Navigating the Post-Quantum Transition:
Integrating FAEST & Classic McEliece into libOQS

Practical Engineering Hurdles, Build System Plumbing, and OpenSSL Provider Integration

Norman Ashley • Software Engineering Technical Leader

SESSION OVERVIEW • 30 MINUTES

Session Agenda & Technical Roadmap

- 1. The Post-Quantum Reality & The Mission of libOQS
- 2. Growing libOQS's Algorithm Set & Tooling Automation
- 3. Algorithm Spotlight: FAEST vs. Classic McEliece
- 4. The 4 Integration Iterations (Architecture, Build, Provider, CI/CD)
- 5. Performance Trade-offs, Lessons Learned & Future Roadmap

The Post-Quantum Migration Reality

- NIST Standards Finalized: FIPS 203 (ML-KEM), FIPS 204 (ML-DSA), and FIPS 205 (SLH-DSA).
- Systemic Cryptographic Risk: Over-reliance on structured lattices demands mathematical diversity.
- NIST On-Ramp Round 3: New signature candidates named.
- Urgent Industry Need: Rapid experimentation with non-lattice primitives in real-world software.

The libOQS Mission & Ecosystem

- Flagship C Library: Open Quantum Safe provides uniform, standard C APIs for quantum-resistant KEMs & Signatures.
- Academic Foundation: Founded by Michele Mosca & Douglas Stebila at the University of Waterloo.
- Linux Foundation PQCA: Core project under the Post-Quantum Cryptography Alliance.
- Downstream Production Feed: Isolates apps from raw research code while feeding OpenSSL 3.x, NGINX, and network clients.

Criteria: NIST Status & Code Maturity

- Continuous Ingestion Goal: We continue adding new algorithms to advance cutting-edge cryptographic research.
- NIST Process Tracking: Prioritizing Round 3 signature on-ramps and Round 4 KEM alternates.
- Peer-Reviewed Maturity: Demanding published cryptanalysis and vetted resistance to known attacks.
- Test Vector Completeness: Requiring verified Known Answer Test (KAT) suites before ingestion.

Criteria: Math Diversity & Licensing

- Mathematical Diversity: Prioritizing Symmetric/VOLE, Error-Correcting Codes, Multivariate, and Isogenies.
- Combating Homogeneity: Ensuring robust fallbacks if lattice hardness assumptions are compromised.
- Licensing Readiness: Clean C implementations under permissive open-source licenses (MIT / Apache 2.0).
- Architectural Fit: Feasibility of conforming to unified caller-allocated memory models.

Integration Tooling & Automation

- API Boilerplate Generation: Automated scripts generating standard OQS_SIG and OQS_KEM C wrapper structs.
- Symbol Renaming: Automated namespacing scripts preventing symbol collisions between algorithms.
- KAT Harness Ingestion: Automated conversion of NIST KAT test vectors into native libOQS test suites.
- CI/CD Build Scaffolding: Automated CMake target generation and multi-architecture test matrix runners.

Maintenance & Author Outreach

- Upstream Evolution: Tracking specification updates and parameter tweaks across NIST rounds.
- Clean Rebase Strategy: Maintaining clear downstream .patch files vs pristine upstream vendor snapshots.
- Early Author Outreach: Engaging algorithm designers early to clarify specification subtleties and edge cases.
- Upstreaming Improvements: Contributing bug fixes and vector optimizations back to authors' reference trees.

Spotlight: FAEST Signature Scheme

- Mathematical Base: Symmetric primitives (AES / Rijndael) + VOLE-in-the-Head zero-knowledge proofs.
- Public Key Footprint: Extremely compact (~32 to 64 Bytes)—fits seamlessly into standard X.509 certs.
- Signature Payload: Moderate size (~5 KB to 20 KB)—larger than classical RSA/ECDSA.
- Performance Profile: Compute-heavy signing (VOLE proof & seed trees); fast and lightweight verification.

Spotlight: Classic McEliece KEM

- Mathematical Base: Error-correcting codes (Goppa codes)—unbroken track record since 1978.
- Public Key Footprint: Massive (~250 KB to 1.04 MB)—stresses network MTUs and memory limits.
- Ciphertext Payload: Ultra-compact (~128 to 240 Bytes)—fits easily in a single TCP packet.
- Performance Profile: Ultra-fast decapsulation -- high initial keygen memory overhead.

The End-to-End Lifecycle

- Iteration 1: Architectural Alignment & API Mapping in libOQS.
- Iteration 2: Build System Plumbing & CMake SIMD Isolation.
- Iteration 3: OpenSSL 3.x Provider Integration via oqs-provider.
- Iteration 4: Cross-Platform Testing, CI/CD & Sanitizer Hardening.
- Production Goal: Fully usable primitives in real protocols (TLS 1.3, X.509, CMS).

Conforming to libOQS Dispatch Structs

- Standard C Structs: Mapping algorithms into uniform OQS_SIG and OQS_KEM function tables.
- Uniform Function Pointers: Standard keypair(), sign(), verify() and encaps(), decaps() signatures.
- Dynamic Length Metadata: Exposing public key, secret key, and ciphertext lengths via struct fields.
- RNG Isolation: Abstracting entropy sources via OQS_RAND for deterministic KAT reproducibility.

Memory Lifecycles & Buffer Pitfalls

- Caller-Allocated Mandate: libOQS enforces caller-allocated buffers for zero-copy efficiency.
- The McEliece 1MB Stack Hazard: Allocating 1MB public keys on the stack causes immediate stack overflow.
- Heap-Based Lifecycles: Enforcing explicit heap management and strict ownership semantics in wrappers.
- State Isolation: Managing deep VOLE scratchpads and zeroizing sensitive memory with OQS_MEM_cleanse.

CMake SIMD & Flag Isolation

- Vector Optimizations: Vendoring upstream SIMD routines (AVX2, AVX-512, ARM NEON).
- Preventing Flag Bleed: Scoping target_compile_options so -mavx2 never leaks into core library code.
- Target Sandboxing: Isolating architecture-specific source files into dedicated CMake object libraries.
- Fallback Integrity: Ensuring generic portable C fallback compiles cleanly across all architectures.

OpenSSL 3.x Provider Dispatch

- Modular Provider Architecture: Replacing legacy ENGINE API with OpenSSL 3.x OSSL_DISPATCH tables.
- OSSL_FUNC_KEYMGMT_*: Implementing key generation, import/export, and parameter validation.
- OSSL_FUNC_SIGNATURE_*: Hooking FAEST into standard EVP_DigestSign and EVP_PKEY_sign APIs.
- OSSL_FUNC_KEM_*: Hooking Classic McEliece into standard EVP_PKEY_encapsulate workflows.

OIDs, ASN.1 & Jumbo Certificates

- Provisional OIDs: Registering experimental OIDs under the OQS arc for all algorithm variants.
- SPKI Serialization: Encoding raw PQC key bytes into standard ASN.1 DER SubjectPublicKeyInfo structures.
- Jumbo Certificate Challenge: Handling >1MB X.509 certificates generated by Classic McEliece.
- Parser Limits: Tuning OpenSSL default ASN.1 buffer boundaries for large certificate test PKIs.

KAT Validation & Memory Sanitizers

- Known Answer Tests (KAT): Validating test vectors with fixed RNG seeds across all CPU architectures.
- AddressSanitizer (ASan): Automated CI runs detecting buffer overflows and out-of-bounds reads.
- UndefinedBehaviorSanitizer (UBSan): Catching integer overflows, misalignment, and shift violations.
- MemorySanitizer (MSan): Catching uninitialized memory reads across GCC and Clang toolchains.

Operational Trade-off Analysis

- Classic McEliece: Instant decapsulation (<0.1 ms) + tiny ciphertext; best for static/cached public keys.
- FAEST: Tiny public key (~32–64 B) fits easily in standard X.509 certs; signature compute is heavier.
- Network Bottlenecks: McEliece increases TLS handshake packet size (requires customized TLS stack); FAEST increases server CPU signing load.
- Architectural Guidance: Choose primitive based on network bandwidth vs compute constraints.

Lessons: Deduplication & Primitives

- 1. Decide How to Handle Duplicate Files: Deduplicate shared helper utilities (RNG, AES, Keccak) to prevent symbol collisions.
- 2. Add Implementations for Missing Primitives: Polyfill missing hash functions or platform intrinsics with clean portable fallbacks.
- 3. Enforce Strict Compiler Cleanliness: Vendor upstream code cleanly with full -Wall -Wextra -Werror compliance.

Lessons: Patches, Authors & Community

- 3. Determine When to Use Patched Files vs. Meta Files: Use explicit .patch manifests for clean rebasing over upstream updates.
- 4. Reach Out to Authors and Involve Them Early: Direct author collaboration clarifies specification nuances and speeds bug fixes.
- 5. Build Community: Engage the broader open-source ecosystem, standard bodies, and the Linux Foundation PQCA.

SESSION CONCLUSION

Summary & Open Q&A

- Modular Abstractions: libOQS decouples raw research code from production-ready applications.
- Flexible and Adaptive Integration: Offers wide ranging algorithm supports.
- OpenSSL Provider Ready: Immediate benchmarking via oqs-provider across real network servers.
- **Presenter: Norman Ashley (nashley@cisco.com) | openquantumsafe.org**