PQC on Microchips: **Processors, Secure Elements, and SoCs**

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

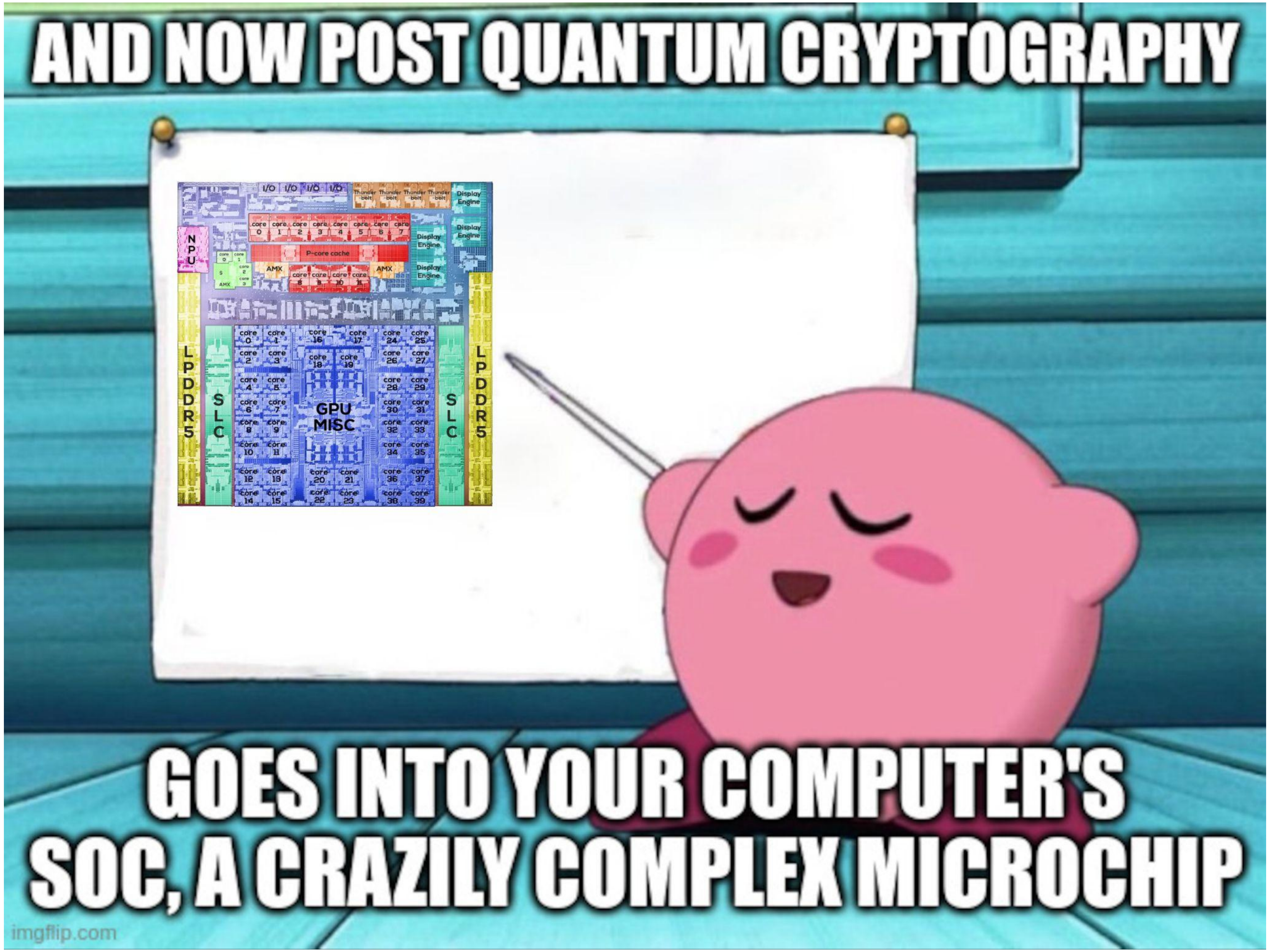
Paris October 11, 2024
IHP / Deployment of PQC Workshop

Tampereen yliopisto
Tampere University

Hello! I'm Markku-Juhani O. Saarinen 👋

Some random biographical things:

- Started as a cryptographer in 1997 at SSH Communications Security. Helped create SSH2.
- Moved to technical consulting (mainly in the Middle East) + Pentest gigs, PCI DSS audits.
- Got bored & went back to school. PhD Royal Holloway 2009 (hash function cryptanalysis.)
- Post-doc periods and more industry gigs followed. Post-Quantum stuff since ~2015.
- First employee at PQShield Ltd, Oxford UK in 2018. Architected, tinkered, prototyped, patented, and helped license hardware PQC modules to semiconductor companies.
- RISC-V Stuff since 2019. I designed some of the now-standard crypto instructions.
- Chair, **RISC-V International** Post-Quantum Cryptography Task Group (**RVI PQC TG**).
- Drifted back to **Finland** in 2023-24, now a **Professor of Practice** at **Tampere University**.
- Program Co-Chair, **PQCrypto 2025** (Taipei April 8-10, 2025): Submit Papers by **October 25**!


AND NOW POST QUANTUM CRYPTOGRAPHY
GOES INTO YOUR COMPUTER'S SOC, A CRAZILY COMPLEX MICROCHIP
I/O
Thunderbolt
Display Engine
core
P-core cache
AMX
NPU
LPDDR5
SLC
GPU MISC
imgflip.com

Hardware View: Crypto functions on a SoC

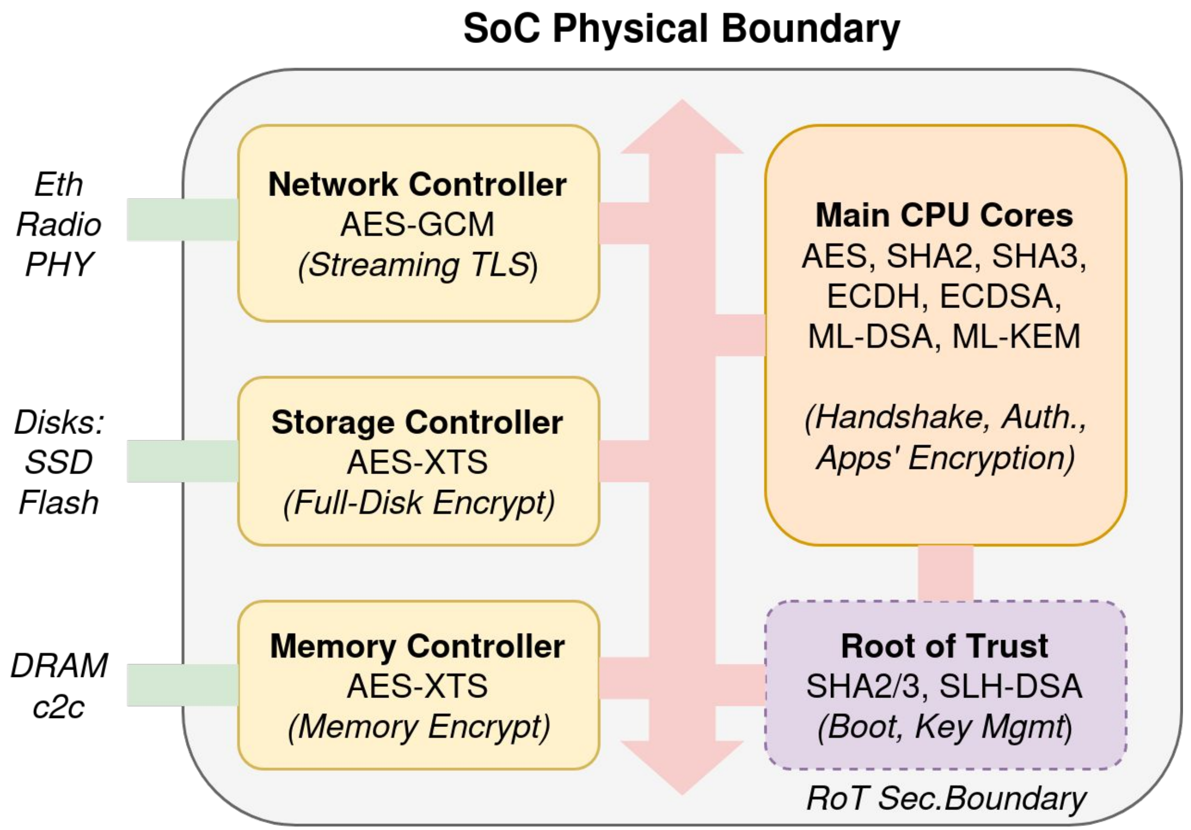
We often want to ***not use*** the main CPU for cryptography tasks..

Main CPU complex: TLS handshakes; asymmetric ops, X.509, crypto in apps. Can have ISA extensions up in the CPU.

Root of Trust (RoT): SoC-wide Platform Security. Isolated MCU + accelerators.

Disk or storage controller: E.g. AES-XTS.

Network Controller: Bulk symmetric encryption (e.g. TLS AES-GCM Frames).



Current Focus: NIST PQC Standards

KEY ESTABLISHMENT

Kyber: FIPS 203 ML-KEM (2024)

Primary PQC **key establishment** algorithm to replace Diffie-Hellman (ECDH) key exchange and RSA public-key encryption. Lattice-based.

Some of **{ HQC, BIKE**, **Classic McEliece }** (2025?)

Being evaluated in "Round 4." Code-based key establishment algorithms. Longer public keys.

Hybrid schemes: One still needs to support traditional Elliptic Curve and RSA methods.

DIGITAL SIGNATURES

Dilithium: FIPS 204 ML-DSA (2024)

Primary "general-purpose" PQC **signature algorithm** to replace ECDSA, RSA signatures. Lattice-based.

XMSS and **LMS**: NIST SP 800-208 (2020)

SPHINCS+: FIPS 205 SLH-DSA (2024)

Hash-based signatures; Firmware signing.

Falcon: FIPS 206 FN-DSA (2025). Lattice-based.

Signature "On-Ramp" Algorithms (2026?)

Helicopter View: Approx CPU and RoT Roles

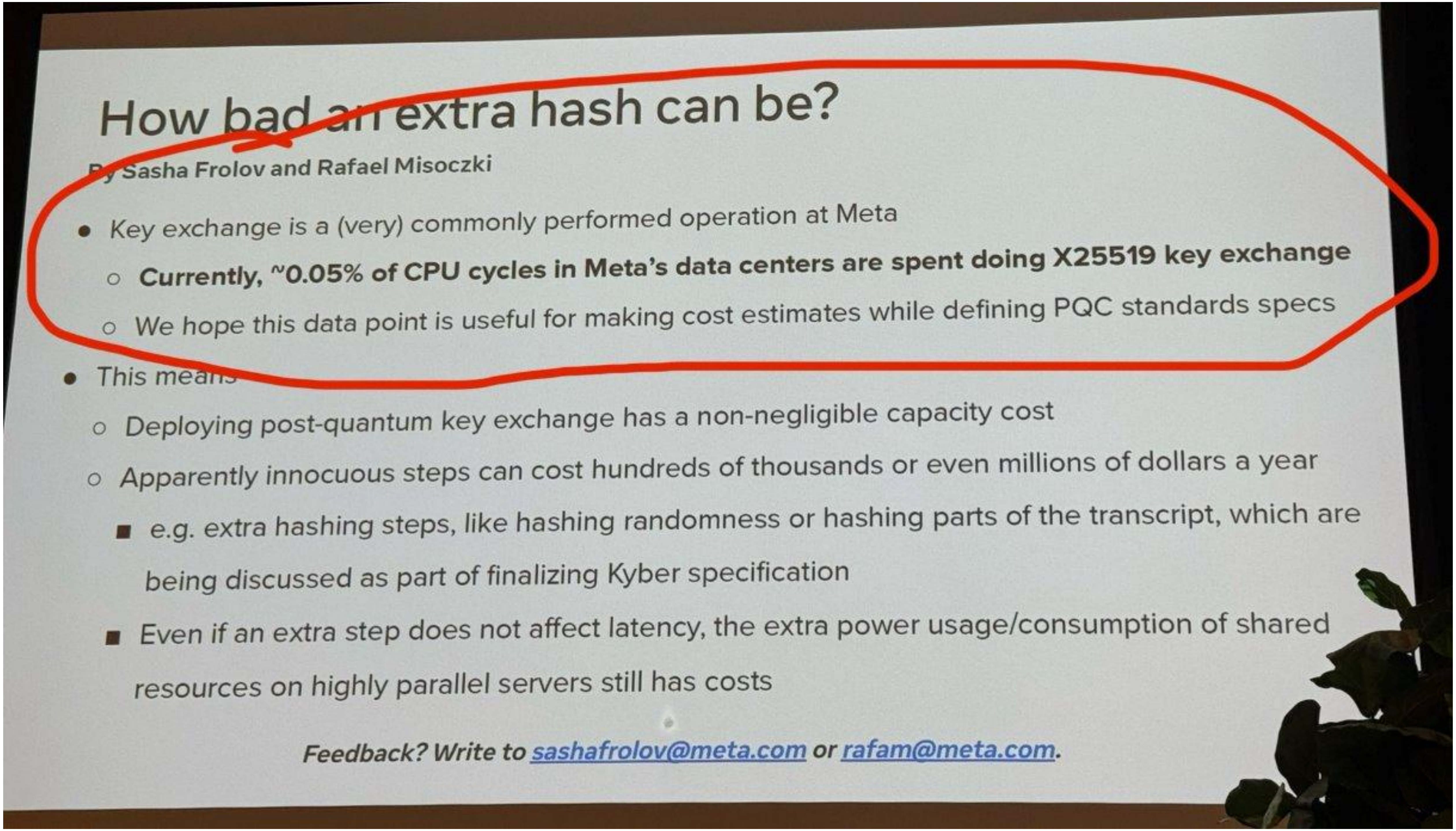
- **Boot process** can use Dilithium, but **hash-based signatures** XMSS/LMS (SP 800-208) or SPHINCS+ (FIPS 205) are also often recommended.

  Boot verification processing is often performed by the **Root of Trust** unit.

- **TLS** (or **QUIC**, **IPSEC**, **SSH**) key exchange latency affects user experience and overall power profile. Both **Kyber** and **Dilithium** will be used here (KEX+Auth.)

  This processing is usually done by the **Application Processor** units.

- **Good news**: New lattice-based PQC algorithms are usually faster or roughly same speed as classical crypto. *But any speedup is welcome.*



How bad an extra hash can be?

By Sasha Frolov and Rafael Misoczki

- Key exchange is a (very) commonly performed operation at Meta
  - **Currently, ~0.05% of CPU cycles in Meta's data centers are spent doing X25519 key exchange**
  - We hope this data point is useful for making cost estimates while defining PQC standards specs
- This means
  - Deploying post-quantum key exchange has a non-negligible capacity cost
  - Apparently innocuous steps can cost hundreds of thousands or even millions of dollars a year
    - e.g. extra hashing steps, like hashing randomness or hashing parts of the transcript, which are being discussed as part of finalizing Kyber specification
    - Even if an extra step does not affect latency, the extra power usage/consumption of shared resources on highly parallel servers still has costs

Feedback? Write to sashafrolov@meta.com or rafam@meta.com.

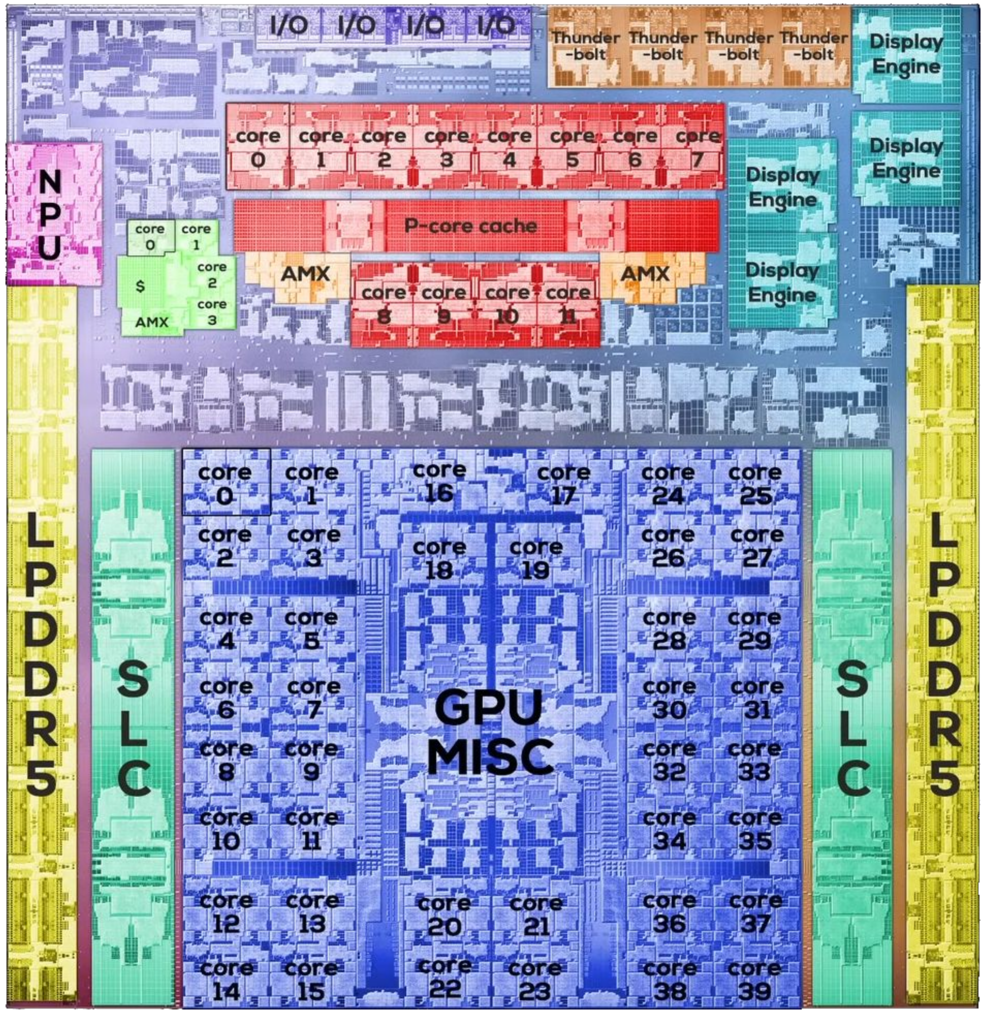
Part 1: Application Processor PQC Support

These cores run the "visible" OS:

- In Kernel: IPSec, WireShark
- Sometimes also storage encryption
- Standard apps and libraries: TLS, QUIC
- OS tools, services: SSH, GnuPG
- User applications: Signal, WhatsApp, ..

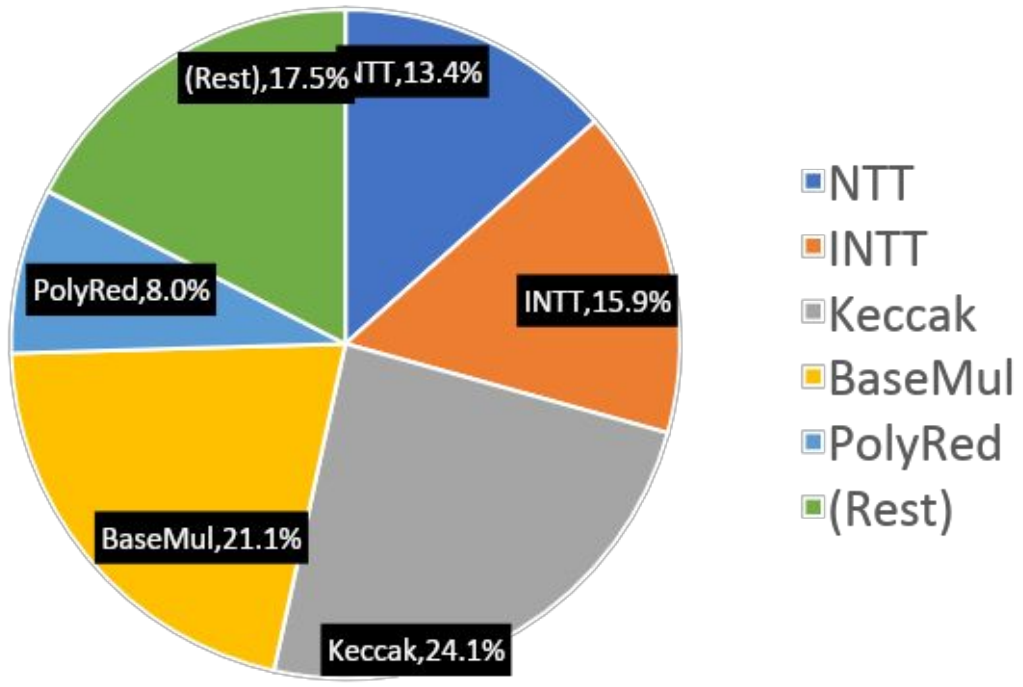
Crypto acceleration mostly done with instruction set features (for scalability).

Typically timing attack protection **only**.



Kyber Compute: Vectors (mod 3329) + Keccak

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



Compute in NIST Lattice Crypto:

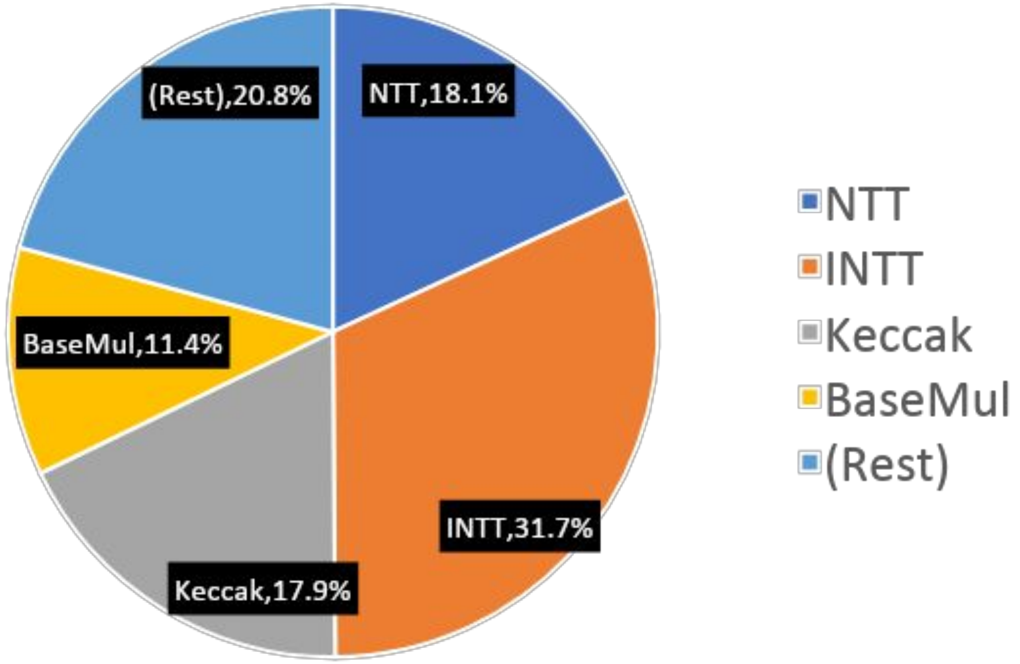
- **Keccak** i.e. SHA3/SHAKE operations. Typically well >50% of overall cycles.
- **Number Theoretic Transforms.** Vectorizable functions (256 x 16/32.)
- **Other polynomial arithmetic.** Mostly integer vectors; shifts, adds, sub.
- **Samplers** (rejection and CBD), rounding, "packing" (serialize).

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

Dilithium: Vectors (mod 8380417) + Keccak

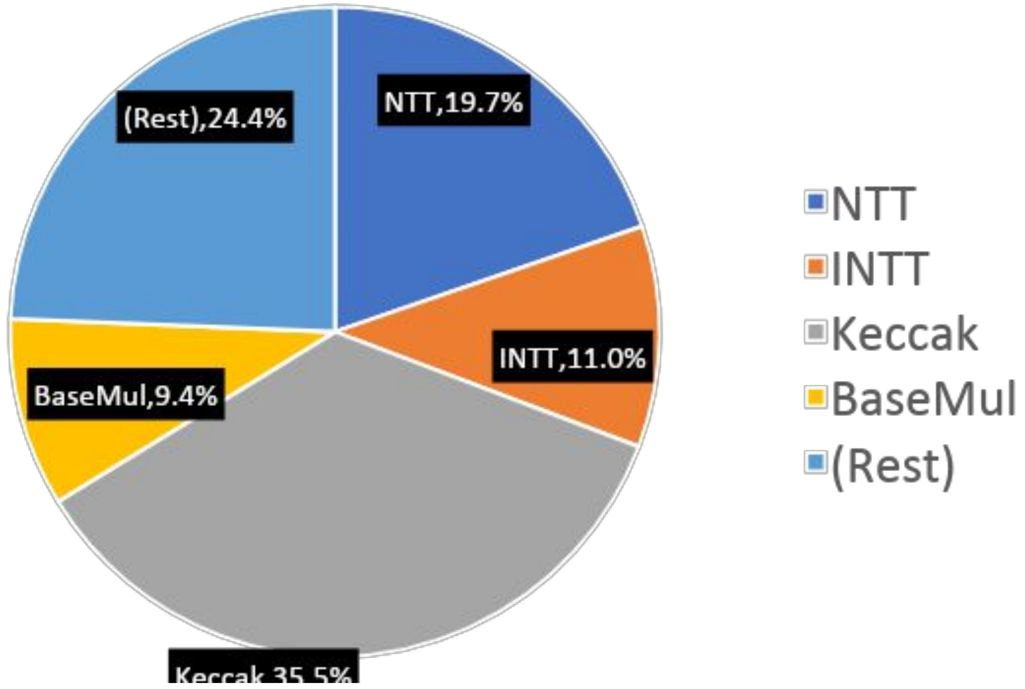
ML-DSA-44 Sign: Avg 4.60M Insn

ML-DSA-87 Sign: Avg 8.37M Insn



ML-DSA-44 Verify: 1.16M Insn

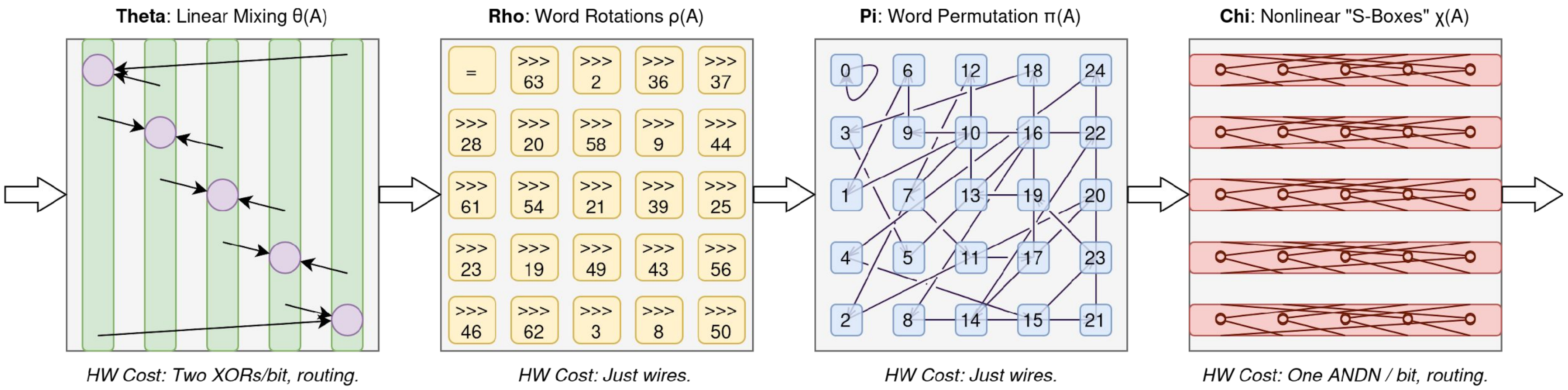
ML-DSA-87 Verify: 3.09M Insn



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

Keccak f1600: The core of SHA3 & SHAKE

- **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, 1600-bit state**. Relatively slow in software but very fast in hardware (but rather large area.) A lot of XORs and NOT-AND gates.



Main PQC TG Proposal: **A Keccak Instruction**

Keccak state is little awkward to fit into vector 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.

(Ed. note: Sorry about jargon & acronyms, this slide was made for RISC-V Summit!!)

Optimizing Kyber & Dilithium with Vector/SIMD

Application-class RISC-V processors have **vector instructions** available, similarly to AVX SIMD on Intel and NEON, SVA on ARM architectures.

We optimized Kyber and Dilithium with **RISC-V Vector Intrinsics** / CLANG 20.

Benchmarked with SpacemiT X60 (VLEN=256) & C908 (VLEN=128) silicon:

- Vector really helps (~5x speedup) with arithmetic parts (NTT) and somewhat with the bit packing and sampling too.
- Vector does **not** help SHA3/SHAKE much -- that becomes a bottleneck.

Impact: Kyber-768 (ML-KEM) Key Exchange

	KeyGen()	Encaps()	Decaps()	TOTAL	Speedup
RV64GC	663,067	815,357	1,006,469	2,484,893	**1.00**
RV64GCV+ZBB	546,631	685,201	858,508	2,090,340	**1.19**
w. Intrinsics	223,400	239,714	262,241	725,355	**3.43**
w. Keccak Insn	49,363	61,632	84,120	195,115	**12.74**

Clang 20.0.0git -O3 with –march=rv64gc / rv64gcv_zbb_zvl256b

Lines 1-3 uses C language reference Keccak f1600; 4,038 instructions.

Line 4 uses SPIKE 1 cycle Keccak. In real-life in hardware ~100 cycles.

Keccak Instruction: Microarchitecture Notes

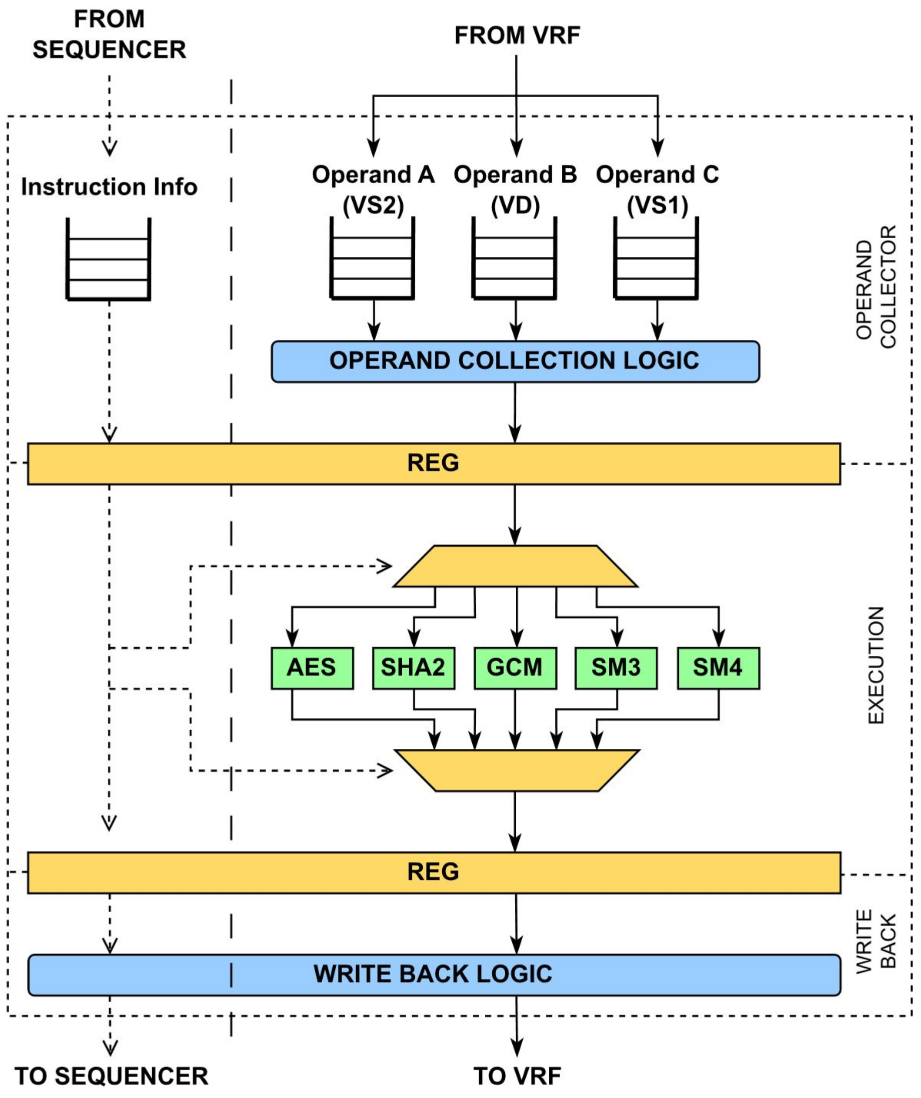
In **"Marian"** we extended the PULP Ara2 vector unit with Zvk crypto instructions.

We used a 256-bit "operand collector" because of the VRF structure. VRF is split into lanes. Each lane only has 64-bit segments of each reg; need to combine.

Keccak would need a 1600-bit collector.

But would probably still be worthwhile!

https://github.com/soc-hub-fi/Marian



PQC Support: RISC-V PQC TG Recommendation

Keccak instruction seems like a winner, giving significant speedups for all PQC algorithms. NTT can also be considered, but RVV already helps a lot with that.

This instruction is replacing thousands of instructions. Core f1600 permutation is 24 cycles. Together with operand collection + writeback can still be under 100.

Hardware note: Permutation alone is about 40kGE + "operand collector" logic.

PQC speedup of Keccak Insn. on RVV ~2-4x. Quite easy to integrate into software.

Microarchitecturally awkward but saves device battery / $$$ in data centre.

Timing Attacks – ISA Support

Some Greatest Hits (in asymmetric crypto TA) Along the Years:

- P.C. Kocher: *"Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems."* (CRYPTO 1996. Target: RSAREF 2.0 running on MS-DOS.)
- D. Brumley and D. Boneh: *"Remote timing attacks are practical."* (USENIX Security 2003. OpenSSL RSA remote key recovery, CVE-2003-0147.)
- B. Brumley and N. Toveri: *"Remote Timing Attacks Are Still Practical."* (ESORICS 2011. OpenSSL ECDSA remote key recovery, CVE-2011-1945.)
- Q. Guo, T. Johansson. A. Nilsson, *"A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM."* (Crypto 2020, PC Oracle, demoed against a claimed const-time impl.)

Every generation gets to learn the special implementation tricks!

Basic Sources of Timing Leaks
(That are avoidable with careful programming)

1. Secret-controlled branches and loops:

```
if <secret> then { delay1(); } else { delay2(); }
```

2. Memory accesses (cache timing attacks). Can be a load or store.

```
ct = SBox[pt ^ key];        // observe latency with different inputs.
```

3. Arithmetic operations whose processing time just depends on inputs

```
x = y % q;        // division and remainder ops are rarely constant-time.
```

When Hiring a (Post-Quantum) Crypto Dev..

Constant-time coding / algorithm knowledge is fundamental

- Have a “library” of solid CT replacements for **`memcmp()`** and similar functions.

- Identify conditionals, transform to straight-line code using Boolean operations? 🤔
  `x = s ? a : b;` *vs.* `x = b ^ ((-(s & 1)) & (a ^ b));`

- Table-lookups: Bit-slicing (entire thing as a Boolean circuit), “full scan / collect”.

- No division instructions in modular arithmetic (use Montgomery, Barrett etc.)

- Know how to test with symbolic execution (e.g. valgrind) or on instruction level..

.. etc .. these are core crypto programming skills!

Analyze source code, Verify Implementation

RISC-V “Data Independent Execution Latency” (DIEL)

- One can use e.g. **valgrind** or other static or dynamic analysis tools to validate constant-time properties. (Secret variables need to be “annotated.”)

- Compilers don’t know about “constant time” – need to verify machine code.

- Note: *`Constant-timeness’’* of **Intel** and **ARM**: mostly from experiments.

- RISC-V CETG codified timing as the **Zkt** extension for scalar, and the **Zvkt** DIEL (Data Independent Extension Latency) instruction list for vector.
  Included in the manual: https://github.com/riscv/riscv-isa-manual/releases

Part 2: The “Invisible” Chip Cryptography

- Vendors (Intel, AMD, Apple, Google, NVIDIA, Qualcomm, Google, … but also RISC-V system makers) need to be **able to update their system chips**.

- This creates ***incredible supply chain risks*** – think of a rootkit or malware in a microcode update to an Intel CPU. This is completely invisible to OS & users.

- Ecosystems (Android, Windows, Apple OS) and device vendors want to protect their devices against “jailbreaking” and unauthorized modification.

- **Consequence**: Much more serious measures are taken to protect the chips and devices *themselves* than any user application running on them. 🤷

Evaluating Attacks against **PQC Modules**

- **High assurance** level (EUCC: equivalent to AVA_VAN.3 or above) is a requirement for Root of Trust IP, Smart Cards, Secure elements, many types of IoT (SESIP).
- While FIPS testing focuses on "checklist compliance", AVA_VAN checks real-life security via a "penetration test."
- ISO 17825 / FIPS 140-3 "non-invasive" leakage assessment ignores many practical physical attacks (e.g. FIA).

"Evaluators must have knowledge and experience of [..] side channel attacks (SCA) such as Timing Analysis,
Machine Learning based SCA,
Simple Power Analysis (SPA),
Differential Power Analysis (DPA),
Differential EM radiation Analysis (DEMA),
Template Attacks (TA);
fault injection attacks such as DFA [..]"

– **EUCC v1.1.1** (also SOG-IS Documents)

Protection Profiles for Chip Security

AVA_VAN.3+ is a common requirement for Root of Trust and Security IC products.
We assume that this will not change (much) with Post-Quantum Cryptography.

[JSADEN011] **“SESIP Profile for PSA Certified™ Level 3”**
Root of Trust (PSA-RoT): 35 person-days of AVA_VAN.3 activities.

[PP-0084] **“Security IC Platform Protection Profile”**
EAL 4 augmented by AVA_VAN.5 and ALC_DVS.2

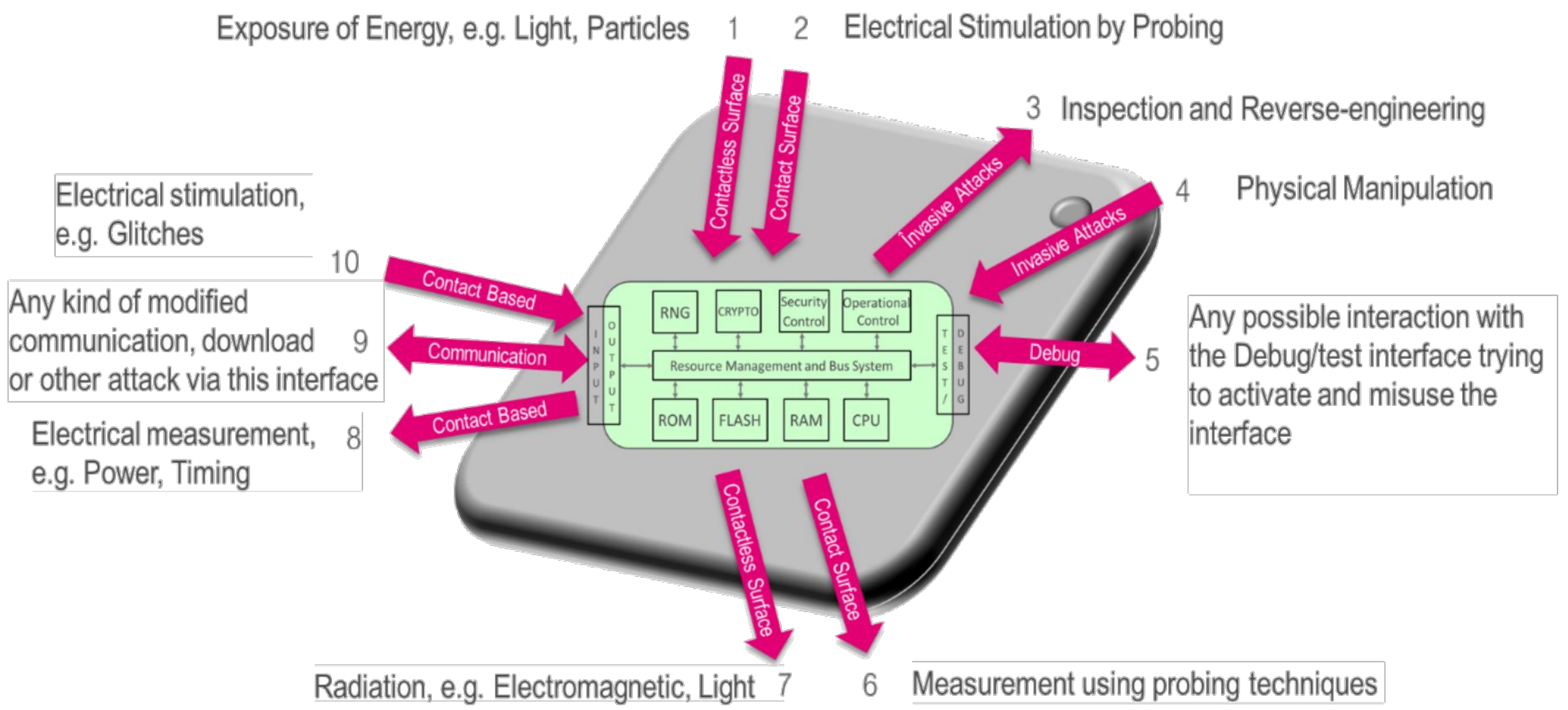
[PP-0117] **“Secure Sub-System in System-on-Chip (3S in SoC)”**
EAL 4 augmented by ATE_DPT.2, AVA_VAN.5, ALC_DVS.2 and ALC_FLR.2

3.2 Threats

The threats described in this section are applicable to the base Protection Profile. For threats related to functional extensions see Chapter 7.

The following figure describes the attacks that are applicable to the TOE. The interactions related to the attacks are marked with red arrows.

(From PP-0117)



Exposure of Energy, e.g. Light, Particles 1
2 Electrical Stimulation by Probing
3 Inspection and Reverse-engineering
4 Physical Manipulation
Contactless Surface
Contact Surface
Invasive Attacks
Invasive Attacks
Electrical stimulation, e.g. Glitches 10
Contact Based
Any kind of modified communication, download or other attack via this interface 9
Communication
Electrical measurement, e.g. Power, Timing 8
Contact Based
INPUT OUTPUT
RNG
CRYPTO
Security Control
Operational Control
Resource Management and Bus System
ROM
FLASH
RAM
CPU
TEST / DEBUG
Debug
5 Any possible interaction with the Debug/test interface trying to activate and misuse the interface
Contactless Surface
Contact Surface
Radiation, e.g. Electromagnetic, Light 7
6 Measurement using probing techniques

https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Reporte/ReportePP/pp0117V2b_pdf.pdf

AVA_VAN: Common Criteria Vulnerability Analysis

Attack Potential is evaluated with a score-based system that roughly maps to the "**cost of attack**." (think $ or €)

Considers attack **Identification** + **exploitation**, with many factors:

- Elapsed time (hours–months)
- Attacker Expertise (multiple)
- Knowledge (how restricted)
- Access to the TOE (samples)
- Equipment (common/bespoke)

("Application of Attack Potential" docs.)

AVA_VAN.1 Vulnerability Survey
- TOE resistance against BASIC Attack Potential (0-15)

AVA_VAN.2 (Unstructured) Vuln. Analysis
- TOE resistance against BASIC Attack Potential (16-20)

AVA_VAN.3 Focused (Unstructured) Vuln. Analysis
- TOE resistance against ENHANCED-BASIC AP (21-24)

AVA_VAN.4 Methodical Vuln. Analysis
- TOE resistance against MODERATE AP (25-30)

AVA_VAN.5 Advanced Methodical Vuln. Analysis
- TOE resistance against HIGH Attack Potential (31-)

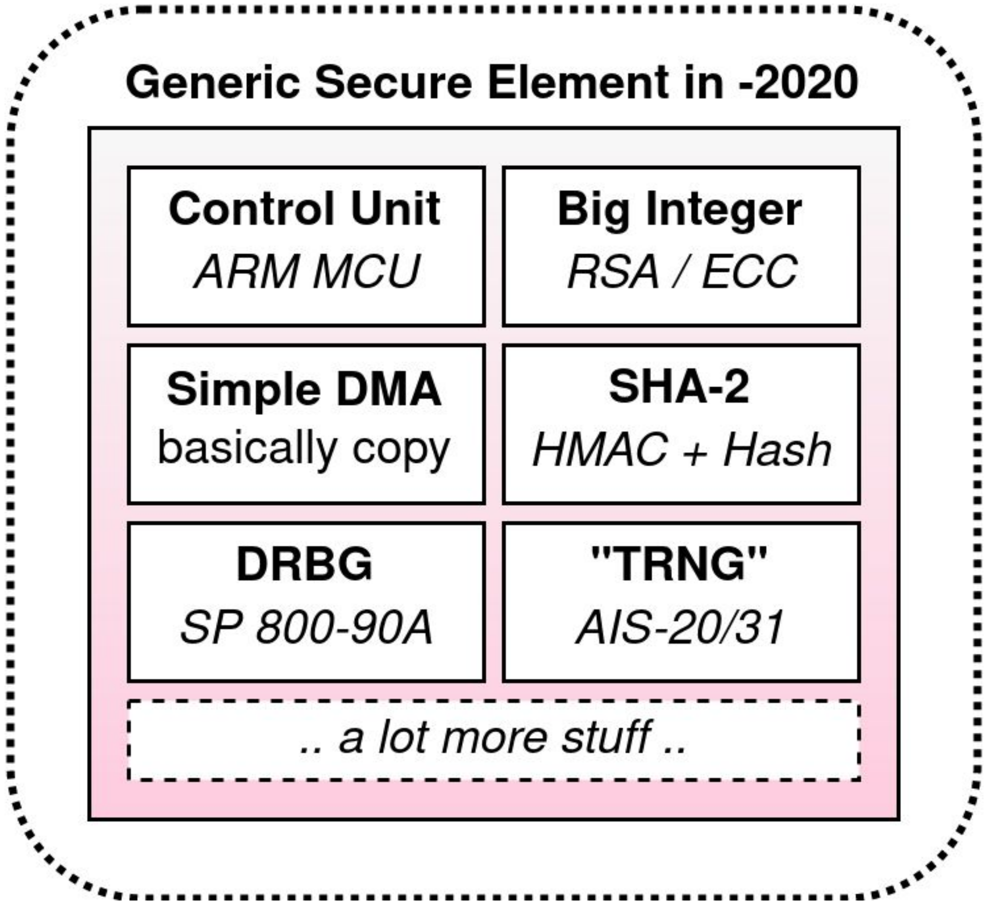
Attack Potential: Example Calculation

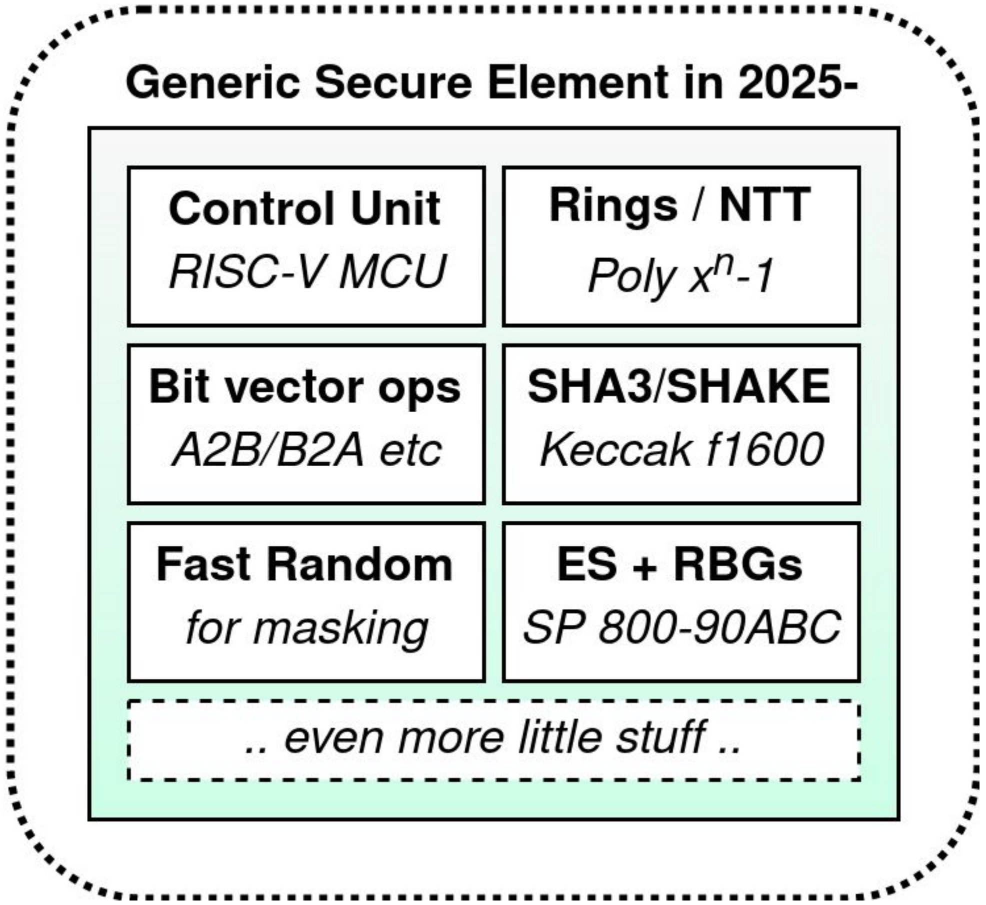
AP Component	Identification	Exploitation
Elapsed time	2 (< one week)	6 (< one month)
Expertise	5 (expert)	4 (expert)
Knowledge of the TOE	4 (sensitive)	0 (public)
Access to the TOE	0 (< 10 samples)	0 (< 10 samples)
Equipment	3 (specialized)	4 (specialized)
Open Samples	0 (public)	0 (public)
Total	28 (AVA_VAN.4 / moderate AP range)	

SOG-IS: "Application of Attack Potential to Smartcards and Similar Devices"

Inherited Application Requirements

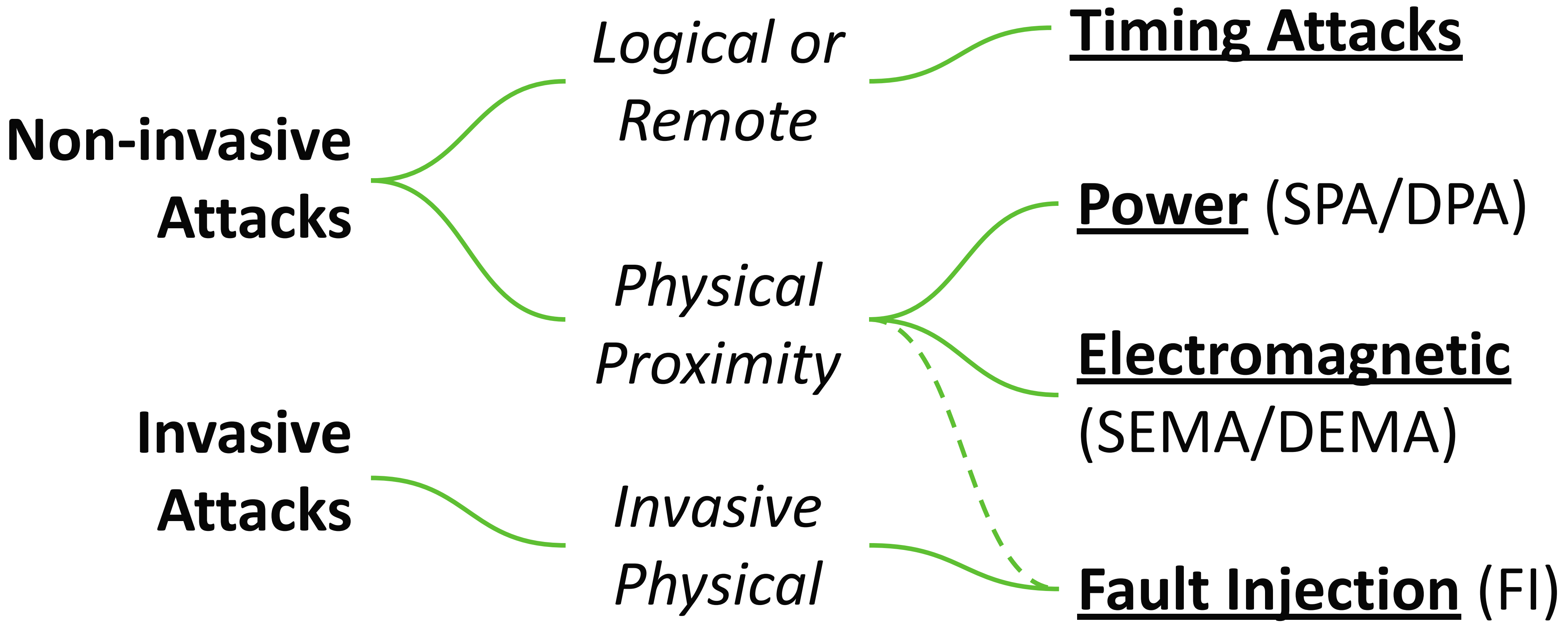
Platform Security / RoT, Smart Cards, Authentication Tokens, etc





Side-Channel and Fault Attacks

Different Applications have Different Requirements


Non-invasive Attacks
Invasive Attacks
Logical or Remote
Physical Proximity
Invasive Physical
Timing Attacks
Power (SPA/DPA)
Electromagnetic (SEMA/DEMA)
Fault Injection (FI)