June 2, 2026

David Adrian
dadrian@

chrome

How PQC will work in PKI

What are you gonna do, send me 35kb of certs?

Product Manager, Chrome Security

David Adrian
dadrian@

Network Security
Memory Safety
Web Platform Security

Previously...

PhD @ University of Michigan

Cofounder, Censys

Principal Engineer, Nametag

https://dadrian.io

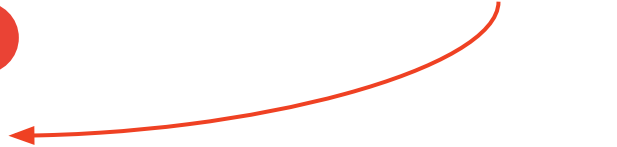
Cryptography in HTTPS

Quantum computers will break classical forms of public/private key (asymmetric) cryptography.

Encryption/Decryption. Encode messages such that a secret key is required to decode the message.
AES, ChaCha-Poly, Simon/Speck

Key Establishment. Securely select a key to use for encryption and decryption
Diffie-Hellman, RSA Encrypt, ECDH

Broken by future quantum computer



Authentication. Ensure the other party is the real thing, not an imposter.
Signatures, RSA Sign, ECDSA

Two Problems: ***Key Agreement*** and ***Authentication***

Cryptography in HTTPS

Quantum computers will break classical forms of public/private key (asymmetric) cryptography.

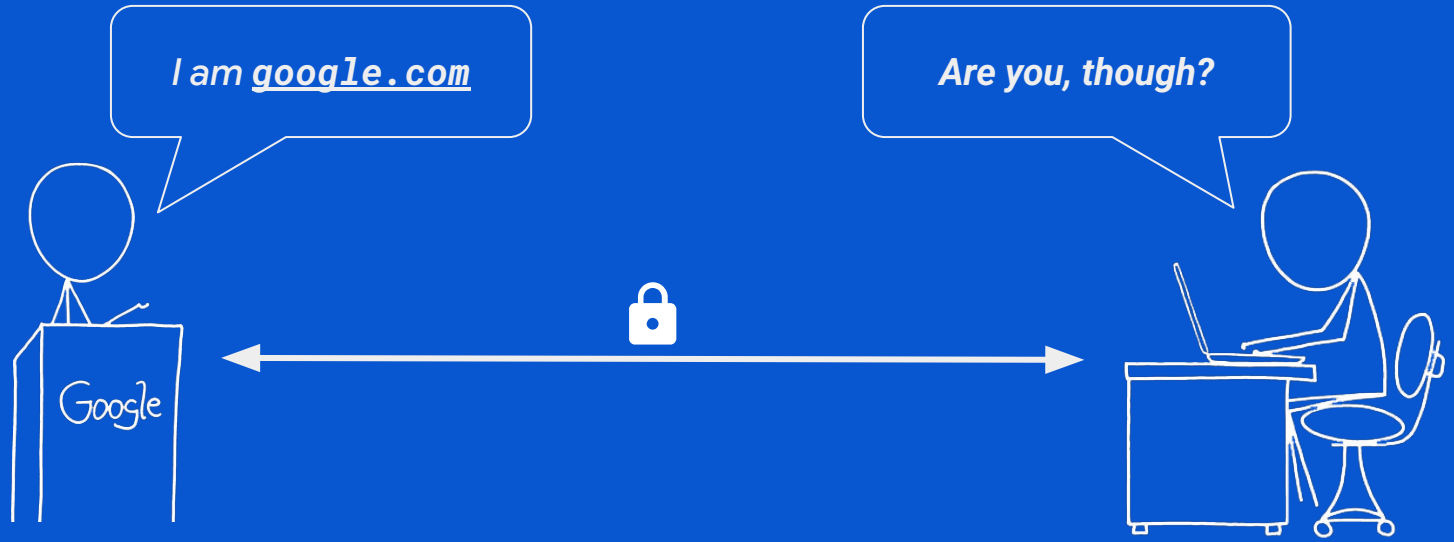
Encryption/Decryption. Encode messages such that a secret key is required to decode the message.
AES, ChaCha-Poly, Simon/Speck

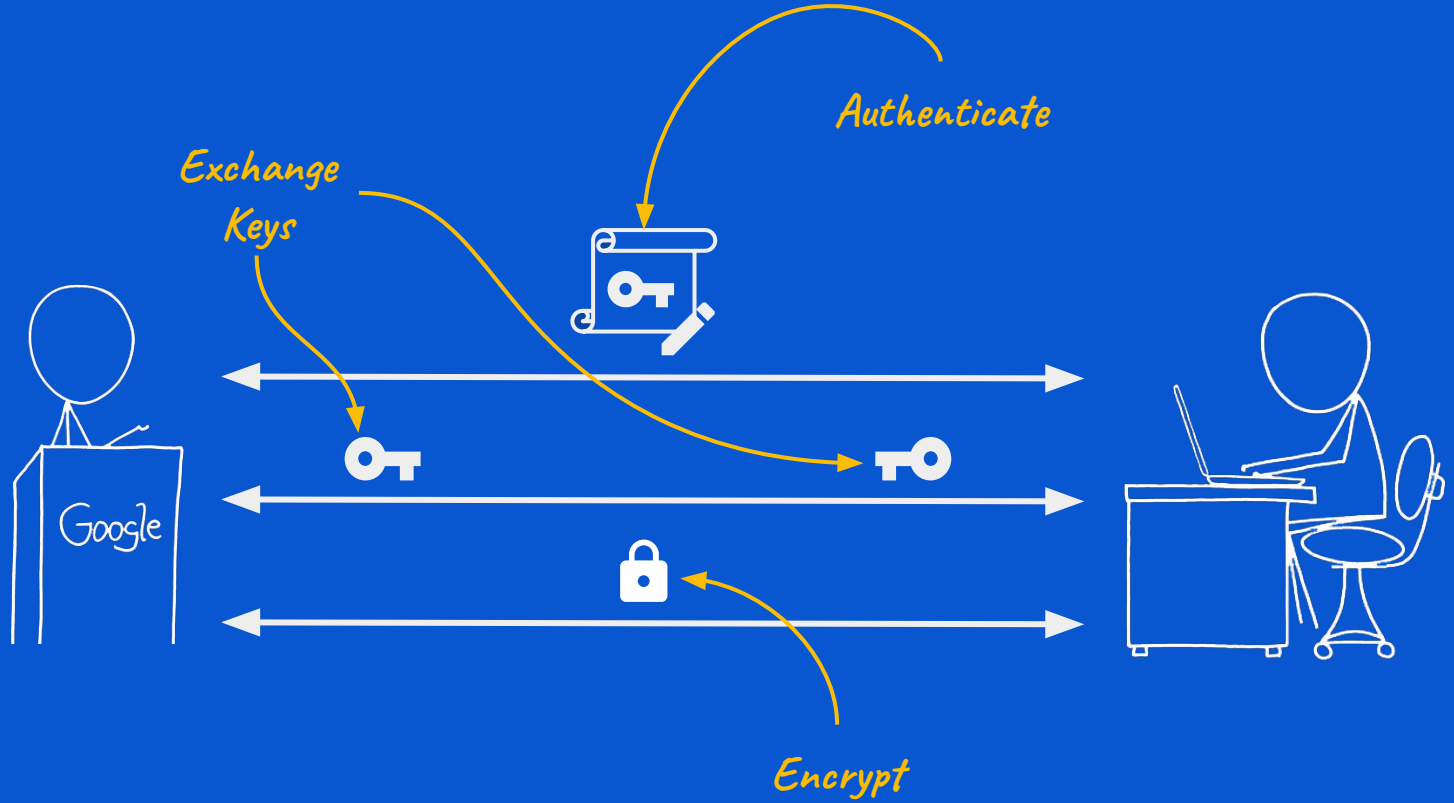
Key Establishment. Securely select a key to use for encryption and decryption
Diffie-Hellman, RSA Encrypt, ECDH

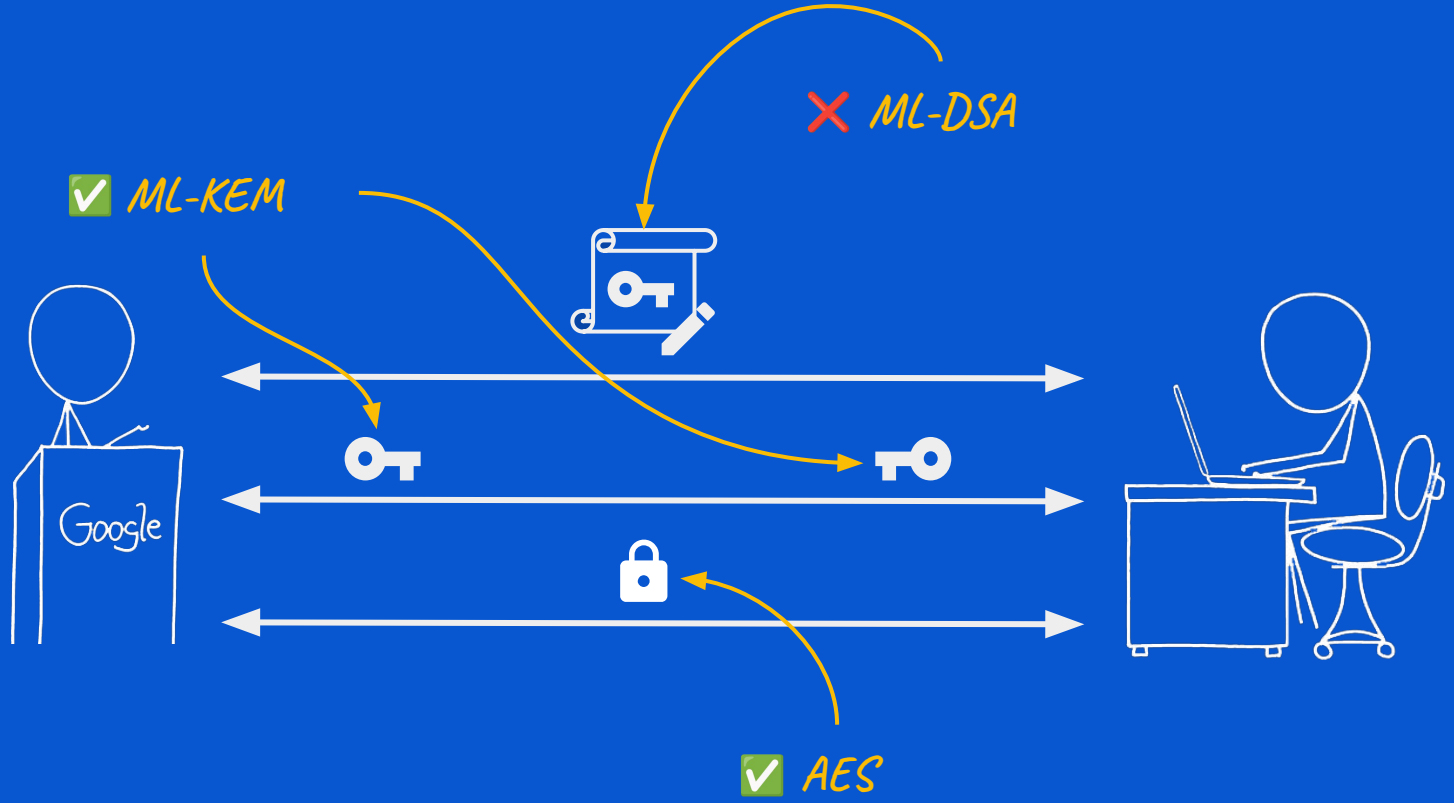
Already quantum resistant in Chrome!

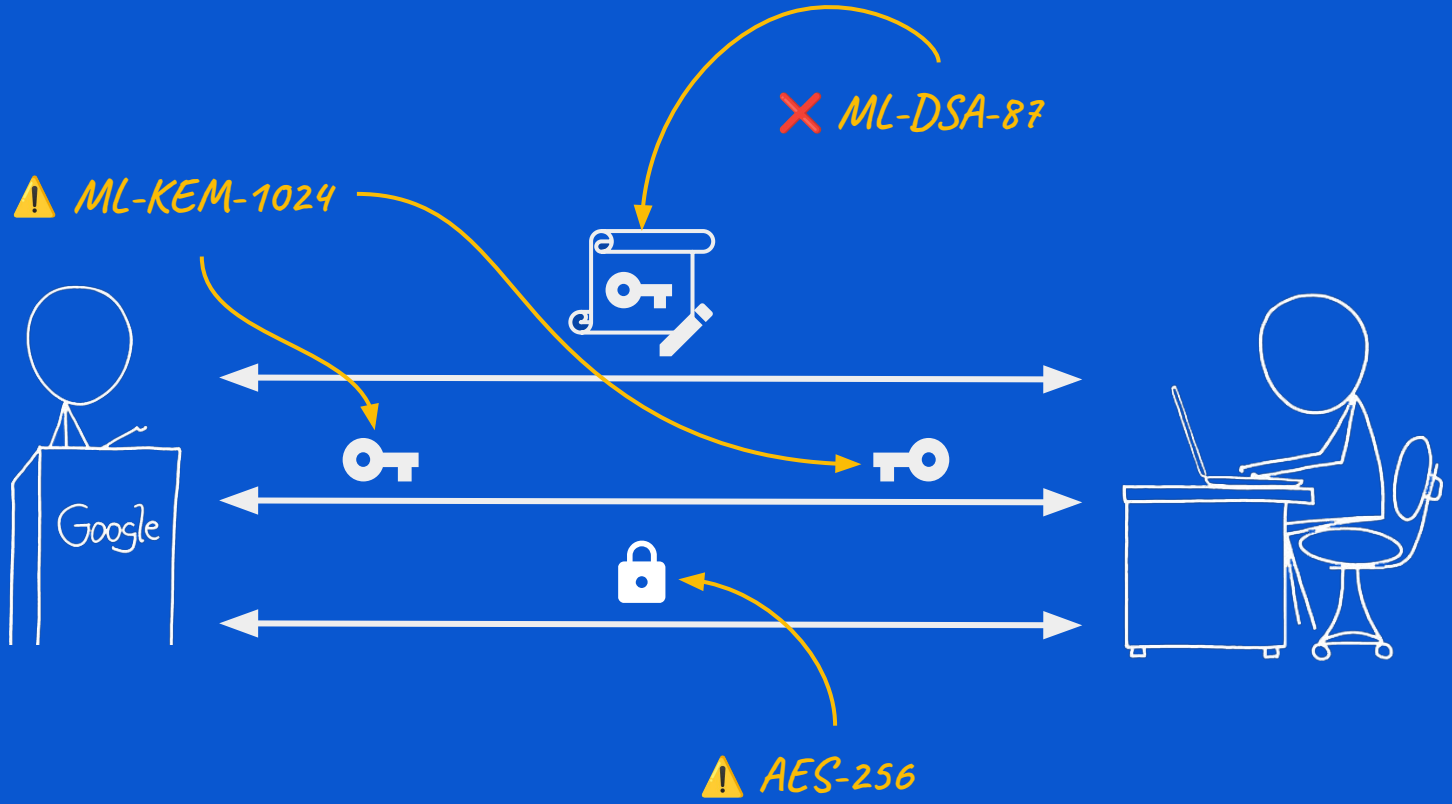
Authentication. Ensure the other party is the real thing, not an imposter.
Signatures, RSA Sign, ECDSA

Two Problems: ***Key Agreement*** and ***Authentication***


I am google.com
Are you, though?
Google


Authenticate
Exchange
Keys
Google
Encrypt


❌ ML-DSA
✅ ML-KEM
Google
✅ AES


ML-DSA-87
ML-KEM-1024
Google
AES-256

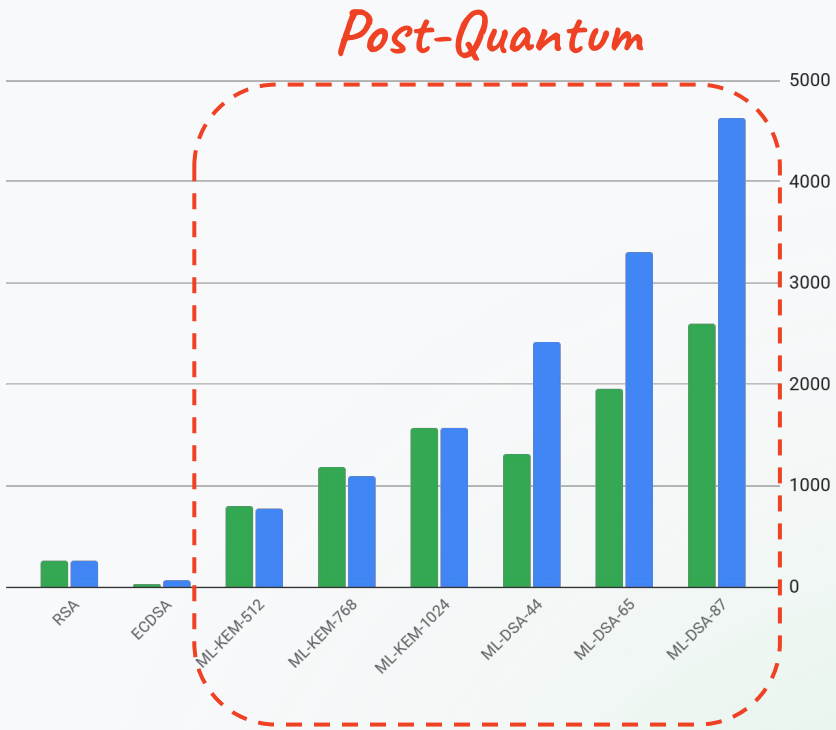
Post-quantum cryptography...
...*is really, really big*.

Cryptography now costs bytes.

A single certificate has multiple keys and signatures.

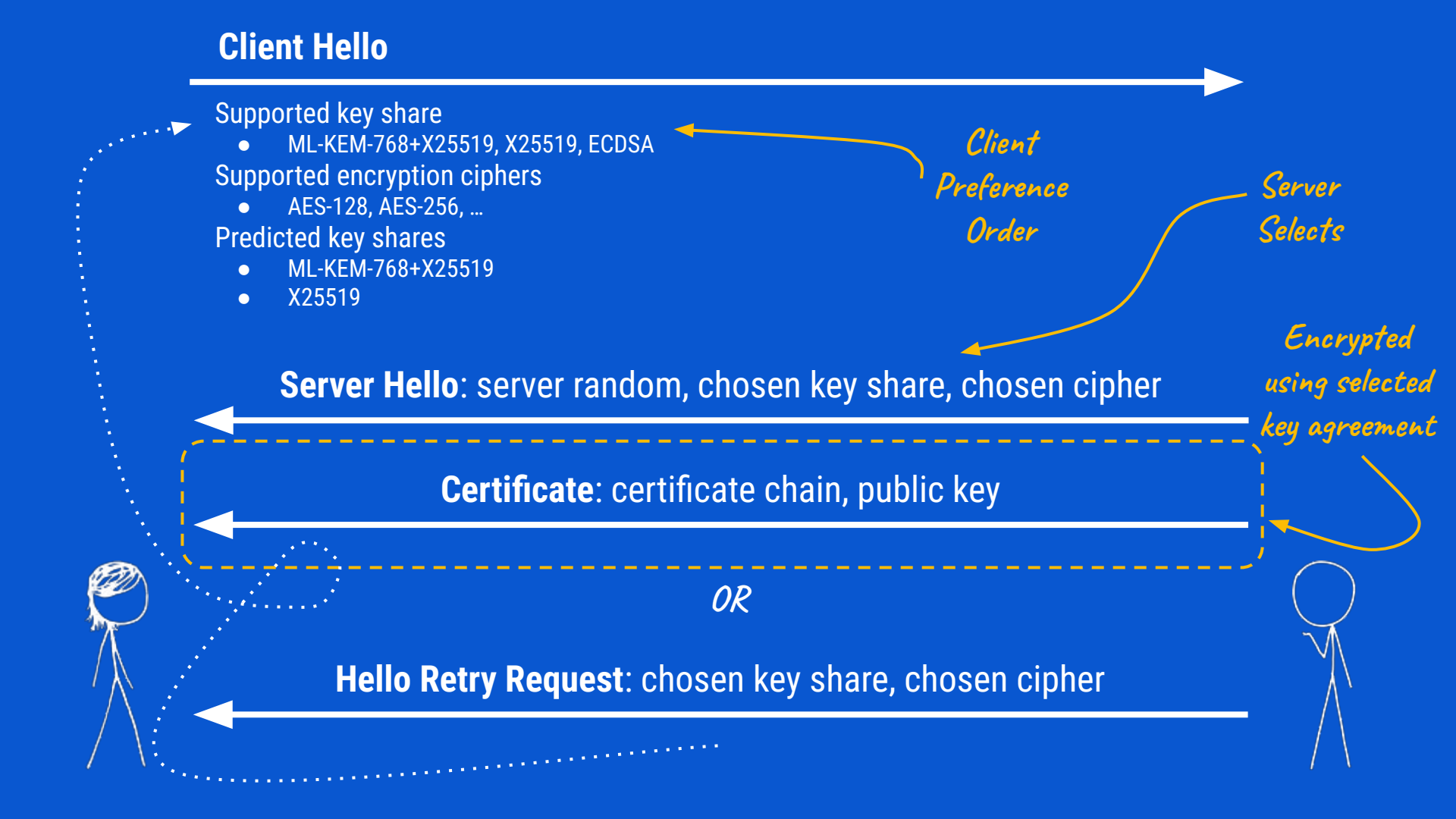
The green bar is the the size of the key, and the blue bar is the size of the signatures. Post-quantum is roughly a 30x increase in size.

A single TLS handshake usually has two keys and five signatures!



More bytes = *slow*

Key exchange and the TLS handshake


Client Hello
Supported key share
ML-KEM-768+X25519, X25519, ECDSA
Supported encryption ciphers
AES-128, AES-256, …
Predicted key shares
ML-KEM-768+X25519
X25519
Client Preference Order
Server Selects
Encrypted using selected key agreement
Server Hello: server random, chosen key share, chosen cipher
Certificate: certificate chain, public key
OR
Hello Retry Request: chosen key share, chosen cipher

PQC Impact on ClientHello

Pre-quantum

- Key agreement algorithm list is *small and cheap*
- Key share prediction is *small and cheap*
- ClientHello is small and cheap (1 packet)

`HelloRetryRequest` is 1 RTT and 3 packets, which is not good but workable.

Post-quantum

- Key agreement algorithm list is *small and cheap*
- Key share prediction is ***big***
- ClientHello is ***2 packets***

`HelloRetryRequest` is > 1 RTT and 5 packets, which is definitely bad.

Corollary: Only get one PQC key share prediction!

Takeaways for the ClientHello

Only get one prediction

PQC is too big to be able to make more than one key share prediction in the handshake.

HelloRetries for others

The Internet has decided on hybrid ML-KEM-768+X25519 as the default. That will always be predicted, for better or for worse. It is broadly adopted (50%+)

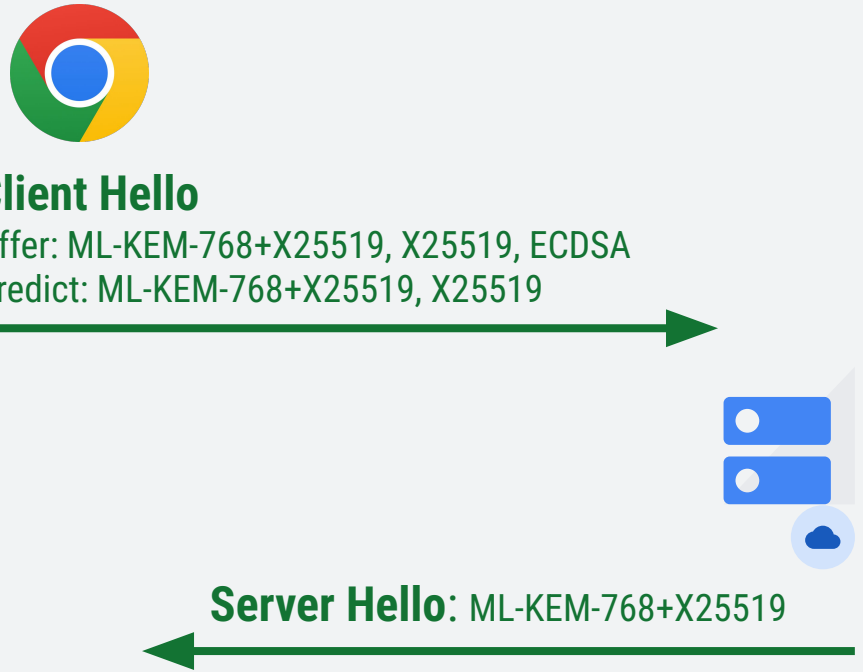
Explicit configuration

Supporting “stronger” (CNSA2) variants simultaneously with the defaults will require explicit configuration *of clients*. Explicit configuration is required to get CNSA2 with Google servers.

ML-KEM in Chrome

Chrome *offers, prefers, and predicts* hybrid ML-KEM **by default** on desktop platforms since Chrome 131 and Android since Chrome 133.

Google Servers *prefer* ML-KEM **by default** for *Google properties* platforms since around the release of Chrome 116.



ML-KEM-1024 in Chrome

Chrome *offers, prefers, and predicts* hybrid ML-KEM **by default** on desktop platforms since Chrome 131 and Android since Chrome 133.

Google Servers *prefer* ML-KEM **by default** for *Google properties* platforms since around the release of Chrome 116.

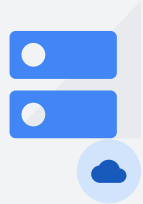
Google Servers *defer to the client* for ML-KEM-1024 vs ML-KEM-768+X25519 **by default** for Google properties.

Chrome can be **configured** to *offer, prefer, and predict* ML-KEM-1024 via enterprise policy since Chrome 144.

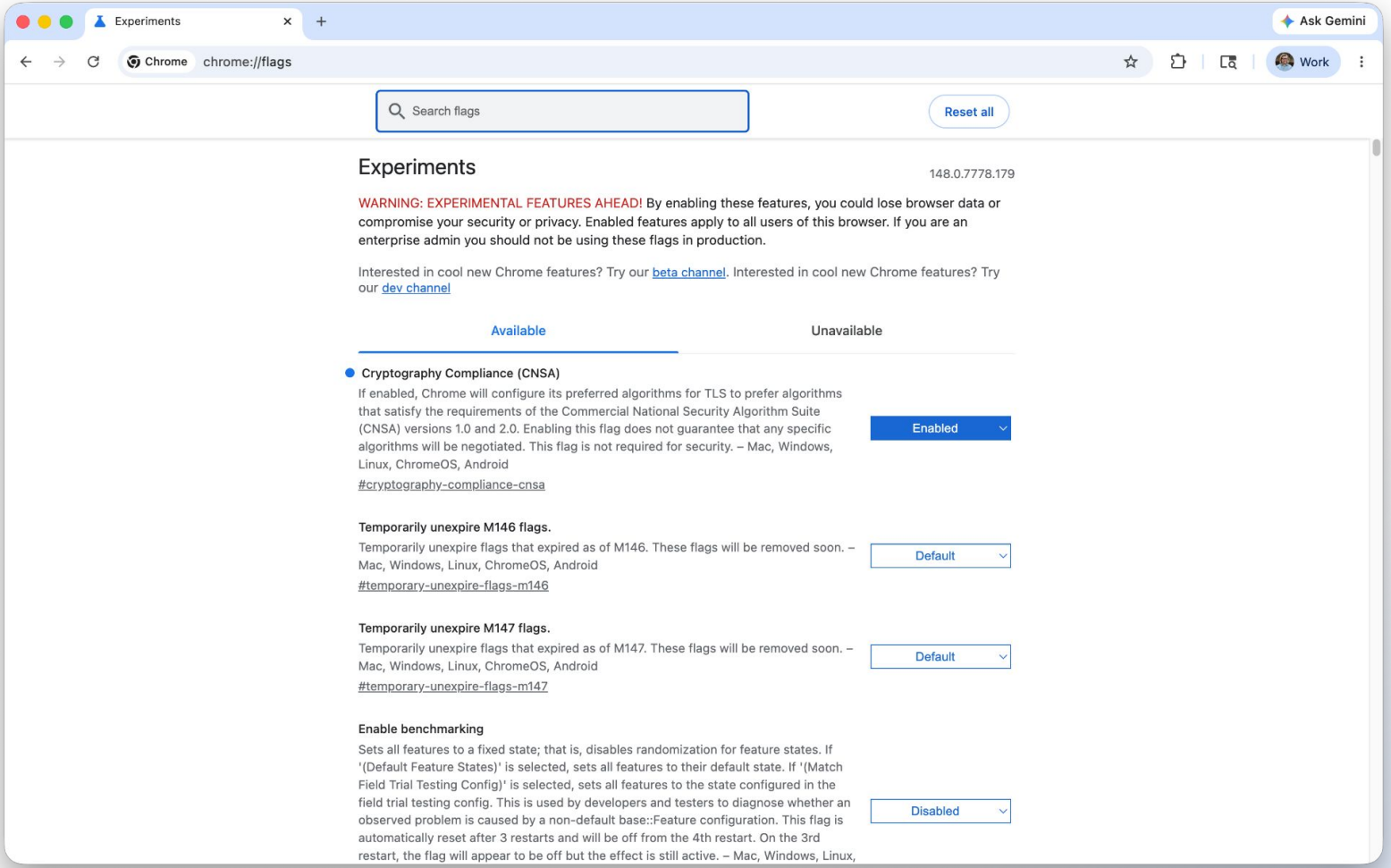
Client Hello

Offer: ML-KEM-1024, ML-KEM-768+X25519, X25519, ECDSA

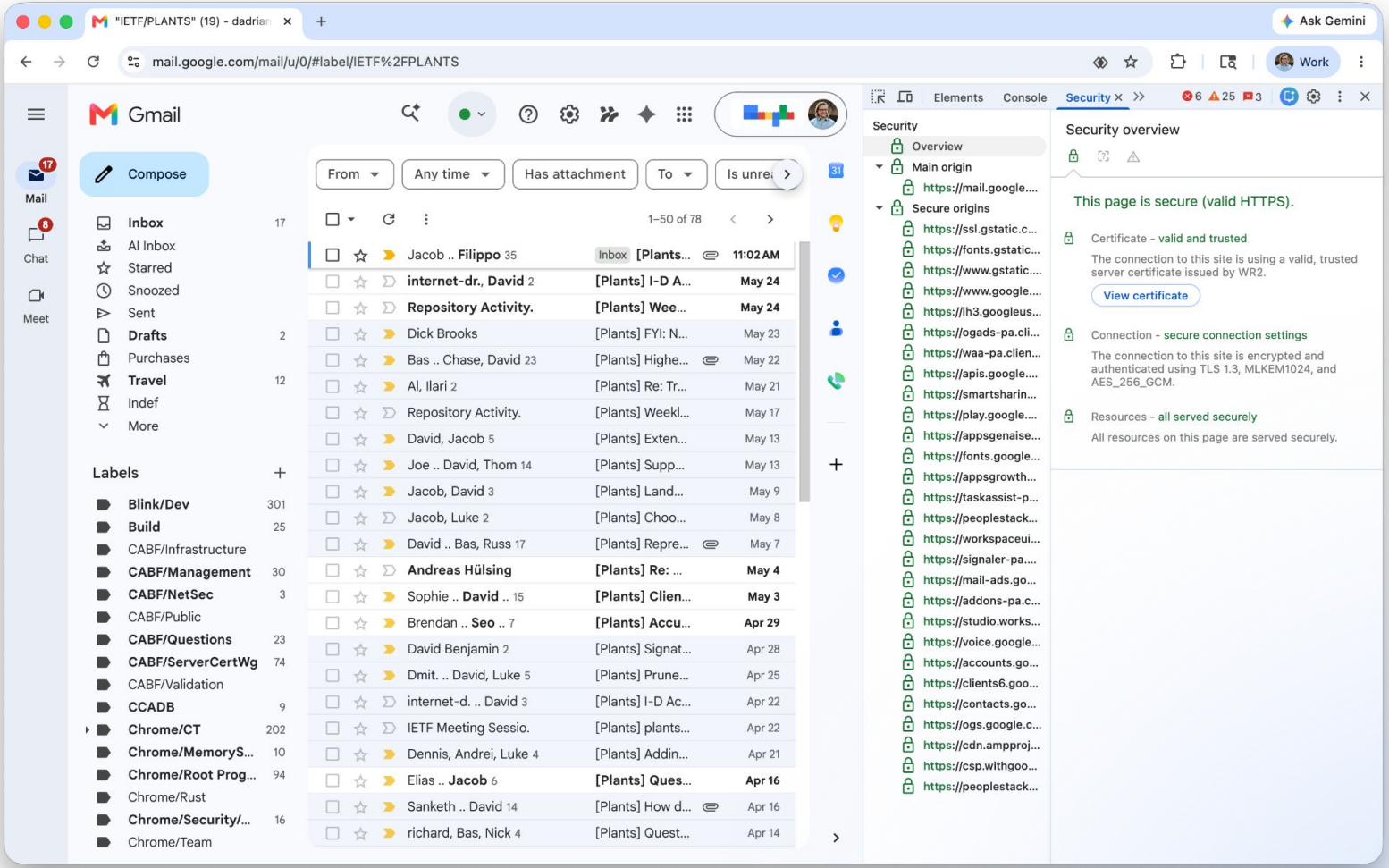
Predict: ML-KEM-768+X25519, X25519



Hello Retry Request: ML-KEM-1024


Experiments
Ask Gemini
Chrome
chrome://flags
Work
Search flags
Reset all
Experiments
148.0.7778.179
WARNING: EXPERIMENTAL FEATURES AHEAD! By enabling these features, you could lose browser data or compromise your security or privacy. Enabled features apply to all users of this browser. If you are an enterprise admin you should not be using these flags in production.
Interested in cool new Chrome features? Try our beta channel. Interested in cool new Chrome features? Try our dev channel
Available
Unavailable
Cryptography Compliance (CNSA)
If enabled, Chrome will configure its preferred algorithms for TLS to prefer algorithms that satisfy the requirements of the Commercial National Security Algorithm Suite (CNSA) versions 1.0 and 2.0. Enabling this flag does not guarantee that any specific algorithms will be negotiated. This flag is not required for security. – Mac, Windows, Linux, ChromeOS, Android
#cryptography-compliance-cnsa
Enabled
Temporarily unexpire M146 flags.
Temporarily unexpire flags that expired as of M146. These flags will be removed soon. – Mac, Windows, Linux, ChromeOS, Android
#temporary-unexpire-flags-m146
Default
Temporarily unexpire M147 flags.
Temporarily unexpire flags that expired as of M147. These flags will be removed soon. – Mac, Windows, Linux, ChromeOS, Android
#temporary-unexpire-flags-m147
Default
Enable benchmarking
Sets all features to a fixed state; that is, disables randomization for feature states. If '(Default Feature States)' is selected, sets all features to their default state. If '(Match Field Trial Testing Config)' is selected, sets all features to the state configured in the field trial testing config. This is used by developers and testers to diagnose whether an observed problem is caused by a non-default base::Feature configuration. This flag is automatically reset after 3 restarts and will be off from the 4th restart. On the 3rd restart, the flag will appear to be off but the effect is still active. – Mac, Windows, Linux,
Disabled

chrome


"IETF/PLANTS" (19) - dadrian
Ask Gemini
mail.google.com/mail/u/0/#label/IETF%2FPLANTS
Work
Gmail
Compose
Mail
Chat
Meet
Inbox 17
AI Inbox
Starred
Snoozed
Sent
Drafts 2
Purchases
Travel 12
Indef
More
Labels
Blink/Dev 301
Build 25
CABF/Infrastructure
CABF/Management 30
CABF/NetSec 3
CABF/Public
CABF/Questions 23
CABF/ServerCertWg 74
CABF/Validation
CCADB 9
Chrome/CT 202
Chrome/MemoryS... 10
Chrome/Root Prog... 94
Chrome/Rust
Chrome/Security/... 16
Chrome/Team
From
Any time
Has attachment
To
1–50 of 78
Jacob .. Filippo 35 Inbox [Plants... 11:02 AM
internet-dr., David 2 [Plants] I-D A... May 24
Repository Activity. [Plants] Wee... May 24
Dick Brooks [Plants] FYI: N... May 23
Bas .. Chase, David 23 [Plants] Highe... May 22
Al, Ilari 2 [Plants] Re: Tr... May 21
Repository Activity. [Plants] Weekl... May 17
David, Jacob 5 [Plants] Exten... May 13
Joe .. David, Thom 14 [Plants] Supp... May 13
Jacob, David 3 [Plants] Land... May 9
Jacob, Luke 2 [Plants] Choo... May 8
David .. Bas, Russ 17 [Plants] Repre... May 7
Andreas Hülsing [Plants] Re: ... May 4
Sophie .. David .. 15 [Plants] Clien... May 3
Brendan .. Seo .. 7 [Plants] Accu... Apr 29
David Benjamin 2 [Plants] Signat... Apr 28
Dmit. .. David, Luke 5 [Plants] Prune... Apr 25
internet-d. .. David 3 [Plants] I-D Ac... Apr 22
IETF Meeting Sessio. [Plants] plants... Apr 22
Dennis, Andrei, Luke 4 [Plants] Addin... Apr 21
Elias .. Jacob 6 [Plants] Ques... Apr 16
Sanketh .. David 14 [Plants] How d... Apr 16
richard, Bas, Nick 4 [Plants] Quest... Apr 14
Elements
Console
Security
Security
Overview
Main origin
https://mail.google....
Secure origins
https://ssl.gstatic.c...
https://fonts.gstatic...
https://www.gstatic....
https://www.google....
https://lh3.googleus...
https://ogads-pa.cli...
https://waa-pa.clien...
https://apis.google....
https://smartsharin...
https://play.google....
https://appsgenaise...
https://fonts.google...
https://appsgrowth...
https://taskassist-p...
https://peoplestack...
https://workspaceui...
https://signaler-pa....
https://mail-ads.go...
https://addons-pa.c...
https://studio.works...
https://voice.google...
https://accounts.go...
https://clients6.goo...
https://contacts.go...
https://ogs.google.c...
https://cdn.ampproj...
https://csp.withgoo...
https://peoplestack...
Security overview
This page is secure (valid HTTPS).
Certificate - valid and trusted
The connection to this site is using a valid, trusted server certificate issued by WR2.
View certificate
Connection - secure connection settings
The connection to this site is encrypted and authenticated using TLS 1.3, MLKEM1024, and AES_256_GCM.
Resources - all served securely
All resources on this page are served securely.

chrome

Pure CNSA2 Servers

If you have a service that you want to run as CNSA2-only, where you do not care about performance and want ML-KEM-1024 for everybody behind an `HelloRetryRequest` but without client configuration, **contact your Cloud PoC**.

Ecosystem Needs

We don't want people to see the bigger number and configure ML-KEM-1024 for no reason, so we are not currently including it in the default configuration *at the back of the list*.

We currently want to push people to hybrid ML-KEM-768.

Government and Compliance

If you're fortunate enough to be able to run a CNSA2-only service with no concern for extra bytes, you only need ML-KEM-1024 to be *at the end of the supported list*, unlike Google servers that need it in front.

We will make this change to Chrome when there is demand.

PQC PKI

Post-quantum cryptography...

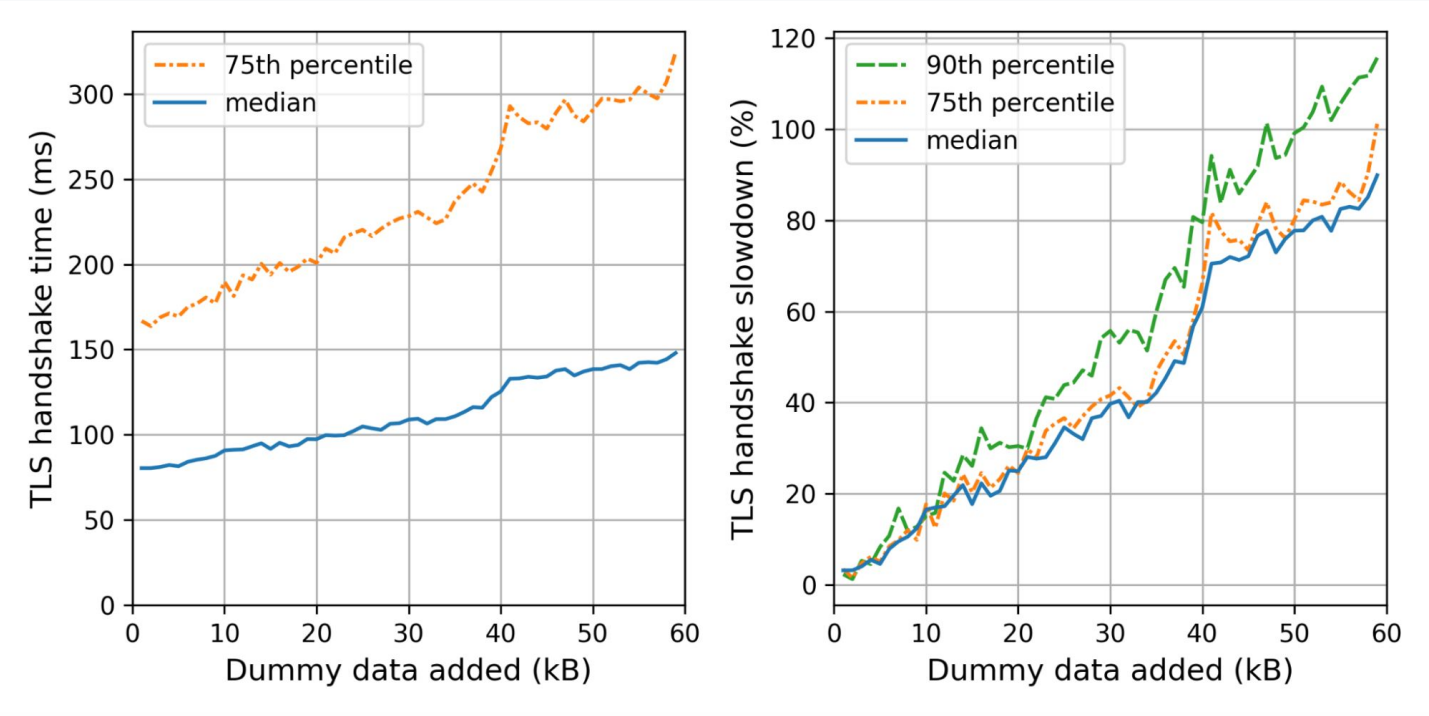
...is really, really big

..... and it only gets worse!

Chonky X.509

What happens if we naively copy ML-DSA into existing X.509 certificate chains?

Scheme	Total Size	Change
Chonky ML-DSA-44	16kb	+12kb
Chonky ML-DSA-87	23kb	+19kb



Data from Cloudflare "State of PQ 2024"

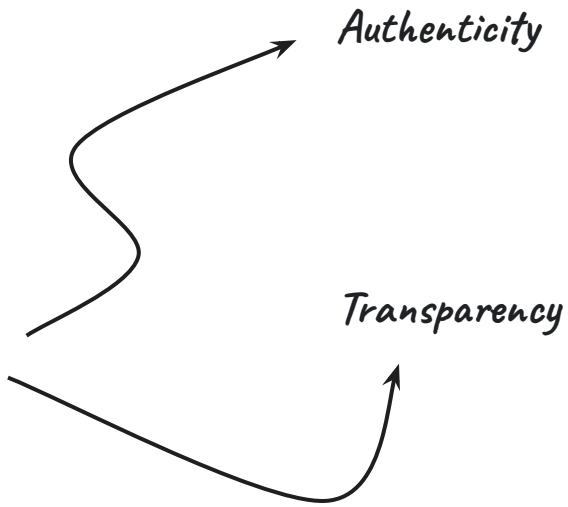
Cryptography in HTTPS

Quantum computers will break classical forms of public/private key (asymmetric) cryptography.

Encryption/Decryption. Encode messages such that a secret key is required to decode the message.
AES, ChaCha-Poly, Simon/Speck

Key Establishment. Securely select a key to use for encryption and decryption
Diffie-Hellman, RSA Encrypt, ECDH

Authentication. Ensure the other party is the real thing, not an imposter.
Signatures, RSA Sign, ECDSA



Two Problems: ***Key Agreement*** and ***Authentication***

Authentication

For HTTPS, authentication binds the key to the site name, and the handshake to the key.

Bind the key to the site name

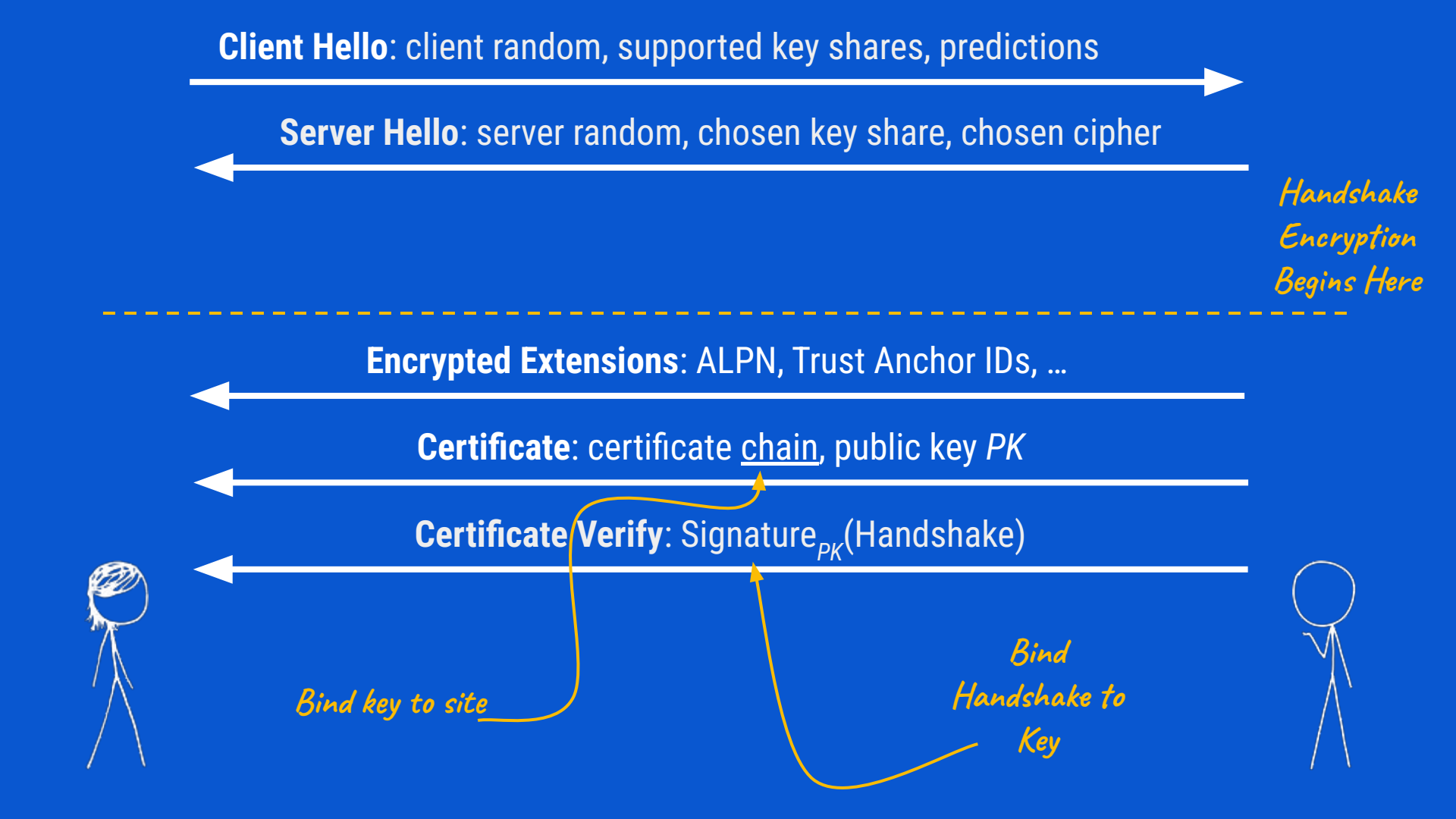
Is this specific key associated with this specific site?

“Is this the correct public key?”

Bind the handshake to the key

Assuming you have the correct key, did the site operator prove that they are the owner of it?

“Does the server have control the private key?”


Client Hello: client random, supported key shares, predictions
Server Hello: server random, chosen key share, chosen cipher
Handshake Encryption Begins Here
Encrypted Extensions: ALPN, Trust Anchor IDs, …
Certificate: certificate chain, public key *PK*
Certificate Verify: Signature$_{PK}$(Handshake)
Bind key to site
Bind Handshake to Key

Transparency

Has the existence of the certificate been logged publicly, i.e. **is the binding from the name to key publicly known**?

Ensure all certificates are known.

Prevent one-off malicious certificates (e.g. from compromising a CA)

“Has this certificate been disclosed?”

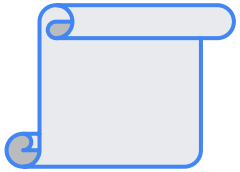
Ensure ecosystem health.

Full knowledge of all issuance is useful for maintaining and evolving the security of the Web PKI

“Are CAs following the rules they wrote for themselves?”

Log what you issue

Transparency for the Web PKI is provided by Certificate Transparency Logs.



A set of append-only logs

There is a set of **Certificate Transparency logs** ran by interested parties. All logs MUST accept certificates from any CA. Logs are cryptographically verifiable trees known as a Merkle Tree.

Log at issuance

CAs submit precerts at issuance time to multiple logs, and achieve of quorum of inclusion promises in the form of Signed Certificate Timestamps (SCTs), which are then included in the final certificate.

Enforced by browsers

Browsers require multiple SCTs from trusted Certificate Transparency logs in order to consider a certificate trusted.

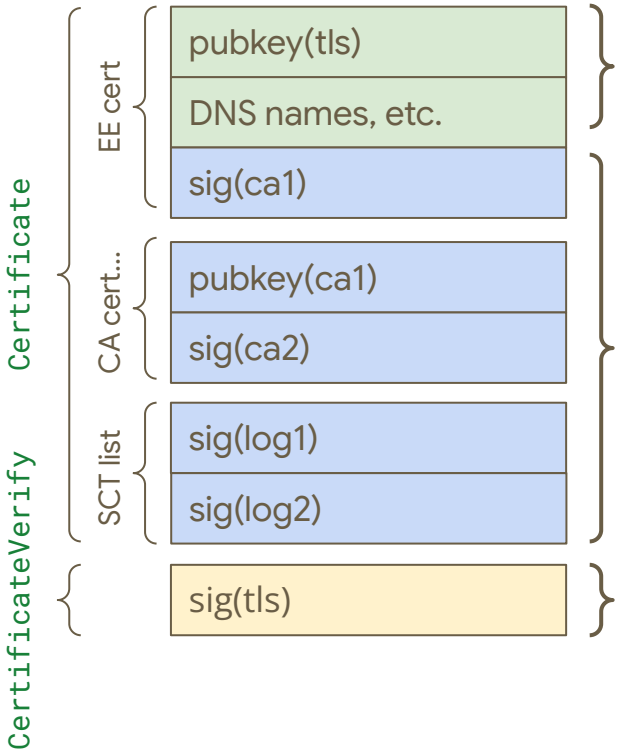
Problems with CT

Who runs the logs? CAs are not incentivized to run a CT log, and the system doesn't work if every CA runs a log. Logs are expensive to run and difficult to operate.

Promises, not proof. Certificates contain a promise that a certificate will be logged, not proof. Detecting that a certificate actually gets logged is a complicated auditing process and can have privacy implications.

Lack of agility. The log list changes faster than a root store, but user agents loosely need to move in lockstep, because there is no "negotiation" of which SCTs are included. As a result, clients must have a "time-bomb" where they stop enforcing CT if they are out of date.

Keys and Signatures in pre-quantum TLS handshake



`Certificate` and `CertificateVerify` contain:

A TLS key and application identifier

Proof from the PKI binds the identifiers to the key

Using the TLS key

Proof is determined by the PKI

Key, identifiers, and usage are determined by the protocol

Merkle Tree Certificates

Let’s replace all those blue signatures with something more size-efficient.

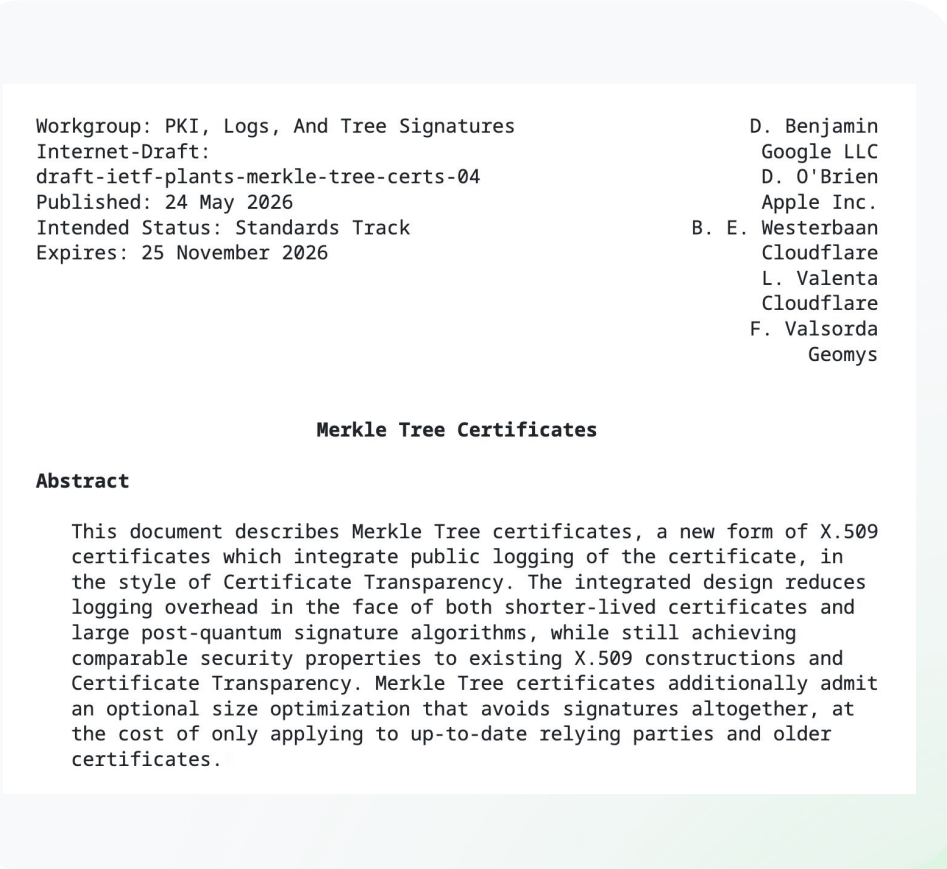
Let’s try to make the system work better than existing Certificate Transparency while we’re at it.

Our solution is **Merkle Tree Certificates (MTCs)**

IETF PLANTS Working Group

MTCs is an adopted draft

The PKIs, Logs, and Tree Signatures (PLANTS) Working Group is newly chartered and tackling the PQC-size problems for PKIs that need transparency.



Workgroup: PKI, Logs, And Tree Signatures
Internet-Draft:
draft-ietf-plants-merkle-tree-certs-04
Published: 24 May 2026
Intended Status: Standards Track
Expires: 25 November 2026

D. Benjamin
Google LLC
D. O'Brien
Apple Inc.
B. E. Westerbaan
Cloudflare
L. Valenta
Cloudflare
F. Valsorda
Geomys

Merkle Tree Certificates

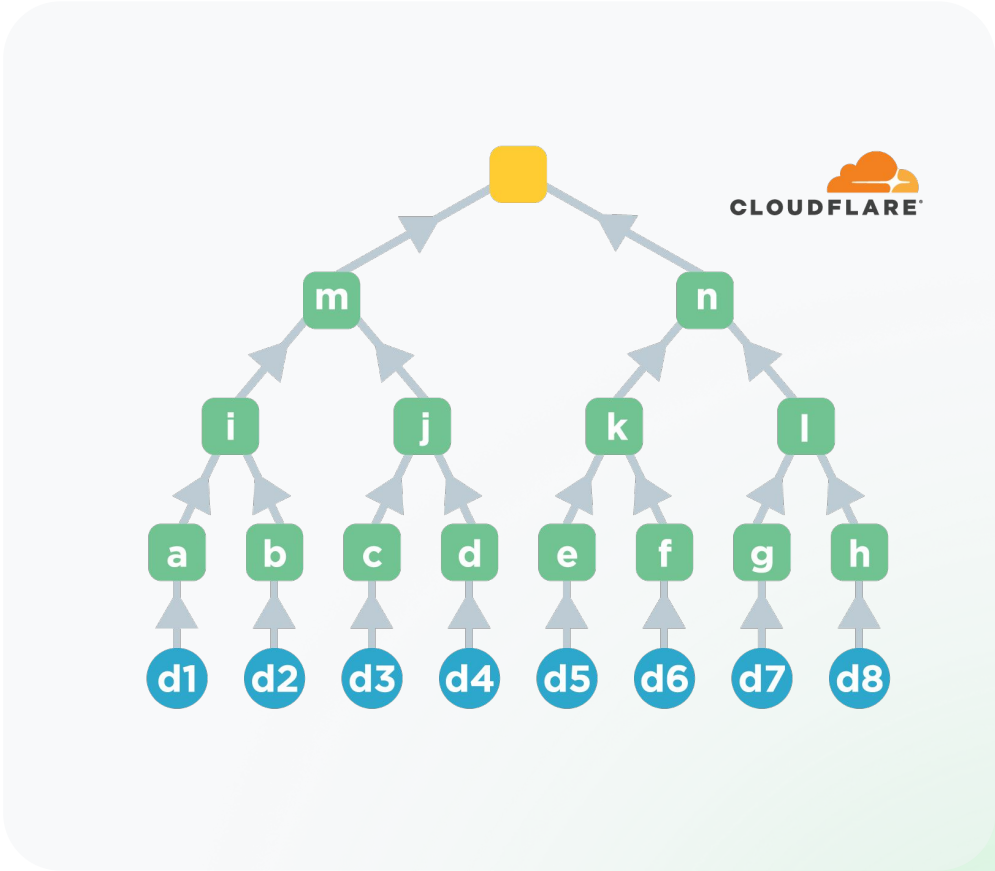
Abstract

This document describes Merkle Tree certificates, a new form of X.509 certificates which integrate public logging of the certificate, in the style of Certificate Transparency. The integrated design reduces logging overhead in the face of both shorter-lived certificates and large post-quantum signature algorithms, while still achieving comparable security properties to existing X.509 constructions and Certificate Transparency. Merkle Tree certificates additionally admit an optional size optimization that avoids signatures altogether, at the cost of only applying to up-to-date relying parties and older certificates.

MTCs exist in Chrome today

Experiment with Cloudflare

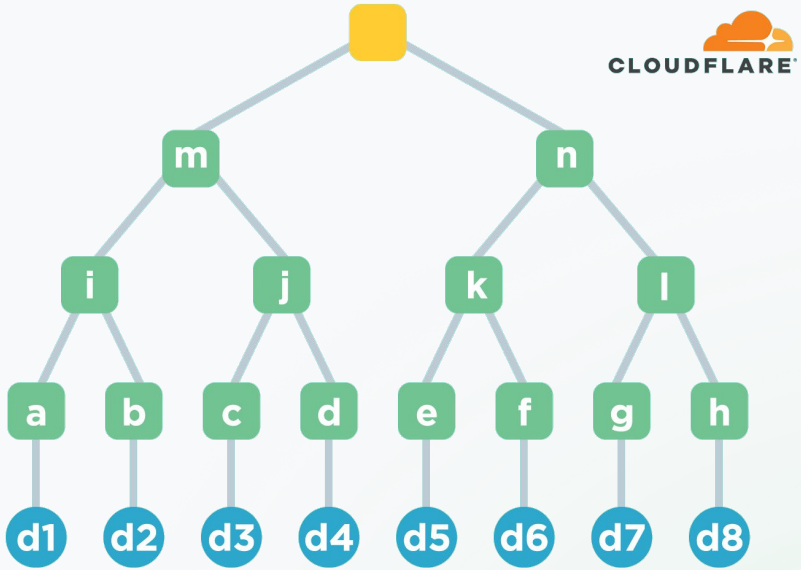
Chrome and Cloudflare are experimenting with a pre-quantum early draft of MTCs right now.



Merkle Tree inclusion proofs

Build a hash chain to the root

In a Merkle Tree, every interior node is the hash of its children. You can prove inclusion of a leaf by showing the nodes required to build a hash chain path from the leaf to the root.



Issue by logging

Certificate transparency was layered on *after the fact*. Merkle Tree Certificates integrate transparency *at issuance time*

A traditional CA first **signs**, *then* **logs** the result and collects SCTs

CT was layered on top of existing PKI
Complications stem from the gap between issuance and logging

A Merkle Tree CA first **logs**, *then* **signs** a checkpoint and collects cosignatures

Each CA runs a separate *issuance log*
Independent *mirrors* and *witnesses* co-sign log state

What's in an MTC?

An MTC is just a regular X.509 certificate with a funny signature algorithm.

A CA signature becomes a *signed checkpoint*

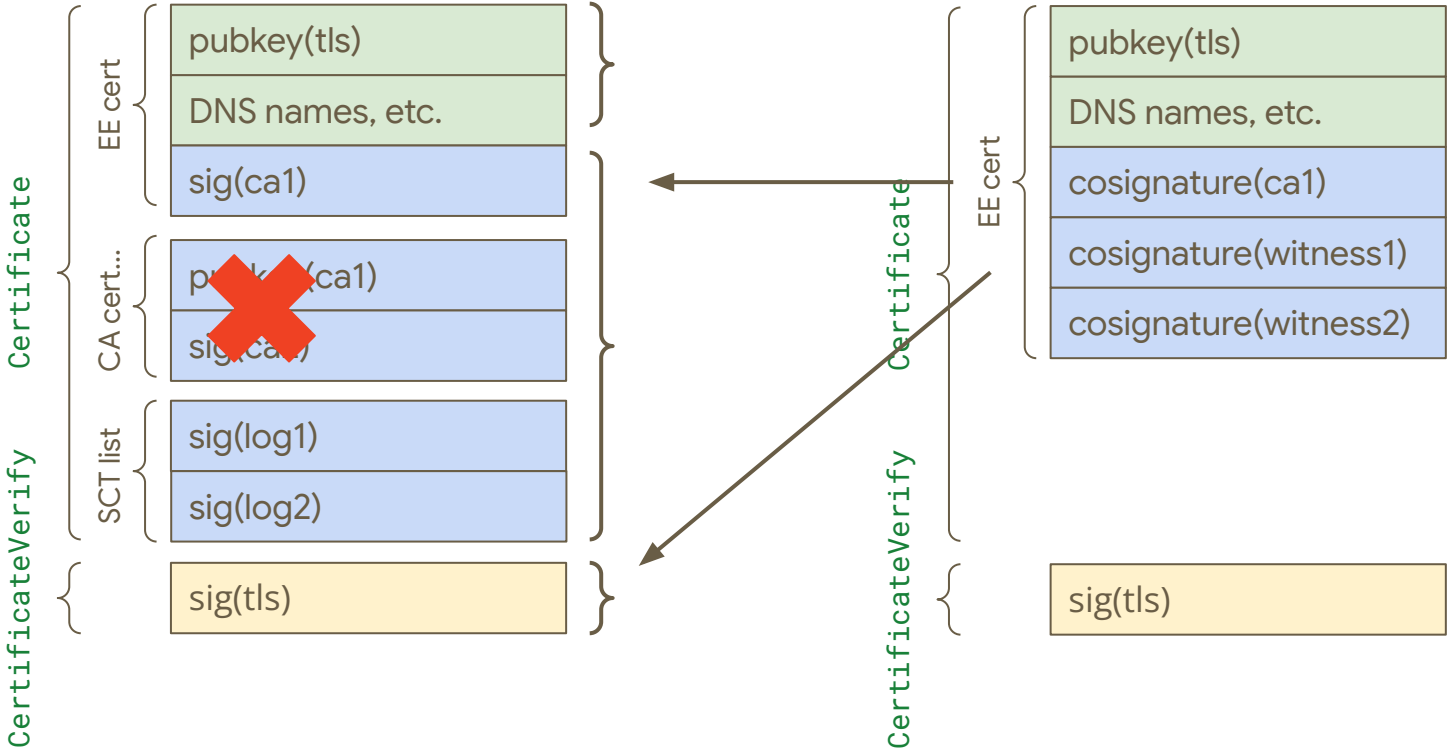
- A checkpoint is just a list of hashes, starting at the certificate hash and ending at the root of the Merkle Tree

SCTs become *cosignatures*

- *Cosignatures* are also just signed checkpoints, they're just signed by *witnesses*
- A witness is like a CT log, but instead of logging, they just attest that the CAs log is well-formed and append-only

Trick: All of this can go into the `signature` field on an X.509 certificate!

Standalone MTCs vs Pre-Quantum



Landmark

A landmark is a *checkpoint* that everyone agrees is worth knowing.

MTC CAs periodically announce checkpoints as a *landmark checkpoint*

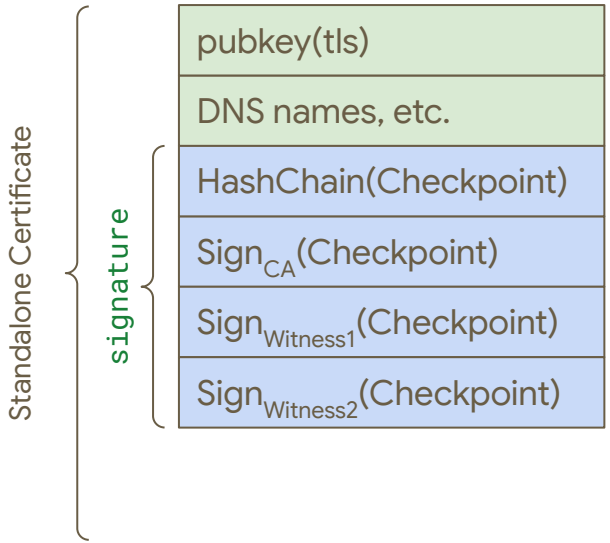
- Landmarks can be issued every hour, or every N checkpoints, etc.
- A landmark is just a checkpoint.
- A checkpoint is just the state of the tree (a root hash)

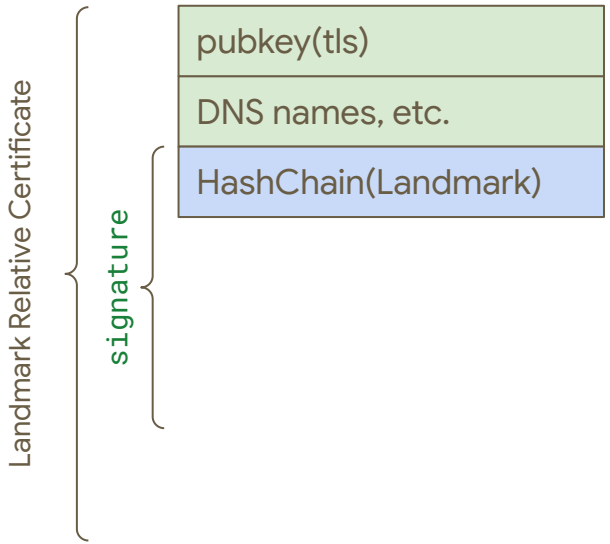
Landmarks can be verified and distributed out-of-band

- Browser operators (Google servers) constantly verify MTC CA landmarks
- Browser operators distribute verified landmark hashes to browser clients (e.g. via Chrome Component Updater)

Trick: A hash chain to a known landmark is proof of issuance!

Standalone vs Landmark-Relative MTCs





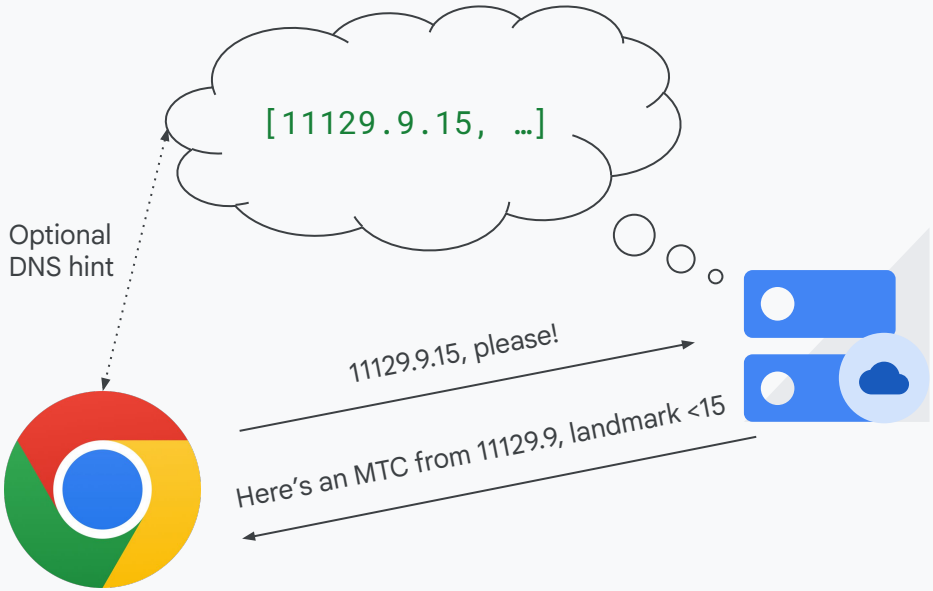
Trick: A validated landmark requires no signatures!

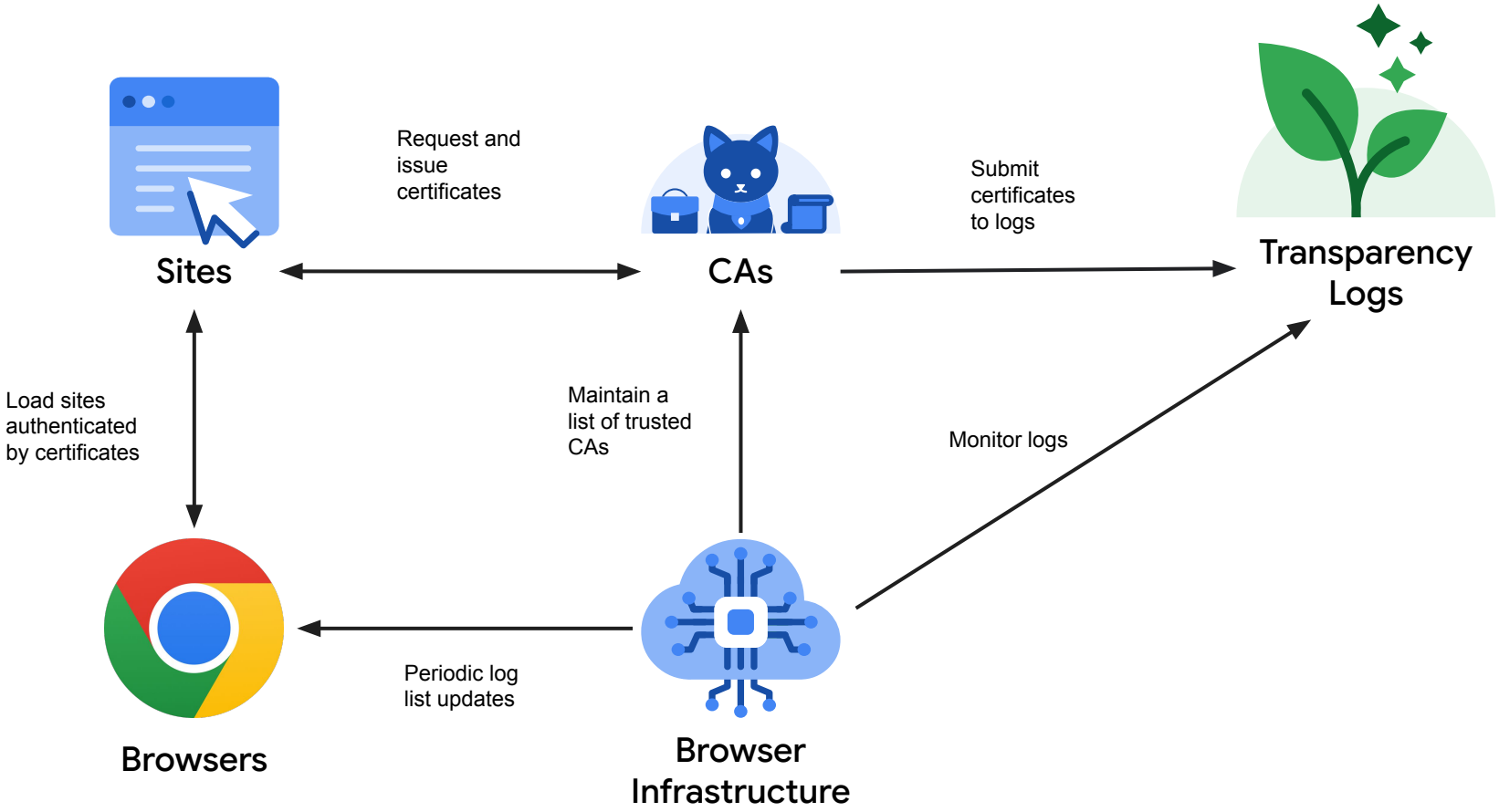
Negotiating MTCs

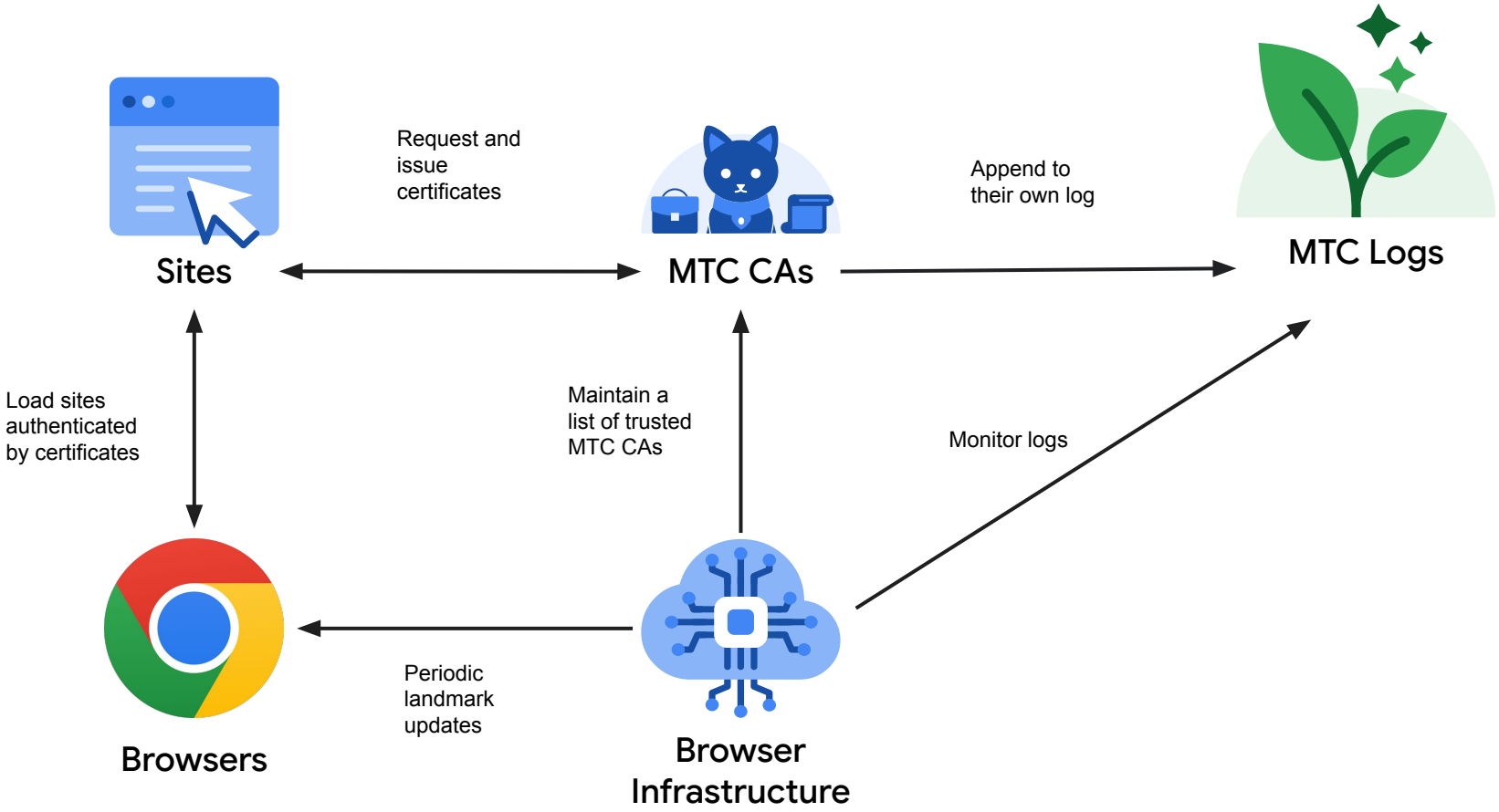
Trust Anchor Negotiation

We have a separate draft in the TLS working group about negotiating certificates based on contents of a trust store, known as **Trust Anchor Identifiers**.

We leverage this draft to indicate support for specific MTC CAs and indicate the latest landmark known to a client.




Request and issue certificates
Submit certificates to logs
Sites
CAs
Transparency Logs
Load sites authenticated by certificates
Maintain a list of trusted CAs
Monitor logs
Periodic log list updates
Browsers
Browser Infrastructure


Request and issue certificates
Append to their own log
Sites
MTC CAs
MTC Logs
Load sites authenticated by certificates
Maintain a list of trusted MTC CAs
Monitor logs
Periodic landmark updates
Browsers
Browser Infrastructure

More Fun Tricks!

Everything just looks like normal X.509 to clients, because it is.

If you don't care about transparency, you can just verify the CA signature

- Only verify the CA cosignature in the signature field, and ignore the cosignature.
- This only requires shipping the MTC CA as a trust anchor, just like you would ship a Chonky X.509 CA as a trust anchor.

If you don't want to acquire or distribute landmarks, a standalone certificate is just like Chonky X.509

- Never negotiate for a landmark certificate using TAI
- Always use standalone certificates
- This is identical traditional chains of depth 1
- No knowledge of Merkle Tree state required

Insight: Both Chonky X.509 and MTCs are "just" a new `signatureAlgorithm`

```
func VerifyCertificate(c *Certificate, issuer *Certificate) bool {
    sigAlg := c.tbsCertficiate.signatureAlgorithm
    switch (sigAlg) {
        case “rsa”:
            return VerifyRSASignature(c.signature, issuer)
        case “ecdsa”:
            return VerifyECDSASiganture(c.signature, issuer)
        case “ml-dsa”:
            return VerifyMLDSASignature(c.signature, issuer)
        case “mtc-ml-dsa”:
            return VerifyMTC(c.signature, issuer)
    }
}
```

Paxos, when presented in plain english, is quite simple.

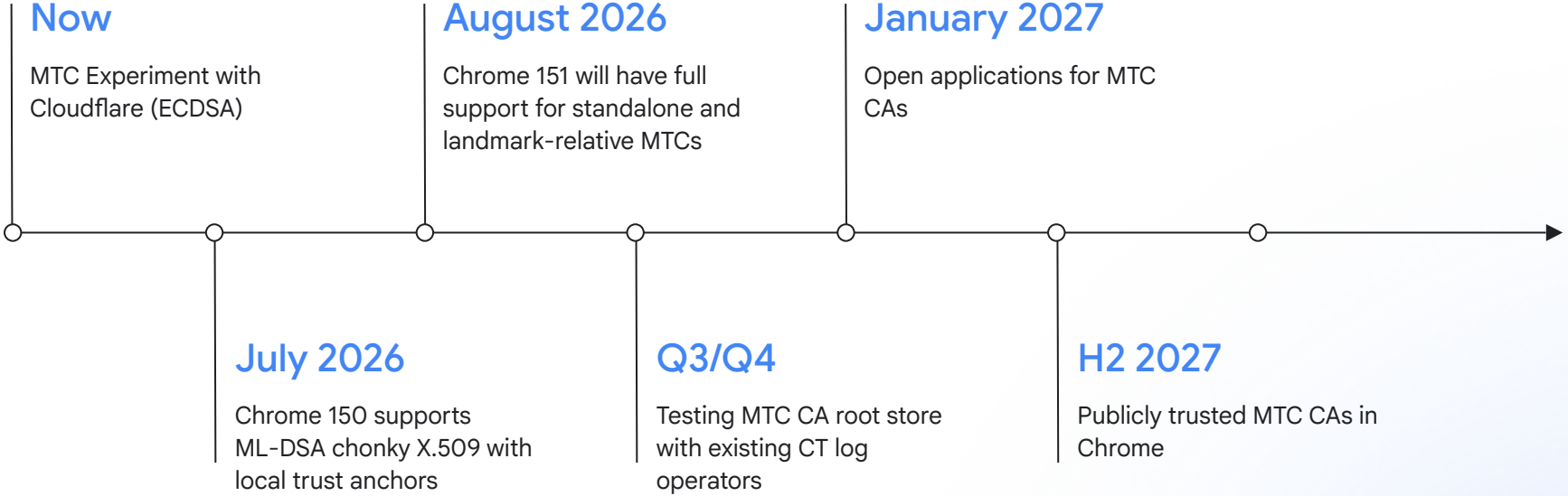
Leslie Lamport

Paxos Made Simple

MTCs, when used by servers and clients, are quite simple.

David Adrian
Google

MTC Timeline



Links

- MTC Draft RFC
- Chrome + Cloudflare MTC Experiment
- PQC Size Problem
- Chrome Blog Post with MTC CA Timeline
- MTC CA Testing Root Store

David Adrian
dadrian@

chrome

How PQC will work in PKI

What are you gonna do, send me 35kb of certs?