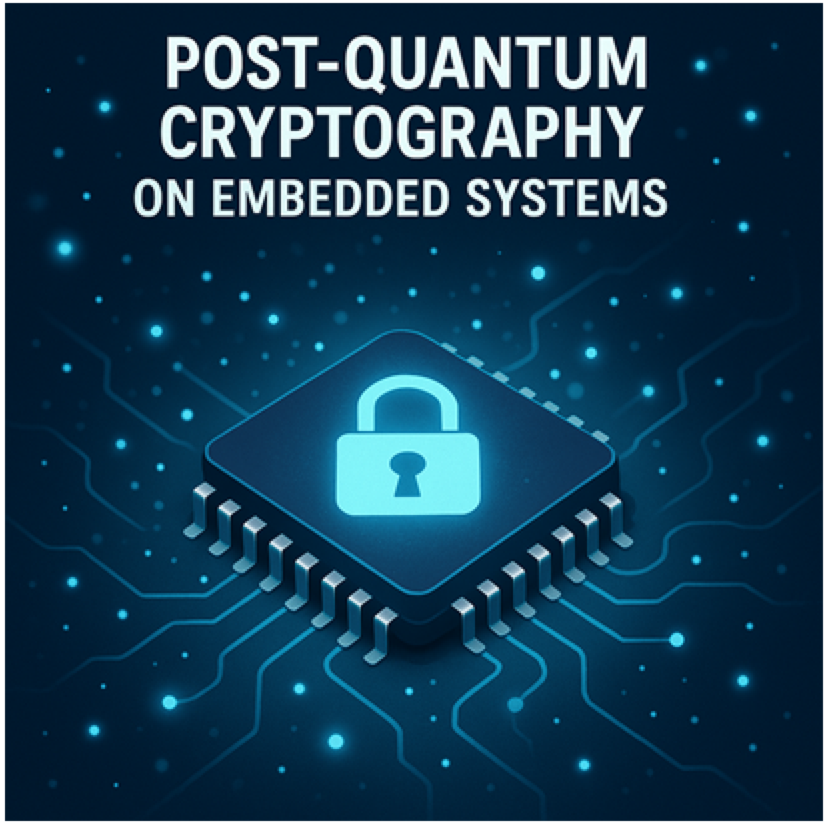
Intro to PQC for Embedded devs



David Cermak, Embedded developer

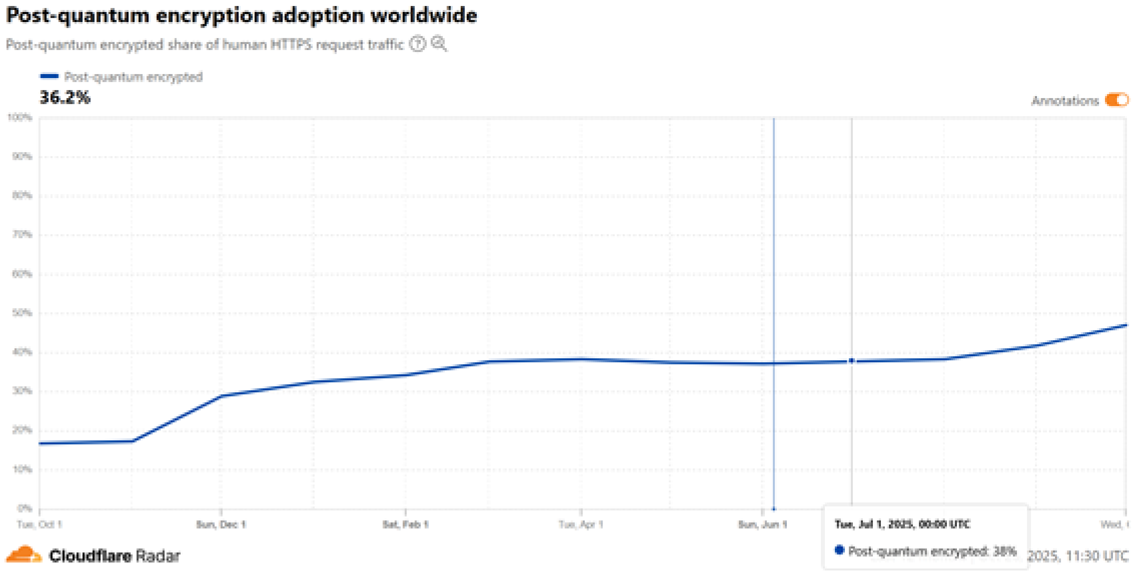
Why Post-Quantum, Why Now

- **Harvest-now, decrypt-later** risk for long-lived data and devices
- **Trust Now, Forge later** risk for OTA
- Embedded/IoT lifetimes span decades; roots of trust must endure
- Standards have landed: **FIPS 203/204/205**; hybrid transition paths exist

TLS handshake worldwide

HTTP requests by post-quantum support time series

- From ~15% to ~50% in the last year



Cloudflare Radar: https://radar.cloudflare.com

OpenSSH ~10.2 warns about non-pq key exchange

Quantum Threat Landscape

- Shor breaks RSA/ECC
- Classical PKI becomes forgeable
- Practical response: start migration with
    - **crypto agility**
    - **hybrids**

PQC Building Blocks & Standards

- KEM: **ML-KEM (Kyber) 512/768/1024**
- Signatures: **ML-DSA (Dilithium), SLH-DSA (SPHINCS+), Falcon**
- Hybrid crypto: combine classical + PQ for transition safety

Embedded Constraints

- Tighter RAM/flash, CPU/energy budgets; larger PQ artifacts
- Side-channel hardening, careful memory planning, zeroization
- Prefer constant-time, vetted implementations; enable crypto agility

Sign Today, Forge Tomorrow (STFT)

Trust Now, Forge Later (TNFL)

- Forged signatures undermine secure boot, OTA, device identity — compromise is immediate and often invisible
- Long lifecycles (15–30 yrs), limited patch windows for bootloaders, hard-coded crypto and roots set at manufacture amplify risk
- Mitigations: crypto-agility, update roots of trust (TPM/HSM/SE), full crypto inventory, staged PQC rollout; gateway validation when devices can’t update
- Underrated vs HNDL — prioritize integrity and safety alongside confidentiality in migration plans

PQC Secure Boot — Why and What

- First PQC use case: software/firmware signing per CNSA 2.0 (by 2025)
- Store PQ public keys in ROM/OTP; verify each boot stage's signature
- Ensure algorithm agility in bootloaders and OTA verifiers
- PQC replaces RSA/ECDSA in verification; keep symmetric AES (prefer AES-256)

PQC Signature Choices (for Boot)

- ML-DSA (Dilithium): stateless lattice; unlimited signs; larger keys; variable signing time
  - plan hybrids
- LMS/XMSS: stateful hash-based; small, fast verify; ideal for boot; requires state management
  - standalone

Hybrid Signatures & Crypto Agility

Tooling & Libraries

- wolfBoot: LMS/XMSS, ML-DSA, hybrid auth; portable; HW accel support
- ST X-CUBE-PQC: LMS/XMSS verify, ML-DSA/ML-KEM for STM32
- PQShield PQMicroLib-Core: tiny PQC for MCUs; constant-time; DPA (Differential Power Analysis) aware

Demo #1: Dedicated Secure Channel (ESP32)

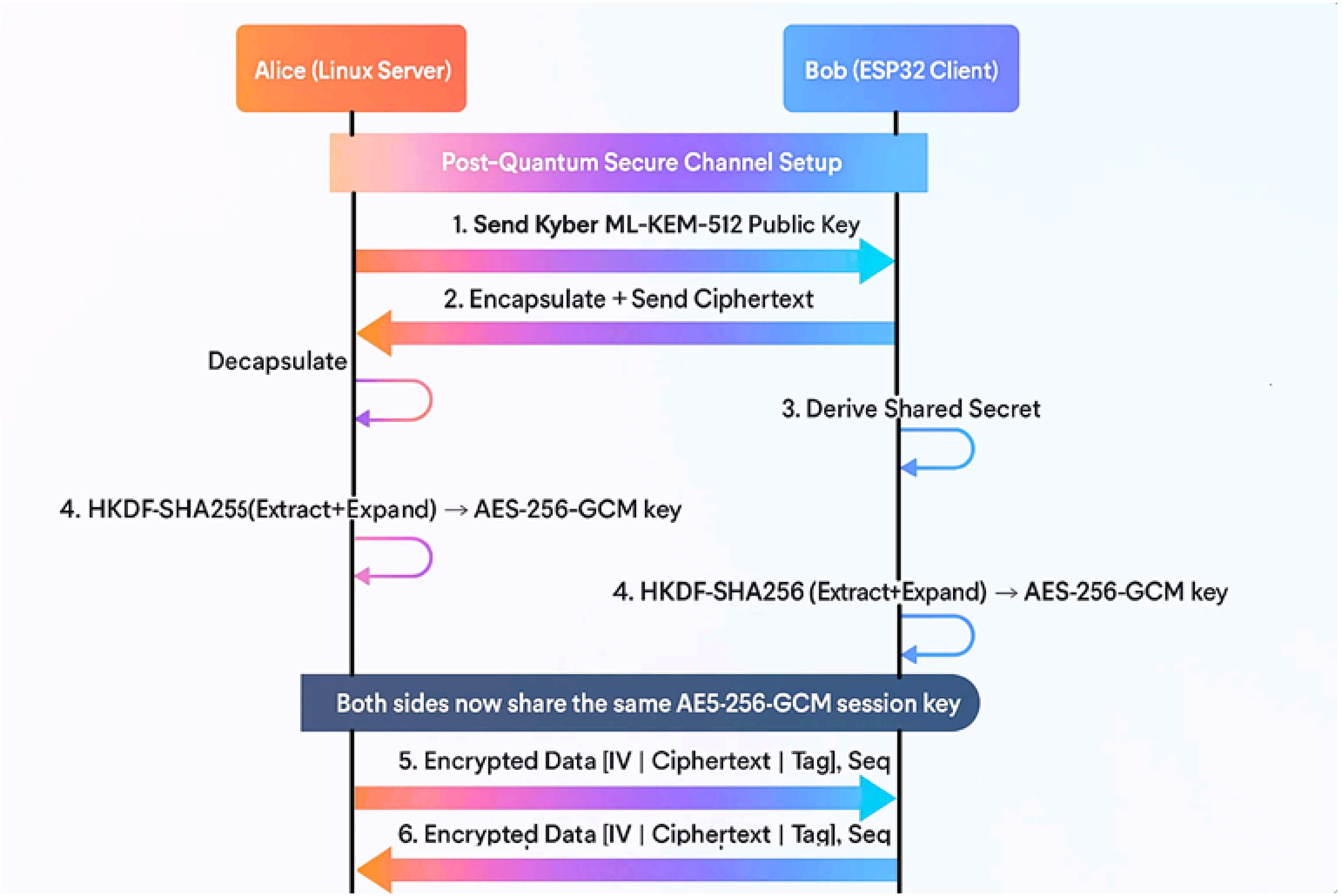
- Goal: **ML-KEM-512 key exchange → AES-GCM data channel**
- Highlight: small stack deltas; feasible on embedded targets

```
cd examples/host && mkdir -p build && cd build
cmake -DCRYPTO_BACKEND_DEFAULT=openssl .. && make
./bin/server
```

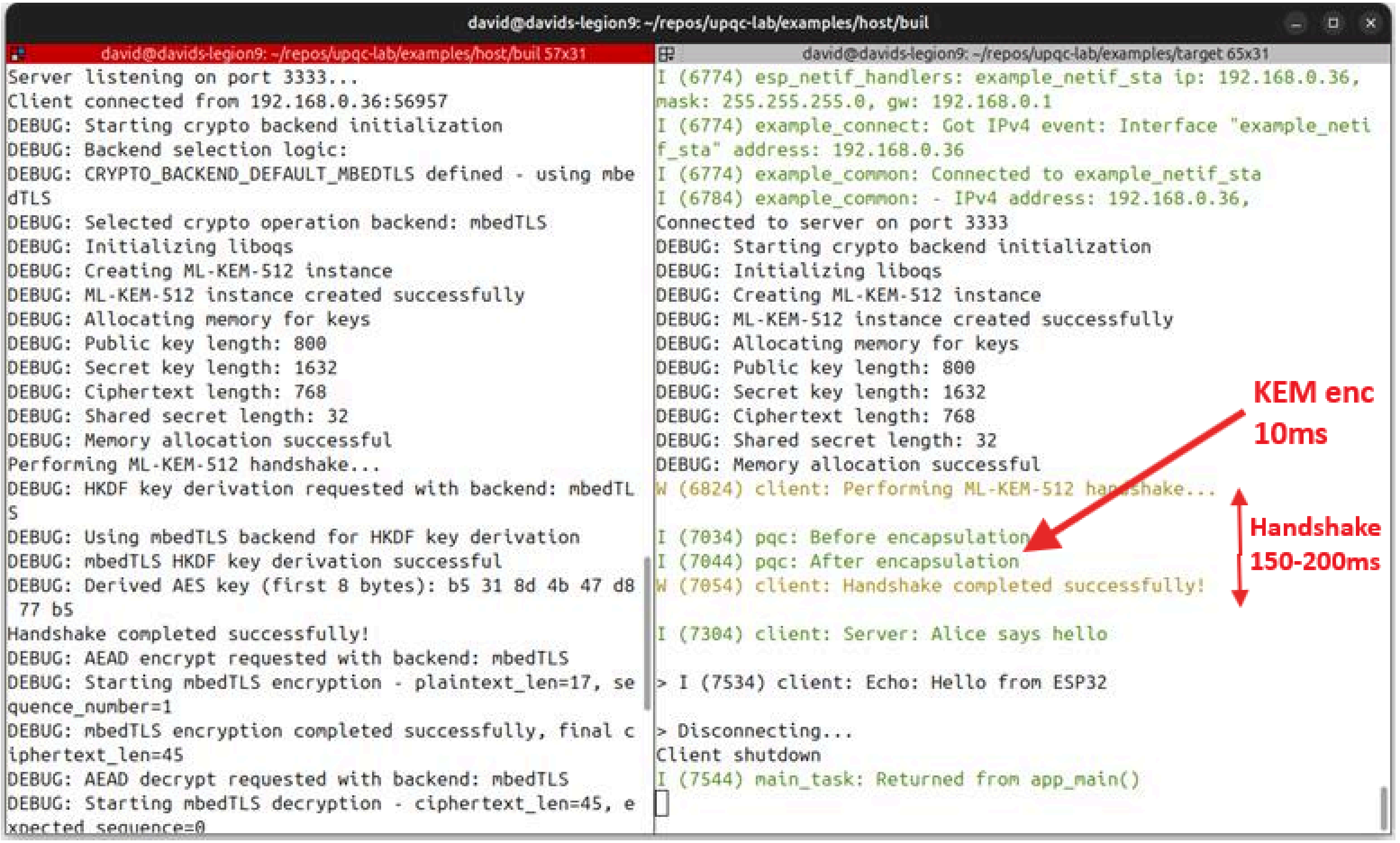
Run ESP32 target (encrypted echo over TCP):

```
cd examples/target
idf.py build flash monitor
```

Demo #1: Dedicated Secure Channel



Demo #1: Dedicated Secure Channel



Demo #2 — TLS 1.3 Hybrid X25519 + ML-KEM-768

- Client: `hybrid/target_client` (ESP-IDF Linux port or ESP32)
- Server: OpenSSL 3.5+ with group `X25519MLKEM768` (IANA 0x11EC)

Build client:

```
cd hybrid/target_client
idf.py build
```

Start local TLS server (terminal A):

```
openssl s_server -accept 8443 -tls1_3 -cert cert.pem -key key.pem \
  -www -msg -debug -groups X25519MLKEM768
```

Expect: "Handshake completed successfully" and server prints negotiated group `X25519MLKEM768`.

```
I (4418) example: Seeding the random number generator
I (4428) example: Skipping certificate bundle for localhost testing...
I (4428) example: Setting hostname for TLS session...
I (4438) example: Setting up the SSL/TLS structure...
I (4438) example: Configuring SSL/TLS for hybrid PQC testing...
I (4448) example: SSL config: authmode=VERIFY_NONE, transport=STREAM
I (4448) example: Connecting to www.cloudflare.com:443...
I (4478) example: Connected.
I (4478) example: Performing the SSL/TLS handshake...
*** UPQC_ENABLE_HYBRID_11EC defined: 1 ***
*** HYBRID GROUP DETECTED: 0x11ec ***
*** CALLING HYBRID KEY EXCHANGE FUNCTION ***
*** Generating hybrid X25519MLKEM768 key exchange ***
W (4498) MLKEM768: Generating ML-KEM-768 keypair
W (4508) MLKEM768: Generated ML-KEM-768 keypair
*** Timing: ML-KEM-768 keypair: 14879 us ***
*** Timing: X25519 keypair (PSA): 723 us ***
*** Generated hybrid key exchange: 1216 bytes ***
*** HYBRID KEY EXCHANGE RESULT: ret=0, len=1216 ***
*** HYBRID KEY SHARE WRITTEN: 1216 bytes ***
*** SIMPLE PRINTF: Starting supported groups extension processing ***
I (4678) wifi:<ba-add>idx:0 (ifx:0, ca:6e:08:18:2f:db), tid:0, ssn:12, winSize:64
*** Parsing hybrid X25519MLKEM768 key share ***
W (4678) MLKEM768: Decapsulating ML-KEM-768 shared secret
W (4698) MLKEM768: Decapsulated ML-KEM-768 shared secret
*** Timing: ML-KEM-768 decapsulate: 19461 us ***
*** Timing: X25519 ECDH derive (PSA): 114455 us ***
*** Computed hybrid shared secret: 64 bytes (stored) ***
I (5108) example: Handshake completed successfully
I (5108) example: Verifying peer X.509 certificate...
I (5108) example: Certificate verified.
I (5108) example: Cipher suite is TLS1-3-AES-256-GCM-SHA384
I (5118) example: Writing HTTP request...
I (5118) example: 84 bytes written
I (5128) example: Reading HTTP response...
HTTP/1.1 301 Moved Permanently
Date: Sat, 01 Nov 2025 17:42:34 GMT
Content-Length: 0
Connection: close
CF-RAY: 997d30835ed287a4-PRG
Location: https://www.cloudflare.com/hello/
Set-Cookie: __cf_bm=pFWqgmsHoHJbCGfeqt3MoPXZaGqwW4_jrSZ7C540Qzs-1762018954-1.0.1.1-J90107U6BLLSV9hMF6Sv_SxMEisMXIbccNAz3D8DXz8YpieMvFw82mR3VB
8jHZZ0bqV5CTnBb5FuTLVNci6T96WUpXFN9utnlbZjfRQlve8fWAHYKIKSa1w1faXnQAJC; path=/; expires=Sat, 01-Nov-25 18:12:34 GMT; domain=.www.cloudflare.c
om; HttpOnly; Secure; SameSite=None
Report-To: {"endpoints":[{"url":"https:\/\/a.nel.cloudflare.com\/report\/v4?s=sobrsZWbgsH20Bey46NvUE6q7GrMEvZVNVJ5%2FUml%2BIT%2BbqviXJuryJaNZ
rw5c0rFegZFfLgLMUBw0OkGUPDbV0FX2bTbzIUHsN%2B4C6f2clmhsa0h5VNq9s0ZVdZpONHvsiQ7pw%3D%3D"}],"group":"cf-nel","max_age":604800}
NEL: {"success_fraction":0,"report_to":"cf-nel","max_age":604800}
```

KEM keypair gen
20ms

KEM decapsulation
20ms

Inspecting the Hybrid Handshake

```
sudo tshark -i lo -Y "tls.handshake.extensions_supported_groups || tls.handshake.extensions_key_share" -O tls
```

- Supported Groups includes `0x11ec`
- ClientHello KeyShare length `1216`; ServerHello `1120`
- capture

Key Takeaways

- Start at the root of trust:
    - PQ firmware signing,
    - then transport
- Use hybrids during transition
- Design for algorithm agility

Timings of ML-KEM-768 on ESP32@160MHz

Operation	Time (ms)
Keypair	17.538
Encaps	19.926
Decaps	22.905

Metrics of ML-KEM-768 on ESP32

Metric	Value
Stack Used	15,188 bytes
Heap Usage	224 bytes

Component	Total Size	Flash Code (.text)	Flash Data (.rodata)
liboqs_mlkem.a	**10,756 bytes**	**10,308 bytes**	**448 bytes**

Links

- uPQC-lab
  - dedicated-channel
  - hybrid-groups
- STFT/TNFL risk
- NIST
  - CNSA 2.0 timeline
- wolfSSL, wolfBoot
- ST X-CUBE-PQC
- PQShield, Secure boot considerations