TLS, mTLS, SNI, ECH, CAA, HTTPS, PKI, Zertifikate und ein bisschen PQC

FroSCon 2026 | Sven Höxter | hello@shxit.de

08/15/26

Agenda

Siehe Titel ;-)

I. Warum dieser Talk?

II. TLS und mTLS Handshake

III. PKI und Zertifikate

IV. DNS und SNI, CAA, HTTPS, ECH: Wo greifen DNS und TLS Ökosystem ineinander?

V. PQC: Hybride Verschlüsselung und Merkel Trees statt x509 Zertifikate?

Warum dieser Talk?

- TLS hat sich nach 2013 durchgesetzt.
- MTLS (Authentifizierung auf beiden Seiten) ist mit Service Mesh Installationen wie istio und API Gateways öfter Thema.
- In der Praxis viele Setup Probleme.
- Chancen, das Gesamtökosystem robuster zu machen, werden nicht genutzt.
- Zielgruppe Admins und interessierte Entwicklerinnen.*
- Bin kein Mathematiker, kein Kryptograph. Ich vereinfache und lasse Details weg.

* Der Talk war für den “System Administration” Track eingereicht.

Minimale Geschichte

- Erfunden von Netscape als SSL (Secure Sockets Layer) in den 90ern für ecommerce.
- Vor 2013 in Verwendung bei Nerds und im Checkout Prozess zur Absicherung der Zahlung, Online Banking.
- TLS x509 Zertifikate waren teure Einzelstücke und wurden alle zwei, drei Jahre von Hand rotiert (oft nachdem sie abgelaufen sind).
- Nach Snowden Enthüllungen 2013:
  - Massive Automatisierung durch ACME.
  - Kostensenkung durch Let’s Encrypt auf nahe null.
  - Senkung der Zertifikatslaufzeit von Jahren auf Monate. In Zukunft auf Wochen (45 Tage in 2028).
- Heute: Unverschlüsselter Traffic sogar im eigenen Datacenter selten.

Source: https://commons.wikimedia.org/wiki/File:Edward_Snowden-2.jpg

Attribution: Laura Poitras / Praxis Films

TLS vs OSI

Layer 7 (Application)	HTTP / IMAP / SMTP
Layer 5 (Session)	**TLS**
Layer 4 (Transport)	TCP (UDP -> DTLS, QUIC)
Layer 3 (Network)	IP

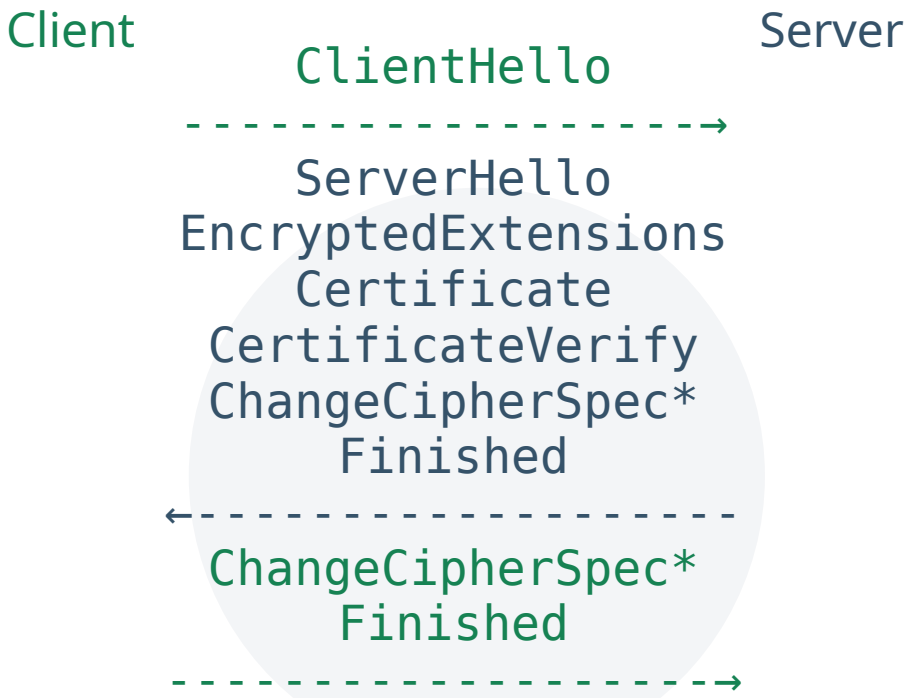
TLS Record Protocol

Bildet die “untere Schicht” aufbauend auf dem Transport Layer.

“Obere Schicht” bestehend aus (content types):

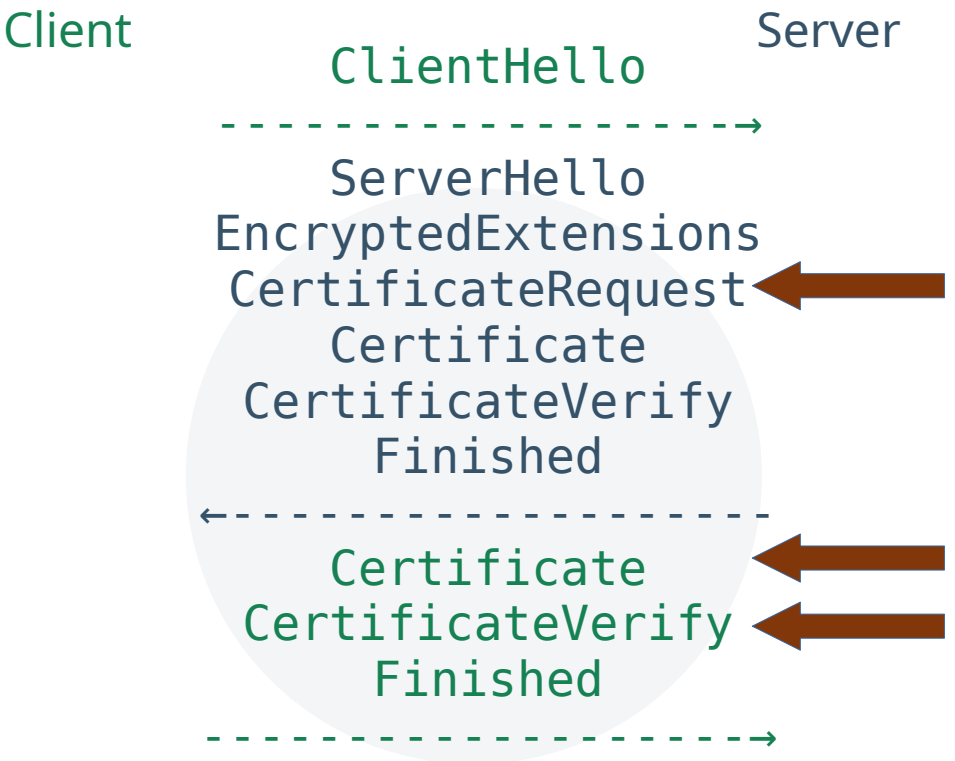
- Handshake
- Change Cipher Spec (ab TLS 1.3 nicht mehr relevant)
- Application Data
- Alert Protocol

TLS Handshake



* In TLS 1.3 nur noch aus Rückwärtskompatiblität

Mutual TLS Handshake



Handshake ClientHello

- Liste der unterstützten Cipher (ab TLS 1.3 sehr kurz).
- Extensions
  - SNI – Angefragter Servername (oder auch IP Adresse)
  - ALPN – Unterstützte Applikations Protokolle
  - ECH – Encrypted Client Hello, später mehr
  - Crypto Parameter (u.a. key_share)
  - Supported Version

Kuriosität: Drei Versionsnummern

TLS 1.3 ClientHello

1. ProtocolVersion (Record Layer): **0x0301 → TLS 1.0**
2. ClientHello Version: **0x0303 → TLS 1.2** (deprecated, Rückwärtskompatiblität)
3. supported_version extension: **0x0304 → TLS 1.3**

Unbekannte extensions werden ignoriert → neue Funktionalität über extensions realisiert.

Handshake ServerHello

- Ausgewählter Cipher.
- Unverschlüsselte Extensions
  - Supported Version
  - key_share
- Encrypted Extensions (ab hier verschlüsselt):
  - ALPN - Next Protocol
- CertificateRequest – wenn Client Zertifikat angefordert wird.
- Certificate – Server Zertifikat + Intermediates
- CertificateVerify – Signatur die beweist das der Server den Private Key besitzt.
- Finished – Hash über den Handshake.

Debugging mit Wireshark

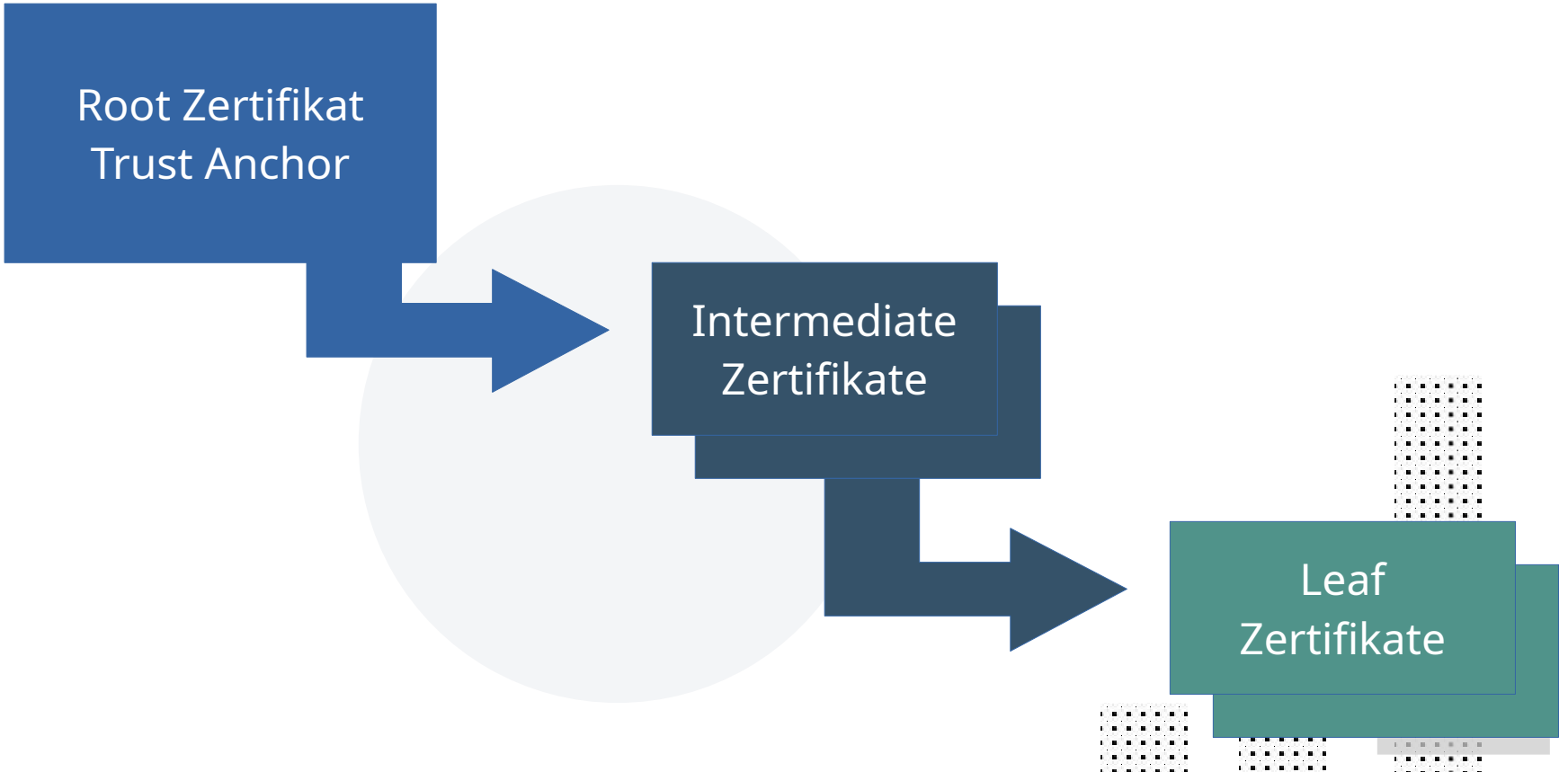
RFC9850

```
SSLKEYLOGFILE=/home/sven/curl.keylog
wireshark -o "tls.keylog_file:${SSLKEYLOGFILE}"
curl -v https://www.cloudflare.com/cdn-cgi/trace
```

PKI und Zertifikate

- X509, Directory Services waren in den 90ern angesagt.
- ASN.1(Abstract Syntax Notation One) kodiertes Binärformat mit Tag Length Value.
- Problem: Alle Zertifizierungsstellen, denen vertraut wird, sind gleich berechtigt.

PKI und Zertifikate



PKI und Zertifikate

Ein Zertifikat angucken:

```
$ openssl x509 -noout -text -in foo.pem
```

Zertifikatsbundle angucken:

```
$ openssl crl2pkcs7 -nocrl -certfile fullchain.pem | \
  openssl pkcs7 -print_certs -noout
```

PKI und Zertifikate

Serial Number:

05:11:b9:73:61:f9:cb:7a:0d:45:55:79:ee:1a:81:d5:d1:4c

Signature Algorithm: ecdsa-with-SHA384

Issuer: C=US, O=Let's Encrypt, CN=YE2

Validity

Not Before: Jun 30 01:32:36 2026 GMT

Not After : Sep 28 01:32:35 2026 GMT

Subject: CN=sven.stormbind.net

Subject Public Key Info:

Public Key Algorithm: id-ecPublicKey

Public-Key: (256 bit)

PKI und Zertifikate

X509v3 extensions:
X509v3 Key Usage: critical
Digital Signature
X509v3 Extended Key Usage:
TLS Web Server Authentication
X509v3 Basic Constraints: critical
CA:FALSE
X509v3 Subject Alternative Name:
DNS:git.sven.stormbind.net,
DNS:sven.stormbind.net
X509v3 CRL Distribution Points:
Full Name:
URI:http://ye2.c.lencr.org/77.crl
CT Precertificate SCTs:
Signed Certificate Timestamp: [...]

X509v3 extensions:
X509v3 Key Usage: critical
Certificate Sign, CRL Sign
X509v3 Extended Key Usage:
TLS Web Server Authentication, TLS
Web Client Authentication
X509v3 Basic Constraints: critical
CA:TRUE

Zertifikate auf dem Server

- Server Zertifikat + Intermediate Zertifikate.
- Nicht das Root Zertifikate mit ausliefern!
  Unnötiger Balast und hilft nicht.
- mTLS: Nur CA + Intermediate welches die Client Zertifikate signiert zur Validierung einbinden.
  Auf keinen Fall den in /etc/ssl/ gedumpten Mozilla Root Store!
- Wenn ACME nicht möglich, Certificate Signing Requests generieren. Kein private Key Weitwurf per Zip Datei und Jira.

Root Store auf dem Linux Client

- Situation immer noch schwierig.
- Initiative von Canonical upki https://discourse.ubuntu.com/t/an-update-on-upki/77063
- RedHat trust tool https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/10/html/securing_networks/using-shared-system-certificates
- Integration Merkel Tree Certificates und Landmark Distribution brauchen neues Tooling.
  Wettet jemand auf systemd-certmanager?

Privater PKI Betrieb

- Orientierung an den CA/Browser Forum Baseline Requirements https://cabforum.org/
- Keine XMAS-Tree Zertifikate (alle Key Usage / Extended Key Usage Attribute einschalten).
- CA: True darf nie auf dem Leaf-Zertifikat auftauchen.
- Root Zertifikat ist ein Trust Anchor, 99 Jahre Laufzeit sind ok.
- Mehr ist besser als weniger – rollt zwei Root Zertifikate aus, legt einen privaten Key in den Tresor.

TLS und DNS: SNI

- Signalisiert für die Gegenseite im Handshake welcher Host erreicht werden soll, und damit welches Server Zertifikat geliefert werden soll.
- Teil des ClientHello → unverschlüsselt!
- Hat für das HTTP Protokoll Virtual Hosting möglich gemacht.
- Heute Standard, außer bei Verbindungen direkt zu IP(v4) Adressen. Hallo Payment Welt. :-)

TLS und DNS: CAA Record

- RFC8659: Deklariert im DNS welche CA für diese Domain Zertifikate ausstellen darf.
  Format:
  `CAA <flags> <tag> <value>`
  `CAA (0|128) (issue|issuewild) “CA Identifier;other-parameter”`
  128 → critical, wenn Parameter durch CA nicht verarbeitet werden können schlägt Verifikation fehl.
- RFC8657: Account URI for ACME: Wer Let’s Encrypt verwendet kann auch den Account im CAA Record hinterlegen.
- RFC8657: validation methods: Einschränkung wie Domain Besitz validiert werden darf.
- Neu: CA/B Ballot SC100 DNSSEC ist jetzt ein muss bei der Abfrage von CAA Records durch CA.
  Entfaltet Wirkung nur bei signierten Zonen.

TLS und DNS: CAA Record

https://letsencrypt.org/docs/caa/

```
$ dig +short IN CAA heise.de
0 issue "amazon.com"
0 issue "letsencrypt.org"
0 iodef "mailto:hostmaster@heise.de"
0 issue "digicert.com"
0 issuewild "sectigo.com"
0 issue "sectigo.com"
```

TLS und DNS: CAA Record accounturi

```
0 issue “letsencrypt.org;accounturi=https://acme-
v02.api.letsencrypt.org/acme/acct/123456789
```

```
# certbot show_account 2>&1|grep "Account URL"|cut -d' '
-f 5

https://acme-v02.api.letsencrypt.org/acme/acct/123456789

# jq -r .uri /etc/letsencrypt/accounts/acme-
v02.api.letsencrypt.org/directory/d3e8497beb846e40dfff77a9
3e09bbf4/regr.json

https://acme-v02.api.letsencrypt.org/acme/acct/123456789
```

TLS und DNS: CAA Record validationmethods

```
CAA 0 issue “letsencrypt.org;validationmethods=dns-01
```

Let’s Encrypt dokumentiert:

`http-01`

`dns-01`

`tls-alpn-01`

Unklar (nicht getestet):

`dns-persist-01`

TLS und DNS: dns-persist-01

https://letsencrypt.org/2026/02/18/dns-persist-01

- DNS-01: _acme-challenge.example.com TXT Record
- DNS-PERSIST-01: _validation-persist.example.com TXT Record

```
_validation-persist.example.com. IN TXT (
  "letsencrypt.org;"
  " accounturi=https://acme-v02.api.letsencrypt.org/acme/acct/1234567890;"
  " policy=wildcard"
  " persistUntil=1767225600"
)
```

TLS und DNS: HTTPS Record

RFC9460

Ergänzung zu A/AAAA Records und HSTS Header

```
dig +short IN HTTPS cloudflare.com
1 . alpn="h3,h2" ipv4hint=104.16.132.229,104.16.133.229
ipv6hint=2606:4700::6810:84e5,2606:4700::6810:85e5
```

Signalisiert:

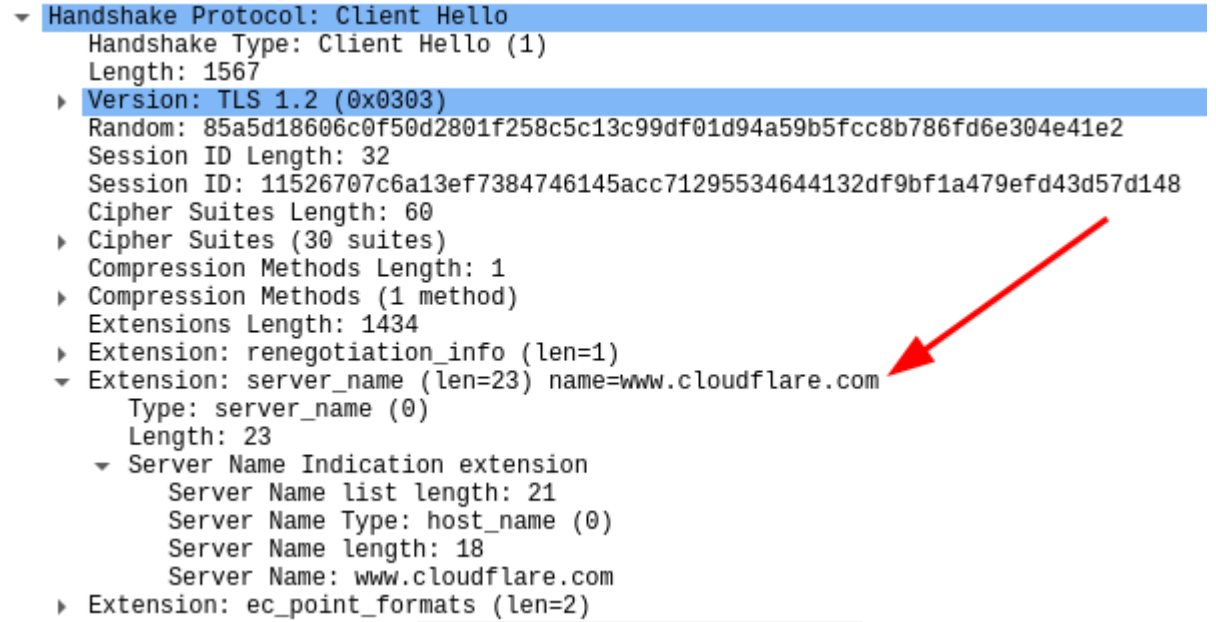
- IP Adresse (ipv4hint / ipv6hint)
- Application Layer Protocol Negotiation (alpn)
- Alternativen Port (port)
- SvcPriority 0 → AliasMode
  `example.com. 3600 IN HTTPS 0 svc.example.net.`

ECH – Encrypted ClientHello

- ClientHello bisher nicht verschlüsselt.
- ServerNameIndication ist damit ein privacy leak.
- Indikator für Zensursysteme um Verbindungen zu unterbrechen.
- Enterprise MITM Systeme setzen auch dort an.

ECH – Un-Encrypted ClientHello

```
$ curl https://www.cloudflare.com
```



ECH – Encrypted ClientHello

- Neu: RFC9849
- Kompatiblität zu Bestandssystemen erhalten, daher:
  - ClientHello Outer – mit dummy SNI
  - ClientHello Inner - verschlüsselt
- Realisiert als extension, kann von älteren Systemen ignoriert werden.
- Browser unterstützen und nutzen ECH bereits.
  - Firefox 119
  - Chrome 105

ECH – ClientHello Outer

```
$ curl --ech true --doh-url https://one.one.one.one/dns-query  \
https://crypto.cloudflare.com/cdn-cgi/trace.cgi | grep sni
sni=encrypted
```

ECH – Anforderungen Client

- TLS 1.3
- Verschlüsselte DNS Abfragen (privacy leak schließen), üblicherweise DoH (DNS über HTTPS)
- DNSSEC (nicht zwingend, aber sinnvoll)
- Problem: Wie bekomme ich den Public Key für die Verschlüsselung des ClientHello Inner zum Client?

ECH – HTTPS RR und ClientHello Inner

Lösung: RFC9460 - DNS :-)

Genauer: HTTPS RR mit Service Parameter Key “ech”

Base64 kodierte ECHConfig, enthält: Public Key, ClientHello Outer SNI Name, hier cloudflare-ech.com

```
$ dig +short IN HTTPS crypto.cloudflare.com

1 . alpn="h2" ipv4hint=162.159.135.79,162.159.136.79
ech=AEX+DQBBfQAgACAwHP9+ObubFXhBiQXc96aqOnS2B6eZgzv0sT
Zv3G/YMgAEAAEAAQASY2xvdWRmbGFyZS1lY2guY29tAAA=
ipv6hint=2606:4700:7::a29f:874f,2606:4700:7::a29f:884f
```

ECH – Anforderung Teil II

- openssl 4
- curl >= 8.8.0
- Debian experimental + buildah + podman helfen.

```
$ cat Dockerfile.exp
FROM docker.io/debian:experimental
RUN apt -t experimental -y -U install curl openssl
RUN mkdir /shared
ENV SSLKEYLOGFILE=/shared/keylogfile
$ buildah build --layers -f Dockerfile.exp -t echtest
```

ECH – Testen mit curl und podman

```
$ podman run \

--mount=type=bind,source=$(pwd)/
shared,destination=/shared \

-ti echtest \

/usr/bin/curl --ech true \

--doh-url https://one.one.one.one/dns-query \
https://crypto.cloudflare.com/cdn-cgi/trace.cgi
```

ECH – Server Seite

- CDN Dienstleiter wie Cloudflare (wachsendes Internet Oligopol)
- Selber machen:
  - Nginx >=1.29.4 + openssl 4 (https://blog.nginx.org/blog/encrypted-client-hello-comes-to-nginx)
  - Defo Patches https://defo.ie/
  - Caddy https://caddyserver.com/docs/automatic-https#encrypted-clienthello-ech

ECH – nginx shared-mode

Webserver verarbeitet ClientHello Outer und ClientHello Inner.

```
# Schlüsselpaar + ECHConfig
# generieren mit Openssl 4:

$ openssl ech \
   -public_name example.com \
   -out example.com.pem.ech
$ cat example.com.pem.ech
-----BEGIN PRIVATE KEY-----
MC4CAQAwBQYDK2VuBCIEIChAU09NbdNui6ObPcDvdHPIhe0pgdS3cWFN
2u8Uc1tq
-----END PRIVATE KEY-----
-----BEGIN ECHCONFIG-----
AD7+DQA6gAAgACAkC7OhCDNraBwMRQpCFWe2a158q03X2W1zzR7OPb1y
YwAEAAEA
AQALZXhhbXBsZS5jb20AAA==
-----END ECHCONFIG-----
```

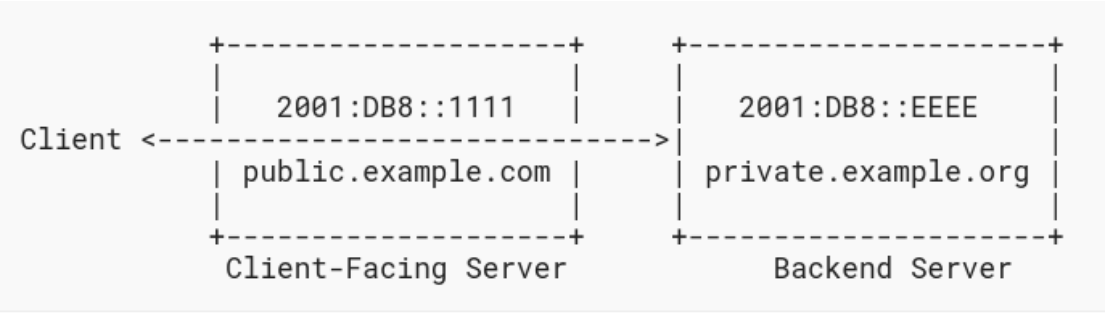
```
http {
    # Ein oder mehrere:
    ssl_ech_file /etc/echkeydir/example.com.pem.ech;

    # Dummy SNI / ClientHello Outer
    server {
        listen 443 ssl;
        server_name example.com;
        ssl_certificate /etc/certs/example.com.crt;
        ssl_certificate_key /etc/certs/example.com.priv;
        [...]
        }
    }

    # ClientHello Inner
    server {
        listen 443 ssl;
        server_name foo.example.com;
        ssl_certificate /etc/certs/foo.example.com.crt;
        ssl_certificate_key /etc/certs/foo.example.com.priv;
        [...]
        }
    }
}
```

ECH – split-mode

- Client-Facing Server entschlüsselt ECH Extension.
  TLS Verbindung kommt kommt Zwischen Client und Backend zustande.



- CDN ist hier Proxy und sieht nur Teile des Handshake.
- Defo.ie hat Patches für HaProxy (nicht getestet).

ECH – Herausforderungen

- Muss selber paketiert werden, container helfen.
- Enge Verzahnung mit DNS nötig.
  - Key Rotation täglich (statischer Key).
  - Ein Vorschlag: https://www.ietf.org/archive/id/draft-ietf-tls-wkech-01.html DNS relevante Informationen via JSON in .well-known verteilen.
- DNSSEC und DoH sinnvoll.
- Anonymity Set (die Menge an Hostnamen die sich hinter meinem outer SNI verbergen) vermutlich meist klein.
- Folge: Wenn mich der Zensor blocken will sperrt er gefahrlos die ganze IP, oder blockt basierend auf dem outer SNI. Kollateralschaden gering. :-(

ECH – GREASE

- RFC8701: Applying Generate Random Extensions And Sustain Extensibility (GREASE) to TLS Extensibility
- Reservierte IDs für Erweiterungen die vom Client mit random Inhalten befüllt werden können.
- Ziel: Verbindungsparameter sollen möglichst gleich aussehen, um ein filtern auf bestimmte Erweiterung möglichst unmöglich zu machen.
- Funktioniert, weil unbekannte Extensions ignoriert werden.

ECH – GREASE

- Browser implementieren ECH Client GREASE bereits.
- Folge: Nur weil im ClientHello eine ECH extension ist, wird nicht zwingend ECH verwendet.
- Hilft es gegen den Zensor, die MITM Boxen?
- Vielleicht ein bisschen gegen den Zensor. MITM Boxen können weiter connection downgrades erzwingen. ECH ist hier auf maximale Kompatiblität angelegt.

PQC – Zwei Probleme

Key agreement
Harvest now decrypt later Angriffe.

Zertifikate / Signaturen
Angriffe auf ganze PKI möglich.

PQC – Key Agreement

Gelöst mittels hybrid ECDHE-MLKEM, X25519MLKEM768.

MLKEM = Module-Lattice-Based Key Encapsulation (ehml. CRYSTALS-Kyber)

Ab Openssl 3.5, ggf. prüfen:

```
$ openssl list -tls-groups|grep MLKEM
```

```
Apache:
SSLOpenSSLConfCmd Curves X25519MLKEM768:X25519:prime256v1
Nginx:
ssl_ecdh_curve  X25519MLKEM768:X25519:secp384r1:prime256v1;
```

Folge: Handshake deutlich größer.

Sollten neuere Verfahren etabliert werden muss diese Konfiguration angepasst werden.

PQC - Zertifikate

WebPKI setzt auf MCTs – Merkel Tree Certificates.

https://letsencrypt.org/2026/06/03/pq-certs

IETF WG: PKI, Logs, And Tree Signatures – PLANTS

Chrome will keinen x509 PQC Root Store anbieten.

Was machen mit privater CA?

Cloudflare setzt erstmal auf ML-DSA-44 (RFC9881)

Unterstützt ab Openssl 3.5 (u.a. Debian Trixie, aktuelles stable Release)

https://blog.cloudflare.com/ml-dsa-will-have-to-do/

https://blog.cloudflare.com/post-quantum-authentication-to-origins/

PQC – ML-DSA Zertifikate

```
$ openssl genpkey \
  -algorithm mldsa44 \
  -provparam ml-dsa.output_formats=seed-only \
  -out mldsa.key
```

```
$ openssl pkey -in mldsa.key -noout -text
ML-DSA-44 Private-Key:
seed:
    53:f6:95:91:09:a6:32:73:91:5e:42:94:38:d1:c9:
    4b:1b:30:d8:38:6e:70:9c:64:fd:fb:36:03:e8:90:
    14:67
[...]
```

PQC - Merkel Tree Certificates

Vorteile (happy case):

- Kleinerer Handshake. Nur eine Signatur, ein public key und ein inclusion proof.
  Benötigt aktuellen “landmark” von Landmark Distribution Point.
- Certificate Transparency build-in.

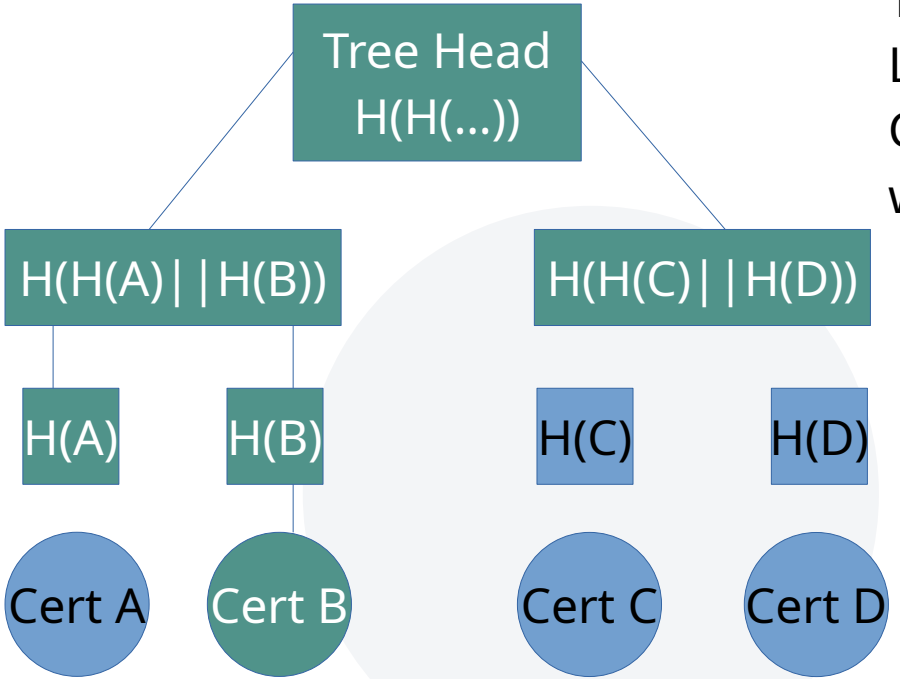
Details

https://blog.cloudflare.com/bootstrap-mtc/

https://datatracker.ietf.org/wg/plants/about/

https://www.ietf.org/archive/id/draft-ietf-plants-merkle-tree-certs-03.html

PQC – MTC Inclusion Proof



Tree Head Hash wird signiert.
Landmarks = Subtree Heads
Client soll im Handshake mitteilen welche Tree Heads er kennt.

Und jetzt?

- CAA Records anlegen.
- HTTPS Records anlegen.
- Hybrides ECDHE-MLKEM aktivieren.
- DNSSEC ausrollen auch wenn es nicht ganz trivial ist.
- DoH breiter aufstellen. Privacy Leaks schließen.
- ECH GREASE im Client nutzen wo immer möglich.
- Auf Merkel Trees warten.
- Plan für die eigene PKI und die Quantencomputer Apocalypse bereit halten.

Fragen?

Sven Höxter
sven@stormbind.net

Workshop / Beratung:
hello@shxit.de