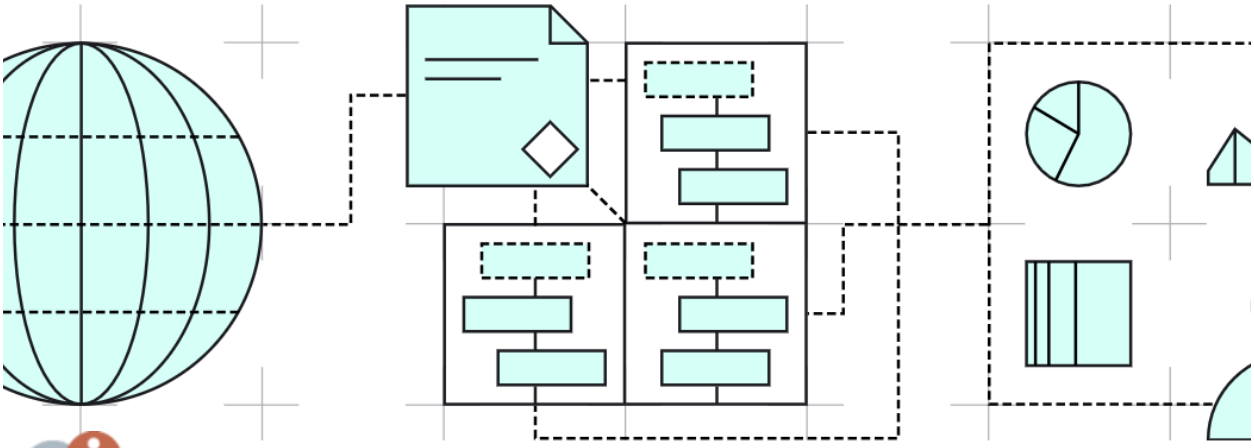


Fork Consistency & Certificate Transparency

Content

- **Why Certificate Transparency?**
- **How CT Works?**
  - **Overview**
- **Merkle Tree**

---

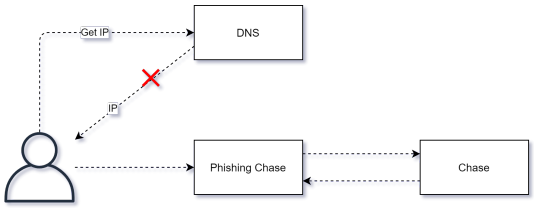
1. Why Certificate Transparency?

▼ **HTTP**

- Communication not encrypted
- Man in the middle
- Example

▼ **HTTPS**

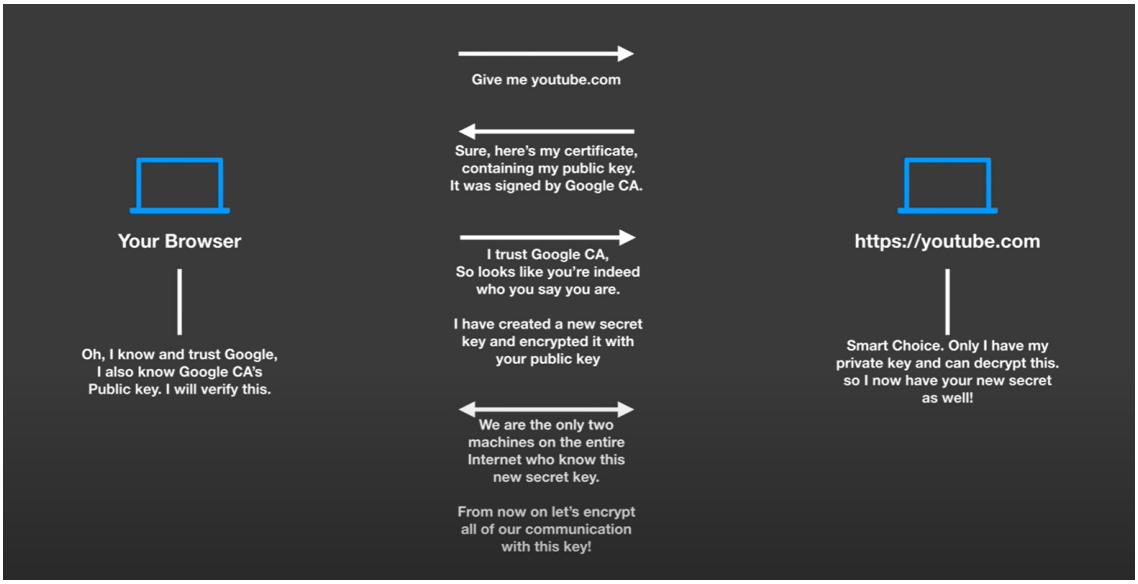
- Encrypted communication
- How it works?

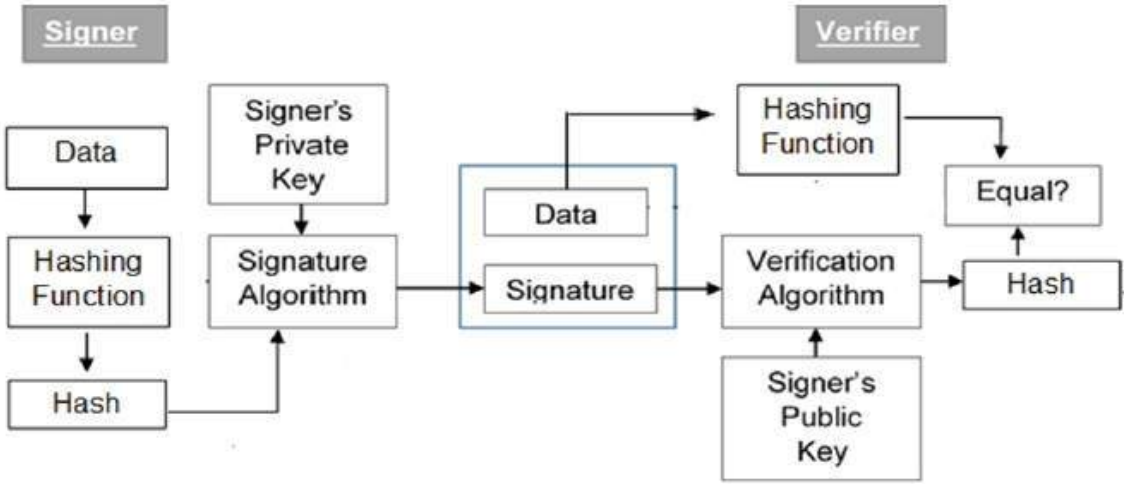


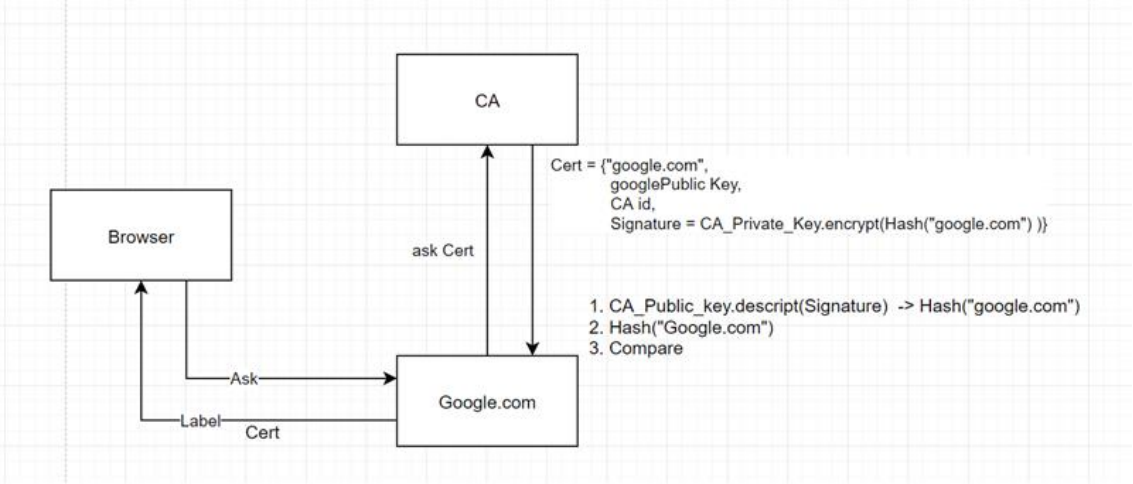
https://www.youtube.com/watch?v=T4Df5_cojAs

▼ **Does HTTPS resolved all issues?**

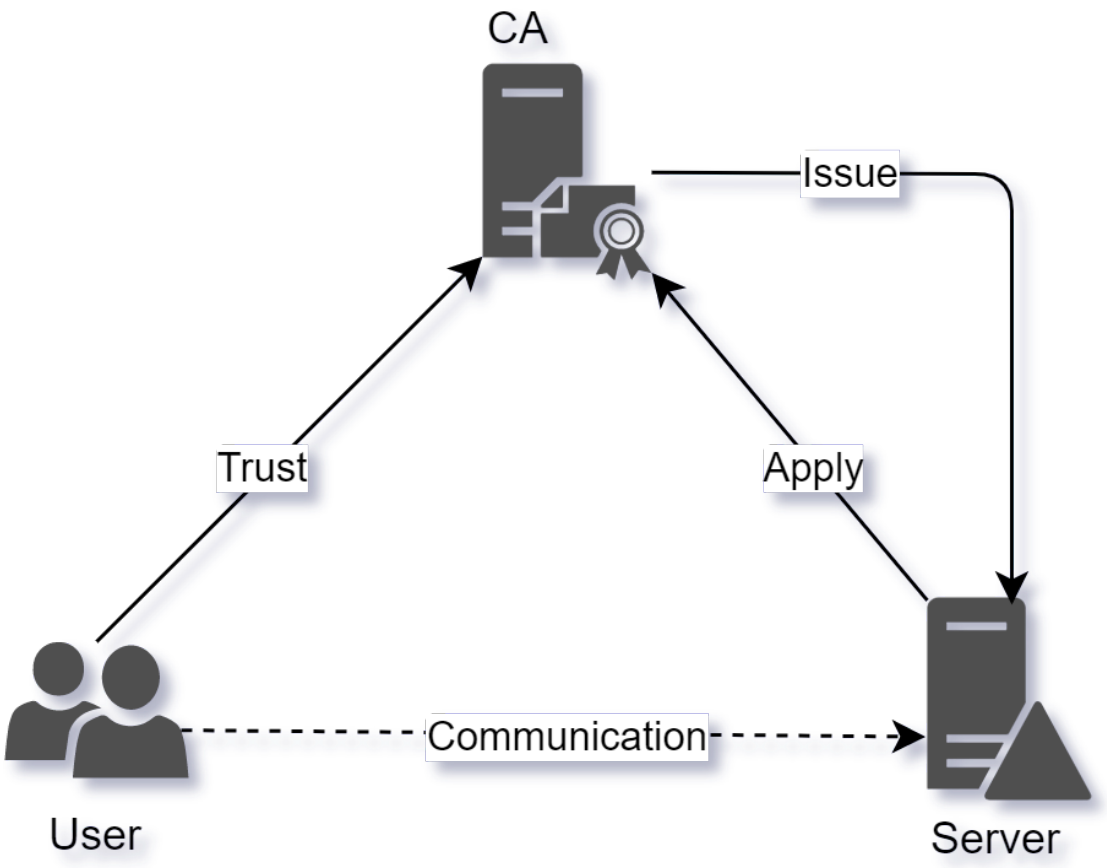
▼ Certificate Authority







▼ Components



▼ Problem

2. How CT Works?

How CT Works : Certificate Transparency

Certificate logs are append-only ledgers of certificates. Because they're distributed and independent, anyone can query them to see what certificates have been included and when. Because they're append-only, they are verifiable by Monitors. Organisations and individuals with the technical skills and capacity

https://certificate.transparency.dev/howctworks/

3. Merkle Tree

▼ **Basis: Cryptographic Hashes**

- A cryptographic hash function:
  - A deterministic function H that maps an arbitrary-size message M to a small fixed-size output $H(M)$. (Ex. SHA-256)

▼ **Serving a Log**

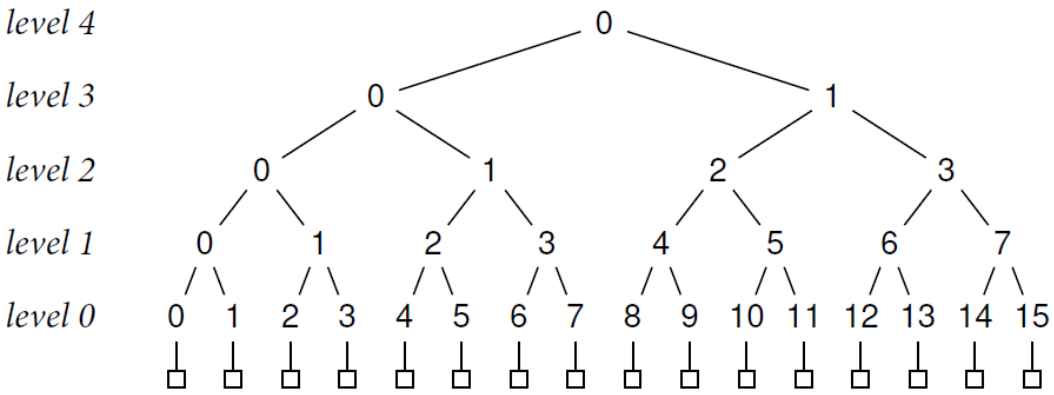
1. *Latest()* returns the **current log size** and top-level hash, cryptographically signed by the server for non-repudiation.
2. *RecordProof(R, N)* returns the proof that record *R* is

- It is infeasible in practice to produce an pair of distinct message $M_1 \neq M_2$ with identical hashes $H(M_1) = H(M_2)$.

contained in the tree of size *N*.

3. *TreeProof(N, N')* returns the proof that the tree of size *N* is a prefix of the tree of size *N'*.
4. *Lookup(K)* returns the record index *R* matching lookup key *K*, if any.
5. *Data(R)* returns the data associated with record *R*.

▼ **How a Merkle tree got constructed?**

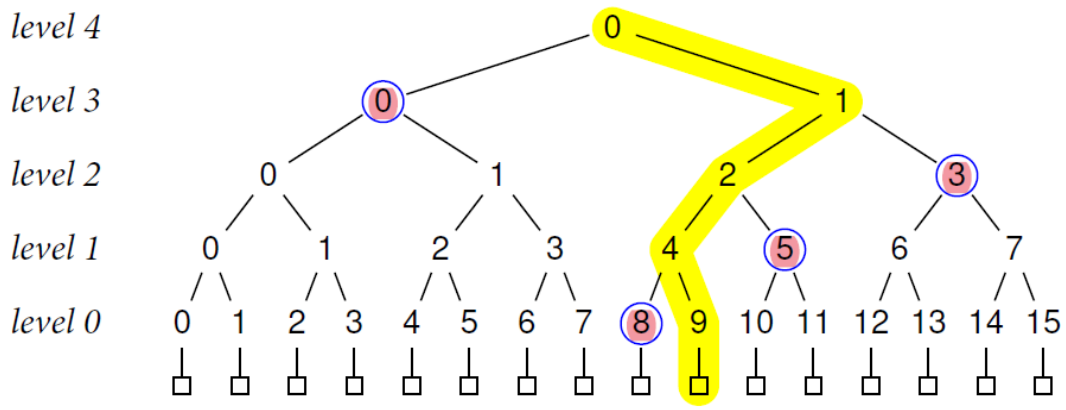


$$h(0, K) = H(recordK)$$

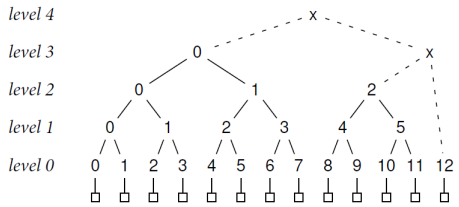
$$h(L+1, K) = H(h(L, 2K), h(L, 2K+1))$$

▼ **How to verify a record is in the log?**

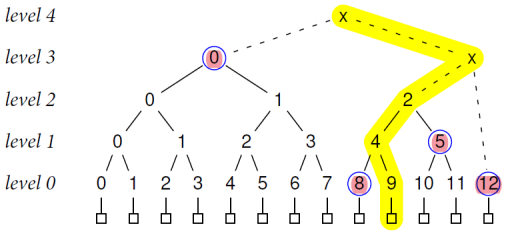
In general, the proof that a given is contained in the tree requires lgN hashes, one for each level below the root.



▼ Generalize the Merkle tree - Construction



▼ Generalize the Merkle tree - Verification



▼ Storing a log

- Log record data;
- Log index;
- Hashes;

▼ Optimization for storing

Maintaining lgN append-only files, each holding the sequence of hashes at one level of the tree.

▼ Verifying a Log

```
validate(bits B as record R):
    if R ≥ cached.N:
        N, T = server.Latest()
```

```
        if server.TreeProof(cached.N, N) cannot be verified:
            fail loudly
        cached.N, cached.T = N, T
    if server.RecordProof(R, cached.N) cannot be verified using B:
        fail loudly
    accept B as record R
```