

SharkFest'26 Europe

Corporate encryption challenges

André Luyer

WIRESHARK

Sponsored by:
PROFITAP DEEP NETWORK OBSERVABILITY
cPacket
TCP/IP training
endace Record. Respond.

$ whoami

@AndreLuyer

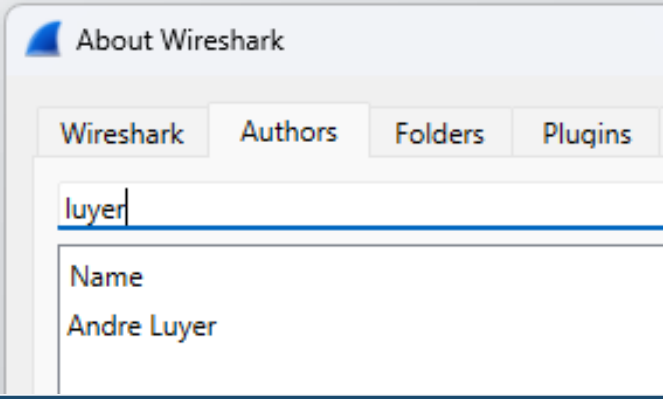


André Luyer

Performance, reliability, availability

My focus is on application performance, so mainly the top-level protocols

Nowadays 99% of the cases it is HTTP(S)



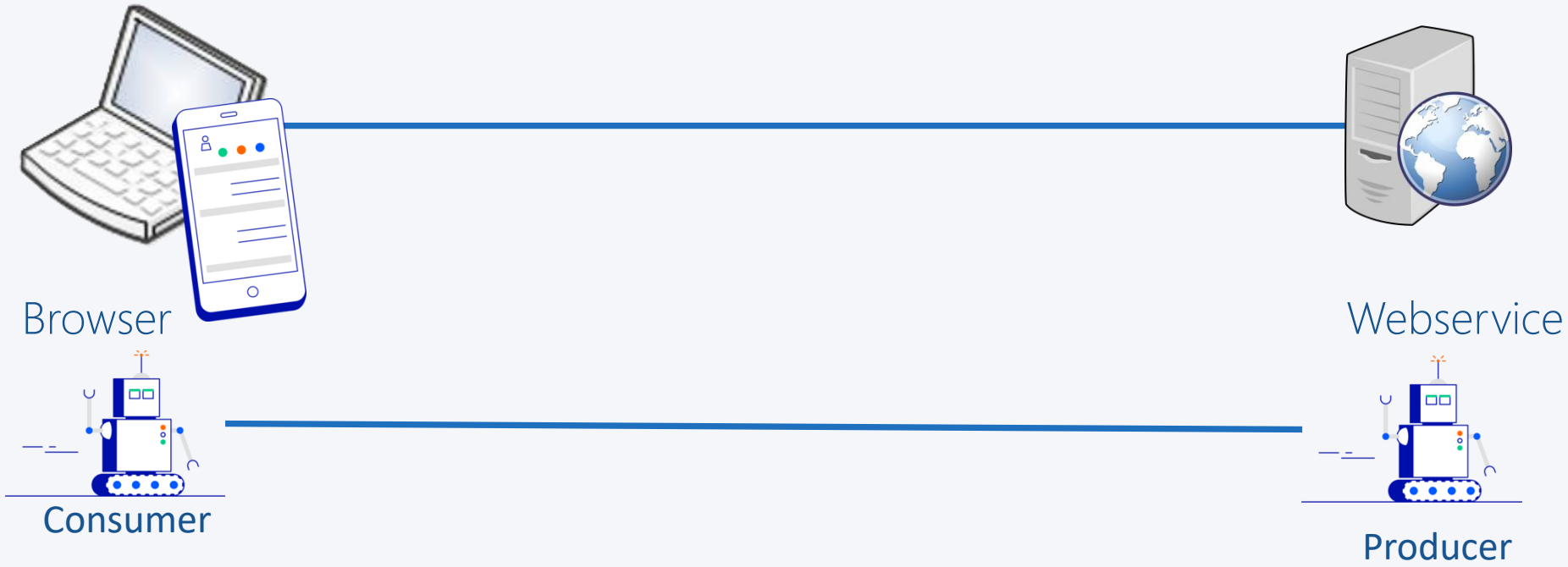
Agenda

Corporate encryption challenges

- Terminology
- TLS / SSL history
- Company policy
- Protocol ossification
- Upgrade to TLS v1.3, what can go wrong
- Post Quantum Cryptography (PQC)
- Mutual-TLS (mTLS)
- Certificates and expiry issues
- Early data feature

Terminology

Transport Layer Security (TLS) best known in combination with HTTP

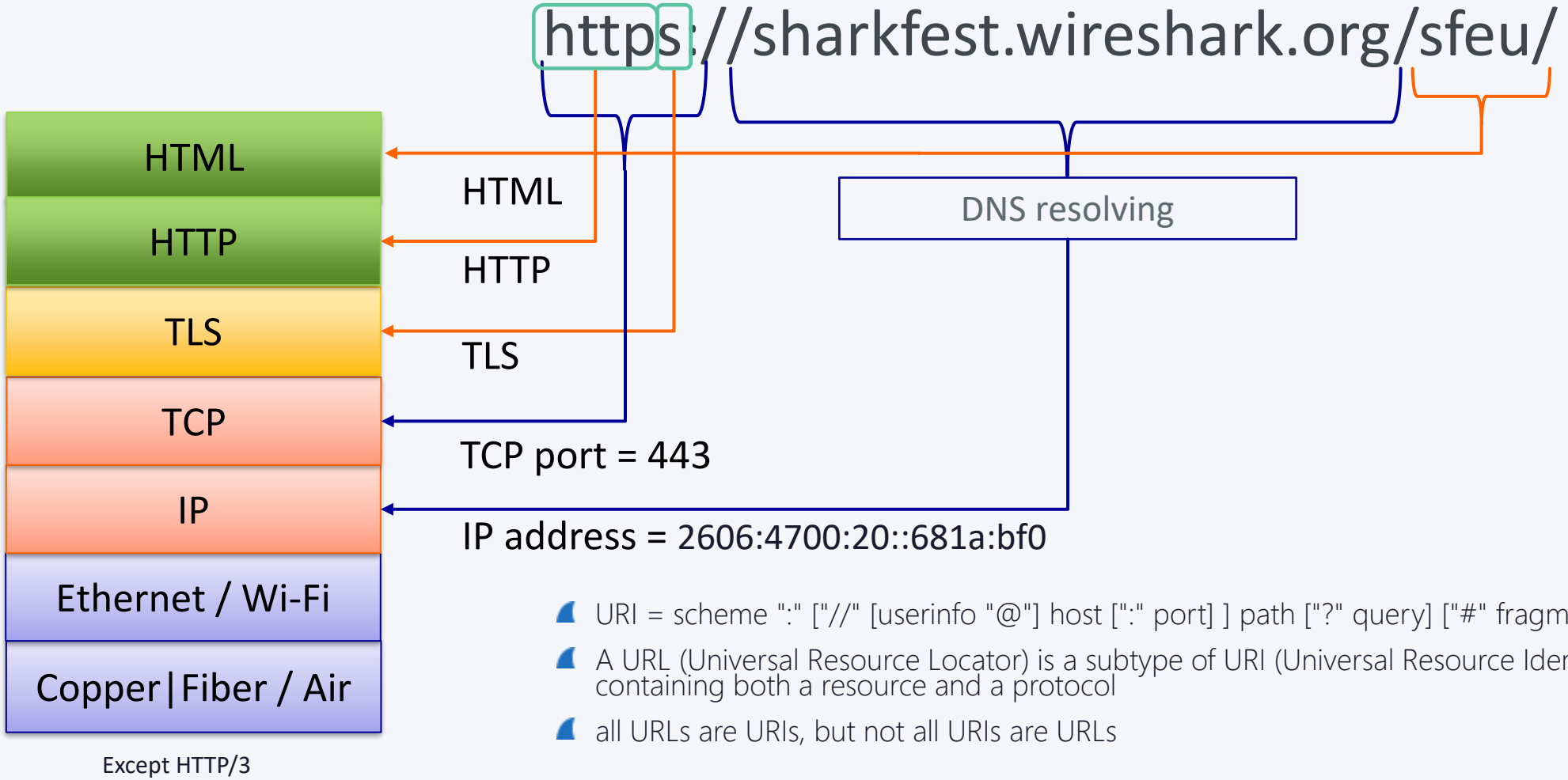


Client Server

TLS protocol described in RFC 9846 (https://www.rfc-editor.org/info/rfc9846/)

Uses the terms ‘client’ and ‘server’

URL to network protocol stack

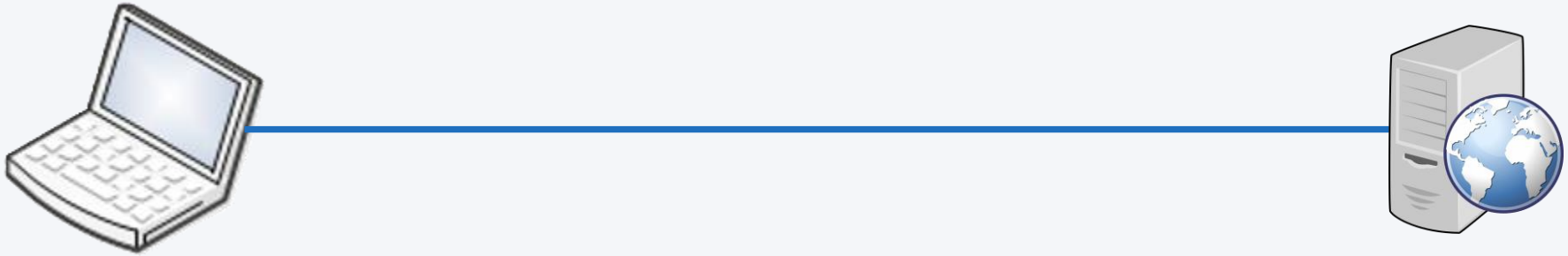


Except HTTP/3

- URI = scheme ":" ["//" [userinfo "@"] host [":" port]] path ["?" query] ["#" fragment]
- A URL (Universal Resource Locator) is a subtype of URI (Universal Resource Identifier) containing both a resource and a protocol
- all URLs are URIs, but not all URIs are URLs

Purpose

Transport Layer Security (TLS)



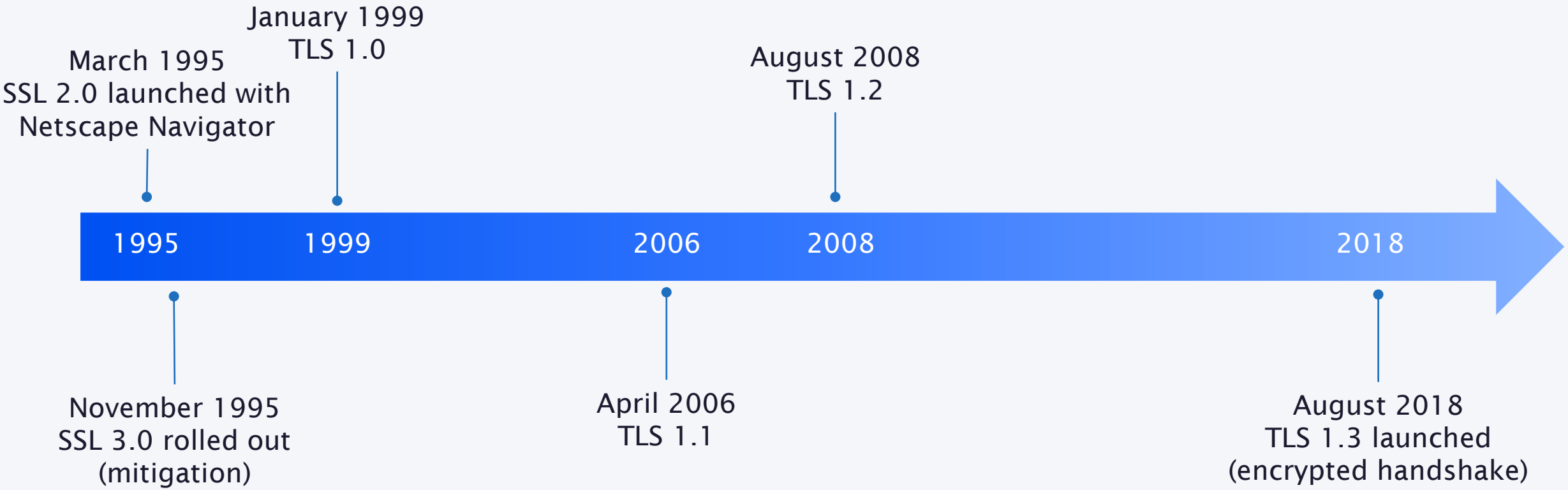
Client

Server

TLS uses 3 features:

1. Encryption of the application data in transit (duh)
2. Authentication, using certificates
3. Ensure integrity and authenticity of a message (using HMAC)

History


March 1995
SSL 2.0 launched with
Netscape Navigator
January 1999
TLS 1.0
August 2008
TLS 1.2
1995
1999
2006
2008
2018
November 1995
SSL 3.0 rolled out
(mitigation)
April 2006
TLS 1.1
August 2018
TLS 1.3 launched
(encrypted handshake)

History

Protocol	Published	Status
TLS 1.3	2018	In use
TLS 1.2	2008	In use
TLS 1.1	2006	Deprecated in 2021
TLS 1.0	1999	Deprecated in 2021
SSL 3.0	1996	Deprecated in 2015
SSL 2.0	1995	Deprecated in 2011
SSL 1.0	Unpublished	Unpublished

Company policy

- Use TLS v1.3
- TLS v1.2 still allowed
- TLS v1.1 and older not allowed (deprecated!)
- Communication over public network must be PQC ready

- Next: PQC enabled internally

  - All servers that support TLS must have TLS 1.3 enabled and verified in production environments.
  - Legacy TLS versions must be disabled, except where an explicit risk acceptance is in place.
  - All applications support post-quantum secure TLS key exchange by enabling the hybrid TLS 1.3 key exchange mechanism X25519MLKEM768 on server-side endpoints, while maintaining compatibility with existing clients and services.

Manual check if TLS v1.3 & PQC is used

Checks server-side support

From the command line using `curl`:

Version ≥ 1.3

```
curl --verbose --tls-max 1.3 --tlsv1.3 https://www.wireshark.org/ |& head -40
```

Sample output:

```
SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384 / X25519MLKEM768 / id-ecPub
```

Similar test using `openssl`:

Version = 1.3

```
openssl s_client -quiet -brief -tls1_3 -connect www.wireshark.org:443 <<<Q
```

Outputs:

```
Protocol version: TLSv1.3
Ciphersuite: TLS_AES_256_GCM_SHA384
...
Negotiated TLS1.3 group: X25519MLKEM768
```

ML-KEM = Module-Lattice Key-Encapsulation Mechanism

Using network capture

- Create a network capture – *decryption not required!*
  e.g., using dumpcap, tcpdump, etc.
- Open file in Wireshark
- Display filter to check if TLS v1.3 is used:
  `tls.handshake.type == 2 && tls.handshake.extensions.supported_version == 0x0304`

Server Hello

TLS v1.3

- For PQC usage add:
  `&& tls.handshake.extensions_key_share_group in { 4587..4589 }`
  for *recommended only* … `== 4588`

X25519MLKEM768

- Using TShark:

```
tshark -n –T fields -e _ws.col.Destination -e _ws.col.Source -e tcp.srcport
-e _ws.col.Protocol -e 'vals(tls.handshake.ciphersuite)’
-e 'vals(tls.handshake.extensions_key_share_group)' -Y 'tls.handshake.type==2 &&
tls.handshake.random !=
cf21ad74e59a6111be1d8c021e65b891c2a211167abb8c5e079e09e2c8a8339c' -r $file |
sort | uniq -c | sort -nr
```

No Hello Retry Request

Capture filter

- Only Server Hello needed
- Capture filter:

```
tcp[tcp[12]>>2:4] & 0xFFFFFC00 = 0x16030000 && tcp[(tcp[12]>>2)+5] < 3
```

- First 2 bytes plus 6 bits of TCP data starts with 0x1603 and 0, meaning “all TLS handshakes, all versions”
- And 6th byte of TCP data < 3, meaning “Client Hello (1) or Server Hello (2)”

- To improve analysis in Wireshark add:

```
tcp[tcpflags] & 7 != 0 || ( … )
```

- Or TCP flags SYN, FIN or RST set, meaning “include start and end of TCP sessions”

Java client check TLS v1.3 & PQC is used

- Java demo

Java client check TLS v1.3 & PQC is used

```
java -Djavax.net.debug=ssl,handshake httpclientdemo.java |& awk '/^"ServerHello":/,/^}/ { if (/^\s*0/) next; print }'
"ServerHello": {
  "server version"      : "TLSv1.2",
  "random"              : "9F90BCC669D0C5D9615E37CA00E41EFFA5261FE3D3C8CB7E4D515851C15339CB",
  "session id"          : "E1528BCAC68D4623E3A0E66B4E744BD25A0B13253692B234836B0EB0D93C67FD",
  "cipher suite"        : "TLS_AES_256_GCM_SHA384(0x1302)",
  "compression methods" : "00",
  "extensions"          : [
    "supported_versions (43)": {
      "selected version": [TLSv1.3]
    },
    "key_share (51)": {
      "server_share": {
        "named group": secp256r1
        "key_exchange": {
        }
```

Protocol ossification



Protocol ossification is the loss of flexibility, extensibility and evolvability of network protocols. This is largely due to middleboxes that are sensitive to the wire image of the protocol, and which can interrupt or interfere with messages that are valid but which the **middlebox does not correctly recognize**. This is a violation of the end-to-end principle. Secondary causes include inflexibility in endpoint implementations of protocols.

Source Protocol ossification – Wikipedia

Middleboxes: firewalls, load balancers, proxies, caches, CDNs, …

TLS v1.3 works around protocol ossification by using:

- Version field no longer used and given a fixed value (TLS v1.2) – also works around bugs
- Handshake looks like a TLS v1.2 resumed session
- Encrypted handshake looks like (opaque) application data
- Hello Retry Request message looks like a normal Server Hello

Upgrade to TLS v1.3, what can go wrong

Case 1: Just enabling TLS v1.3 broke the application

next to v1.2

Normal approach:

1. Enable the new version
2. See if new version is used correctly
3. Disable old version



Normal behavior for TLS is to use highest/best version both client and server can agree on.

So TLS v1.3 when available.

Just enabling TLS v1.3 broke the application

What happened:

- Server disconnects after receiving HTTP POST
- Server logs this as HTTP 403

So normal handshake + HTTP request,

then failure – server dropped connection

HTTP POST

Protocol	Info
TLSv1.3	Client Hello
TLSv1.3	Server Hello, Change Cipher Spec, Application Data
TLSv1.3	Application Data
TLSv1.3	Application Data
TLSv1.3	Application Data, Application Data
TLSv1.3	Change Cipher Spec, Application Data, Application Data
TLSv1.3	Application Data
TLSv1.3	Application Data
TCP	443 → 47912 [FIN, ACK] Seq=7001 Ack=628 Win=64128 Len=0
TCP	443 → 47912 [RST] Seq=6977 Win=0 Len=0
TCP	47912 → 443 [FIN, ACK] Seq=739 Ack=7002 Win=65280 Len=0
TCP	443 → 47912 [RST] Seq=7002 Win=0 Len=0
TCP	443 → 47912 [RST, ACK] Seq=7002 Ack=740 Win=0 Len=0

Case 1: Using TLS v1.2 works

Info	Info
Client Hello	Client Hello
Server Hello	Server Hello
Certificate, Server Key Exchange, Server Hello Done	Certificate, Server Key Exchange, Server Hello Done
Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message	Client Key Exchange, Change Cipher Spec, Finished
New Session Ticket, Change Cipher Spec, Encrypted Handshake Message	New Session Ticket, Change Cipher Spec, Finished
Application Data	POST /t HTTP/1.1
Encrypted Handshake Message	Hello Request
Encrypted Handshake Message	Client Hello
Encrypted Handshake Message	Server Hello
Encrypted Handshake Message	Certificate
Encrypted Handshake Message	Server Key Exchange
Encrypted Handshake Message, Encrypted Handshake Message	Certificate Request, Server Hello Done
Encrypted Handshake Message, Encrypted Handshake Message, Change Cip	Certificate, Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
Encrypted Alert	Alert (Level: Fatal, Description: Handshake Failure)

Case 1: root cause

Removed functionality in TLS v1.3

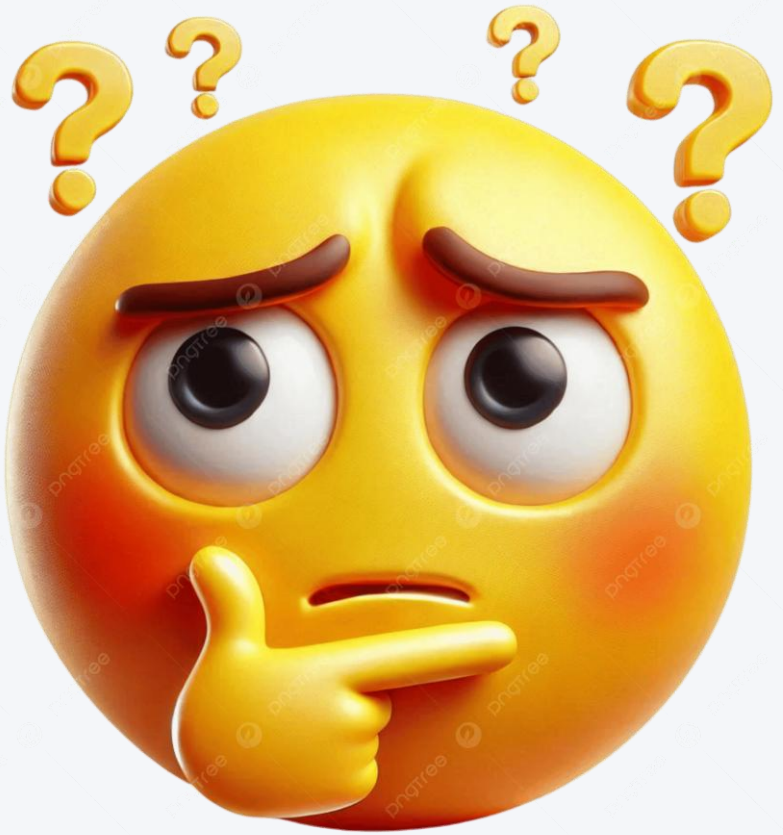
- It is no longer possible to start a new TLS handshake once a handshake is completed.
  The Hello Request message has been removed.
  This feature in v1.2 and older could be used when:
  - To renew the encryption for long running sessions – as in many hours
    In practice this was hardly used and a reliable alternative is by setting up a new connection instead.
  - Step up authentication. For example, when Mutual-TLS is required based on the URL used.

- Older ciphers are removed from the set of ciphers.
  Those that remain are all Authenticated Encryption with Associated Data (AEAD) algorithms.

- Superfluous messages in the handshake were removed,
  such as Change Cipher Spec – except when needed for middlebox compatibility.

Case 2: Disabling TLS v1.2 broke the application

When no legacy applications are in use, we can safely turn off TLS ≤ v1.2, right?



Disabling TLS v1.2 broke the application

What happened:

- Working fine until TLS v1.2 was disabled
- All connections refused

Cause:

- For a successful connection, both client and server must support at least 1 mutual method:
  - TLS version
  - Cipher suite
  - Key share group (ML-KEM for PQC)

When there is no match for TLS v1.3, but TLS v1.2 cipher suite does match, then TLS v1.2 is used

Cipher suites

- TLS ≤ v1.2
  - TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xC02C)
  - Cipher suite is actually a 16-bit number
  - Root cause:
    When cipher suit is set in config, thus overruling the default TLS named groups, enabling TLS v1.3 is not enough.
    You also need to add the TLS v1.3 cipher suits
  - By default, JDK 27 enables Post-Quantum Traditional (PQ/T)
- TLS v1.3
  - TLS_AES_256_GCM_SHA384 (0x1302)
  - New cipher suites only work in TLS v1.3
  - They do not specify the certificate type or the key exchange mechanism
  - TLS_CHACHA20_POLY1305_SHA256 (0x1303)
    TLS_AES_128_GCM_SHA256 (0x1301)
    ~~TLS_AES_128_CCM_SHA256 (0x1304)~~
    ~~TLS_AES_128_CCM_8_SHA256 (0x1305)~~

Cipher Suite vs Key Share

Feature	Cipher Suite	Key Share
Nature	A policy/menu of approved algorithms.	A live cryptographic value (public share).
When used	Negotiated at the very start of the handshake to decide what rules to follow.	Exchanged during the handshake to actually compute the session key.
TLS 1.3 behavior	Only specifies encryption and hashing (e.g., AES-256-GCM, SHA384).	Handled via the key_share extension, independent of the cipher suite choice.

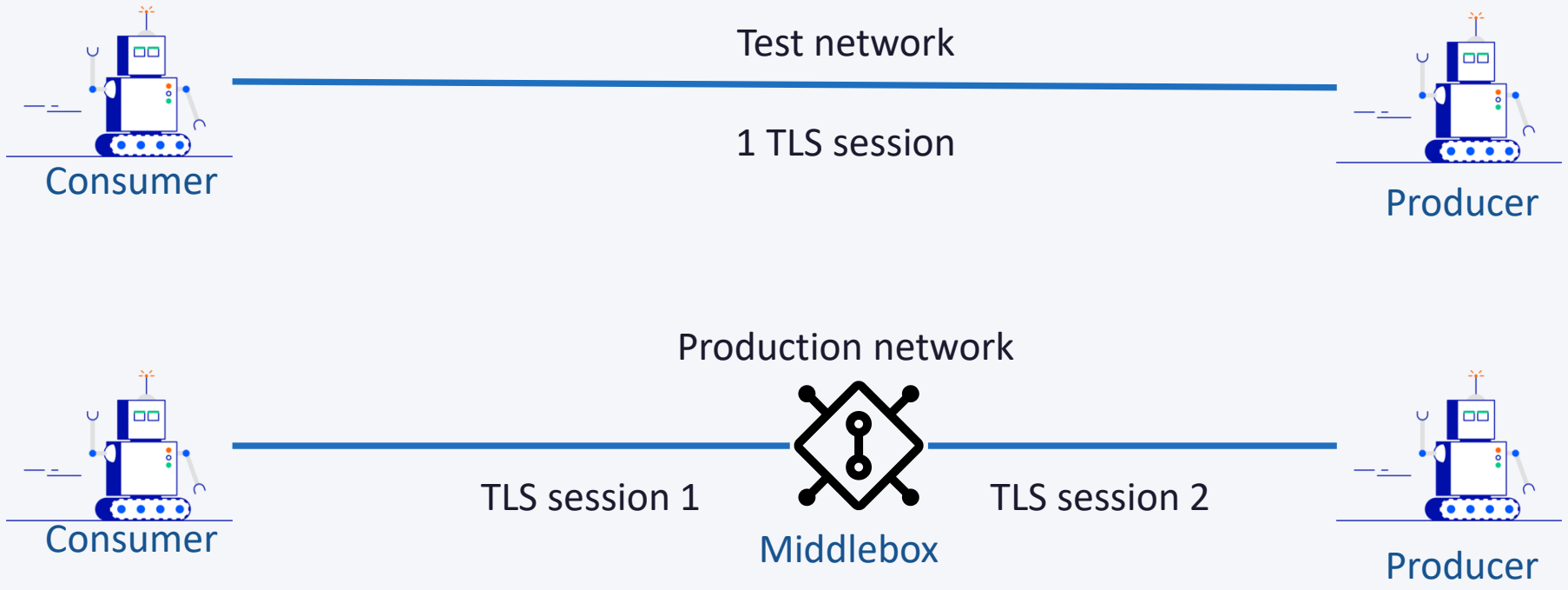
Case 3: Disabling TLS v1.2 broke the application again



Disabling TLS v1.2 broke the application again

What happened:

- App was tested, working fine
- Failure in production



Case 4: TLS connection closed unexpectedly after handshake



TLS connection closed unexpectedly

What happened:

- TLS handshake works fine
- Connection remains idle, odd but no error
- Server drops connection unexpectedly (after timeout)



Delta time	Info
0.001552000	Client Hello
0.001843000	Server Hello
0.004029000	Certificate, Server Key Exchange
0.000001000	Certificate Request, Server Hello Done
0.022122000	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec
0.000001000	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
20.0161940…	Encrypted Alert
85.0225860…	Application Data

TLS connection closed unexpectedly

While TLS session itself looks fine,

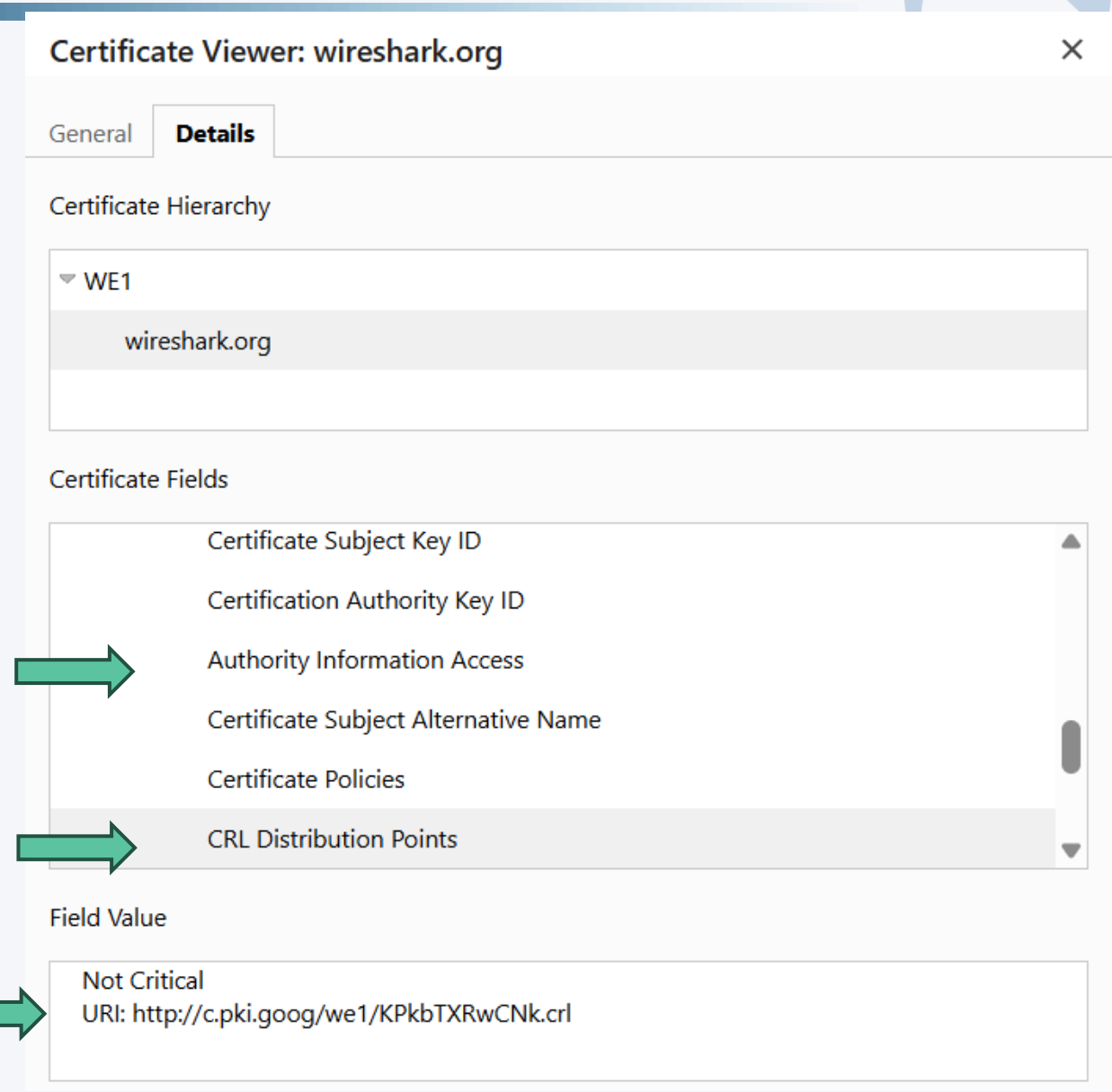
there were failing connection attempts over port 80 (http) in parallel

Delta time	Info
0.001552000	Client Hello
0.001843000	Server Hello
0.004029000	Certificate, Server Key Exchange
0.000001000	Certificate Request, Server Hello Done
0.022122000	Certificate, Client Key Exchange, Certificate Verify, Change Cip
0.000001000	New Session Ticket, Change Cipher Spec, Encrypted Handshake Mess
0.000000000	59138 → 80 [SYN, ECE, CWR] Seq=0 Win=64240 Len=0 MSS=1460 WS=256
1.003498000	[TCP Retransmission] 59138 → 80 [SYN, ECE, CWR] Seq=0 Win=64240
2.000015000	[TCP Retransmission] 59138 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=
4.000631000	[TCP Retransmission] 59138 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=
0.000000000	59144 → 80 [SYN, ECE, CWR] Seq=0 Win=64240 Len=0 MSS=1460 WS=256
8.012046000	[TCP Retransmission] 59138 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=
1.014285000	[TCP Retransmission] 59144 → 80 [SYN, ECE, CWR] Seq=0 Win=64240
1.999990000	[TCP Retransmission] 59144 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=
20.0161940…	Encrypted Alert
4.006779000	[TCP Retransmission] 59144 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=
8.000665000	[TCP Retransmission] 59144 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=
85.0225860…	Application Data

TLS connection closed unexpectedly

Cause:

- Certificate Revocation List (CRL) Distribution Point and Authority Information Access check failed
- URL in certificate not accessible
- http (not https) blocked



Post Quantum Cryptography (PQC)



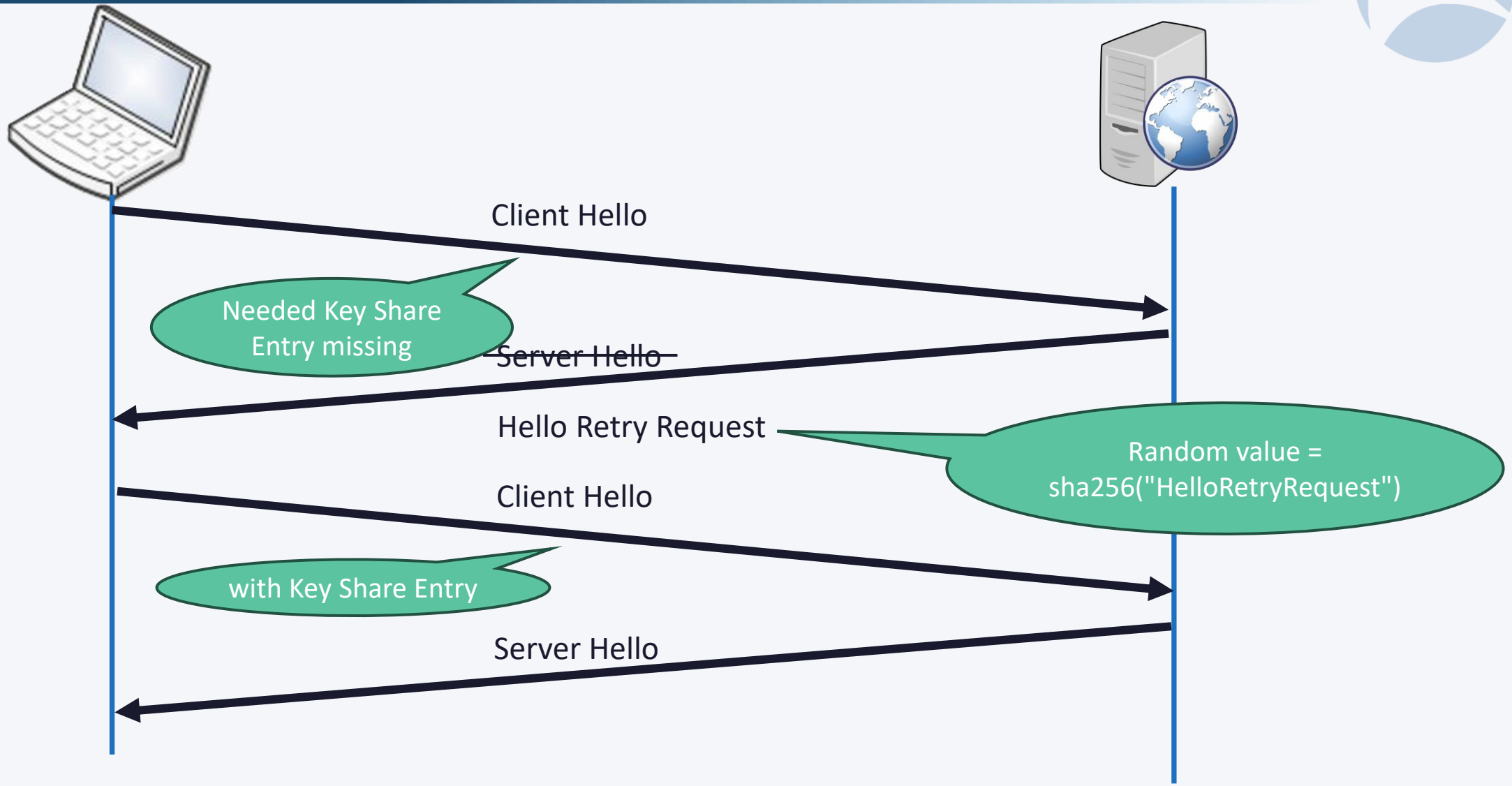
Post Quantum Cryptography (PQC)

- Required:
  - TLS v1.3
  - Key share group:

Name	Value	DTLS-OK	Recommended	Combining
X25519MLKEM768	4588 (0x11EC)	Y	Y	X25519 ECDH with ML-KEM-768
SecP256r1MLKEM768	4587 (0x11EB)	Y	N	secp384r1 ECDH with ML-KEM-768
SecP384r1MLKEM1024	4589 (0x11ED)	Y	N	secp384r1 ECDH with ML-KEM-1024

- Typically combined with X25519 group by client in case server does not support PQC
- Both sides! Client & Server. Server support can be checked with command line tools
- TLS Server Hello message contains actually used method
- Obsoleted Groups X25519Kyber768Draft00 and SecP256r1Kyber768Draft00
- Performance: PQC handshake can have extra round-trip, effect small when used for many HTTP requests + TLS resumption

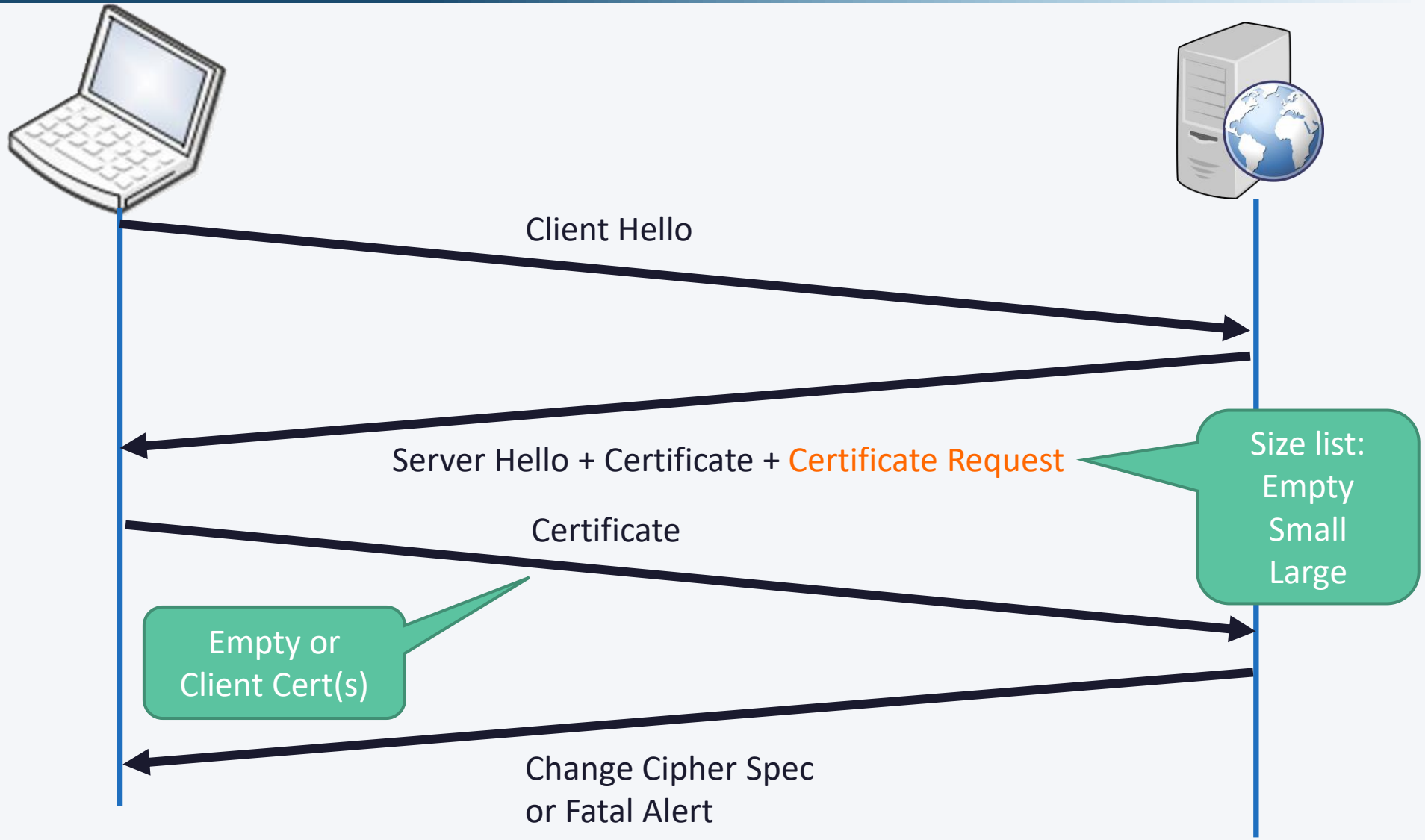
Hello Retry Request



Mutual-TLS (mTLS)



Mutual-TLS (mTLS)



Mutual-TLS (mTLS) (2)

- Certificate Request message
  - Empty list: “just send your certificate (plus intermediates), I’ll will check”
  - Small list: “you SHOULD use one of these Root CAs”
    - Most clients abort when not able to comply
  - Huge list: “just sending all root CAs I know”, effectively the same as empty list, but wasting ~15 kbyte
    - Typically a misconfiguration, but ‘works’

- Client Certificate message
  - Empty list: “I don’t have a client certificate”
    - Server continues when mTLS is configured as optional
  - Leaf plus optional intermediate certificates: “here you go”
    - Server checks the certificate(s), aborts on error

Mutual-TLS (mTLS) (3)

Filtering

For TLS ≤ v1.2:

- Not encrypted, so:
  - Certificate Request message: `tls.handshake.type == 13`
  - or Certificate message sent by client: `tls.handshake.type == 11 && tcp.srcport == <client port> && ip.src == <client addr>`

For TLS v1.3:

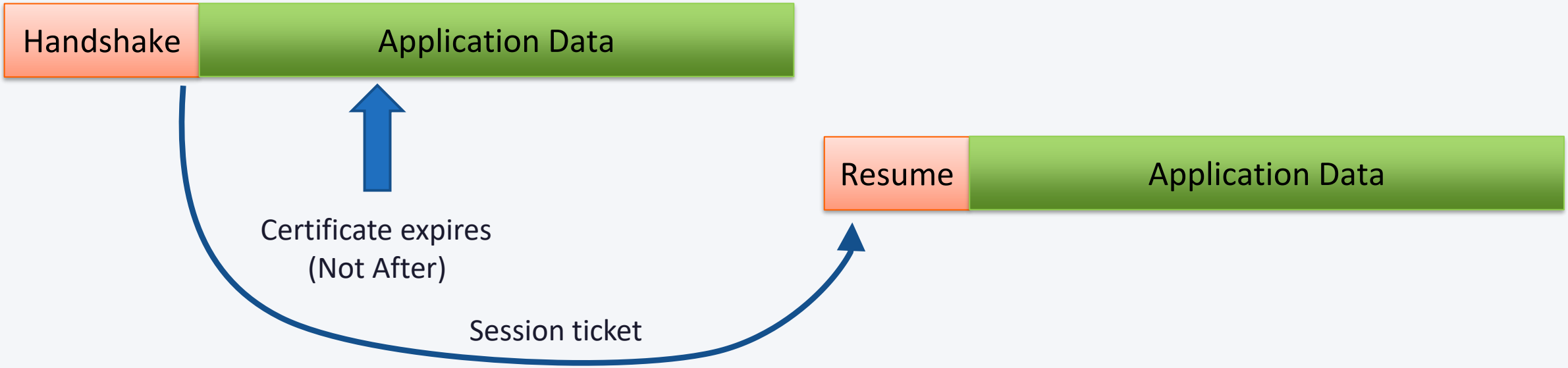
- When decrypted using session keys: same as above
- When encrypted: use heuristic approach:
  - First Application Data after Change Cipher Spec message from client is large (so not a Finished message)
  - Normally sent in the same TCP segment
  - `tls.record.content_type == 20 && tls.record.length > 100 && tcp.srcport == <client port> && ip.src == <client addr>`

Certificates and expiry



Certificates and expiry

- Certificate lifetime becomes shorter
  - Within Cloud can be 1 day or shorter
- Certificate expiry while session is active is no longer an edge-case



- What happens with TLS session when certificate expires?
- Can you resume a TLS session using *this* session ticket while the certificate has expired?
- Lifetime session ticket independent of certificate

TLS v1.3 Early data



TLS v1.3 Early data feature

- Properties:
  - Enables 0-RTT
  - Resumed only
  - Replay attacks possible (only use HTTP GET or HEAD)

- Opaque Application Data sent before Server Hello
- Filter, not depending on decryption:

```
tls.handshake.type == 1 && tls.handshake.extension.type == 42
```

- Must be enabled in webservice
  - E.g., Nginx `ssl_early_data on;`
- Otherwise `HTTP 425 Too Early` error message

Summary



Summary

- Both sides, client and server, must support the version, cipher and key share to make it work
  - Client presents lists what it is able and willing to use (Client Hello)
  - Server selects the method used in that TLS stream (Server Hello)

- When the network path is via middleboxes, multiple TLS sessions may be in use
- Using TLS v1.3 and Post Quantum Cryptography in itself is not difficult
- But check if it actually works
  - Enable logging
  - Capture & analyze traffic

- Documented at
  - TLS: https://www.rfc-editor.org/info/rfc9846/
  - Post-Quantum Traditional (PQ/T) Hybrid Key Agreement Mechanisms for TLS 1.3: https://www.rfc-editor.org/info/rfc10024/

Thank You!

WIRESHARK



Useful links – protocol descriptions

RFCs

Overview:	https://developer.mozilla.org/en-US/docs/Web/HTTP/Resources_and_specifications	
HTTP:	https://www.rfc-editor.org/info/rfc9110/	
HTTP/2:	https://www.rfc-editor.org/info/rfc9113/	
HTTP/3:	https://www.rfc-editor.org/info/rfc9114/	
TLS v1.3:	https://www.rfc-editor.org/info/rfc9846/	
PQ/T:	https://www.rfc-editor.org/info/rfc10024/	Post-Quantum Traditional (PQ/T)
TCP:	https://www.rfc-editor.org/info/rfc9293/	(First RFC 793 September 1981)
IP:	https://www.rfc-editor.org/info/rfc791/	(First RFC 760 January 1980)
DNS:	https://www.rfc-editor.org/info/rfc1035/	(First RFC 882 November 1983)

History: https://www.computerhistory.org/internethistory/
https://en.wikipedia.org/wiki/HTTP_cookie