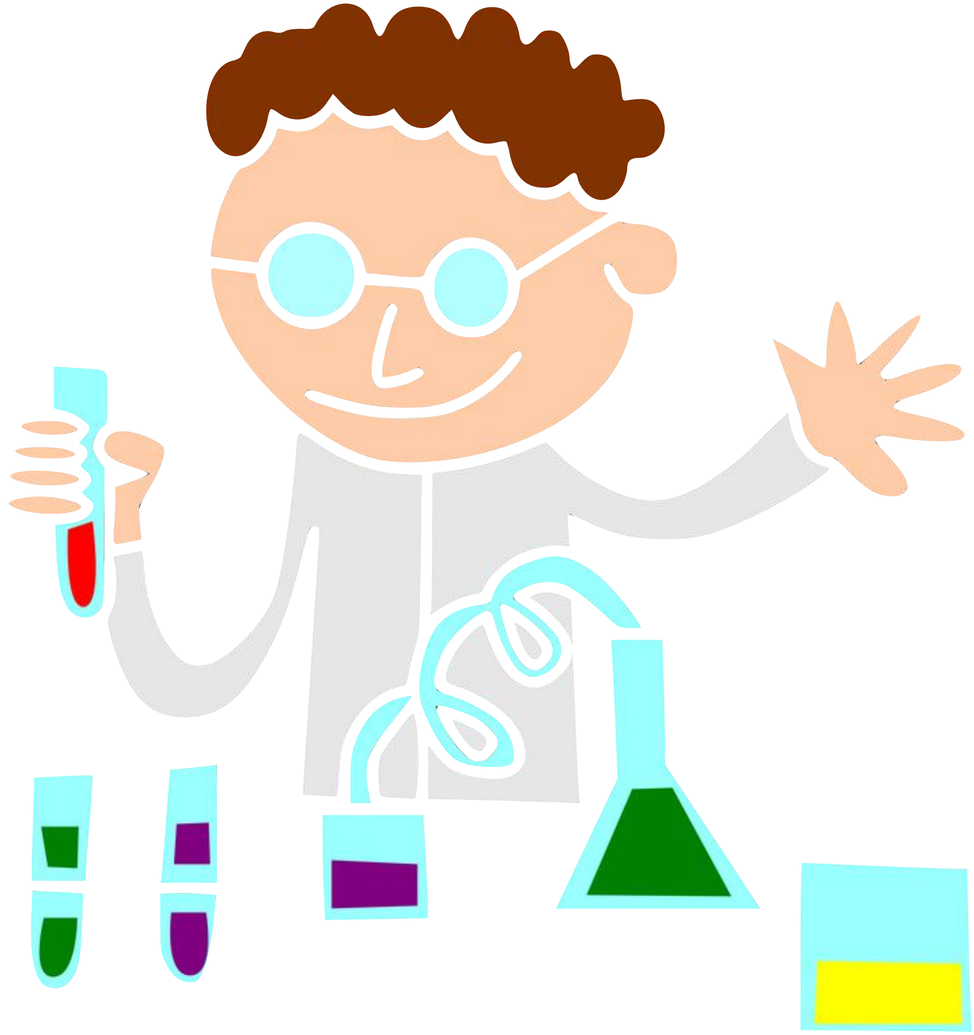
Technology	Websites	%
LetsEncrypt	119,253 *	45.08
Comodo SSL	76,847	29.05
GeoTrust SSL	28,533	10.79
Thawte SSL	12,926	4.89
GlobalSign	8,815	3.33

https://trends.builtwith.com/ssl/

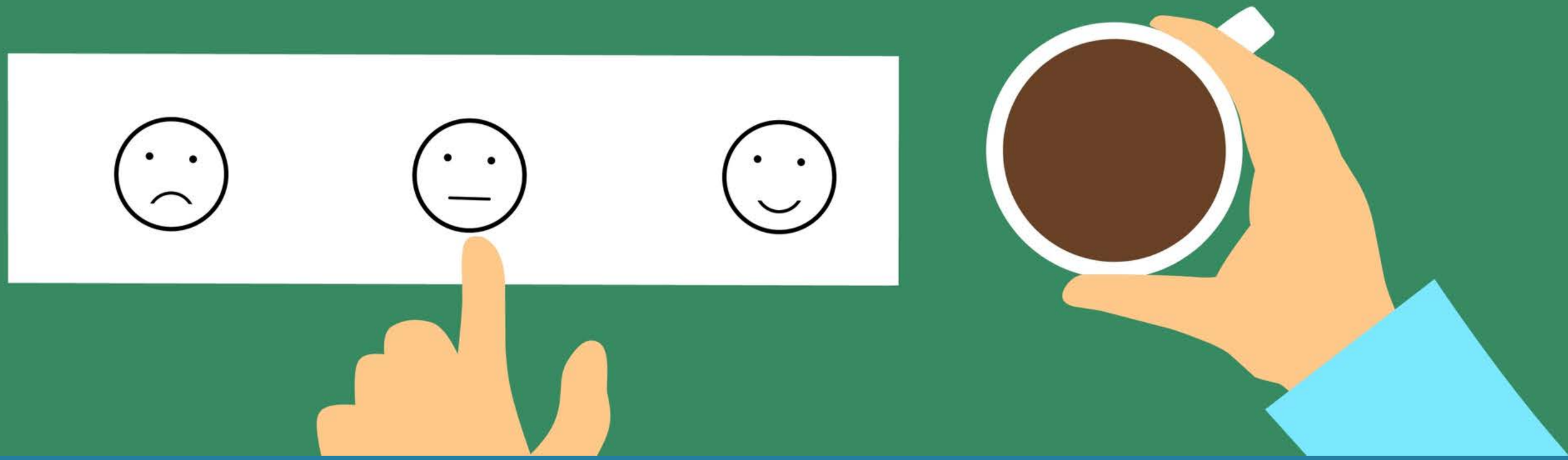

Trust!

Demo & Exercise

- Demo
  - https://nu.nl
  - https://tweakers.net
- Exercise
  - Visit 5 to 10 of your favorite websites
    - Are they redirecting to HTTPS?
    - What is the common name of the certificate?
    - Are there any alternative names?
    - In which period is the certificate valid?
    - What kind of public key is used? How many bits?
    - Which signig method was used?
    - Is it an DV, OV or EV certificate?
    - How many intermediate certificates? Who is the Root CA?
  - Write the details down for the 5th site you visit

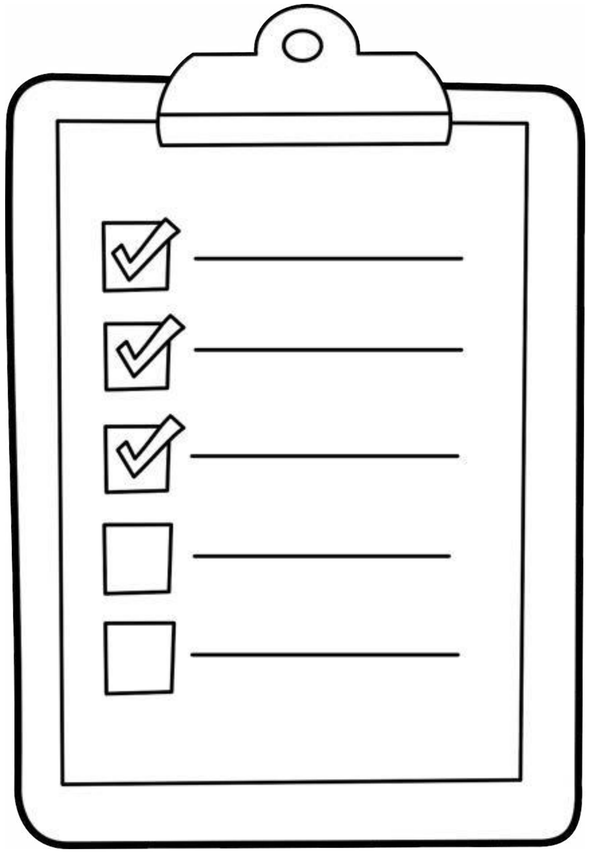


Time for a poll



Agenda

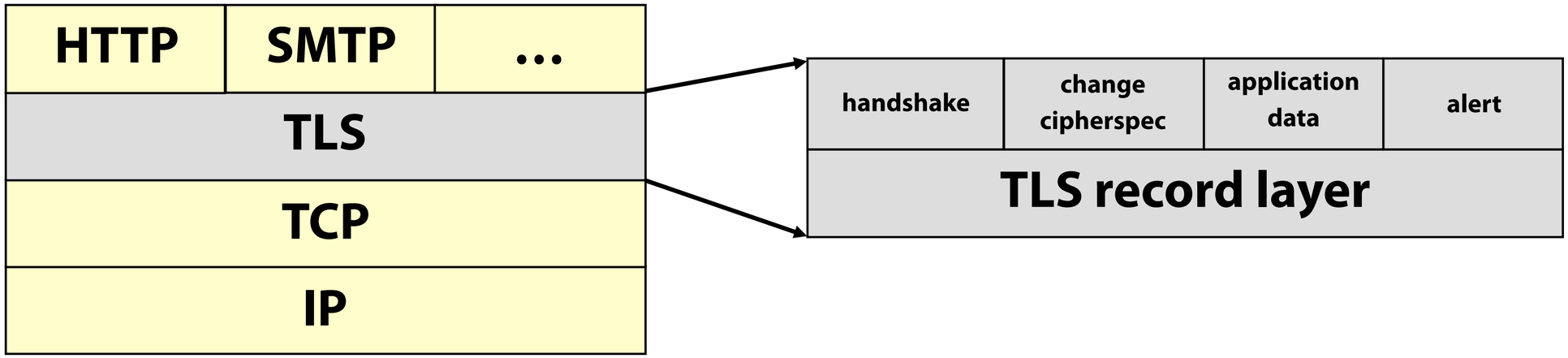
- TLS fundamentals
  - The need for TLS (and what happend to SSL?)
  - Cryptology 101 & PKI
  - **Troubleshoot the TLS handshake (SSLv3 - TLSv1.2)**
  - Troubleshoot the TLS handshake (TLSv1.3)
- Analysing Application Data
  - Without decryption
  - With decryption using the servers private key
  - With decryption using TLS session keys
- Creating configuration profiles for TLS analysis
- SSL/TLS vulnerabilities & privacy concerns
- Summary and Q&A



https://www.needpix.com/photo/download/94588/checklist-action-check-list-verification-tick-approved-okay-check-mark

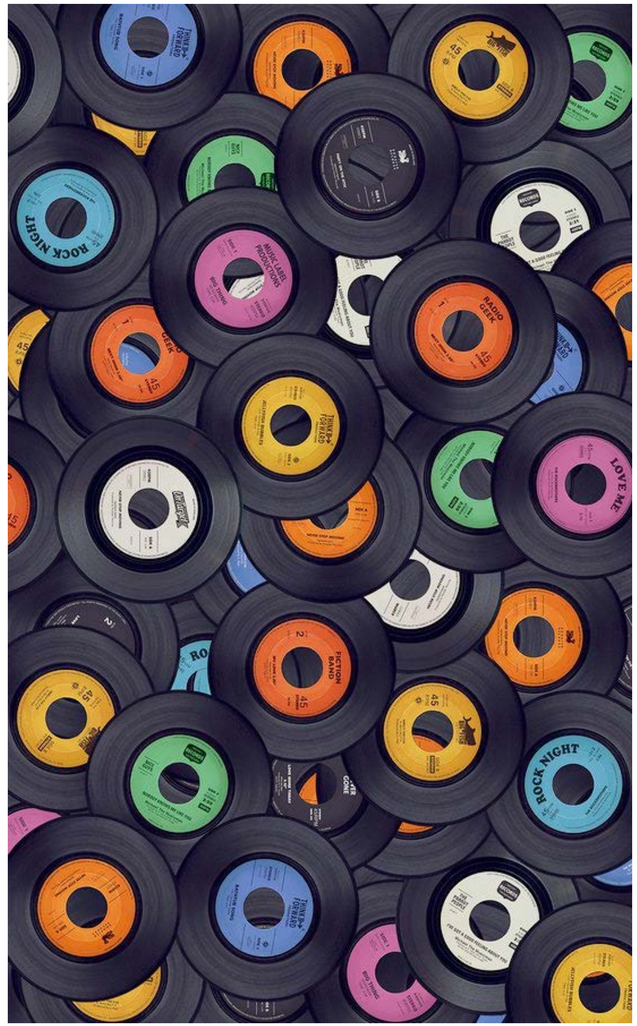
Place in TCP/IP stack

- Between transport and application layer
- Protocol independent



SSL Record Layer

- Provides fragmentation
  - (max cleartext size 2^14)
- Multiple SSL messages (of one content type) per SSL Record allowed
- SSL Record can be split over multiple TCP-segments (2^14 > MSS!)
- One TCP-segment can contain multiple SSL Records (or fragments)



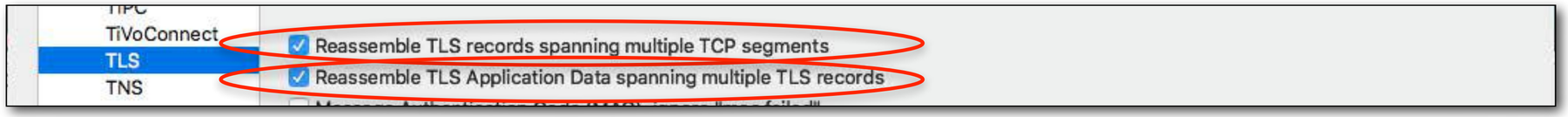
SSL Content Types

- Handshake Protocol (0x16)
  - responsible for authentication and session key setup

- ChangeCipherSpec Protocol (0x14)
  - Notify start of encryption

- Alert Protocol (0x15)
  - Reporting of warnings and fatal errors
- Application Protocol (0x17)
  - Actual encryption and transport of data

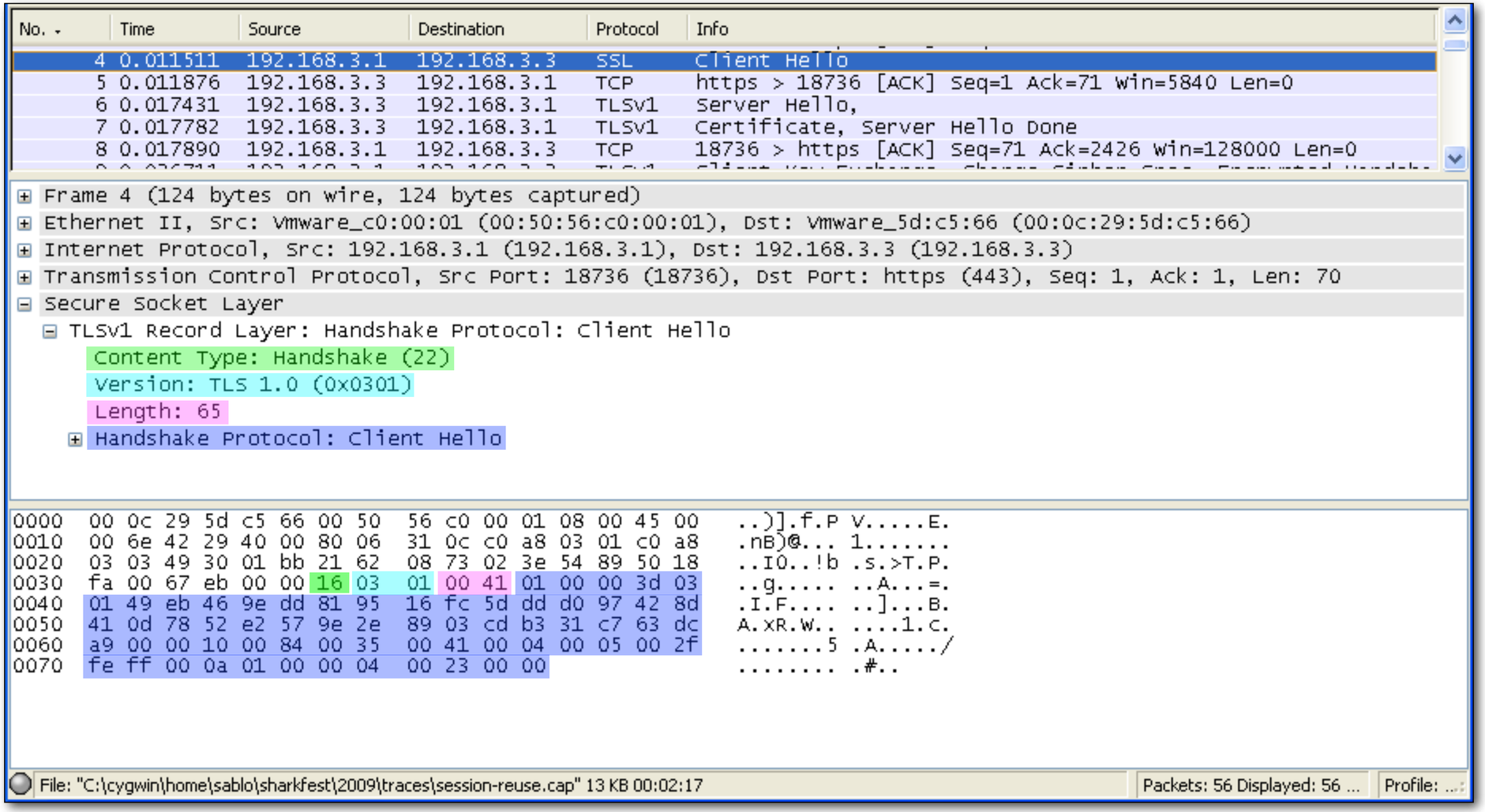
Use the right protocol preferences



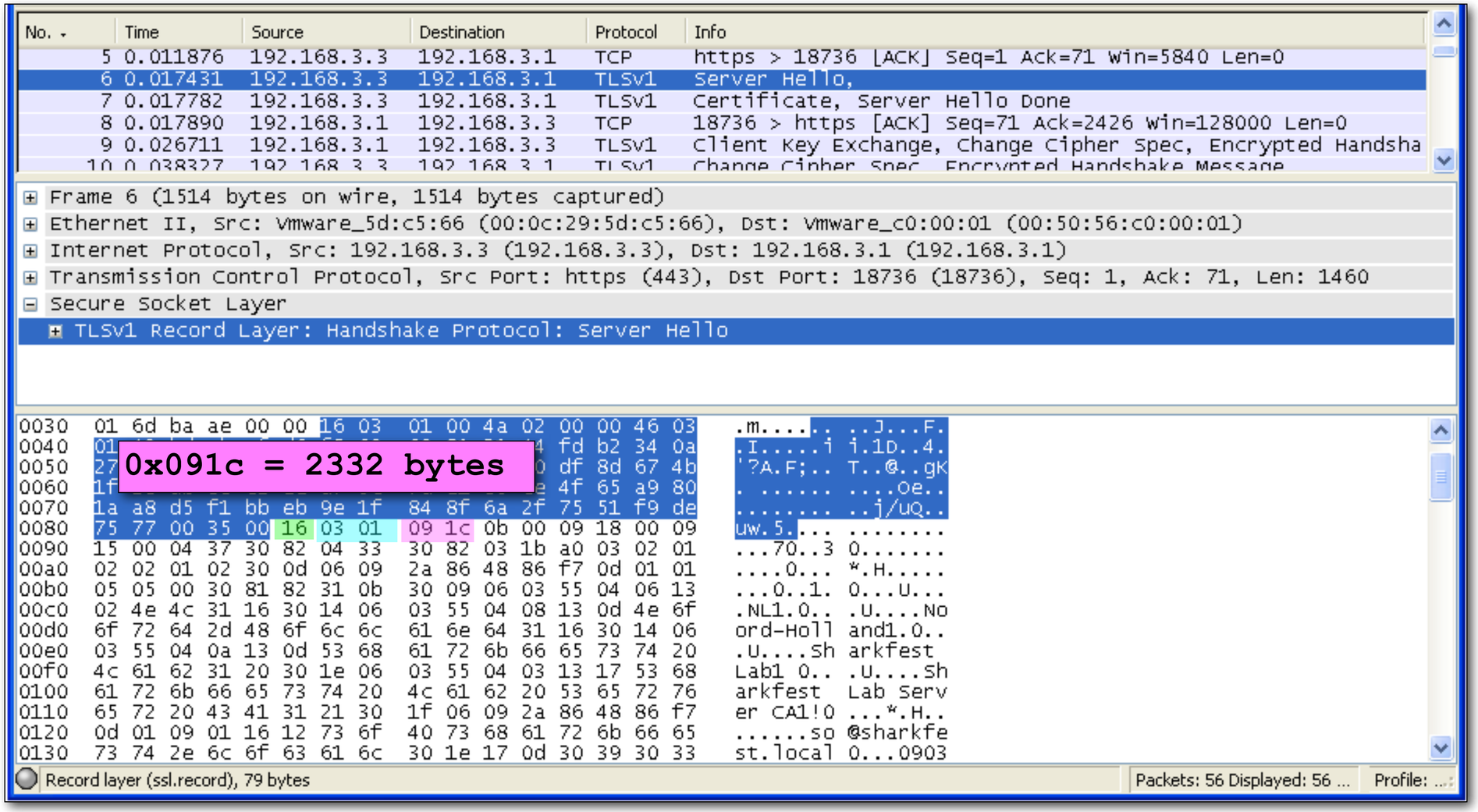

```
sake@MacSake:~$ tshark -G currentprefs | egrep "^#?(ip|tcp|tls).(defrag|deseg|check)"
#ip.defragment: TRUE
#ip.check_checksum: FALSE
#tcp.check_checksum: FALSE
#tcp.desegment_tcp_streams: TRUE
#tls.desegment_ssl_records: TRUE
#tls.desegment_ssl_application_data: TRUE
sake@MacSake:~$
```

Luckily those are all default settings

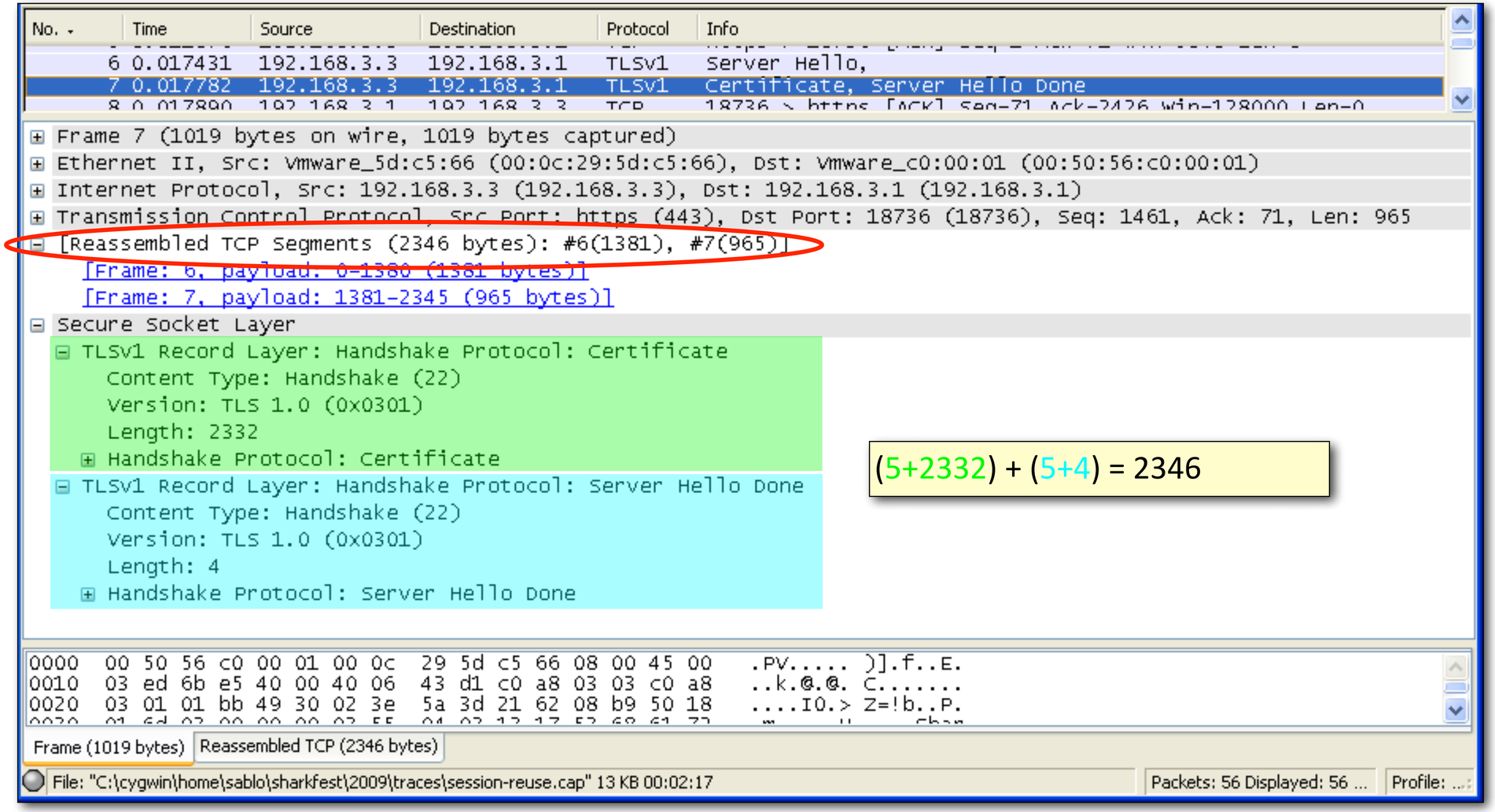
Analyzing the TLS record layer (1)



Analyzing the SSL record layer (2)



Analyzing the SSL record layer (3)



(5+2332) + (5+4) = 2346

Troubleshooting case

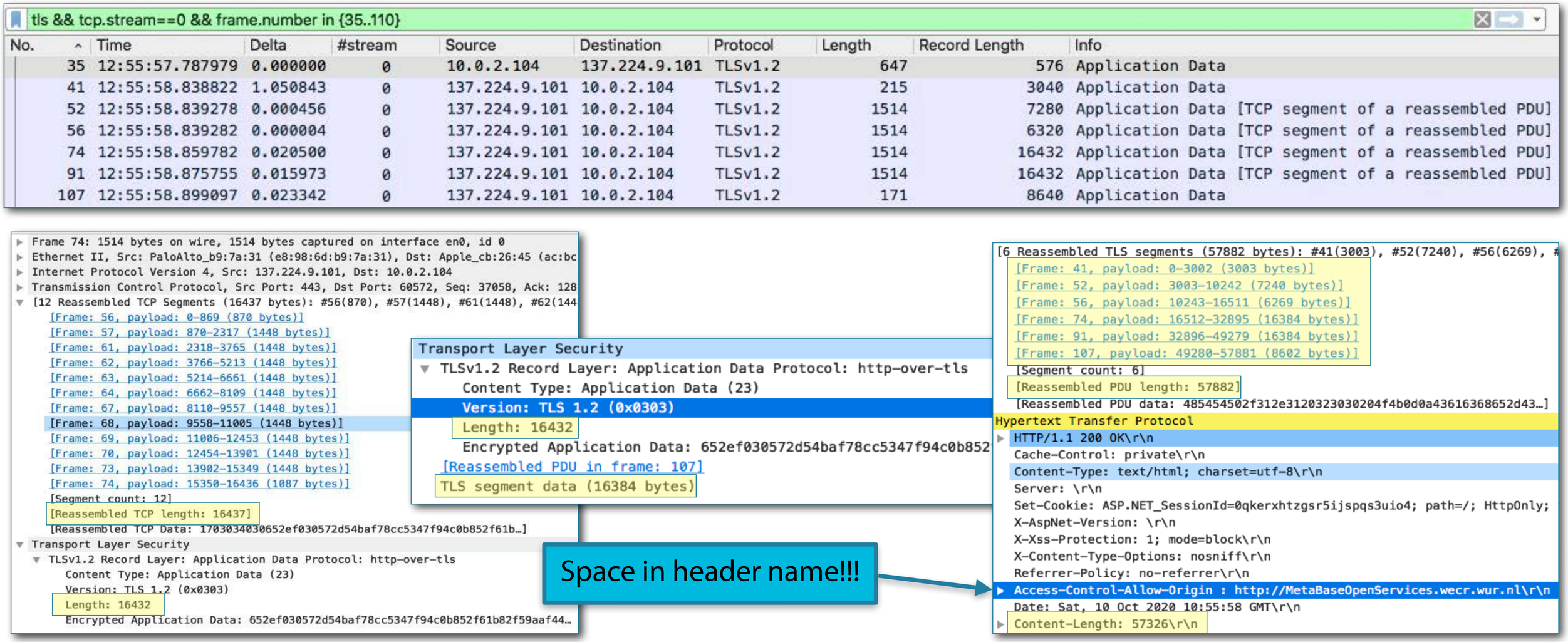
- Upgrade F5 loadbalancer
- Site stops working
- F5 does not see response from backend server
- Trace in the backend:

ion	Protocol	Length	Record Length	Info
16.118	TLSv1.2	552	448	Application Data
64.37	TCP	99		443 → 25298 [ACK] Seq=4074 Ack=773 Win=64133 Len=0
64.37	TCP	1507		443 → 25298 [ACK] Seq=4074 Ack=773 Win=64133 Len=1408 [TCP
64.37	TCP	1507		443 → 25298 [ACK] Seq=5482 Ack=773 Win=64133 Len=1408 [TCP
64.37	TCP	1507		443 → 25298 [ACK] Seq=6890 Ack=773 Win=64133 Len=1408 [TCP
64.37	TCP	1507		443 → 25298 [ACK] Seq=8298 Ack=773 Win=64133 Len=1408 [TCP
64.37	TCP	1507		443 → 25298 [ACK] Seq=9706 Ack=773 Win=64133 Len=1408 [TCP
16.118	TCP	99		25298 → 443 [ACK] Seq=773 Ack=11114 Win=15373 Len=0
64.37	TCP	1507		443 → 25298 [ACK] Seq=11114 Ack=773 Win=64133 Len=1408 [TCP
64.37	TCP	1507		443 → 25298 [ACK] Seq=12522 Ack=773 Win=64133 Len=1408 [TCP
64.37	TCP	1507		443 → 25298 [ACK] Seq=13930 Ack=773 Win=64133 Len=1408 [TCP
64.37	TCP	1507		443 → 25298 [ACK] Seq=15338 Ack=773 Win=64133 Len=1408 [TCP
64.37	TCP	1507		443 → 25298 [ACK] Seq=16746 Ack=773 Win=64133 Len=1408 [TCP
64.37	TCP	1507		443 → 25298 [ACK] Seq=18154 Ack=773 Win=64133 Len=1408 [TCP
64.37	TLSv1.2	1507	16464	Application Data [TCP segment of a reassembled PDU]
64.37	TCP	1507		443 → 25298 [ACK] Seq=20970 Ack=773 Win=64133 Len=1408 [TCP
64.37	TCP	1507		443 → 25298 [ACK] Seq=22378 Ack=773 Win=64133 Len=1408 [TCP
16.118	TCP	99		25298 → 443 [ACK] Seq=773 Ack=23786 Win=28045 Len=0
16.118	TCP	87		25298 → 443 [RST, ACK] Seq=773 Ack=23786 Win=0 Len=0

```
▶ Transmission Control Protocol, Src Port: 443, Dst Port: 25298, Seq: 19562, Ack: 773, Len
▶ [12 Reassembled TCP Segments (16469 bytes): #19(1408), #20(1408), #21(1408), #22(1408),
▼ Transport Layer Security
  ▼ TLSv1.2 Record Layer: Application Data Protocol: http-over-tls
      Content Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 16464
      Encrypted Application Data: 2ca1cbf1fa261fdb359ae7f6dbd2a0f6ac2149b489b346c8…
```

Too Large???

Clientside trace of working version

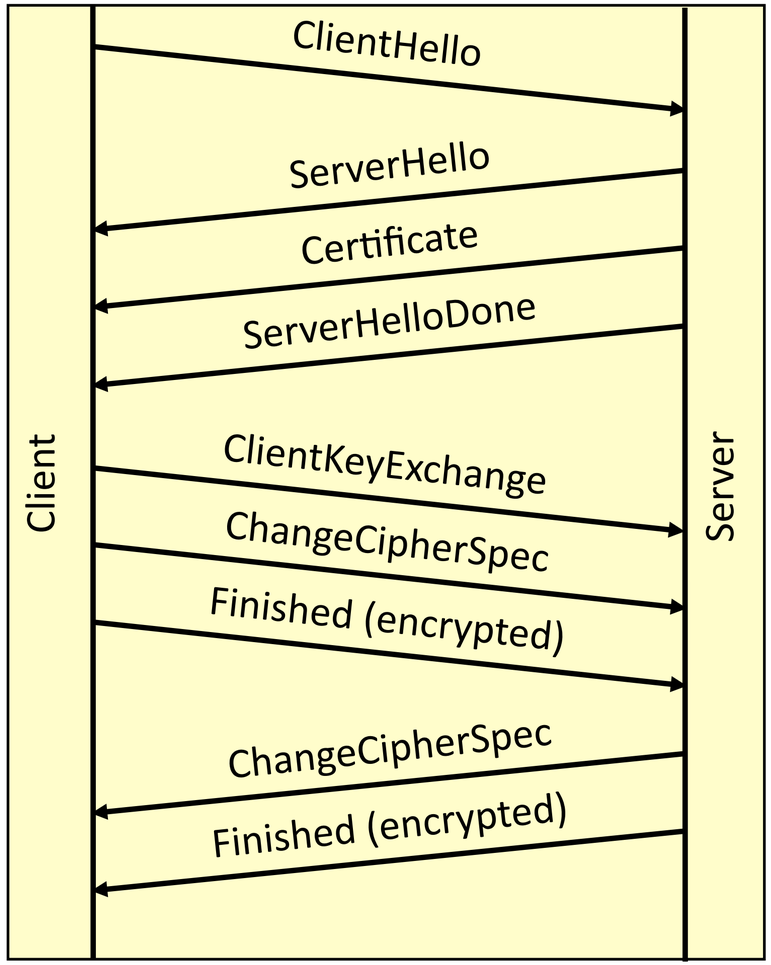


tls && tcp.stream==0 && frame.number in {35..110}

No.	Time	Delta	#stream	Source	Destination	Protocol	Length	Record Length	Info
35	12:55:57.787979	0.000000	0	10.0.2.104	137.224.9.101	TLSv1.2	647	576	Application Data
41	12:55:58.838822	1.050843	0	137.224.9.101	10.0.2.104	TLSv1.2	215	3040	Application Data
52	12:55:58.839278	0.000456	0	137.224.9.101	10.0.2.104	TLSv1.2	1514	7280	Application Data [TCP segment of a reassembled PDU]
56	12:55:58.839282	0.000004	0	137.224.9.101	10.0.2.104	TLSv1.2	1514	6320	Application Data [TCP segment of a reassembled PDU]
74	12:55:58.859782	0.020500	0	137.224.9.101	10.0.2.104	TLSv1.2	1514	16432	Application Data [TCP segment of a reassembled PDU]
91	12:55:58.875755	0.015973	0	137.224.9.101	10.0.2.104	TLSv1.2	1514	16432	Application Data [TCP segment of a reassembled PDU]
107	12:55:58.899097	0.023342	0	137.224.9.101	10.0.2.104	TLSv1.2	171	8640	Application Data

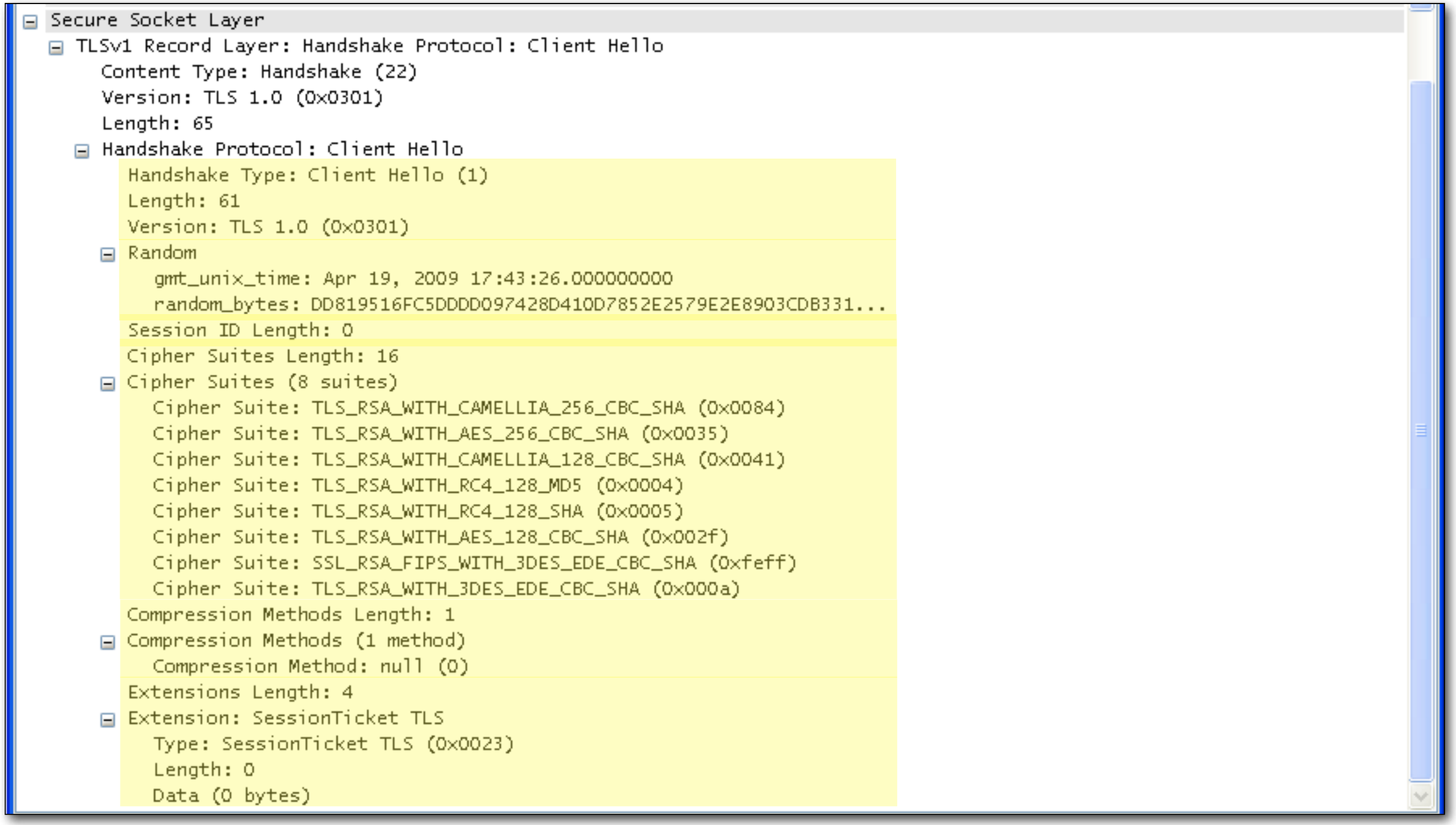
Space in header name!!!

Handshake with RSA key exchange

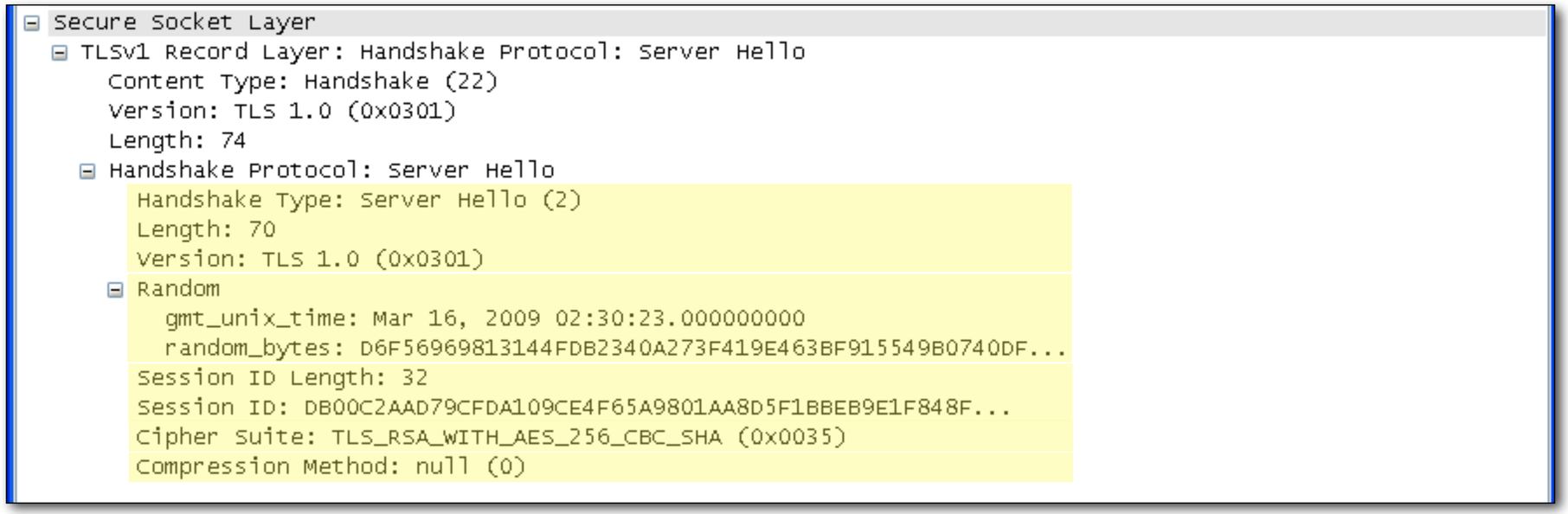


No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.3.1	192.168.3.3	TCP	18736 > https [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=1
2	0.000309	192.168.3.3	192.168.3.1	TCP	https > 18736 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 WS=
3	0.000357	192.168.3.1	192.168.3.3	TCP	18736 > https [ACK] Seq=1 Ack=1 Win=128000 Len=0
4	0.011511	192.168.3.1	192.168.3.3	TLSv1	Client Hello
5	0.011876	192.168.3.3	192.168.3.1	TCP	https > 18736 [ACK] Seq=1 Ack=71 Win=5840 Len=0
6	0.017431	192.168.3.3	192.168.3.1	TLSv1	Server Hello,
7	0.017782	192.168.3.3	192.168.3.1	TLSv1	Certificate, Server Hello Done
8	0.017890	192.168.3.1	192.168.3.3	TCP	18736 > https [ACK] Seq=71 Ack=2426 Win=128000 Len=0
9	0.026711	192.168.3.1	192.168.3.3	TLSv1	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Mes
10	0.038327	192.168.3.3	192.168.3.1	TLSv1	Change Cipher Spec, Encrypted Handshake Message

ClientHello (client)

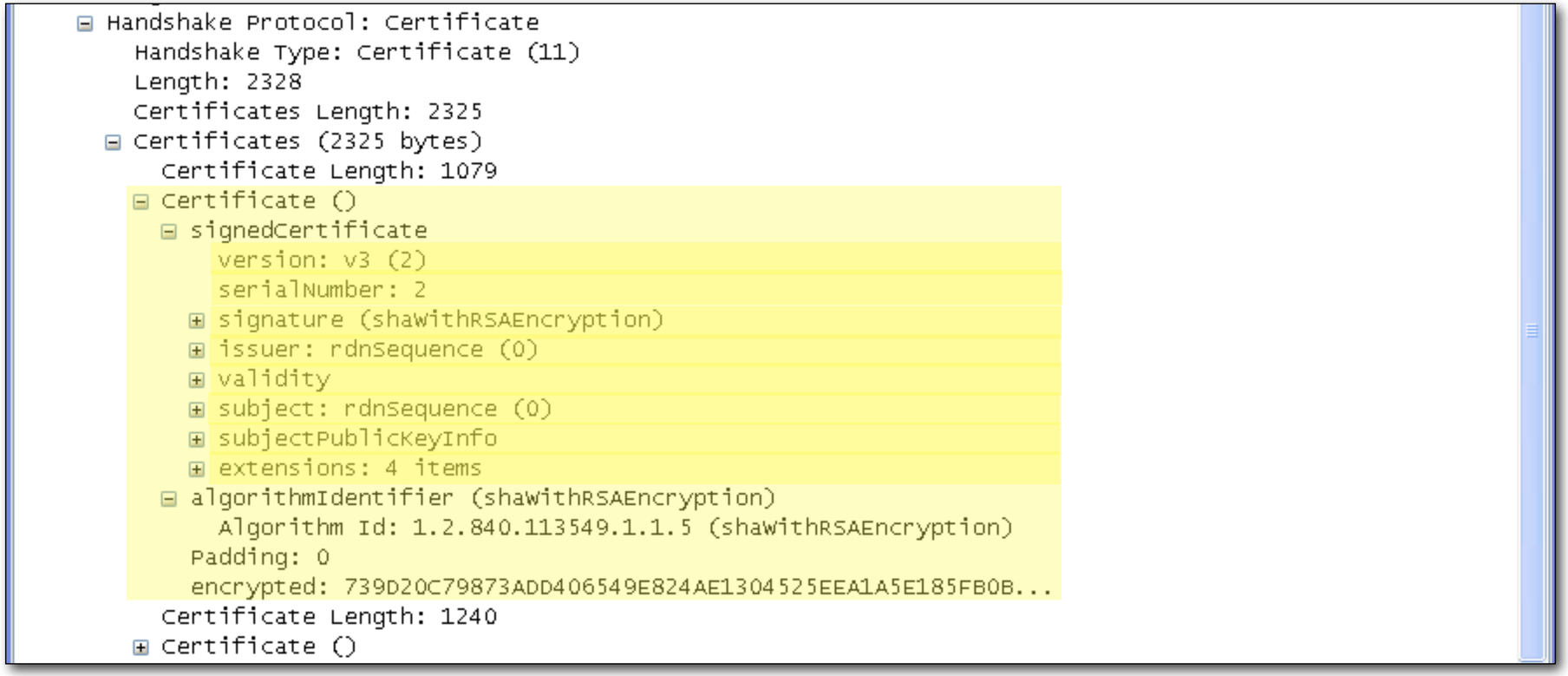


ServerHello (server)

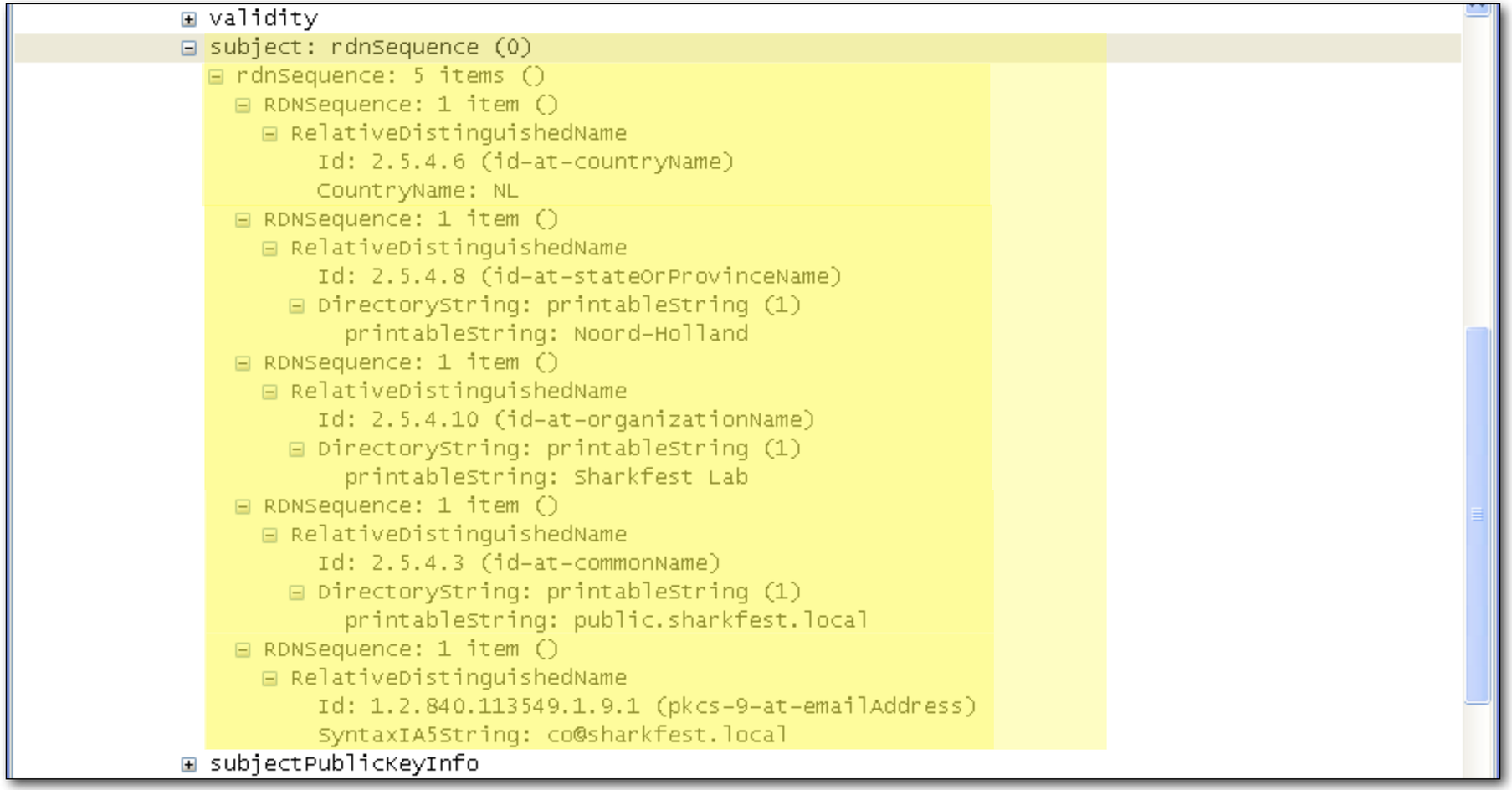


Certificate I (server)

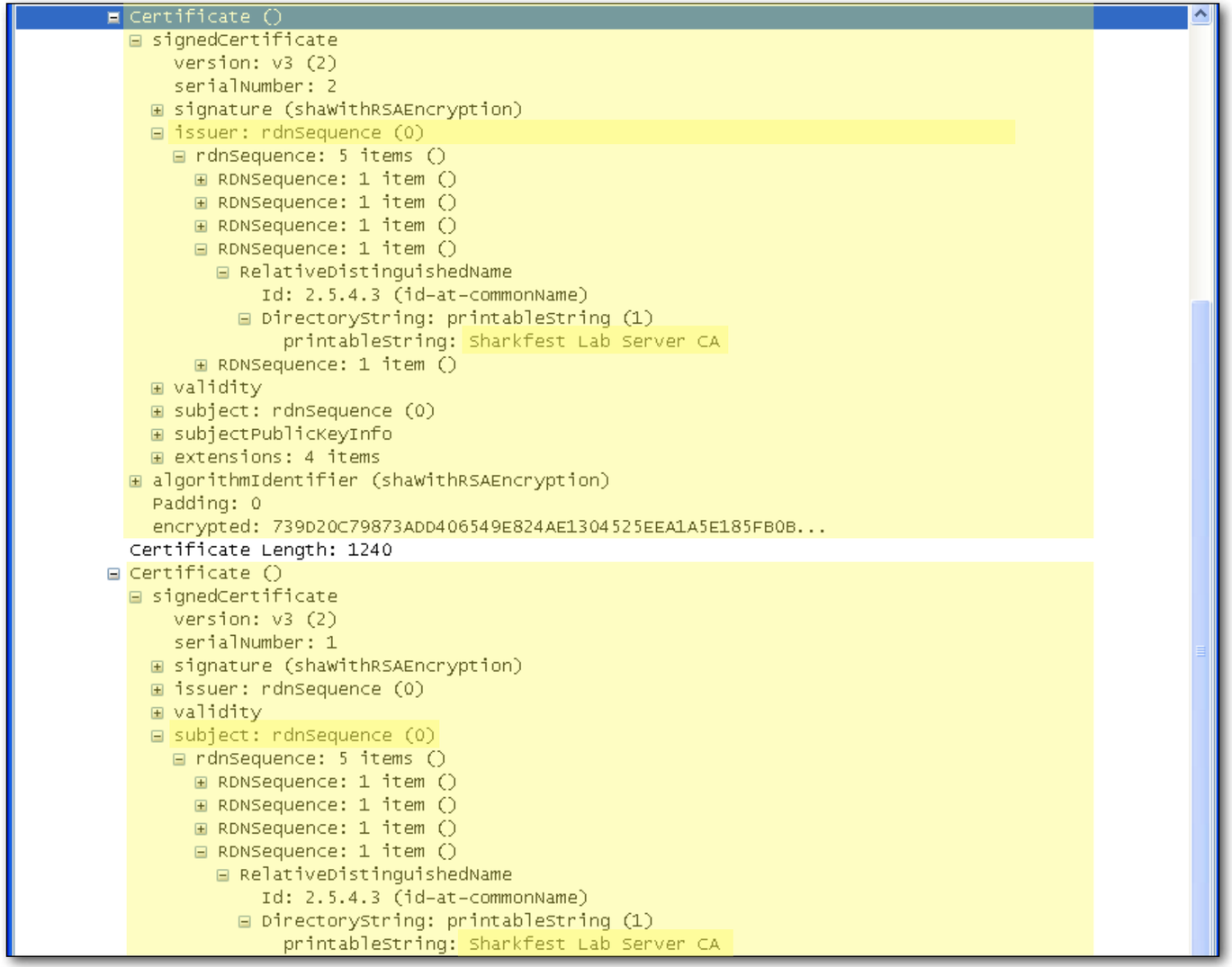
Certificate II (server)



Certificate III (server)



Certificate IV (server)

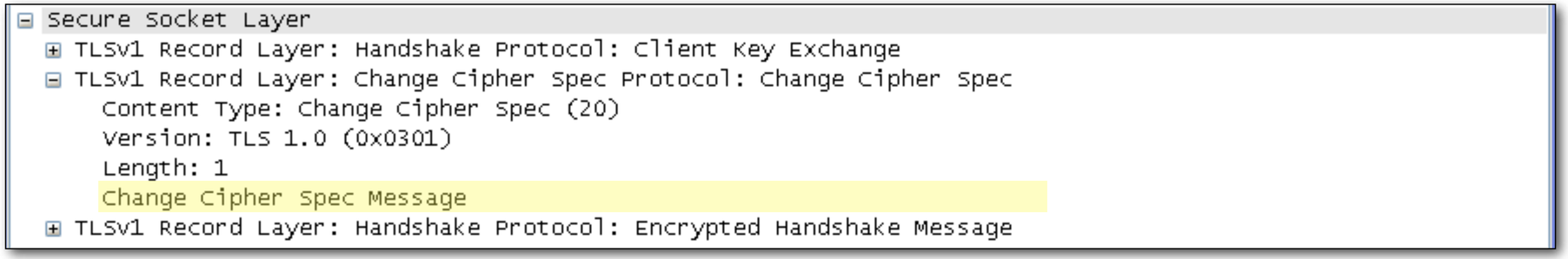


ServerHelloDone (server)

ClientKeyExchange (client)

```
Secure Socket Layer
  TLSv1 Record Layer: Handshake Protocol: Client Key Exchange
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 134
    Handshake Protocol: Client Key Exchange
        Handshake Type: Client Key Exchange (16)
        Length: 130
  TLSv1 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
  TLSv1 Record Layer: Handshake Protocol: Encrypted Handshake Message
```

ChangeCipherSpec (client)



Finished (client)

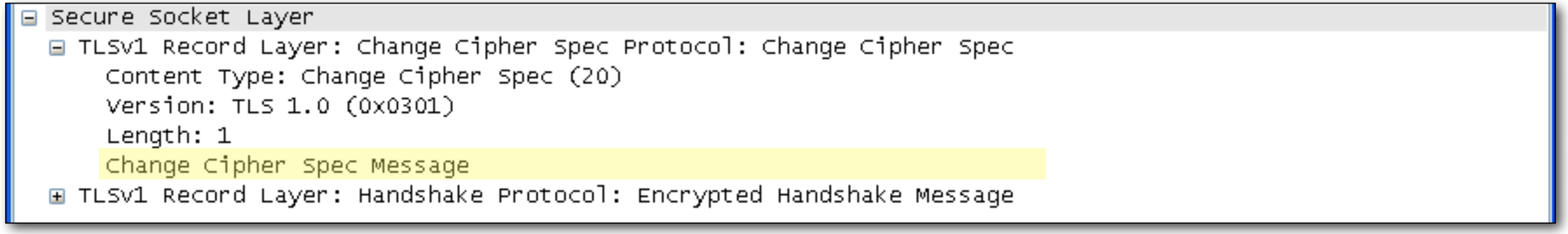
Without decryption:

```
Secure Socket Layer
  TLSv1 Record Layer: Handshake Protocol: Client Key Exchange
  TLSv1 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
  TLSv1 Record Layer: Handshake Protocol: Encrypted Handshake Message
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 48
      Handshake Protocol: Encrypted Handshake Message
```

With decryption:

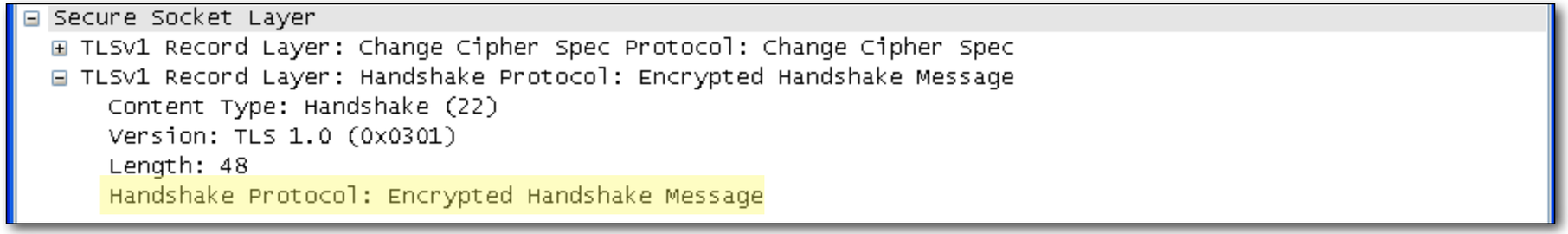
```
Secure Socket Layer
  TLSv1 Record Layer: Handshake Protocol: Client Key Exchange
  TLSv1 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
  TLSv1 Record Layer: Handshake Protocol: Finished
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 48
    Handshake Protocol: Finished
        Handshake Type: Finished (20)
        Length: 12
        Verify Data
```

ChangeCipherSpec (server)

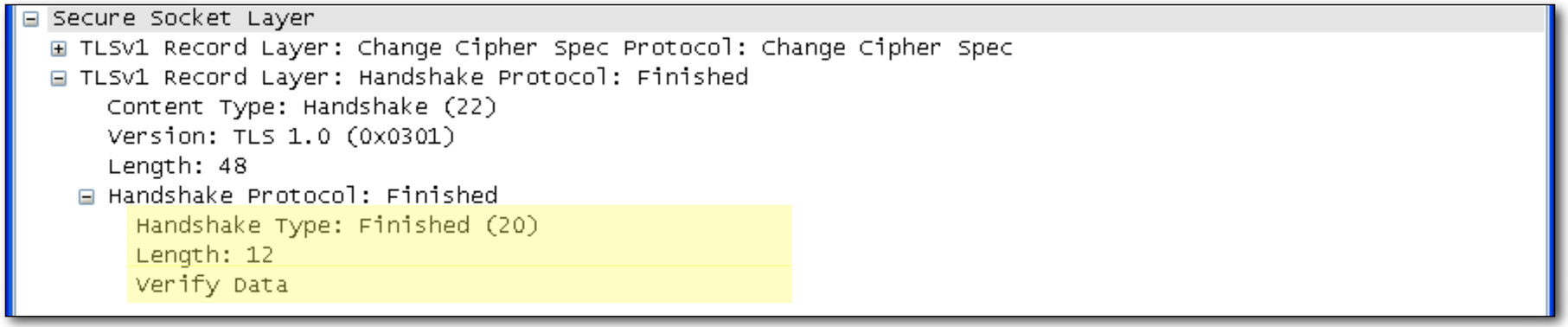


Finished (server)

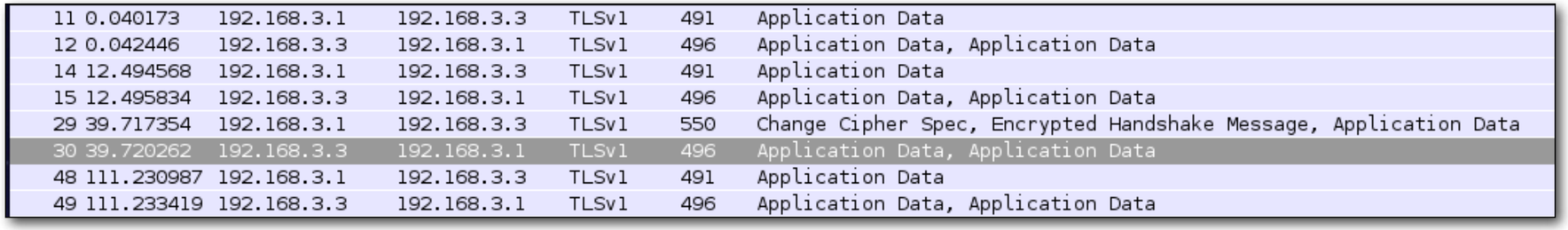
Without decryption:



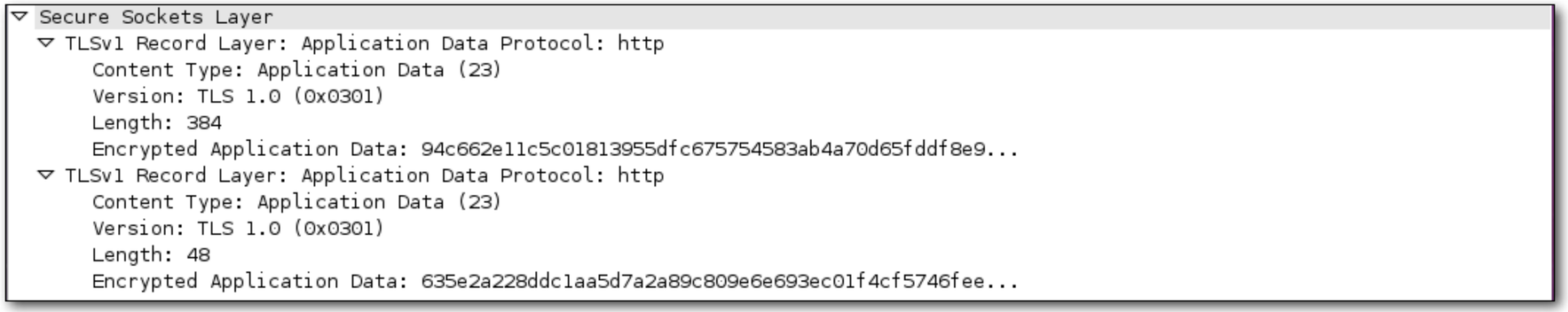
With decryption:



Analyzing TLS Application Data



11	0.040173	192.168.3.1	192.168.3.3	TLSv1	491	Application Data
12	0.042446	192.168.3.3	192.168.3.1	TLSv1	496	Application Data, Application Data
14	12.494568	192.168.3.1	192.168.3.3	TLSv1	491	Application Data
15	12.495834	192.168.3.3	192.168.3.1	TLSv1	496	Application Data, Application Data
29	39.717354	192.168.3.1	192.168.3.3	TLSv1	550	Change Cipher Spec, Encrypted Handshake Message, Application Data
30	39.720262	192.168.3.3	192.168.3.1	TLSv1	496	Application Data, Application Data
48	111.230987	192.168.3.1	192.168.3.3	TLSv1	491	Application Data
49	111.233419	192.168.3.3	192.168.3.1	TLSv1	496	Application Data, Application Data



```
▽ Secure Sockets Layer
  ▽ TLSv1 Record Layer: Application Data Protocol: http
      Content Type: Application Data (23)
      Version: TLS 1.0 (0x0301)
      Length: 384
      Encrypted Application Data: 94c662e11c5c01813955dfc675754583ab4a70d65fddf8e9...
  ▽ TLSv1 Record Layer: Application Data Protocol: http
      Content Type: Application Data (23)
      Version: TLS 1.0 (0x0301)
      Length: 48
      Encrypted Application Data: 635e2a228ddc1aa5d7a2a89c809e6e693ec01f4cf5746fee...
```

Later today: More on analysing Application Data

Analyzing SSL Alerts

Without decryption:

```
14 12.494568  192.168.3.1   192.168.3.3   TLSv1   Application Data
15 12.495834  192.168.3.3   192.168.3.1   TLSv1   Application Data, Application Data
17 27.530927  192.168.3.3   192.168.3.1   TLSv1   Encrypted Alert
20 32.811207  192.168.3.1   192.168.3.3   TLSv1   Encrypted Alert
```

```
Secure Socket Layer
  TLSv1 Record Layer: Encrypted Alert
      Content Type: Alert (21)
      Version: TLS 1.0 (0x0301)
      Length: 32
      Alert Message: Encrypted Alert
```

With decryption:

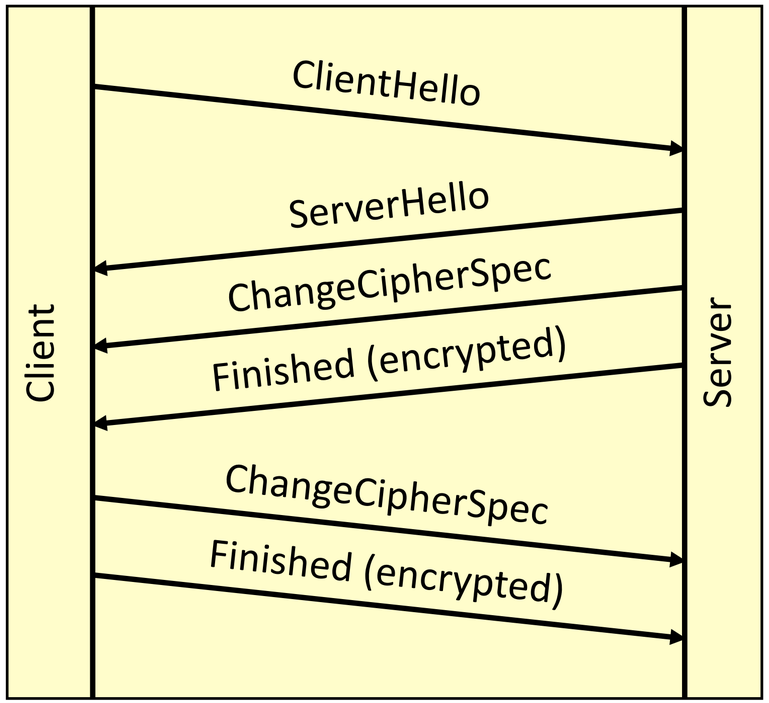
```
14 12.494568  192.168.3.1   192.168.3.3   HTTP    GET / HTTP/1.1
15 12.495834  192.168.3.3   192.168.3.1   HTTP    HTTP/1.1 200 OK  (text/html)
17 27.530927  192.168.3.3   192.168.3.1   TLSv1   Alert (Level: Warning, Description: Close Notify)
20 32.811207  192.168.3.1   192.168.3.3   TLSv1   Alert (Level: Warning, Description: Close Notify)
```

```
Secure Socket Layer
  TLSv1 Record Layer: Alert (Level: Warning, Description: Close Notify)
      Content Type: Alert (21)
      Version: TLS 1.0 (0x0301)
      Length: 32
    Alert Message
        Level: Warning (1)
        Description: Close Notify (0)
```

Reusing SSL sessions

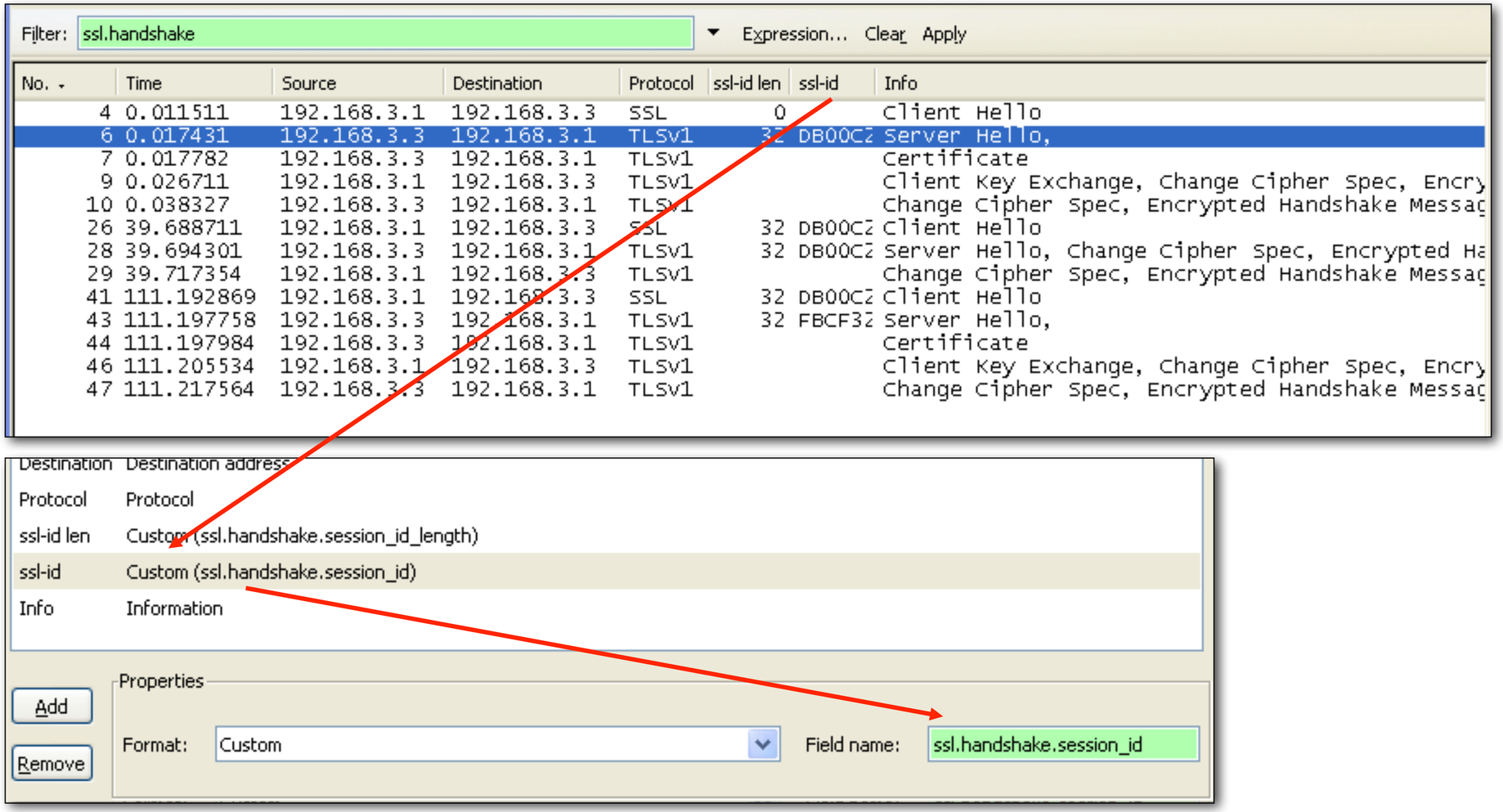
- Key negotiation "expensive"
- Cache session keys and re-use for new TCP sessions
- SSL session ID is used as Index
- Timeout on SSL session ID is an "absolute timeout" not an "idle timeout"
  - Old IE: 2 minutes, now 10 hours

Handshake of a Reused Session

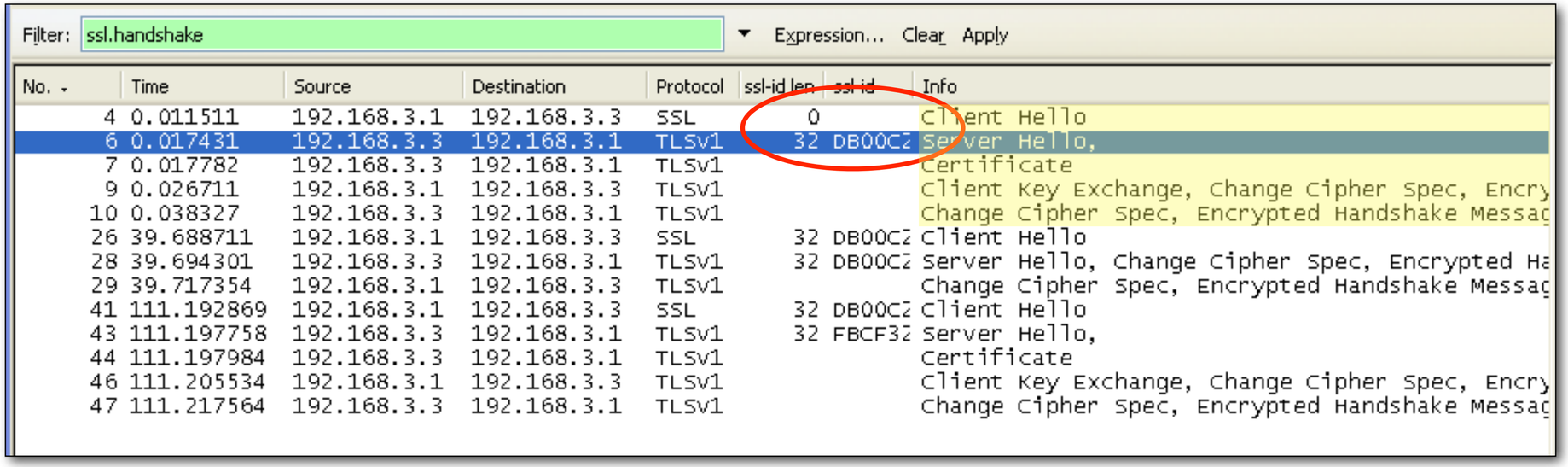


No.	Time	Source	Destination	Protocol	Info
23	39.68772(	192.168.3.1	192.168.3.3	TCP	18774 > https [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=1
24	39.68810]	192.168.3.3	192.168.3.1	TCP	https > 18774 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 WS=
25	39.68814(	192.168.3.1	192.168.3.3	TCP	18774 > https [ACK] Seq=1 Ack=1 Win=128000 Len=0
26	39.68871]	192.168.3.1	192.168.3.3	TLSv1	Client Hello
27	39.68898:	192.168.3.3	192.168.3.1	TCP	https > 18774 [ACK] Seq=1 Ack=103 Win=5840 Len=0
28	39.69430]	192.168.3.3	192.168.3.1	TLSv1	Server Hello, Change Cipher Spec, Encrypted Handshake Message
29	39.71735(	192.168.3.1	192.168.3.3	TLSv1	Change Cipher Spec, Encrypted Handshake Message, Application Dat

Adding a column for SSL session ID

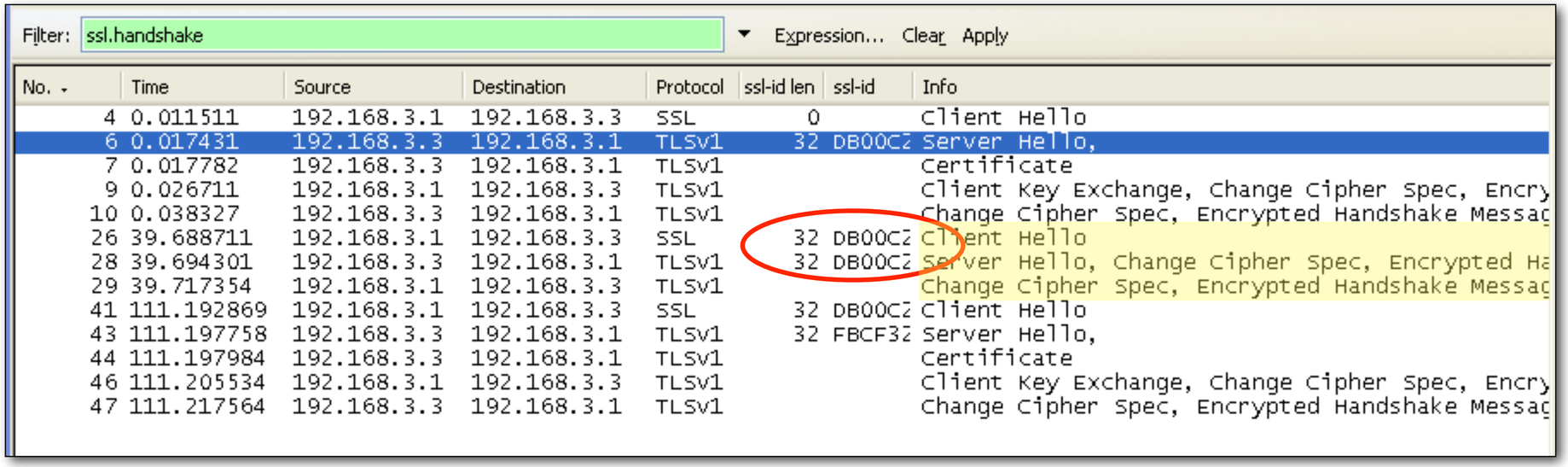


SSL session reuse I



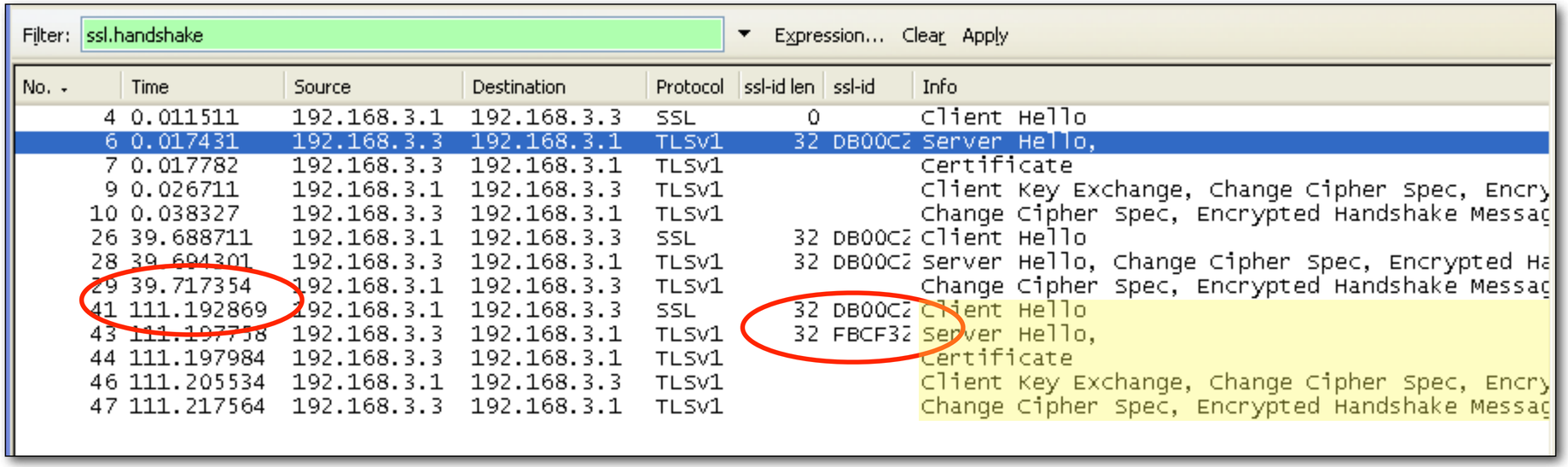
Full Handshake

SSL session reuse II



Partial Handshake

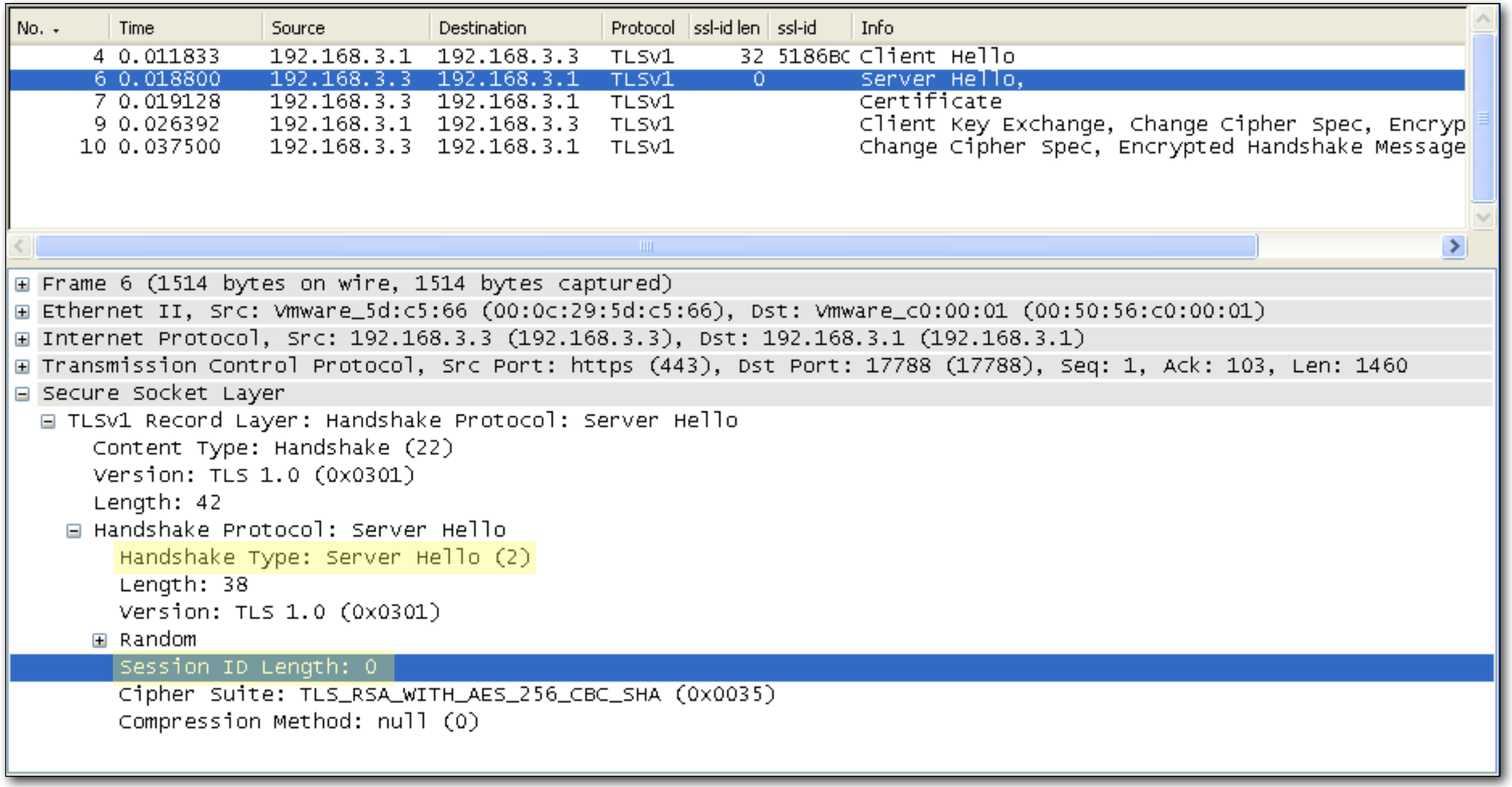
SSL session reuse III



Expired session

```
#   Inter-Process Session Cache:
#   Configure the SSL Session Cache: First the mechanism
#   to use and second the expiring timeout (in seconds).
#SSLSessionCache         dbm:/var/run/apache2/ssl_scache
SSLSessionCache        shmcb:/var/run/apache2/ssl_scache(512000)
SSLSessionCacheTimeout  60
```

No SSL session reuse



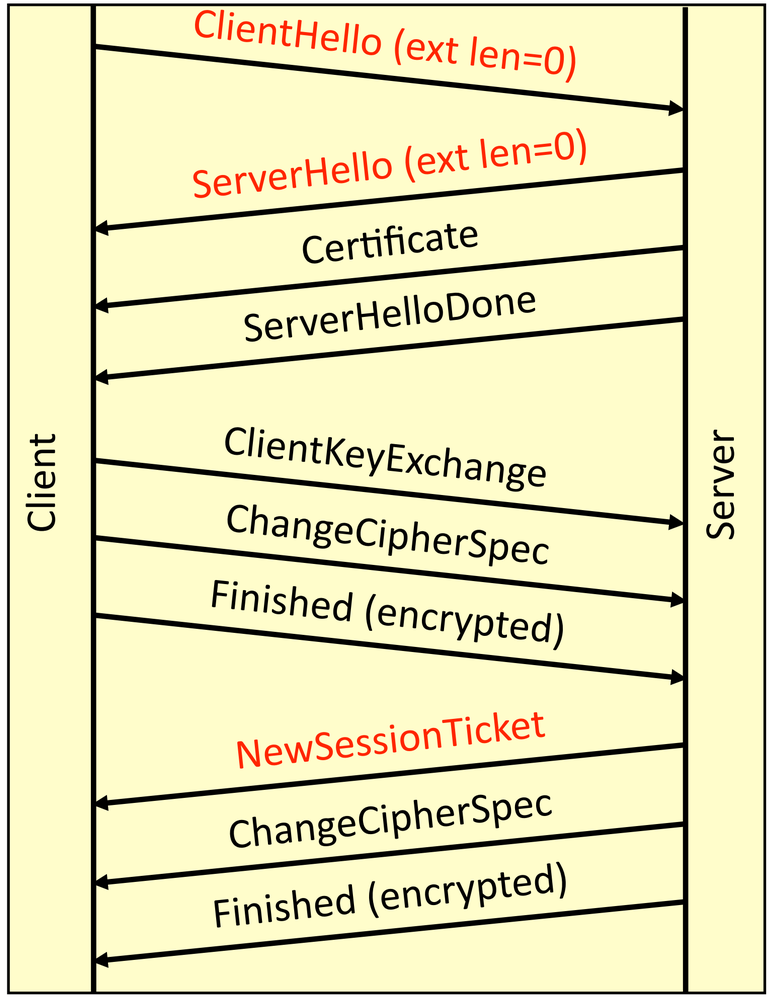
No.	Time	Source	Destination	Protocol	ssl-id len	ssl-id	Info
4	0.011833	192.168.3.1	192.168.3.3	TLSv1	32	5186BC	Client Hello
6	0.018800	192.168.3.3	192.168.3.1	TLSv1	0		Server Hello,
7	0.019128	192.168.3.3	192.168.3.1	TLSv1			Certificate
9	0.026392	192.168.3.1	192.168.3.3	TLSv1			Client Key Exchange, Change Cipher Spec, Encryp
10	0.037500	192.168.3.3	192.168.3.1	TLSv1			Change Cipher Spec, Encrypted Handshake Message

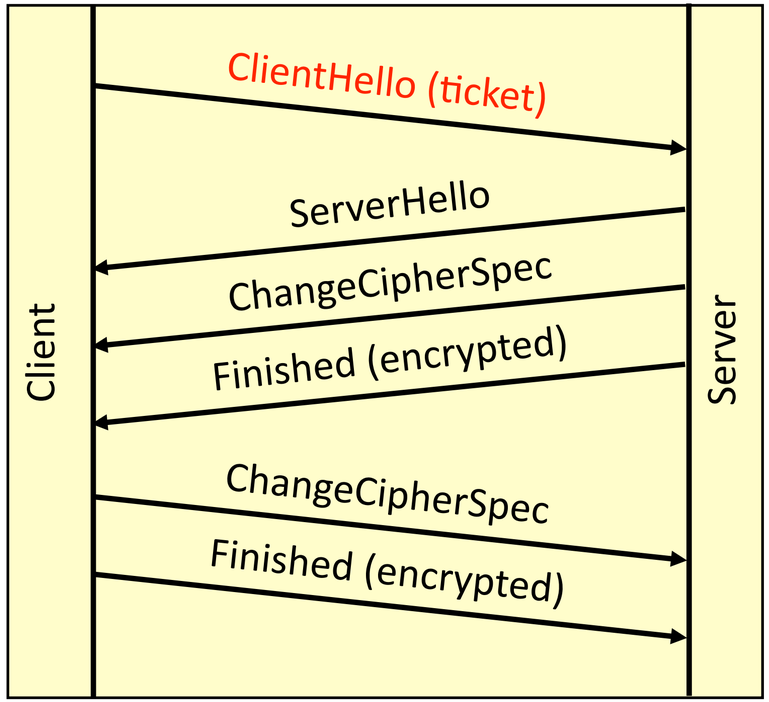
```
⊞ Frame 6 (1514 bytes on wire, 1514 bytes captured)
⊞ Ethernet II, Src: Vmware_5d:c5:66 (00:0c:29:5d:c5:66), Dst: Vmware_c0:00:01 (00:50:56:c0:00:01)
⊞ Internet Protocol, Src: 192.168.3.3 (192.168.3.3), Dst: 192.168.3.1 (192.168.3.1)
⊞ Transmission Control Protocol, Src Port: https (443), Dst Port: 17788 (17788), Seq: 1, Ack: 103, Len: 1460
⊟ Secure Socket Layer
  ⊟ TLSv1 Record Layer: Handshake Protocol: Server Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 42
    ⊟ Handshake Protocol: Server Hello
        Handshake Type: Server Hello (2)
        Length: 38
        Version: TLS 1.0 (0x0301)
      ⊞ Random
        Session ID Length: 0
        Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
        Compression Method: null (0)
```

TLS session Tickets

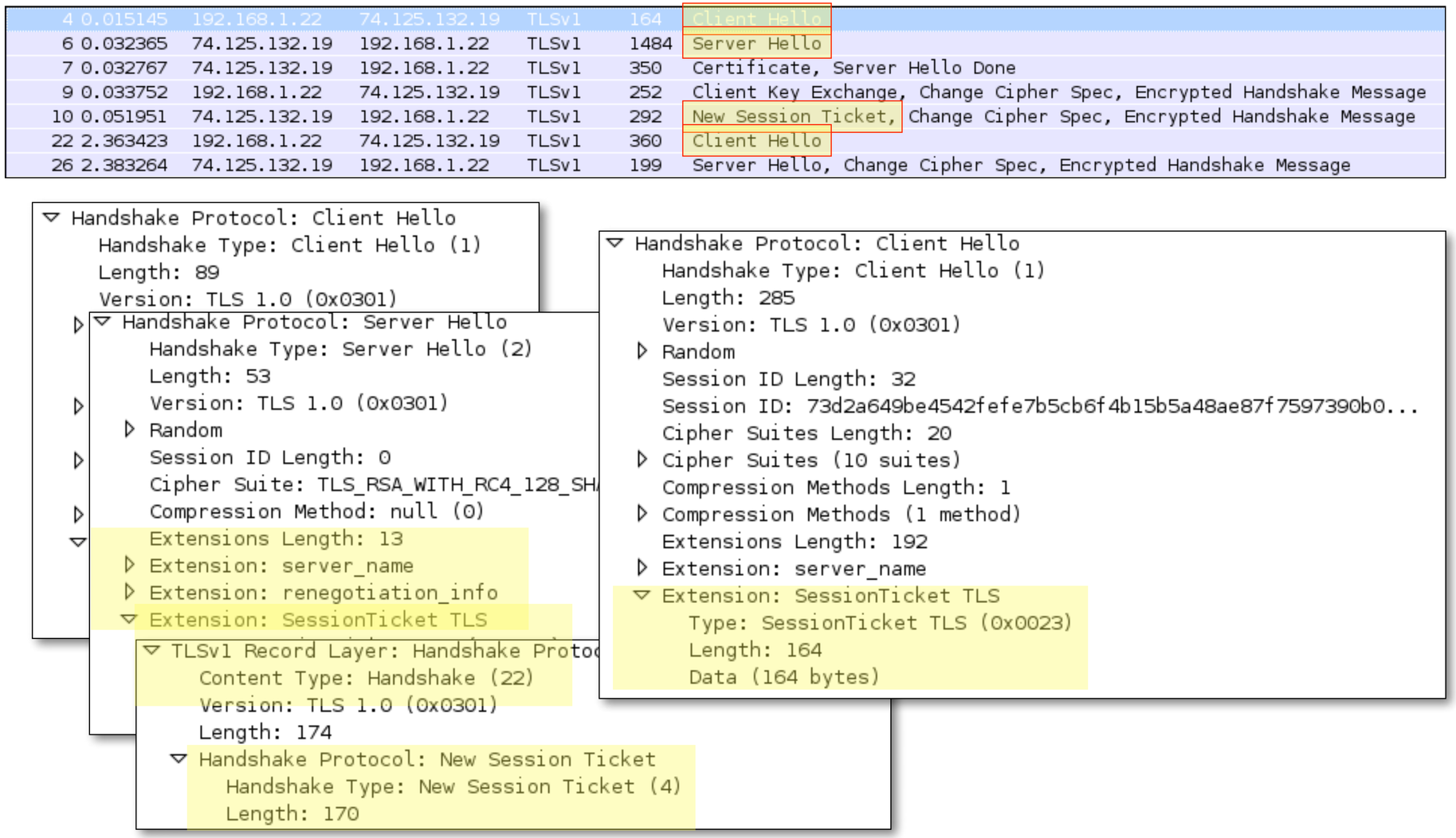
- TLS session tickets (RFC 5077)
- Do not keep state on server, only on client
- TLS extension in ClientHello and ServerHello
- New TLS HandshakeType: **NewSessionTicket**

TLS session Tickets (2)





TLS session Tickets

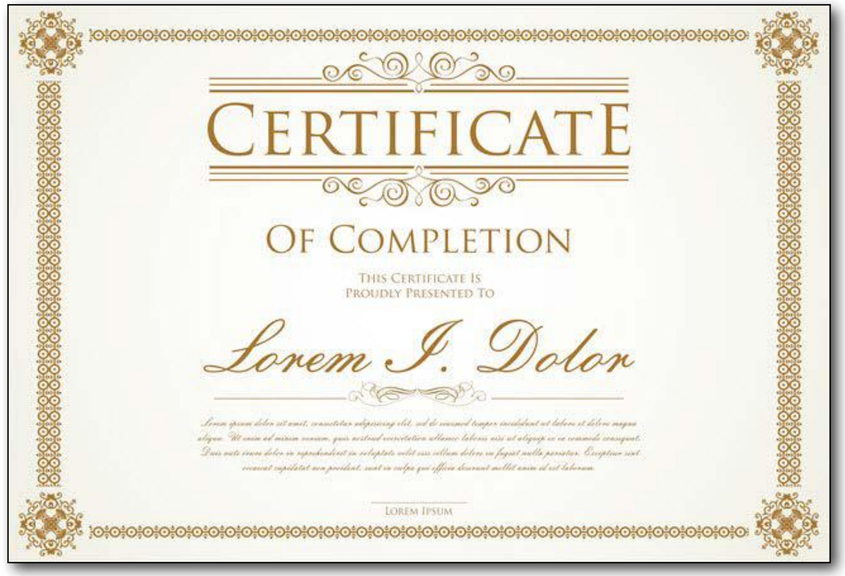


Common TLS handshake problems I

- Session stops right after “ClientHello”
  - No mutually supported TLS version
  - No mutually accepted cipher
  - No certificate for SNI name

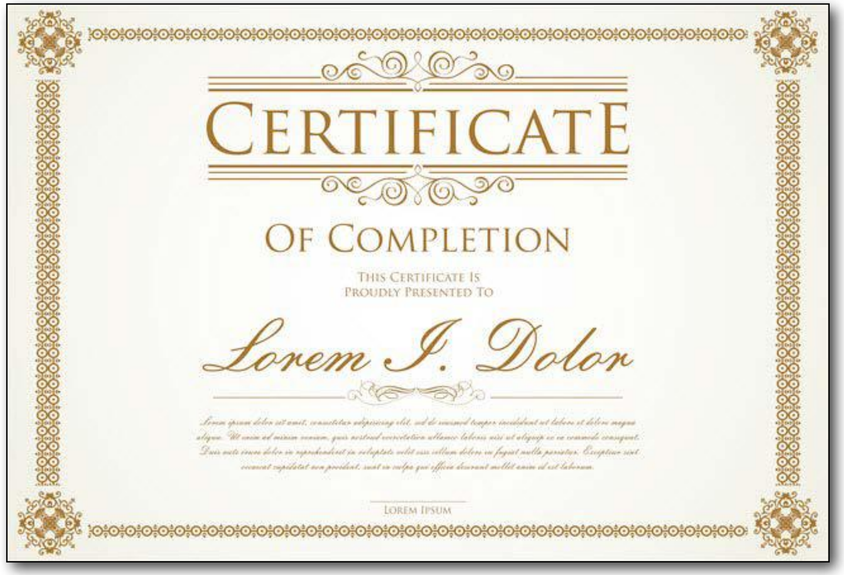
Common TLS handshake problems II

- Session stops right after “Certificate”
  - The Common Name in the certificate does not match with the requested hostname
  - The root certificate for the server certificate is not in the client’s trust store
  - Client has root certificate, but server does not send the intermediate CA certificate
  - Server certificate has expired
  - Server certificate has been revoked

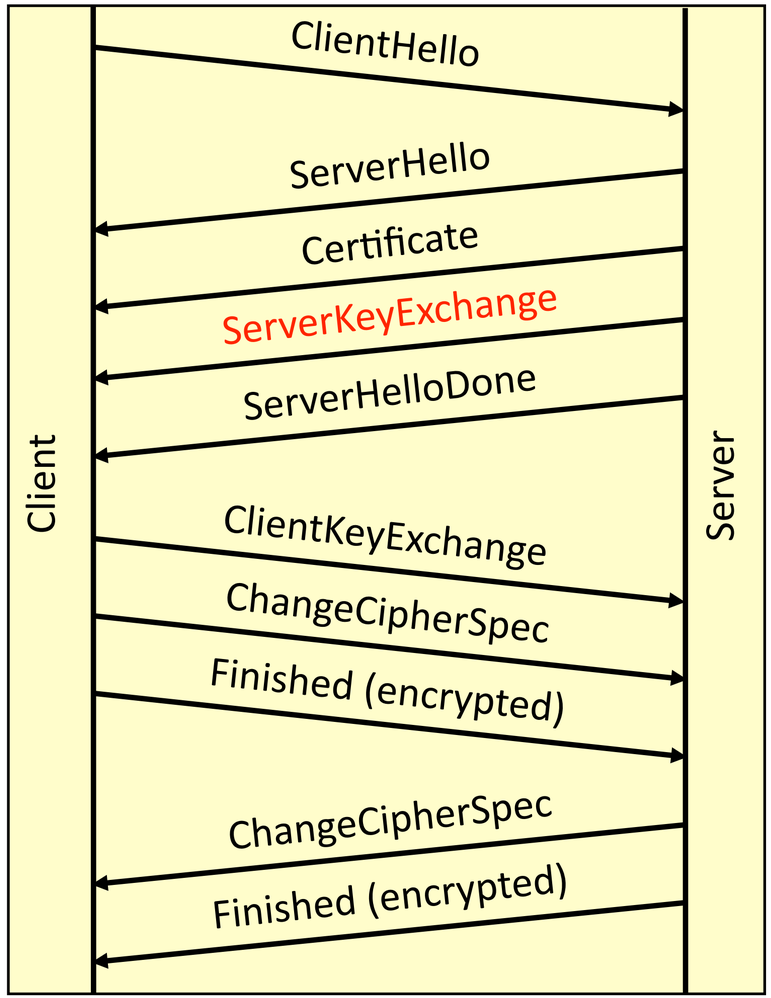


Common TLS handshake problems III

- Session stops right after "Certificate" (from the client)
  - The root certificate for the client certificate is not in the server's trust store
  - Server has root certificate, but client does not send the intermediate CA certificate
  - Client did not send a certificate
    - perhaps it did not have a certificate signed by one of the CA's in the DN list
  - Client certificate has expired
  - Client certificate has been revoked
  - The CRL has expired
  - Client certificate chain is too long

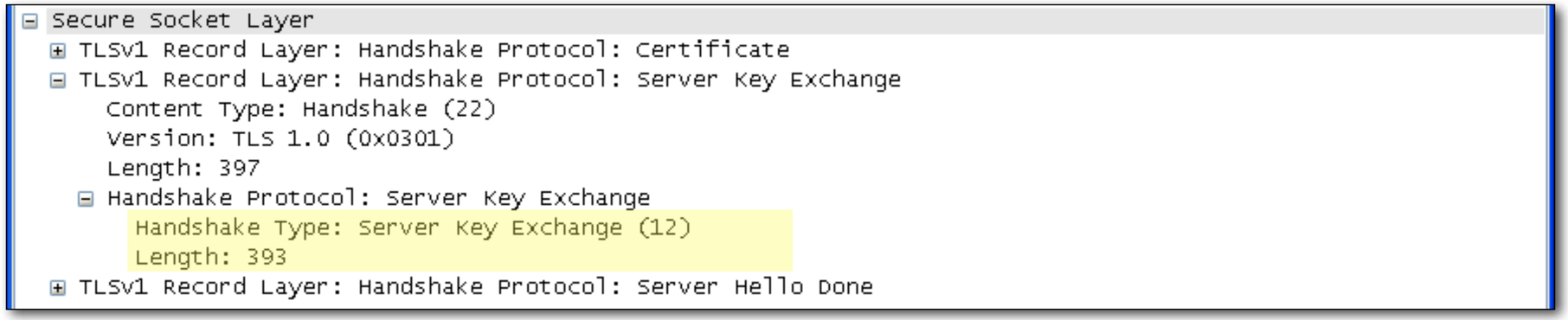


Handshake with DHE key generation



No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.3.1	192.168.3.3	TCP	42370 > https [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=1
2	0.000577	192.168.3.3	192.168.3.1	TCP	https > 42370 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=146
3	0.000618	192.168.3.1	192.168.3.3	TCP	42370 > https [ACK] Seq=1 Ack=1 Win=128000 Len=0
4	0.026109	192.168.3.1	192.168.3.3	SSL	Client Hello
5	0.026465	192.168.3.3	192.168.3.1	TCP	https > 42370 [ACK] Seq=1 Ack=107 Win=5840 Len=0
6	0.070925	192.168.3.3	192.168.3.1	TLSv1	Server Hello,
7	0.071108	192.168.3.3	192.168.3.1	TLSv1	Certificate, Server Key Exchange, Server Hello Done
8	0.071172	192.168.3.1	192.168.3.3	TCP	42370 > https [ACK] Seq=107 Ack=2828 Win=128000 Len=0
9	0.090279	192.168.3.1	192.168.3.3	TLSv1	Client Key Exchange, Change Cipher Spec, Encrypted Handshak
10	0.090657	192.168.3.3	192.168.3.1	TCP	https > 42370 [ACK] Seq=2828 Ack=305 Win=6912 Len=0
11	0.110494	192.168.3.3	192.168.3.1	TLSv1	Change Cipher Spec, Encrypted Handshake Message

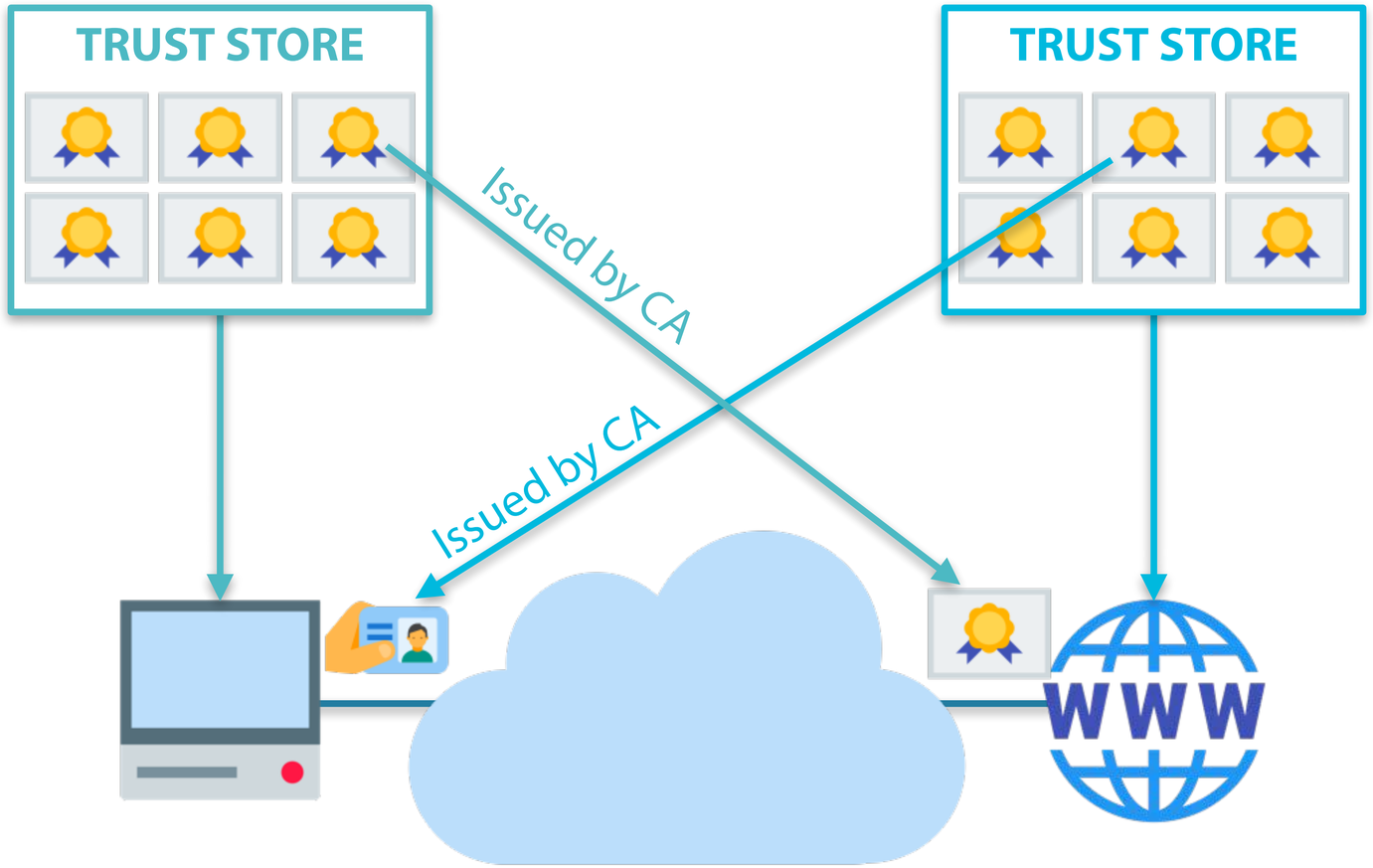
ServerKeyExchange



Client Authentication / Mutual TLS

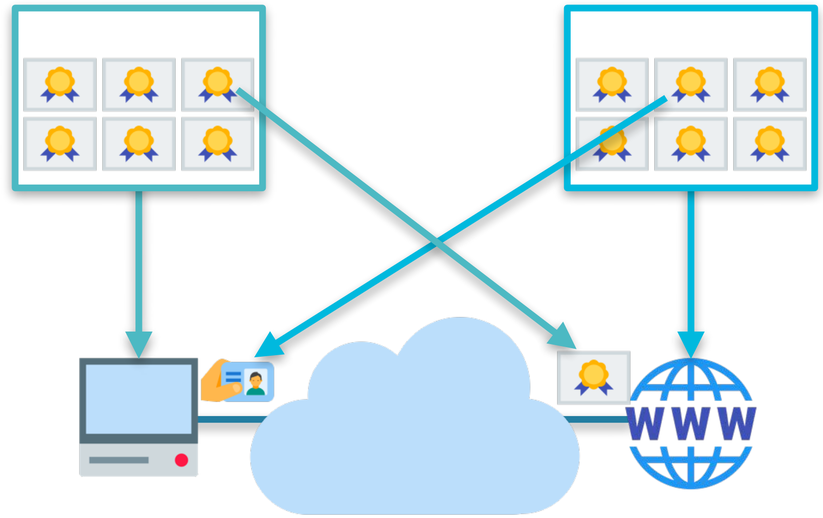
- Server asks certificate from the client
- Client presents its certificate
- Server performs the checks
- ID in the client certificate is used for authentication / authorization

Mutual TLS trust relationships



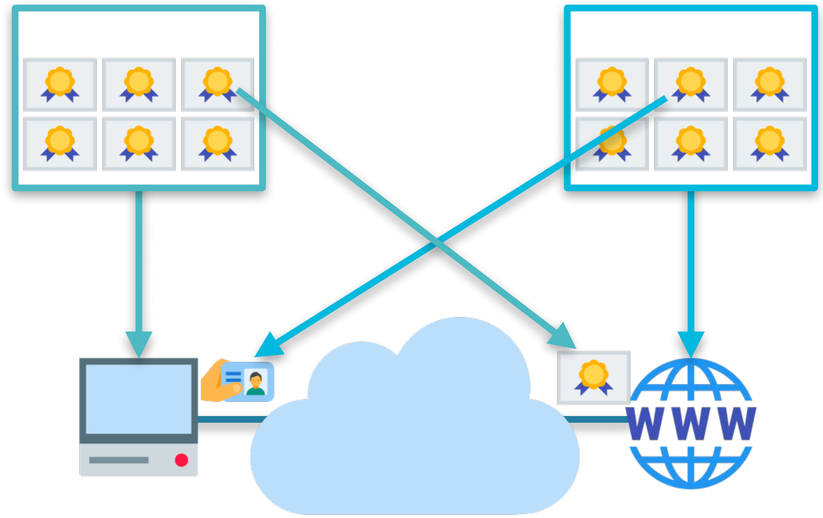
First configure regular TLS

- Server side
  - Add certificate
  - Add Intermediate CA's (don't include the RootCA)
  - Configure TLS version (TLS 1.2)
  - Limit cipher suites (check with SSL-labs)
- Client side
  - Add RootCA (already done by browser vendor)



Then configure the client authentication

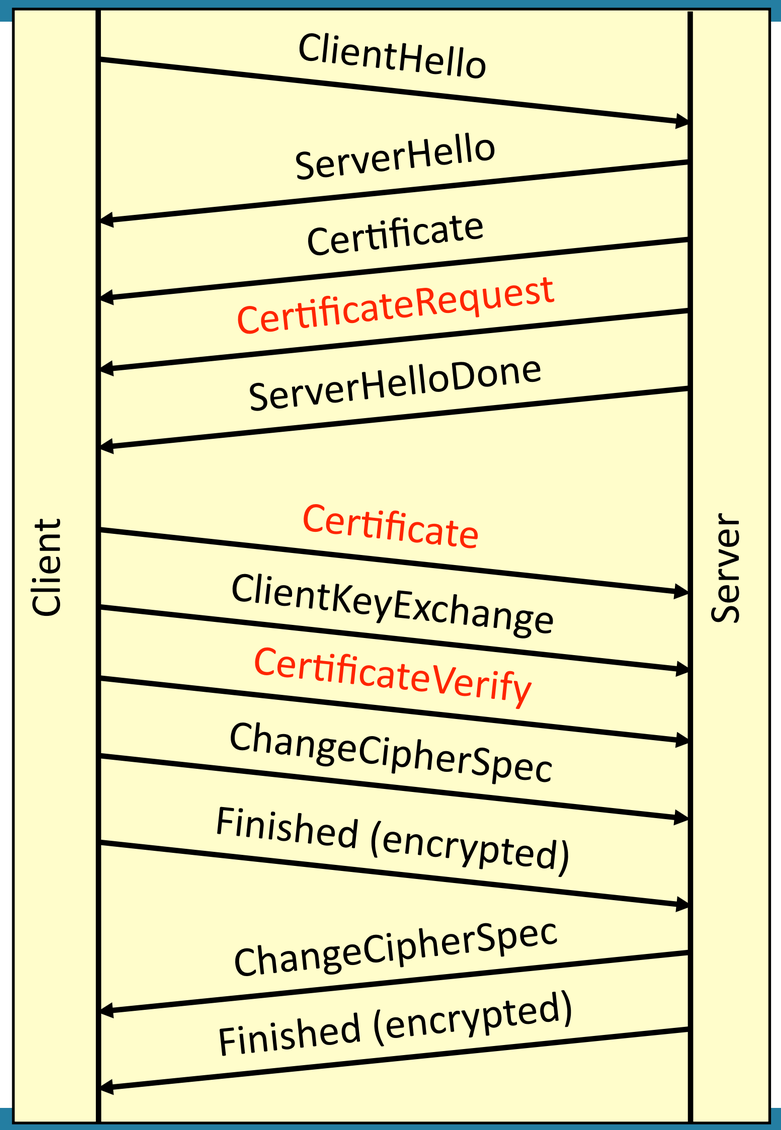
- Server side
  - Add list of CA's that from which client certificates are trusted
    - Optional: some devices need you to provide a list of DN's separately
  - Set "client certificates" to Request or Require
  - Configure CA depth (optional)
- Client side
  - Add client certificate (and private key), usually a PFX or P12 file
  - Add intermediate CA's (don't include the RootCA)



New Handshake Messages

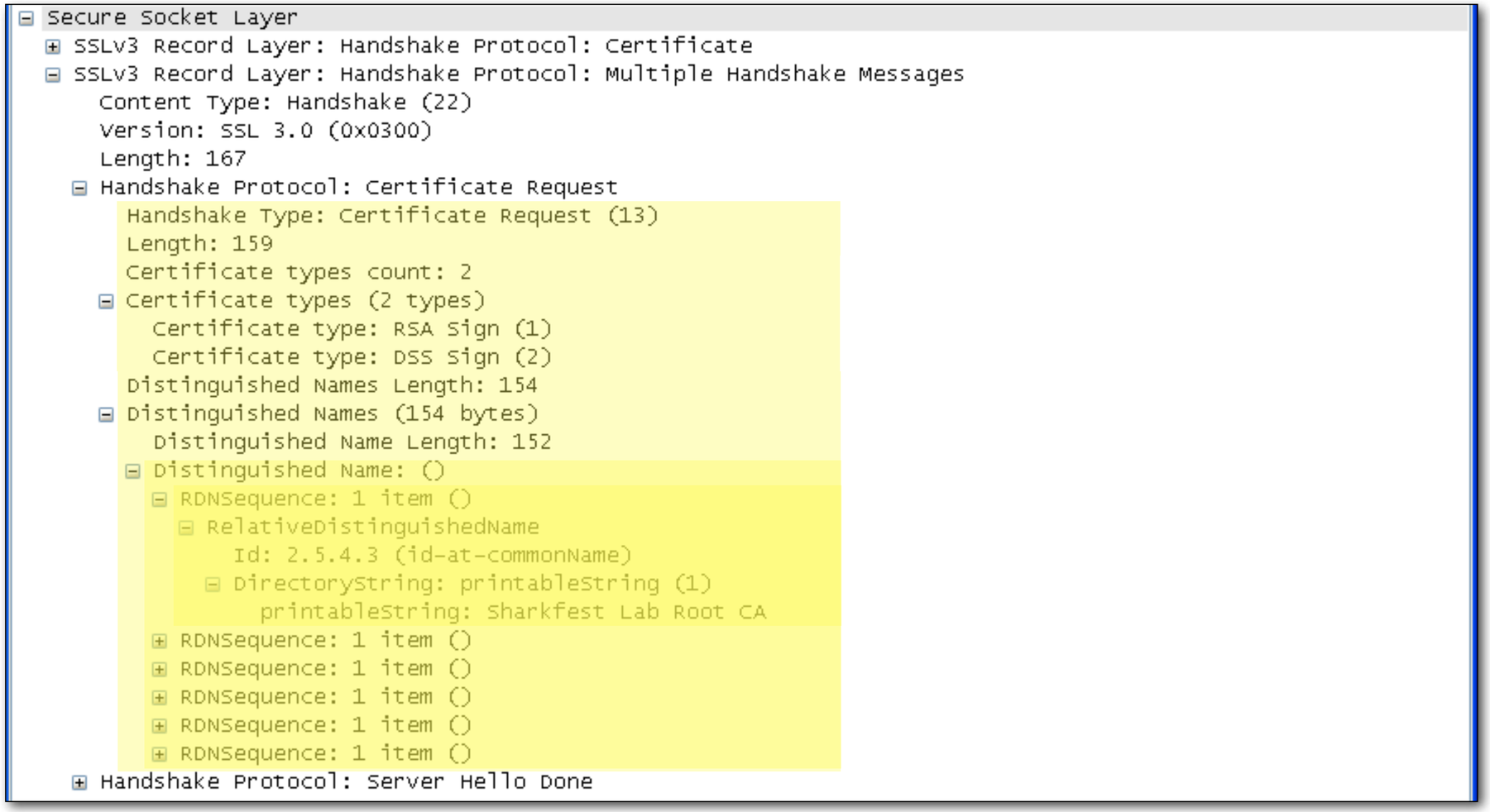
- CertificateRequest
  - Ask the client to present a certificate
  - Provide a list of DN's of CA's so the client can select the correct client certificate
    - Does NOT mean these CA's are trusted, configure properly!
- Certificate (client to server)
  - The client offers the certificate and intermediate CA's
  - Always sent, even when not sending a certificate
    - Then the length will be 0
- CertificateVerify
  - Provide signature of previous handshake messages
  - Used to proof possession of the private key

Client Authentication

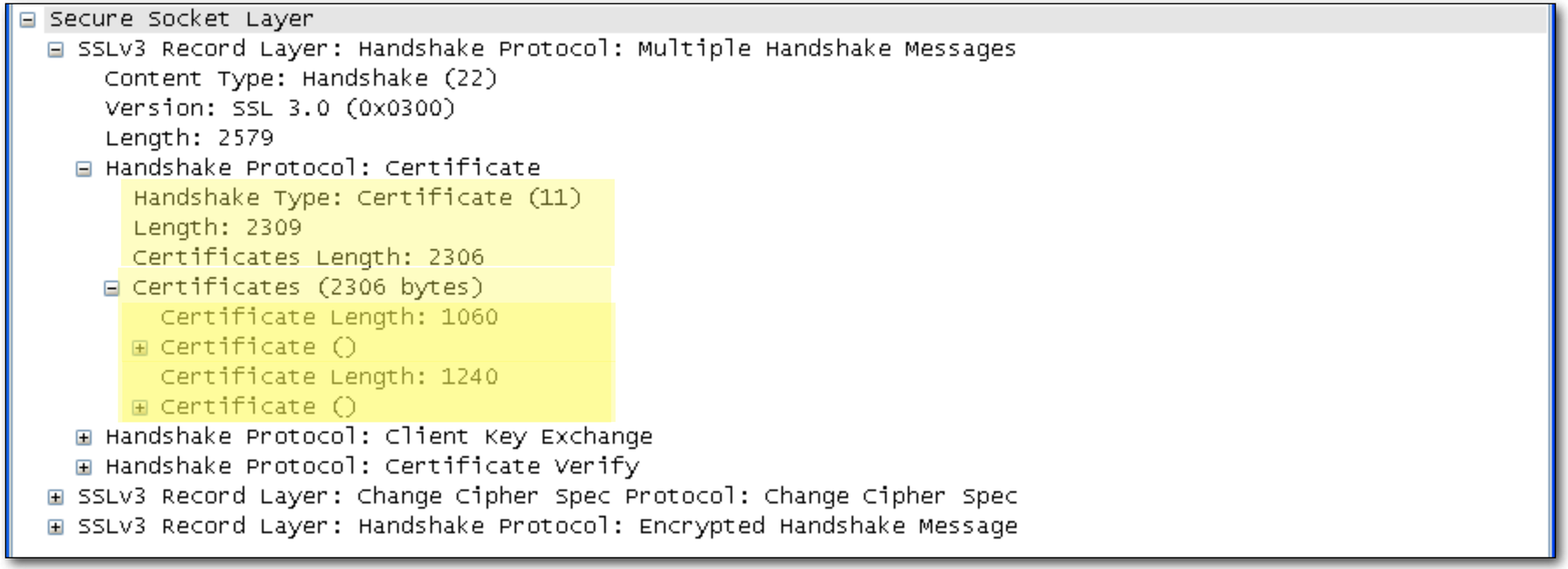


No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.3.1	192.168.3.4	TCP	14980 > https [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=1
2	0.000372	192.168.3.4	192.168.3.1	TCP	https > 14980 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=146
3	0.000400	192.168.3.1	192.168.3.4	TCP	14980 > https [ACK] Seq=1 Ack=1 Win=128000 Len=0
4	0.015645	192.168.3.1	192.168.3.4	SSLv2	Client Hello
5	0.015824	192.168.3.4	192.168.3.1	TCP	https > 14980 [ACK] Seq=1 Ack=52 Win=5840 Len=0
6	0.017894	192.168.3.4	192.168.3.1	SSLv3	Server Hello,
7	0.017988	192.168.3.4	192.168.3.1	SSLv3	Certificate, Certificate Request, Server Hello Done
8	0.018015	192.168.3.1	192.168.3.4	TCP	14980 > https [ACK] Seq=52 Ack=2590 Win=128000 Len=0
9	4.089191	192.168.3.1	192.168.3.4	TCP	[TCP segment of a reassembled PDU]
10	4.089622	192.168.3.4	192.168.3.1	TCP	https > 14980 [ACK] Seq=2590 Ack=1512 Win=8768 Len=0
11	4.089949	192.168.3.1	192.168.3.4	SSLv3	Certificate, Client Key Exchange, Certificate Verify, Chan
12	4.107141	192.168.3.4	192.168.3.1	SSLv3	Change Cipher Spec, Encrypted Handshake Message

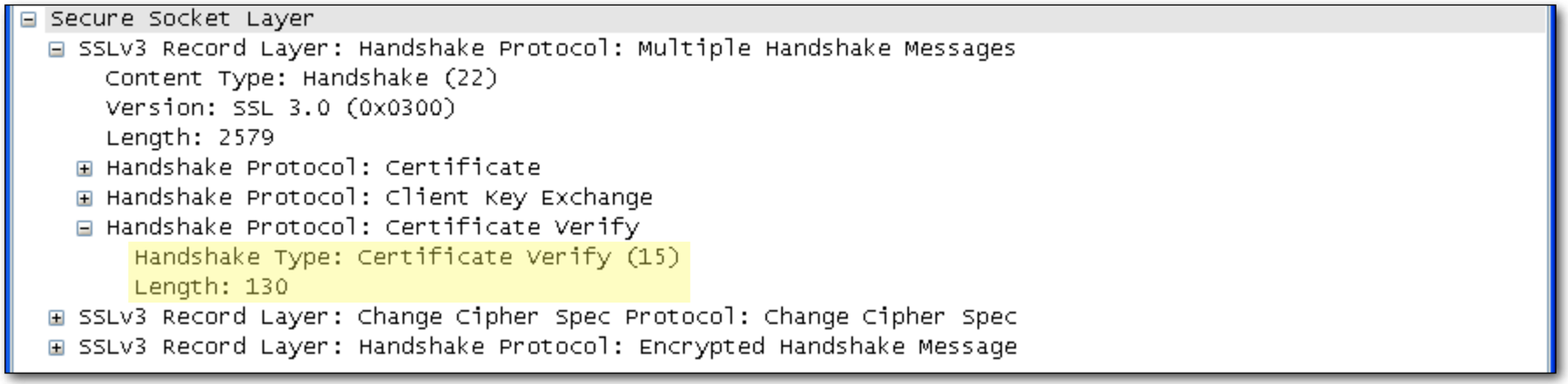
CertificateRequest (server)



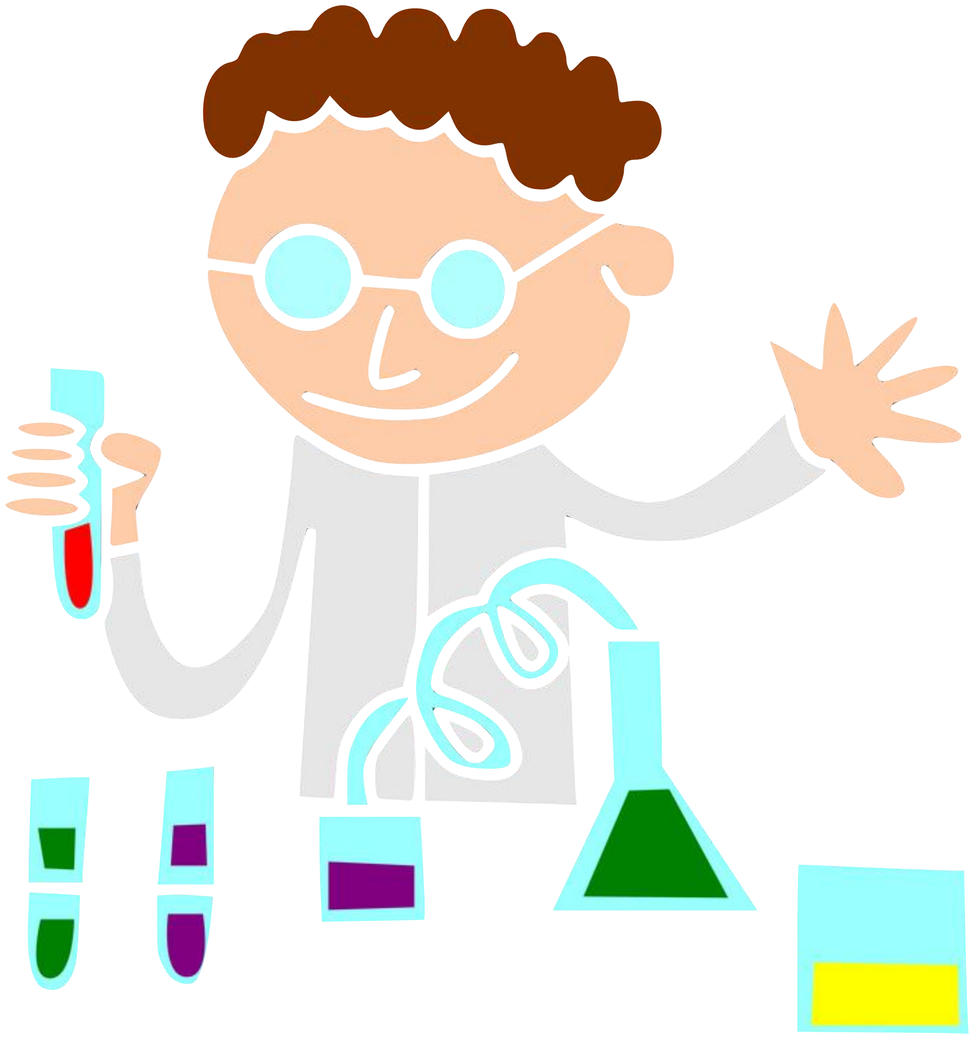
Certificate (client)



CertificateVerify (client)

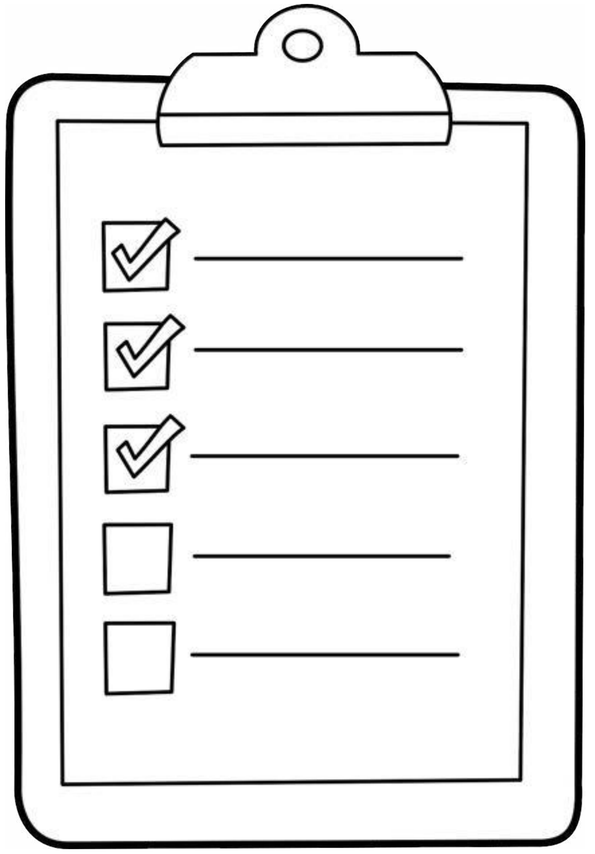


LAB 1



Agenda

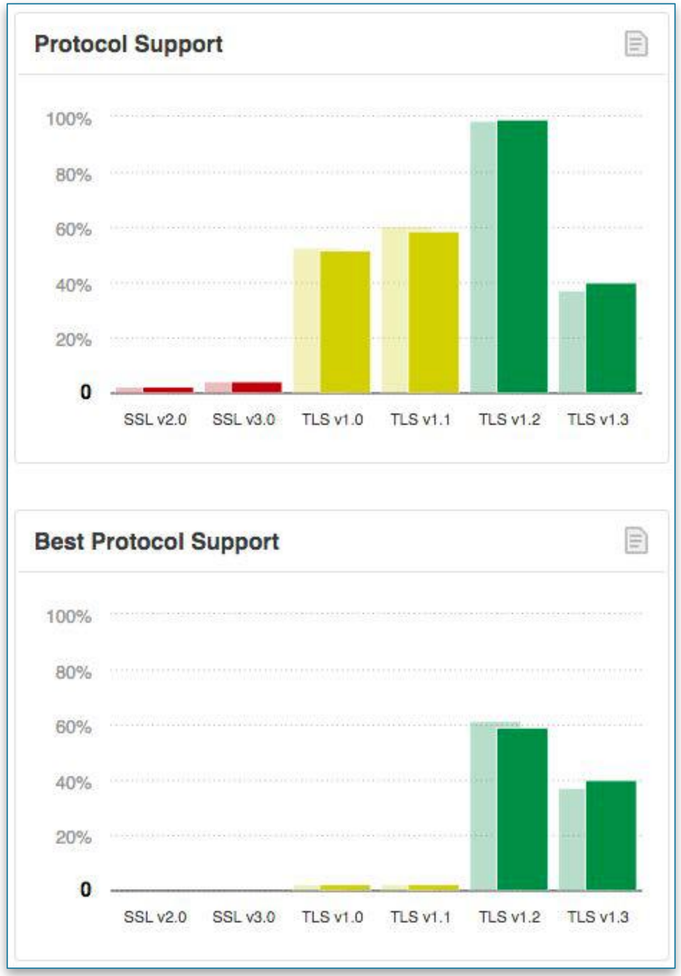
- TLS fundamentals
  - The need for TLS (and what happend to SSL?)
  - Cryptology 101 & PKI
  - Troubleshoot the TLS handshake (SSLv3 - TLSv1.2)
  - **Troubleshoot the TLS handshake (TLSv1.3)**
- Analysing Application Data
  - Without decryption
  - With decryption using the servers private key
  - With decryption using TLS session keys
- Creating configuration profiles for TLS analysis
- SSL/TLS vulnerabilities & privacy concerns
- Summary and Q&A



https://www.needpix.com/photo/download/94588/checklist-action-check-list-verification-tick-approved-okay-check-mark

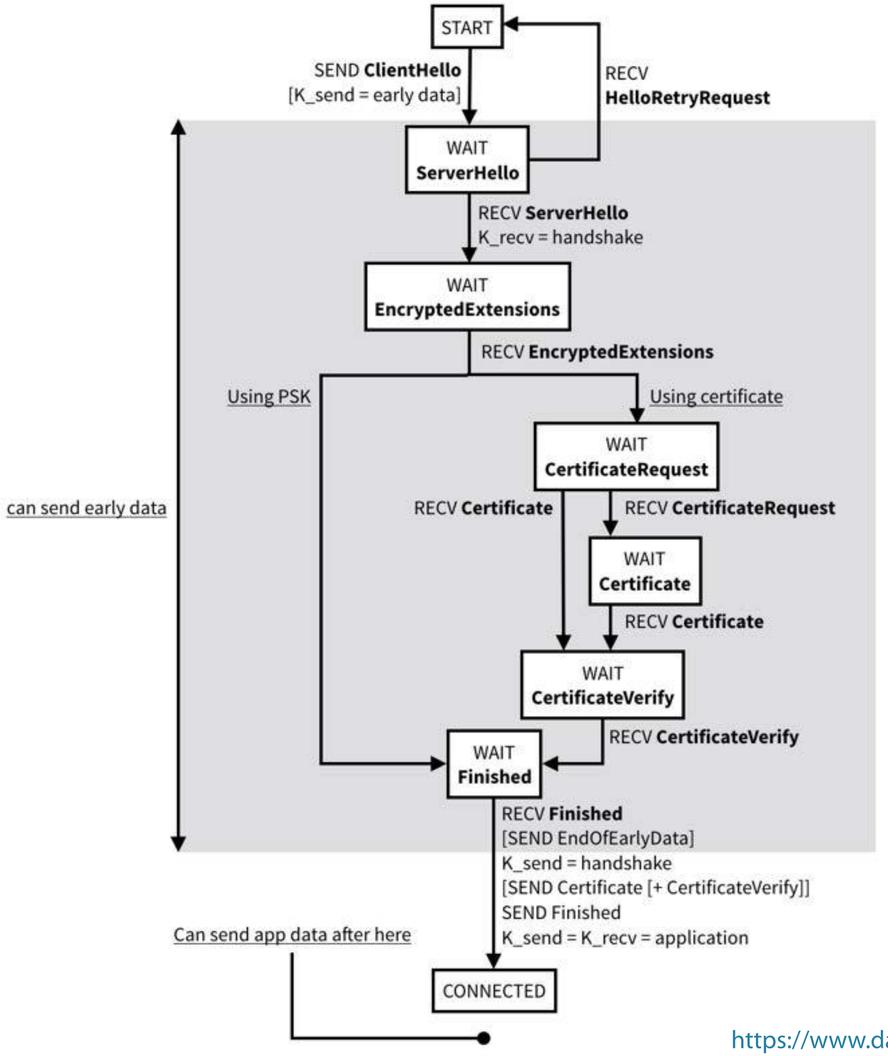
From TLSv1.2 to TLSv1.3

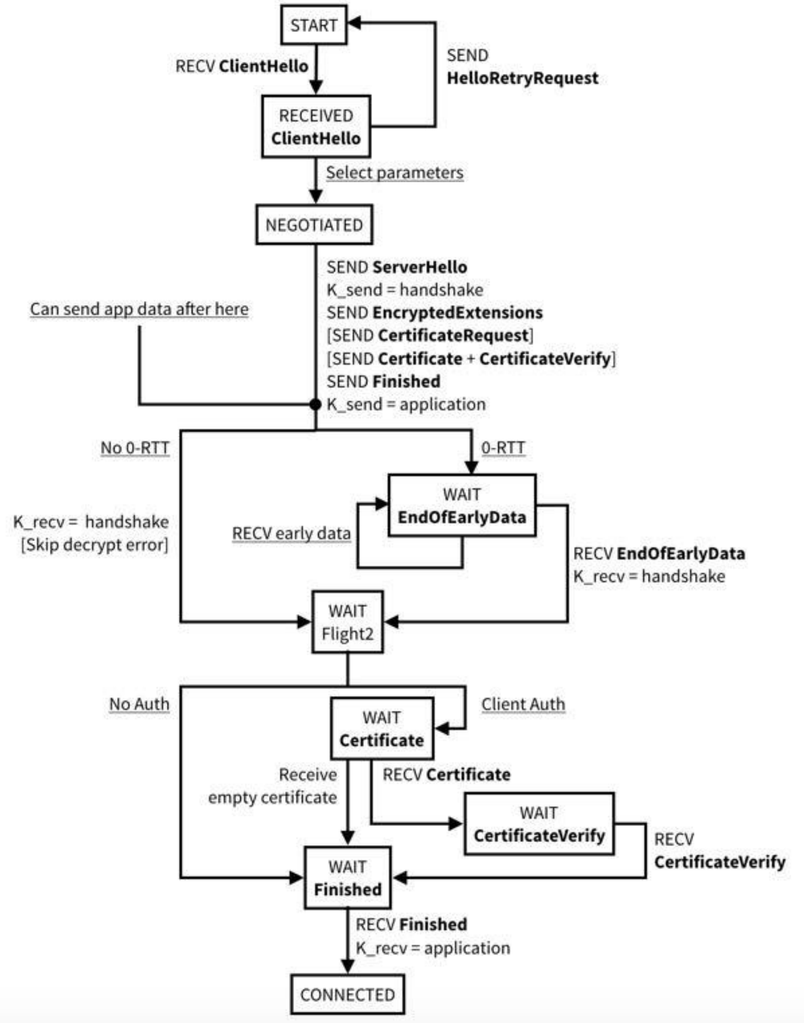
- Increased security by removal of support for:
  - legacy bulk encryption protocols like RC4
  - static RSA and DH cipher suites (now always PFS)
  - Old hashing algorithms like MD5 and SHA1
  - Compression
  - Renegotiation
- Added features:
  - Significantly restructured handshake
    - 1-RTT handshake by default, optional 0-RTT
    - Middlebox compatibility
  - encryption starts after ServerHello
  - Downgrade protection
  - Support for ECC in base spec and new algorithms included
  - TLS version negotiation extension
  - Single PSK exchange for session resumption



https://cds.cern.ch/record/39437

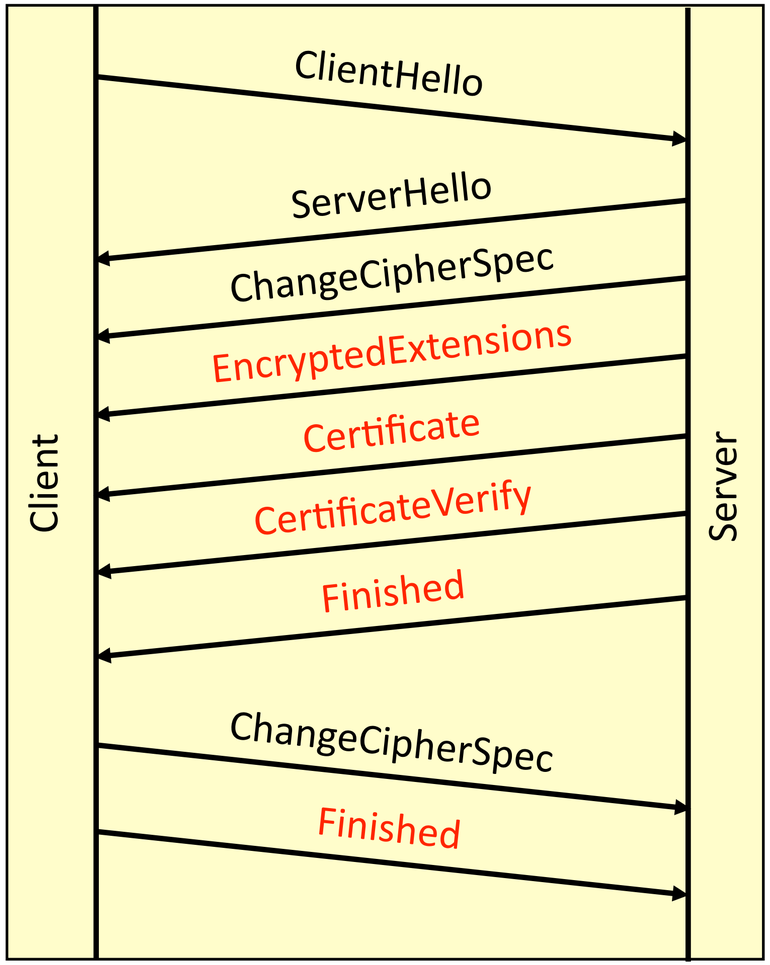
TLSv1.3 state diagrams





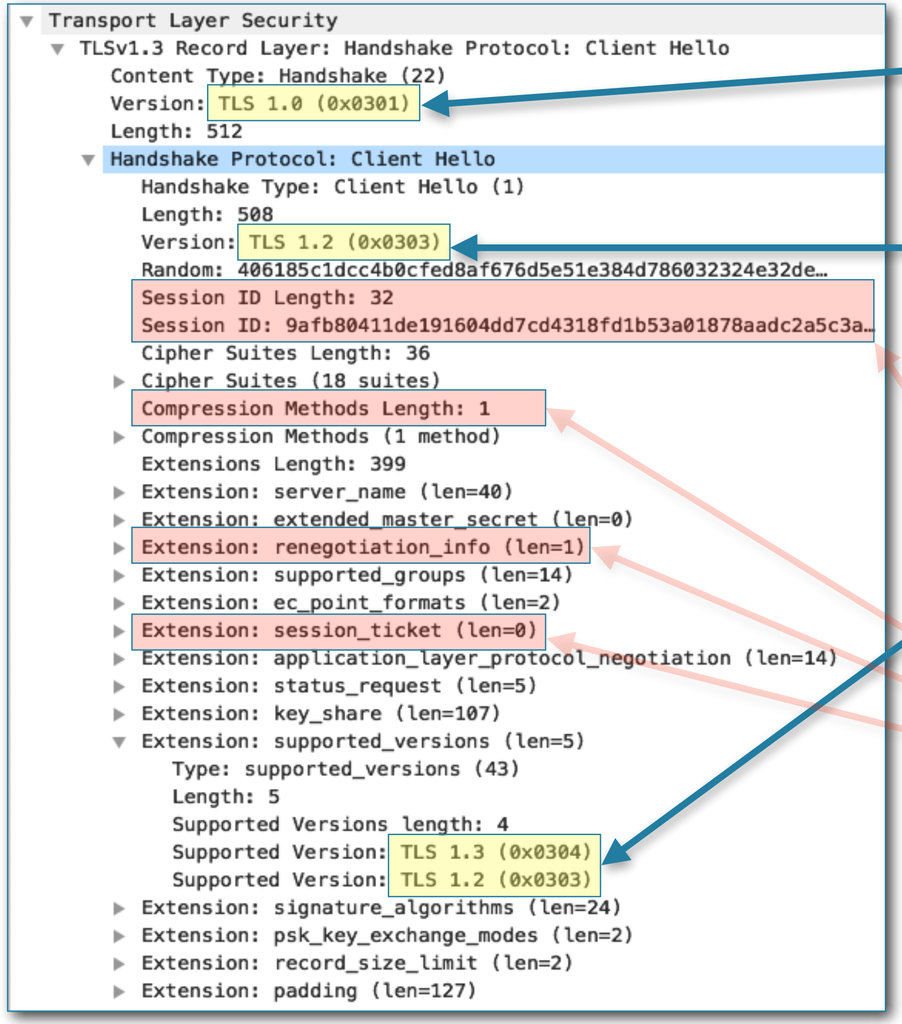
https://www.davidwong.fr/tls13/

TLSv1.3 Handshake



No.	Time	Delta	Source	Destination	Protocol	Lengt	Info
12	12:42:15.609765	0.000000	192.168.2.3	13.227.219.2	TCP	78	54463 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32
19	12:42:15.617106	0.007341	13.227.219.2	192.168.2.3	TCP	74	443 → 54463 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=
20	12:42:15.617142	0.000036	192.168.2.3	13.227.219.2	TCP	66	54463 → 443 [ACK] Seq=1 Ack=1 Win=131744 Len=0
23	12:42:15.619839	0.002697	192.168.2.3	13.227.219.2	TLSv1.3	583	Client Hello
25	12:42:15.628080	0.008241	13.227.219.2	192.168.2.3	TCP	66	443 → 54463 [ACK] Seq=1 Ack=518 Win=30208 Len=0
26	12:42:15.630108	0.002028	13.227.219.2	192.168.2.3	TLSv1.3	1514	Server Hello, Change Cipher Spec, Encrypted Extensions
27	12:42:15.630111	0.000003	13.227.219.2	192.168.2.3	TCP	1514	443 → 54463 [ACK] Seq=1449 Ack=518 Win=30208 Len=1448 [
28	12:42:15.630155	0.000044	192.168.2.3	13.227.219.2	TCP	66	54463 → 443 [ACK] Seq=518 Ack=2897 Win=128864 Len=0
32	12:42:15.633724	0.003569	13.227.219.2	192.168.2.3	TLSv1.3	1011	Certificate, Certificate Verify, Finished
33	12:42:15.633759	0.000035	192.168.2.3	13.227.219.2	TCP	66	54463 → 443 [ACK] Seq=518 Ack=3842 Win=130112 Len=0
40	12:42:15.641691	0.007932	192.168.2.3	13.227.219.2	TLSv1.3	130	Change Cipher Spec, Finished

ClientHello (client)



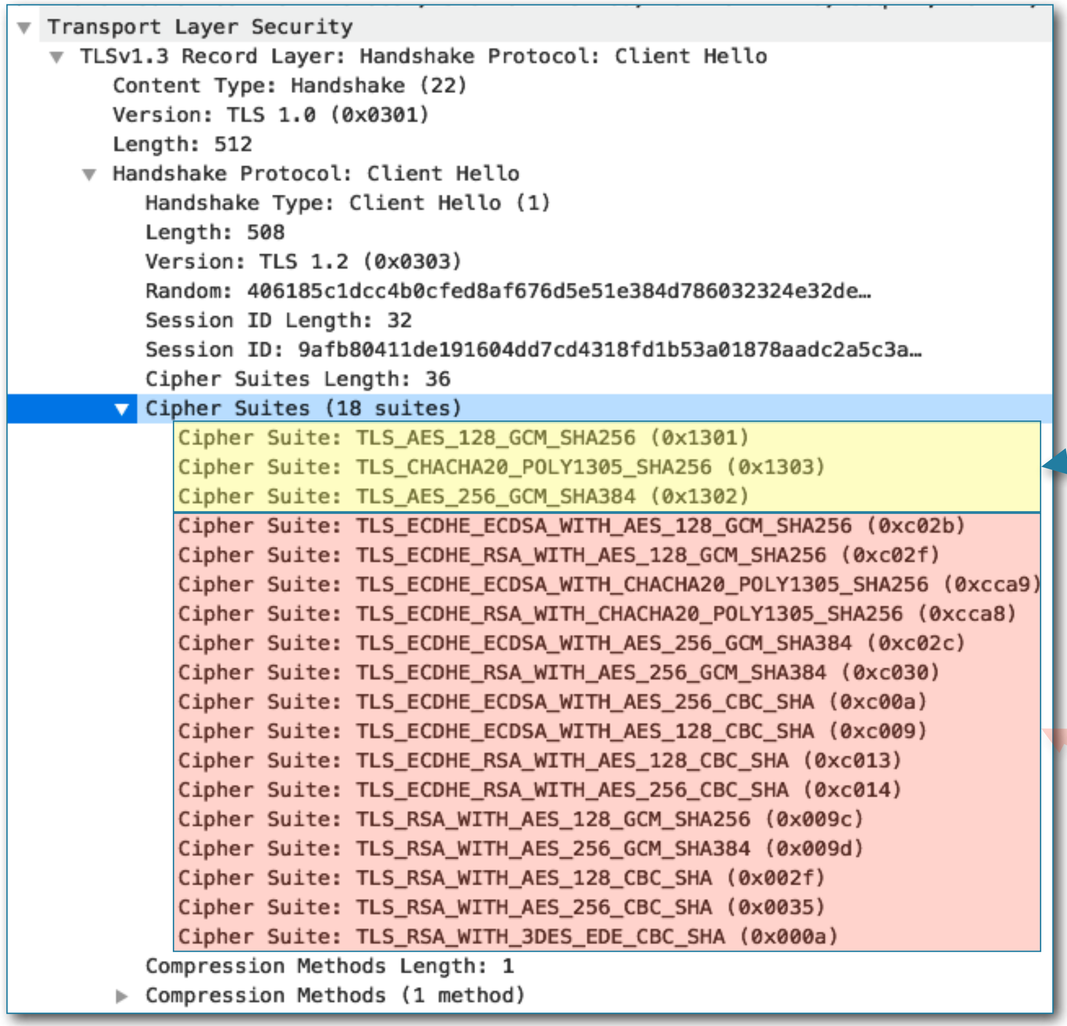
Minimum supported version

maximum supported version (TLSv1.2 compatibility)

supported versions (TLSv1.3 extension)

Not (or differently) used in TLSv1.3

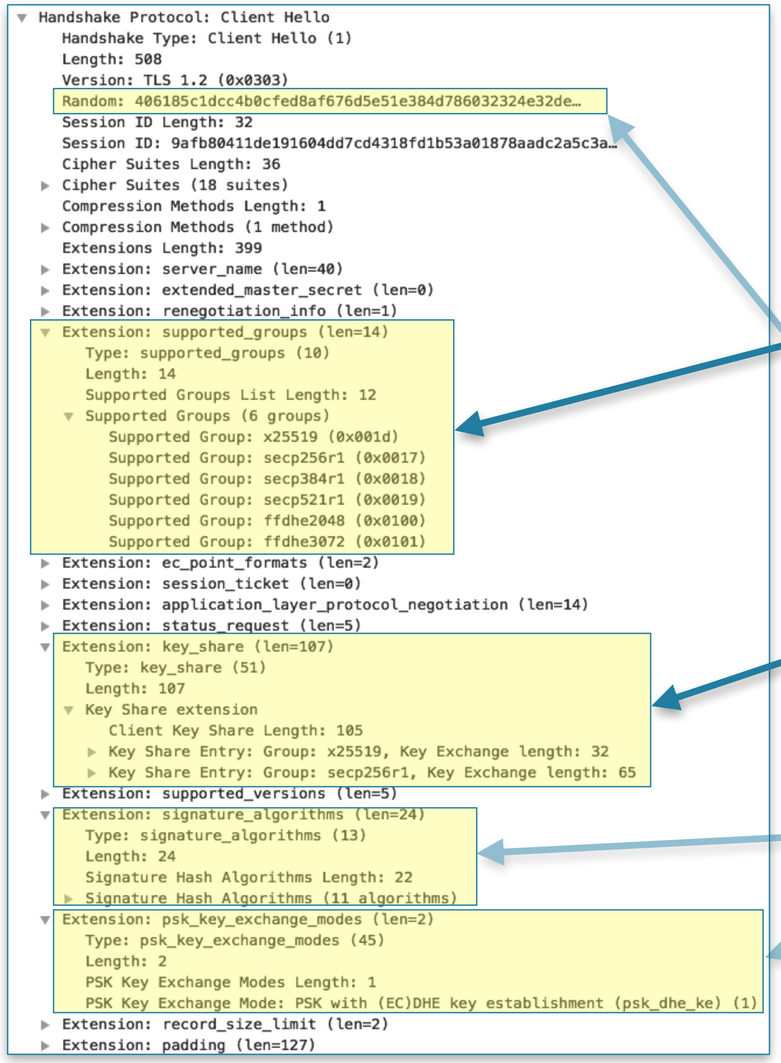
ClientHello (client)



TLSv1.3 ciphersuites

TLSv1.2 ciphersuites

ClientHello (client)

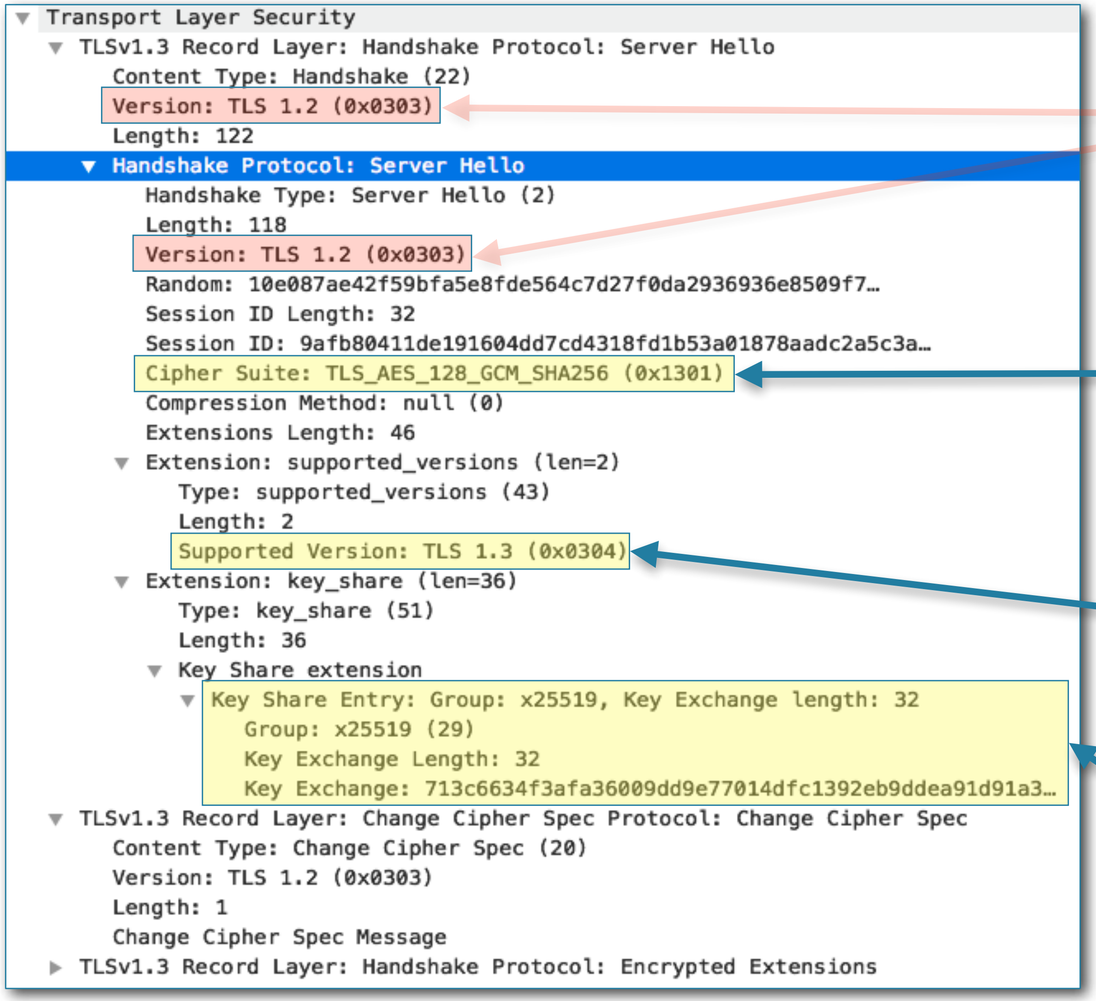


Supported Elliptic Curve Groups

Client Key Share (for some of the supported groups)

Also needed to derive the session keys

ServerHello (server)



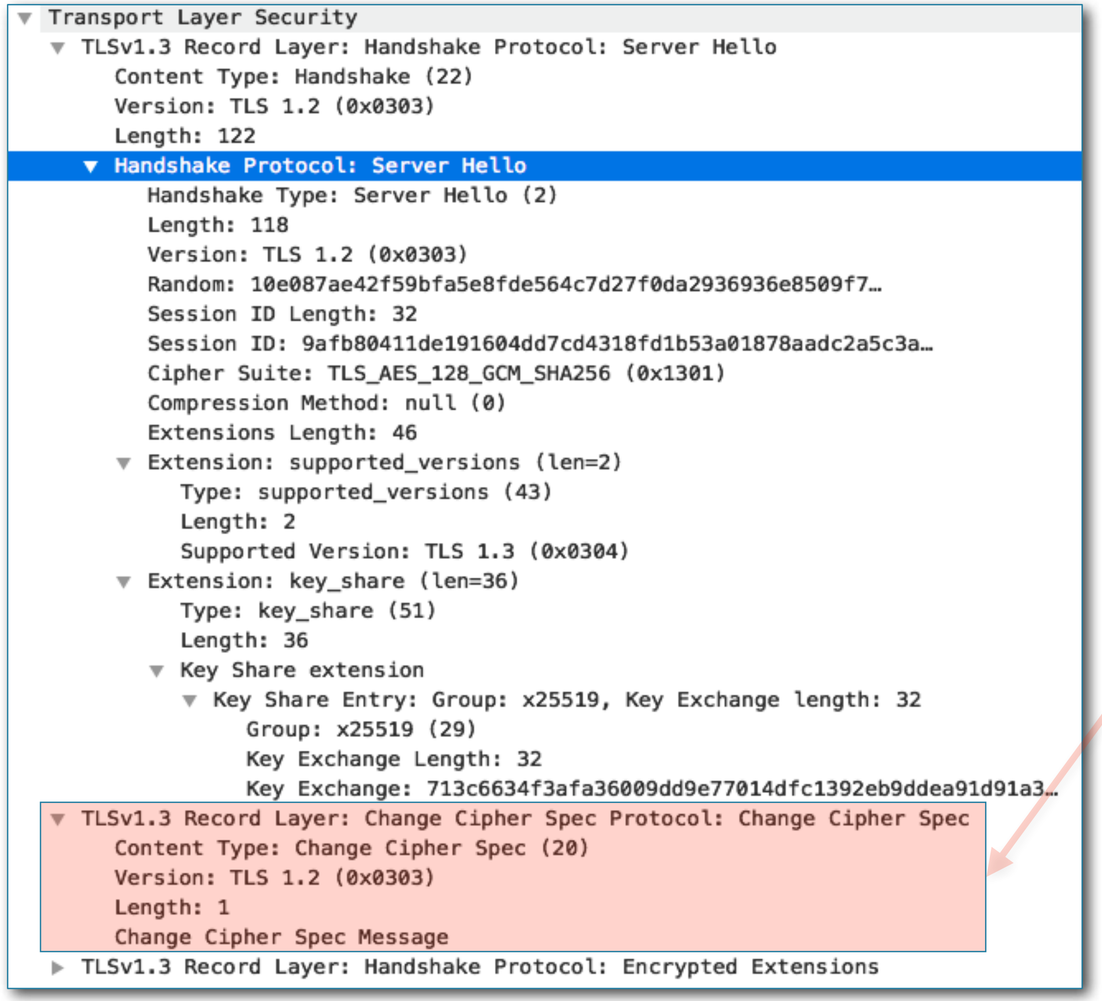
Dummy protocol version for middleboxes

Chosen cipher suite

Chosen protocol version

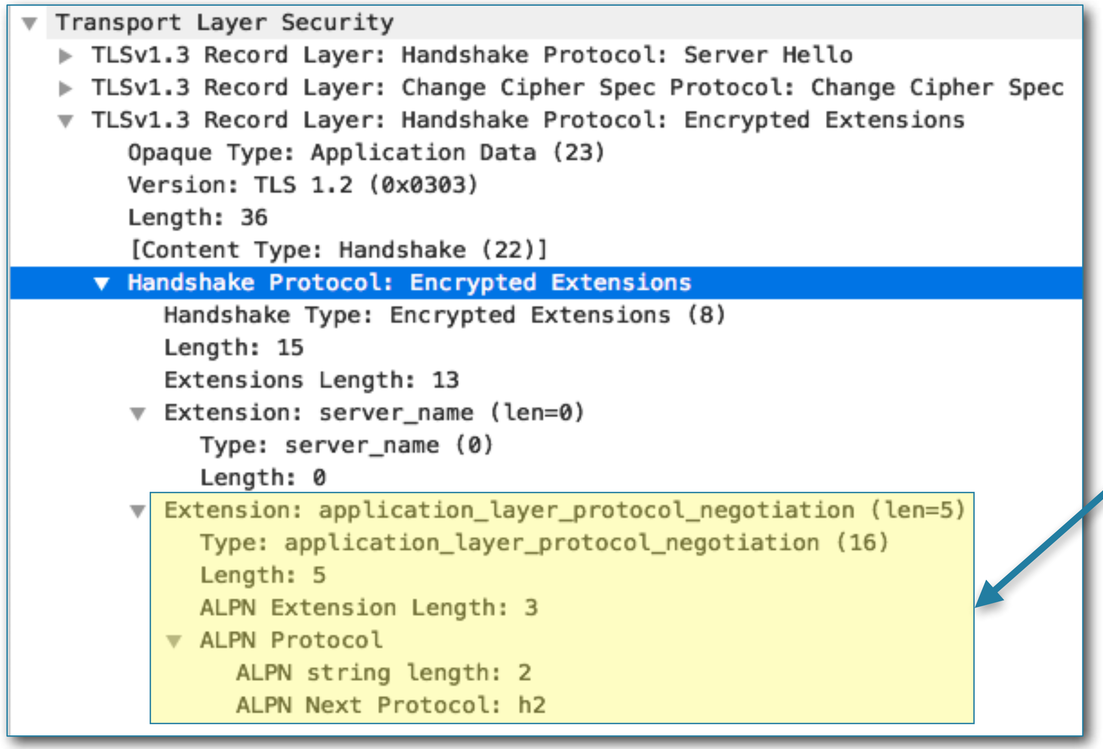
Chosen Group and server key share

Change Cipher Spec (server)



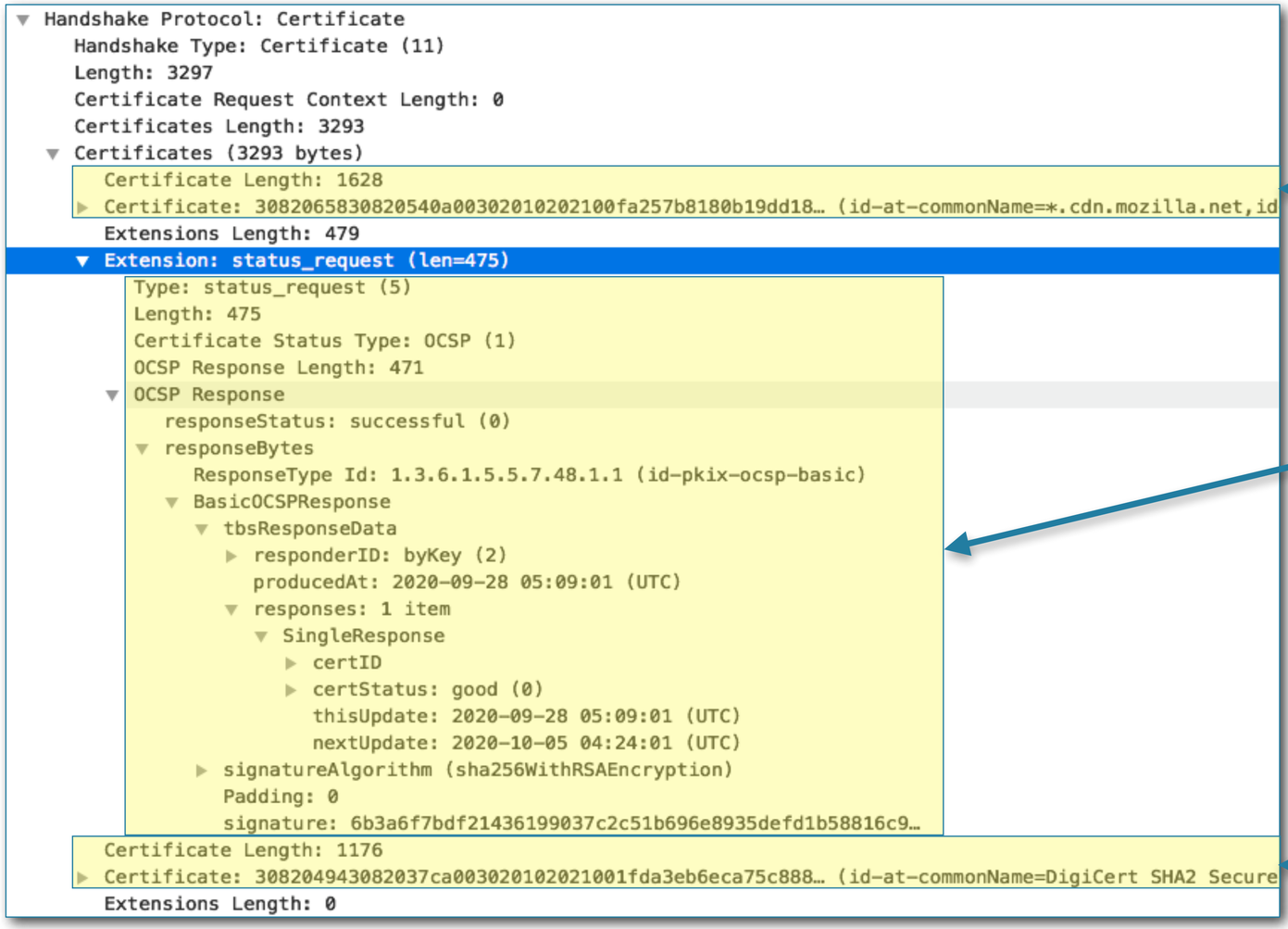
Dummy ChangeCipherSpec for middleboxes

EncryptedExtensions (server)



Application Layer Protocol Negotiation

Certificate (server)

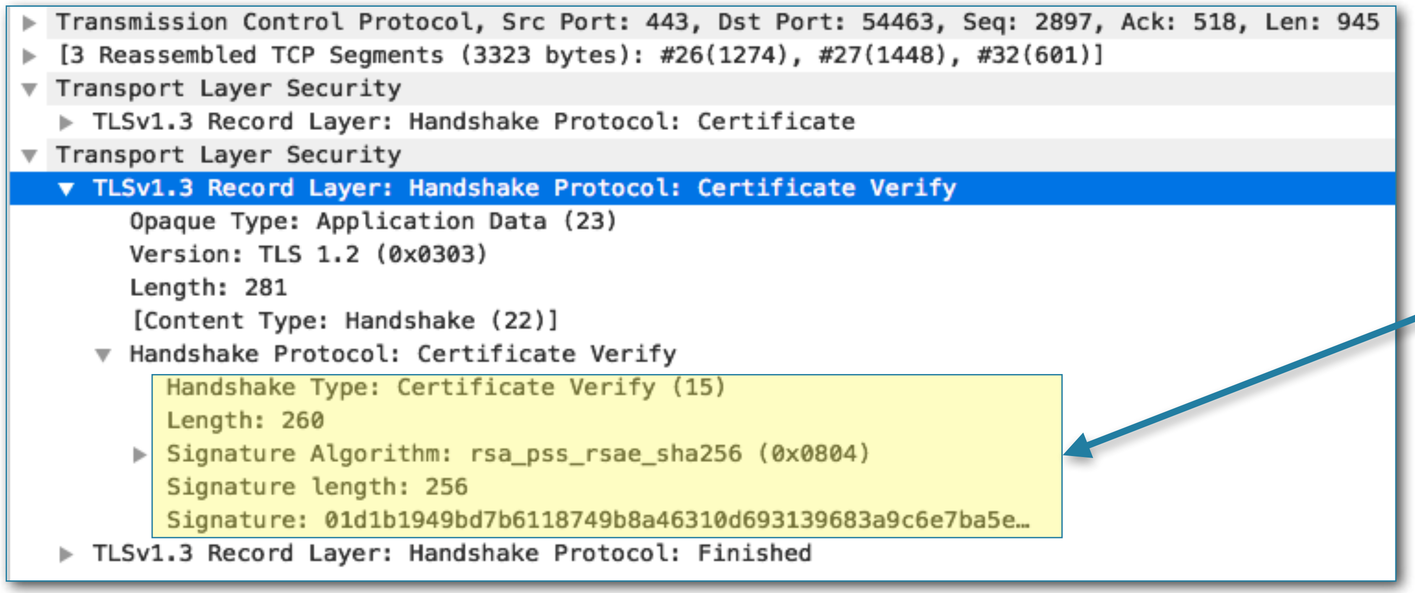


Sever Certificate

Certificate extensions, in this case OCSP

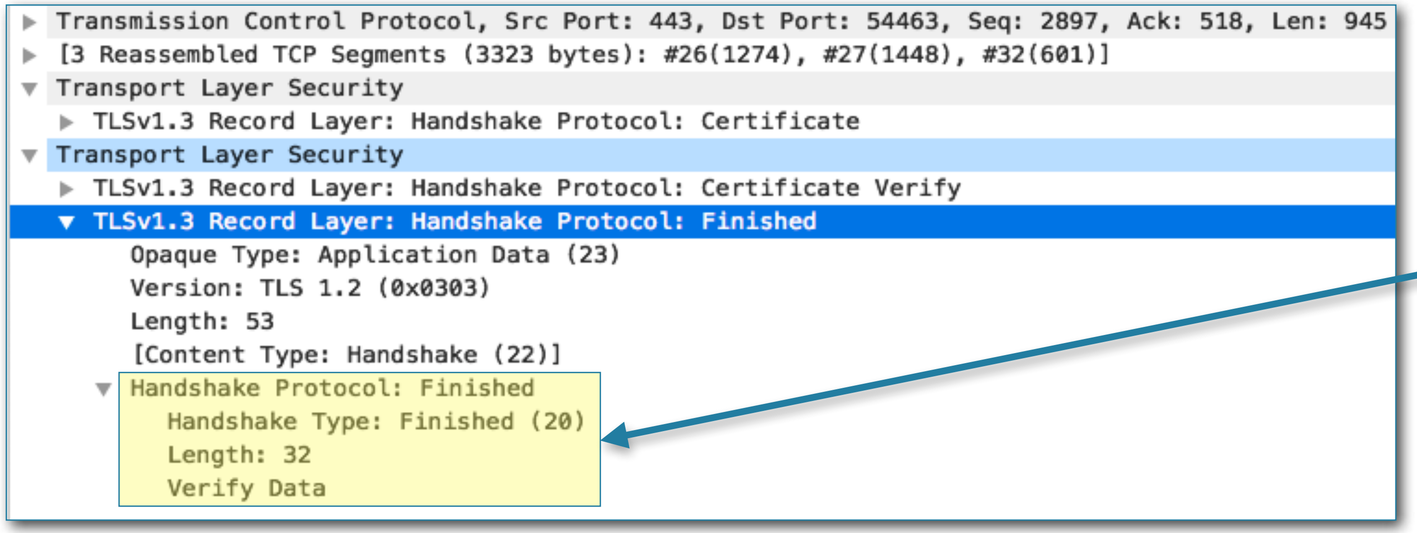
(Intermediate) CA Certificate

Certificate Verify (server)



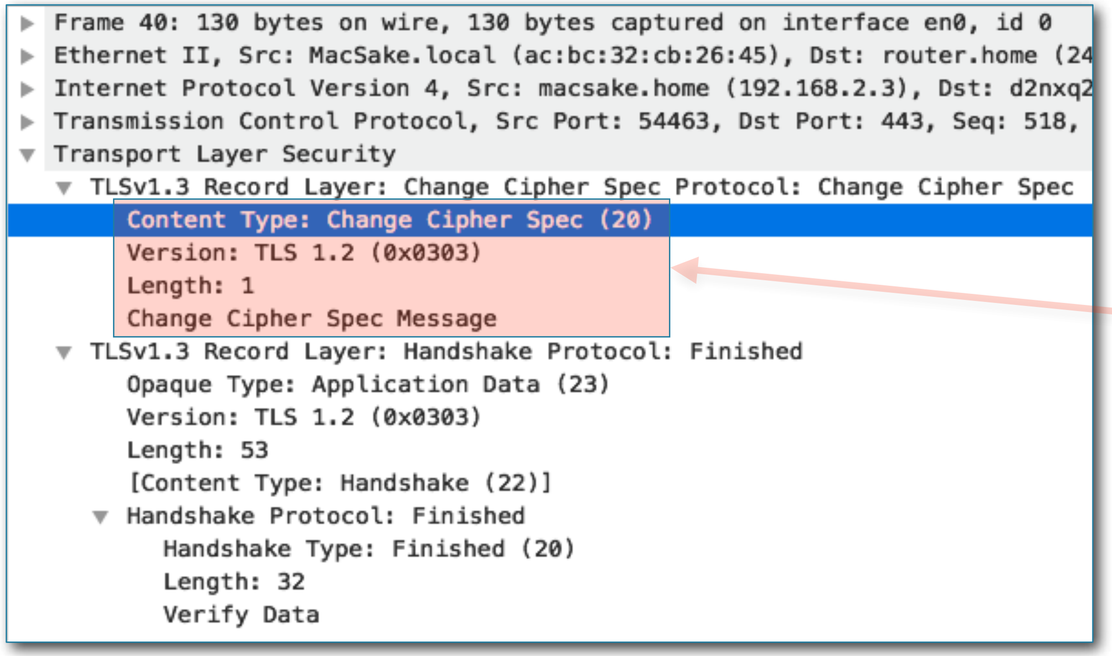
Certificate Verify proves possession of the private key of the server certificate

Finished (server)



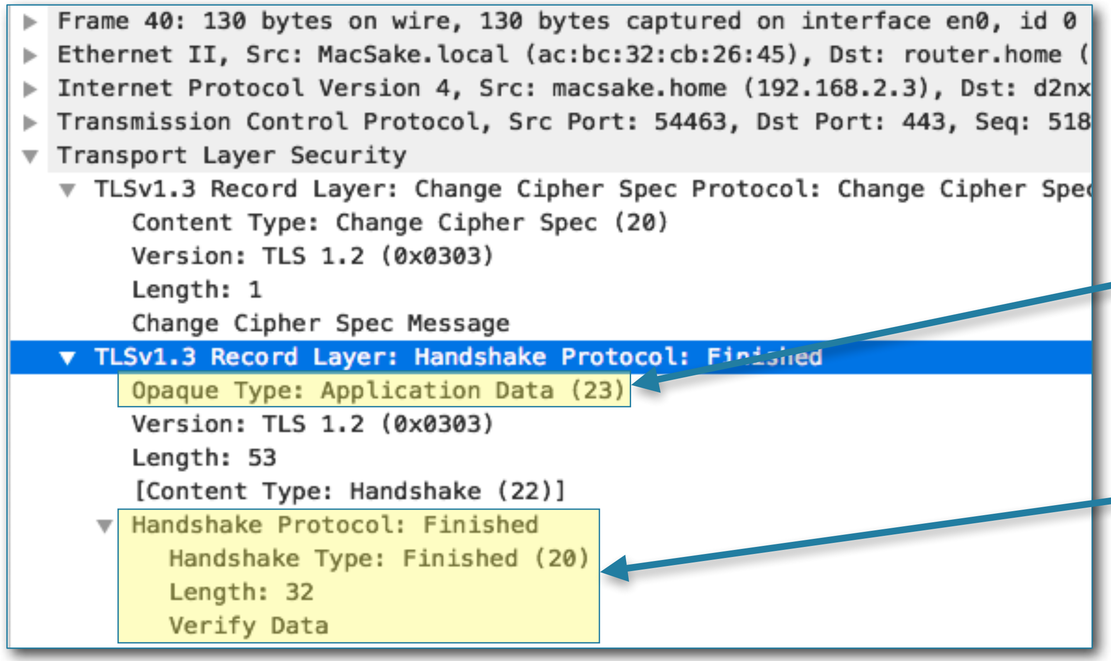
Finished message proves nothing in the handshake has been altered (mitm protection)

Change Cipher Spec (client)



Dummy ChangeCipherSpec for middleboxes

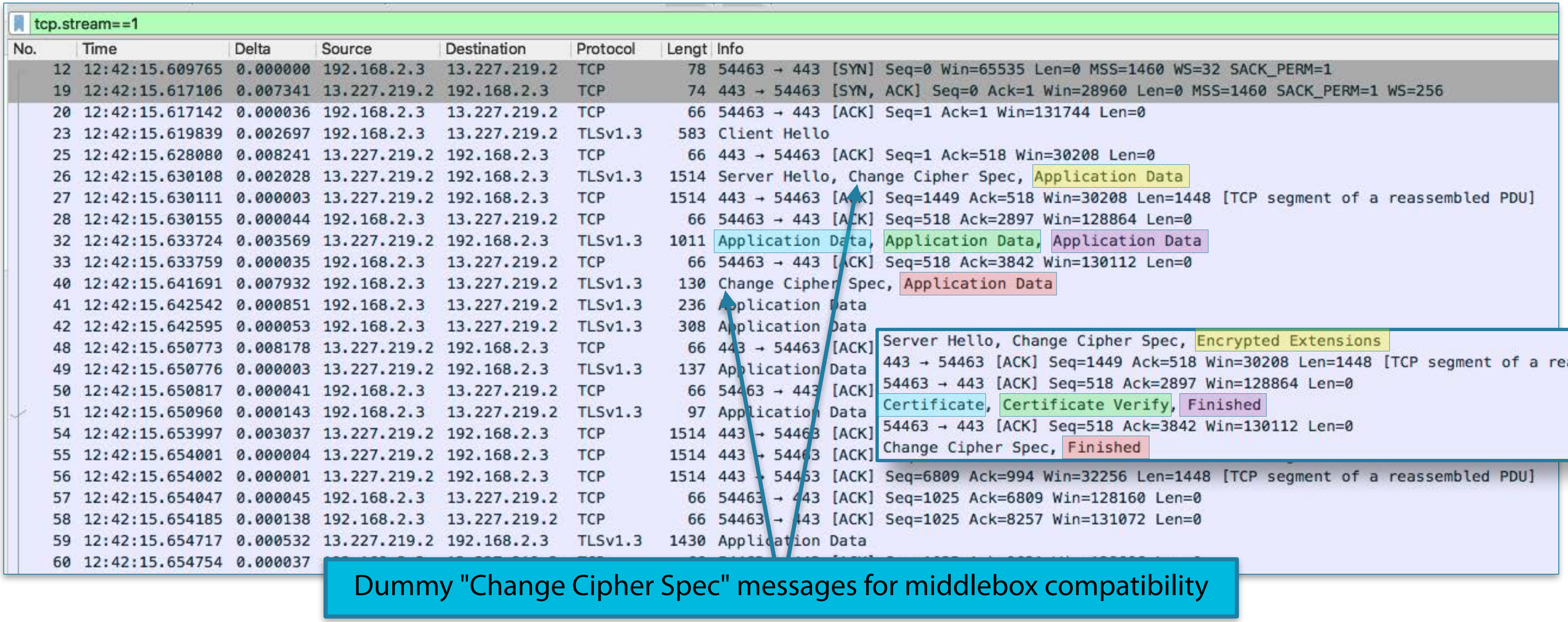
Finished (client)



Application Data instead of Handshake to please the middle boxes

Finished message proves nothing in the handshake has been altered (mitm protection)

TLSv1.3 handshake partially encryped



tcp.stream==1

No.	Time	Delta	Source	Destination	Protocol	Lengt	Info
12	12:42:15.609765	0.000000	192.168.2.3	13.227.219.2	TCP	78	54463 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 SACK_PERM=1
19	12:42:15.617106	0.007341	13.227.219.2	192.168.2.3	TCP	74	443 → 54463 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 WS=256
20	12:42:15.617142	0.000036	192.168.2.3	13.227.219.2	TCP	66	54463 → 443 [ACK] Seq=1 Ack=1 Win=131744 Len=0
23	12:42:15.619839	0.002697	192.168.2.3	13.227.219.2	TLSv1.3	583	Client Hello
25	12:42:15.628080	0.008241	13.227.219.2	192.168.2.3	TCP	66	443 → 54463 [ACK] Seq=1 Ack=518 Win=30208 Len=0
26	12:42:15.630108	0.002028	13.227.219.2	192.168.2.3	TLSv1.3	1514	Server Hello, Change Cipher Spec, Application Data
27	12:42:15.630111	0.000003	13.227.219.2	192.168.2.3	TCP	1514	443 → 54463 [ACK] Seq=1449 Ack=518 Win=30208 Len=1448 [TCP segment of a reassembled PDU]
28	12:42:15.630155	0.000044	192.168.2.3	13.227.219.2	TCP	66	54463 → 443 [ACK] Seq=518 Ack=2897 Win=128864 Len=0
32	12:42:15.633724	0.003569	13.227.219.2	192.168.2.3	TLSv1.3	1011	Application Data, Application Data, Application Data
33	12:42:15.633759	0.000035	192.168.2.3	13.227.219.2	TCP	66	54463 → 443 [ACK] Seq=518 Ack=3842 Win=130112 Len=0
40	12:42:15.641691	0.007932	192.168.2.3	13.227.219.2	TLSv1.3	130	Change Cipher Spec, Application Data
41	12:42:15.642542	0.000851	192.168.2.3	13.227.219.2	TLSv1.3	236	Application Data
42	12:42:15.642595	0.000053	192.168.2.3	13.227.219.2	TLSv1.3	308	Application Data
48	12:42:15.650773	0.008178	13.227.219.2	192.168.2.3	TCP	66	443 → 54463 [ACK]
49	12:42:15.650776	0.000003	13.227.219.2	192.168.2.3	TLSv1.3	137	Application Data
50	12:42:15.650817	0.000041	192.168.2.3	13.227.219.2	TCP	66	54463 → 443 [ACK]
51	12:42:15.650960	0.000143	192.168.2.3	13.227.219.2	TLSv1.3	97	Application Data
54	12:42:15.653997	0.003037	13.227.219.2	192.168.2.3	TCP	1514	443 → 54463 [ACK]
55	12:42:15.654001	0.000004	13.227.219.2	192.168.2.3	TCP	1514	443 → 54463 [ACK]
56	12:42:15.654002	0.000001	13.227.219.2	192.168.2.3	TCP	1514	443 → 54463 [ACK] Seq=6809 Ack=994 Win=32256 Len=1448 [TCP segment of a reassembled PDU]
57	12:42:15.654047	0.000045	192.168.2.3	13.227.219.2	TCP	66	54463 → 443 [ACK] Seq=1025 Ack=6809 Win=128160 Len=0
58	12:42:15.654185	0.000138	192.168.2.3	13.227.219.2	TCP	66	54463 → 443 [ACK] Seq=1025 Ack=8257 Win=131072 Len=0
59	12:42:15.654717	0.000532	13.227.219.2	192.168.2.3	TLSv1.3	1430	Application Data
60	12:42:15.654754	0.000037					

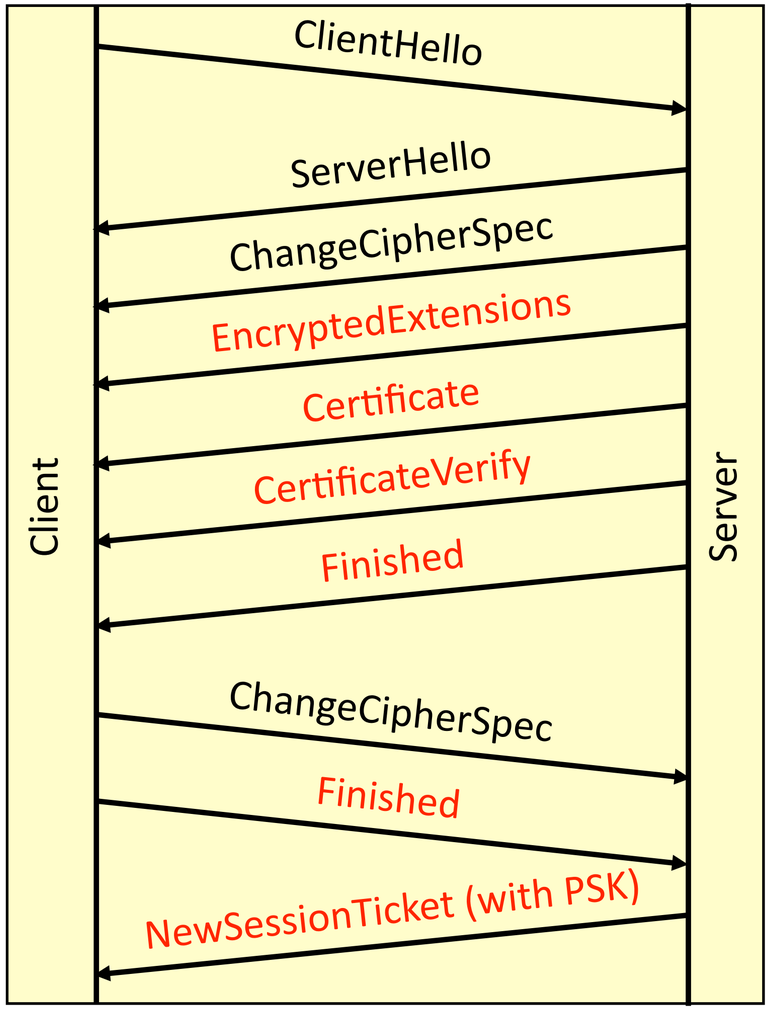
Server Hello, Change Cipher Spec, Encrypted Extensions
443 → 54463 [ACK] Seq=1449 Ack=518 Win=30208 Len=1448 [TCP segment of a rea
54463 → 443 [ACK] Seq=518 Ack=2897 Win=128864 Len=0
Certificate, Certificate Verify, Finished
54463 → 443 [ACK] Seq=518 Ack=3842 Win=130112 Len=0
Change Cipher Spec, Finished

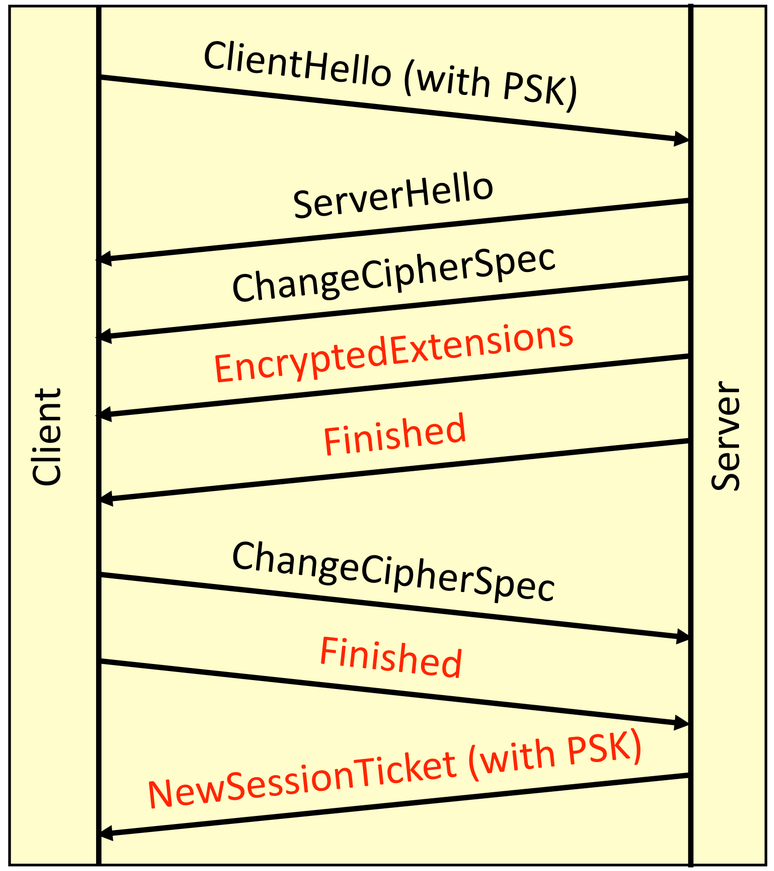
Dummy "Change Cipher Spec" messages for middlebox compatibility

TLSv1.3 session resumption

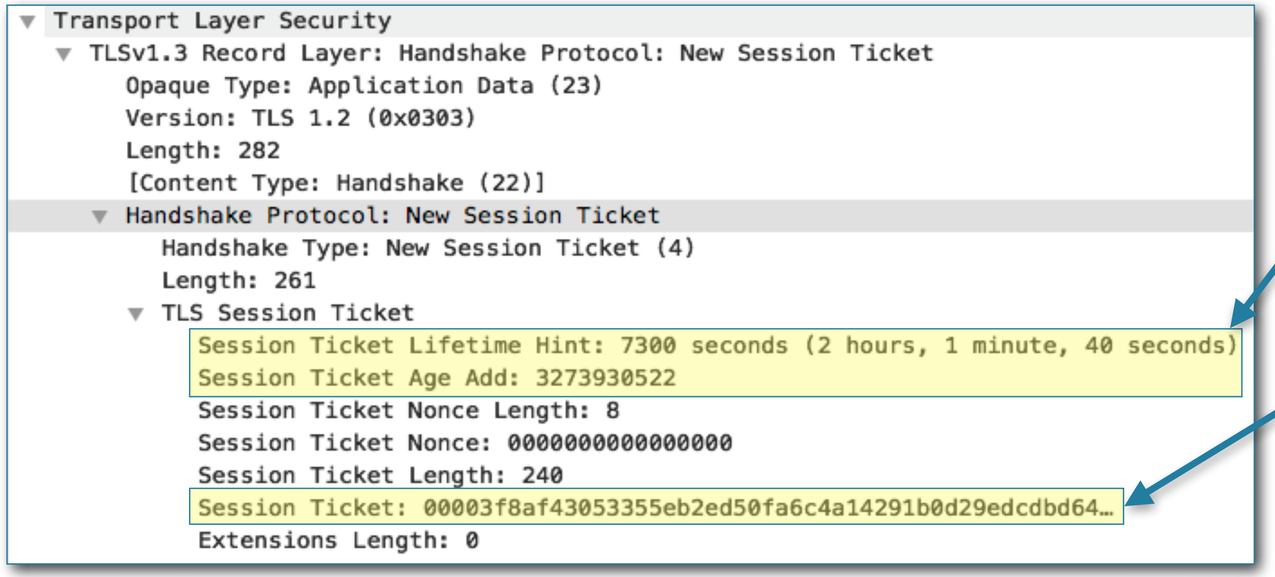
- Server creates PSK and either
  - stores it and sends the pointer as the PSK Identity
  - encrypts and sends the PSK as the PSK Identity
- TLS extension in ClientHello and ServerHello
- Reuse TLS HandshakeType "NewSessionTicket"
  - to send the PSK Identity
  - More than one can be sent
  - Can be implemented as one-time-tickets

TLS session Tickets (2)





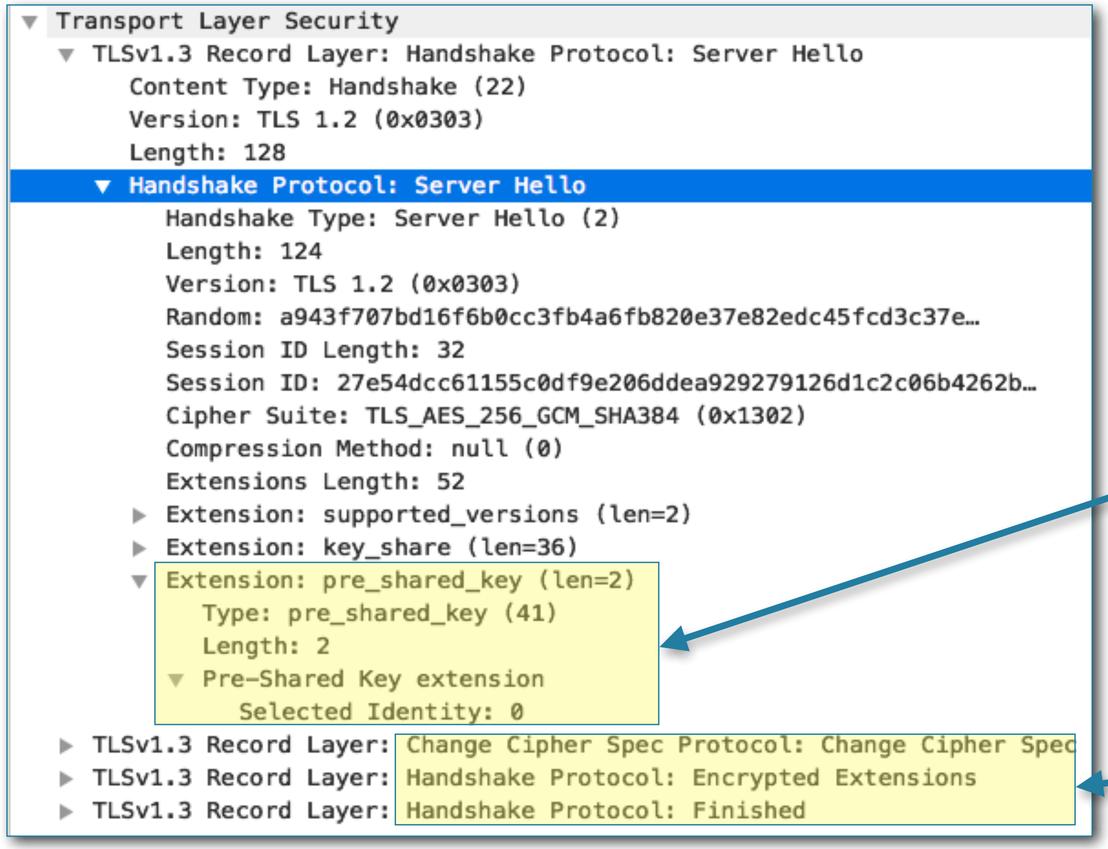
New Session Ticket (server)



Lifetime for the PSK (and offset)

PSK identity for session resumption

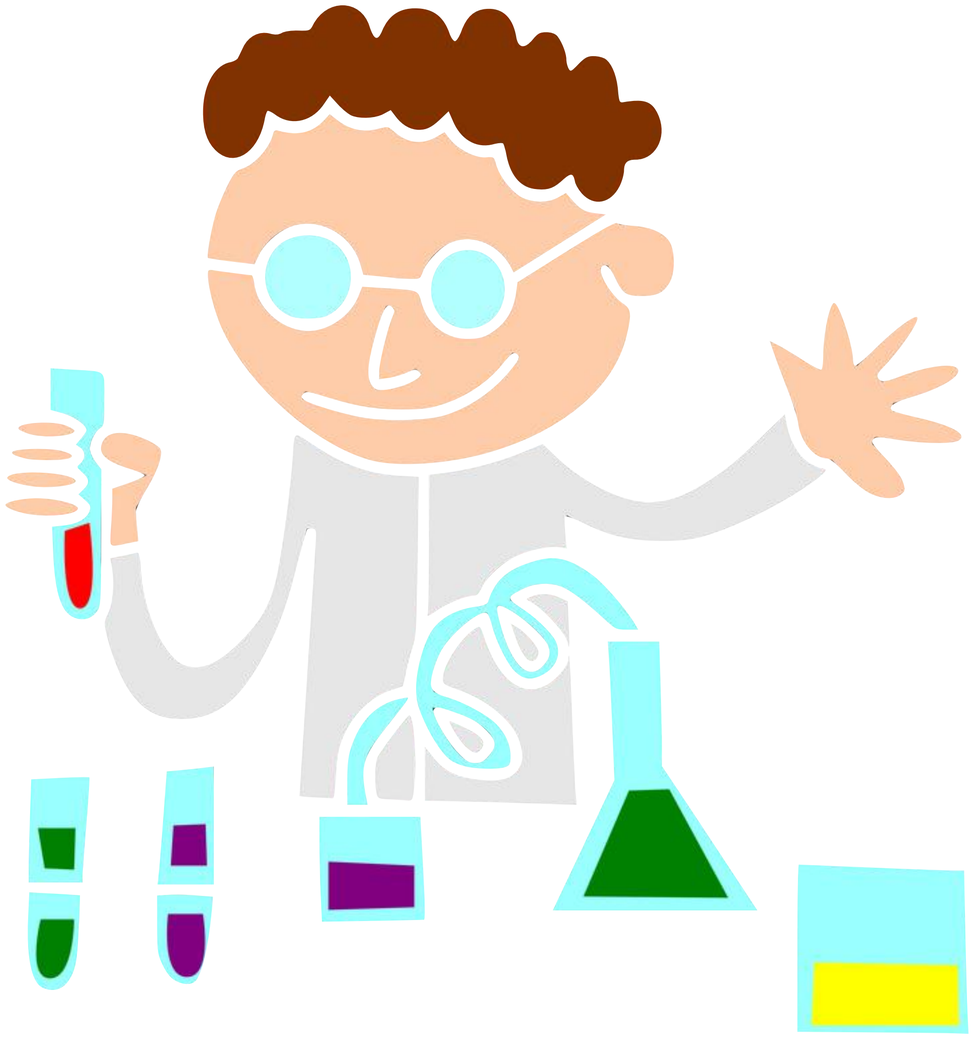
ServerHello (server)



Selected PSK Identity

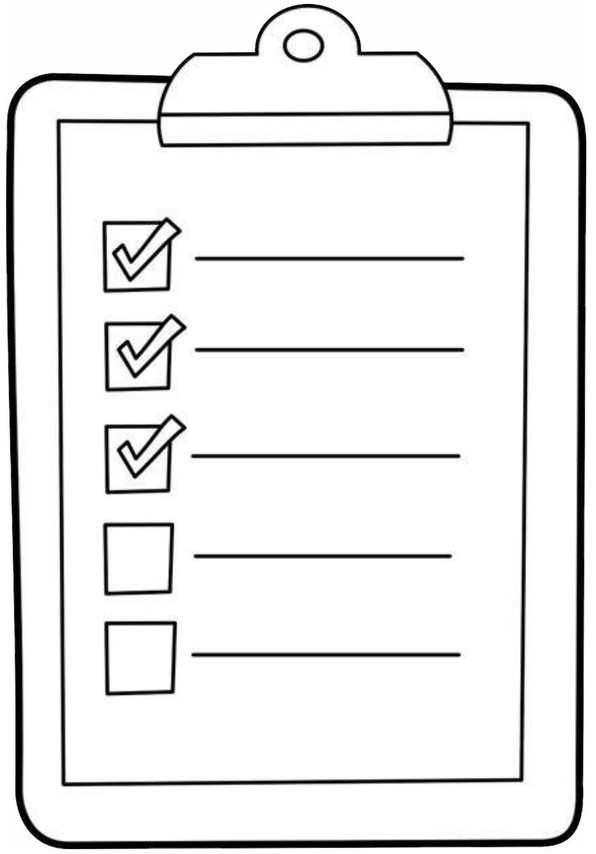
No more Certificate and Certificate Verify

LAB 2



Agenda

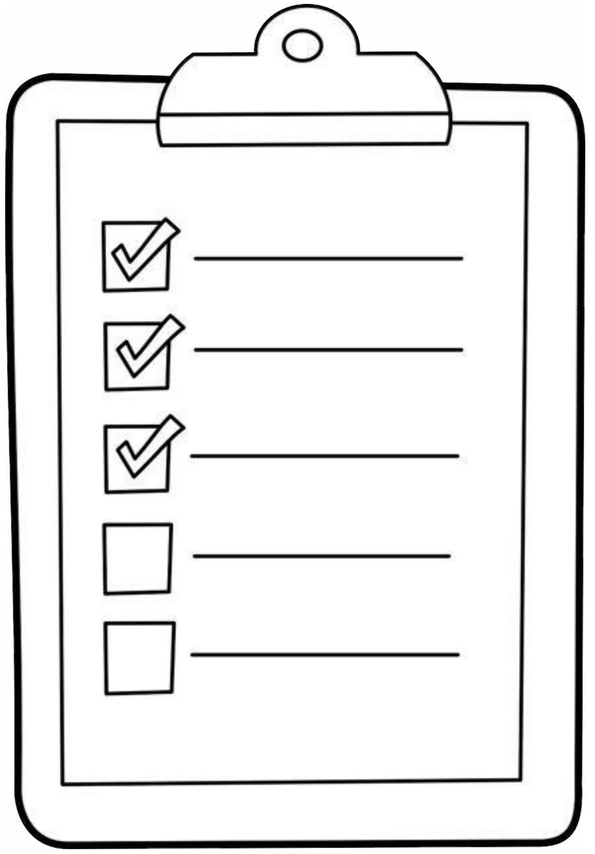
- TLS fundamentals
  - The need for TLS (and what happend to SSL?)
  - Cryptology 101 & PKI
  - Troubleshoot the TLS handshake (SSLv3 - TLSv1.2)
  - Troubleshoot the TLS handshake (TLSv1.3)
- **Analysing Application Data**
  - Without decryption
  - With decryption using the servers private key
  - With decryption using TLS session keys
- Creating configuration profiles for TLS analysis
- SSL/TLS vulnerabilities & privacy concerns
- Summary and Q&A



https://www.needpix.com/photo/download/94588/checklist-action-check-list-verification-tick-approved-okay-check-mark

Agenda

- TLS fundamentals
  - The need for TLS (and what happend to SSL?)
  - Cryptology 101 & PKI
  - Troubleshoot the TLS handshake (SSLv3 - TLSv1.2)
  - Troubleshoot the TLS handshake (TLSv1.3)
- Analysing Application Data
  - **Without decryption**
  - With decryption using the servers private key
  - With decryption using TLS session keys
- Creating configuration profiles for TLS analysis
- SSL/TLS vulnerabilities & privacy concerns
- Summary and Q&A



https://www.needpix.com/photo/download/94588/checklist-action-check-list-verification-tick-approved-okay-check-mark

Analyzing encrypted TLS data

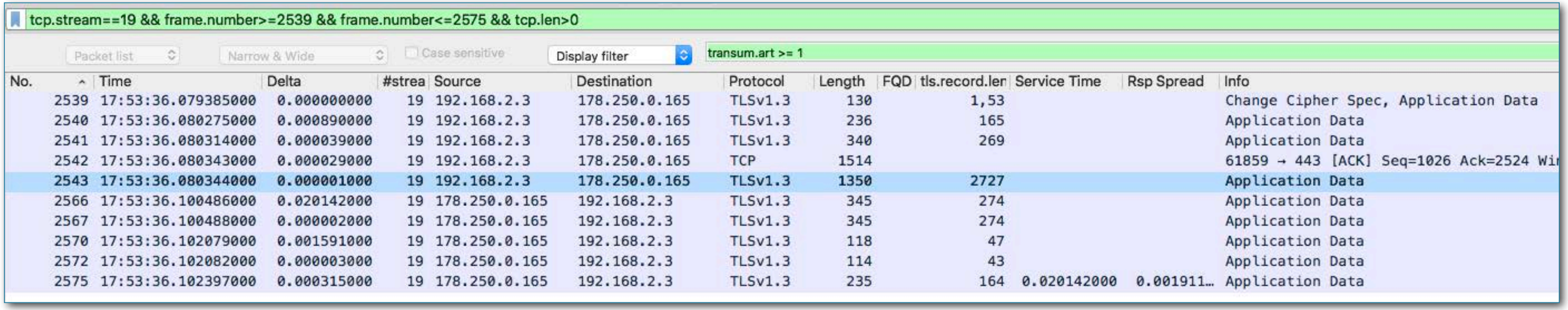
- Filter on application data with:
  tls.record.content_type == 23
- Add an extra custom column with:
  tls.record.length
- Look at the request / response pattern
  - check timestamps between c->s and s->c packets
  - check the ssl record lengths for an indication of the request/response sizes (actual data is less because of TLS overhead)
- Use TRANSUM dissector

Analyzing encrypted SSL data

Filter: ssl.record.content_type == 23 Expression... Clear Apply Save

No.	Time	Delta	Source	Destination	Protocol	Length	ssl.len	Info
11	0.040173	0.000000	192.168.3.1	192.168.3.3	TLSv1	491	432	Application Data
12	0.042446	0.002273	192.168.3.3	192.168.3.1	TLSv1	496	384,48	Application Data, Application Data
14	12.494568	12.452122	192.168.3.1	192.168.3.3	TLSv1	491	432	Application Data
15	12.495834	0.001266	192.168.3.3	192.168.3.1	TLSv1	496	384,48	Application Data, Application Data
29	39.717354	27.221520	192.168.3.1	192.168.3.3	TLSv1	550	1,48,432	Change Cipher Spec, Encrypted Handshake Message, Application Data
30	39.720262	0.002908	192.168.3.3	192.168.3.1	TLSv1	496	384,48	Application Data, Application Data
48	111.230987	71.510725	192.168.3.1	192.168.3.3	TLSv1	491	432	Application Data
49	111.233419	0.002432	192.168.3.3	192.168.3.1	TLSv1	496	384,48	Application Data, Application Data

TRANSUM



tcp.stream==19 && frame.number>=2539 && frame.number<=2575 && tcp.len>0

Packet list | Narrow & Wide | Case sensitive | Display filter | transum.art >= 1

No.	Time	Delta	#strea	Source	Destination	Protocol	Length	FQD	tls.record.len	Service Time	Rsp Spread	Info
2539	17:53:36.079385000	0.000000000	19	192.168.2.3	178.250.0.165	TLSv1.3	130		1,53			Change Cipher Spec, Application Data
2540	17:53:36.080275000	0.000890000	19	192.168.2.3	178.250.0.165	TLSv1.3	236		165			Application Data
2541	17:53:36.080314000	0.000039000	19	192.168.2.3	178.250.0.165	TLSv1.3	340		269			Application Data
2542	17:53:36.080343000	0.000029000	19	192.168.2.3	178.250.0.165	TCP	1514					61859 → 443 [ACK] Seq=1026 Ack=2524 Wi
2543	17:53:36.080344000	0.000001000	19	192.168.2.3	178.250.0.165	TLSv1.3	1350		2727			Application Data
2566	17:53:36.100486000	0.020142000	19	178.250.0.165	192.168.2.3	TLSv1.3	345		274			Application Data
2567	17:53:36.100488000	0.000002000	19	178.250.0.165	192.168.2.3	TLSv1.3	345		274			Application Data
2570	17:53:36.102079000	0.001591000	19	178.250.0.165	192.168.2.3	TLSv1.3	118		47			Application Data
2572	17:53:36.102082000	0.000003000	19	178.250.0.165	192.168.2.3	TLSv1.3	114		43			Application Data
2575	17:53:36.102397000	0.000315000	19	178.250.0.165	192.168.2.3	TLSv1.3	235		164	0.020142000	0.001911…	Application Data

```
▶ Frame 2575: 235 bytes on wire, 235 bytes captured on interface en0, id 0
▶ Ethernet II, Src: zte_be:f5:f0 (24:58:6e:be:f5:f0), Dst: Apple_cb:26:45 (ac:bc:32:cb:26:45)
▶ Internet Protocol Version 4, Src: 178.250.0.165, Dst: 192.168.2.3
▶ Transmission Control Protocol, Src Port: 443, Dst Port: 61859, Seq: 3182, Ack: 3758, Len: 169
▶ Transport Layer Security
▼ TRANSUM RTE Data
    [RTE Status: OK]
    [Req First Seg: 2539]
    [Req Last Seg: 2543]
    [Rsp First Seg: 2566]
    [Rsp Last Seg: 2575]
    [APDU Rsp Time: 0.023012000 seconds]
    [Service Time: 0.020142000 seconds]
    [Req Spread: 0.000959000 seconds]
    [Rsp Spread: 0.001911000 seconds]
    [Trace clip filter: tcp.stream==19 && frame.number>=2539 && frame.number<=2575 && tcp.len>0]
    [Calculation: Generic TCP]
```

TPNCP
TRANSUM
TSDNS
TSP
TTE
TURNCHANNEL
TUXEDO
TZSP
UA3G
UASIP
UAUDP
UBDP
UBERTOOTH
UCP
UDP

TRANSUM RTE Data

Capture position: Client

- [x] Subdissector reassembly enabled

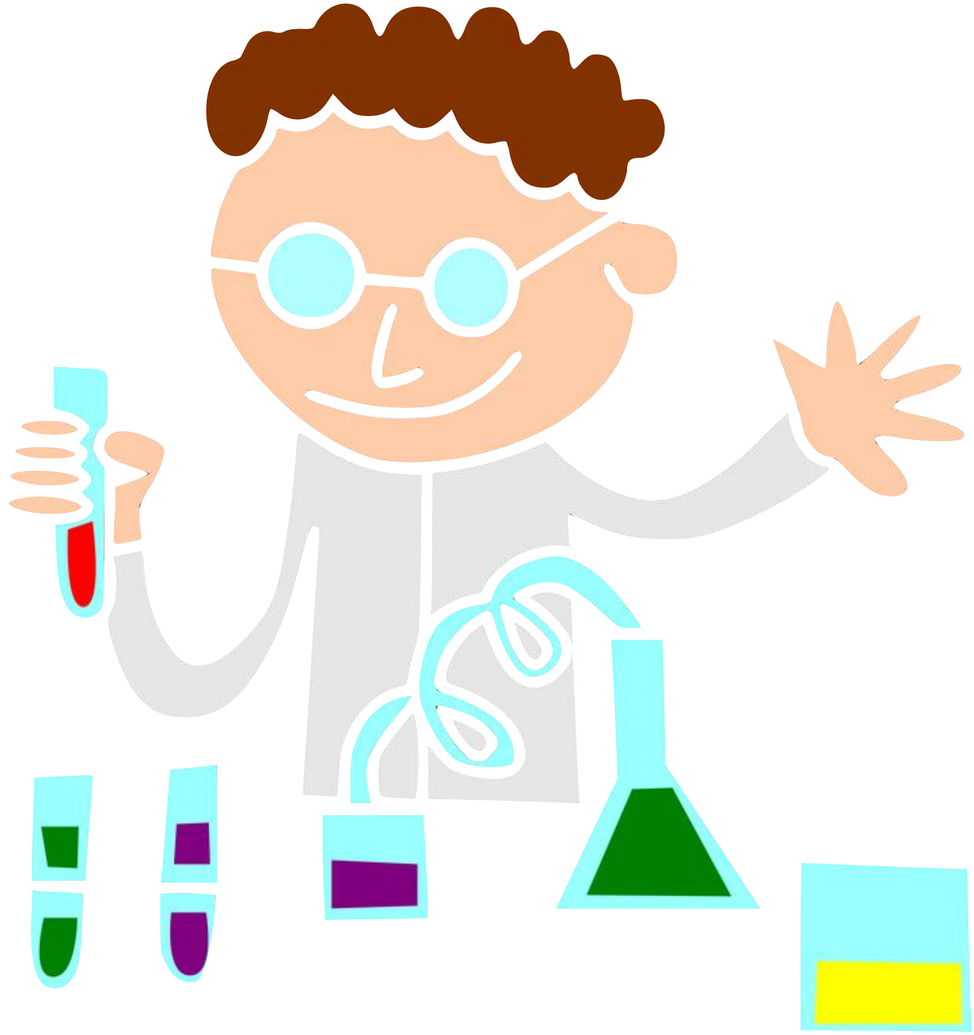
Output RTE data for these TCP service ports: 25,80,443,1433,50001

Output RTE data for these UDP service ports: 137-139

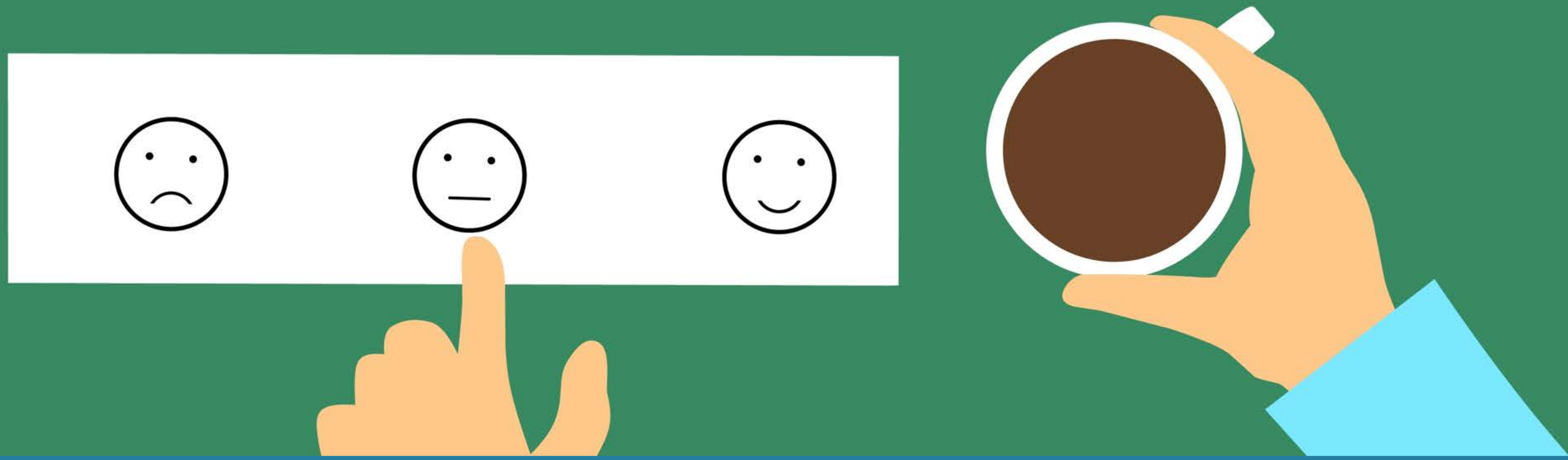
- [ ] Discard orphaned TCP Keep-Alives
- [ ] Add RTE data to the first request segment
- [ ] Add RTE data to the last request segment
- [ ] Add RTE data to the first response segment
- [x] Add RTE data to the last response segment
- [ ] Enable debug info

Demo & Exercise

- Demo
  - File: dv-ov-ev-wildcard.pcapng
- Exercise
  - use filter "tcp.port==61945 && tcp.len>0"
  - How many HTTP requests/responses?
  - What was the "size" of the last request?
  - What was the "size" of the last response?

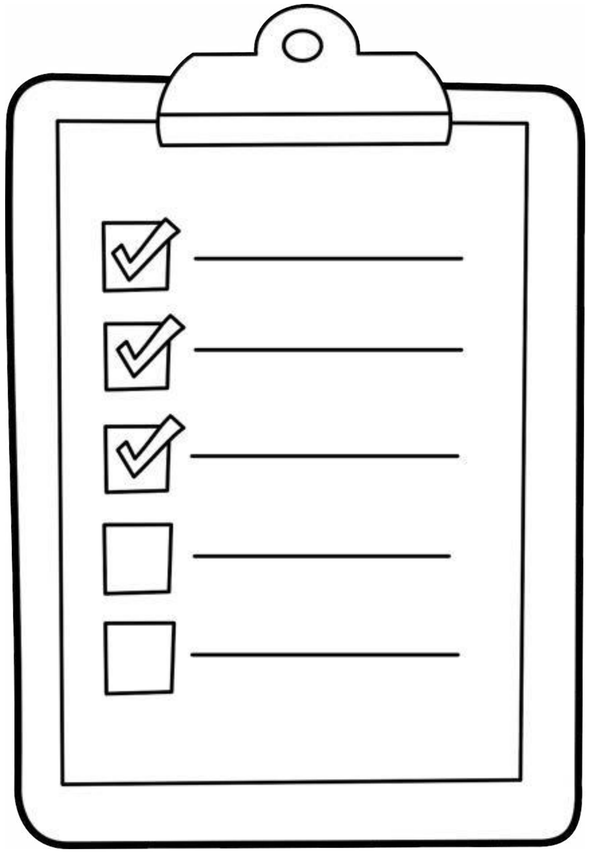


Time for a poll



Agenda

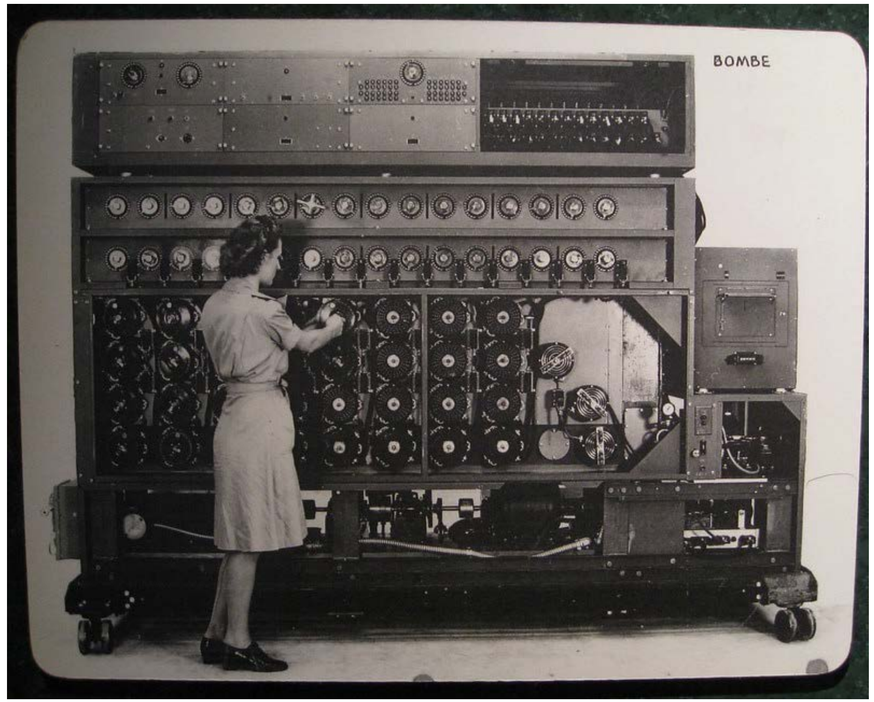
- TLS fundamentals
  - The need for TLS (and what happend to SSL?)
  - Cryptology 101 & PKI
  - Troubleshoot the TLS handshake (SSLv3 - TLSv1.2)
  - Troubleshoot the TLS handshake (TLSv1.3)
- Analysing Application Data
  - Without decryption
  - **With decryption using the servers private key**
  - With decryption using TLS session keys
- Creating configuration profiles for TLS analysis
- SSL/TLS vulnerabilities & privacy concerns
- Summary and Q&A



https://www.needpix.com/photo/download/94588/checklist-action-check-list-verification-tick-approved-okay-check-mark

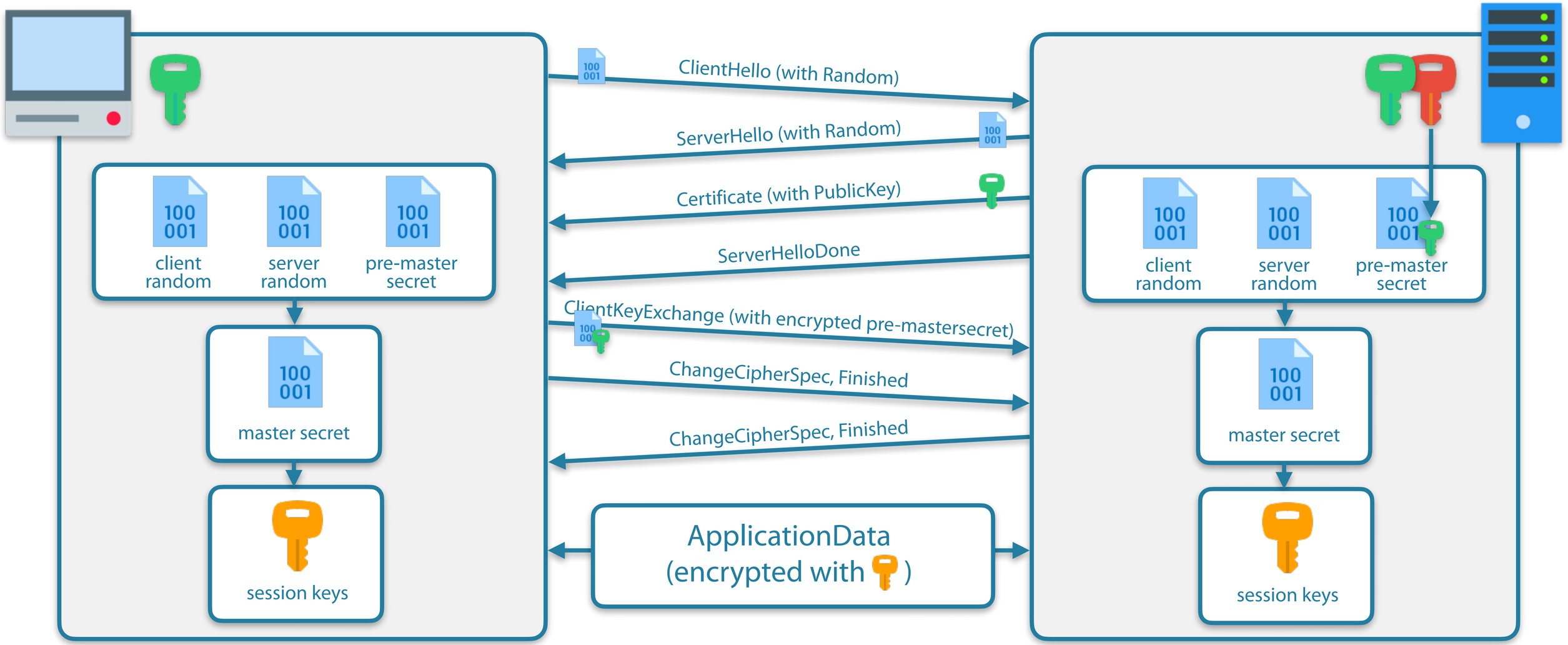
Why TLS decryption?

- Application Performance Monitoring
  - Which requests were slow???
- Troubleshooting
  - Why is this not working???
- Security
  - Malware/virus/etc. detection
    - Some level of detection by behavioural analysis or fingerprinting (JA3, JA3S, nDPIv3,etc)
  - Dataloss detection and prevention
    - GDPR
  - Forensics



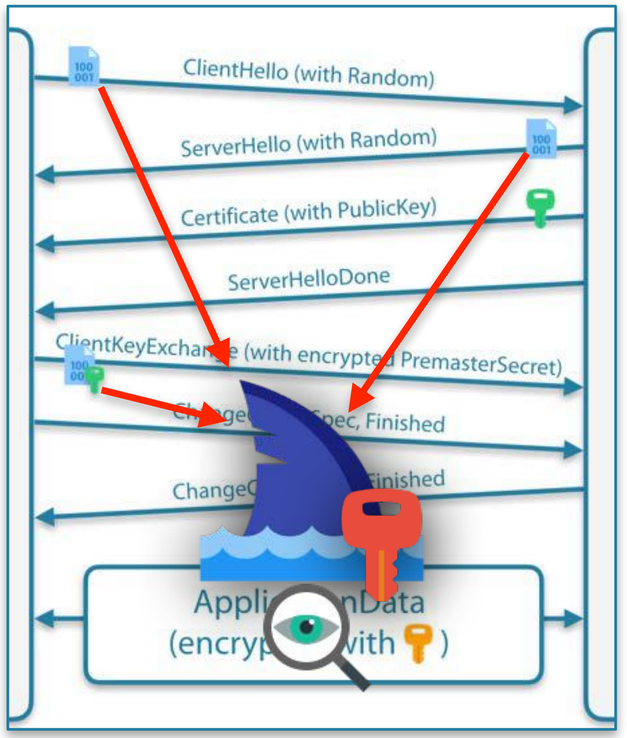
https://www.flickr.com/photos/brewbooks/3318600273

RSA key exchange



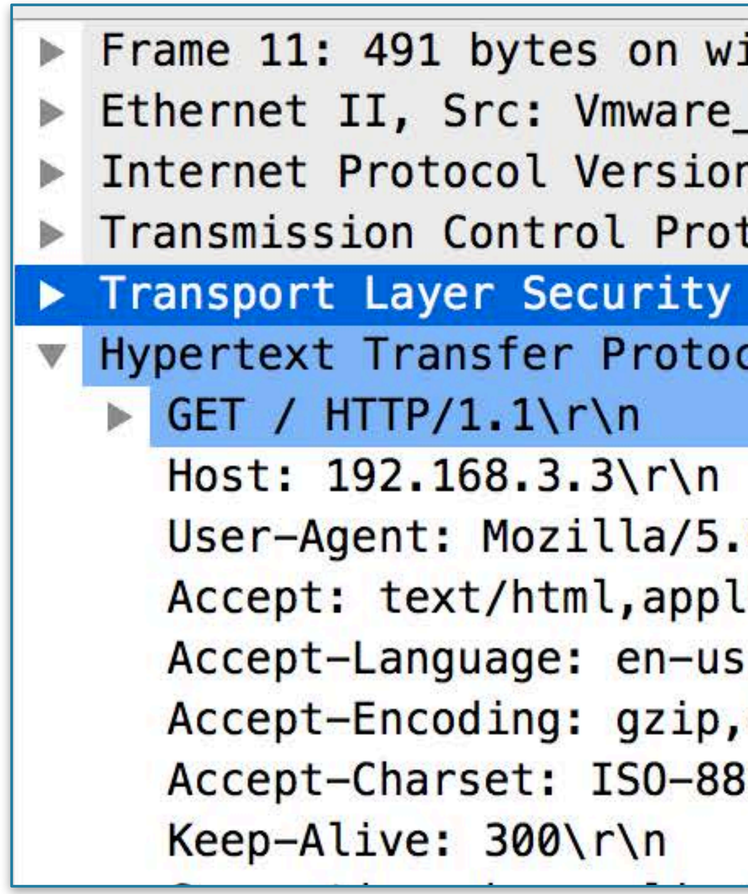
Decryption Recipe I

- Client Random
  - Captured in the **ClientHello**
- Server Random
  - Captured in the **ServerHello**
- Pre-master Secret
  - Captured in the **ClientKeyExchange** but encrypted with server public key
  - Use server **private key** to decrypt
- Calculate Master Secret
- Calculate Session Keys
- Decrypt application traffic with Session Keys

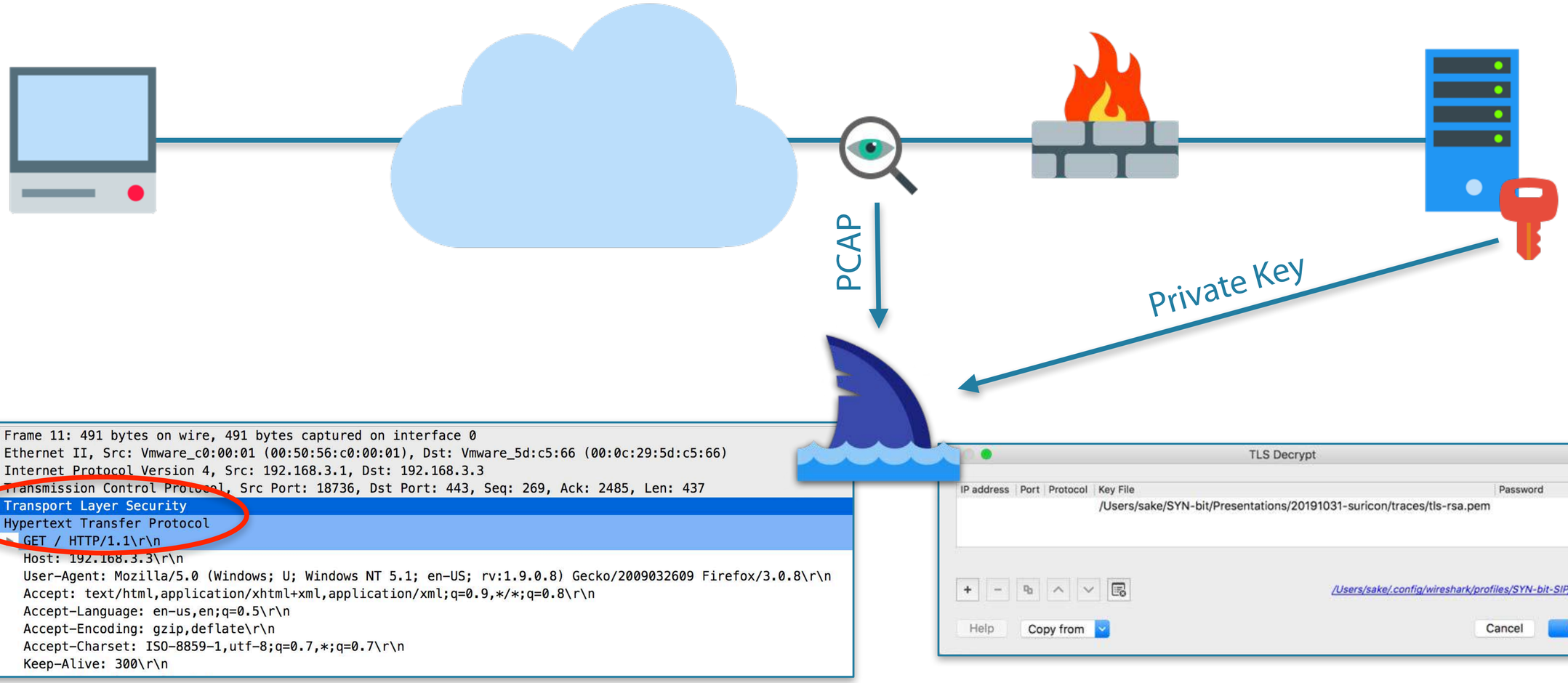


Decryption in Wireshark I

- Start capturing **before** opening the browser
  - Otherwise you will miss the full TLS handshake
- Provide the (right?!) server private key
  - PKCS#1 in PEM format without passphrase
    - Convert format and/or strip passphrase with OpenSSL if needed
  - PKCS#12 format (optionally with passphrase)
  - Connect to NetHSM (some support recently added)
- Want to try yourself?
  - Go to: https://www.cloudshark.org/captures/a25364a5f672
  - Click on More Info -> Download
  - And look a the capture file comments for instructions



TLS Decryption with private key



Accessing the private key

- You should **NOT** be able to get the private key
  - ...unless it is a dev/test/acc environment
- Out-of-luck with HSM cards
- You could interface with NetHSM devices
  - Limited support added to Wireshark added recently
- Protect the private key with your life!
  - **Do you really want to have it on your laptop???**

```
# cat private.pem
-----BEGIN RSA PRIVATE KEY-----
MIICXgIBAAKBgQDrHdbb+yGE6m6EZ03bXURpZCjch2H6g97ZAkJVGrjLZFfettBA
EYa8vYYxWsf8KBpEZeksSCsDA9MnU2H6QDjzqdOnaSWfeXMAr4OsCOpauStpreq7
q1hk8iOqy+f4KijRrhWplh1QW1A8gtSIg137pyUhW+WsfwxKwmzjGIC1SwIDAQAB
AoGBAMneA9U6KIxjb+JUg/99c7h9W6wEvTYHNTXjf6psWA+hpuQ82E65/ZJdszL6
+8vfbrYdPfdcOznKdehE61GgBIRjhbOOeOxTxfbt1OWcVeqW17t52QiZbaK/dt/j
Fn8RyixrU4NSSPWG7NCyLJwYU2lgstUbhHg2Hz9Mz59GgS2xAkEA+fXI/b65qF29
/OAz8iHrmGU/qmwrP2+I+gNB1ji2bIPfXLg/V3Q70golpeRhcORB1SbMYP8+PCU5
5NsC1iRGuQJBAPDMO/w8Rzl4waw885c3DF1Iar81H3VCBoswUqPBDdQvtSatMq1c
BA27q5Re+XewBGx4AbP84lPPsDD7aX8eWiMCQQCEUfVdRgq4My+w3usAwa4bFXYX
fH2EbkG/v9upUIpZdZHXXn3BiPll3hNB910RyvOCp7BHpLbIVhiIqtucisWZAkAs
b6QKMh16r5wd6smQ+CmhOEnqqyT5AIwwl2RIr9GbfIpTbtbRQw/EcQOCx9wFiEfo
tGSsEFi72rHK+DpJqRI9AkEA72gdyXRgPfGOS3rfQ3DBcImBQvDSCBa4cuU1XJ1/
MO93a8v9Vj87/yDm4xsBDsoz2PyBepawHVlIvZ6jDD0aXw==
-----END RSA PRIVATE KEY-----
#
```

Converting keys

Removing passphrase:

```
root@mgmt# openssl rsa -in encrypted.key -out cleartext.key
Enter pass phrase for encrypted.key: <passphrase>
writing RSA key
root@mgmt#
```

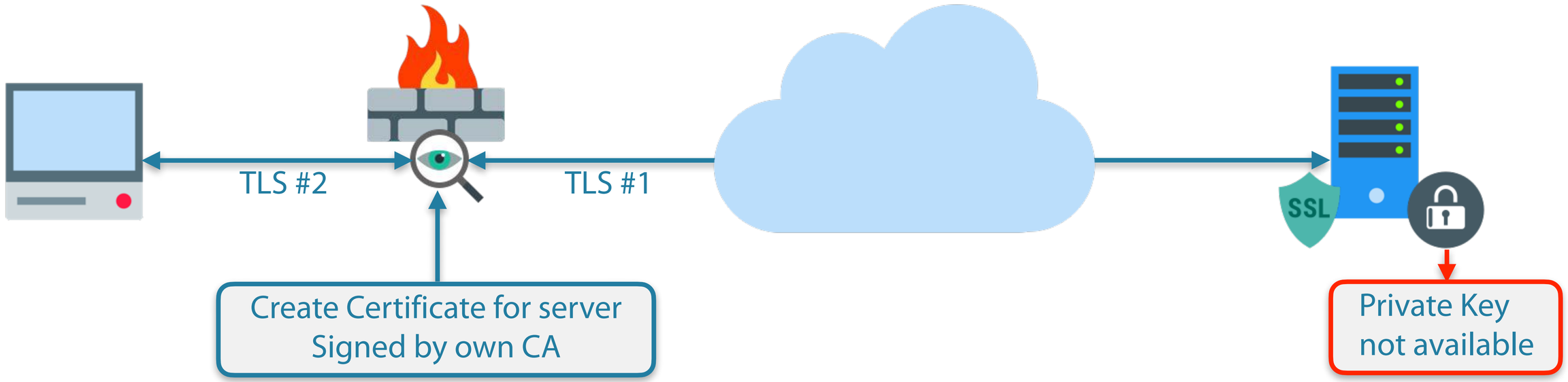
Converting from DER to PEM (and removing passphrase):

```
root@mgmt# openssl rsa -inform DER -in der.key -out pem.key
Enter pass phrase for encrypted.key: <passphrase>
writing RSA key
root@mgmt#
```

Converting from PEM to PKCS12 (and adding passphrase):

```
root@mgmt# openssl pkcs12 -in pem.cert -inkey pem.key -export -out cert.pkcs12
Enter Export Password: <new-passphrase>
Verifying - Enter Export Password: <new-passphrase>
root@mgmt#
```

Active (MITM) Decryption



Caveats:

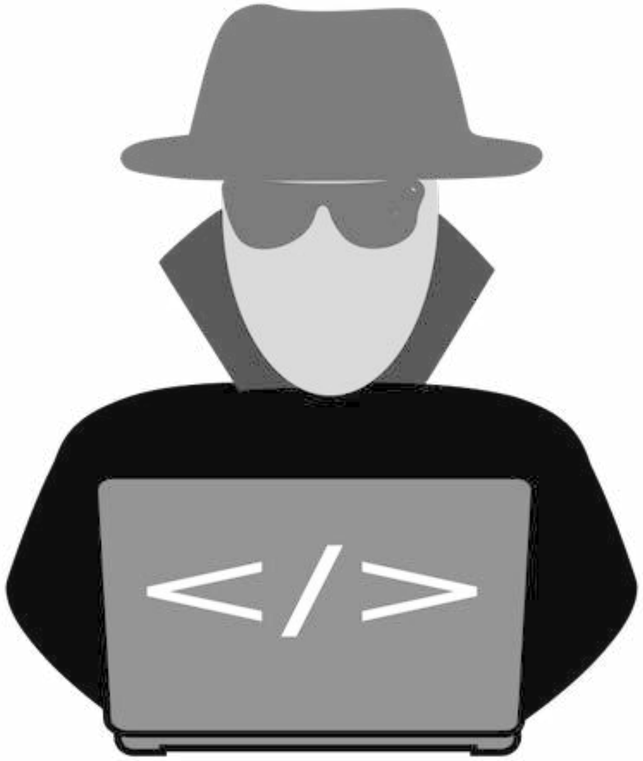
Client needs to have the RootCA of the Firewall in it's trust store

Certificate pinning will fail

Mutual TLS is not possible

Some MITM solutions

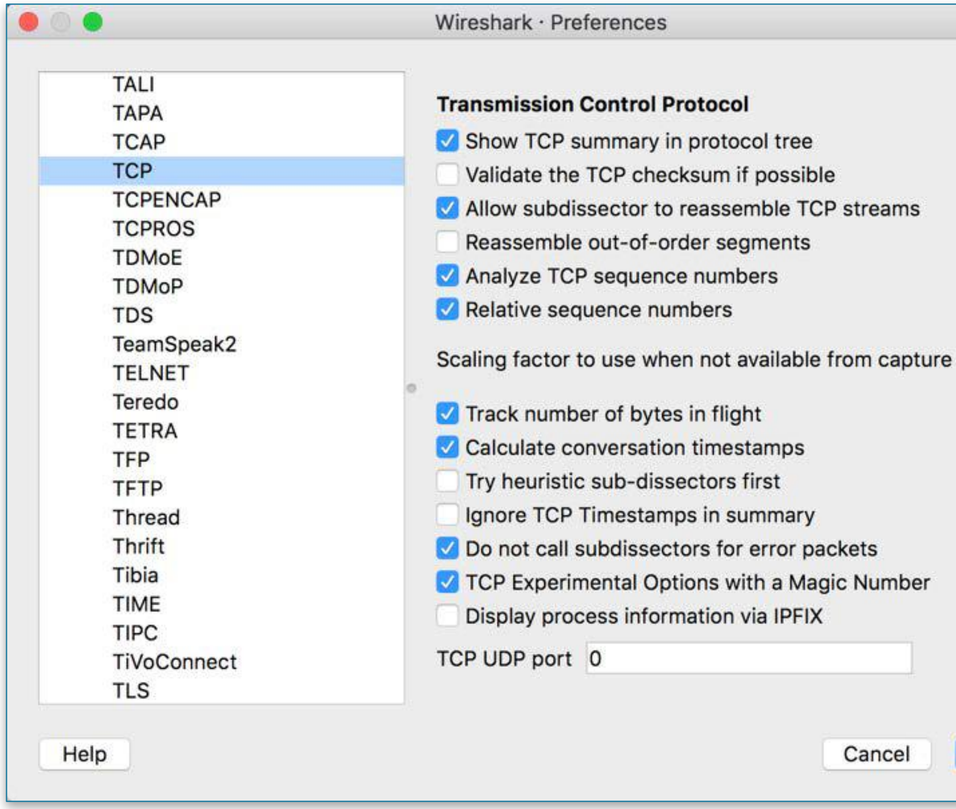
- Next-Gen firewalls
  - Palo Alto Networks (can send decrypted traffic)
  - Checkpoint
  - Fortigate
- Application Delivery Controllers
  - F5 (can export TLS session keys with an iRule)
  - NetScaler (can export TLS session keys when capturing)
- Proxy Applications
  - **mitmproxy (can export TLS session keys)**
  - Fiddler
  - Charles Proxy
  - BurpSuite
  - PolarProxy (can save decrypted traffic in pcap)



https://publicdomainvectors.org/en/free-clipart/Spy-behind-computer/68287.html

Wireshark requirements

- Enable defragmentation on IP layer
- Enable reassembly at TCP and TLS layer
  - And also on HTTP layer if traffic is HTTPS
- Disable checksum validation on IP and TCP
  - Checksum offloading results in bad checksums
  - Reassembly will be disabled with bad chacksums
- Good news, these are all default settings!

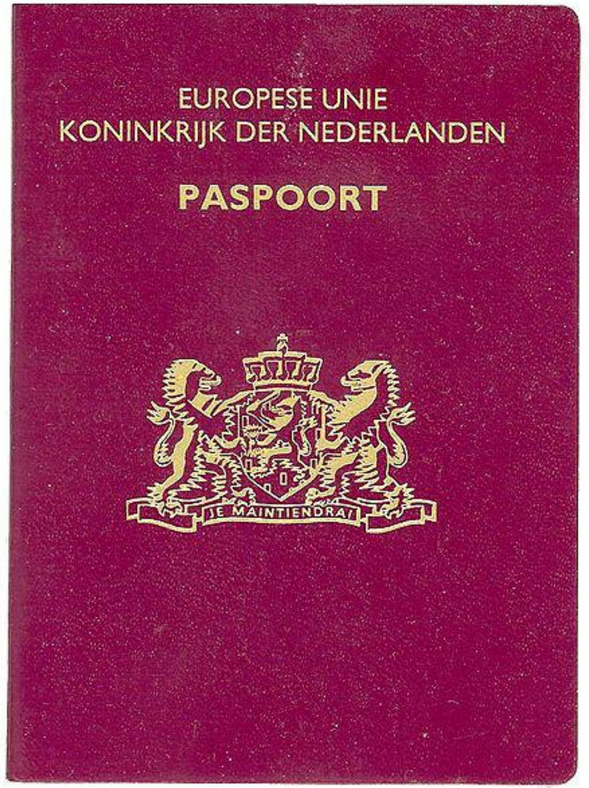


TLS session resumption

- Master Secret is...
  - Cached on Client and Server (indexed by SessionID)
  - Sent in TLS session tickets
    - master secret encrypted by server, stored on client
- So no {Client,Server}KeyExchange in PCAP???
  - **No SessionKey, no decryption!**
- However, decryption will still work if you have captured the initial (full) TLS handshake

Mutual TLS

- Server requests certificate from client
- Client presents certificate to server
  - Contains the clients public key
- Client sends CertificateVerify message
  - Encrypted with clients private key
- Server can decrypt with public key
  - This proves the client has the private key
- Encryption/Decryption is not affected
- **So still only server private key needed for decryption**



https://nl.wikipedia.org/wiki/Bestand:Paspoort_NL.jpg

Check if decryption works

- Higher layer dissection might fail or packet-loss and re-ordering might break decryption later in the session so it might look like decryption failed
- Best check if decryption was succesful:
  - Filter on tls.handshake
  - Check if "Encrypted handshake message" has changed to "Finished"

tls.handshake

No.	Time	Delta	Source	Destination	Protocol	Length	Info
4	15:17:36.050028	0.000000	201.0.113.10	35.153.107.164	TLSv1.2	587	Client Hello
6	15:17:36.161459	0.111431	35.153.107.164	201.0.113.10	TLSv1.2	1518	Server Hello
12	15:17:36.162608	0.001149	35.153.107.164	201.0.113.10	TLSv1.2	1252	Certificate, Server Key Exchange, Server Hello Done
14	15:17:36.164351	0.001743	201.0.113.10	35.153.107.164	TLSv1.2	196	Client Key Exchange, Change Cipher Spec, Finished
15	15:17:36.276708	0.112357	35.153.107.164	201.0.113.10	TLSv1.2	121	Change Cipher Spec, Finished

SSL debug logfile

SSL debug log:

```
ssl_init keys string:
192.168.3.3,443,http,c:\temp\public.sharkfest.local.key
ssl_init found host entry 192.168.3.3,443,http,c:\temp\public.sharkfest.local.key
ssl_init addr '192.168.3.3' port '443' filename 'c:\temp\public.sharkfest.local.key' password(only for p12 file) '(null)'
ssl_load_key: can't import pem data
```

PEM keyfile *without* passphrase:

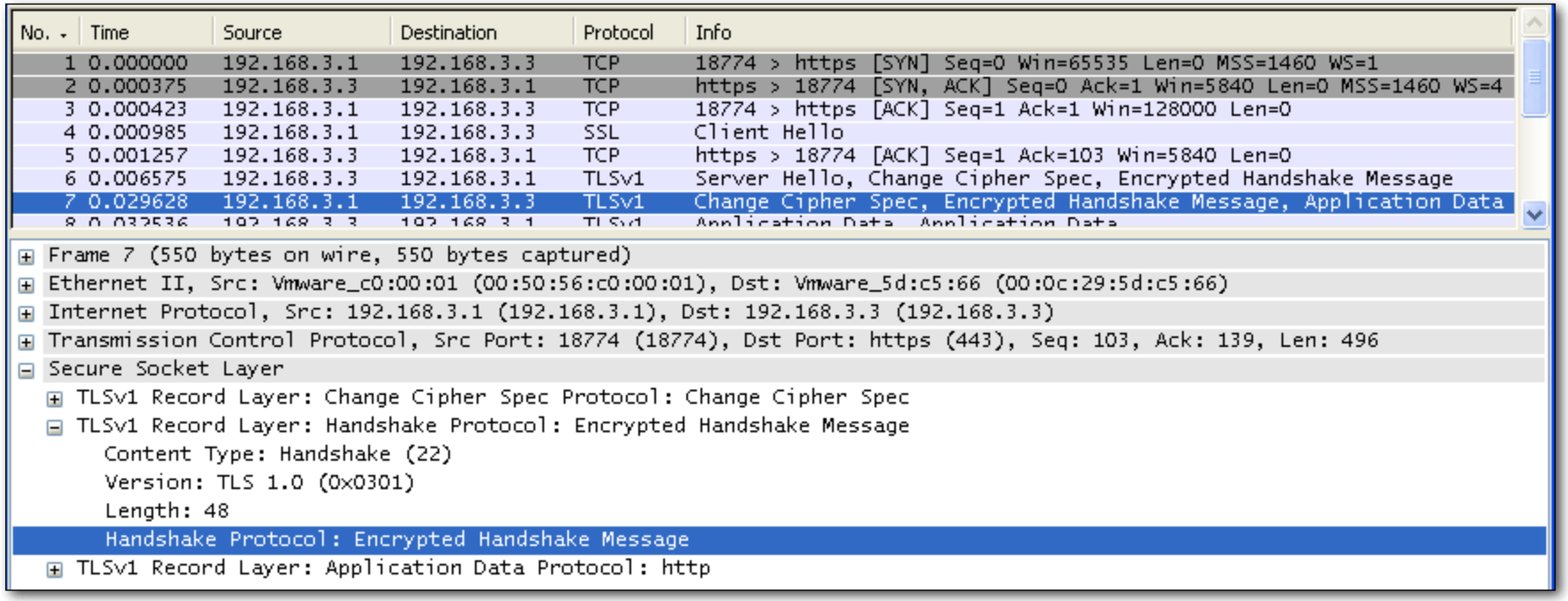
```
-----BEGIN RSA PRIVATE KEY-----
MIICXgIBAAKBgQDrHdbb+yGE6m6EZ03bXURpZCjch2H6g97ZAkJVGrjLZFfettBA
EYa8vYYxWsf8KBpEZeksSCsDA9MnU2H6QDjzqdOnaSWfeXMAr4OsCOpauStpreq7
q1hk8iOqy+f4KijRrhWplh1QW1A8gtSIg137pyUhW+WsfwxKwmzjGIC1SwIDAQAB
AoGBAMneA9U6KIxjb+JUg/99c7h9W6wEvTYHNTXjf6psWA+hpuQ82E65/ZJdszL6
...
b6QKMh16r5wd6smQ+CmhOEnqqyT5AIwwl2RIr9GbfIpTbtbRQw/EcQOCx9wFiEfo
tGSsEFi72rHK+DpJqRI9AkEA72gdyXRgPfGOS3rfQ3DBcImBQvDSCBa4cuU1XJ1/
MO93a8v9Vj87/yDm4xsBDsoz2PyBepawHVlIvZ6jDD0aXw==
-----END RSA PRIVATE KEY-----
```

PEM keyfile *with* passphrase:

```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,F6C218D4FA3C8B66

FR2cnmkkFHH45Dcsty1qDiIUy/uXn+9m/xeQMVRxtiSAmBmnUDUFIFCDDiDc9yif
ERok2jPr2BzAazl5RBxS2TY/+7x0/dHD11sF3LnJUoNruo77TERxqgzOI0W1VDRA
...
ygw5JslxgiN18F36E/cEP5rKvVYvfEPMa6IsiRhfZk1jLAuZihVWc7JodDf+6RKV
yBXrK/bDtdEih+bOnYu+ZDvjAzVz9GhggCW4QHNboDpTxrrYPkj5Nw==
-----END RSA PRIVATE KEY-----
```

Decrypt-problem I (1)



No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.3.1	192.168.3.3	TCP	18774 > https [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=1
2	0.000375	192.168.3.3	192.168.3.1	TCP	https > 18774 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 WS=4
3	0.000423	192.168.3.1	192.168.3.3	TCP	18774 > https [ACK] Seq=1 Ack=1 Win=128000 Len=0
4	0.000985	192.168.3.1	192.168.3.3	SSL	Client Hello
5	0.001257	192.168.3.3	192.168.3.1	TCP	https > 18774 [ACK] Seq=1 Ack=103 Win=5840 Len=0
6	0.006575	192.168.3.3	192.168.3.1	TLSv1	Server Hello, Change Cipher Spec, Encrypted Handshake Message
7	0.029628	192.168.3.1	192.168.3.3	TLSv1	Change Cipher Spec, Encrypted Handshake Message, Application Data
8	0.032536	192.168.3.3	192.168.3.1	TLSv1	Application Data, Application Data

```
⊞ Frame 7 (550 bytes on wire, 550 bytes captured)
⊞ Ethernet II, Src: Vmware_c0:00:01 (00:50:56:c0:00:01), Dst: Vmware_5d:c5:66 (00:0c:29:5d:c5:66)
⊞ Internet Protocol, Src: 192.168.3.1 (192.168.3.1), Dst: 192.168.3.3 (192.168.3.3)
⊞ Transmission Control Protocol, Src Port: 18774 (18774), Dst Port: https (443), Seq: 103, Ack: 139, Len: 496
⊟ Secure Socket Layer
  ⊞ TLSv1 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
  ⊟ TLSv1 Record Layer: Handshake Protocol: Encrypted Handshake Message
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 48
      Handshake Protocol: Encrypted Handshake Message
  ⊞ TLSv1 Record Layer: Application Data Protocol: http
```

```
ssl_init keys string:
192.168.3.3,443,http,c:\temp\public.sharkfest.local.key
ssl_init found host entry 192.168.3.3,443,http,c:\temp\public.sharkfest.local.key
ssl_init addr '192.168.3.3' port '443' filename 'c:\temp\public.sharkfest.local.key' password(only for p12 file)
'(null)'
Private key imported: KeyID B8:2B:EA:B8:F8:BD:62:50:E3:0C:2D:3D:06:09:91:64:...
ssl_init private key file c:\temp\public.sharkfest.local.key successfully loaded
association_add TCP port 443 protocol http handle 04086228
```

Decrypt-problem I (2)

Checking ssl debug log:

```
[...]
dissect_ssl enter frame #7 (first time)
  conversation = 07411870, ssl_session = 07411BC8
  record: offset = 0, reported_length_remaining = 496
dissect_ssl3_record: content_type 20
dissect_ssl3_change_cipher_spec
association_find: TCP port 18774 found 00000000
packet_from_server: is from server - FALSE
ssl_change_cipher CLIENT
  record: offset = 6, reported_length_remaining = 490
dissect_ssl3_record: content_type 22
decrypt_ssl3_record: app_data len 48 ssl, state 0x17
association_f
packet_from_s
decrypt_ssl3
decrypt_ssl3
dissect_ssl3
  record: of
dissect_ssl3
decrypt_ssl3
association_find: TCP port 18774 found 00000000
packet_from_server: is from server - FALSE
decrypt_ssl3_record: using client decoder
decrypt_ssl3_record: no decoder available
association_find: TCP port 18774 found 00000000
association_find: TCP port 443 found 047AF518
[...]
```

Make sure that the whole SSL session (which can be made out of multiple TCP streams) is in the tracefile. Starting with the handshake and up to the current frame.

Decrypt-problem II (1)

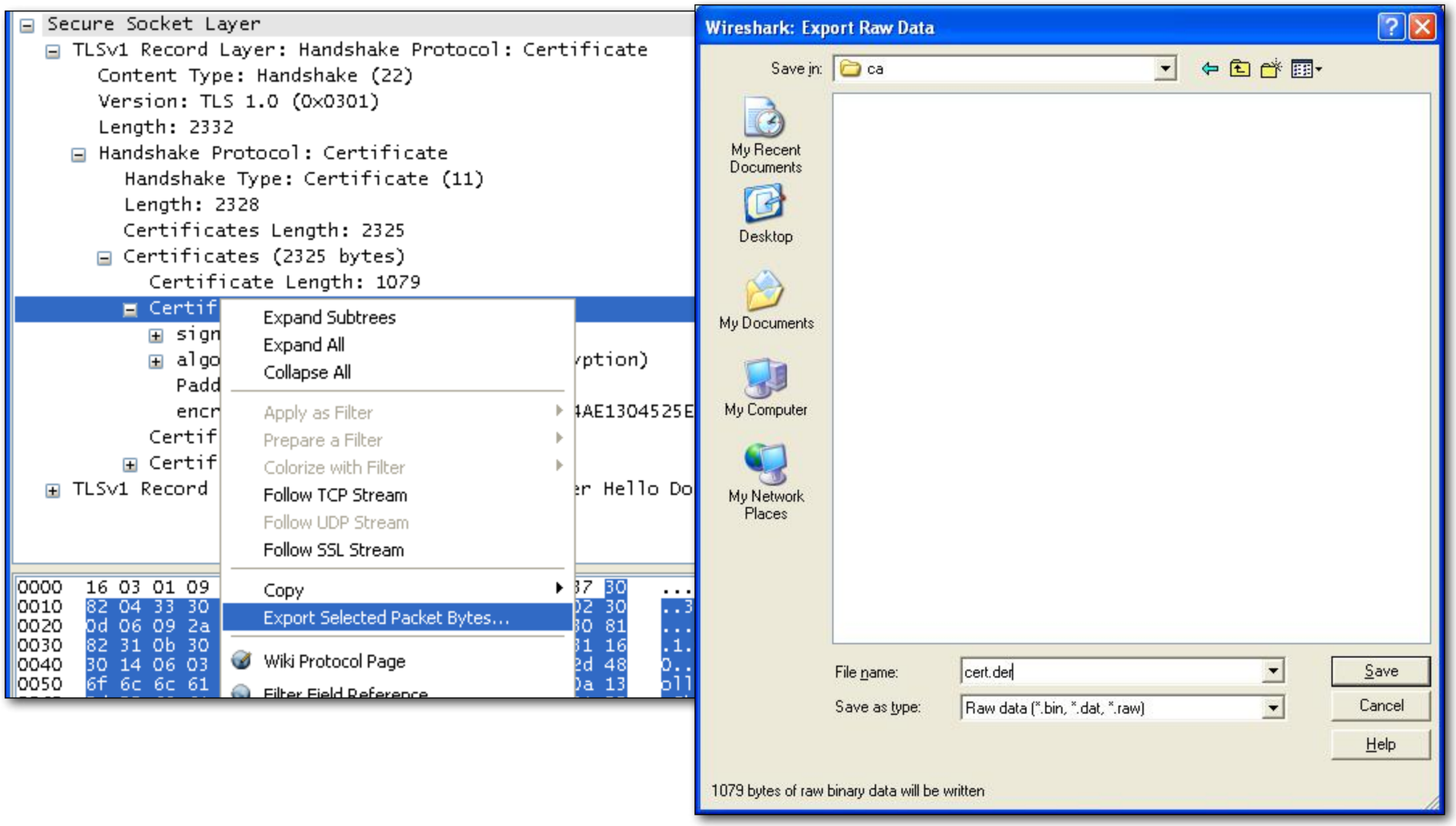
Checking ssl debug log:

```
ssl_association_remove removing TCP 443 - http handle 04086F30
ssl_init keys string:
192.168.3.3,443,http,c:\temp\public.sharkfest.local.key
ssl_init found host entry 192.168.3.3,443,http,c:\temp\public.sharkfest.local.key
ssl_init addr '192.168.3.3' port '443' filename 'c:\temp\public.sharkfest.local.key' password(only for p12 file)
'(null)'
Private key imported: KeyID FA:56:73:A4:38:9C:A1:4F:28:23:88:76:83:42:13:86:...
ssl_init private key file c:\temp\public.sharkfest.local.key successfully loaded
association_add TCP port 443 protocol http handle 04086F30

[...]

ssl_decrypt_pre_master_secret:RSA_private_decrypt
pcry_private_decrypt: stripping 0 bytes, decr_len zd
decrypted_unstrip_pre_master[128]:
6a f7 2a 4b 45 17 72 47 c2 11 d1 dd ad dc af b6
04 76 cb 3c 32 1c d1 01 57 4a 83 79 af d9 40 af
aa a8 71 1f bd 6f 70 d5 cc 49 e6 be 44 42 07 7c
45 b7 5b 5b 52 de 3e 58 d3 42 8d 5f bc 99 3e 13
f5 7d 27 a1 3e 7f b2 3f 8b 9d e5 fb 60 ec 40 26
87 8f 24 41 fb d4 ec f7 0e ea 04 46 c2 d7 5f 7b
4a d2 40 47 07 7b 0d 63 d8 d6 0f e6 9e 98 92 02
58 13 51 72 1b 85 69 04 52 42 74 12 40 e2 a5 bb
ssl_decrypt_pre_master_secret wrong pre_master_secret length (128, expected 48)
dissect_ssl3_handshake can't decrypt pre master secret
```

Decrypt-problem II (2)



Decrypt-problem II (3)

In wireshark preferences:

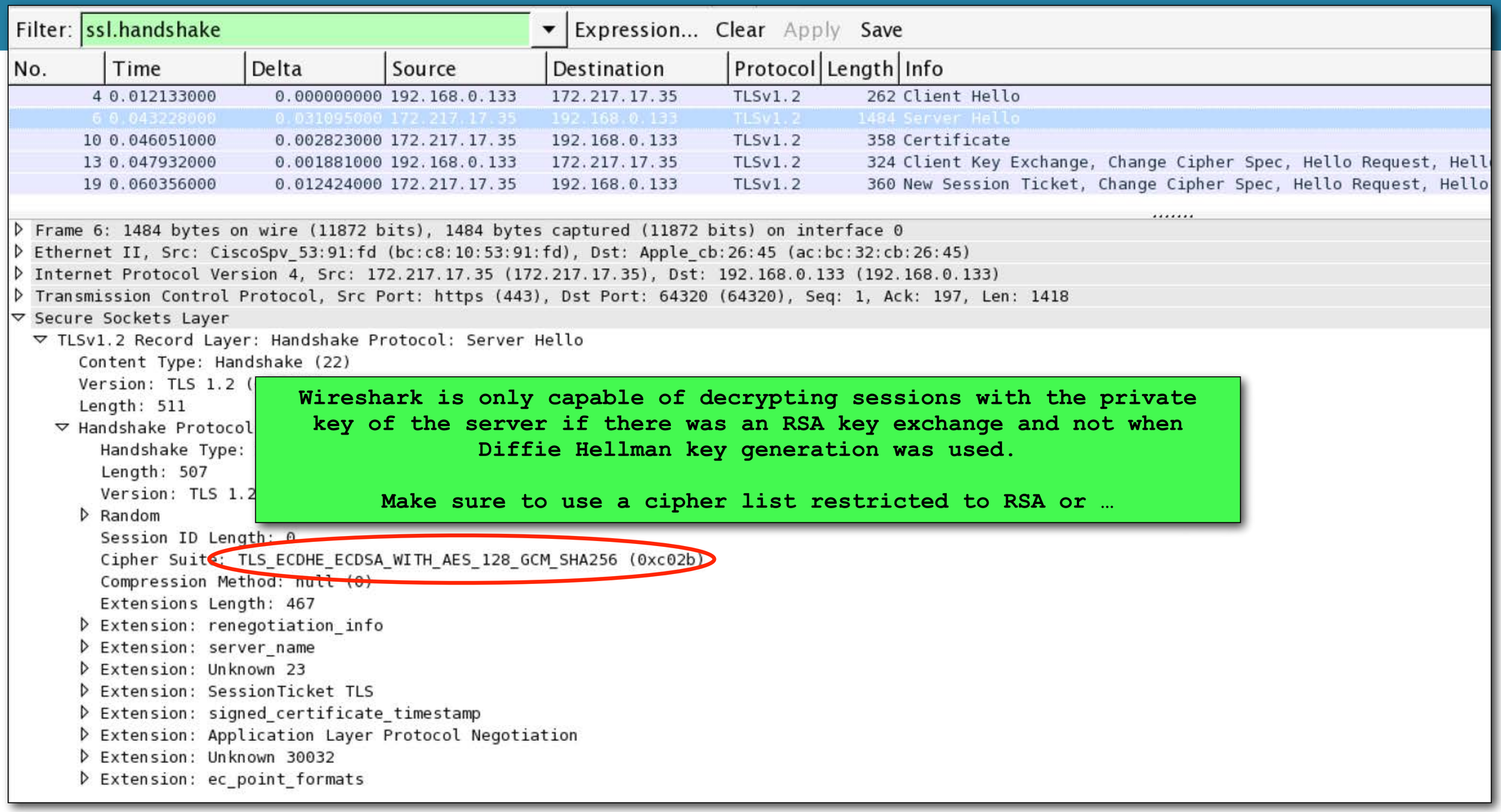
```
ssl.keys_list: 192.168.3.3,443,http,c:\temp\public.sharkfest.local.key
```

Checking whether certificate and key match:

```
$ openssl x509 -in cert.der -inform DER -noout -text | grep "Subject:"
        Subject: C=NL, ST=Noord-Holland, O=Sharkfest Lab, CN=public.sharkfest.local/
emailAddress=co@sharkfest.local
$
$ openssl
a29682af82
$
$ openssl
ce71158d3851a885314c264863142389
$
$ openssl rsa -noout -modulus -in private.sharkfest.local.key | openssl md5
a29682af822b4cd064d39d4ccd1e0e6c
$
```

Make sure that the private key matches the (server) certificate that is used in the tracefile.

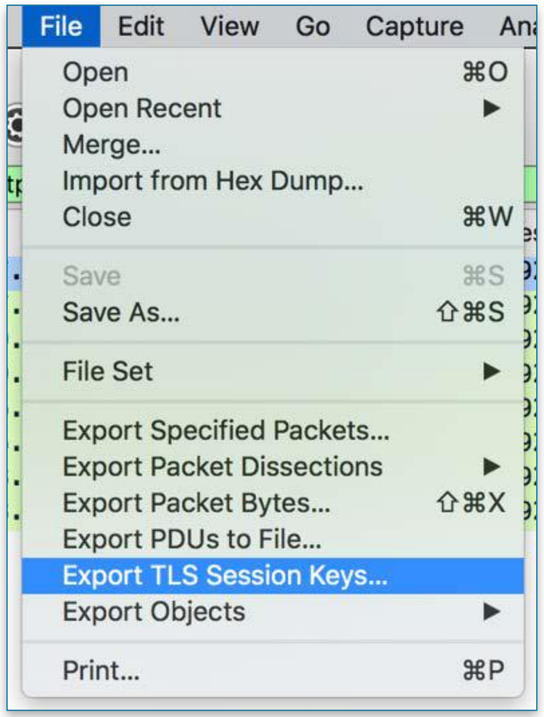
Decrypt-problem III



Wireshark is only capable of decrypting sessions with the private key of the server if there was an RSA key exchange and not when Diffie Hellman key generation was used.

Make sure to use a cipher list restricted to RSA or …

Exporting TLS Session Keys



The name of a file which contains a list of
(pre-)master secrets in one of the following formats:

RSA <EPMS> <PMS>
RSA Session-ID:<SSLID> Master-Key:<MS>
CLIENT_RANDOM <CRAND> <MS>
PMS_CLIENT_RANDOM <CRAND> <PMS>

Where:
<EPMS> = First 8 bytes of the Encrypted PMS
<PMS> = The Pre-Master-Secret (PMS) used to derive the MS
<SSLID> = The SSL Session ID
<MS> = The Master-Secret (MS)
<CRAND> = The Client's random number from the ClientHello message

(All fields are in hex notation)

```
$ cat session.keys
RSA Session-ID:fbcf322128ed0a00b272d6ac85843f50deccdd94ac33261523189639f5ba189a Master-Key:bda6ea472f6c39a9fcfd5dc79eb161d1a4cae5d924fdde800f276263fd6df1ee8ed246b5a6412e328eb85744c9bf7cf2
RSA Session-ID:db00c2aad79cfda109ce4f65a9801aa8d5f1bbeb9e1f848f6a2f7551f9de7577 Master-Key:92cdc769c670ba6f48cfe756992ad435401a26d0235900c0f67c846b5f360c108df167ca6b6f443f4d2b118de0ccadb8
CLIENT_RANDOM 49eb46c69e4c71ebfb757f67630e84ad2da1b0c2c1a9cbba1d7a48744ed2e640 92cdc769c670ba6f48cfe756992ad435401a26d0235900c0f67c846b5f360c108df167ca6b6f443f4d2b118de0ccadb8
CLIENT_RANDOM 49eb470dd56239e9c8c4e7acb752614255177a3e77f47660a445385268335563 bda6ea472f6c39a9fcfd5dc79eb161d1a4cae5d924fdde800f276263fd6df1ee8ed246b5a6412e328eb85744c9bf7cf2
CLIENT_RANDOM 49eb469edd819516fc5dddd097428d410d7852e2579e2e8903cdb331c763dca9 92cdc769c670ba6f48cfe756992ad435401a26d0235900c0f67c846b5f360c108df167ca6b6f443f4d2b118de0ccadb8
CLIENT_RANDOM c8c82c81e65b2ba9df1234e660e5db4cc35b93720ba4eb2a1c01138b7ad56eaa 91788e318de3d91734c1e6eb34366f26709db19f56d9c8dde5248bb9a1560d2c9e5a2dc92d167c9df1fadf36fb395a09
$
```

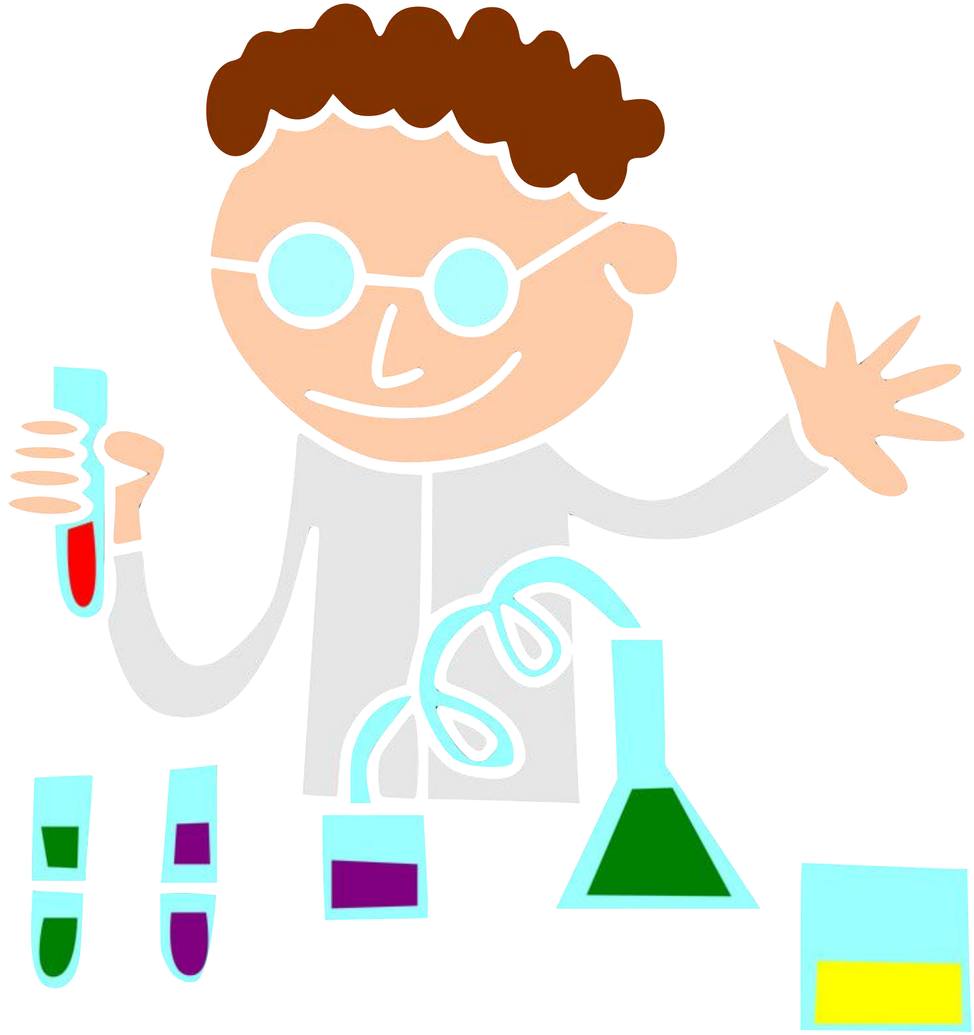
Embedding TLS Sesssion Keys

```
$ editcap -h
[...]
  --inject-secrets <type>,<file>  Insert decryption secrets from <file>. List
                                  supported secret types with "--inject-secrets help".
  --discard-all-secrets  Discard all decryption secrets from the input file
                         when writing the output file.  Does not discard
                         secrets added by "--inject-secrets" in the same
                         command line.
[...]
$ editcap --inject-secrets tls,session.keys tls-rsa.pcapng tls-embedded.pcapng
$
```

```
$ fgrep -f <(tshark -r tls-rsa.pcapng -Y tls.handshake.random -T fields -e tls.handshake.random) session.keys
CLIENT_RANDOM 49eb46c69e4c71ebfb757f67630e84ad2da1b0c2c1a9cbba1d7a48744ed2e640 92cdc769c670ba6f48cfe756992ad435401a26d0235900c0f67c846b5f360c108df167ca6b6f443f4d2b118de0ccadb8
CLIENT_RANDOM 49eb470dd56239e9c8c4e7acb752614255177a3e77f47660a445385268335563 bda6ea472f6c39a9fcfd5dc79eb161d1a4cae5d924fdde800f276263fd6df1ee8ed246b5a6412e328eb85744c9bf7cf2
CLIENT_RANDOM 49eb469edd819516fc5dddd097428d410d7852e2579e2e8903cdb331c763dca9 92cdc769c670ba6f48cfe756992ad435401a26d0235900c0f67c846b5f360c108df167ca6b6f443f4d2b118de0ccadb8
$
```

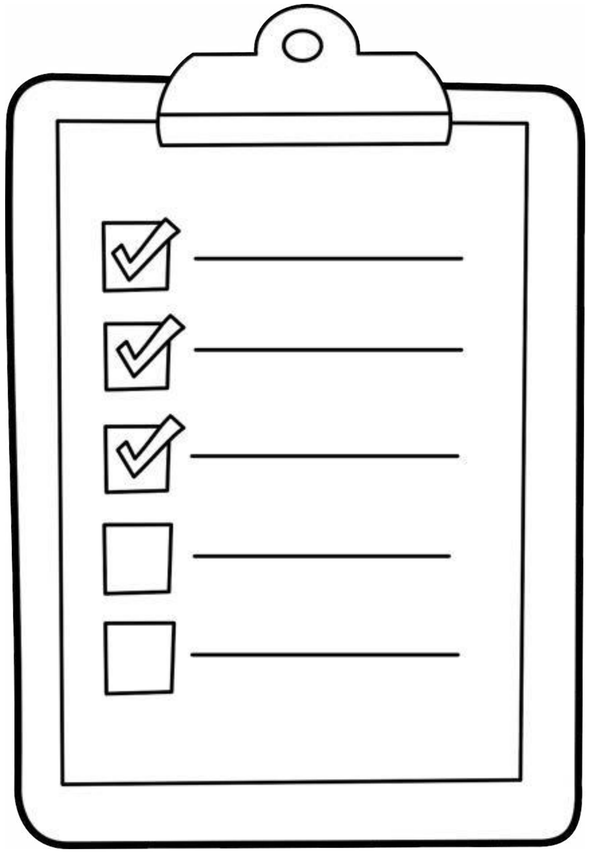
Exercise

- Exercise
  - Open tls-rsa.pcapng
  - Use tls-rsa.pem to decrypt
  - Export session keys to tls-rsa.keys
  - Edit file and remove one entry
  - Change TLS protocol settings
    - Remove the RSA private key
    - Add (Pre)-master-secret log filename
  - Is everything still encrypted?
  - Good way to provide traces to 3rd parties



Agenda

- TLS fundamentals
  - The need for TLS (and what happend to SSL?)
  - Cryptology 101 & PKI
  - Troubleshoot the TLS handshake (SSLv3 - TLSv1.2)
  - Troubleshoot the TLS handshake (TLSv1.3)
- Analysing Application Data
  - Without decryption
  - With decryption using the servers private key
  - **With decryption using TLS session keys**
- Creating configuration profiles for TLS analysis
- SSL/TLS vulnerabilities & privacy concerns
- Summary and Q&A



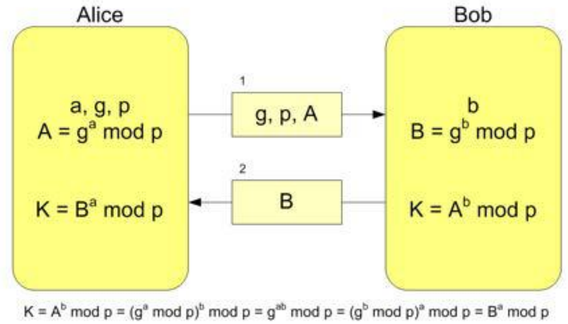
https://www.needpix.com/photo/download/94588/checklist-action-check-list-verification-tick-approved-okay-check-mark

So long and thanks for all the fish

- Perfect Forwarding Secrecy
  - **Diffie-Hellman algorithm uses ephemeral keys to create pre-master secret**
  - Public key of the server only used for authentication
  - Prevents decryption of current pcaps if private key is leaked in the future

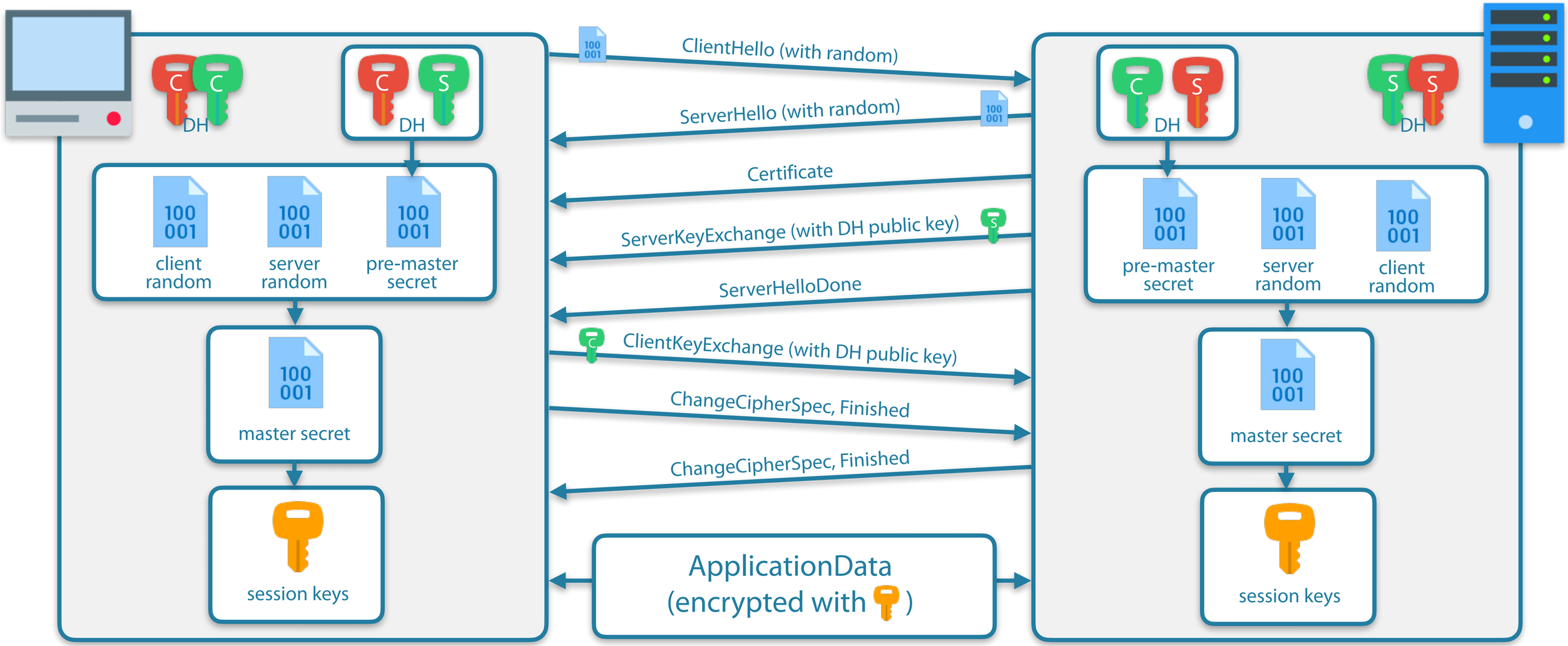
- Today, >80% sessions use a cipher with DHE key negotiation
  - See: https://notary.icsi.berkeley.edu/#connection-cipher-details

- TLSv1.3 finalised in august 2018 (RFC 8446)
  - After 5 years and 28 versions!
  - **No more RSA key exchanges!**



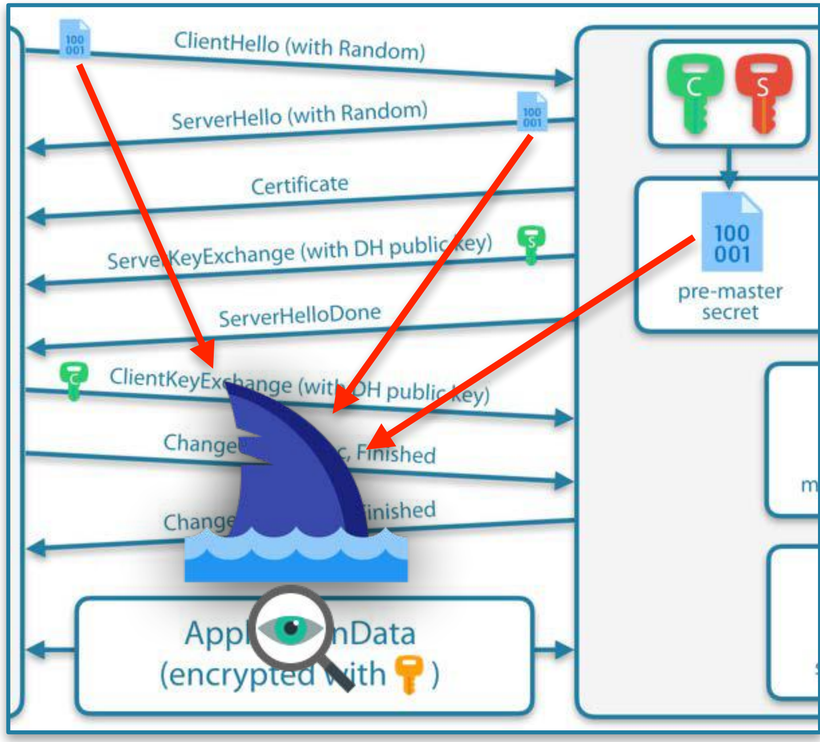
https://nl.wikipedia.org/wiki/Diffie-Hellman-sleuteluitwisselingsprotocol

DHE key generation (TLSv1.2)



Decryption Recipe II

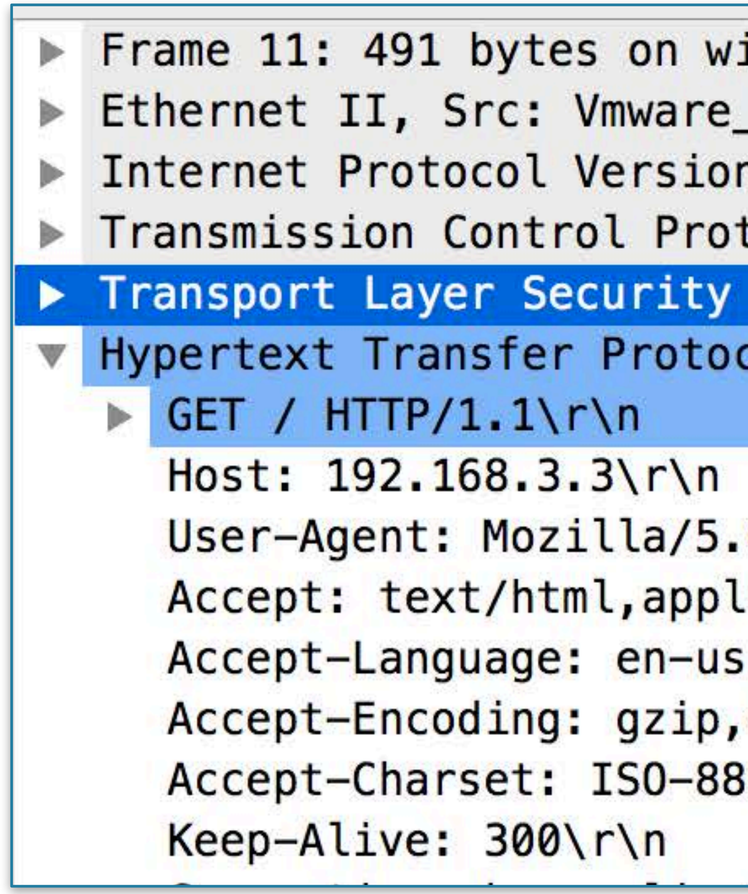
- Client random
  - Captured in the **ClientHello**
- Server random
  - Captured in the **ServerHello**
- Pre-master Secret
  - Can't be calculated, because DH private keys are ephemeral and only available in memory
  - **Pre-master Secret must be logged by either end-point**
- Calculate Master Secret
- Calculate Session keys
- Decrypt application traffic with Session keys



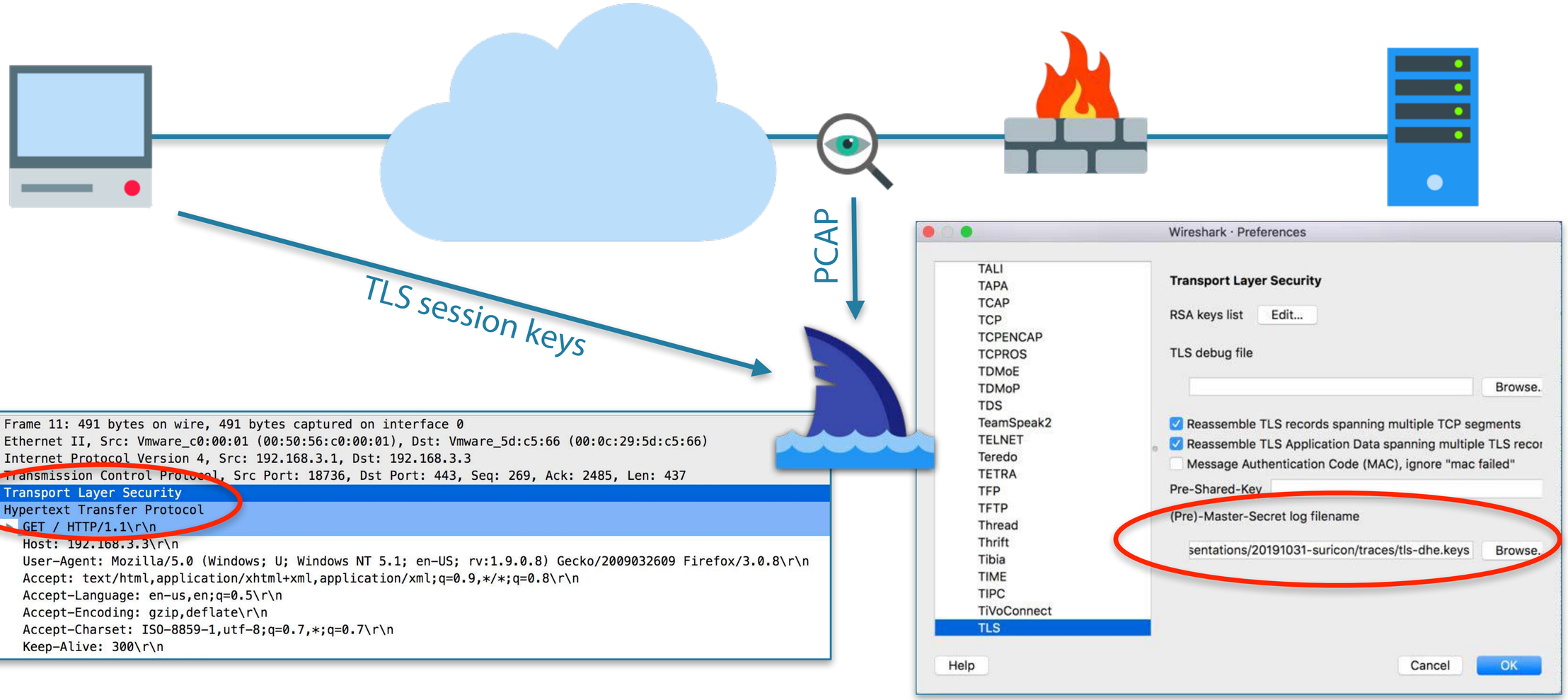
Decryption in Wireshark II

- Start capturing **before** opening the browser
  - Otherwise you will miss the full TLS handshake
- use SSLKEYLOGFILE environment variable
  - with cURL, Chrome, Firefox
- Point to the file with the TLS (Pre)-Master Secrets

- Want to try yourself?
  - Go to: https://www.cloudshark.org/captures/1cd191698a5c
  - Click on More Info -> Download
  - And look a the capture file comments for instructions

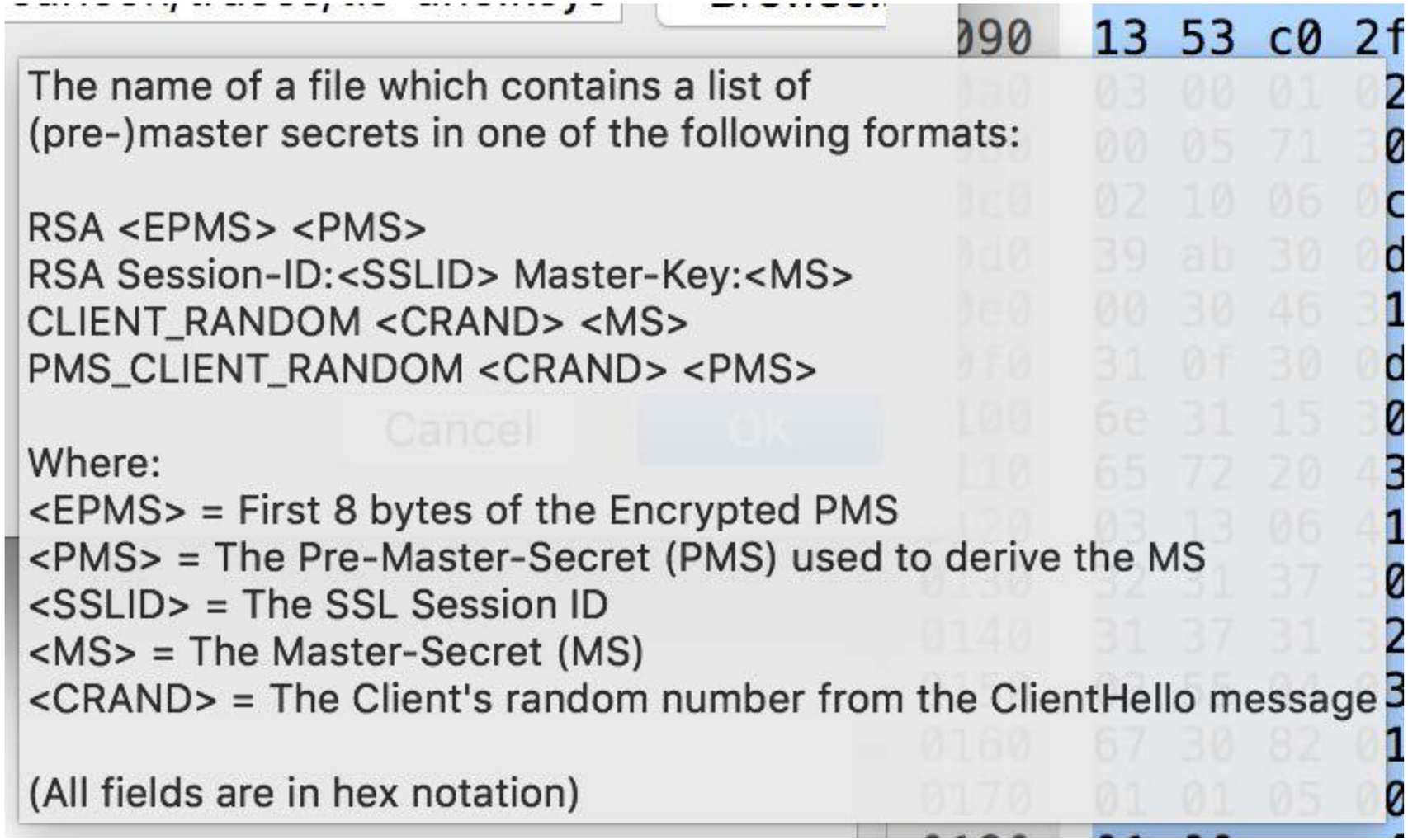


TLS Decryption with session keys



(Pre)-Master Secret logfile

- format of entries
- TLSv1.3 records not shown here



How to log TLS session keys I

- Applications using NSS or GnuTLS
  - Like Chrome and Firefox
  - use SSLKEYLOGFILE environment variable
- Applications using OpenSSL (>1.1.1) or BoringSSL (>d28f59c27bac 2015-11-19)
  - Use `SSL_CTX_set_keylog_callback` function
- cURL (> 7.58.0)
  - use SSLKEYLOGFILE environment variable
- openssl s_client
  - Logs Master-Key in output

How to log TLS session keys II

- Java Applications
  - use jSSLKeyLog (http://jsslkeylog.sourceforge.net/)
- From F5 Application Delivery Controllers
  - Can be done with a few iRule statements
- SChannel (TLS library in Windows)
  - Feature request is pending
- SecureTransport (TLS library in macOS)
  - No support yet

How to 'steal' TLS session keys

- Extract from memory (on the host/VM itself)
  - From a memory dump (for forensics)
  - From live memory (for monitoring)
    - Not just research, first commercial product already seen
- Extract from VM memory (on the hypervisor)
  - TeLeScope
  - TLSkex
- Use a proxy and log the keys
  - E.g. use mitmproxy (set SSLKEYLOGFILE)

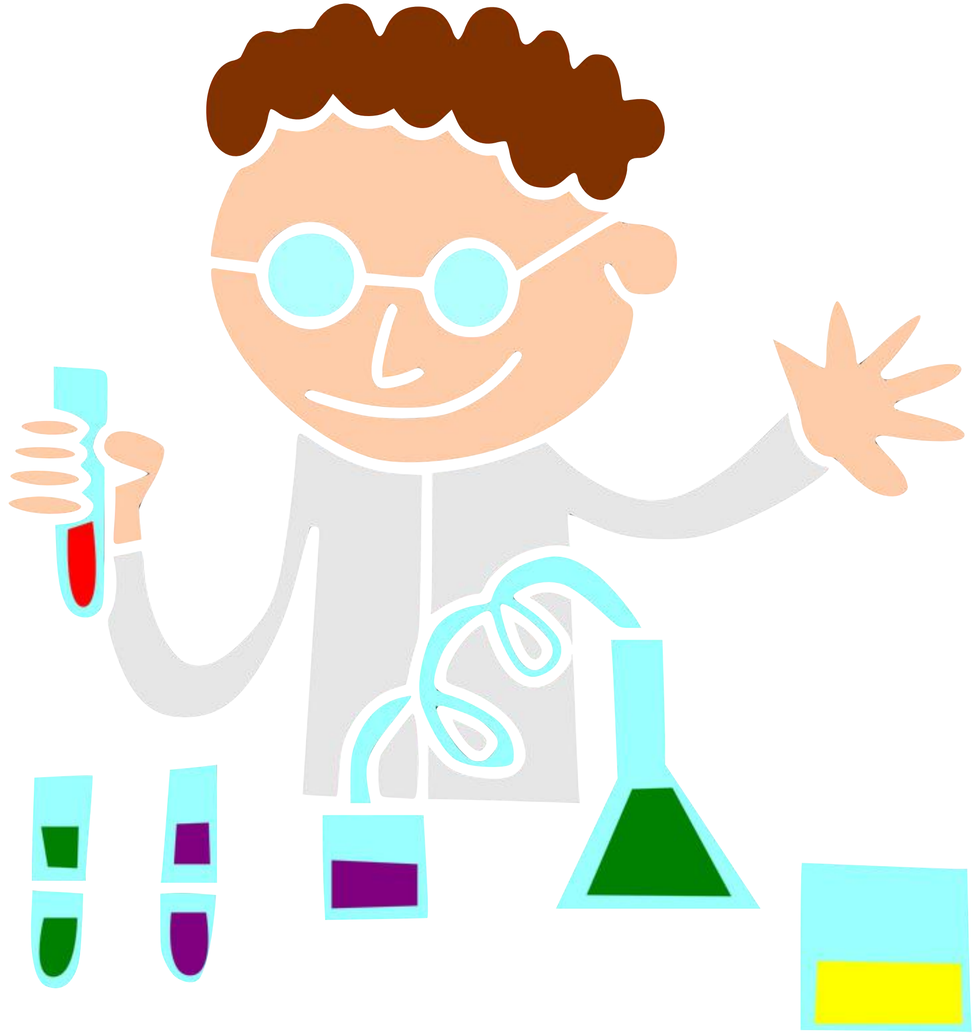
Using SSLKEYLOGFILE

```
sake@MacSake:~$ tcpdump -nli en0 -w tls.pcapng -s0 host www.syn-bit.nl 2>/dev/null &
[1] 62366
sake@MacSake:~$ SSLKEYLOGFILE=./tls.keys curl https://www.SYN-bit.nl > /dev/null 2>&1
sake@MacSake:~$ kill %1
sake@MacSake:~$ cat tls.keys
CLIENT_RANDOM 8643368556C3DFBA4F25F5B8C14AC3AD47EE62A01BABDFF03E11AC384661547E BCDC163071B12AFE7B9FCC27A804137BA3AB4A4DDEFAF8B4476351D6499616CD0047B06CD51E8274569143ABABBDB1EB
[1]+  Done                    tcpdump -nli en0 -w tls.pcapng -s0 host www.syn-bit.nl 2> /dev/null
sake@MacSake:~$ tshark -r tls.pcapng -o tls.keylog_file:tls.keys -Y "tls.handshake"
    4   0.062445 0.000000   10.0.0.102 ? 77.111.240.149 TLSv1 299 0x0000 (0) Client Hello
    6   0.102868 0.040423 77.111.240.149 ? 10.0.0.102   TLSv1.2 1514 0xe95c (59740) Server Hello
    7   0.103027 0.000159 77.111.240.149 ? 10.0.0.102   TLSv1.2 1514 0xe95d (59741) Certificate [TCP segment of a reassembled PDU]
    8   0.103033 0.000006 77.111.240.149 ? 10.0.0.102   TLSv1.2 155 0xe95e (59742) Server Key Exchange, Server Hello Done
   11   0.105671 0.002638   10.0.0.102 ? 77.111.240.149 TLSv1.2 159 0x0000 (0) Client Key Exchange, Change Cipher Spec, Finished
   12   0.144614 0.038943 77.111.240.149 ? 10.0.0.102   TLSv1.2 117 0xe95f (59743) Change Cipher Spec, Finished
sake@MacSake:~$
```

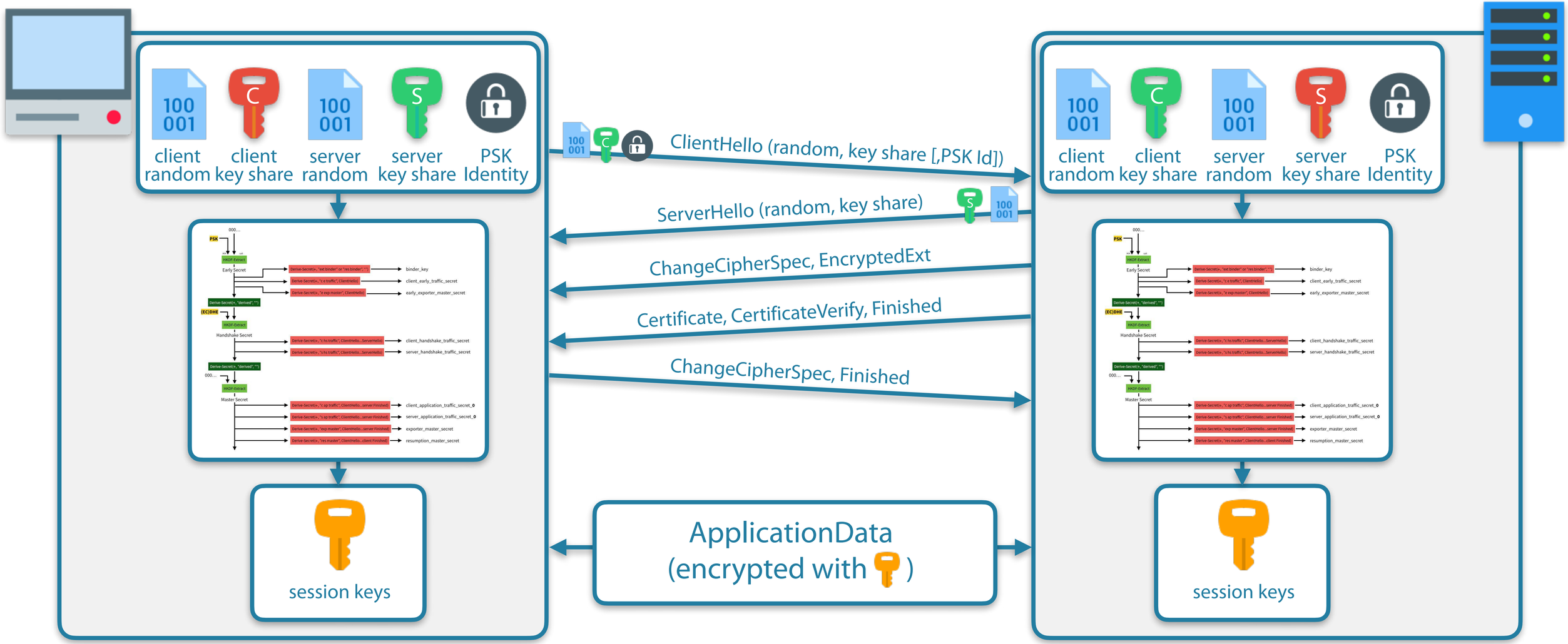
https://redflagsecurity.net/2019/03/10/decrypting-tls-wireshark/

Demo & Exercise

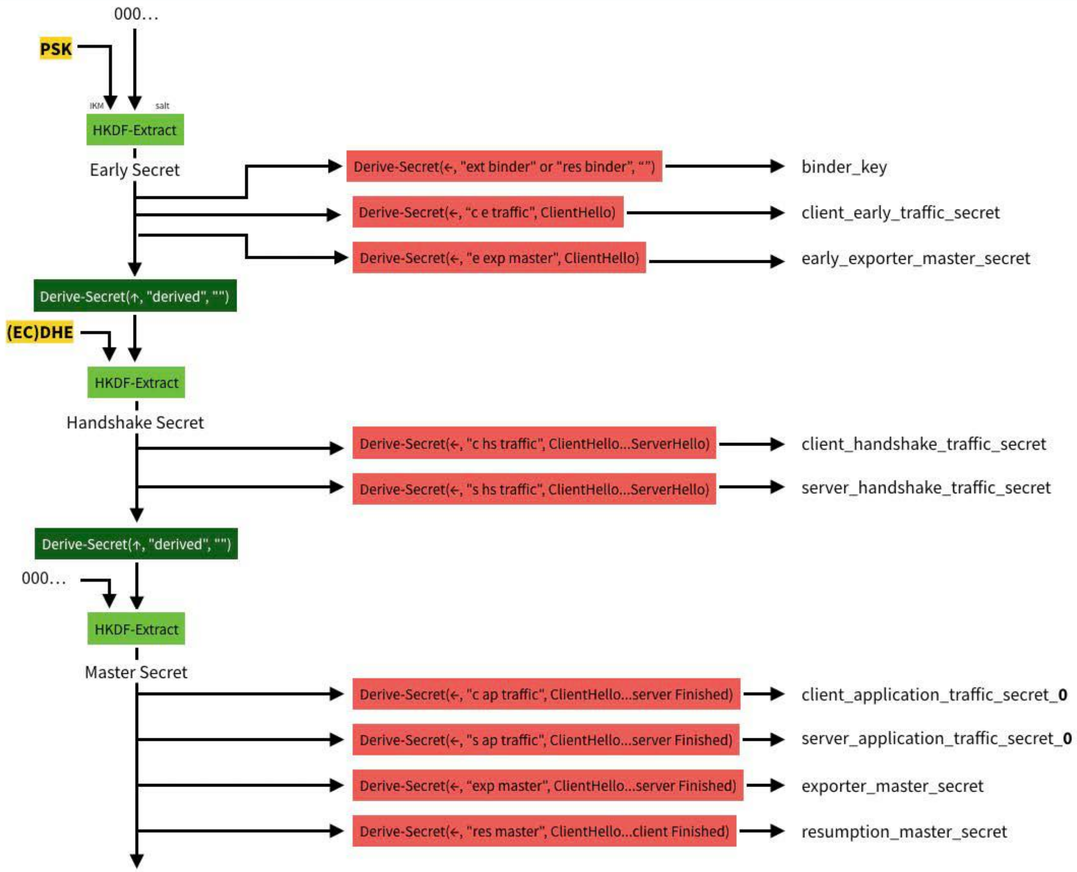
- Demo
  - Using SSLKEYLOGFILE
- Exercise
  - Use SSLKEYLOGFILE with Chrome or Firefox
  - Start Wireshark
  - Point to key file
  - Go to https:// site
  - Filter on "tls and (http || http2)"



Key generation in TLSv1.3



Deriving TLSv1.3 keying material

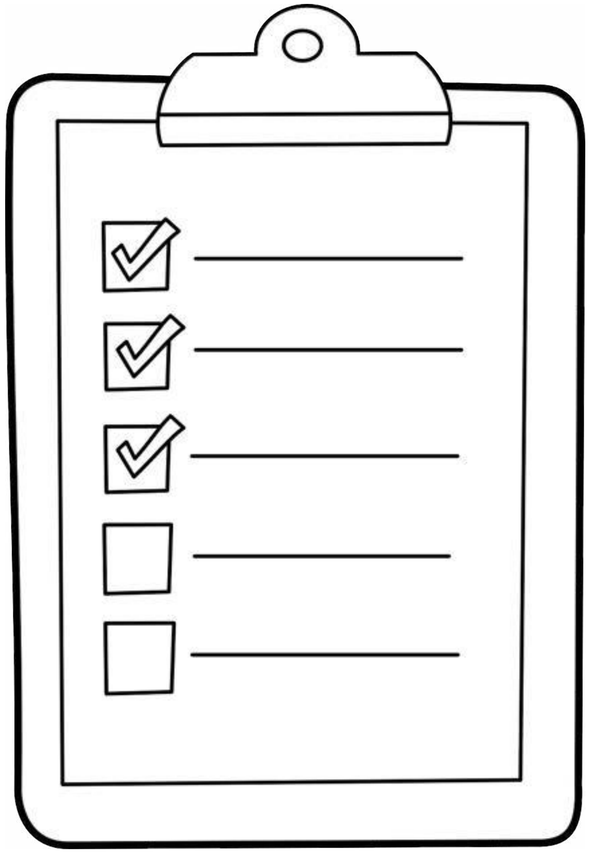


https://www.davidwong.fr/tls13/

```
sake@MacSake:~$$ grep 042812bd0d8df6c71c5405789efdb77d06a076b94127492a4554e85e2f623f87 dv*keys
CLIENT_HANDSHAKE_TRAFFIC_SECRET 042812bd0d8df6c71c5405789efdb77d06a076b94127492a4554e85e2f623f87 984238a19484ce65e8395a23faad5117649e40533187ed7b5e883569412115cf3b9c59b156a164c3f86c31b8adea9af7
SERVER_HANDSHAKE_TRAFFIC_SECRET 042812bd0d8df6c71c5405789efdb77d06a076b94127492a4554e85e2f623f87 089af9caf6b42ec56793cfc91b95f833692e3807ce1f2a218bc7aaacfbd0b7e4bf3ee0bbbbd8cc5eac2083626bf15b8e
CLIENT_TRAFFIC_SECRET_0 042812bd0d8df6c71c5405789efdb77d06a076b94127492a4554e85e2f623f87 530f5a1811f170bb957bc6a092c77ced8dd459697cfb5d8ec7f0a47ba63747386f394a8a0b824c91e656686876ee32a5
SERVER_TRAFFIC_SECRET_0 042812bd0d8df6c71c5405789efdb77d06a076b94127492a4554e85e2f623f87 9e426f3a086837024198448b077b1da63215b184b6b27deadc509463c479a918fb84a98f673627574a3f554037a5e9f8
EXPORTER_SECRET 042812bd0d8df6c71c5405789efdb77d06a076b94127492a4554e85e2f623f87 6add3613f5fcc8679aa5f9313a683f03acfb02141d3897eb34ef16ae4a0f24872a9ae97fe7cea5fb1ea0c4a2fde41062
sake@MacSake:~$
```

Agenda

- TLS fundamentals
  - The need for TLS (and what happend to SSL?)
  - Cryptology 101 & PKI
  - Troubleshoot the TLS handshake (SSLv3 - TLSv1.2)
  - Troubleshoot the TLS handshake (TLSv1.3)
- Analysing Application Data
  - Without decryption
  - With decryption using the servers private key
  - With decryption using TLS session keys
- **Creating configuration profiles for TLS analysis**
- SSL/TLS vulnerabilities & privacy concerns
- Summary and Q&A



https://www.needpix.com/photo/download/94588/checklist-action-check-list-verification-tick-approved-okay-check-mark

What is a configuration profile?

- all wiresharks customizations in one package
- Easily switch between profiles
- Optimize workflow
- Share profiles with others

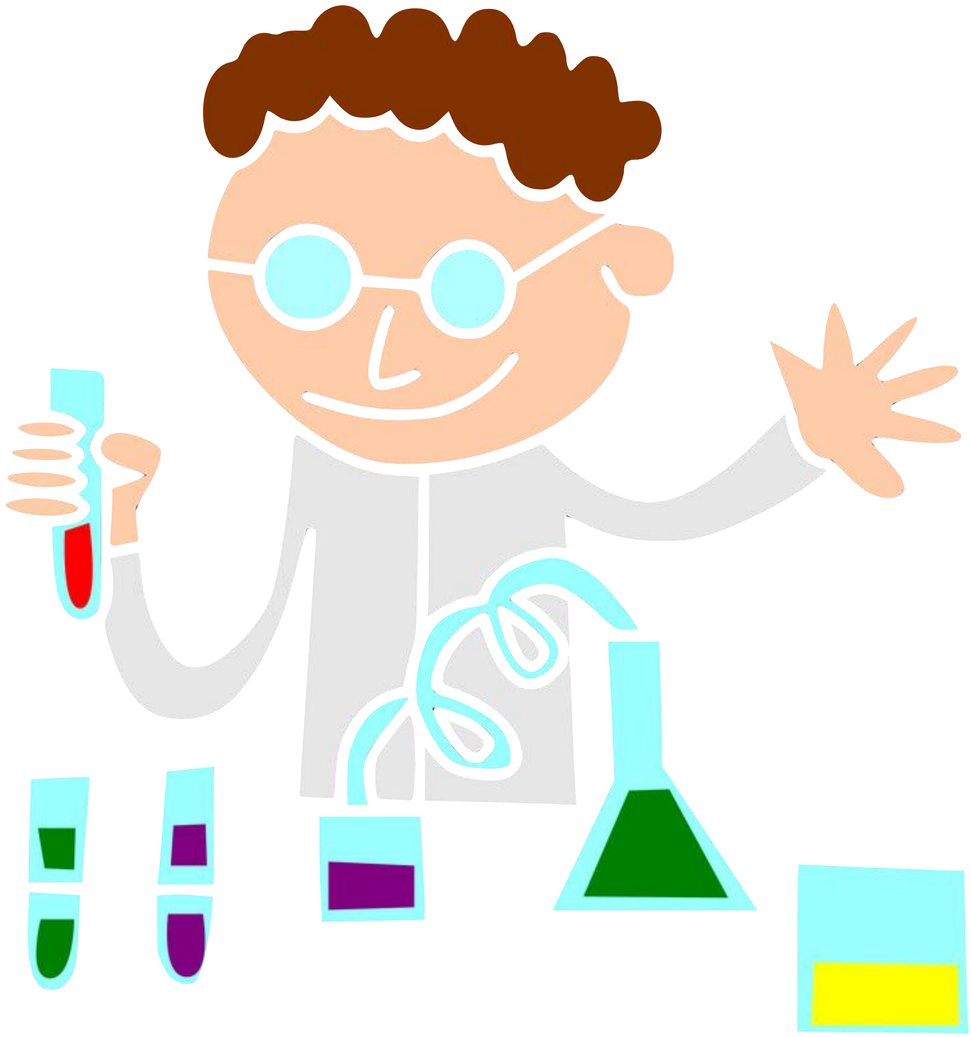
Why use configuration profiles?

- Differentiate your analysis workflow
  - Per protocol?
  - Per application?
  - Per customer?
- Increase performance
  - disable protocols
  - disable reassembly etc.
- Adapt to different setups (e.g. screen size)

Common Configuration Profile Elements

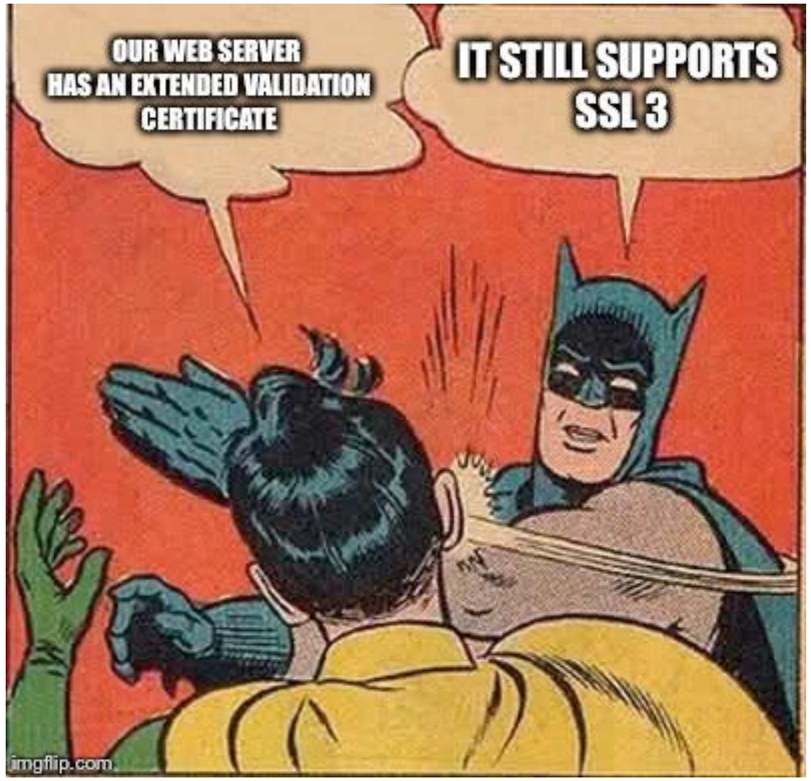
- GUI
  - Layout
  - Columns
  - Colors
- Workflow
  - filter buttons
  - name resolution
- Dissection
  - Enabled protocols
  - Protocol preferences

Demo



Agenda

- TLS fundamentals
  - The need for TLS (and what happend to SSL?)
  - Cryptology 101 & PKI
  - Troubleshoot the TLS handshake (SSLv3 - TLSv1.2)
  - Troubleshoot the TLS handshake (TLSv1.3)
- Analysing Application Data
  - Without decryption
  - With decryption using the servers private key
  - With decryption using TLS session keys
- Creating configuration profiles for TLS analysis
- **SSL/TLS vulnerabilities & privacy concerns**
- Summary and Q&A



Vulnerabilities

- SSLv2 : multiple vulnerabilities
- SSLv3 : POODLE
- Weak hashing :
  - MD5
  - SHA-1
- Weak keylengths :
  - Minimum safe keylength for RSA : 2048
  - Minimum safe keylength for ECDSA : 233
- Beast, Crime, Freak
- OpenSSL : Heartbleed
- Renegotiation
- RC4
- And many more...

https://www.digicert.com/cert-inspector-vulnerabilities.htm

TLS best practices

- Certificate management & monitoring
  - Use a reliable CA that matches your purpose
  - Use Certificate Authority Authorization (CAA)
  - Use Signed Certificate Timestamps (SCT)
  - Protect your private keys!
- TLS configuration
  - Use 2048 bit RSA and/or 256 bit ECDSA
  - Use SHA256 signatures
  - Use TLSv1.2 and/or TLSv1.3
  - Restrict ciphers to known safe ones
  - Use PFS (by using ECDHE)
  - Use OCSP stapling

https://www.flickr.com/photos/barrydahl/6675297699

Certificate Authority Authorization

- RFC 8659 (obsoletes 6844)
- Add CAA records to DNS to list accepted CA's
- Mandatory for CA's to check (sep 2017)
  - but allowed to sign certificates if no CAA record present
- Use of DNSSEC recommended

Signed Certificate Timestamps

- RFC 6962
- Log all certificates to an append-only log
- Monitor / Audit the logs for irregularities

Embedded SCTs	
Log ID	7D:3E:F2:F8:8F:FF:88:55:68:24:C2:C0:CA:9E:52:89:79:2B:C5:0E:78:09:7F:2E:6A:97:68:99:7E:22:F0:D7
Name	Google "Xenon2021"
Signature Algorithm	SHA-256 ECDSA
Version	1
Timestamp	9/2/2020, 8:28:03 AM (Central European Summer Time)
Log ID	44:94:65:2E:B0:EE:CE:AF:C4:40:07:D8:A8:FE:28:C0:DA:E6:82:BE:D8:CB:31:B5:3F:D3:33:96:B5:B6:81:A8
Name	Cloudflare "Nimbus2021"
Signature Algorithm	SHA-256 ECDSA
Version	1
Timestamp	9/2/2020, 8:28:03 AM (Central European Summer Time)

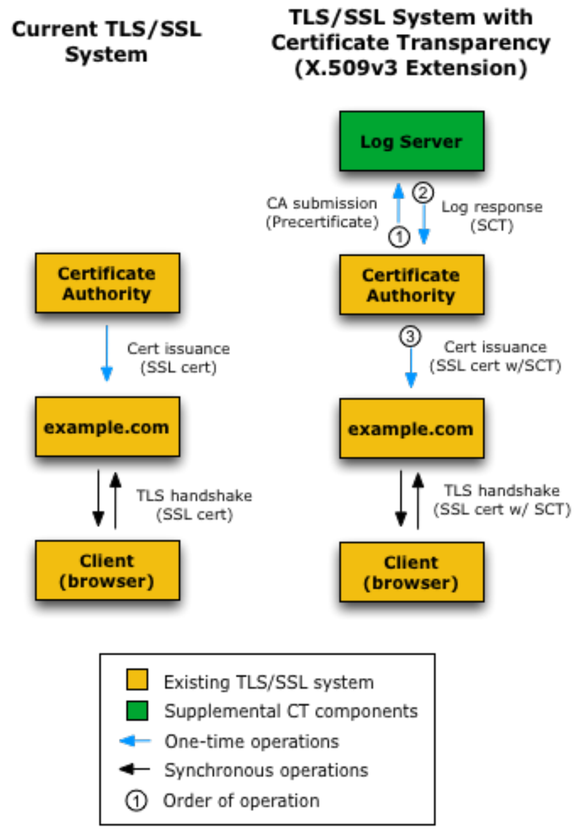


Figure 1

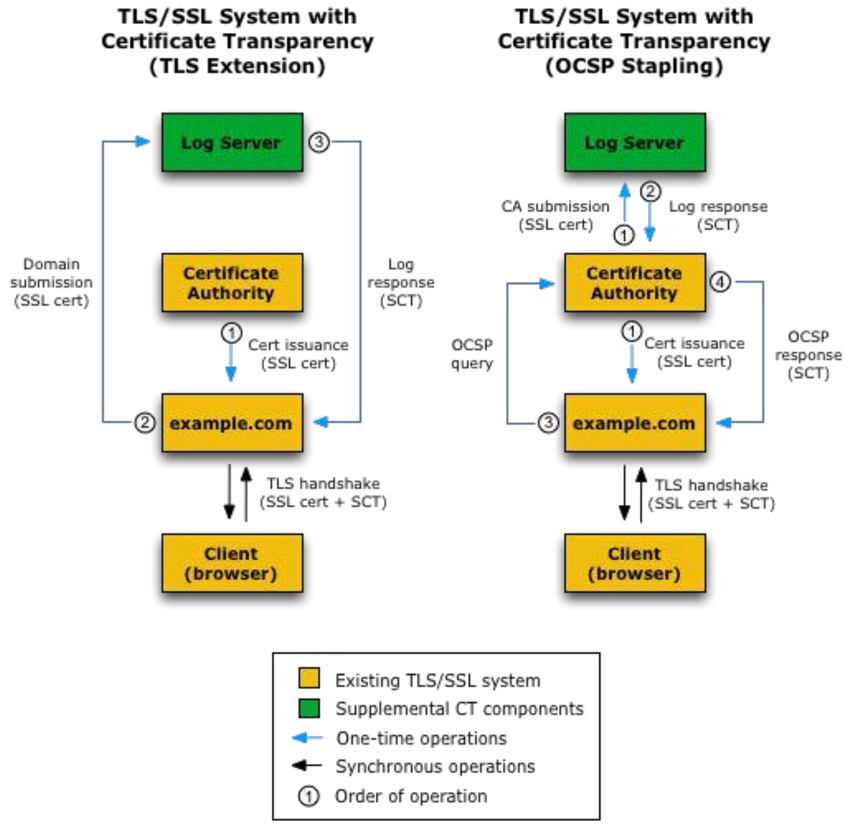
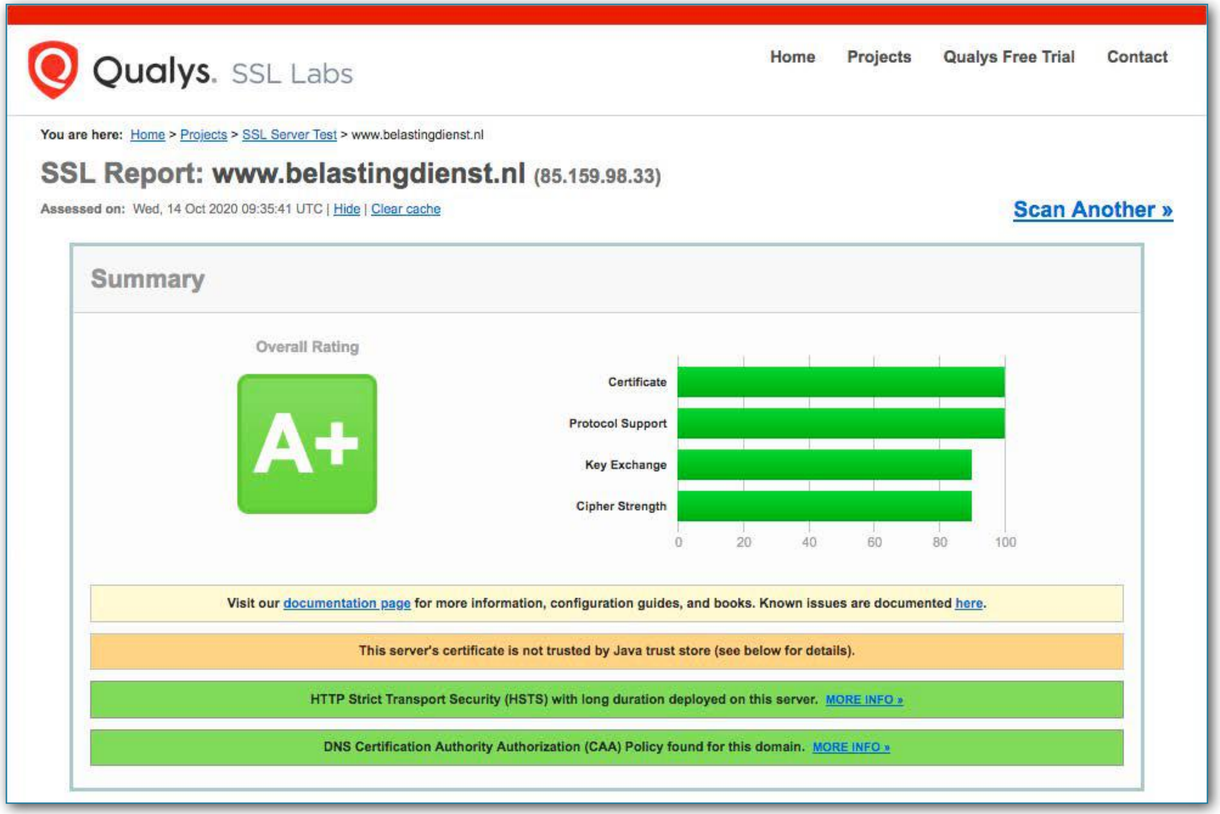
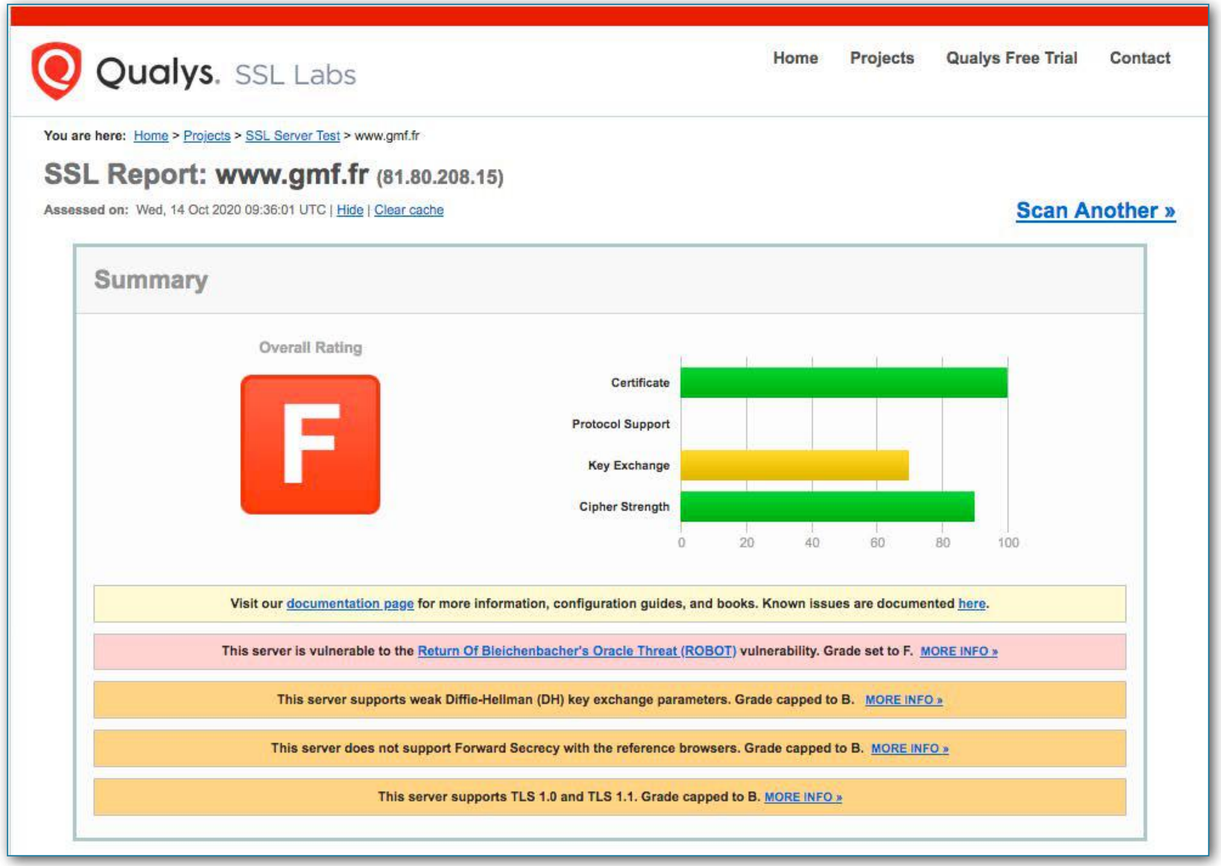


Figure 2

Test and monitor



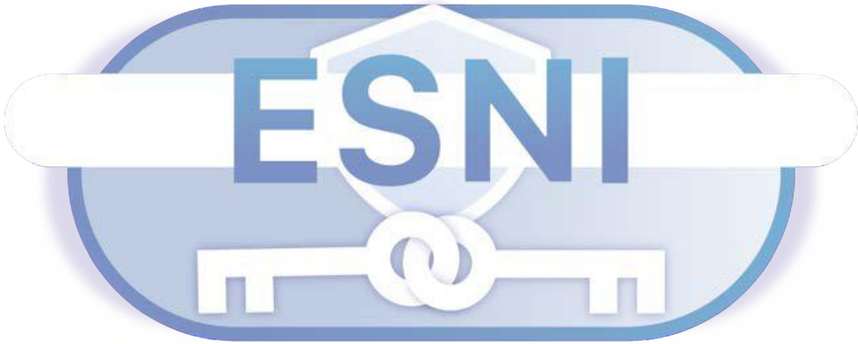
https://www.ssllabs.com/ssltest/analyze.html?d=www.belastingdienst.nl



https://www.ssllabs.com/ssltest/analyze.html?d=www.gmf.fr

Privacy: Encrypted SNI

- Not a standard yet
- Needs TLS 1.3
  - needs an encrypted Certificate handshake message
- Public key provided by DNS
  - Needs DoH or DoT, otherwise name is visible in DNS
- Watch out for IP addresses

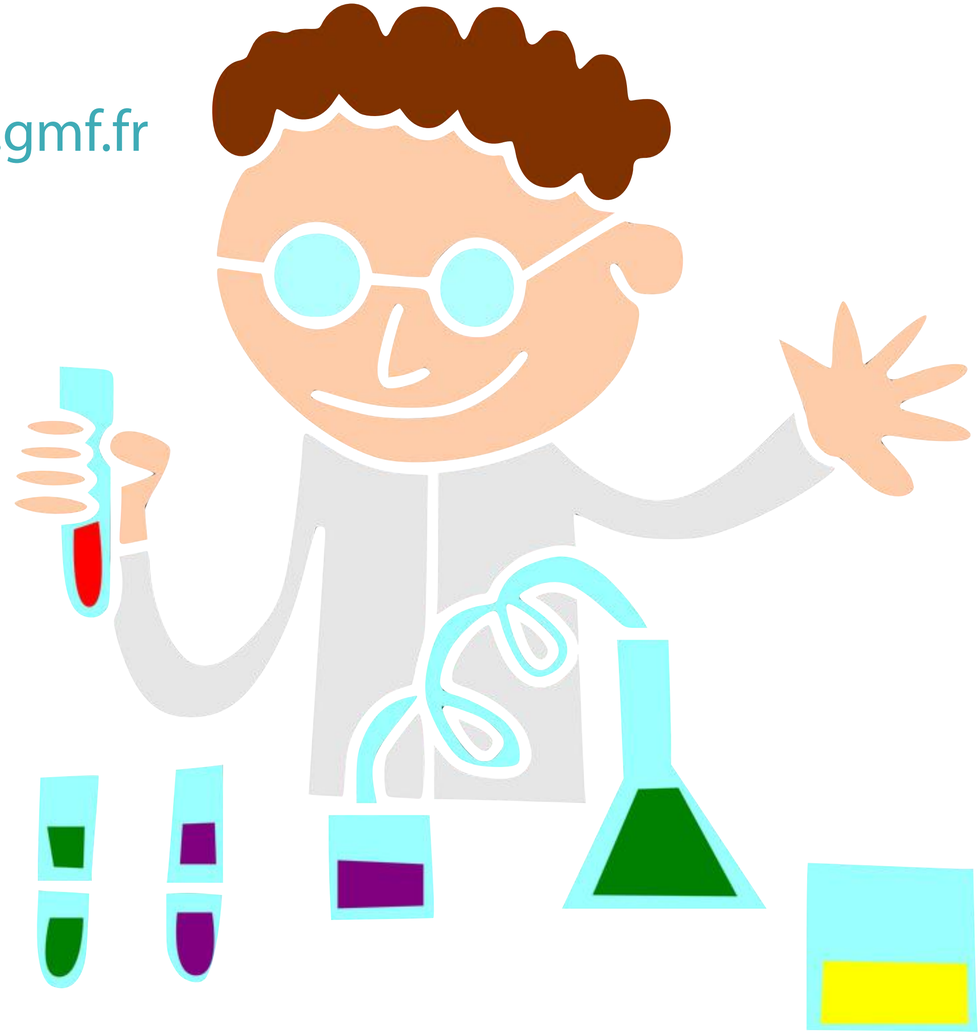


https://blog.cloudflare.com/encrypted-sni/

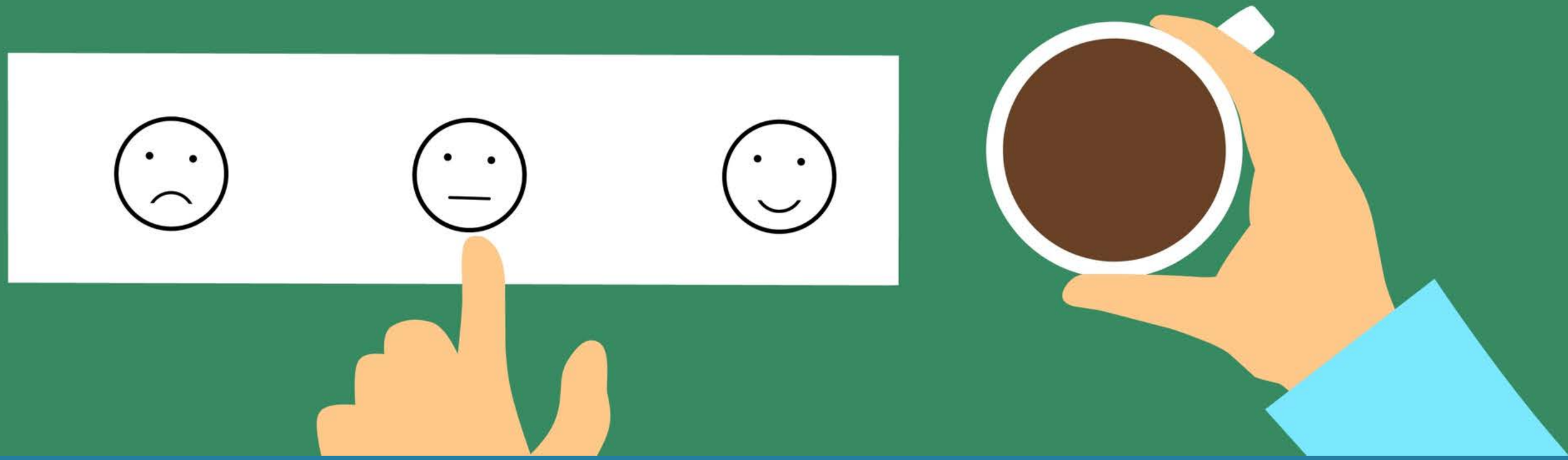
- Protection for client, but now impossible to enforce rules based on server name
  - Chinese FW blocks all TLS v1.3 traffic that uses eSNI

Demo & Exercise

- Demo
  - https://www.ssllabs.com/ssltest/analyze.html?d=www.gmf.fr
- Exercise
  - Check a site you manage (or use a lot) at:
    - https://www.ssllabs.com/ssltest/
  - Check whether it supports CAA with:
    - nslookup -q=CAA example.com
  - Check the certficate for SCT entries
  - Check whether it supports ESNI with:
    - nslookup -q=TXT _esni.www.example.com

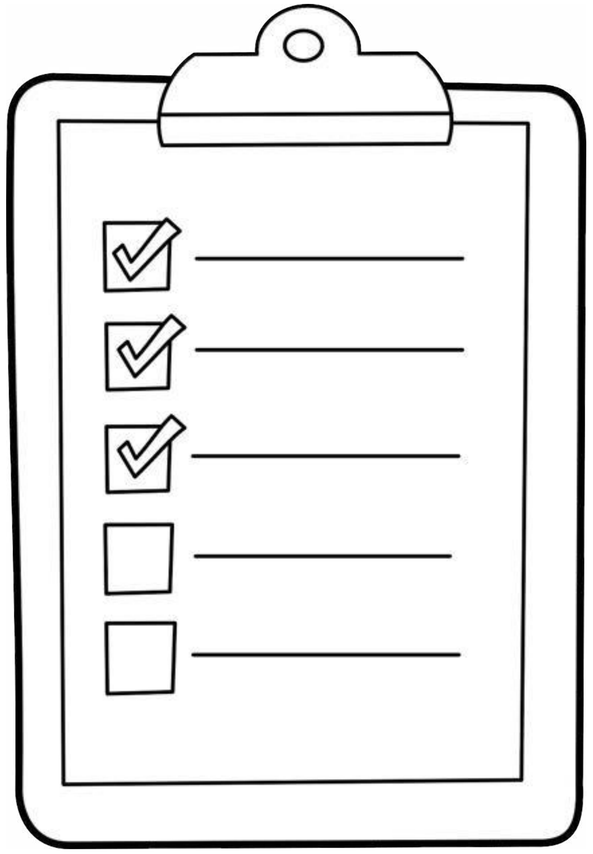


Time for a poll



Agenda

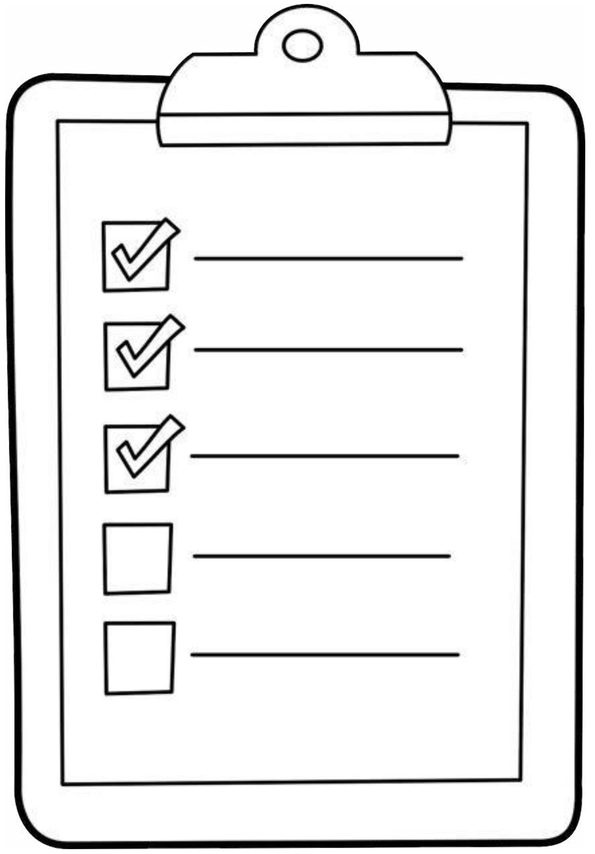
- TLS fundamentals
  - The need for TLS (and what happend to SSL?)
  - Cryptology 101 & PKI
  - Troubleshoot the TLS handshake (SSLv3 - TLSv1.2)
  - Troubleshoot the TLS handshake (TLSv1.3)
- Analysing Application Data
  - Without decryption
  - With decryption using the servers private key
  - With decryption using TLS session keys
- Creating configuration profiles for TLS analysis
- SSL/TLS vulnerabilities & privacy concerns
- **Summary and Q&A**



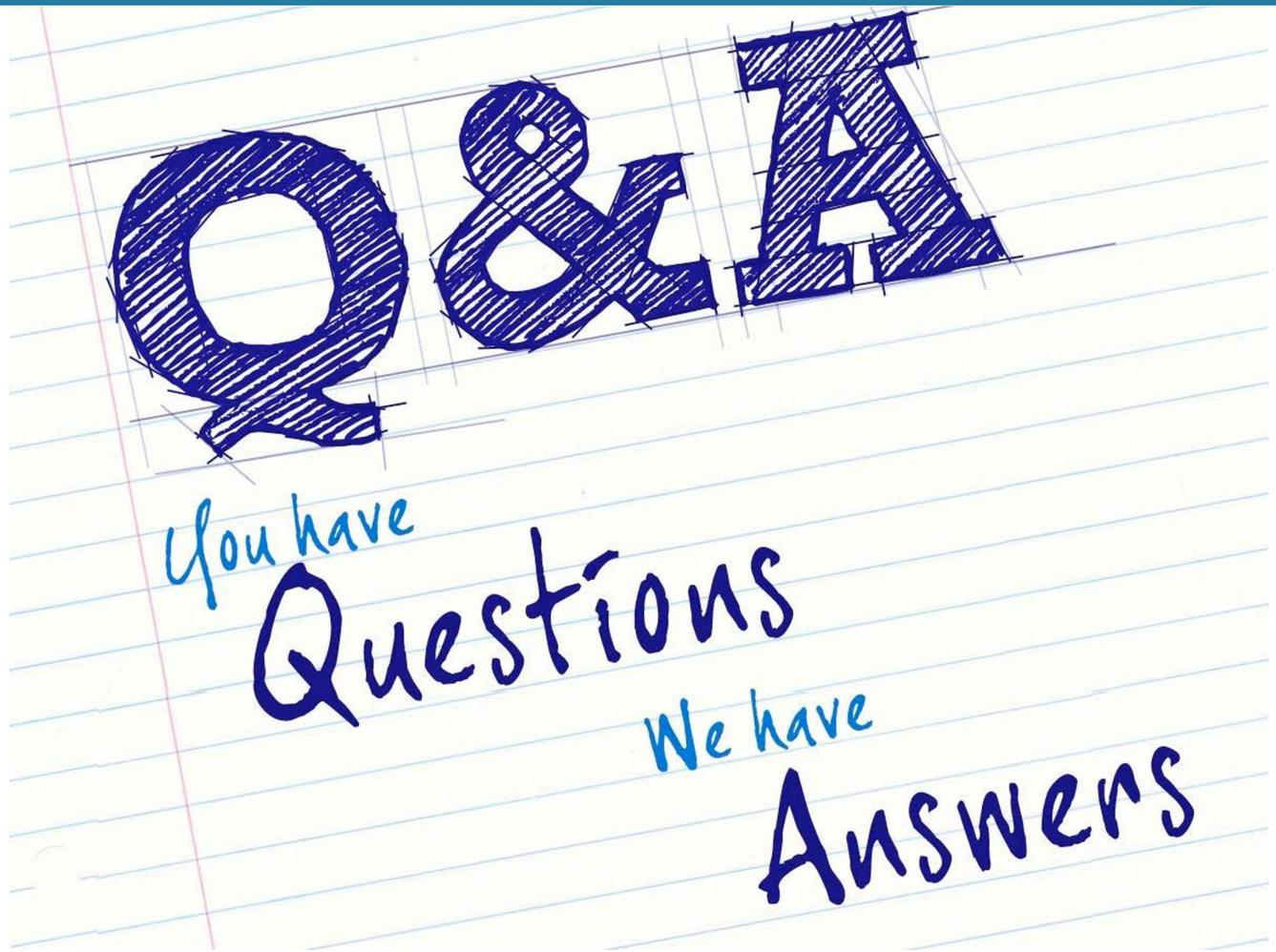
https://www.needpix.com/photo/download/94588/checklist-action-check-list-verification-tick-approved-okay-check-mark

Summary

- Get familiar with the basics of cryptography
- Analysing the TLS handshake
  - Pay attention to:
    - Root certificates in Trust Stores
    - Supported versions & ciphersuites
  - TLSv1.3 is a different beast (compared to SSLv3-TLSv1.2)
- Analysing application traffic
  - Without decryption it's limited and hard to do!
  - Decryption with private keys is possible, but fading away
  - Only decryption with TLS session keys will be possible in the (near?) future
  - Access to TLS session keys is new challenge



https://www.needpix.com/photo/download/94588/checklist-action-check-list-verification-tick-approved-okay-check-mark



Please take a minute to
fill in the survey at:

https://SYN-b.it/survey

... or scan the following QR-code to
go directly to the survey:

Links

- TLS illustrated:
  https://tls.ulfheim.net/ and https://tls13.ulfheim.net/
- https://www.ssllabs.com/ssl-pulse/
- Test your TLS infrastructure (Client, Server, best practives, how are others doing, etc):
  https://www.ssllabs.com/
- Historic facts on SSL and TLS
  https://www.feistyduck.com/ssl-tls-and-pki-history/
- TLSkex:
  https://www.sciencedirect.com/science/article/pii/S1742287616300081
- TeLeScope:
  https://conference.hitb.org/hitbsecconf2016ams/wp-content/uploads/2015/11/D1T1-Radu-Caragea-Peering-into-the-Depths-of-TLS-Traffic-in-Real-Time.pdf
- Using iRule on F5 to extract keys:
  https://support.f5.com/csp/article/K12783074
- Sharkfest presentations on SSL & TLS
  https://sharkfesteurope.wireshark.org/assets/presentations16eu/07.pdf & https://youtu.be/oDaDY9QCnXk
  https://lekensteyn.nl/files/wireshark-ssl-tls-decryption-secrets-sharkfest18eu.pdf
  https://sharkfestus.wireshark.org/assets/presentations19/02-26.pptx & https://www.youtube.com/watch?v=Fp_7g5as1VY
  https://lekensteyn.nl/files/wireshark-tls-debugging-sharkfest19us.pdf & https://www.youtube.com/watch?v=Ha4SLHceF6w
- Some legal reflections [dutch]:
  https://www.security.nl/posting/516591/Juridische+vraag%3A+Mag+mijn+werkgever+ssl-verkeer+decrypten+en+inspecteren+om+datalekken+te+voorkomen%3F
- From SSL to TLS, why the name change:
  http://tim.dierks.org/2014/05/security-standards-and-name-changes-in.html

Credits:
Most icons by icons8
https://icons8.com

FIN/ACK/FIN/ACK

If you still have any questions:

sake.blok@SYN-bit.nl

SYN-bit
deep traffic analysis