Post-Quantum

Cryptography Conference

# Implementing the Post-Quantum Survivors: A Retrospective

**David Hook**
VP Software Engineering at Keyfactor

KEYFACTOR CRYPTO4A SSL.com ENTRUST HID

**October 28 - 30, 2025 - Kuala Lumpur, Malaysia**



PKI Consortium

KEYFACTOR

# Implementing the Post-Quantum Survivors: A Retrospective

David Hook,
Legion of the Bouncy Castle/Keyfactor

# The Project

- Provide implementations of the round 3 finalists and alternate candidates. Implementations required for Java and C#.
- Project supported by the Australian Cyber Security Centre (ACSC) with additional support for the BC team from Keyfactor.
- Additional funding made available to have the implementations independently reviewed.

# The Plan


- Some algorithms were already in progress with the BC team.
- The remaining algorithms were outsourced to research groups at two Australian Universities – Monash University and University of Wollongong.
- Implementation review work is being conducted by University of Adelaide.
- Legals handled by Terra Schwartz

# So how did it progress?

- Kick off for external algorithm implementers was in mid-February 2022.
- Kick off for review work July 2022.
- First two external algorithms were completed at the end of June.
- BC team set of algorithms finished at the start of July.

# Implementation Issues

- Not everything is fully covered in the existing specifications.
- Reference implementations need to be used for resolving ambiguities.
- Generation of KAT data used for implementation testing sometimes complicated by introduction of machine generated randomness.
- The KAT random is a bit “odd”


KEYFACTOR

# New Features in the BC APIs

- KEM API:
  - EncapsulatedSecretGenerator – “secret encryption” step.
  - EncapsulatedSecretExtractor – “secret extraction” step.
- Support for use of hybrid cryptography
  - Use of suppPrivateKeyInfo (SP 800-56B)
  - Use of secret concatenation (SP 800-56C)
- Composite Keys, Composite Signatures
- External Keys
- Common approach taken in C# and Java low-level API

# Adaptation to the JCA

- Signatures as expected.
- KEM constructs are a little odd in JCE terms.
- KEMs provide for Key Wrapping
- KEMs can also act as generators for secrets.
- KEMs are a bit “Cipher” and a bit “KeyGenerator”.
- Supported both as it prevented any ambiguity of purpose creeping in.

KEYFACTOR

# PKI Additions

- These are both IETF drafts.
- Composite Keys – allows for multiple public keys to appear in an X.509 certificate
- Composite Signatures – signature “algorithm” allowing multiple signatures to appear in place.
- External Keys – Key encoding which uses hash of a SubjectPublicKeyInfo rather than the full key for a public key reference in a certificate.

# 2022: Initial Review

- Review work done at University of Adelaide
- Implementations reviewed at two levels:
  - interoperability with the standard reference implementations.
  - code analysis looking doing taint checking and side channel analysis.
- Second item was automated using a static code analyzer (prototype).
- Analysis found some examples of secret leakage due to side channels which were subsequently fixed in BC 1.73.
- Static analyzer only worked with Java, C# checked by comparison.

# 2025: Review Round Two

- While almost everyone involved subsequently changed universities, research continued.
- Tool has since evolved into a full byte code analyzer.
- Current incarnation of the tool is called “Robusta”
- Primary authors: Deepak Bhargavan Pillai(1), Anirban Chakraborty(3), Chitchanok Chuengsatiansup(2), Matthew Roughan(1), Peter Schwabe(3,4), and Yuval Yarom(5)
  - Organizations: (1) The University of Adelaide, (2) Hasso Plattner Institute, (3) Max Planck Institute for Security and Privacy, (4) Radboud University, and (5) Ruhr University Bochum

KEYFACTOR

# Real Issues and False Positives

- Robusta team contacted Bouncy Castle in July 2025.
- Initial report identified 57 possible issues.
- Byte code analyzer able to provide details to the line number in the original source.
- Use of Byte Code analysis has meant new rendition of tool a lot more thorough than the original work commissioned in 2022.
- We're still going through the report, majority of items are false positives, however tool has also found some timing related issues which were previously missed.
- Tool currently lacks the ability to suppress false positives from the report.

# Some Reflections on Progress

- Taint checking is hard to do based simply on references without context. Simple inputs and DRBG's can get flagged.
- For constant time analysis, in some cases a simple branch might be flagged where the circumstances are such that it's not a “real issue”.
- That said, new tool output appears to show that an issue in the category of interest will no longer get through undetected.
- Real timing analysis shows that in many cases the byte code “tells” are replicated in the JIT.
- In some cases, what appeared to be a false positive in the report, turned out to be a real issue when real life sampling was done.
- Likewise it is important to rerun the tool after “fixing” something. The results are not always what you might expect or hope for.

# Initial Conclusions

- While the current fixes we have found do improve the constant time characteristics of the APIs, we haven't been able to suggest an exploit path, nor is the tool able to do so.
- That said, we are following the idea, “if it is not in there, it cannot be exploited”.
- That said, new tool output appears to show that an issue in the categories of interest will no longer get through undetected.
- Real timing analysis shows that in many cases the byte code “tells” are replicated in the JIT.
- In some cases, what appeared to be a false positive in the report, turned out to be a real issue when real life sampling was done.
- This is not a journey that will end anytime soon!

# Thanks for listening.

# Questions?

PKI Consortium

PQ Secure

# PQC Formal Verification: Challenges and Tools for Formal Verification of Post-Quantum Cryptography”

Reza Azarderakhsh

CEO PQSecure

Kuala Lumpur, Malaysia
October 2025

# PQC Standardization

PQ Secure

## NIST Standardized PQC Schemes

CNSA 2.0 Schemes

| Key Exchange | Digital Signature |
|---|---|
| ML-KEM FIPS-203 | ML-DSA FIPS-204 |
| | SLH-DSA FIPS-205 |

## Finalists for Upcoming Next PQC NIST Standardization (by 2026)

| Key Exchange | Digital Signature |
|---|---|
| HQC-KEM FIPS-207 | FN-DSA FIPS-206 |

NIST National Institute of Standards and Technology

## On-ramp Digital Signatures (2nd Round)

| | |
|---|---|
| CROSS | LESS |
| SQIsign | HAWK |
| Mirath | MQOM |
| PERK | RYDE |
| SDitH | MAYO |
| QR-UOV | SNOVA |
| UOV | FAEST |


Public-key
ML-KEM-1024
ML-DSA-87

Symmetric-key
AES-256
SHA-384/SHA-512
SHA3-384/SHA3-512

Software and Firmware Updates
XMSS
LMS

NIST SP 800-208: Recommendation for Stateful Hash-Based Signature Schemes (XMSS/LMS)
NIST SP 800-227: Recommendation for Key-Encapsulation Mechanisims

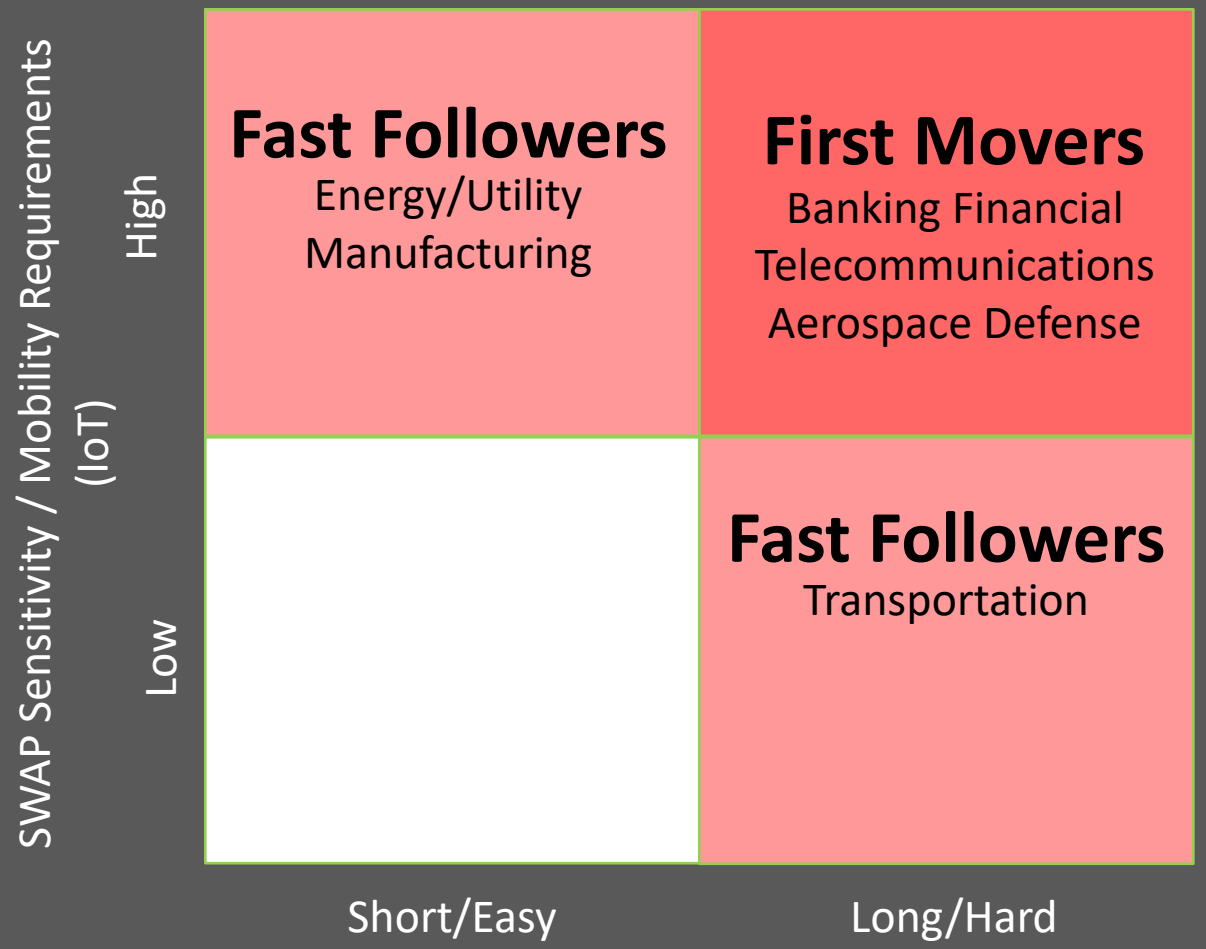
# PQC Agility

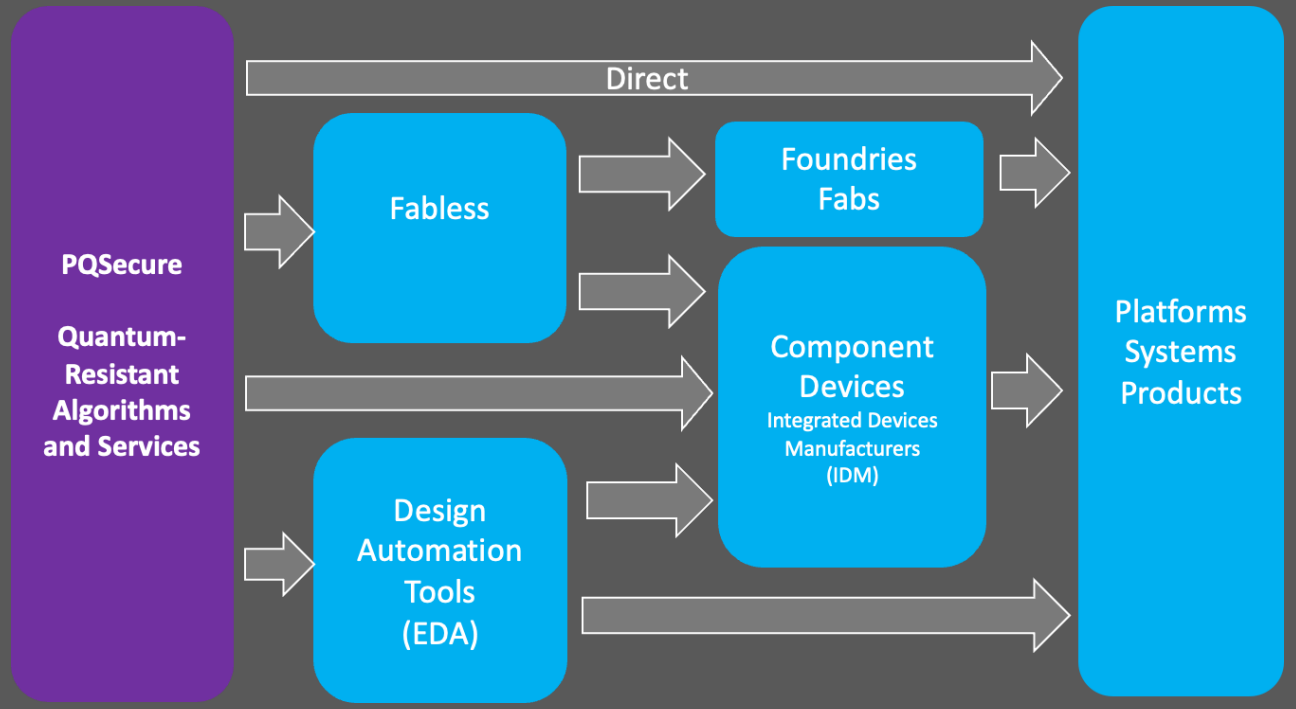
PQ Secure

| Type of Agility | Definition |
|---|---|
| **Implementation** | The Capability to swiftly configure interfaces and implement updates across various systems or applications |
| **Compliance** | The capacity to adapt cryptographic configurations in accordance with compliance requirements. |
| **Security Strength** | The capability to dynamically adjust the level of security strength based on configuration, allowing for scalable security measures. |
| **Migration** | The capability to transition and convert between cryptographic algorithms seamlessly. |
| **Retirement** | Ability to retire obsolete or insecure cryptographic algorithms |
| **Composability** | The capability to securely integrate multiple cryptographic primitives for composability. |
| **Platform** | Ability to use assured cryptographic algorithms across different platform types |
| **Context** | Ability to use a derived cryptographic algorithm policy with the flexibility from system attributes |

# PQC to Final Products

PQ Secure



# CHIPS Act: Agility and HW/SW for PQC

PQ Secure

AUGUST 09, 2022

## FACT SHEET: CHIPS and Science Act Will Lower Costs, Create Jobs, Strengthen Supply Chains, and Counter China

***Key Strategy 1.1.5: Prioritize hardware integrity and security as an element in co-design strategies across the stack.***

In the face of threats from nation-state and criminal adversaries, the potential for the insertion of malicious alterations into components ranging from circuits to software combined with the need to prepare for a post-quantum-computing world make it essential that integrity and cybersecurity be a foundational component of system design.[30,31] Co-design of hardware with software is needed to meet this challenge in a way that provides maximum protection while minimizing the impact on system performance.[32] The design process must allow for iteration between hardware, software, and security constraints. To meet economic and national security needs, security must be incorporated in co-design R&D as a design constraint at the same level as performance.

16

[30] Cybersecurity R&D challenges and goals for hardware and software are described in NITRD's *Federal Cybersecurity Research and Development Strategic Plan* , https://www.nitrd.gov/pubs/Federal-Cybersecurity-RD-Strategic-Plan-2019.pdf.

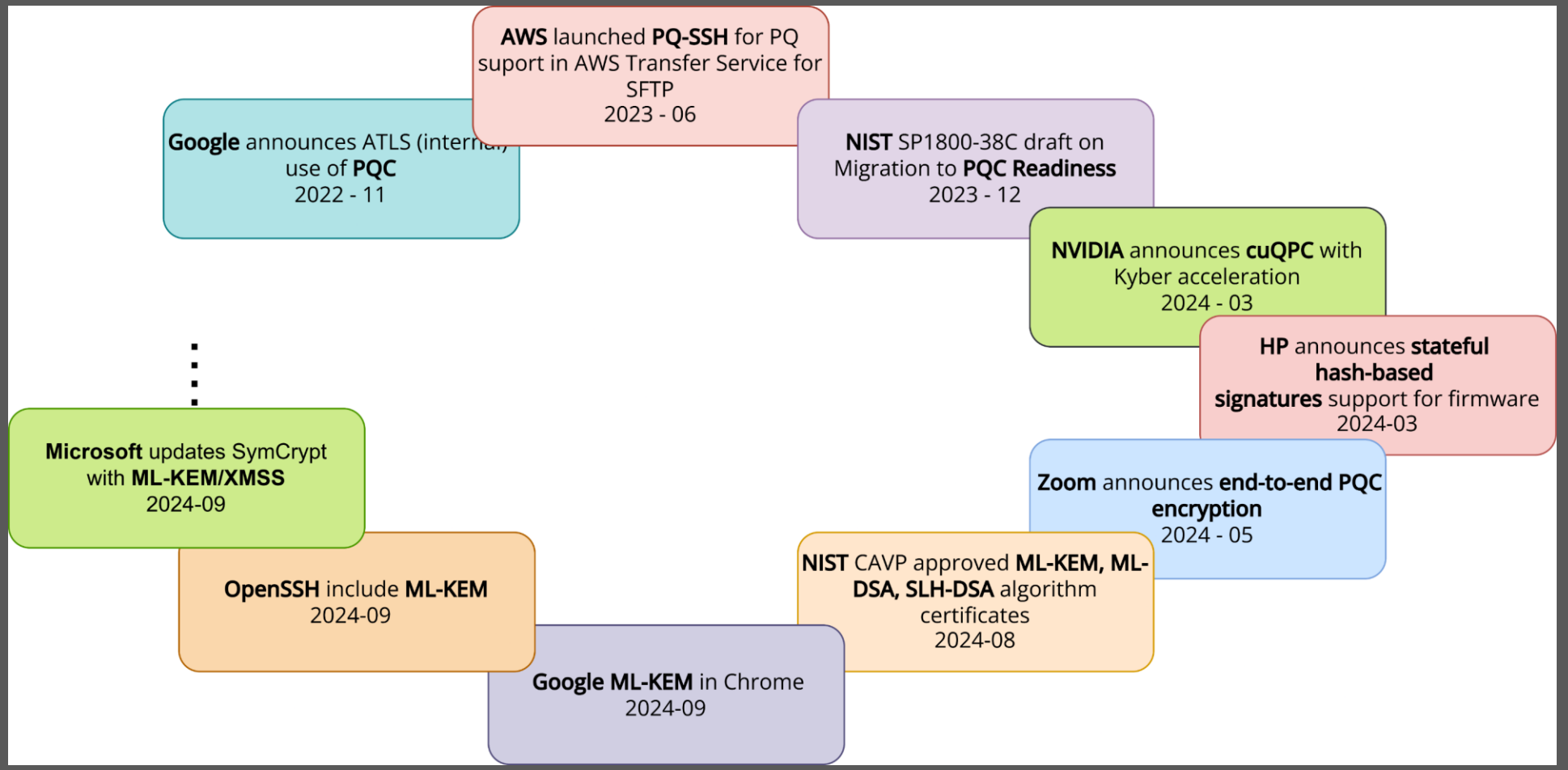
[31] https://www.whitehouse.gov/briefing-room/statements-releases/2022/05/04/national-security-memorandum-on-promoting-united-states-leadership-in-quantum-computing-while-mitigating-risks-to-vulnerable-cryptographic-systems/

[32] See, for example, D. Dangwai et al., *SoK: Opportunities for Software-Hardware-Security Codesign for Next Generation Secure Computing*, arxiv.org/abs/2105.00378.

– 12 –

# Pre-finalized PQC Standard Deployments

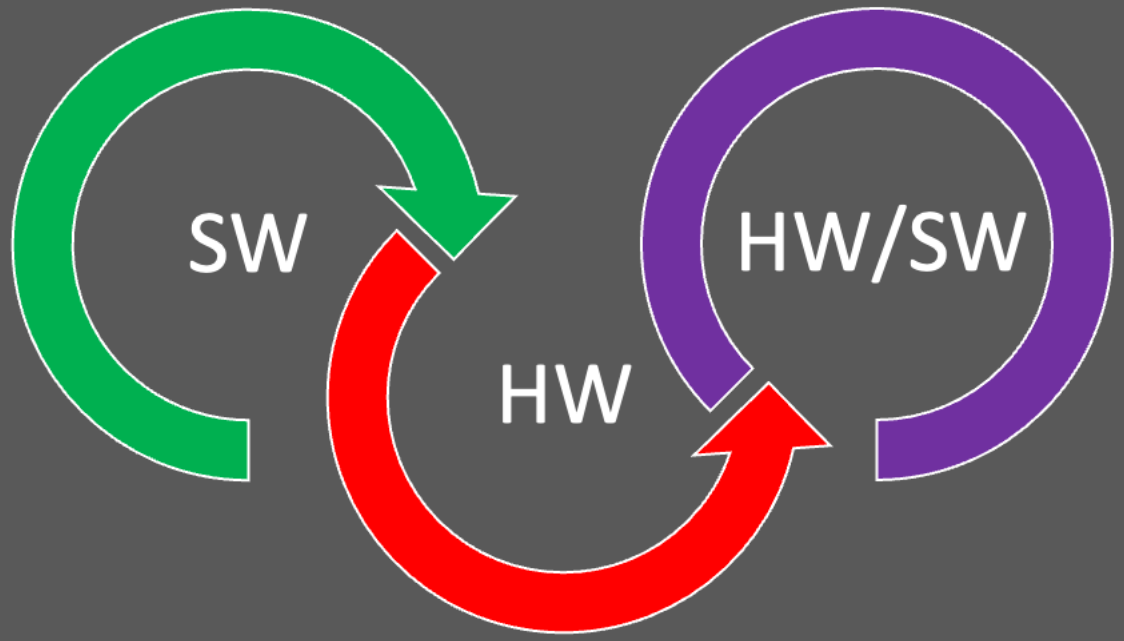
PQ Secure



# SW-First and then HW/SW Co-design

PQ Secure

- Reference Codes
- ANSI C
- Device level optimized ASM
- C-to-Rust
- Side-channel protection (?)
- Formal Verification

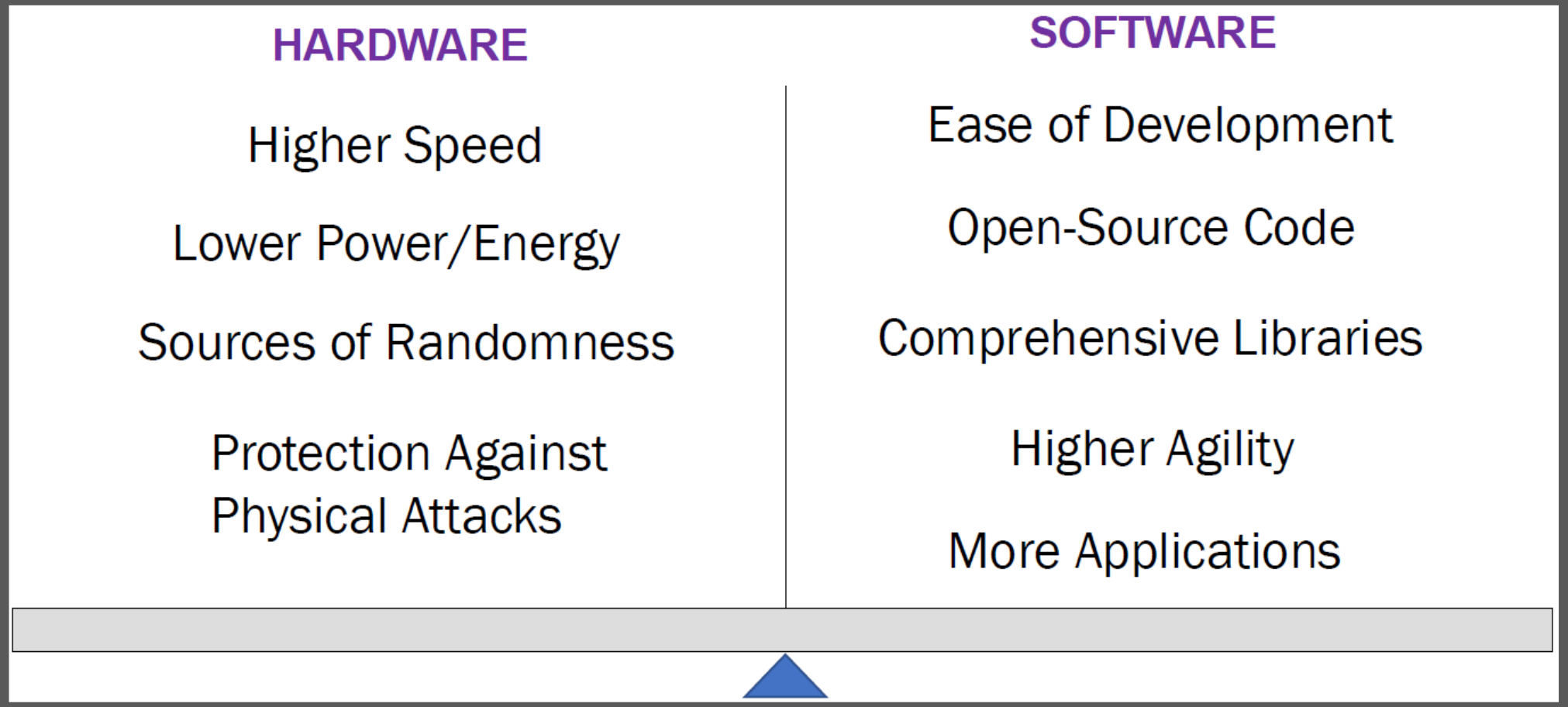



- CPU/FPGA co-design
- RISC-V/FPGA co-design
- ASIC with embedded co-design
- Side-channel protection
- Testing and Verification
- Assurance

- Reference HDL
- FPGA Implementations
- ASIC Synthesis kGE
- Side-Channel Protection
- Verification and Assurance

# HW vs. SW

PQ Secure



# Everyone Does Cryptography

PQ Secure

**Software Libraries**: Provide flexible cryptographic functionality for general use.

**Dedicated Hardware Instructions**: Enhance cryptographic operations with specialized processor commands.

**Trusted Platform Modules (TPMs)**: Secure hardware solutions that store encryption keys and perform cryptographic tasks.

**Hardware Security Modules (HSMs)**: Purpose-built hardware devices designed to protect cryptographic keys and processes.

**Secure Enclaves**: Isolated, secure environments within processors to handle sensitive data ensure secure execution.

# Not Everyone Does Good Cryptography

PQ Secure

- **Incorrect Implementations**:
  - **CVE-2022-0778 in OpenSSL** allowed denial-of-service by exploiting a flaw in certificate parsing.
- **Weak Protocols**:
  - **ROCA (2022)** affected Infineon TPMs, making RSA keys vulnerable to factorization attacks.
- **Bad Random Number Generation**:
  - **CVE-2023-23946 in GitHub Actions**, where weak random numbers led to easily guessable session tokens.
- **Information Leakage Through Side-Channels**:
  - **PLATYPUS attack (2020)** on Intel CPUs exploited power consumption data to extract cryptographic keys.

# Cryptographic Engineering

PQ Secure


- **I can do it**
  - Experienced in designing SoC with crypto accelerators.
  - Only need IP blocks from PQSecure.
- **Do it for me**
  - Limited resources or expertise, need full support.
  - Require design, integration, and verification assistance.
- **Give me a starting point**
  - Need a secure solution tailored to specific needs.
  - Require customizable IP as a base to avoid reinventing the wheel.

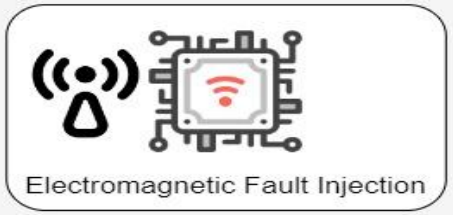
# SCA Challenges

PQ Secure

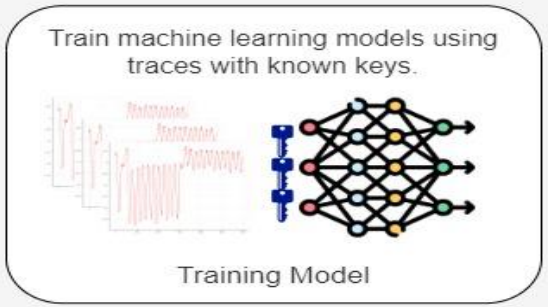
## Fault Injection Attacks

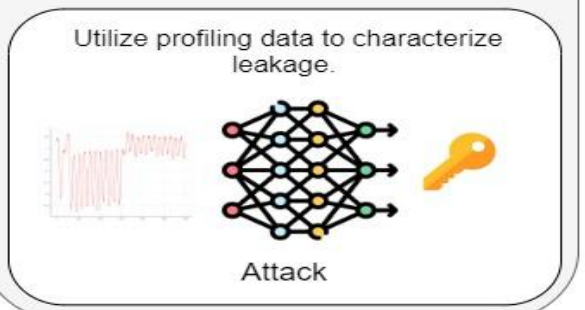
Relies on introducing physical disturbances to cause errors in the system's execution, which can then be analyzed to recover secret information.





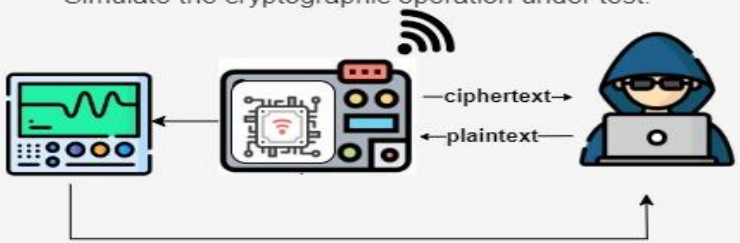

## AI-Assisted Profiling Attack

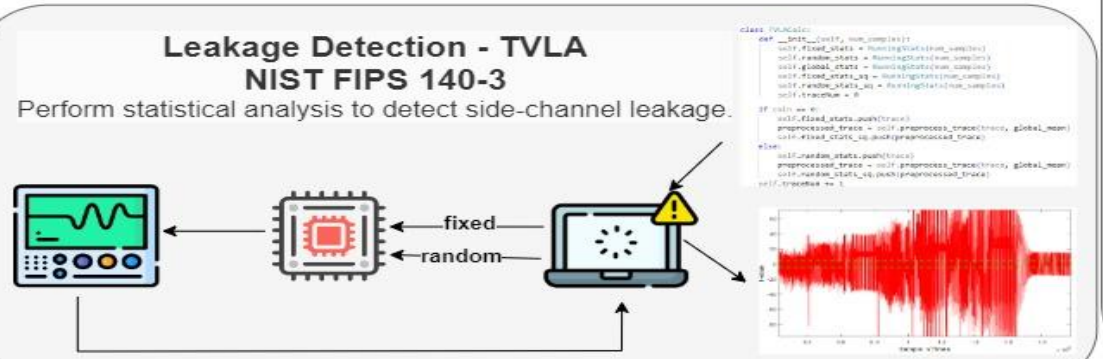

## Side-Channel Setup

Simulate the cryptographic operation under test.



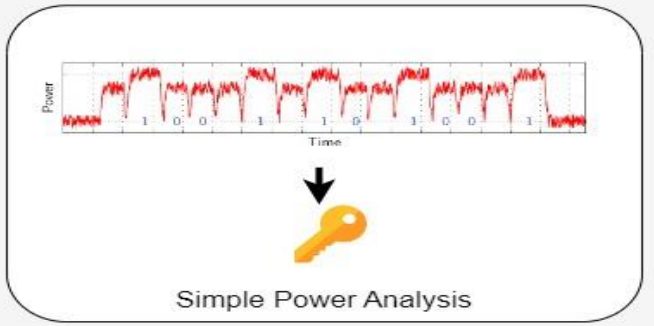

## Leakage Detection - TVLA
## NIST FIPS 140-3

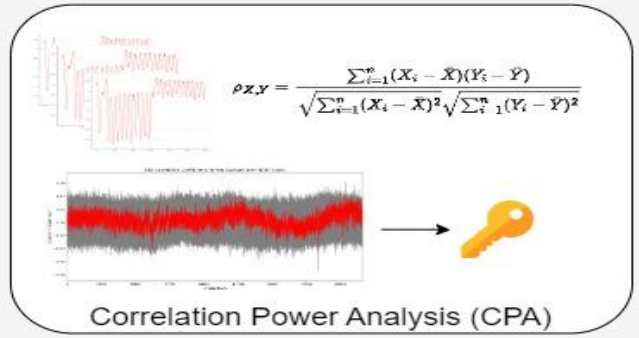
Perform statistical analysis to detect side-channel leakage.

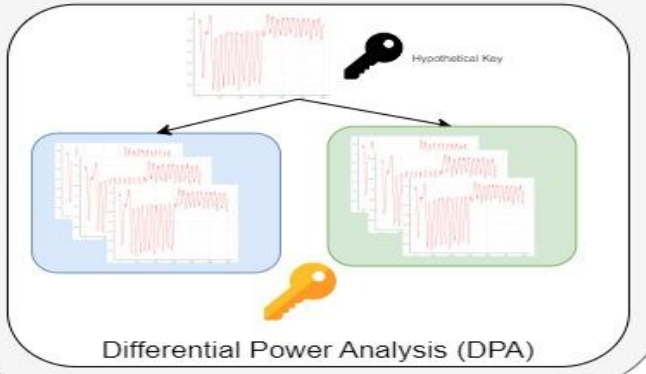



## Power Analysis Attacks

Exploit the power consumption of a device to extract information about the secret key or internal states. without prior profiling.



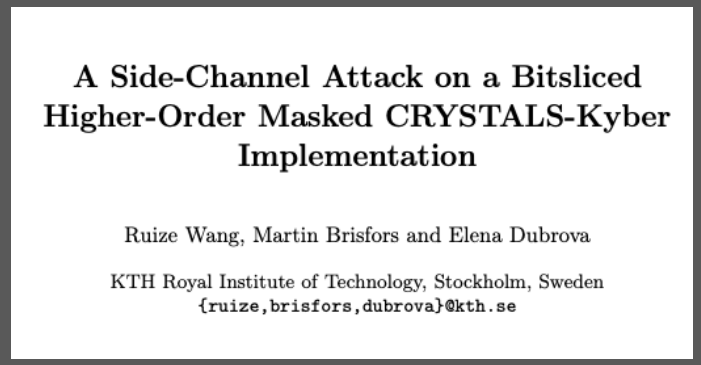



# Why Software Implementations are Vulnerable?

PQ Secure

## Side-Channel Attacks Against Software

- Even software implementations with **first-order** or **higher-order masking** have been broken.


A Side-Channel Attack on a Bitsliced Higher-Order Masked CRYSTALS-Kyber Implementation

Ruize Wang, Martin Brisfors and Elena Dubrova

KTH Royal Institute of Technology, Stockholm, Sweden
{ruize,brisfors,dubrova}@kth.se

https://eprint.iacr.org/2023/1042.pdf


A side-channel attack on a masked and shuffled software implementation of Saber

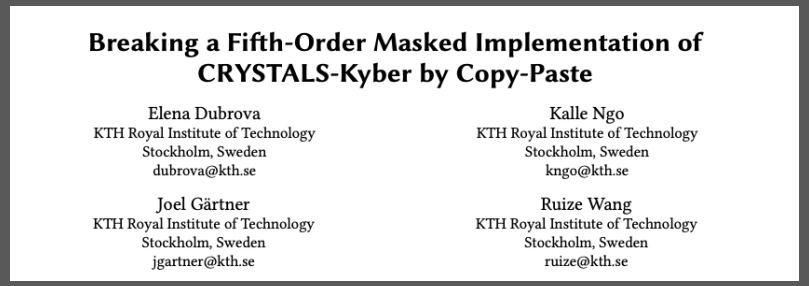
Kalle Ngo, Elena Dubrova, Thomas Johansson

LTH Profile Area: AI and Digitalization, Networks and Security, ELLIIT: the Linköping-Lund initiative on IT and mobile communication

KTH Royal Institute of Technology

*Research output: Contribution to journal › Article › peer-review*

https://portal.research.lu.se/en/publications/a-side-channel-attack-on-a-masked-and-shuffled-software-implement


Breaking a Fifth-Order Masked Implementation of CRYSTALS-Kyber by Copy-Paste

Elena Dubrova
KTH Royal Institute of Technology
Stockholm, Sweden
dubrova@kth.se

Kalle Ngo
KTH Royal Institute of Technology
Stockholm, Sweden
kngo@kth.se

Joel Gärtner
KTH Royal Institute of Technology
Stockholm, Sweden
jgartner@kth.se

Ruize Wang
KTH Royal Institute of Technology
Stockholm, Sweden
ruize@kth.se

https://dl.acm.org/doi/pdf/10.1145/3591866.3593072


Unpacking Needs Protection

A Single-Trace Secret Key Recovery Attack on Dilithium

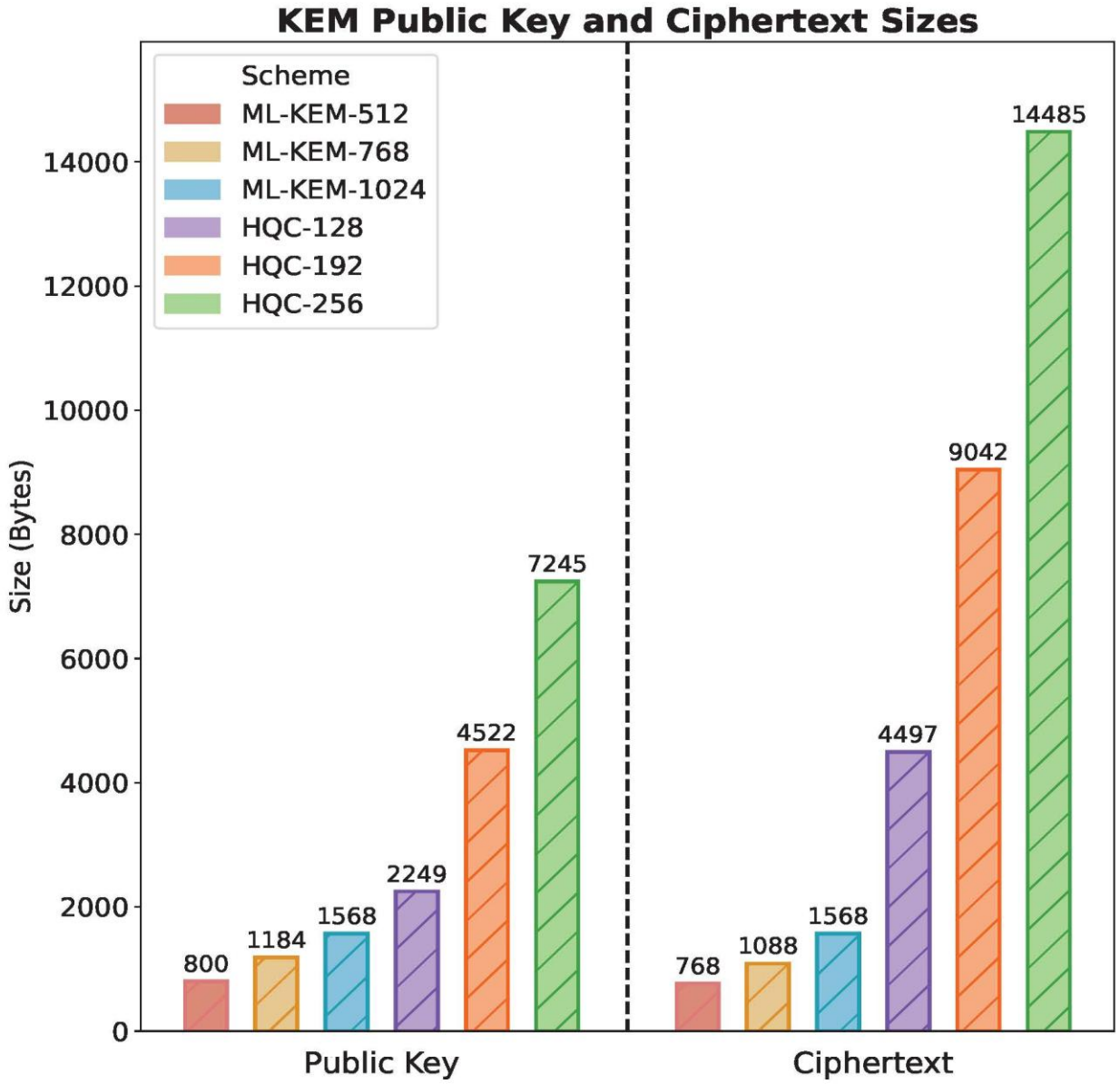
Ruize Wang, Kalle Ngo, Joel Gärtner and Elena Dubrova

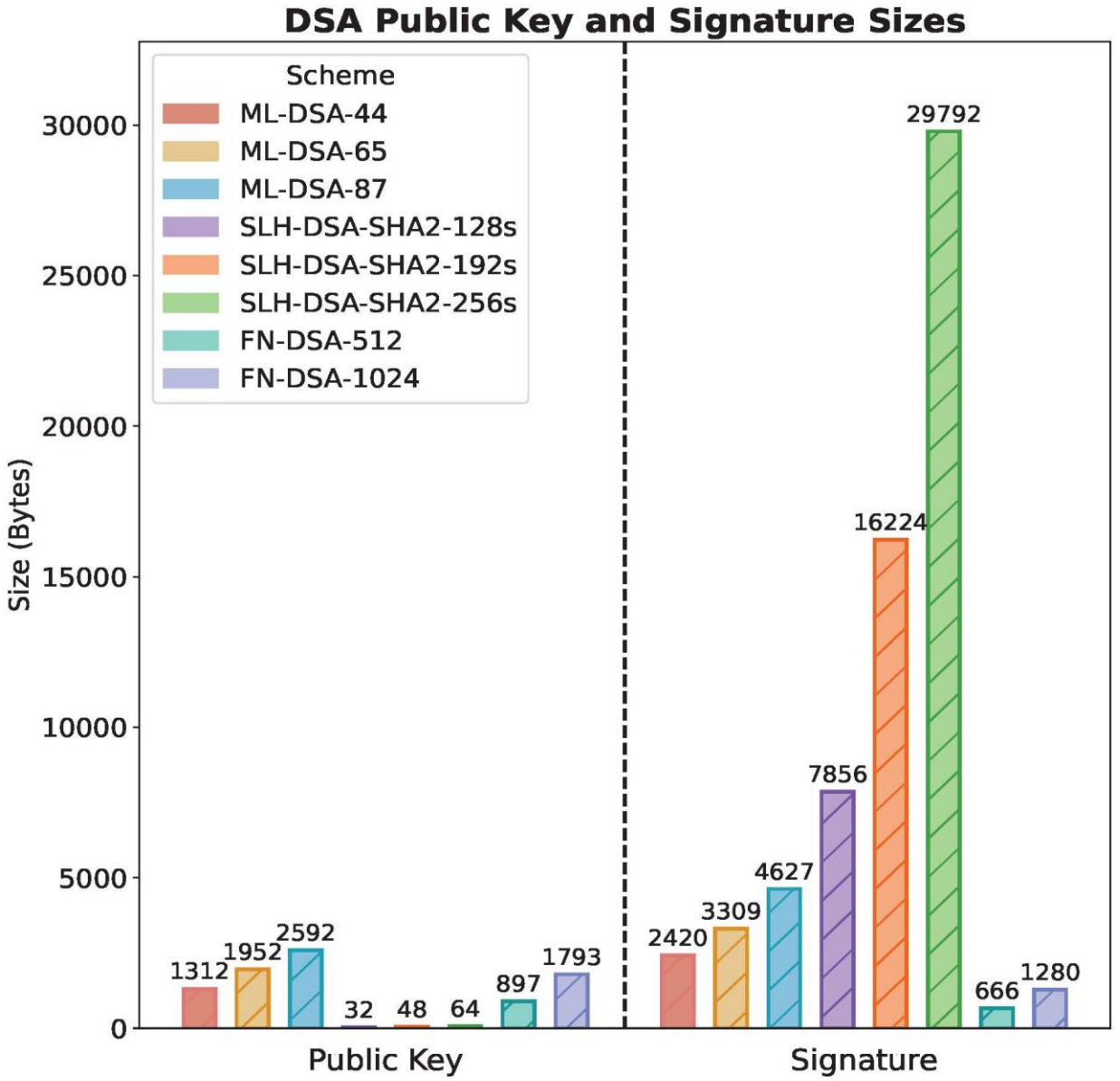
KTH Royal Institute of Technology, Stockholm, Sweden

https://cic.iacr.org/p/1/3/12/pdf

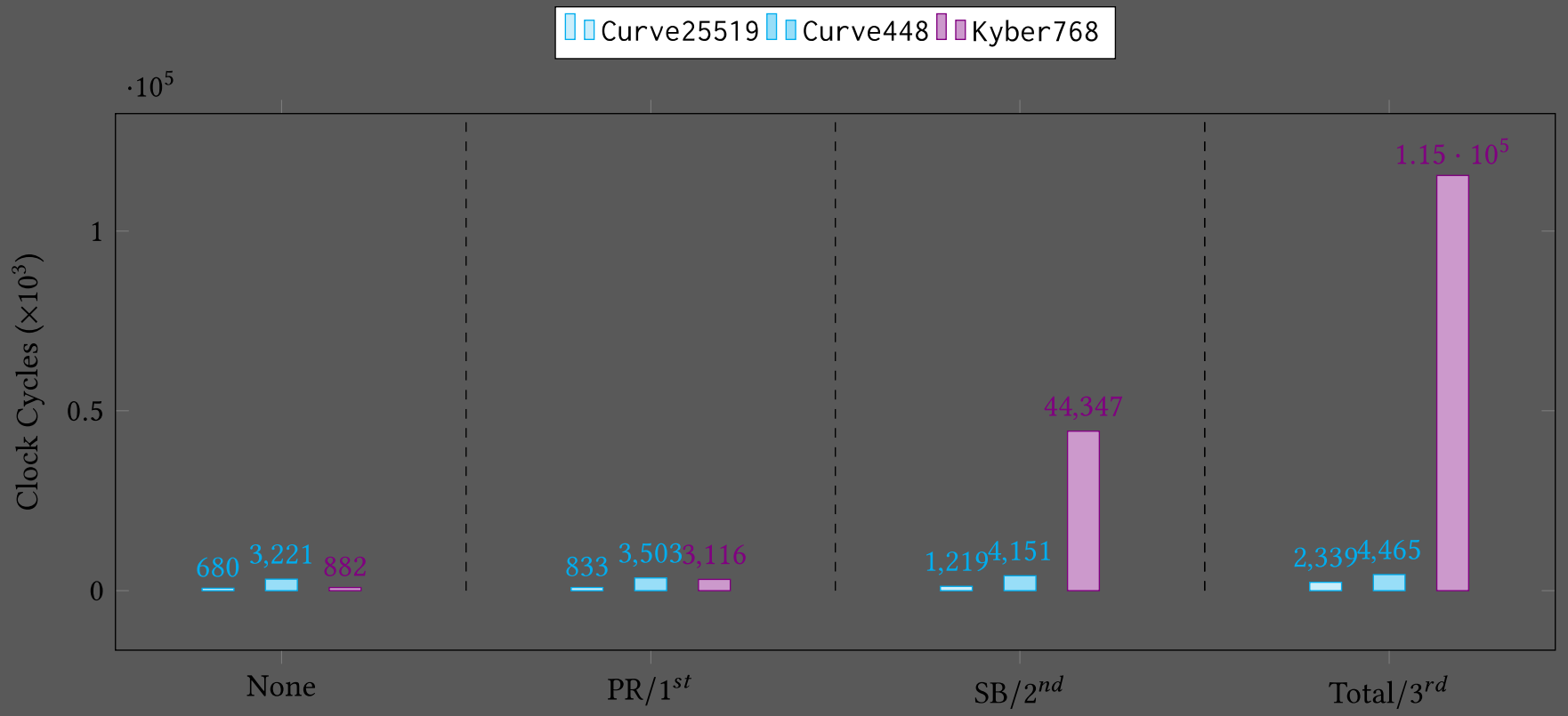
# Key sizes for NIST PQC Algorithms

PQ Secure



# SW SCA for Kyber vs. ECC




- **Kyber needs stronger SCA solutions.**
- **Masking of 2nd order adds 19x and 10x more performance overhead than Curve25519 and Curve448, respectively.**
- **Masking of 3rd order adds 26x and 49x more performance overhead than Curve25519 and Curve448, respectively.**

# Many Stages for PQC Migration

PQ Secure

Perform risk assessment

Explore PQC algorithms

Study algorithms for the users

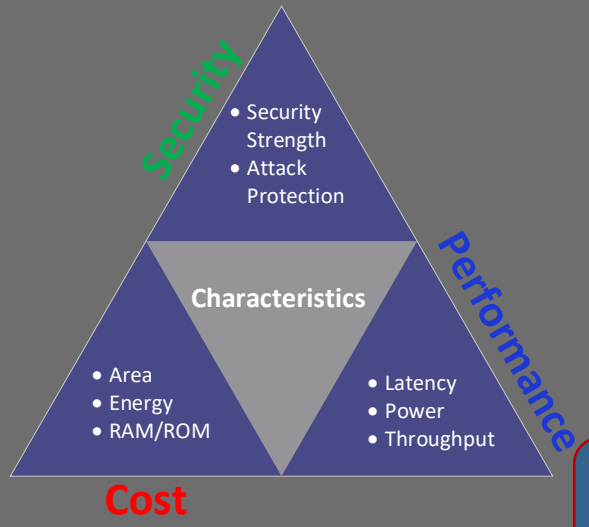
Study implementations on real HW

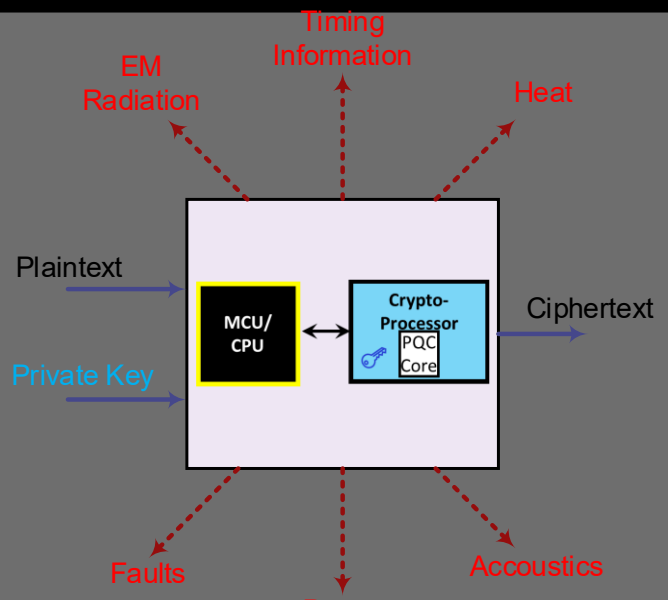
Study Physical security attacks SCA, FI, etc.

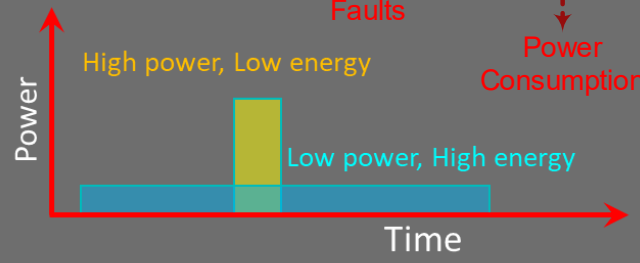
Focus on deployment for target performance

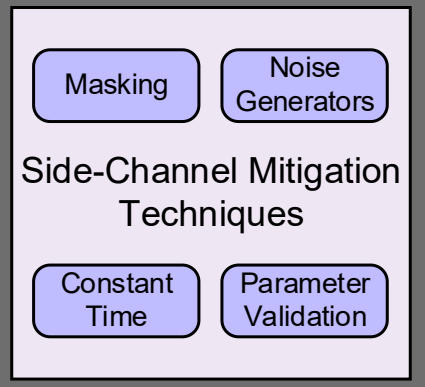
Integrate securely on read-world apps

Study reliable and high assurance



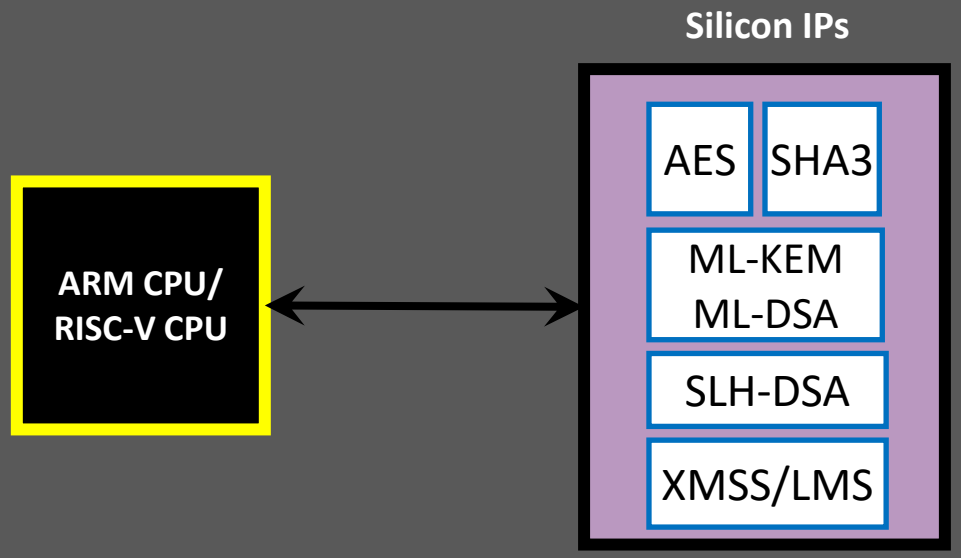

**FIPS 140-3 "non-invasive SCA"**

Implementing PQC primitives is more complex than implementing ECC/RSA, and the community has many years fewer experience with what could go wrong.

# Agile PQC Design Choices

PQ Secure

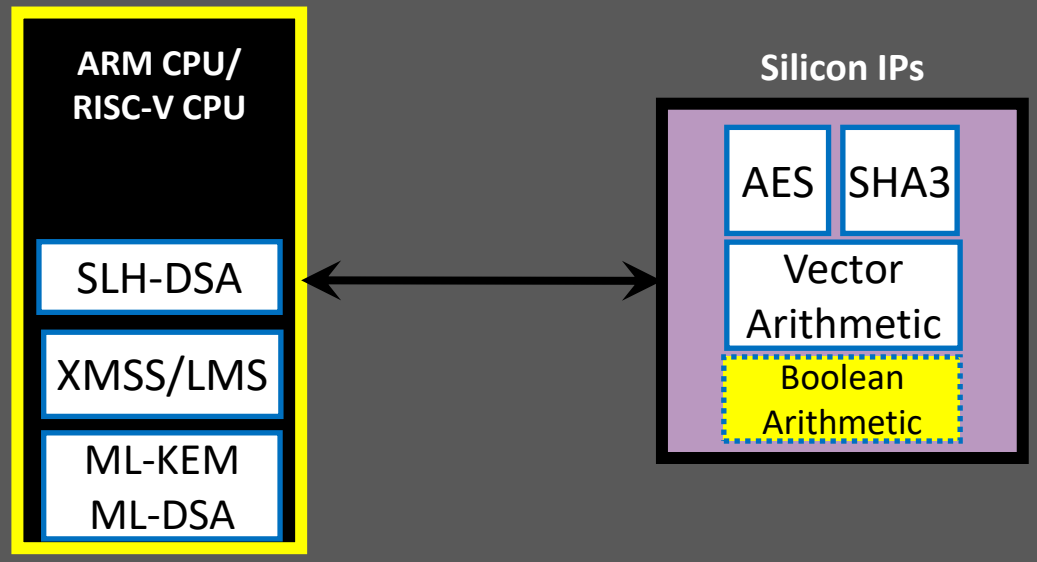
## Pure HW Design




- ✓ Not scalable
- ✓ Dedicated for SWaP-C
- ✓ Very Fast
- ✓ Very Efficient
- ✓ Complete CPU off-load
- ✓ Low power/energy



## HW/SW Co-Design




- ✓ Scalable
- ✓ Flexible
- ✓ Allows OTA updates
- ✓ Fast
- ✓ Extra Silicon for RSIC-V or
- ✓ Utilization of host MCU

# Formal Verification

PQ Secure

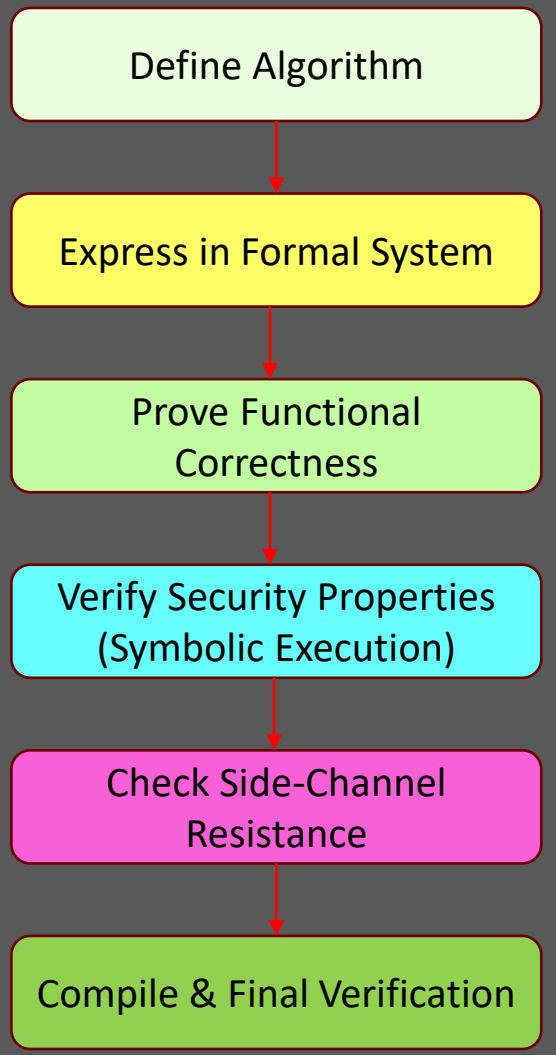
- Consider the field multiplication over $\mathbb{F}p$ with $p$ = 2^255 − 19.
  - Number of inputs: 2^255 × 2^255
  - How many of them can be tested?
  - What about those inputs which are never tested?

- Formal verification aims to prove the absence of bugs through logical or mathematical reasoning.
  - Mathematical proof of correctness *for all possible inputs.*
  - Complements testing by providing exhaustive confidence.

# What is Formal Verification?

PQ Secure

- Mathematical proof-based approach
- Ensures cryptographic implementations behave as intended
- Proves correctness across all possible inputs
- Why is Formal Verification Critical for PQC?
  - PQC is new and complex – traditional testing is insufficient
  - Verification prevents future vulnerabilities
  - Meets compliance standards (FIPS 140-3)



# Why Formal Verification Tools Needed for PQC?

PQ Secure

- Ensuring correctness in PQC algorithms requires rigorous verification *beyond* traditional testing
- PQC algorithms too complex for manual proofs → automation through formal tools **necessary**
- Verifies correctness, memory safety, and protocol security *before* deployment
- Key Areas of PQC Verification:
  - Mathematical Proof Verification → Theorem Proving
  - Protocol-level Security Checking → Model Checking
  - Software Execution Analysis → Symbolic Execution
  - Ensuring Compiler Security → PCC

# What Tools Are Available for PQC?

PQ Secure

| Category | Tools Used | Key Focus |
|---|---|---|
| Theorem Proving | Coq, Isabelle/HOL, EasyCrypt | Proving mathematical correctness of cryptographic algorithms |
| Model Checking | Tamarin, ProVerif | Verifying protocol security (e.g., PQC key exchange) |
| Symbolic Execution | SAW, KLEE, Angr | Checking functional correctness and memory safety |
| Compiler & Low-level Execution | Jasmin, CompCert, LLVM Verified Backend | Ensuring compiled cryptographic implementations remain secure and correct |

# Formal Verification: Programming Languages

PQ Secure

- C has been standard language for writing crypto implementations
- Reminder: classical crypto standards should NOT apply to PQC
- So, should we still use C for PQC algorithms?
- C is not ideal for PQC:
  - Lack of built-in memory safety
  - Unpredictable optimizations by compilers
  - NO built-in formal verification

"Security engineers have been fighting with C compilers for years"

--Simon, Chisnall, Anderson, 2018

"We argue that we must stop fighting the compiler, and instead make it our ally"

# C/C++ versus Rust for Secure cryptography

PQ Secure

| **Features** | C/C++ | Rust |
| --- | --- | --- |
| Memory Safety | ❌ | ✅ |
| Clear, Defined Semantics | ❌ | ✅ |
| Mandatory Initialization | ❌ | ✅ |
| Built-in Runtime Checks | ❌ | ✅ |
| Secret vs Public Data Separation | ❌ | ✅ |
| Microarchitectural Attack Protection | ❌ | ✅ |
| Secure Data Erasure (Zeroization) | ❌ | ✅ |
| Strong Type Safety | ❌ | ✅ |
| Concurrency Safety | ❌ | ✅ |
| High Performance | ✅ | ✅ |

# C-to-Rust Cryptography

PQ Secure

C source code

```c
#define KYBER_INDCPA_MSGBYTES 123
#define KYBER_N 256
#define KYBER_Q 3329

#include <stdint.h>

void poly_tomsg(uint8_t msg[KYBER_INDCPA_MSGBYTES], const poly *a)
{
  unsigned int i,j;
  uint32_t t;

  for(i=0;i<KYBER_N/8;i++) {
    msg[i] = 0;
    for(j=0;j<8;j++) {
      t  = a->coeffs[8*i+j];
       t += ((int16_t)t >> 15) & KYBER_Q;
       t  = (((t << 1) + KYBER_Q/2)/KYBER_Q) & 1;
      msg[i] |= t << j;
    }
  }
}
```

c2rust.com

Generated Rust source code

```rust
#[no_mangle]
pub unsafe extern "C" fn poly_tomsg(mut msg: *mut uint8_t, mut a: *const libc::c
    let mut i: libc::c_uint = 0;
    let mut j: libc::c_uint = 0;
    let mut t: uint32_t = 0;
    i = 0 as libc::c_int as libc::c_uint;
    while i < (256 as libc::c_int / 8 as libc::c_int) as libc::c_uint {
        *msg.offset(i as isize) = 0 as libc::c_int as uint8_t;
        j = 0 as libc::c_int as libc::c_uint;
        while j < 8 as libc::c_int as libc::c_uint {
            t = (t as libc::c_uint)
                .wrapping_add(
                    (t as int16_t as libc::c_int >> 15 as libc::c_int
                        & 3329 as libc::c_int) as libc::c_uint,
                ) as uint32_t as uint32_t;
            t = (t << 1 as libc::c_int)
                .wrapping_add((3329 as libc::c_int / 2 as libc::c_int) as libc::
                .wrapping_div(3329 as libc::c_int as libc::c_uint)
                & 1 as libc::c_int as libc::c_uint;
            let ref mut fresh0 = *msg.offset(i as isize);
            *fresh0 = (*fresh0 as libc::c_uint | t << j) as uint8_t;
            j = j.wrapping_add(1);
        }
        i = i.wrapping_add(1);
```

- Unsafe C converts to unsafe Rust
- Translation uses Raw pointer instead of Rust native type
- Non idiomatic Rust, relies on C type

=> Less safe, less readable, and more error-prone than idiomatic Rust code written from scratch.

# Pure Rust implementation

PQ Secure

```rust
fn poly_tomsg(msg: &mut [u8; KYBER_INDCPA_MSGBYTES], a: &Poly) {
    for (msg_byte, coeffs) in msg.iter_mut().zip(a.coeffs.chunks(8)) {
        *msg_byte = coeffs.iter().enumerate().fold(0, |acc, (j, &coeff)| {
            let t =
                ((((coeff as u32) << 1)
                .wrapping_add(1665)
                .wrapping_mul(80635) >> 28) & 1)
                as u8;
            acc | (t << j)
        });
    }
}
```

Idiomatic Rust implementation:

- Bound checks, explicitly handle arithmetic overflow
- Rust native type safety: Memory safety, thread safety
- No NULL pointer, no UNDEFINED behavior
- Side-channel resistance

# Final Notes

PQ Secure

- PQC's complexity requires rigorous security guarantees
- Formal Verification ≠ academic concept
- …it is essential to securing next-gen crypto implementations
- Continued advancements must be made to ensure correctness, security, and efficiency in PQC
- AI-driven automation and increased industry adoption promises a bright future for PQC and formal verification!

# Questions?

PQ Secure

Reza Azarderakhsh
CEO PQSecure
razarder@pqsecurity.com