Transport Layer Security (TLS) and Recent Advances

From TLS 1.2 to TLS 1.3, Encrypted Client Hello, and Merkle Trees for PKI

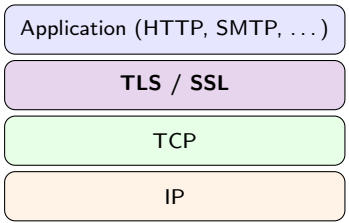
Luca Campa

University of Klagenfurt – Introduction to Cybersecurity

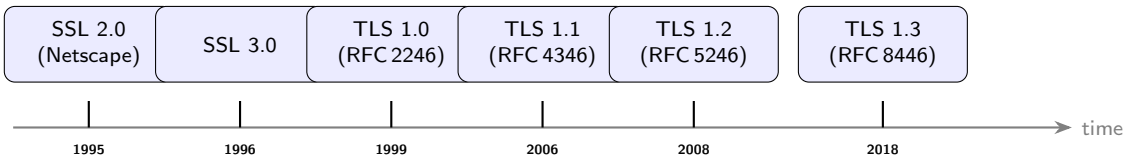
Outline

Why Transport-Level Security?

- HTTP is inherently **unencrypted**: any on-path observer can read or modify traffic.
- We need:
  - **Confidentiality**
  - **Integrity**
  - **Authentication**
- TLS sits *above TCP* and *below HTTP*, transparent to applications.

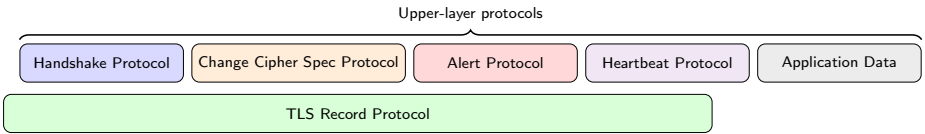


Brief History



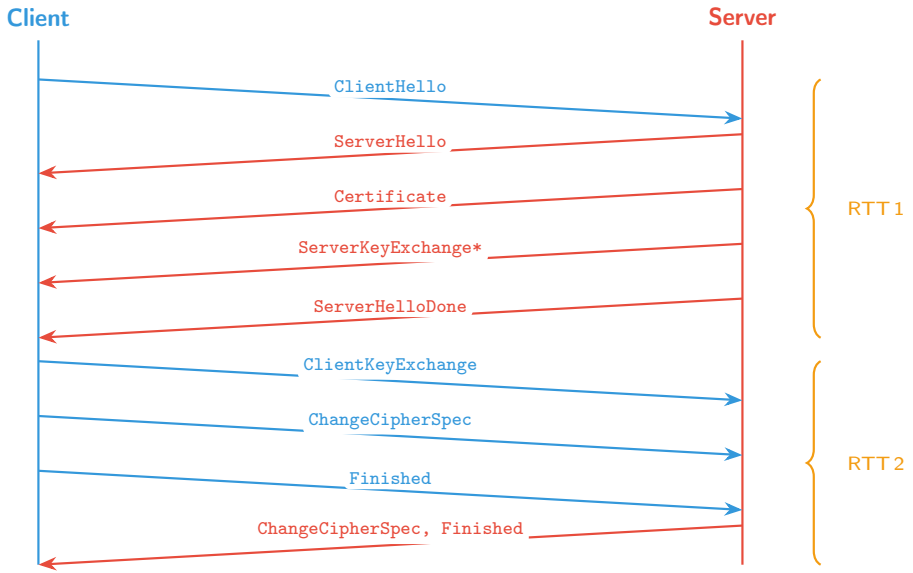
- SSL 2.0/3.0 deprecated (insecure); TLS 1.0/1.1 deprecated in 2021 (RFC 8996).
- **TLS 1.3** is the current standard – major redesign for security and performance.

TLS 1.2 Architecture



- **Record Protocol**: fragmentation, (optional) compression, MAC, encryption, framing.
- **Handshake**: negotiate cipher suite, authenticate server (optionally client), derive keys.

TLS 1.2 Full Handshake (2-RTT)



TLS 1.3 – Key Improvements over 1.2

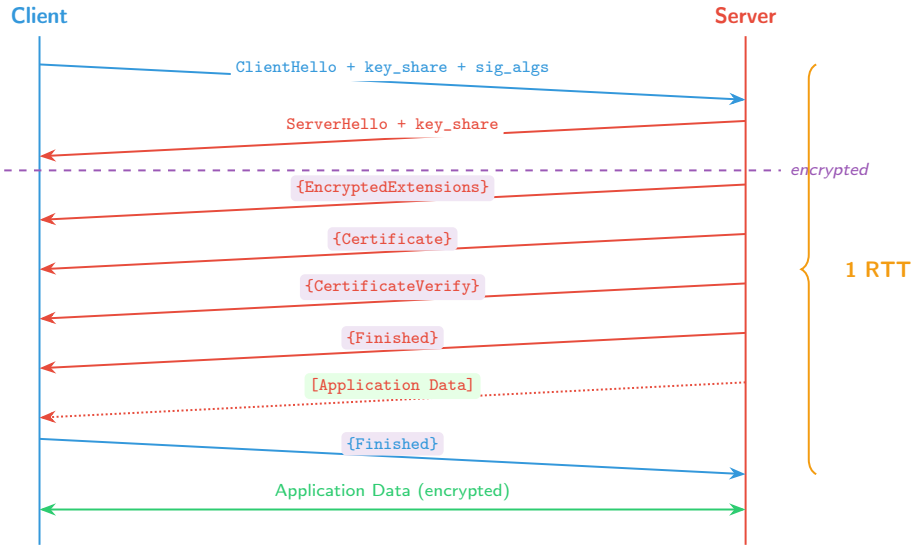
Removed (legacy / insecure):

- Static RSA key exchange
- DH without forward secrecy
- CBC-mode ciphers, RC4, 3DES
- Compression
- Renegotiation
- ChangeCipherSpec message

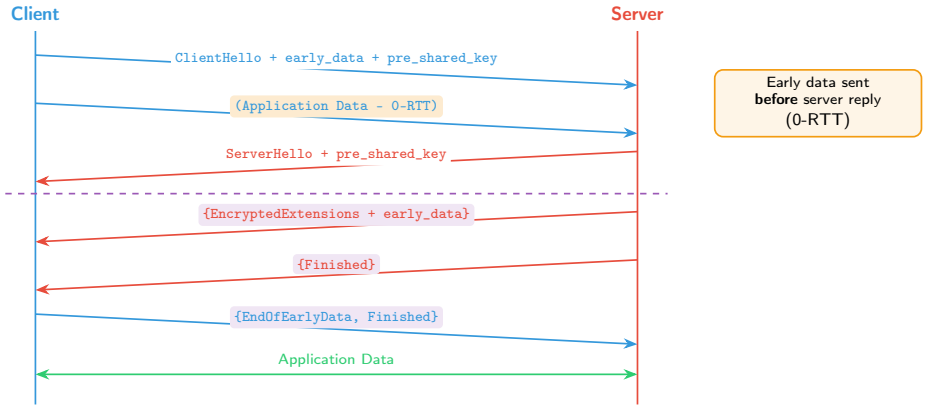
Added / Changed:

- **1-RTT** handshake (reduced from 2-RTT)
- **0-RTT** early data for repeat connections
- All ciphers are **AEAD** (e.g. AES-GCM, ChaCha20-Poly1305)
- **HKDF**-based key derivation
- Handshake encrypted after ServerHello
- Every key exchange provides **forward secrecy**

TLS 1.3 Full Handshake (1-RTT)

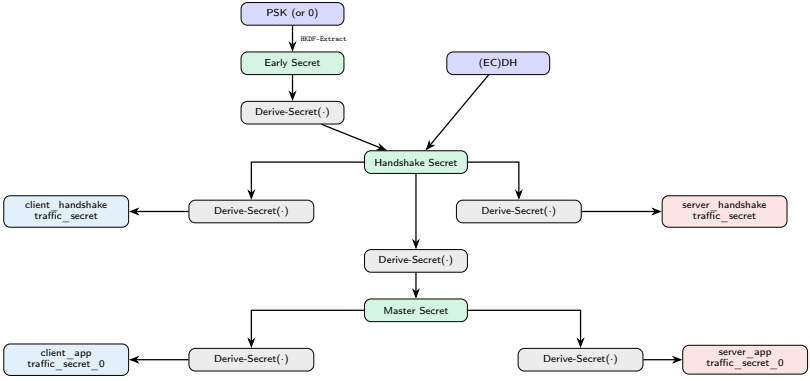


TLS 1.3: 0-RTT (optional)



- Uses a Pre-Shared Key (PSK) from a previous session.
- **Trade-off**: 0-RTT data is *not* forward-secret and is *replayable* – servers must implement replay protection.

TLS 1.3 Key Derivation (HKDF)



Procedures:

- HKDF-Extract : SHA2-256-based HKDF algorithm
- Derive-Secret(Secret, Label, Messages) $\simeq$ HKDF-Expand(Secret, Label || SHA2-256(Messages))

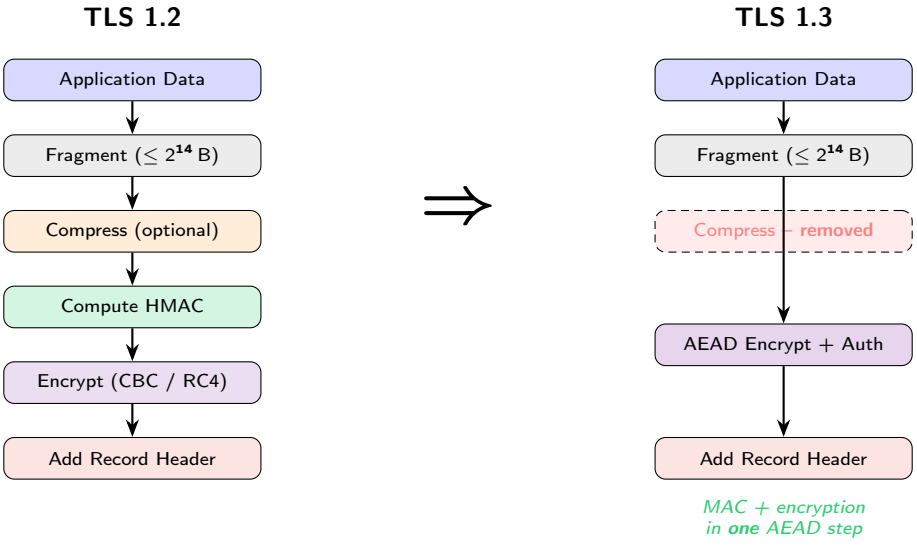
Further information: RFC 5869 (HKDF), RFC 8446 Section 7.1 (TLS 1.3 key schedule).

TLS 1.3 Cipher Suites

Cipher Suite	AEAD	Hash
TLS_AES_128_GCM_SHA256	AES-128-GCM	SHA-256
TLS_AES_256_GCM_SHA384	AES-256-GCM	SHA-384
TLS_CHACHA20_POLY1305_SHA256	ChaCha20-Poly1305	SHA-256

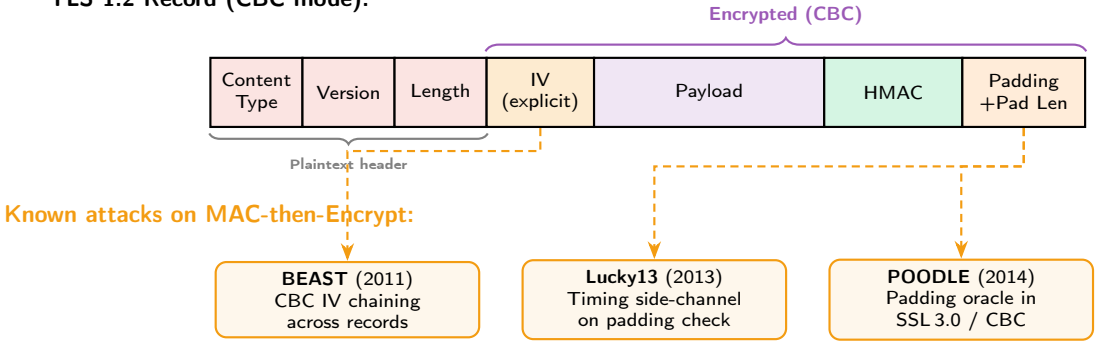
- Only **AEAD** algorithms remain (Authenticated Encryption with Associated Data).
- Cipher suite *only* specifies record protection + hash; key exchange and authentication are negotiated separately via extensions.
- Mandatory to implement: `TLS_AES_128_GCM_SHA256`.

TLS Record Protocol: Processing Pipeline



TLS 1.2 Record: MAC-then-Encrypt

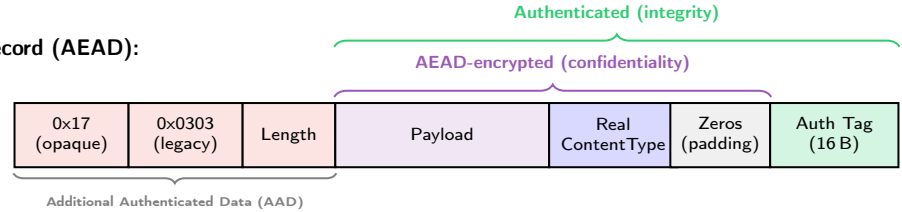
TLS 1.2 Record (CBC mode):



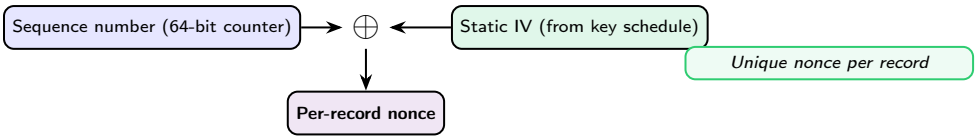
- **MAC-then-Encrypt**: compute HMAC over plaintext, then encrypt payload + MAC + padding together.
- Receiver must *decrypt first*, then check MAC – padding errors and HMAC computation leak information via timing.

TLS 1.3 Record: AEAD Construction

TLS 1.3 Record (AEAD):



Nonce construction:



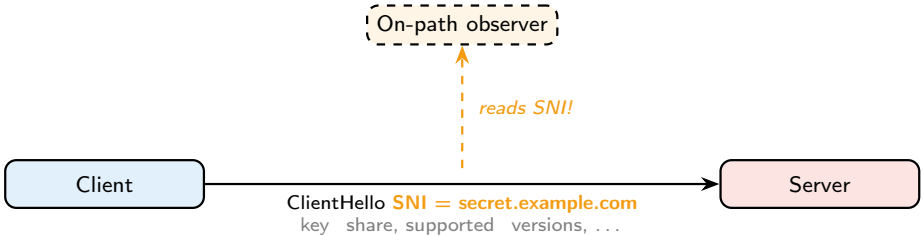
- Real content type is **inside** the encrypted payload – observer cannot distinguish handshake from application data.
- No separate MAC, no padding bytes, no compression – **eliminates entire classes of attacks**.

Record Protocol: TLS 1.2 vs 1.3 Comparison

	TLS 1.2	**TLS 1.3**
Encryption modes	CBC, RC4, stream ciphers	**AEAD only** (AES-GCM, ChaCha20-Poly1305)
Integrity	Separate HMAC (MAC-then-Encrypt)	Integrated authentication tag
Compression	Supported (CRIME attack)	**Removed**
Padding	PKCS#7 (oracle attacks)	Zero-padding inside AEAD
Content type	Plaintext in header	**Encrypted** inside payload
Nonce / IV	Explicit IV per record (CBC)	XOR of sequence # with static IV
Renegotiation	Supported (triple handshake attack)	**Removed**

Design principle of TLS 1.3: reduce complexity $\Rightarrow$ eliminate attack surface.
Every removed feature corresponds to a known class of attacks.

What TLS 1.3 Still Leaks



- TLS 1.3 encrypts most of the handshake *after* ServerHello.
- But the **ClientHello** is sent in the clear, including:
  - **Server Name Indication (SNI)** – reveals which site the client visits.
  - **ALPN** list – reveals which application protocol.
- Combined with encrypted DNS (DoH/DoT), SNI is the **last plaintext signal** of the destination.

Encrypted Client Hello (ECH): Overview

RFC 9849 (March 2026) "TLS Encrypted Client Hello"

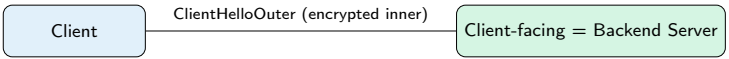
- Encrypts the **entire ClientHello** (including SNI, ALPN, ...) under a server's **public key** published in DNS.
- The ClientHello is split into:

  ClientHelloOuter Innocuous values + encrypted payload. Contains a *public_name* (e.g. `cloudflare-ech.com`).

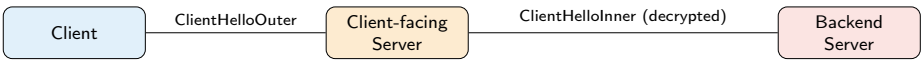
  ClientHelloInner The real ClientHello (encrypted). Contains the true SNI.
- To an observer, all connections to the same provider look **identical** – which specific site is hidden.
- Uses **HPKE** (Hybrid Public Key Encryption, RFC 9180) for the encryption.

ECH: Topologies

Shared Mode

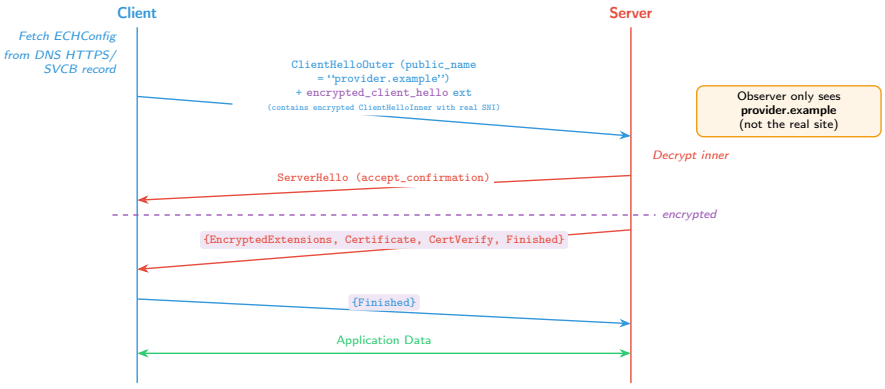


Split Mode



*Terminates TLS; client-facing does **not** see plaintext data.*

ECH: Handshake Flow



Further details: RFC 9849 Section 4 (ECH protocol), RFC 9180 (HPKE).

ECH: Configuration Structure

ECHConfig (published in DNS HTTPS/SVCB records):

- `config_id` – 1-byte identifier
- `kem_id` – HPKE KEM (e.g. X25519)
- `public_key` – HPKE public key
- `cipher_suites` – KDF + AEAD pairs
- `maximum_name_length` – for padding
- `public_name` – client-facing server DNS name

Key features:

- **GREASE**: clients without ECH config send dummy extensions, so ECH traffic is not highlighted.
- **Retry**: if decryption fails, server returns fresh ECHConfig in EncryptedExtensions.
- Required cipher suite: DHKEM(X25519) + HKDF-SHA256 + AES-128-GCM.

ECH: Security and Privacy Goals

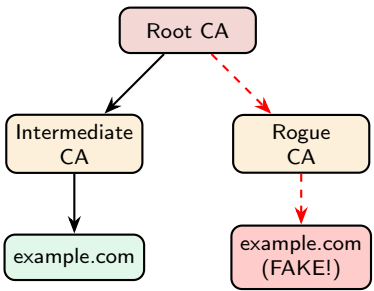
1. **Security preservation**: ECH does not weaken any existing TLS 1.3 security properties.
2. **Handshake privacy**: connections to servers in the same *anonymity set* are indistinguishable from an observer's point of view.
3. **Downgrade resistance**: an attacker cannot force a connection back to non-ECH TLS.

Requires complementary technologies:

- **Encrypted DNS** (DoH, DoT, DoQ) – hides DNS queries.
- Multiple domains behind the **same IP/provider** – forms the anonymity set.
- Same ECHConfig for many domains – maximises anonymity set size.

The CA Trust Problem

- Web PKI relies on ∼150 root **Certificate Authorities (CAs)**.
- Any CA can issue a certificate for *any* domain.
- Historical incidents of CA misissuance (DigiNotar 2011, Symantec 2015, . . .).
- How can domain holders **detect** unauthorised certificates?



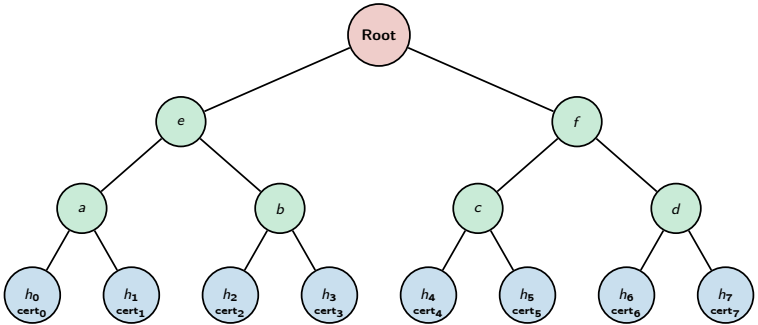
Certificate Transparency (CT): Overview

RFC 6962 / RFC 9162 – initiated by Google engineers.

- **Append-only, public logs** of all issued TLS certificates.
- Anyone can submit certificates; anyone can query and audit the logs.
- When a CA issues a certificate, it submits it to ≥ 1 CT log and receives a **Signed Certificate Timestamp (SCT)**.
- The SCT is a promise to incorporate the certificate within the **Maximum Merge Delay (MMD)** – typically 24 hours.
- TLS clients (Chrome, Safari) **require** valid SCTs – certificates *not in a log* are rejected.

Key actors: CAs (submitters), Log servers, Monitors, Auditors, TLS clients.

Merkle Hash Trees: Structure



Signed Tree Head (STH)

Nodes = SHA-256(0x01‖L‖R)

Leaves = SHA-256(0x00‖cert)

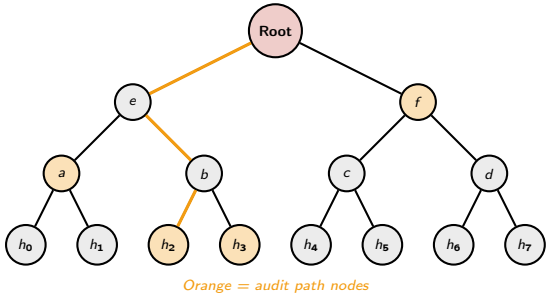
Merkle Audit Proofs (Inclusion Proof)

Goal: Prove that cert_2 is in the tree without downloading all entries.

Audit path for cert_2: $\{h_3, a, f\}$

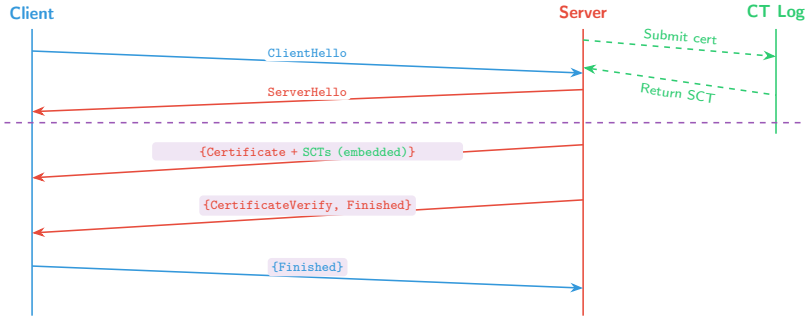
1. Compute $h_2 = \text{H}(0\text{x}00 \| \text{cert}_2)$
2. Compute $b = \text{H}(0\text{x}01 \| h_2 \| h_3)$
3. Compute $e = \text{H}(0\text{x}01 \| a \| b)$
4. Compute $\text{Root} = \text{H}(0\text{x}01 \| e \| f)$
5. Compare with signed root $\Rightarrow$ ✓

Proof size: $O(\log n)$ – efficient



CT in the TLS Handshake

Client verifies:
1. Certificate chain
2. SCT signatures
3. $\geq$ 2 SCTs from distinct logs

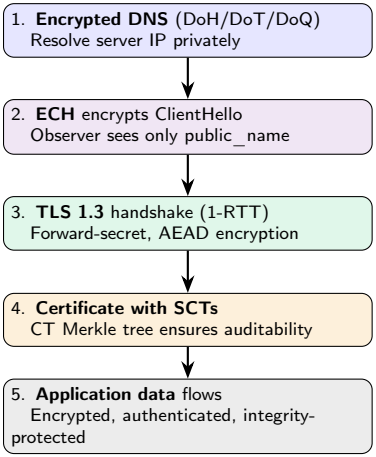


Merkle Trees for PKI Trust

Why Merkle Trees?

- **Efficient auditability**: inclusion proofs are $O(\log n)$ – a log with 1 billion certificates requires only ≈ 30 hash values for a proof.
- **Append-only guarantee**: consistency proofs cryptographically prevent tampering with history.
- **Decentralised trust**: no need to trust a single log – monitors and auditors cross-check via *gossip protocols*.
- Makes CA misbehaviour **publicly detectable** within the Maximum Merge Delay window.
- Nowadays shorter-lived certificates and large signatures from Post-Quantum Cryptography algorithms make traditional X.509 certificate chains less efficient (see draft: `https://datatracker.ietf.org/doc/draft-ietf-plants-merkle-tree-certs/`).

Modern Secure Connection (2026)



Passive observer sees:
– Encrypted DNS query
– IP of CDN/provider
– "provider.example" as SNI
– Encrypted TLS data

Cannot determine
which site is visited

Summary

TLS 1.3 (RFC 8446)

- 1-RTT handshake (0-RTT optional)
- AEAD-only ciphers
- Forward secrecy mandatory
- HKDF key schedule
- Encrypted handshake after ServerHello

Encrypted Client Hello (RFC 9849)

- Encrypts ClientHello (including SNI)
- Uses HPKE with DNS-published keys
- GREASE for indistinguishability
- Supported by Chrome, Firefox, Cloudflare

Certificate Transparency

- Append-only Merkle tree logs
- SCTs embedded in certificates
- $O(\log n)$ inclusion & consistency proofs
- Google Chrome requires CT since 2018

Full privacy stack (2026):

1. Encrypted DNS (DoH / DoT)
2. ECH (hides SNI)
3. TLS 1.3 (forward-secret channel)
4. CT (auditable certificates)