RISC-V SUMMIT
NORTH AMERICA 2025

PQCP Support for RISC-V Vector, Future Keccak ISE

Markku-Juhani O. Saarinen
Cryptography SIG Chair

Post-Quantum Cryptography (PQC) Standards

Main NIST Standards (ratified in August 2024)

- FIPS 203: ML-KEM ("Kyber") Key Establishment – already widely in TLS
- FIPS 204: ML-DSA ("Dilithium") Digital Signature – coming into PKI
- FIPS 205: SLH-DSA ("SPHINCS+") Digital Signature – hash-based

Additional Standards Coming

- FIPS 206: FN-DSA (Falcon) draft expected soon – smaller signatures
- HQC-KEM standardization in progress

Other efforts

- NIST additional signatures competition: 14 candidates in round 2
- ISO/IEC 18033-2 with PQC (Likely: Classic McEliece, FrodoKEM, ML-KEM)

CNSA 2.0 = PQC / National Security Systems (NSS)

and intent of this policy is to complete transition by 2035 as stated in NSM-10, *National Security Memorandum on Promoting United States Leadership in Quantum Computing While Mitigating Risks to Vulnerable Cryptographic Systems,* (Reference e) and to promote adoption of CNSA 2.0 algorithms as rapidly as standardization, production, and validation allows. While products incorporating validated implementations of CNSA 2.0 are to be preferred, CNSA 1.0 algorithms are acceptable in all products through 31 December 2025. Beginning 1 January 2027, unless otherwise excepted through public messaging on nsa.gov, protection profile, capabilities package, or waived through the waiver process, CNSA 2.0 algorithms will be required in all new products and services that provide cryptographic protection for users or for updates. Barring such an exception, equipment and services which cannot or will not be updated to CNSA 2.0 algorithms must be phased out and replaced with equipment or services which support CNSA 2.0 algorithms by 31 December 2030. CNSA 2.0 algorithms are to be preferred in protocol negotiations once compatible equipment is deployed. CNSA 2.0 algorithms are mandated for all protocol use by 31 December 2031, unless excepted as noted above or waived through the waiver process.

CNSA 2.0 includes ML-KEM-1024, ML-DSA-87 and older hash-based LMS and XMSS. **No RSA, DL, or Elliptic Curve Crypto.**

European PQC Transition

On 23 June 2025, EU member states & Commission issued a "coordinated roadmap" for PQC transition.

This policy asks for European cyber standardization (Cyber Resilience Act, CRA etc) to enforce transition.

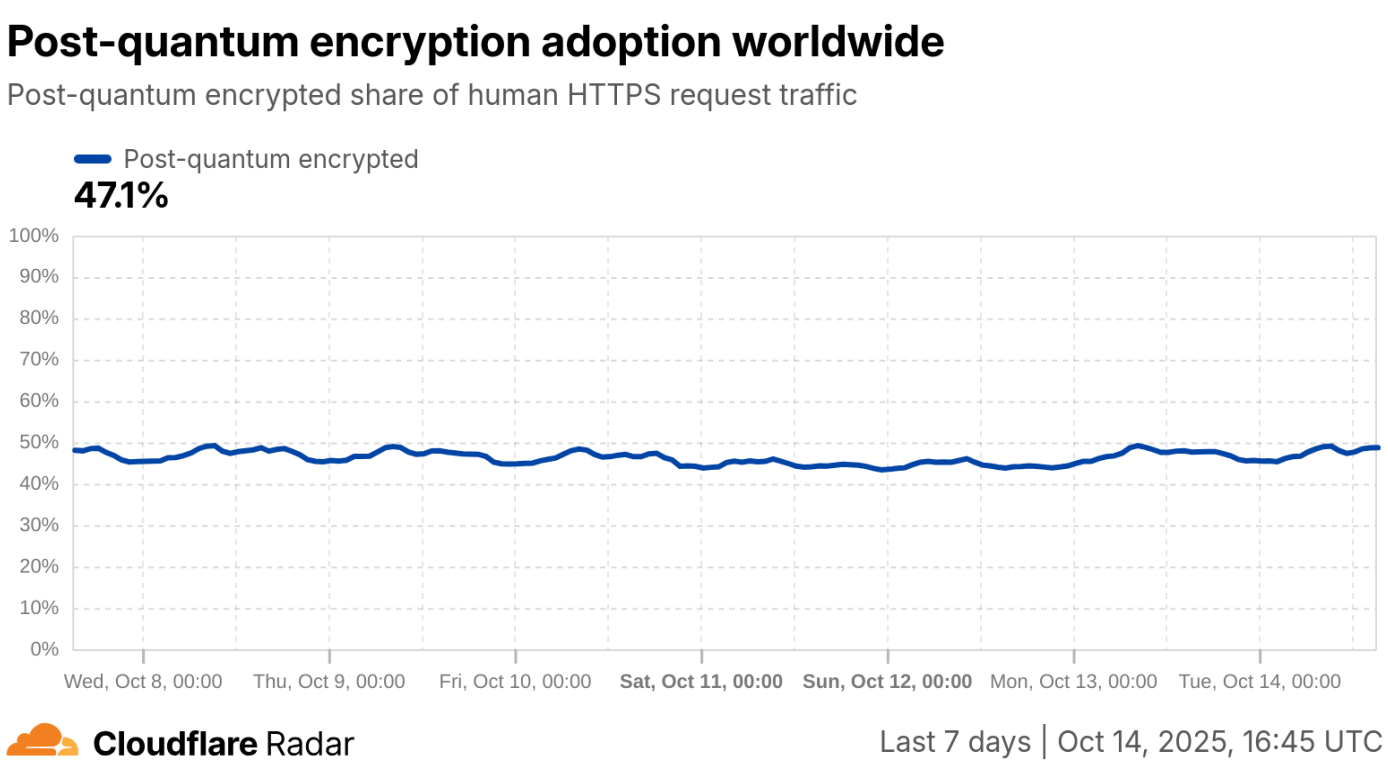
Software updates, “High-risk cases” before the end of 2030.

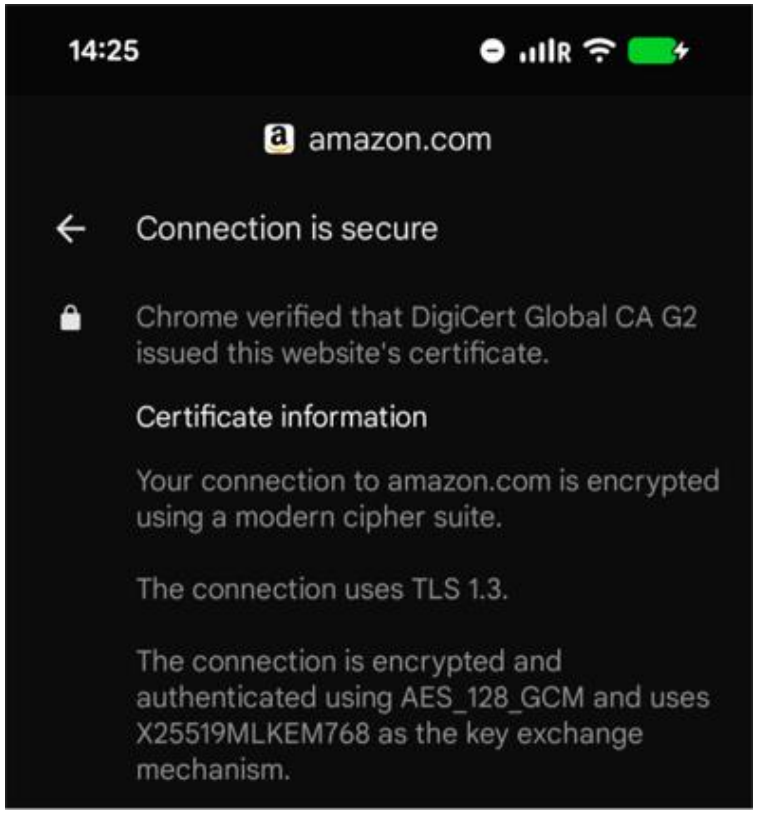
Rest before the end of 2035.

Timeline for the transition to PQC

1. By **31.12.2026**:
   - At least the *First Steps* have been implemented by all Member States.
   - Initial national PQC transition roadmaps have been established by all Member States.
   - PQC transition planning and pilots for high- and medium-risk use cases have been initiated.
2. By **31.12.2030**:
   - The *Next Steps* have been implemented by all Member States.
   - The PQC transition for high-risk use cases has been completed.
   - PQC transition planning and pilots for medium-risk use cases have been completed.
   - Quantum-safe software and firmware upgrades are enabled by default.
3. By **31.12.2035**:
   - The PQC transition for medium-risk use cases has been completed.
   - The PQC transition for low-risk use cases has been completed as much as feasible.

Reality Oct 2025: Almost 50% of non-bot HTTPS is PQC





- **Google** transitioned the Chrome browser and their services last year.
- **Cloudflare** (fronts a lot of big sites), **Apple**, **Amazon** have also transitioned.
- This is almost entirely **X25519MLKEM768** hybrid TLS 1.3 cipher suite.

PQCP: Post-Quantum Code Package

A Post-Quantum Cryptography Alliance (PQCA) effort (Linux Foundation)

Mission: High-speed, high-assurance implementations of NIST-standardized post-quantum cryptographic algorithms

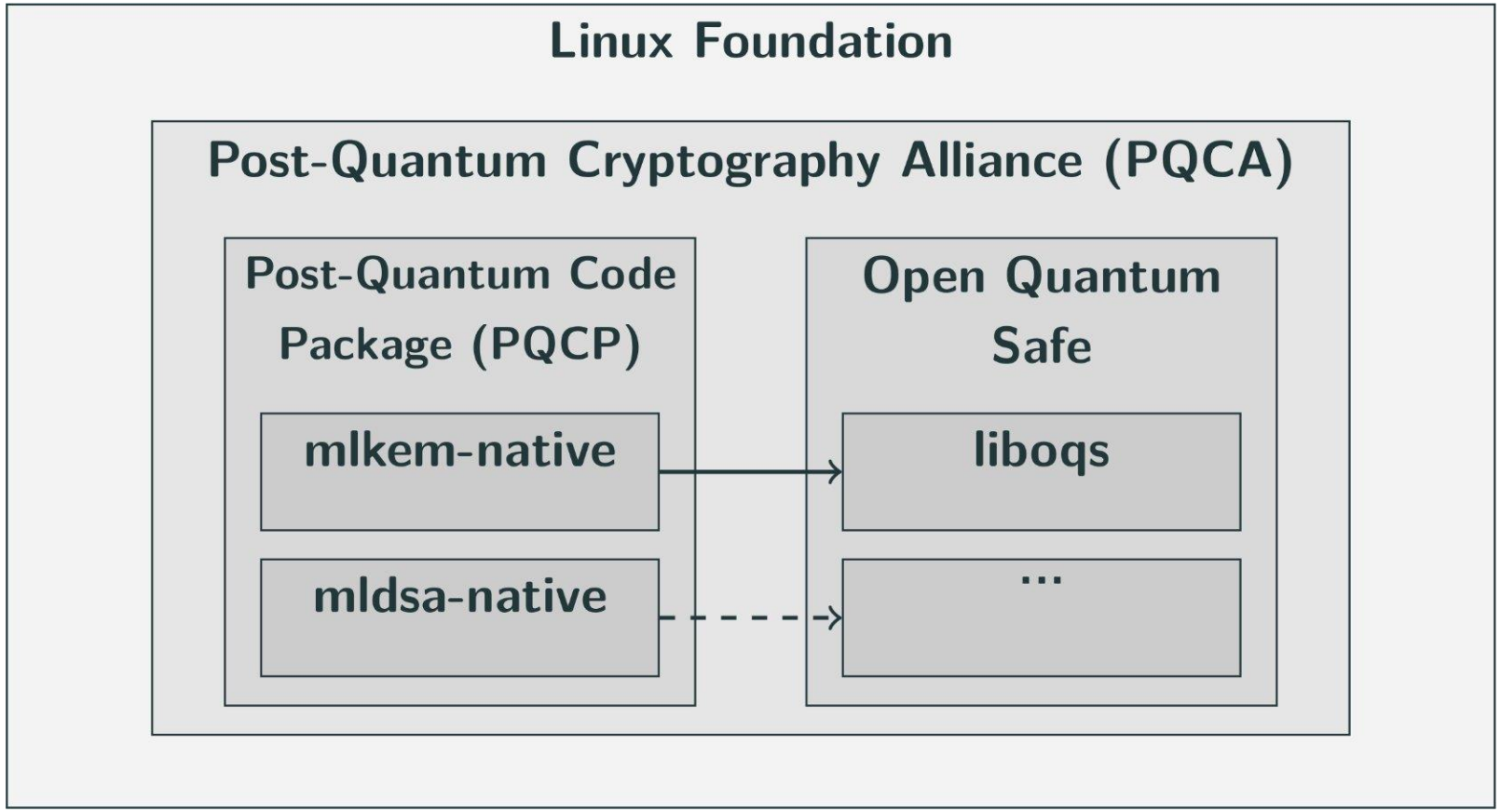
Focus: Production-ready code with rigorous security analysis

- Constant time, High-assurance implementations
- Multiple languages, platforms
- Liberally licensed: Default is Apache 2.0 (mlkem-native/mldsa-native: Apache-2.0 or ISC or MIT)

Projects:

mlkem-native, mldsa-native, slhdsa-c, mlkem-libjade, mlkem-rust-libcrux

mlkem-native: governance & maintenance



https://github.com/pq-code-package/mlkem-native

Maintainers: Hanno Becker (AWS), Matthias Kannwischer (Chelpis)

ML-KEM: computationally

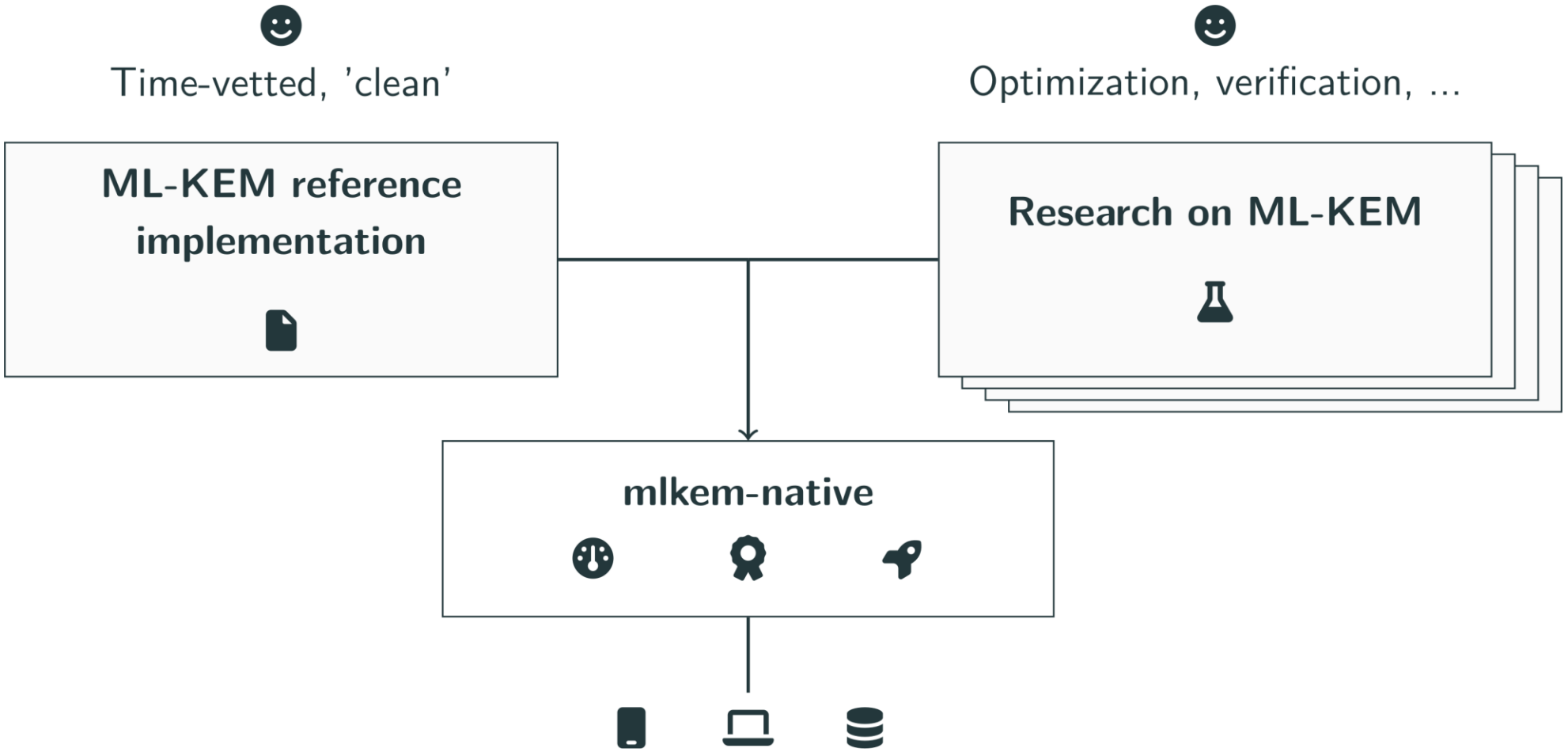
Key: Batching of independent instances

$$\text{⏱}(\text{ML-KEM}) \approx \text{⏱}(\text{SHA3/SHAKE}) + \text{⏱}\left(\mathbb{F}_{3329}[x]/(x^{256}+1)\right)$$

Key:

- Modular arithmetic mod 3329
- Avoiding modular reduction
- Number-theoretic transform

mlkem-native: research and optimizations



https://github.com/pq-code-package/mlkem-native

mlkem-native: consumers

libOQS of the Open Quantum Safe project

- Since version 0.13.0 (as the default ML-KEM implementation)
- https://github.com/open-quantum-safe/liboqs

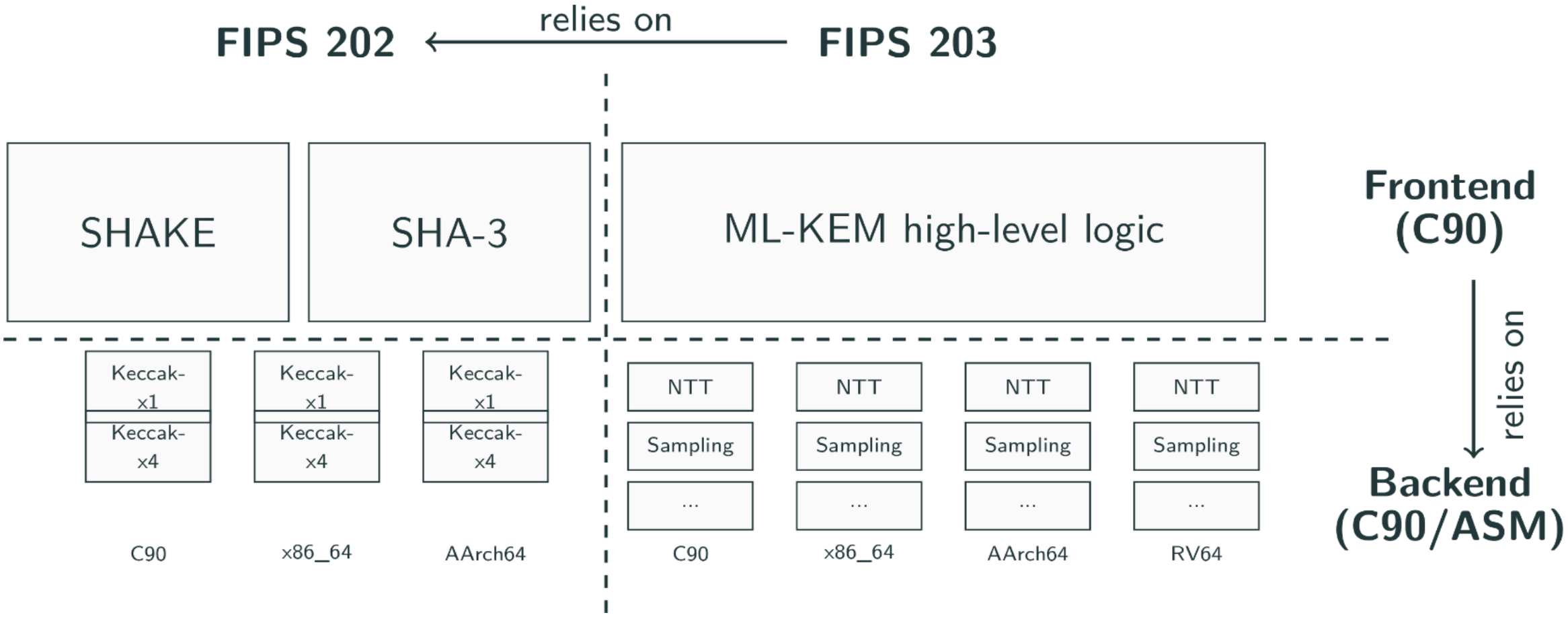
AWS-LC - AWS' Cryptography library

- Since version v1.50.0 https://github.com/aws/aws-lc

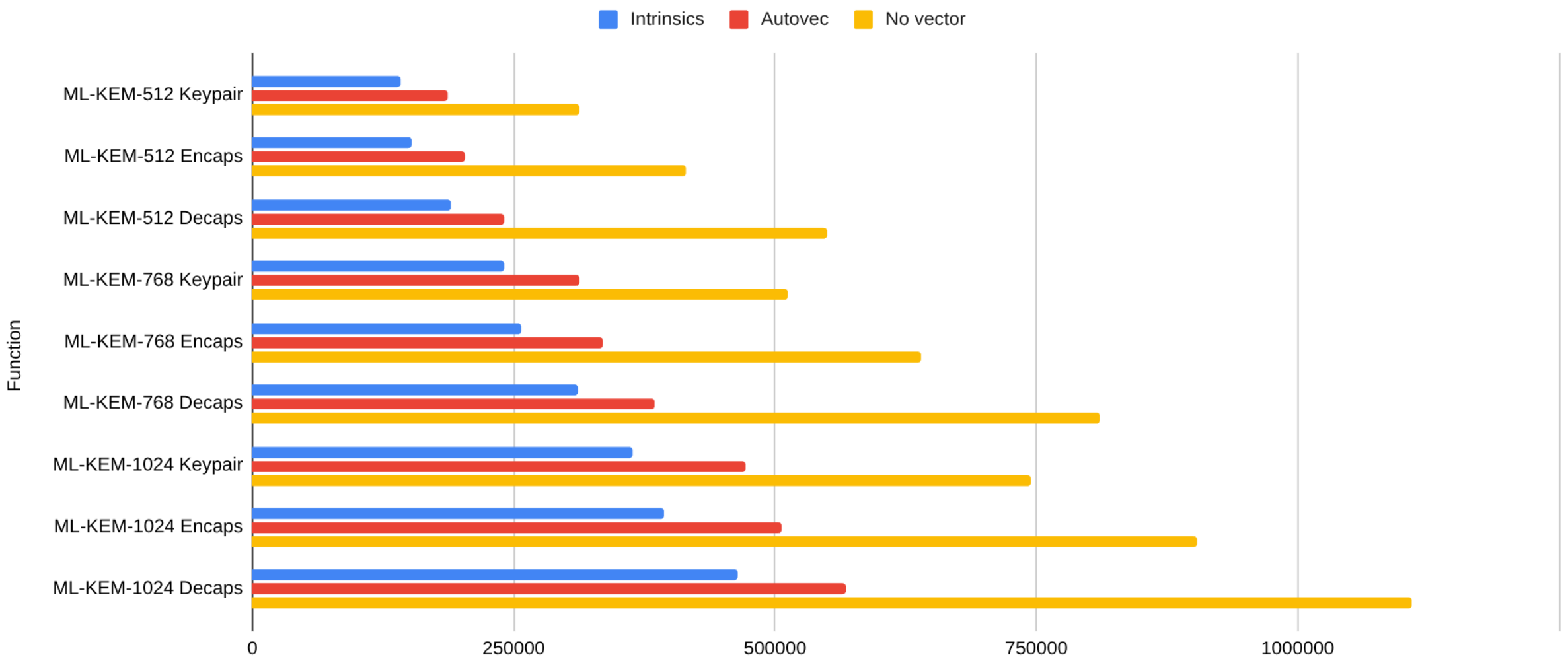
rustls - TLS library written in Rust

- Since version 0.23.28 (through AWS-LC as the default provider)
- https://github.com/rustls/rustls

mlkem-native: modularity



Cycles (LLVM 22, X60 core: RVA 22, VLEN=256)



Autovectorization doubles speed, w. Intrinsics 2.4x

- Using intrinsics + autovectorization means **2.42x** (clang) or **2.18x** (gcc) speedup over not using vector instructions.
- Intrinsics give a **28%** (clang) or **48%** (gcc) speedup over autovectorization.
- LLVM 22git autovectorization is 29% faster than GCC 14.2.1 autovectorization. (Compile flags: `-O3 -march=rv64gcv`)
- With plain RV64GC ISA (`-O3 -march=rv64gc`), the two compilers perform roughly the same on a pure C implementation..
- OpenSSL (`ml_kem.c`) and Google's BoringSSL (`kyber.cc`) seem to rely on autovectorization alone.

mlkem-native: formal verification

Goal: Type-safety and memory-safety (no overflows)

Tool: C Bounded Model Checker (CBMC)
https://github.com/diffblue/cbmc

Approach: Automatic proofs from per-function in-source contracts, invariants, bounds

Coverage: All C code

Continuous Integration: Runs on every change (~15 min / parameter set)

Note: Assembler backends (ARM, x86) also use HOL-Light for proofs that the implementation matches a mathematical model.

mlkem-native: timing attack protection

All code in mlkem-native is written with constant-time in mind:

- No secret-dependent branches or memory accesses
- No variable-time instructions
- Value-barriers preventing harmful compiler optimizations
- Barriers allow inlining critical functions and link-time optimization
  (But we can't make assumptions about consumer's optimization flags..)

Use valgrind to test for violations:

- Will catch violations of RISC-V DIEL (Zkt and Zvkt extensions.)

Sister project: mldsa-native

https://github.com/pq-code-package/mldsa-native

mldsa-native: High-speed, high-assurance C90/assembly ML-DSA

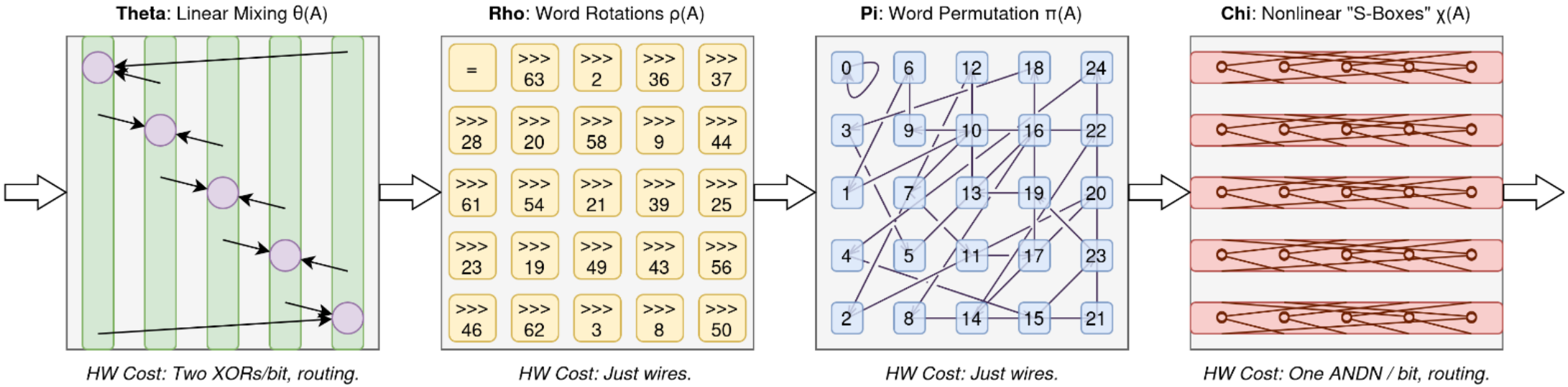
Goal: Same high-assurance approach as mlkem-native for digital signatures

Current Status:

- CBMC proofs - Complete for frontend and C backend
- Constant-time hardening & testing - Value barriers
- CI - Testing infrastructure almost on par with mlkem-native
- Configuration support - Most mlkem-native configuration options supported
- Backends: x86_64, AArch64 for most essential functions (not complete yet)

PQC Task Group: Keccak instruction proposal

- These implementations >50% of cycles on the Keccak f1600 permutation used by SHA3 / SHAKE, which vectorizes poorly.
- The main PQC TG proposal remains a Keccak instruction: Doubles not only ML-KEM but also ML-DSA (signing) speed.



PQC TG: Keccak instruction

Architecturally unusual:

- Performs the 1600-bit Keccak permutation with 1 instruction.
- Needs to read/write a register group containing 25 × 64-bit words.
- But provides incredible speedup, from ~2000 cycles down to ~40.
- Much faster than "partial" SHA3 instructions on ARM ISA.

Current status:

- Has spike, benchmarks (for a while now). But no HW PoC yet.
- Spec + Internal review scheduled before the end of the year.

Future Work

PQCP Findings and ML-KEM RISC-V status:

- Autovectorization alone gives lattice PQC 2× boost, vector intrinsics for NTT and other key steps: additional +30% or +40%
- NTT probably needs to be factored for different hardware VLEN values (128, 256, 512, 1024, others?)
- We should also add "parallel" Keccak (using Zvbb).

PQ Task Group and the Keccak instruction:

- Needs a hardware PoC for the highly effective Keccak instruction.

RISC-V SUMMIT

NORTH AMERICA 2025