Building Efficient Verifiable Logs:
Introducing Trillian Tessera and TesseraCT

Pass the SALT 2025, Lille

Google TrustFabric
Google Open Source
Security Team

Philippe Boneff & Roger Ng

Jul 2025

Philippe Boneff
phboneff@

Engineer @ Google Open Source Security, TrustFabric
Certificate Transparency Tech Lead

Roger Ng
rogerng@

Engineer @ Google Open Source Security, TrustFabric

Deter bad behaviour by making it discoverable.

Building Efficient Verifiable Logs:
Introducing Trillian Tessera and TesseraCT

01 From Certificate Transparency to Transparency

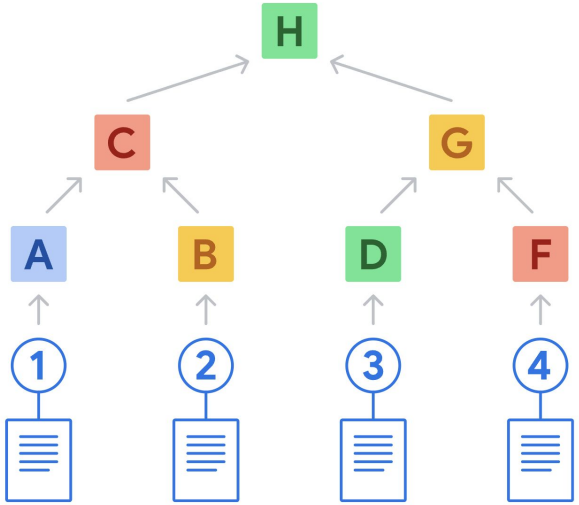
02 Tessera & TesseraCT

03 Demo

Certificate Transparency case study

Premise	User contacts a domain over HTTPS and wants to ensure they are connected with the authentic domain owner.
Requisite	User gets a certificate for this domain **that proves ownership** of this domain.
Problem	How does the user know this proof of ownership is **authentic**?
Solution	**Convince** the user that **domain owners** would be **aware** of any mis-issued certificate, and **would react.**
How?	Policies requiring to log **all TLS certificates** in an **append-only**, publicly accessible and **verifiable** data structure.

Meet Merkle Trees



Append only: once you add an entry to it, it cannot be removed *undetectably*

Verifiable: using cryptographic proofs, verifiers can check that an entry *is* in the tree and that two version of the tree are *consistent*

Great. Now, log 10B certs per year, live, globally, with no corruption.

And maybe...

... all Go packages?

... all E2EE messaging public keys?

Proton Mail WhatsApp iMessage

... all build artifacts?

What else?

Private computing, private information retrieval ...

C2SP specs: common APIs

https://c2sp.org/

01

Checkpoint

Common tree root representation.

02

Witnessing

Protocol protecting **against split-view attacks** by verifying and attesting that logs grow consistently.

03

Tiles

POSIX-compatible format to serve log data.

Scalable and **cheap to operate**, in terms of engineering and storage resources.

04

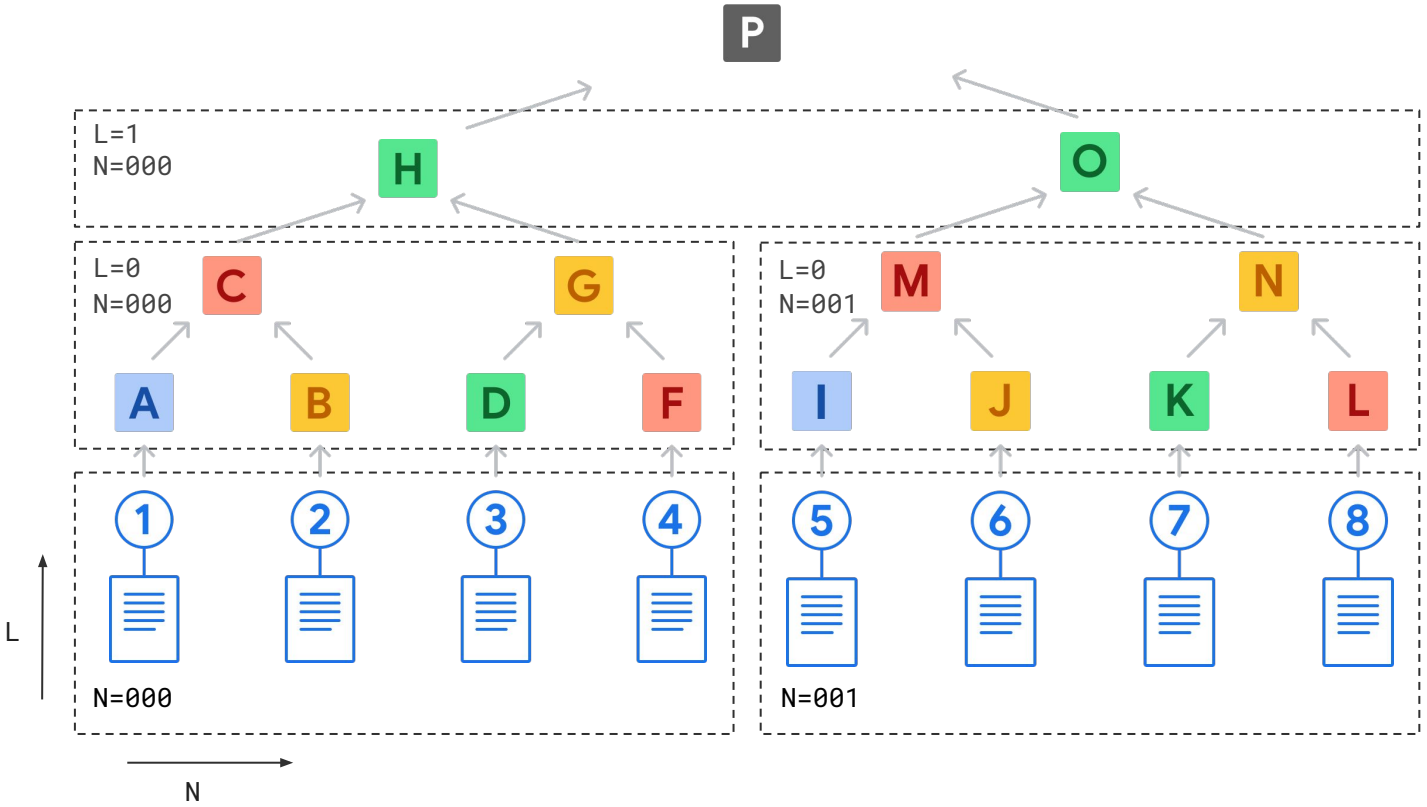
Static CT API

All of the previous points, but **backwards compatible with RFC6962**:

diff:

- Checkpoints
- Different paths
- Leaf format

Tiling a log




```
prefix/
  checkpoint
  tile/<L>/<N>[.p/<W>]
    1/
      000
    0/
      000
      001
    entries/<N>[.p/<W>]
      000
      001
```

Tessera - next generation transparency log

Open source

Available in beta
Written in Go

It's a library

You embed it in your own server

Philosophy

Simplicity

Multi-implementation storage

Asynchronous integration in storage implementation

Resilience and availability

APIs

Write
`appender.Add($DATA)`

Read
C2SP Tiles specs (compatible with CT)

Tessera - under the hood

Choose your own backend

- **POSIX**: you can run it locally!
- **MySQL**: one single SQL DB

- **GCP**: (Spanner + GCS)
  no server on the read path
- **AWS**: (AuroraDB + S3)
  no server on the read path

Performance adapted to your needs

- From 1 to 8k QPS *depends on backend and $
- Multiple, 1, (0?!) server

Tessera - Flexible, but opinionated

Tessera puts you on a **safe** path
↳ but you remain in control

Supports multiple **concurrent** servers
↳ better reliability

Antispam (deduplication), pushback
↳ no denial of service attack

Wait for matching index or checkpoint
↳ ensure entries are integrated

Integrate with witnessing
↳ no split view attack

```
opts := tessera.NewAppendOptions().
             WithCheckpointSigner(signer).
             WithAntispam(uint(antispamCacheSize), antispam).
             WithCheckpointInterval(*checkpointInterval).
             WithBatching(*batchMaxSize, *batchMaxAge).
             WithPushback(*pushbackMaxOutstanding)

appender, shutdown, reader, err := tessera.NewAppender(ctx,
                                        driver, opts)
if err != nil {
       panic(err)
}

index, err := appender.Add(ctx, tessera.NewEntry(data))()
```

TesseraCT vs Tessera

TesseraCT is a binary

TesseraCT is a binary using Tessera

All the CT logic
certificate parsing, SCTs, etc.

TesseraCT can run on
GCP, AWS (for now)

Kudos to Sunlight, Itko, Azul,
Compact logs built independently

Endpoints are different

`s/data/tile`: different path

`get-root`: CT specific read endpoint

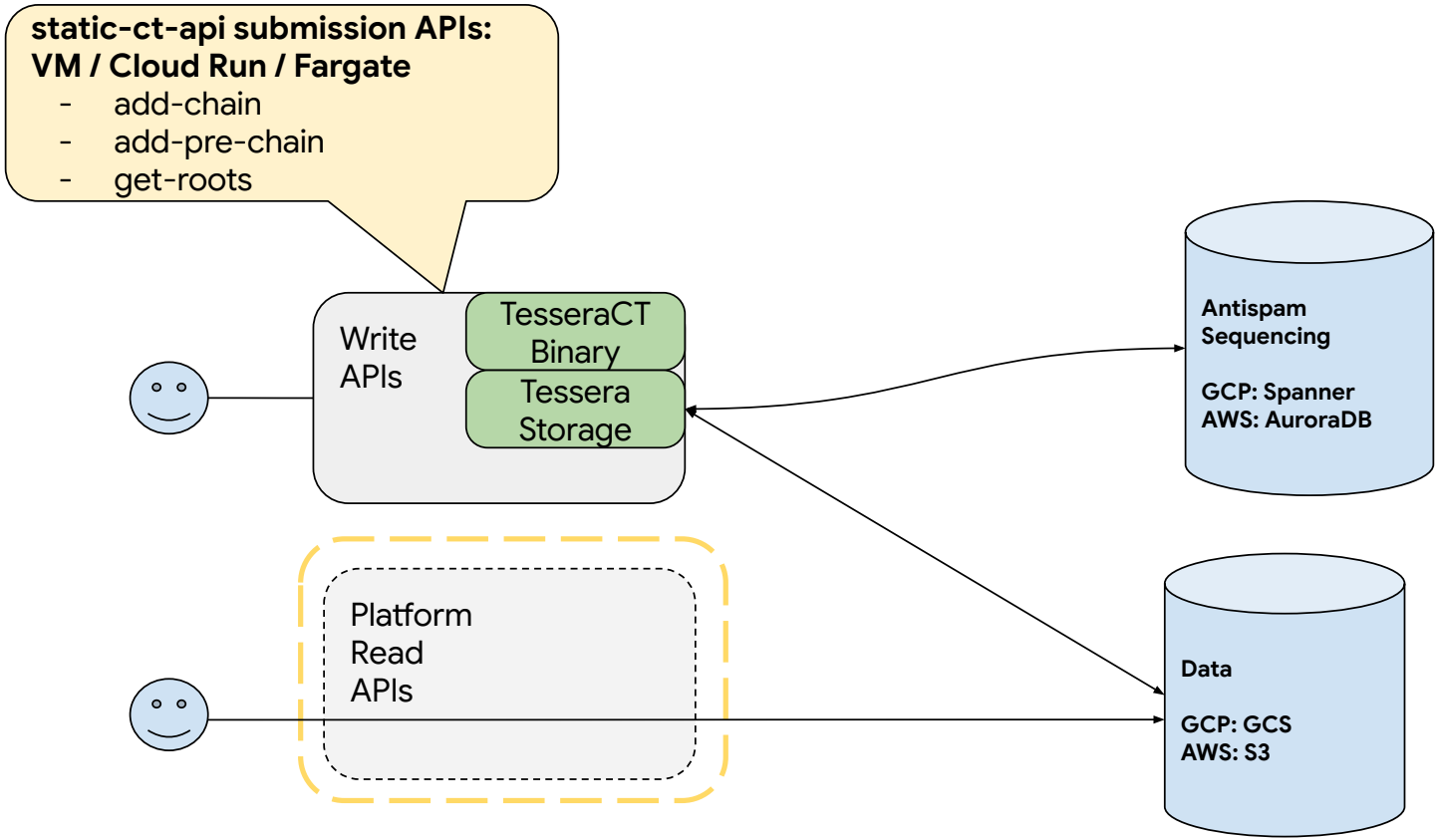
`issuer/`: CT specific read endpoint

Format is different

Checkpoint signature for
backward compatibility

Data format

TesseraCT: infrastructure

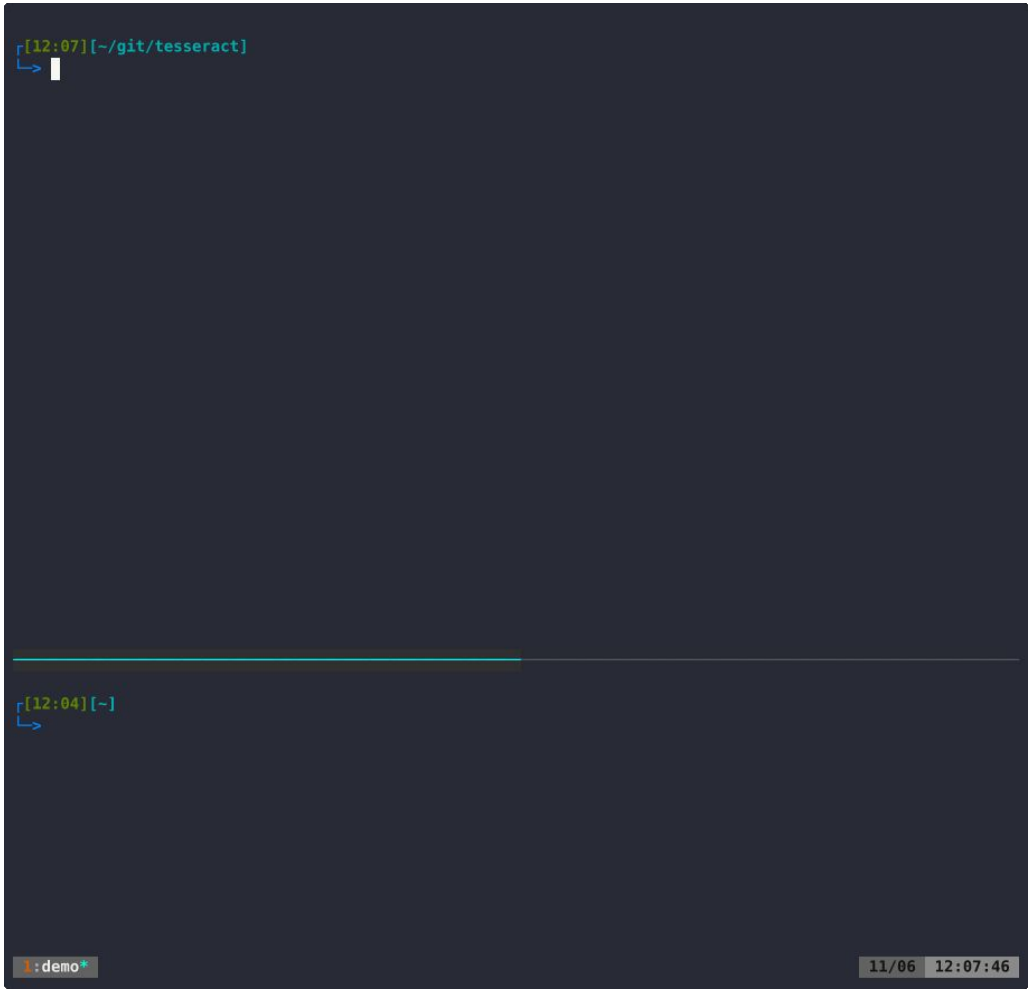


TesseraCT: usability

Configured with terraform
↳ no admin server

Bringing a new log up takes a few minutes
↳ one log = Server+DB+Bucket

Reads go to S3 and GCS directly
↳ decoupled form writes



```
┌[12:07][~/git/tesseract]
└─>
```

```
┌[12:04][~]
└─>
```

Links and questions

https://transparency.dev
https://blog.transparency.dev

https://github.com/transparency-dev/tessera
https://github.com/transparency-dev/tesseract
https://c2sp.org

transparency-dev slack

rogerng@ phboneff@