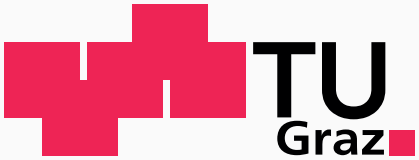


Transport Layer Security

Shaking Hands over the Web

Lena Heimberger

Secure Application Design

May 23, 2025

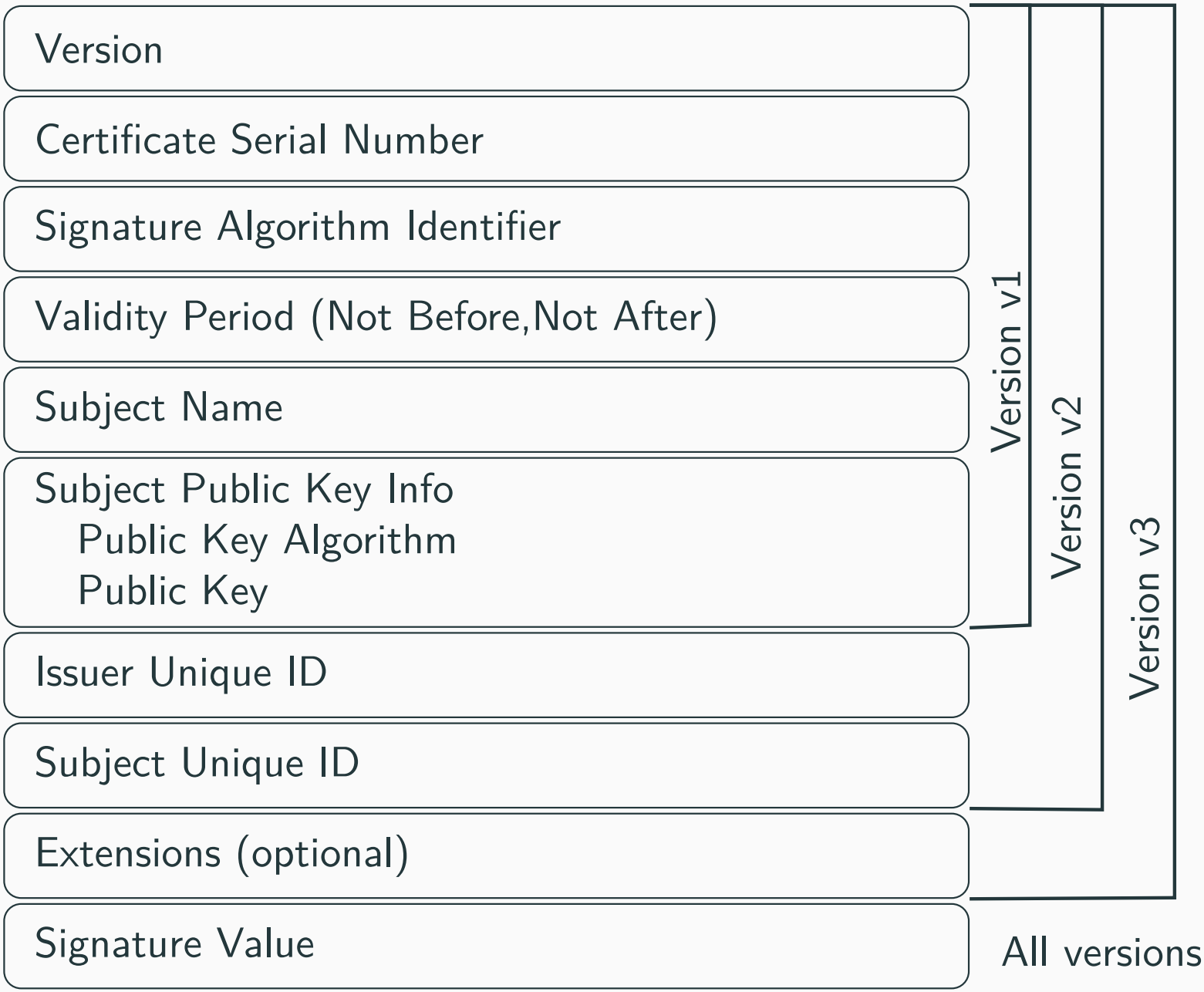
Table of Contents

Introduction

Transport Layer Security (TLS)

- adds confidentiality, authentication and integrity to the protocol
- **Confidentiality** through encryption
- **Identification and Authentication** through X.509 certificates
- **Integrity** through message digests

X.509 Certificates



TLS 1.3

- finalized in 2018
  `https://www.rfc-editor.org/rfc/rfc8446`
- removed a lot of unnecessary add-on's
  - e.g. non-AEAD symmetric algorithms and static RSA/DH cipher suites
  - prevents passive decryption attacks: each session is negotiated again
- Forward secrecy with all public-key based key exchanges, include elliptic curve algorithms
- handshake only requires one roundtrip (instead of two)
- resilient against downgrade attacks

Record Protocol

- main bridge between TLS applications and internal subprotocols
  - fragments data streams into managable blocks, consisting of 2^{14} bytes or fewer
  - protects the record with encryption
  - verifies, decrypts and reassembles received data

Record Protocol Header

- records are typed by the used subprotocol, or ContentType:
  - handshake (0x16)
  - application data (0x17)
  - alert(0x15)
  - ~~change cipher spec (0x14)~~
- version (03 04 for TLS 1.3)
- length of remaining record

Byte	+0	+1	+2	+3	+4
0	Content type	Version		Length	
5..n	Payload				
n..m	MAC				
m..p	Padding (block ciphers only)				

0x16: TLS Handshake

- starts the communication session
- agree on protocol version
- agree on used cryptographic algorithms
- establishes shared (secret) keys using public key cryptography
- authenticates server (and optionally client)
- detailed: `https://tls12.xargs.org/`

0x17: Application Data

- sent after Finished message from handshake
  MAC over entire handshake
- `early_data`: client can send data before
- unauthenticated, but fast!

0x15: Alert protocol

- indicate closure information and errors
- either a warning or fatal
- also has a level (for logging) and a description
- can indicate a number of problems:
  - unexpected message
  - handshake failure
  - bad/revoked/expired/unknown certificate
  - access denied
  - ...

Example: Closure Alerts

- enables server and client to know that a connection is ending
- introduced to avoid truncation attack:
  - blocks account logout request so user remains logged into a web service
  - when request to logout is sent, attacker injects FIN message to drop the connection
- `close_notify` from a party means they will send no more messages on this connection.

TLS 1.3

Cipher Suites

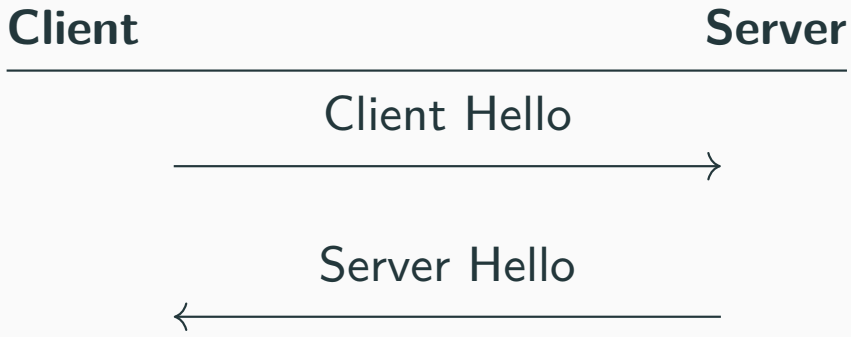
- supported cryptographic algorithms
- TLS 1.3 Cipher Suites:
  - `TLS_AES_128_GCM_SHA256` - must be implemented by a TLS-compilant application
  - `TLS_AES_256_GCM_SHA384` - should be implemented
  - `TLS_CHACHA20_POLY1305_SHA256`- should be implemented
- So, how does it look in practice?
  - `TLS_AES_256_GCM_SHA384` (`https://www.tugraz.at`)
  - `TLS_AES_128_GCM_SHA256` (`https://eprint.iacr.org/`)
  - `TLS_CHACHA20_POLY1305_SHA256` (`https://heimberger.xyz/`)
  - `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256` (`https://orf.at/`)
- ORF still uses TLS 1.2!
  it gets harder to find pages with TLS 1.2 each year

Client Hello

- Client says hi!
- uses the opportunity to indicate:
  - which TLS versions are supported
  - which cipher suites are supported
  - client random
- may include extensions
  - Server Name Indication (SNI)
    tells server we try to reach e.g. `tugraz.at` to get the appropriate certificate for the hostname
  - Supported Signature Algorithm
  - Key Share
- already encrypted premaster secret

Server Hello

- Server Hello!
- includes chosen cipher suite, server random
- performs DH key exchange
- All handshake messages after Server Hello are encrypted



Server Certificates

- SSL certificate can be used by the client to authenticate the server (i.e. it is interacting with the actual owner of the domain)
- Certificates, Signatures
- attach encrypted signature of handshake for transcript verification
- finished

Finalizing

- Client verifies signature and certificate
- generates master secret:
  PRF(*premaster_secret*, "master secret", *ClientRandom* + *ServerRandom*)
- sends finished message, but it does not count to the trips because we just assume it is done

Protocol Ossification

- ~~The natural process of bone formation.~~
- loss of flexibility due to middleboxes (and inflexible endpoint implementations)
- Ossification:
  or: the more implementations, the more likely ossification occurs
- in the beginning, they did not support TLS 1.3
- revision 22: make TLS 1.3 look like TLS 1.2
- E.g. unnecessary fields were added to ClientHello to pad to size
- TLS 1.3 advertises itself as TLS 1.2!
- Server who does not understand TLS 1.3 would not see that TLS 1.3 was used and believes it to be TLS 1.2

Demo: TLS in Wireshark

Attacks

Attack: Weak Premaster Secret[1]

- Researchers found that Netscape Navigator only used three randomness sources in 1996:
  - time of the day
  - process ID
  - parent process ID
- $\rightarrow$ guess in 25 seconds (in 1996!)

[1] Article: *How secure is the World Wide Web?*, by Ian Goldberg and David Wagner, `https://people.eecs.berkeley.edu/~daw/papers/ddj-netscape.html`

Server Random, Client Random and Premaster Secrets today

- 48 bytes of randomness
- client-generated value $\rightarrow$ we need the server random
- also: MitM replay attack $\rightarrow$ same premaster secret

Adoption Rate

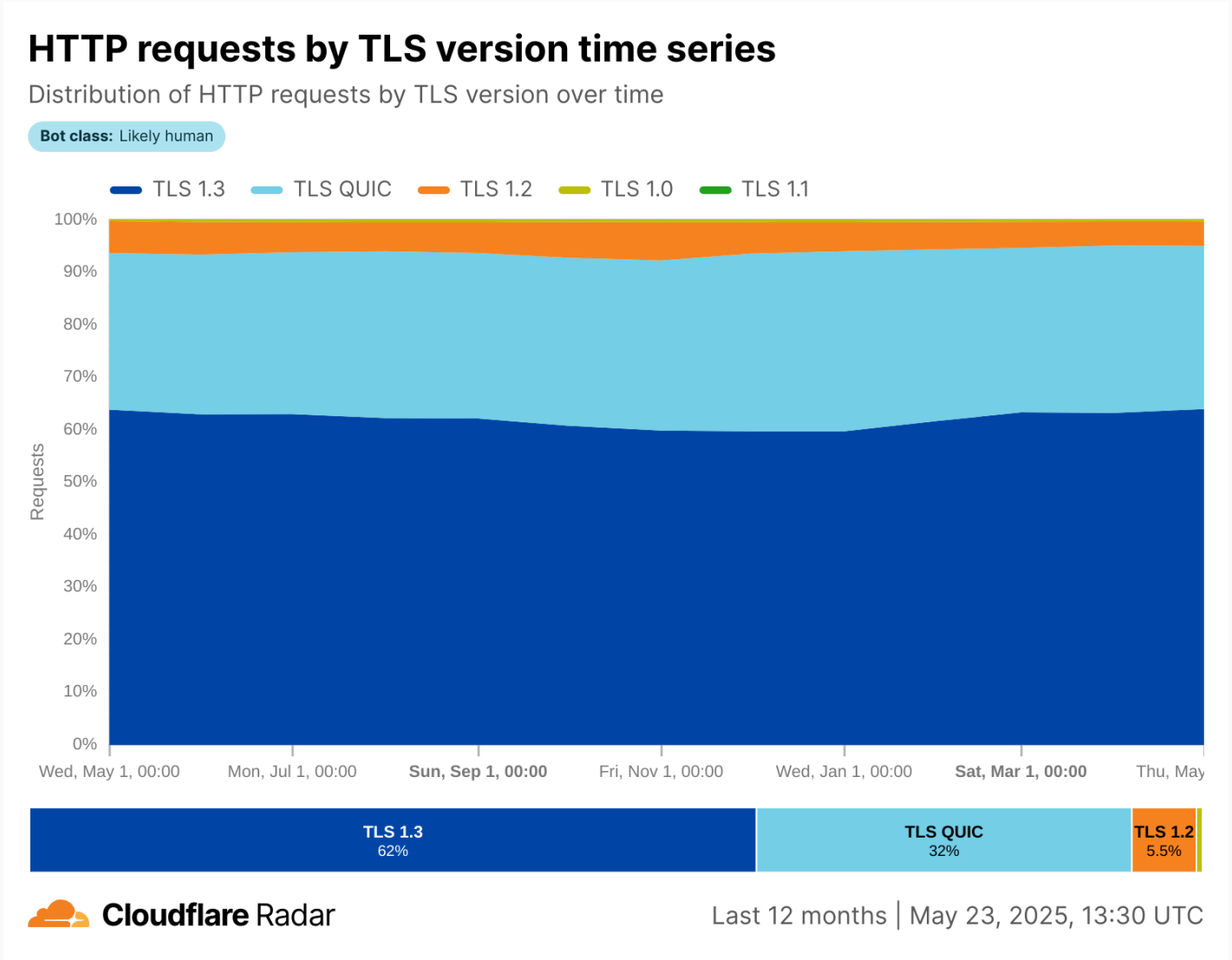


Figure 1: Adoption of TLS protocols
`https://radar.cloudflare.com/adoption-and-usage`

How can I build my own?

- `https://pkg.go.dev/crypto/tls`
- Pretty much 1:1 `https://eli.thegreenplace.net/2021/go-https-servers-with-tls/`

Extensions

Zero-Roundtrip Time Resumption

- One round can take a long time
- idea: if we have authenticated a server before, we can trust them the next time
- Use a Session Ticket, including a preshared key issued by the server to the client
- derive shared secret from the last session: resumption main secret

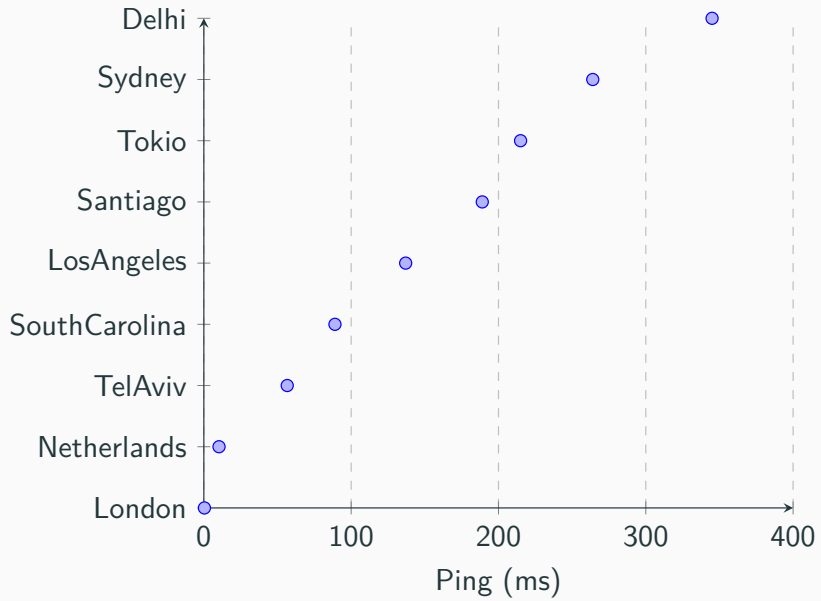


Figure 2: Ping between London and other cities, on Debian 11 using the simplest Google Cloud instance, 2023.

0-RTT Replay attacks[2]

- Use the ticket for several instances in different zones
- spec says not to handle requests that modify data until after replay window is up (handshake is finished)
- solved on application level (or with fancy cryptography)

[2] https://eprint.iacr.org/2017/082.pdf

mTLS

- mutual TLS
- Client needs a certificate and prove their identity as well!
- e.g. internal services or sign-on
- follows paradigm: public key crypto $>$ passwords

Post-Quantum TLS

Moving to Post-Quantum

- TLS needs:
  - ✔ Symmetric ciphers
  - ○ Key agreement protocols
    harvest now, decrypt later more urgent, KEMS are being rolled out
  - ○ Signatures
    less urgent, less performant than KEMs
- IETF Post-Quantum Guidance for TLS[3]
  3. Start using hybrid KEMs: once practical, do not use non-hybrid groups
  4. Do nothing for now on signatures

[3] https://www.ietf.org/archive/id/draft-farrell-tls-pqg-00.html

Adoption Rate: PQ TLS

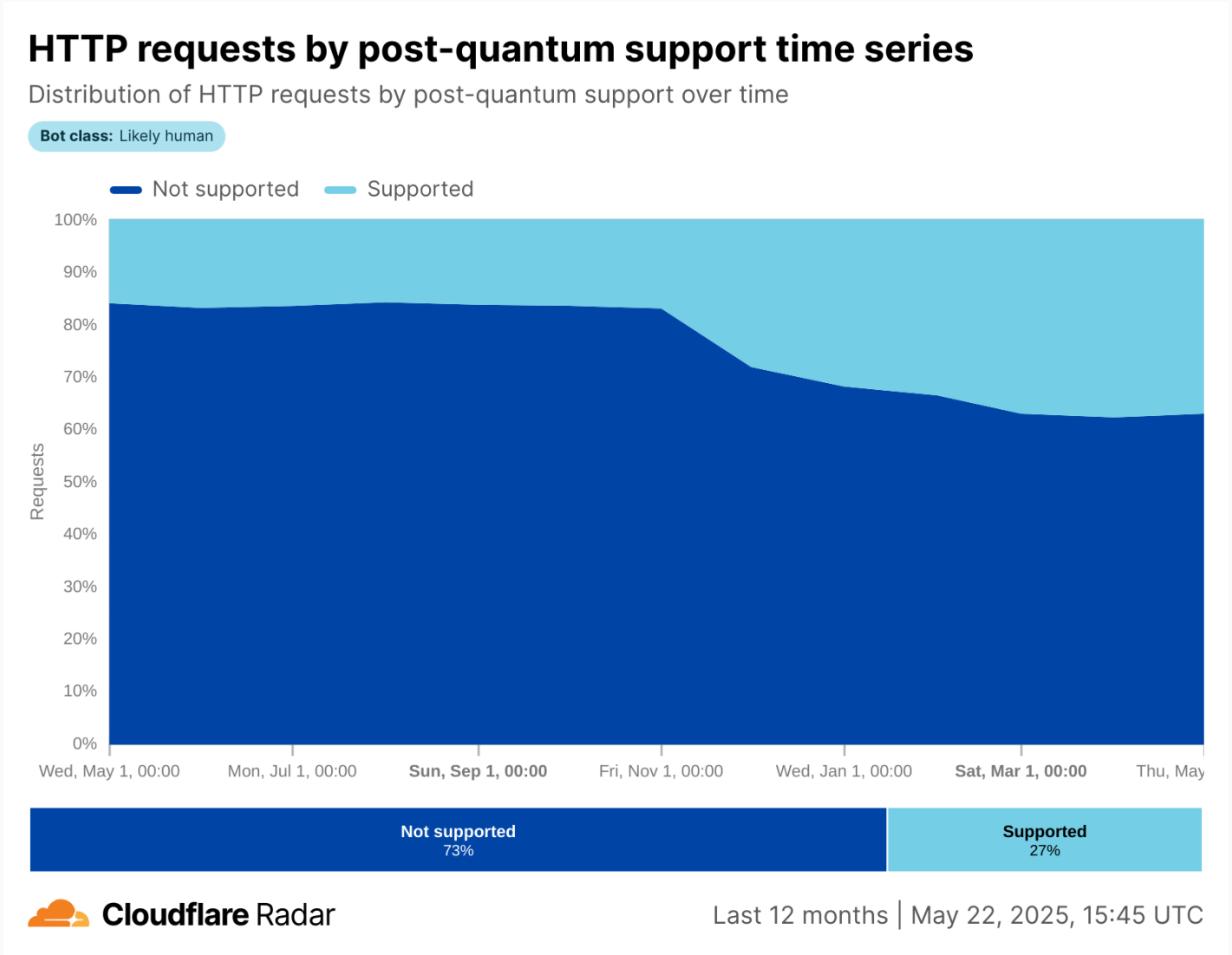


Figure 3: Post-Quantum cryptography on the Internet
`https://radar.cloudflare.com/adoption-and-usage`

KEM-TLS

KEM-TLS

- authentication in a post-quantum setting:
  - PQ-KEM (Key Encapsulation Mechanism) for encryption main difference: result is used to decrypt (decapsulate) a (session) key
  - PQ-Signature? quite big and slow compared to KEMs
- idea: to authenticate, we encapsulate a secret with the public key
- authentication works if the party we want to authenticate can decapsulte the secret
  $\rightarrow$ prove knowledge of key!
  the secret is used to derive a key and then MAC-checked
- `https://kemtls.org/`

Challenges for KEMTLS

- do you see the problem?
- TLS 1.3: single roundtrip to finish handshake!
- solution: use implicit authentication
  The client encapsulates to the long-term public key but does not wait for the MAC: just starts sending using the derived key. (unfortunate if mac check fails)
- works for 0-RTT! also may work with predistributed KEM public keys, but: statefulness may cause problems

What you should know

- ...how a TLS handshake works
- ...that TLS 1.3 is the current version
- ...what types of records exist in TLS 1.3
- ...why TLS 1.3 has forward secrecy
- ...that there are extensions, and you should be able to name some

Join Losfuzzy's Spritzerstand and win a free drink by helping Edona and I!

Demo: TLS 1.3 in Golang

References i

- illustrated TLS 1.3 connection: `https://tls13.xargs.org/`
- `https://eli.thegreenplace.net/2021/go-https-servers-with-tls/`
- again, David Wong's book *Real-World Cryptography*
- RFC!

Bonus

Encrypted Client Hello

- Server Name Indication (SNI) is transmitted in plaintext!
- predecessor: ESNI, used DNS to get server public key and encrypt SNI
- irrelevant for self-hosted, but not if you use CDN!
- ...but that leaks the server name via the DNS query
- idea: SNI is only decryptable by client or client-facing server
- realized by using details in HTTPS record
- no more adavantage in idenifiying the server than guessing (in set of possible servers)
- also other extensions, e.g. the protocol to decide which application layer protocol should be used (ALPN) are protected
- e.g. firefox deactivates it in child protection mode