THE LINUX FOUNDATION

OPEN SOURCE SUMMIT INDIA

Quantum-Safe TLS in Practice

with Open Quantum Safe & OpenSSL 3

Divyanshu Agrawal · Shubham Bhardwaj · Anitha Natarajan
Tekton Friends @Red Hat

AGENDA

One story Of Quantum Safe TLS, two demos

DEMO 1 · DEPLOY IT

The present

Deploy PQC today with native OpenSSL 3.5 - no provider. Certs, a live handshake.

DEMO 2 · EVALUATE NEXT

The frontier

How the community decides what you deploy in 5 years? oqs-provider + a candidate signature, SNOVA.

Note on our abstract: we planned this around oqs-provider. Since, OpenSSL 3.5 shipped the standardized algorithms natively we have framed the session in 2 stages - The present with native OpenSSL and the frontier with liboqs

WHY NOW

This is not a future problem

~57%

of browser-initiated TLS connections already send a post-quantum key share (early 2026)

- **Apr 2024** — Chrome enables PQC key exchange by default
- **Nov 2024** — Firefox follows
- **Oct 2025** — Apple ecosystem joins
- **Jan 2026** — Akamai: PQC default for all origin connections

Go 1.24+ enables X25519MLKEM768 by default, much of the Kubernetes ecosystem already negotiates PQC.
Reference: https://kubernetes.io/blog/2025/07/18/pqc-in-k8s/

The threat, framed honestly

No Q-day yet

No quantum computer today breaks X25519 or RSA. Credible timelines: “over a decade” to “maybe never.”

Shor's algorithm

Breaks asymmetric crypto: key exchange and signatures. This is the real target.

Grover's algorithm

Only weakens symmetric crypto. AES-256 remains sufficient, no panic needed here.

HARVEST NOW, DECRYPT LATER — the reason to act today

Traffic recorded today is decryptable on Q-day. If your data must stay confidential for 10+ years - health, finance, government - the recording risk applies to you right now.

ANATOMY

The three crypto jobs inside TLS 1.3

1. Key exchange

Today: X25519 (ECDHE): Creates the session secret

`ClientHello · key_share`

2. Authentication

Today: ECDSA / RSA certificates: proves who you talk to

`Certificate + CertVerify`

3. Bulk encryption

Today: AES-GCM (symmetric): protects the data

`Application data records`

Band 1 produces the session secret · band 3 consumes it · band 2 proves identity. “PQC in TLS” is NOT one switch.

SAME MAP, OVERLAID

What quantum breaks — and the remedy

Key exchange	**Shor: broken** HNDL — retroactive	→	**X25519MLKEM768** hybrid — deploy now
Signatures	**Shor: broken** not retroactive	→	**ML-DSA / SLH-DSA** slower clock, PKI-wide
Symmetric	**Grover: weakened** no break known	→	**AES-256** already sufficient

Urgent (HNDL)

Breaks at Q-day only

Weakened, not broken

What NIST gave us

FIPS 203

ML-KEM

Key encapsulation (lattice) — the key-exchange fix

FIPS 204

ML-DSA

Digital signatures (lattice) — the certificate fix

FIPS 205

SLH-DSA

Hash-based signatures — conservative alternative

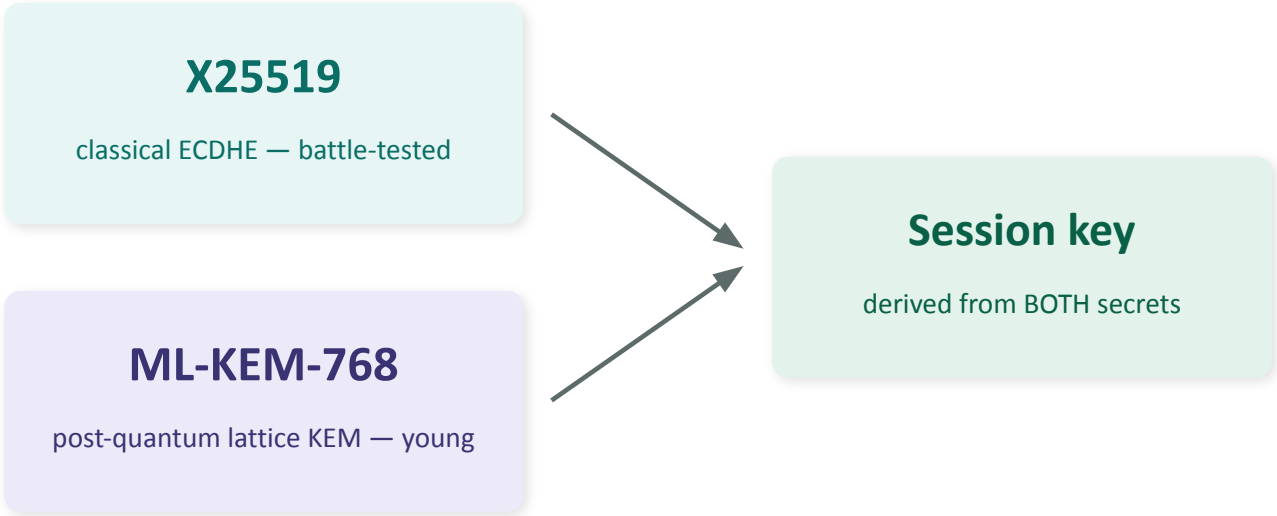
2025: HQC selected too

A code-based KEM, deliberate diversity hedge against a lattice break. FN-DSA (Falcon) standard still pending.

The honest gap

Wire format is standardized: RFC 9881 (ML-DSA in X.509, Oct 2025). The gap is governance: CA/Browser Forum policy + root programs. A local CA is the only path today.

Why hybrid, not pure post-quantum



Attacker must break both.

The hedge costs ~1 KB in the ClientHello — cheap insurance on a young algorithm.

Chrome, Firefox, Safari, Cloudflare and Akamai all chose this.

Nuance: hybrid isn't universal doctrine — NSA's CNSA 2.0 mandates PURE ML-KEM-1024 / ML-DSA-87 for national-security systems. Jurisdiction matters.

The present: deploy PQC today

Everything runs in one Fedora 43 container — system OpenSSL is native 3.5, exactly what every command needs. No provider, no plugin, no special build.

```
$ podman run --rm -it -w /work -v ~/pqc-lab:/work:Z fedora:43 bash
# inside the container:
$ dnf -y install openssl   # the openssl 3.5 CLI on Fedora 43
```

One requirement

OpenSSL 3.5+ is the hard line. On 3.0–3.4 every command below fails and that failure is itself the readiness lesson.

All three FIPS algorithms ship native

```
$ openssl list -providers
  default    (and base)              ← no oqs-provider in sight
$ openssl list -signature-algorithms | grep -i ml-dsa
  ML-DSA-65 @ default                 ← the tag is the inventory truth
```

FIPS 203

ML-KEM @ default

key exchange — the workhorse

FIPS 204

ML-DSA @ default

signatures — the TLS default

FIPS 205

SLH-DSA @ default

hash-based — conservative, niche

Build a PQC cert chain and feel the cost

```
# Root CA (ML-DSA-65), self-signed
$ openssl req -x509 -newkey mldsa65 \
    -keyout ca.key -out ca.crt ...
# Leaf + sign with the CA, then:
$ cat leaf.crt ca.crt > chain.crt

ec-ca.crt      ~450 bytes
ca.crt       ~4,800 bytes
chain.crt    ~9,500 bytes  ← the real cost
```

What to point at

- Zero special flags — mldsa65 is native. You did NOT load a provider.
- The ML-DSA chain is kilobytes; the ECDSA control is hundreds of bytes.
- A full PQC chain can approach or pass the ~14 KB initial congestion window — that's where size becomes an extra round trip.
- **The cost is BYTES, not CPU.**

Verify, don't trust the filename: openssl x509 -in leaf.crt -text shows ML-DSA-65 on both the public-key and signature lines. “Usable, not just named” is what migration sign-off actually requires.

SLH-DSA - Sneak Peek

```
$ openssl genpkey -algorithm \
    SLH-DSA-SHA2-128s -out slh.key
# note -rawin: SLH-DSA is one-shot,
# NOT hash-then-sign
$ openssl pkeyutl -sign -rawin ...
Signature Verified Successfully
$ wc -c message.sig  → ~7,800
```

Signature size, side by side

ECDSA	~64 B
ML-DSA-65	~3.3 KB
SLH-DSA-128s	~7.8 KB

The deliberate anticlimax

Fully native, fully FIPS-205 — but the signature is huge and signing is visibly slow. That's exactly why SLH-DSA is the conservative, niche choice (firmware, code-signing) and ML-DSA is the TLS workhorse. Don't build an SLH-DSA TLS chain on stage; keep the handshake on ML-DSA.

• DEMO 1

A real handshake — and a silent downgrade

```
# server (Terminal A):
$ openssl s_server -cert chain.crt \
    -key leaf.key -www -tls1_3 -accept 4433
# client (Terminal B): default groups
$ openssl s_client -connect localhost:4433 ...
Negotiated group: X25519MLKEM768
Peer signature type: mldsa65

# now handicap the client:  -groups x25519
Peer Temp Key: X25519, 253 bits
Peer signature type: mldsa65  ← still PQC
```

- You didn't ASK for PQC key exchange - handshake chose the hybrid by default.
- One flag (-groups x25519) silently drops you to classical — no error.
- But the signature stays mldsa65: KEX and auth are independent. A downgrade can hit one, not the other.

OpenSSL providers: the mental model

A provider = a pluggable crypto backend. OpenSSL 3 separates the algorithms from the protocol, so you can add new ones without touching the TLS stack. It ships three:

`default`

normal algorithms — incl. native ML-KEM / ML-DSA on 3.5+

`fips`

the FIPS-validated subset

`legacy`

old algorithms (MD2, RC4, …)

```
$ openssl list -signature-algorithms -provider oqsprovider -provider default | grep snova
  snova2454 @ oqsprovider          ← a candidate, supplied by the plugin
```

The "@ provider" tag IS the concept: it tells you which backend answers for each algorithm. Crypto inventory is exactly that question.

Open Quantum Safe in 2026

liboqs 0.14 + oqs-provider 0.10
A matched pair. Part of the Post-Quantum Cryptography Alliance, under the Linux Foundation — same family as this event.

Self-described purpose:
“prototyping and evaluating” quantum-resistant crypto.
OQS itself says: rely on NIST standards for deployment.

The field moves fast — even inside liboqs:

- SLH-DSA (FIPS 205) graduated to a standard — SPHINCS+, its pre-standard ancestor, retired in liboqs 0.16; Dilithium already removed
- SNOVA needs oqs-provider ≥ 0.10.0 — 0.9.0 ships MAYO/CROSS/UOV but not SNOVA. Names and enabled sets drift between commits.
- **Practical takeaway: PIN release tags, build the container once. Never track main in a workshop or in CI.**

The deferral lesson

Load oqs-provider on OpenSSL 3.5+ and grep for ML-KEM under it — it's absent. oqs-provider deliberately defers ML-KEM / ML-DSA to native. The “missing” algorithm engineers panic about is correct behavior, not a bug.

The 2026 division of labor

Job	Right tool
Deploy standardized PQC in TLS (ML-KEM, ML-DSA)	**Native OpenSSL 3.5+**
Add PQC to a stack stuck on OpenSSL 3.0–3.2	**oqs-provider**
Evaluate non-standardized / candidate algorithms (SNOVA, BIKE, FrodoKEM, FN-DSA…)	**liboqs + oqs-provider**

▶ DEMO 2

SNOVA via the unmodified binary

```
# SNOVA is absent natively:
$ openssl list -signature-algorithms | grep snova
  SNOVA: not present natively

# load the provider — it appears:
$ openssl list ... -provider oqsprovider | grep snova
  snova2454 @ oqsprovider

$ openssl genpkey -algorithm snova2454 ...
$ openssl pkeyutl -sign -rawin ...
Signature Verified Successfully
```

The teaching beat

The SAME openssl binary performs a brand-new signature scheme purely because a provider was loaded. No recompile. No patched TLS stack. This is how the field test-drives candidates before standardization.

Most build-fragile module — default to the container recording unless the dry-run was clean.

DEMO 2 · ARCHITECT PAYOFF

ML-DSA vs SNOVA: the trade, in numbers

Same binary, side by side — a standardized signature and a candidate. The only honest way to weigh a candidate is to measure it.

	Signature size	Status
ML-DSA-65	~3.3 KB (balanced)	**Standardized · native**
SNOVA-24-5-4	**small (few hundred B)**	**Candidate · provider**

The decision, made concrete: SNOVA's tiny signature looks attractive but is still a candidate and different variants of SNOVA has bigger key size which will impact the cert size. THIS is why we evaluate behind a provider before betting a fleet on it.

Key Takeaways

SNOVA advanced to NIST Round 3 — but some of its Round-2 parameter sets were cryptanalytically attacked. NIST kept it because its unbroken parameter sets still show promise. That is exactly why you evaluate candidates behind an experimental provider and never in production.

The take-home: a repeatable evaluation recipe

1

Availability

build it; confirm the @ oqsprovider tag

2

Functional

keygen + sign/verify (or a real handshake)

3

Quantitative

signature AND public-key sizes — wire cost

4

Fit + maturity

use case + is the parameter set unbroken?

The standards radar

Decision area	Read this	Status
Key exchange (deploy now)	FIPS 203 · draft-ietf-tls-ecdhe-mlkem · draft-ietf-tls-hybrid-design	**Codepoints live, RFC pending**
Signatures in TLS	FIPS 204 · draft-ietf-tls-mldsa	**WG draft**
Certificates / PKI	RFC 9881 (ML-DSA X.509) · RFC 9909 (SLH-DSA X.509) · RFC 9882 / 9814 (CMS) · LAMPS composite drafts	**RFCs done; CA/B pending**
Prerequisite	RFC 8446 (TLS 1.3) — tls12-frozen: PQC will never come to TLS 1.2	**Migrate to 1.3 first**

The compliance clock — adoption timelines

NIST IR 8547
RSA/ECC: deprecated 2030, disallowed 2035

UK NCSC
plan 2028 · high-priority 2031 · complete 2035

EU roadmap
high-risk by 2030 · full by 2035

CNSA 2.0
NSS acquisitions 2027 · pure PQC

CHECKLIST

Are you PQC ready?

- ✓ Inventory: where do you terminate TLS? What OpenSSL is each piece built against?
- ✓ Still on TLS 1.2 anywhere? PQC will never come to it (tls12-frozen) — migrate to 1.3 first
- ✓ Key exchange: enable X25519MLKEM768 — often just an upgrade (Go 1.24+, OpenSSL 3.5, browsers already default)
- ✓ Watch your proxies & middleboxes — the silent-downgrade point
- ⏳ Signatures / PKI: RFC 9881 format is ready — track CA/Browser Forum; practice with a local CA today
- Evaluate candidates with liboqs — and pin your release tags

Data with 10+ year confidentiality? Harvest-now-decrypt-later applies to you NOW.

Thank you!

Questions?

```
https://github.com/random-expermients/oss-workshop/blob/iteration-2/revised-modules/quantum-safe-tls-hands-on-lab.md  -> everything we showed, ready to run
```

Divyanshu Agrawal · Shubham Bhardwaj · Anitha Natarajan — Tekton Friends @ Red Hat