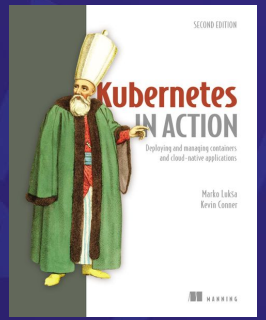

HONK
OpenSSF Community Day
NORTH AMERICA 2026

OpenSSF Community Day
NORTH AMERICA 2026

Quantum Proofing Sigstore: A Tale of Three Approaches

Kevin Conner, Red Hat



Agenda

Why Post-Quantum?

The threat landscape and why supply chain security must adapt now

Log Protection: Transparency Log Hybrid

Post-Quantum Cryptography on Signed Entry Timestamps (SETs) and checkpoints

Cert Protection A: Hybrid X.509

Dual signatures in every certificate, self-contained, backward compatible

Cert Protection B: Merkle Tree Certificate

Batch PQC with one signature per batch, most efficient at scale

Comparison & Lessons

How the layers compose and trade-offs across approaches

Why PQC in Sigstore?

Sigstore is the open standard for signing and verifying software artifacts.
Used by npm, PyPI, Kubernetes, and more.

The Threat

- Identity impersonation
- Software signatures must remain verifiable for years or decades

- NIST PQC standards finalized (FIPS 203, 204, 205)
- Migration is inevitable, but how?

Sigstore Components

- Fulcio: Certificate Authority
- Rekor: Transparency Log
- Cosign: Container signing
- Sigstore-go: Verification library
- TUF: Trust root distribution
- Each component needs PQC strategy

OpenSSF Community Day
NORTH AMERICA 2026

Quantum frontiers may be closer than they appear

Mar 25, 2026
2 min read

We're setting a timeline for post-quantum cryptography migration to 2029.

Heather Adkins
VP, Security Engineering

S

Sophie Schmieg
Senior Staff
Cryptography Engineer

Sh

https://blog.google/innovation-and-ai/technology/safety-security/cryptography-migration-timeline/

The PQC Size Problem

Post-quantum signatures are 45-65x larger than classical signatures

Algorithm	Signature Size	Public Key	Security Level
ECDSA P-256	70-72 bytes	65 bytes	128-bit classical
ML-DSA-65	3,309 bytes	1,952 bytes	NIST Level 3 (PQ)
ML-DSA-87	4,627 bytes	2,592 bytes	NIST Level 5 (PQ)

Impact at Scale

2,500 signatures x 72B (ECDSA) = 180 KB vs 2,500 signatures x 3.3KB (ML-DSA-65) = 8.25 MB vs 2,500 signatures x 4.6KB (ML-DSA-87) = 11.5 MB

Two Orthogonal Concerns

PQC protection for the log and for the certificate are independent and compose together

Log Protection

Transparency Log Hybrid Mode

- Dual-sign SETs and checkpoints
- Rekor signs with ECDSA + ML-DSA
- Works with ANY certificate approach

Certificate Protection

A: Hybrid X.509

- Dual sigs per cert via ITU-T extensions
- Self-contained, backward compatible

B: Merkle Tree Certificates

- Batched PQC, one classical and one PQC sig per batch
- Most efficient, new cert format

Full PQC stack = Log Protection (Tlog Hybrid) + Certificate Protection (Hybrid X.509 and/or MTC)

OpenSSF Community Day
NORTH AMERICA 2026

Log Protection: Transparency Log Hybrid Mode

PQC on SETs and checkpoints

Tlog Hybrid: What It Provides

Orthogonal to certificate protection. Add PQC signatures to the transparency log layer.

What IS Protected

- Signed Entry Timestamps (SETs) - multiple signers
  (e.g. ECDSA + ML-DSA-87)
- Checkpoints - multiple signers
  (e.g. ECDSA + ML-DSA-87)
- Detection of log tampering even by a quantum adversary

What Is NOT Protected

- Certificate signatures remain classical ECDSA only
- Fulcio CA signing key is still vulnerable to quantum attack
- Artifact signatures remain classical
- A quantum adversary can forge certificates (but log detects it)

Key Insight

Transparency Log Hybrid is the log protection layer. It can be combined with the certificate approaches (Hybrid X.509 and/or MTC).

Tlog Hybrid: Signing Sequence

1. Client signs artifact with classical key (ECDSA)
2. Client send request to rekor
3. Rekor creates log entry, signs SET with primary key
4. Rekor signs same SET with additional keys (e.g. PQC)
5. Checkpoint signed with all keys
6. Bundle contains multiple SETs (with logIDs) + multi-signed checkpoint

SETs: signed with ALL configured signers | signatures[0] = primary | signatures[1:] = additional, each paired with logID
Checkpoints: signed with ALL signers | Each signer adds its signature to the checkpoint

Tlog Hybrid: Verification Sequence

1. **Verify Primary SET**
   Look up verifier by logID in trusted root, call signature.LoadVerifier(), verify SET over canonicalized payload
2. **Verify Additional SETs**
   For each additional SET: extract logID, look up verifier directly in trusted root, verify SET signature
3. **Verify Inclusion Proof + Checkpoint**
   Merkle inclusion proof + checkpoint verification: similar to SET above
4. **Verify Certificate Chain**
   Standard X.509 chain verification, classical ECDSA only (no PQC here)
5. **Verify Artifact Signature**
   Standard signature verification against public key from certificate (no PQC here)

Steps 1-3 are PQC-protected (SETs + checkpoint) | Steps 4-5 remain classical | Each additional SET carries its logID for direct verifier lookup

Tlog Hybrid: Problems & Limitations

Security Gaps

- Certificate forgery is still possible by a quantum adversary
- Only detects compromise, does not prevent it
- Requires online log access for the PQC protection to be meaningful
- Artifact signatures not protected

Practical Concerns

- Additional SET adds ~4.6 KB per log entry (ML-DSA-87 signature)
- Checkpoint grows with each additional signer signature (~4.6 KB)
- Trusted root must include entries for each signing key (by log ID)
- Clients must update to verify additional SETs and checkpoints

Summary

Tlog Hybrid is the least disruptive change: zero changes to certificates, Fulcio, or the signing API.
It provides a PQC-signed audit trail, not PQC-signed artifacts. Think of it as "PQC monitoring" rather than "PQC signing."

Bundle size comparison: ~6.5 KB vs ~16.9 KB, rekor size comparison: ~1.7 KB vs ~6.4 KB

Demo Time

Transparency Log Hybrid Mode

OpenSSF Community Day
NORTH AMERICA 2026

Cert Protection A: Hybrid X.509 Certificates

Dual classical + PQC signatures in every certificate

Hybrid Certs: What It Provides

Based on ITU-T X.509 (2019) Clause 9.8. Embed PQC directly into each certificate.

What IS Protected

- Certificate has PQC signature from the CA (ML-DSA over preTBS)
- Self-contained: no network needed for PQ verification
- Backward compatible: legacy clients ignore unknown extensions
- Both classical + PQ must pass

Combined with Tlog Hybrid

- Log layer protected by tlog hybrid (dual-signed SETs + checkpoints)
- Certificate layer protected by hybrid extensions (ML-DSA sig)
- Together: full PQC from CA through log to verifier
- Artifact sig depends on signer key

Three ITU-T Extensions in Every Certificate

SubjectAltPublicKeyInfo (OID 2.5.29.72) = CA's PQ public key | AltSignatureAlgorithm (OID 2.5.29.73) = ML-DSA | AltSignatureValue (OID 2.5.29.74) = PQ signature (~4.6 KB)

Hybrid Certs: Signing Sequence

The signing order is critical. PQ signature must be computed before the classical signature.

1. Add SubjectAltPublicKeyInfo extension (CA's PQ key)
2. Add AltSignatureAlgorithm extension (ML-DSA OID)
3. Compute preTBSCertificate (TBS without AltSignatureValue)
4. Sign preTBS with PQ key => ML-DSA signature (~4.6 KB)
5. Add AltSignatureValue extension (PQ signature)
6. Sign complete TBS with classical key => ECDSA sig

PQ signature covers preTBS (without AltSignatureValue) | Classical signature covers complete TBS (all three extensions)
Verification reverses this: strip AltSignatureValue, verify PQ sig on preTBS, then verify classical sig on full TBS

Hybrid Certs: Verification Sequence

Legacy Client (unchanged)

1. Parse X.509 certificate normally
2. Ignore unknown non-critical extensions
3. Verify classical ECDSA signature
4. Validate certificate chain

=> Works exactly as before
=> Zero changes required

PQ-Aware Client

1. ParseHybridExtensions(cert)
2. Extract CA PQ key from trusted root
3. buildVerificationPreTBS(cert)
   (strip AltSignatureValue extension)
4. signature.LoadDefaultVerifier(altKey)
5. Verify ML-DSA sig on preTBS
6. Both classical + PQ must pass

Implementation Challenge

Go's crypto/x509 rejects ML-DSA keys and PQ algorithms. Solution: custom ASN.1 parsing (tbsCertificateForHybrid struct), cryptographic algorithm extensions, DER re-encoding. CA alt public key stored in FulcioCertificateAuthority.AltPublicKey in trusted root.

Hybrid Certs: Problems & Limitations

Storage

- Classical cert: ~750 B
- Hybrid cert: ~7.9 KB
- ML-DSA-87 sig: +4.6 KB
- ML-DSA-87 key: +2.6 KB
- ~10x larger certs
- Every Rekor entry grows similarly

Complexity

- Custom ASN.1 parsing
- preTBS reconstruction
- Go x509 workarounds
- Fulcio must manage two signing keys
- Trusted root needs CA alt public key

Scale Problem

2,500 classical certs = ~1.9 MB | 2,500 hybrid certs = ~20 MB | ~10x storage increase
Every certificate carries the FULL PQC overhead (sig + key), regardless of batch size. This may not scale well for high-volume environments.

Bundle size comparison: ~6.5 KB vs ~43.6 KB, rekor size comparison: ~1.7 KB vs ~19.3 KB

Demo Time

Hybrid X.509 Certificates

OpenSSF Community Day
NORTH AMERICA 2026

Cert Protection B: Merkle Tree Certificates

Batched PQC with minimal per-certificate overhead

MTC: What It Provides

Batch certificates into a Merkle tree. ONE PQC signature covers the entire batch.

What IS Protected

- Certificate issuance: Fulcio signs subtreeRoot with multiple keys
- Transparency: log entry is PQ-signed via SET and checkpoint
- Certificate binding: Merkle proof ties each cert to the signed root
- Complete PQC pipeline from CA through log to verifier

Efficiency Improvement

- ONE signing event per batch (not per certificate)
- TWO Rekor log entries per batch:
  rhmtcmonitoring + rhmtccommitment
- Per-cert overhead: ~100 bytes (Merkle inclusion proof hashes)
- 295 certs share 1 signature (rekor-tiles limit): ~2 bytes/cert

Bundle size comparison: ~6.5 KB vs ~15.1 KB, rekor size comparison: ~1.7 KB (per cert) vs ~490 B/cert amortized (295-cert batch)

rhmtcmonitoring stores binary TBS batch; rhmtccommitment stores subtree signatures and monitoring binding

MTC: Signing Sequence

1 — Authenticate
OIDC login & key generation

2 — Request Cert
Fulcio queues TBSCertLogEntry

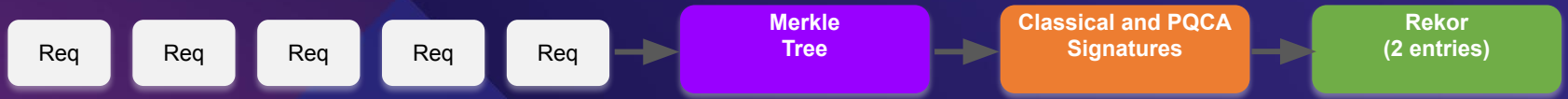
3 — Log Monitoring
Upload rhmtcmonitoring to Rekor

4 — Batch + Sign
Build Merkle tree + sign

5 — Log Commitment
Upload rhmtccommitment to Rekor

6 — Return
Bundle with inclusion proof

Batch Accumulation:



Triggers: time interval OR batch size threshold

Each signer receives: TBSCertLogEntry + Merkle inclusion proof (~100B) + subtree signature(s) + Rekor checkpoint + log inclusion proof + Rekor SETs + key IDs

MTC: Verification Sequence

1. **Verify Public Key Hash**

   SHA256(publicKey) == tbsCertLogEntry.pubKeyHash, binds signer to certificate

2. **Certificate Inclusion Proof**

   MerkleVerify(leafIndex, leafHash, hashes) == subtreeRoot, cert is in the batch

3. **Subtree Signatures (PQC)**

   Verify signed message (subtreeRoot || logID || start/end || monitoring_log_id || monitoring_log_index || monitoring_leaf_hash)

4. **rhmtccommitment + SET + Checkpoint**

   Reconstruct commitment from bundle; verify generated leafHash matches, Merkle inclusion, SETs and checkpoint signatures

5. **Artifact Signature**

   Verify(mtcPublicKey, messageDigest, signature), the actual artifact verification

Every step in the chain is PQ-protected | All verification is offline once bundle is obtained

MTC: Problems & Limitations

Breaking Changes

- New certificate format (not X.509)
- Not backward compatible with existing clients
- New protobuf types and bundle format
- All clients must update
- Complex verification (5-step chain)

Latency

- Must wait for batch to fill (time or size)
- Signing is not instant:
  queue + batch + sign
- Latency varies with traffic volume
- Low-traffic periods mean longer waits

Infrastructure

- Fulcio needs batching logic and additional keys
- Rekor and rekor-tiles includes new types
- Trusted root needs MTC signing authorities
- New bundle media type and protobuf messages

The Trade-off

MTC provides the strongest PQC protection with the best efficiency, but requires the most significant changes to the entire Sigstore stack.
This is a "big bang" migration, not an incremental rollout.

Demo Time

Merkle Tree Certificates

Comparison

Property	Tlog Hybrid (log)	Hybrid X.509 (cert)	MTC (cert)
Concern	Log protection	Cert protection	Cert protection
Composable?	Used by both =>	Yes + Tlog Hybrid	Yes + Tlog Hybrid
Fulcio changes	None	Dual-key signing	Batching + dual signing
Rekor changes	Multi-signer	None	Two new types: rhmtcmonitoring (binary TBS batch) + rhmtccommitment (subtree sigs)
Client changes	Minimal	Moderate	Significant
Backward compat	Yes	Yes (legacy ok)	No
Cert format	X.509 unchanged	X.509 + extensions	New (MTC)
Latency impact	None	None	Batching delay

Size Comparison

	Standard Sigstore	Tlog Hybrid	Hybrid X.509	MTC
Certificate	~750 B	~750 B	~7.9 KB	~550 B
Rekor entry	~1.7 KB	~6.4 KB	~19.3 KB	~490 B/cert amortized (295-cert batch, 2 Rekor entries)
Bundle total	~6.5 KB	~16.9 KB	~43.6 KB	~15.1 KB (one per certificate)

Security: How the Layers Compose

Log protection and certificate protection are independent.

	Tlog Hybrid alone	Hybrid X.509 + Tlog Hybrid	MTC + Tlog Hybrid
Artifact Signature	Classical Only	Classical Only	Classical Only
Certificate (CA -> Signer)	Classical Only	PQ Protected	PQ Protected
Transparency Log (SET)	PQ Protected	PQ Protected	PQ Protected
Log Checkpoint	PQ Protected	PQ Protected	PQ Protected

Lessons Learned

Assumptions in existing code created unexpected challenges

Go's crypto/x509 is a gatekeeper

Standard library rejects unknown key types and signature algorithms. Required extensions for key marshalling, supported algorithms and certificate parsing (tbsCertificateForHybrid).

Algorithm agility is essential

Using signature.LoadVerifier() instead of hardcoded ECDSA made the tlog hybrid change a 10-line diff. Pairing each additional SET with its logID enables direct verifier lookup (same pattern as the primary SET) across tlog/entry.go, verify/rhmtc_tlog.go and verify/rhmtc.go.

And more generally

Size assumptions are everywhere

Databases, protobuf messages, HTTP requests all assume small signatures. PQC breaks these assumptions across the entire stack.

Next Steps & Community Discussion

Open Questions

- Which approach(es) to adopt?
- Can they coexist in production?
- Migration path for existing users?
- Performance benchmarks at scale?
- Interoperability with other ecosystems?
- Algorithm agility for future changes?
- Timeline for PQC migration?

Get Involved

- All demos are open source: github.com/signed-sealed-delivered
- Sigstore community Slack (sigstore.slack.com)
- GitHub: github.com/sigstore
- Post-Quantum Cryptography for Sigstore (Public)
- Feedback welcome on all approaches

OpenSSF Community Day
NORTH AMERICA 2026

Thank You

Questions?

Kevin Conner: kconner@redhat.com

Sigstore GitHub Organization
POC & Demo GitHub Organization

github.com/sigstore
github.com/signed-sealed-delivered

Manning discount, expires June 5th
conner45

First 5 questions get **Kubernetes in Action** ebook