Benchmarking RISC-V Post-Quantum Crypto

Markku-Juhani O. Saarinen
<markku-juhani.saarinen@tuni.fi>

RISC-V Summit 2023 / November 8, 2023

Talk Outline

1. Sitrep: Post-Quantum Cryptography Standards

2. Kyber and Dilithium quantitative analysis on RISC-V

3. New Vector Instructions proposed for PQC / modern crypto

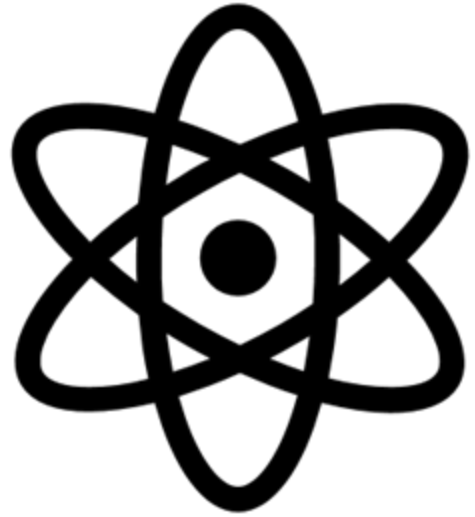
One-Minute Introduction to PQC

Post-Quantum Cryptography (PQC) = Cryptography that is **not** vulnerable to attacks by quantum algorithms when bigger quantum computers are built.

Symmetric cryptography (AES, SHA2, SHA3) is not vulnerable. There is no need for Quantum RNGs. Most of the current RISC-V Crypto Extensions (Zk*) are fine.

RSA (factoring) and Elliptic Curve (ECDL) cryptography has to go. Quantum algorithms (Shor's and Regev's) break these very efficiently -- polynomial time, increasing key size not much help.

After 7 years of analysis, newer PQC / Quantum-Secure (lattice-based, code-based, hash-based, ..) signatures and key establishment standards are being rolled out into applications.



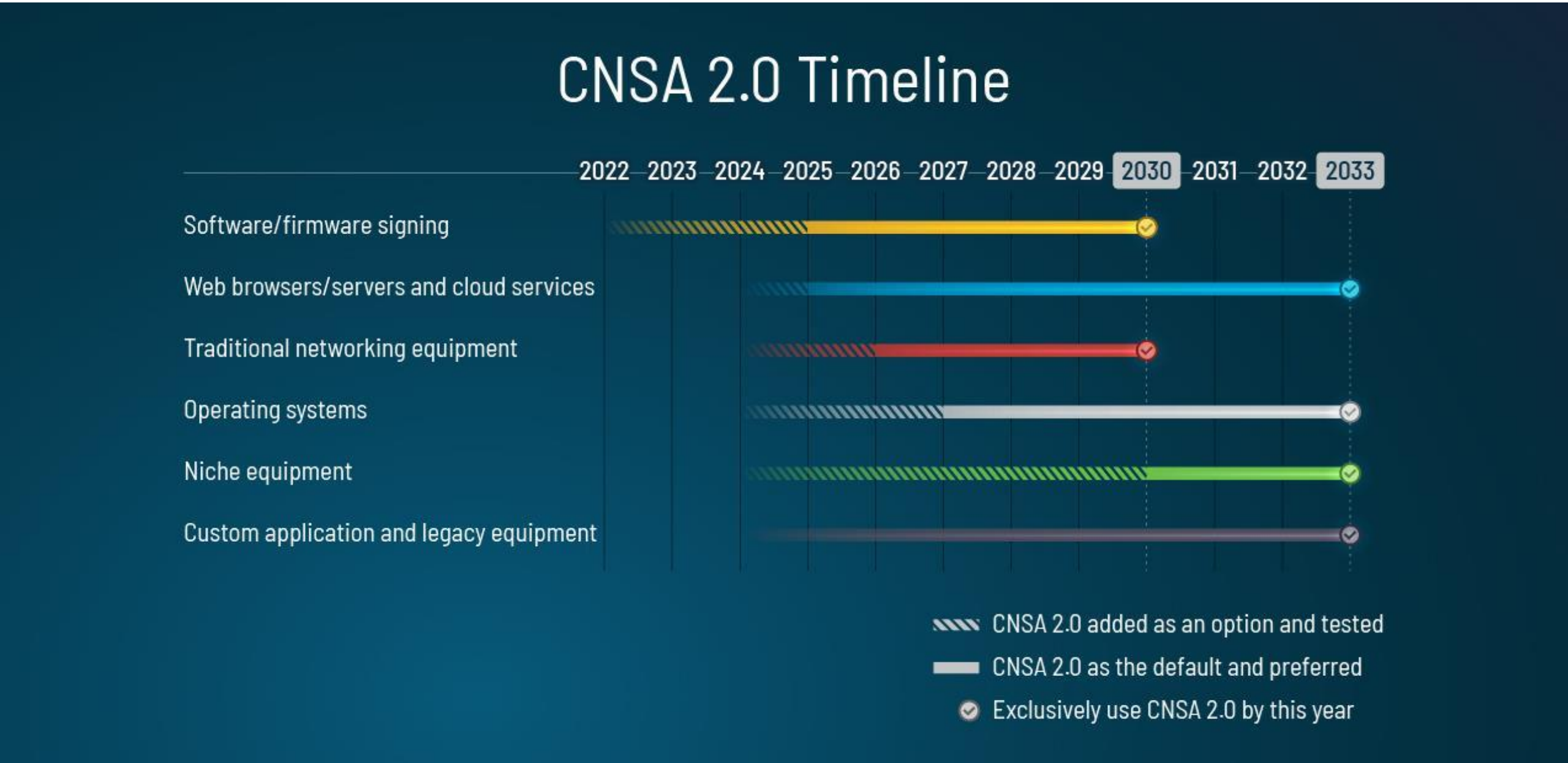
PQC Standards

The primary standards are lattice algorithms from the "PQ Crystals" suite.

PQC can be also used in "hybrid" modes alongside legacy crypto algorithms.

FIPS 203 ML-KEM ("**Kyber**") for Key Establishment. Replaces EC Diffie-Hellman key exchange (example: TLS handshake), ElGamal and RSA in Encryption.
FIPS 204 ML-DSA ("**Dilithium**") for Signatures. Replaces RSA and ECDSA signatures in web authentication, PKI certificates, software updates.
FIPS 205 SLH-DSA ("**SPHINCS+**") Stateless Hash-based Digital Signature Algorithm. Likely to see use in "root of trust" applications, firmware updates.
Initial Public Draft (IPD) Standard Specifications (2023-08-24): https://csrc.nist.gov/pubs/fips/203/ipd (replace 204, 205..)
FIPS 206 FN-DSA ("Falcon") IPD will follow later (2024?) Also: Round 4 KEMs, and the "signature on-ramp." These are not primary algorithms.

This is the CNSA (DoD & IC) transition timeline. The U.S. Federal Government and some civilian organization have their own timelines. Note "default" by 2025-27..



Talk Outline

1. Sitrep: Post-Quantum Cryptography Standards

2. Kyber and Dilithium quantitative analysis on RISC-V

3. New Vector Instructions proposed for PQC / modern crypto

Kyber Quant: Implementations Measured

Implementation 1: Reference / PQClean code (Optimized C)

Used (or adopted in some form) by many vendors. Example: via Rust wrappers now in libsignal / PQXDH of production Signal (Sep 2023.)

Implementation 2: BoringSSL / Google code (Optimized C)

Production code: In Chrome 116+ (Aug 2023), Android 14 (Oct 2023.)

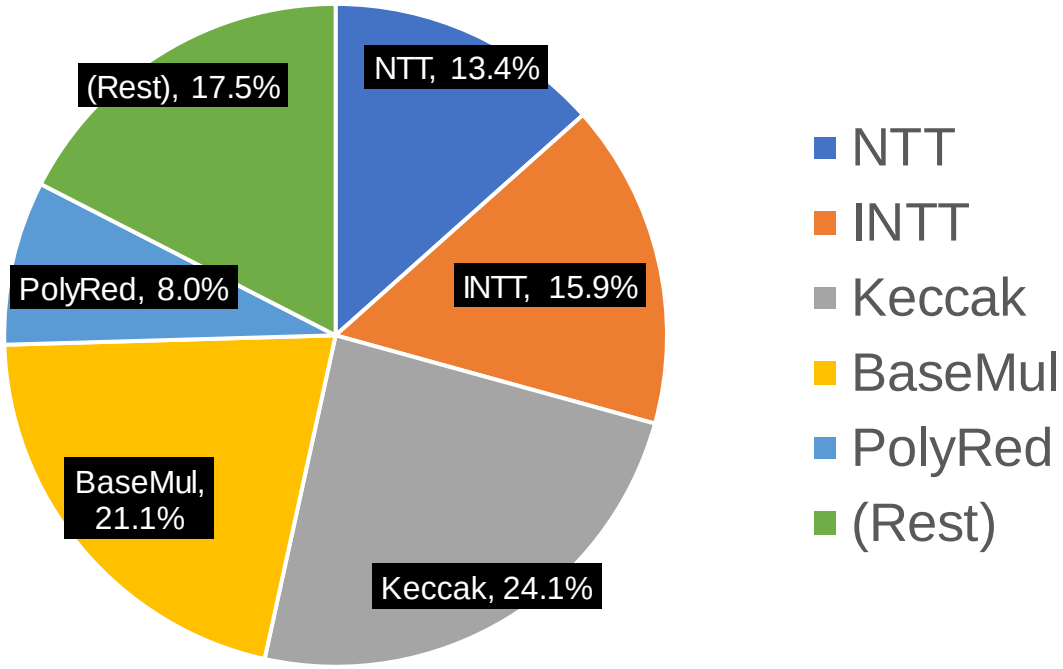
Benchmarked Kyber768, which is used in current draft TLS standard.

Ephemeral Key establishment = sum of KeyGen + Encaps + Decaps.

Kyber: Mostly Keccak + mod 3329 arithmetic

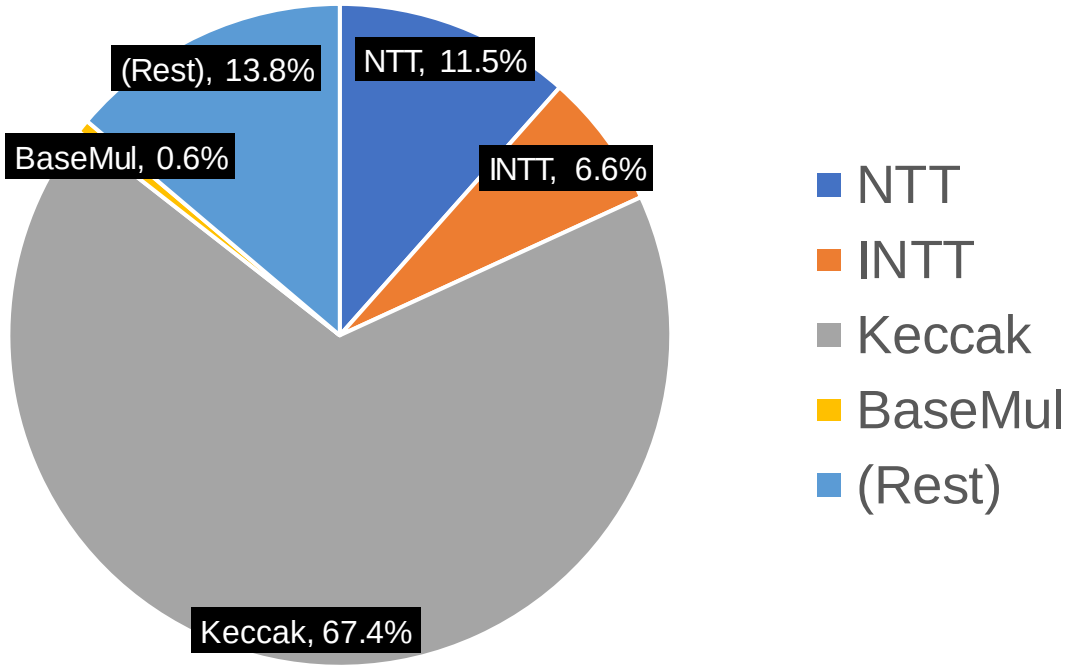
Reference Kyber768: 2.26M Insn

KG 600k + Enc 734k + Dec 921k



BoringSSL Kyber768: 2.26M Insn

KG 789k + Enc 978k + Dec 491k



Instret (with vlen:128,elen:64) - LLVM 18 snapshot, Oct 2023. `-Ofast -march=rv64gcv_zbb (zvk)`

"Keccak"
~½ of PQC Perf

SHA3 & SHAKE (FIPS 202) are built from Keccak f1600 keyless permutations.

- Fast scalar code fits the 1600-bit state into 25*64 registers, unrolls 1 round.
- Use RORI and ANDN from bitmanip (or krypto) for additional +50% perf!
- Vectorization is complex.

Why 67% of Keccak in BoringSSL but only 24% in Reference Kyber? 🤔

Same Kyber768: Same number (~132) of Keccak f1600 calls in KG+Encap+Decap.

BoringSSL's f1600 is just pretty slow.

BoringSSL: 11440 Insn (rv64gcv_zbb)

Reference: 4123 Insn (rv64gcv_zbb)

Without Zbb/Zkn: 6127 Insn (rv64gcv)

Crypto vector ext's: no impact currently.

Another ½ of Kyber: Mod 3329 Arithmetic

- The "structured lattice" ring arithmetic (NTT, INTT) in Kyber uses a single fixed modulus q = **0xD01** (at all security levels). Often stored as 16-bit.
- Modular arithmetic in finite field GF(q): Every multiplication requires a Montgomery and/or Barrett reduction step (`rem[u]` not constant time.)
- Implementations use different techniques. But it's the same Kyber768, same number of NTT (×15), INTT (×9), and BaseMul / ScalarMul (×36).
- NTT/INTT and ring arithmetic = vectorizable butterfly operations. BoringSSL already benefits a bit from compiler autovectorization! 🔍😯
- *We will have to look esp. at vectorizing the A matrix rejection sampler. The rest of Kyber cycles is are mostly spent on Serialization/Deserialization etc.*

Vectorized Barrett Reduction is great..

BoringSSL kyber.c reduce()

```
// constant time reduce x mod kPrime using Barrett
   reduction. x must be less
// than kPrime + 2×kPrime².
static uint16_t reduce(uint32_t x) {
  assert(x < kPrime + 2u * kPrime * kPrime);
  uint64_t product = (uint64_t)x * kBarrettMultiplier;
  uint32_t quotient = (uint32_t)(product >> kBarrettShift);
  uint32_t remainder = x - quotient * kPrime;
  return reduce_once(remainder);
}
```

.. but (mod q) multiply-add is still half-dozen instructions.

Autovectorized reduce()

```
; uint16_t odd = reduce(step_root * s->c[j + offset]);
1481a: b3 07 77 01   add         a5, a4, s7
1481e: 07 d4 87 22   vl2re16.v   v8, (a5)
14822: d7 76 20 0d   vsetvli     a3, zero, e32, m4, ta, ma
14826: 57 26 83 4a   vzext.vf2   v12, v8
1482a: 57 64 ca 96   vmul.vx     v8, v12, s4
1482e: 57 70 b0 0d   vsetvli   zero, zero, e64, m8, ta, ma
14832: 57 28 83 4a   vzext.vf2   v16, v8
; uint64_t product = (uint64_t)x * kBarrettMultiplier;
14836: 57 68 08 97   vmul.vx     v16, v16, a6
1483a: 57 70 20 0d   vsetvli   zero, zero, e32, m4, ta, ma
1483e: 57 36 0c b3   vnsrl.wi     v12, v16, 24
etc ...
```

Dilithium Quant Analysis Targets

Reference / PQClean code (Optimized C)

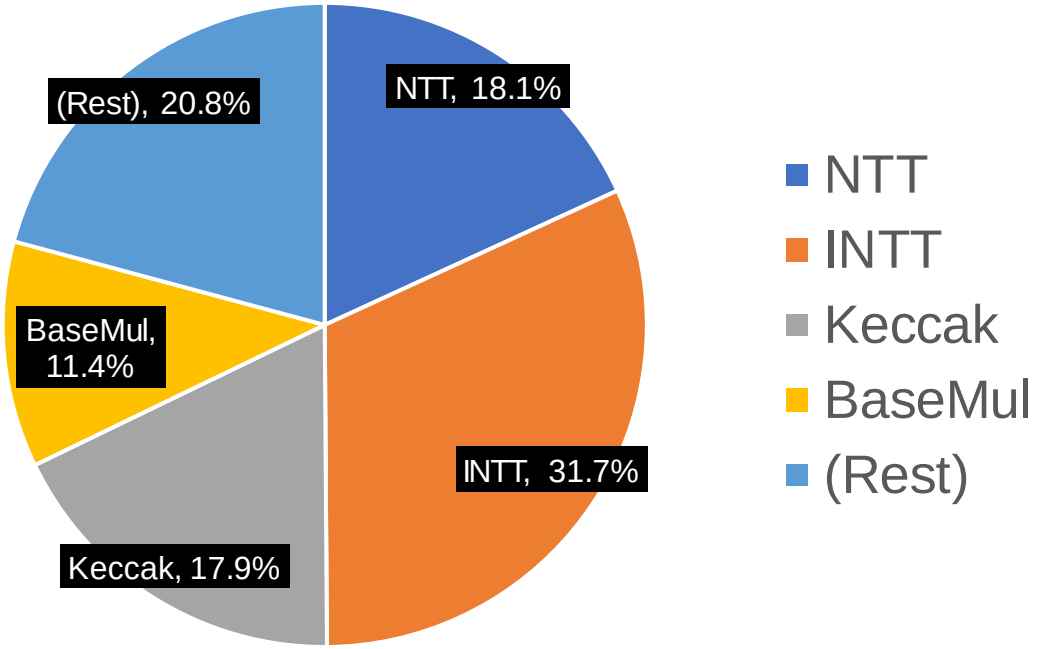
Code adopted by many projects, including PKI, HSM vendors.

- Param set ML-DSA-44 (Level 2) expected to be popular in PKI.
- Param set ML-DSA-87 (Level 5) has been selected for CNSA.
- Signing is probabilistic; We report average over 100.
- Note: (TLS) handshake has more verifications than signing.
- Dilithium KeyGen is fast -- roughly the same speed as verify (but this is not very important with signature schemes.)

Dilithium: Mod 8380417 arithmetic, Keccak

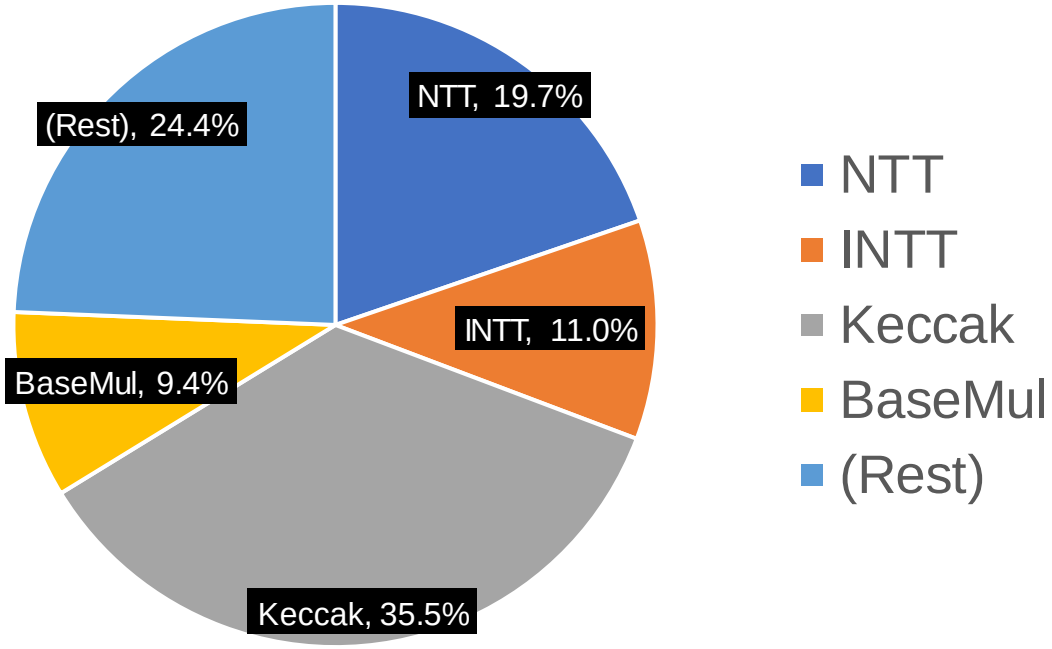
ML-DSA-44 Sign: Avg 4.60M Insn

ML-DSA-87 Sign: Avg 8.37M Insn



ML-DSA-44 Verify: 1.16M Insn

ML-DSA-87 Verify: 3.09M Insn



Instret (with vlen:128,elen:64): LLVM 18 snapshot, Oct 2023. `-Ofast -march=rv64gcv_zbb (zvk)`

Dilithium instruction mix: Similar to Kyber

- With Dilithium, **q = 0x7FE001** = 2^{23}-2^{13}+1 is the special fixed prime used in ring arithmetic and NTT "butterfly" ops.
- Keccak is the same (for SHA3 and SHAKE functions.)
- Most of the rest of the cycles: Serialization and complex but vectorizable "bit-dropping" arithmetic.

Opinion: *With Kyber & Dilithium as FIPS standards, I'm comfortable fixing the two q primes in hardware if there are substantial performance/energy/security gains.*

Some RISC-V Dilithium NTT Optimization

To optimize, I inlined the Dilithium reduction functions, utilizing some vectorization and almost halving the signing instruction count to:

ML-DSA-44 Sign: Avg. 2.36M Insn. Verify: 776k Insn.

After opt, modular mul (Montgomery) still has ~6 arithmetic instructions:

```
vsext.vf2 vmul.vx vsra.vx vmul.vx vmacc.vx vnsrl.wx
```

Observation: Vectorized small-integer modular arithmetic seems to always require widening, 1+ additional reduction multiplications, narrowing ops.

Talk Outline

1. Sitrep: Post-Quantum Cryptography Standards

2. Kyber and Dilithium quantitative analysis on RISC-V

3. New Vector Instructions proposed for PQC / modern crypto

Number Theoretic Transform Extension

Proposal 1: Vector Single-Width modular Integer multiply: `vmulq`

```
vmulq.vv  vd, vs1, vs2, vm  # vd[i] = (vs2[i] * vs1[i]) % q
vmulq.vx  vd, rs1, vs2, vm  # vd[i] = (vs2[i] * x[rs1]) % q
```

-> No *external* widening/narrowing required, replaces 5 or 6 instructions.

-> For fixed q values, efficient, const-time hardwired reduction is possible.

-> Impact: More than triples Kyber and Dilithium NTT performance.

How? Hardwire SEW=16: q=0xD01, else if SEW=32: q=0x7FE001 for `vmulq` **?**

Alternatives: Special “mod q” fixed-point rounding mode in `vxrm` and use vector fixed-point instructions? Or Would modular single width modular multiply-add be even faster? (“q” versions: `vmaccq vnmsacq vmaddq vnmsubq`).

Further Speedup of Keccak?

- Dedicated hardware does a Keccak f1600 permutation in **24 cycles** (with fast clock). Without vectorization 4000+ RV64 instructions, with "maximal" general-purpose vector optimization probably still hundreds.
- Shake/Keccak is important elsewhere too: SPHINCS+ (FIPS 205 SLH-DSA) and older XMSS, LMS (NIST SP 800-208) signature algs are 95% hashing.

<u>There have been some academic proposals for RISC-V Keccak:</u>

- Example: have a full f1600 permutation round in hardware and make it visible via the the double-precision floating point register file (!)
- Also: Instructions for 5x5 slides, specific rotates, etc. Each round requires many "Keccak insns" (I don't think this is necessary.)

Proposed Keccak Permutation Extension

- Vector Crypto (Zvk) already has `vandn`, `vrol` variants which allow substantial speedup and parallelized instances.
- Would need optimize `vrgather/vrgatherei` .. still quite complex.

- **Proposal 2:** Simply a dedicated 1600-bit Keccak Round(s) Instruction.

```
vkeccakp.wi vd, vs2, imm  # imm = num rounds
```

- Computes one or more rounds Rnd(A, i_r) of Keccak-p[1600,24].
- Apparent advantages in having this as a multi-round ("all-rounds") due to the overhead of getting 1600 bits from/to a register group.

Conclusions

Two main suggestions for standard FIPS PQC Algorithms:

1. Vector single-width modular multiply instruction, q={Kyber, Dilithium}
2. (Multi-round) Keccak f1600 permutation on a vector register group

- Substantial speedup: Help to establish thousands of TLS/QUIC.. connections per second, minimize connection latency.
- Proposed extensions may be *initially* worthwhile mainly on network accelerators and some types of high-end servers.
- Keccak/SHA3/SHAKE is assumed to be widely useful in the future.

Thank You!

Questions?