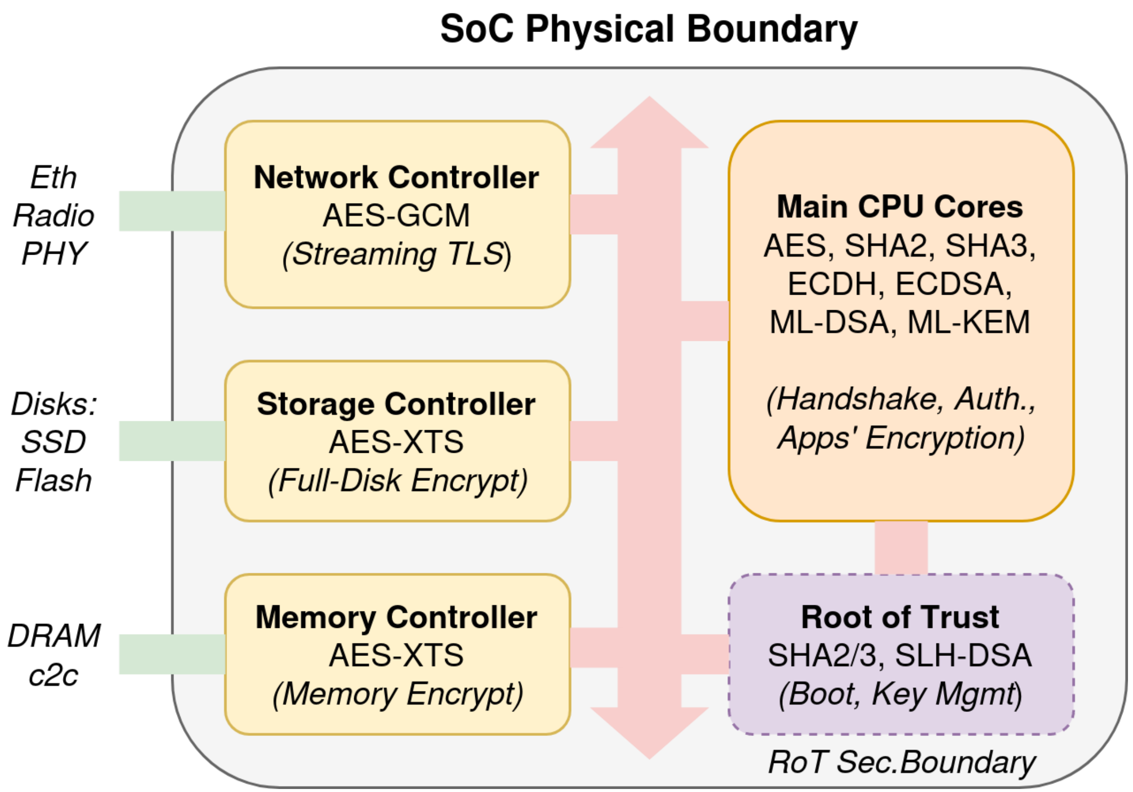
RISC-V Cryptography Evolution:
High Assurance Cryptography (HAC TG)
Post-Quantum Cryptography (PQC TG)

Markku-Juhani O. Saarinen *(Tampere University, Finland)*
G. Richard Newell *(Microchip Technology, USA)*, Nicolas Brunie *(SiFive, USA)*

Real World Crypto Symposium
Toronto - March 25, 2024

From the top (SoC view): We often want to *remove* the CPU from the encrypt loop..

- Perform disk encryption at disk or storage controller (transparently.)
- Offload bulk symmetric network encryption to the NIC or Modem.
- SoC-wide Platform Security crypto isolated in a Root of Trust (RoT).
- **CPU**: Left with (TLS) handshakes, asymmetric ops, crypto in apps. Need to speed them up *in the CPU*.
- *This is really, really simplified..*



Instruction Set Architectures: Everything Old is New Again

RISC-V is a lot like SPARC, MIPS, PowerPC, PA-RISC, DEC Alpha.. A bit less like ARM, and much, *much* less like x86.

RISC-V Vector (v-extension) is a "real" vector architecture a la Cray-I (1976), not fixed-length SIMD (AVX2, Neon).

In RISC-V, almost everything is an extension (other ISAs expand too, with "features".) Two kinds of extensions:

- **Custom**: DIY instructions at custom opcodes.
- **Ratified**: Consistent specs, reserved opcodes, simulator, compiler, operating system, support, ...

RISC-V® Org (RISC-V International)

- Oversees ISA Development, Certification. 4000 Members in 70 Countries.
- Big open-source/technical standardization org: Committees galore.
- Crypto "TGs" are under the Unprivileged Spec "IC" and Security "HC."

Some basic rules for new RISC-V instructions:

- Instructions need to demonstrate substantial, measurable advantages.
- Instructions need to fit into the big-picture RISC-V architecture.
- You need to *explicitly contribute* the instruction to the RISC-V ISA (membership.) This is to protect everyone against IP / patent problems.

Cryptography Extensions ("K")

Done: **Scalar Crypto** (Ratified 2021)**:** AES, SHA2, SM3, SM4, CMUL (GCM) with 32- and 64-bit **scalar registers**. + "Constant time" & Entropy Source.

Done: **Vector Crypto** (Ratified 2023)**:** AES, SHA2, SM3, SM4, GCM with **vector registers**: Make bulk crypto even faster with *parallel* AES-GCM etc.

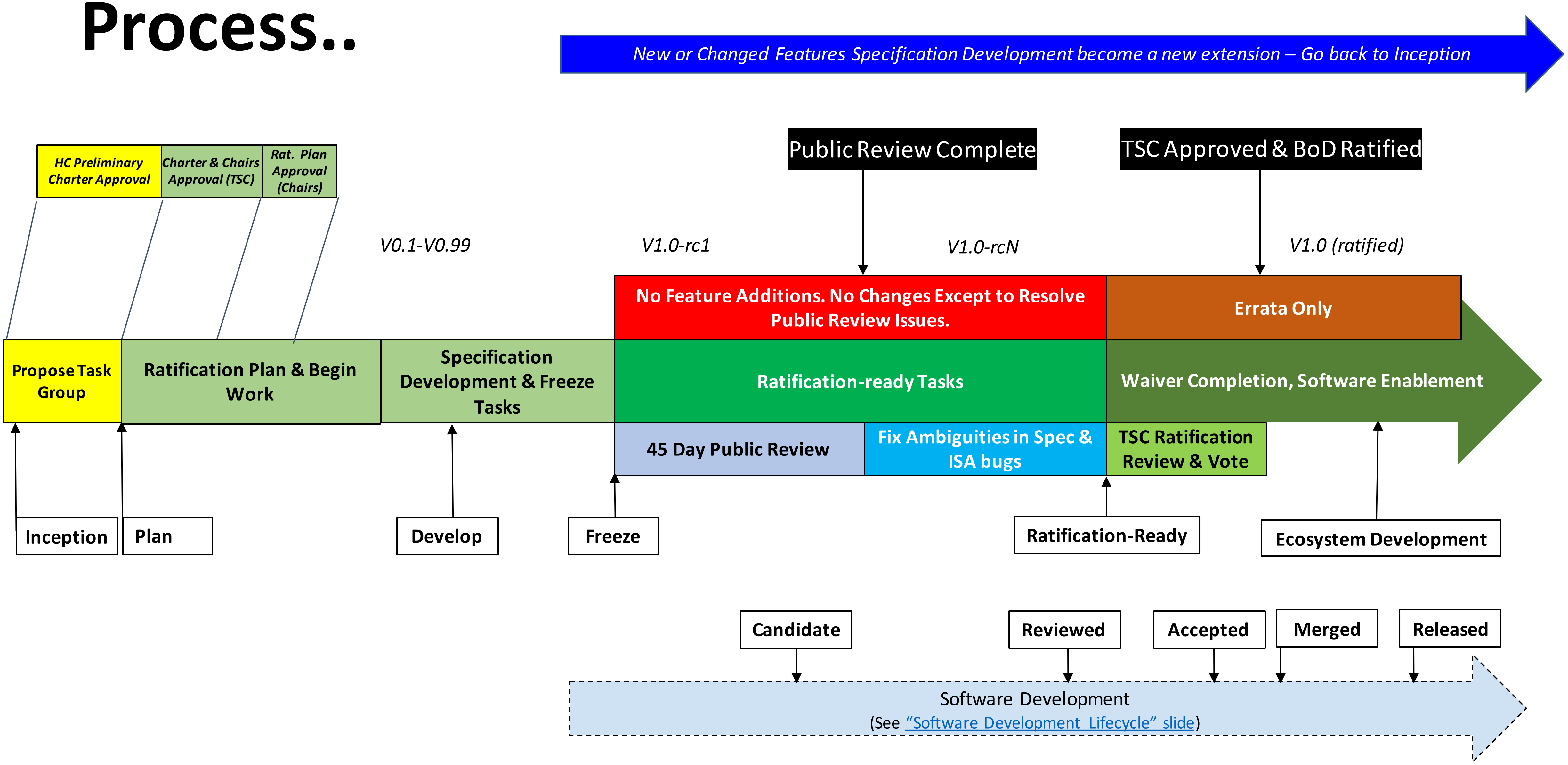
-> many of these now In Linux Kernel, OpenSSL, going into Android Platform

Being worked on:

High Assurance Crypto TG (From late 2023)**:** "Full-rounds" AES allowing emission/power side-channel security. Key management features.

Post-Quantum Crypto TG (From late 2023)**:** What can we do to assist standard PQC algs (notably FIPS 203,204,205 - Kyber, Dilithium, SPHINCS+) ?

Process..



New or Changed Features Specification Development become a new extension – Go back to Inception

HC Preliminary Charter Approval

Charter & Chairs Approval (TSC)

Rat. Plan Approval (Chairs)

Public Review Complete

TSC Approved & BoD Ratified

V0.1-V0.99

V1.0-rc1

V1.0-rcN

V1.0 (ratified)

No Feature Additions. No Changes Except to Resolve Public Review Issues.

Errata Only

Propose Task Group

Ratification Plan & Begin Work

Specification Development & Freeze Tasks

Ratification-ready Tasks

Waiver Completion, Software Enablement

45 Day Public Review

Fix Ambiguities in Spec & ISA bugs

TSC Ratification Review & Vote

Inception

Plan

Develop

Freeze

Ratification-Ready

Ecosystem Development

Candidate

Reviewed

Accepted

Merged

Released

Software Development

(See “Software Development Lifecycle” slide)

HAC TG: High Assurance Cryptography

RISC-V HAC TG is interested at cryptography instructions with additional security features such as side-channel resistance.

- Regular scalar and vector AES extensions leak side-channel information even in complex SoCs (https://ia.cr/2022/230).
- To use techniques like masking we will need a **Full-rounds AES**.
- We'd like to also remove "plaintext" secret keys from memory and the register file (use handles of some kind). Cold boot attacks etc.

Similar features already exist in other ISAs. *Need new ideas..*

PQC TG: Post-Quantum Cryptography

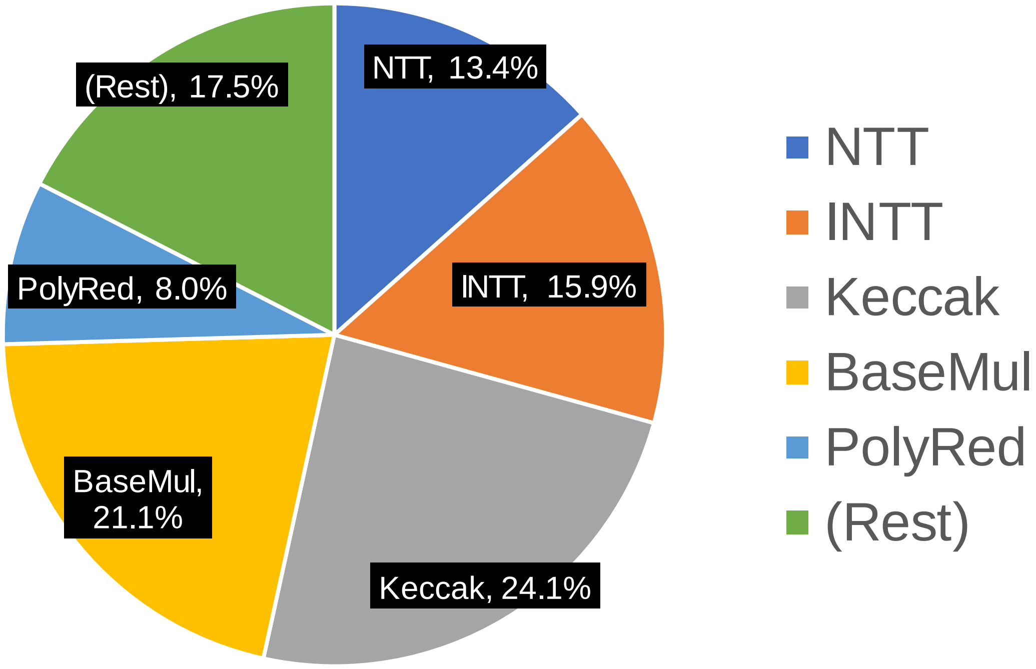
- **Kyber** (FIPS 202 ML-KEM) and **Dilithium** (FIPS 203 ML-DSA) are just as fast -- *or faster* -- than { RSA, ECDSA, ECDH } on current RVV CPUs.
- Due to flexibility required (e.g. hybrid crypto) and external interface complexities, asymmetric crypto is likely to remain in main CPUs.
- PQC TG evaluates **helper instructions**. To be considered, substantial perf advantages must be demonstrated, without too much cost.

Evaluation metric:

- What matters: **End-to-end latency** (µs/op) - when instantiated in a typical application (most often a TLS stack; server or client.)

Kyber: Vectors (mod 3329) + Keccak

Reference Kyber-768: 2.26M Insn
KG 600k + Enc 734k + Dec 921k



Compute in NIST Lattice Crypto:

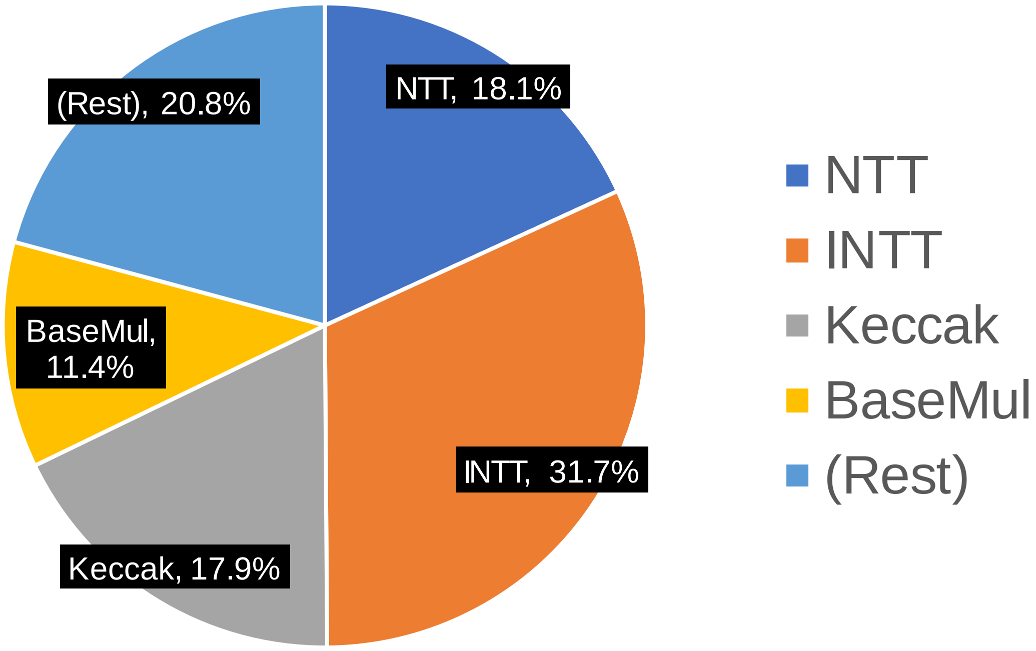
- **Keccak** i.e. SHA3/SHAKE operations. Can be even >50% of overall cycles.
- **Number Theoretic Transforms.** Vectorizable functions (256 x 16/32.)
- **Other polynomial arithmetic.** Mostly integer vectors; shifts, adds, sub.
- **Samplers** (rejection and CBD), rounding, "packing" (serialize).

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

Dilithium: Vectors (mod 8380417) + Keccak

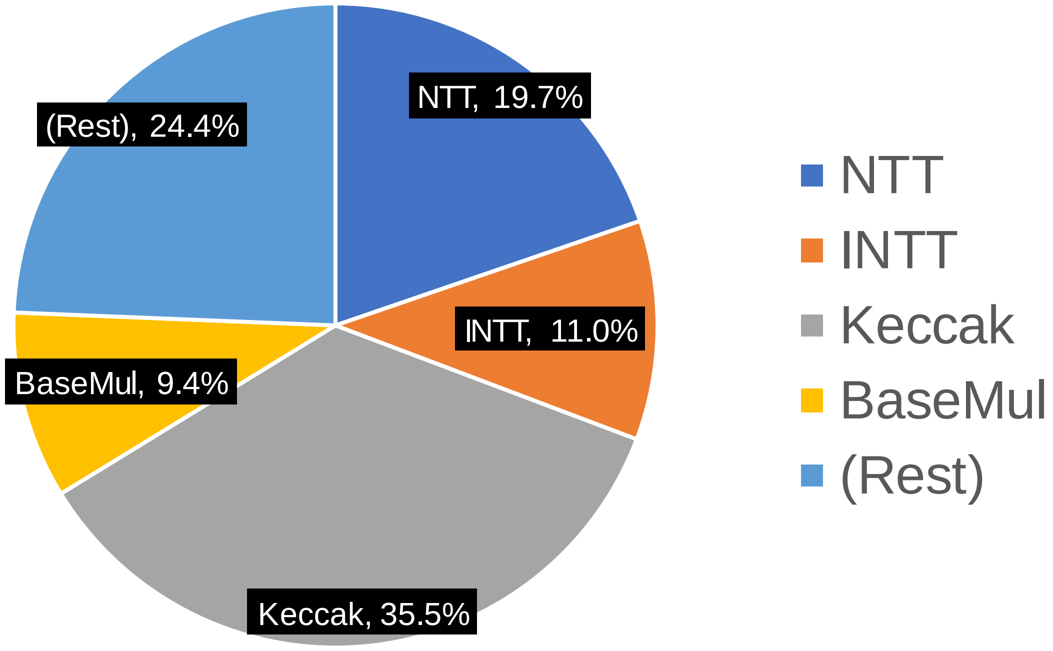
ML-DSA-44 Sign: Avg 4.60M Insn

ML-DSA-87 Sign: Avg 8.37M Insn



ML-DSA-44 Verify: 1.16M Insn

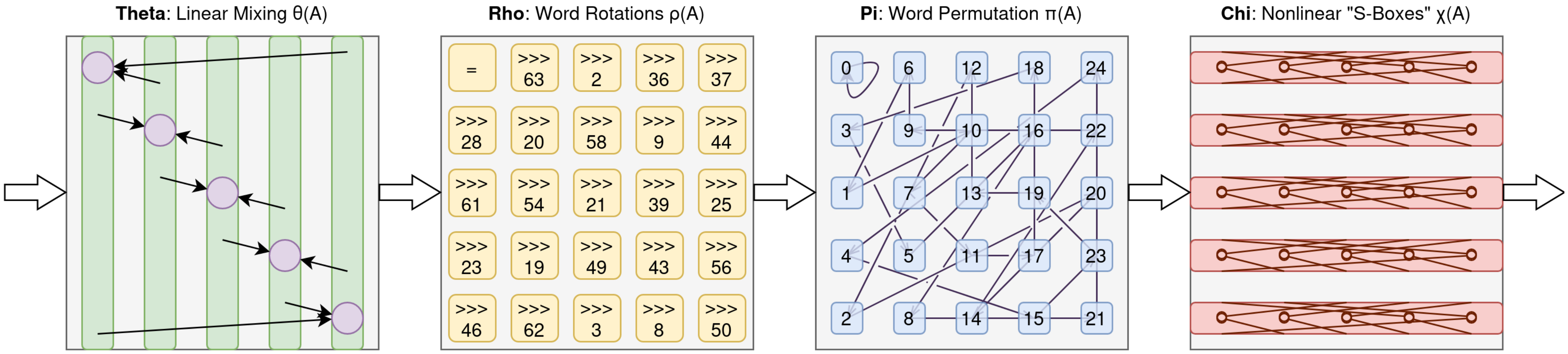
ML-DSA-87 Verify: 3.09M Insn



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

Bottleneck 1: Keccak f1600 properties

- **SHA3** and **SHAKE** (FIPS 202) are built on the 25×64=1600-bit Keccak permutation. ~50% of ML-KEM, ML-DSA Cycles, >90% SLH-DSA here.
- 24 Rounds. The rounds have an incredibly short critical path in hardware (fast hw!), but **vectorization is disappointing** (<2× scalar?)



Theta: Linear Mixing θ(A)

Rho: Word Rotations ρ(A)

=	>>> 63	>>> 2	>>> 36	>>> 37
>>> 28	>>> 20	>>> 58	>>> 9	>>> 44
>>> 61	>>> 54	>>> 21	>>> 39	>>> 25
>>> 23	>>> 19	>>> 49	>>> 43	>>> 56
>>> 46	>>> 62	>>> 3	>>> 8	>>> 50

Pi: Word Permutation π(A)

Chi: Nonlinear "S-Boxes" χ(A)

HW Cost: Two XORs/bit, routing.

HW Cost: Just wires.

HW Cost: Just wires.

HW Cost: One ANDN / bit, routing.

Bottleneck 1: A Proposal for Keccak

Keccak state is awkward to fit into vector registers and architecture:

- Seemingly VLEN ≥ 256 is required (the max LMUL value is 8.)
- Element EEW = 64. Element group EGS = 32, LMUL = 2048 / VLEN:
  - VLEN = 256: LMUL = 8: A group of 8 vector registers of 256 bits.
  - VLEN = 512: LMUL = 4: A group of 4 vector registers of 512 bits.

Multi-round instruction (due to complexity of accessing 25 words):

```
vkeccak.vi vd, vs2, imm   # imm = 5-bit num rounds
```

Computes 24 rounds of Keccak-p[1600,24] permutation with imm=24.

SPHINCS$^+$: Impact on FIPS 205 SLH-DSA

FIPS 205 SLH-DSA *"Stateless Hash-Based Digital Signature Standard"* (a.k.a. SPHINCS$^+$) has two parameter instantiations, SHA2 and SHAKE.

SLH-DSA-SHAKE is made at least 20 times faster by **vkeccak.vi**.

Note that holding the Keccak state in vector registers allows "padding template" forming and Winternitz iteration (https://ia.cr/2024/367).

Similar speedup for SHAKE variants of LMS & XMSS in SP 800-208.

Bottleneck 2: Number Theoretic Transforms

- **Number Theoretic Transforms** (NTT) are used by Kyber and Dilithium for fast polynomial ring $\mathbb{Z}_q[x]/(x^{256} + 1)$ multiplication.

To justify new instructions for NTT and ring arithmetic,

we need to understand the limits of existing RISC-V Vector.

- NTT is a divide-and-conquer algorithm. This is the finite field analog of FFT, DFT, etc, a common pattern for vectorization.
- **Observation**: Even though "hand" vectorization of Keccak didn't have significant returns, NTT is sped up even by autovectorization.

"PQCMark" [sic]: BoringSSL Kyber & Dilithium

- LLVM / **Clang** (v19 snapshot, March 2024) RISC-V Cross-Compilers.

  Clang 18.1+ has **RISC-V Vector Crypto** support (no longer "experimental".)

- Modified **Spike** ("golden" RISC-V software sim) with a patch from Nicolas Brunie that provides a Keccak instruction, state in element group in/out.

- For "real-life" code, checked out commit cf4f615 (March 2024) of **BoringSSL**, the standard cryptography stack in Google land: Android, Chromium, etc.

  Has **Kyber-768** (≈ ML-KEM-768) and (had) **Dilithium3** (≈ ML-DSA-65).

- Modified `keccak_f()` in `crypto/keccak/keccak.c` to optionally use the Keccak instruction (simple wrapper, not the ideal way to use it.)

Studies on RISC-V (Auto)vectorization

Clang 19.0git vectorizers (https://llvm.org/docs/Vectorizers.html) have reporting flags (e.g. `-Rpass-analysis=loop-vectorize`) to help identify reasons for failed/successful vectorization.

- *I Inlined some functions to make them available in the same compilation unit and removed some blockers (e.g. an assembler "const-time barrier".)*
- *It's still "constant-time" and It is still "ANSI C"; no pragmas or RVV intrinsics. The built-in cost model still makes the actual vectorization decisions.*

Findings: In Kyber and Dilithium (Reference & BoringSSL code), Clang can vectorize Barrett reduction, Montgomery reduction, and even a plain remainder (into constant-time shifts and adds). Typically 6 to 10 instructions.

Autovectorization Example

BoringSSL's 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.

```
// kyber.c:177 function scalar_ntt(): uint16_t
   odd = reduce(step_root * s->c[j + offset]);
da8:   vl2re16.v     v8,(a0)
dac:   vsetvli       a1,zero,e32,m4,ta,ma
db0:   vzext.vf2     v12,v8
db4:   vmul.vx       v8,v12,s7
db8:   vsetvli       zero,zero,e64,m8,ta,ma
dbc:   vzext.vf2     v16,v8
dc0:   vmul.vx       v16,v16,a6
dc4:   vsetvli       zero,zero,e32,m4,ta,ma
dc8:   vnsrl.wi      v12,v16,24
dcc:   vmadd.vx      v12,t0,v8
dd0:   vsetvli       zero,zero,e16,m2,ta,mu
dd4:   vnsrl.wi      v8,v12,0
dd8:   vadd.vx       v10,v8,s6a   … … …
```

Instruction counts certainly drop..

BoringSSL	Non-vector ISA	+ Bitmanip ext.	+ Vectors	+ Keccak Insn.
Kyber-768	rv64gc	+_zbb	+v_zbb_zvbb	**+ "zvkeccak"**
Key Generation	667,117	567,987	383,341	**193,853**
(Parse PK +) Encaps	876,970	747,789	432,228	**194,179**
Decapsulate	765,651	713,088	294,174	**214,501**
Dilithium3	rv64gc	+_zbb	+v_zbb_zvbb	**+ "zvkeccak"**
Key Generation	2,385,466	1,882,970	1,441,127	**613,974**
Signing	12,076,148	10,377,617	6,195,554	**4,824,263**
Signature Verify	2,749,363	2,221,302	1,650,091	**884,683**

```
--varch=vlen:256,elen:64  Flags: -Ofast -DNDEBUG
```

One possible NTT Acceleration Extension

Proposal 2: 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, hardwired reduction is possible.

-> Impact: Up to 50% increase 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? Same, but modular multiply-add?

Conclusions

- **Vector Crypto** (ratified in 2023) has been picked up in main Application-class processor profiles and in RISC-V Android.
- **New**: High Assurance (HAC) and Post-Quantum (PQC) task groups.
- **HAC** is working on a full-rounds AES for side-channel security goals, and also key management / "key hiding."
- **PQC** is doing quantitative analysis with FIPS standards for Kyber, Dilithium, and SPHINCS+. Vectors and bitmanip already help a lot.
- **Keccak** instruction seems like a winner, giving significant speedups for all PQC algorithms. NTT acceleration may also be considered.

Thank You!

Questions?