Max Planck Institute for Security and Privacy

Cryptographic Engineering in Post-Quantum Cryptography

Vincent Hwang

November 6th, 2025, National Taiwan University, Taipei, Taiwan

About Myself

- BSc., NTU CSIE, Taiwan (2016.09.01 $\sim$ 2021.06.31).
  - ~ 2020: graph theory/algorithms, complexity theory (surveying).
  - $2020 \sim 2021$, cryptographic engineering:
    - Coures Post-Quantum Cryptography.
    - Lattice-based cryptography (course).
    - Assembly optimizations (internship).
- MSc., NTU CSIE, Taiwan (2021.09.01 $\sim$ 2022.06.31).
  - Cryptographic engineering (assembly, lattices).
- PhD (on going), MPI-SP, Germany (2023.01.01 $\sim$ Now).
  - Cryptographic engineering:
    - Assembly programming for lattices.
    - Formal verification of assembly programs.
  - Elliptic-curve discrete logarithm (ongoing research).

Cryptography

Insecure channel

Alice	G	Bob
$(Q_a = d_a G, d_a)$	$Q_a \longrightarrow$ $\longleftarrow Q_b$	$(Q_b = d_b G, d_b)$
$P_a = d_a Q_b$		$P_b = d_b Q_a$
$\text{Encrypt}_{P_a}(s_a)$	$\longrightarrow$	$s_a = \text{Decrypt}_{P_b}(\dots)$
$s_b = \text{Decrypt}_{P_a}(\dots)$	$\longleftarrow$	$\text{Encrypt}_{P_b}(s_b)$

$P_a = P_b$. Challenge: solve (d_a, d_b, P_a, P_b) from (G, Q_a, Q_b).

Cryptographic Engineering (Traditional Way)

$$
\begin{tikzcd}
\text{Algorithms} \arrow[dr] & & \text{Software} \arrow[dr] & & \\
& \text{Cryptosystem} \arrow[ur] \arrow[rr] & & \text{Hardware} \arrow[r] & \text{Deployment} \\
\text{Security proofs} \arrow[ur] & & & &
\end{tikzcd}
$$

Cryptographic Engineering (Holistic Way)

Cryptosystem

Cryptographic Engineering (Holistic Way)

Algorithms

↓

Cryptosystem

Cryptographic Engineering (Holistic Way)

$$
\begin{tikzcd}
\text{Algorithms} \arrow[d] & \\
& \text{Security proofs} \arrow[dl] \\
\text{Cryptosystem} &
\end{tikzcd}
$$

Cryptographic Engineering (Holistic Way)

Algorithms

Security proofs

Cryptosystem

Software/Hardware

Cryptographic Engineering (Holistic Way)

Algorithms

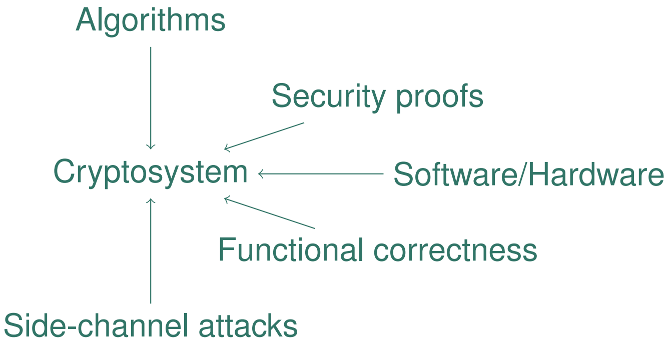
Security proofs

Cryptosystem

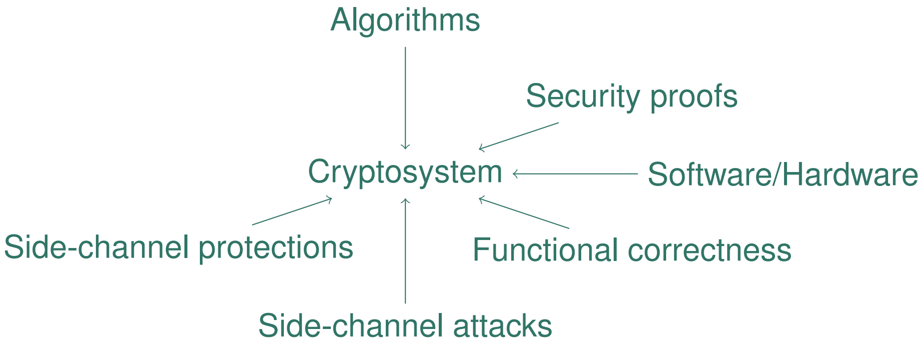
Software/Hardware

Functional correctness

Cryptographic Engineering (Holistic Way)



Cryptographic Engineering (Holistic Way)



Cryptographic Engineering (Holistic Way)

Algorithms

Security proofs

Functional equivalence → Cryptosystem ← Software/Hardware

Side-channel protections

Functional correctness

Side-channel attacks

Cryptographic Engineering (Holistic Way)

Algorithms

Formalization (protections)

Security proofs

Functional equivalence → Cryptosystem ← Software/Hardware

Side-channel protections

Functional correctness

Side-channel attacks

Cryptographic Engineering (Life Cycle)

Recommendations
- Which parameters?
  - Module dimension
  - Polynomial dimension
  - Coefficient ring
- How to compute?
  - FFT
  - Float
  - NTT
  - Modular arith.
- Which hashes?
  - SHA-2
  - SHA-3
- Randomness sampling?

High-level descriptions

Formal verification

Programs

Implementations
- Timing side-channels
  - Conditional jumps
  - Table lookups
  - Variable-time inst.
- Power side-channels
- Performance (hardware)
  - Area
  - Latency
- Performance (software)
  - Low-level arithmetic
  - Register pressure
  - Memory access
  - Vectorization

Post-Quantum Cryptography

- Pre-quantum: integer factorization (`IF`) and elliptic-curve discrete logarithm (`ECDLP`).
- Post-quantum (believed to be secure against quantum computers):
  - Lattice-based.
    - NTRU, LWE, M/R-LWE, M-LWR, M-SIS.
    - Polynomial multiplications, matrix multiplications over $\mathbb{Z}_q$ $(\mathbb{Z}/q\mathbb{Z})$.
  - Supersingular-isogeny-based.
  - Multivariate-based.
  - Code-based.
  - Hash-based.
  - Rich algebraic structures except for the hash-based ones.

NIST Post-Quantum Cryptography Standards

Post-Quantum Cryptography Standardization by the National Institute for Standards and Technology (NIST).

- Selecting cryptosystems: 2016 $\sim$ 2025. Four rounds.
- `FIPS203` (ML-KEM, Kyber), 2024.
  - Lattice. Module-Learning-With-Error (M-LWE).
- `FIPS204` (ML-DSA, Dilithium), 2024.
  - Lattice. M-LWE, Module-Short-Integer-Solution (M-SIS).
- `FIPS205` (SLH-DSA, SPHINCS$^+$), 2024.
  - Hash-based.
- `FIPS206` (FN-DSA, Falcon), 2025?
  - Lattice. SIS over NTRU, fast Fourier sampling,
- HQC, 2027?
  - Code-based.

What Am I Doing?

- Assembly optimization for polynomial multiplications.
  - Assembly: Armv7-M (microcontrollers), Armv8-A and AVX2 (smartphones, personal computers).
  - Lattices: Dilithium, Kyber, NTRU, NTRU Prime, and Saber.
  - Polynomial rings: $\mathbb{Z}_q[x]/\langle x^{256}+1\rangle$, $\mathbb{Z}_{2^{13}}[x]/\langle x^{256}+1\rangle$, $\mathbb{Z}_{2^k}[x]/\langle x^n-1\rangle$, $\mathbb{Z}_q[x]/\langle x^p-x-1\rangle$
  - Optimizations for modular arithmetic.
  - Optimizations for polynomial transformations.
- Formal verification of assembly.
  - Floating-point in Falcon (lattice).
- Cryptanalysis.
  - `ECDLP` on data-center-level GPUs.

This talk.

Multiplying Polynomials without Powerful Multiplication Instructions

Vincent Hwang, YoungBeom Kim, and Seog Chung Seo

- Variable-time long multiplication instructions.
- Constant-time modular arithmetic.
- Prime modulus: generalized Barrett multiplication suitable for multi-limb arithmetic.
- Power-of-two modulus: Nussbaumer.
- Paper: `https://tches.iacr.org/index.php/TCHES/article/view/11926`.
- Artifact: `https://github.com/vincentvbh/PolyMul_Without_PowerfulMul`.

Where Are We?

Recommendations

- Which parameters?
  - Module dimension
  - Polynomial dimension
  - Coefficient ring
- How to compute?
  - FFT
  - Float
  - NTT
  - Modular arith.
- Which hashes?
  - SHA-2
  - SHA-3
- Randomness sampling?

High-level descriptions

Formal verification

Programs

Implementations

- Timing side-channels
  - Conditional jumps
  - Table lookups
  - Variable-time inst.
    - Emulation
- Power side-channels
- Performance (hardware)
  - Area
  - Latency
- Performance (software)
  - Low-level arithmetic
  - Register pressure
  - Memory access
  - Vectorization

What Is Constant-Time in Cryptographic Implementations?

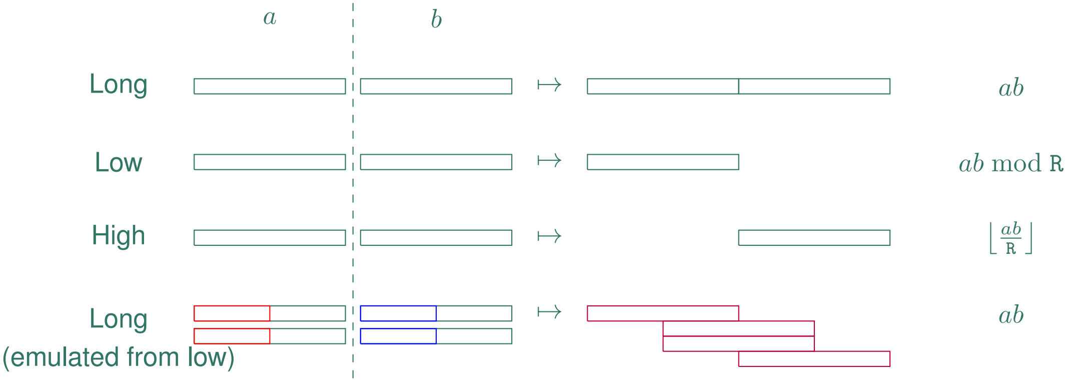
- Constant-time: ~~constant execution time~~, **secret-independent** execution time.
- Variable-time: **secret-dependent** execution time.
- Common computations that leak timing side-channels if input is secret.
  - Conditional branches.
  - Table lookup.
  - Indexing with secret index.
  - Some assembly instructions.
    - Divisions.
    - Floating-point arithmetic.
    - Long multiplications.
    - Typical resolution: emulate with constant-time instructions.

Modular arithmetic in ML-DSA

- ML-DSA.
- Module-lattice-based digital signature.
- $q = 2^{23} - 2^{13} + 1$. Modular arithmetic in $\mathbb{Z}_q$.
- Let `int32_t a b;` we often need one of the following:
  - A 64-bit product.
    - `int64_t c = (int64_t)a * b;`
  - The highest 32 bits of a 64-bit product.
    - `int64_t c = (int32_t)(((int64_t)a * b) >> 32);`
- Cortex-M3: 32-bit Armv7-M instruction set architecture (ISA).
  - 32-bit long multiplications are variable-time $\rightarrow$ leak secret information.
  - Multi-limb arithmetic for the long product $\rightarrow$ significant overhead.

Multiplication Instructions

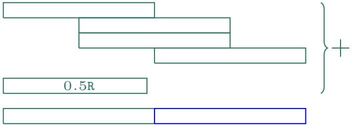
$\mathtt{R}$ is a power of two, 2^{32} on Cortex-M3.



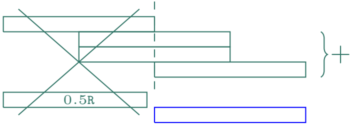
Generalized Barrett Multiplication (2-Limb)

Goal: compute a $c \equiv ab \pmod{q}$. $\left\lfloor \frac{ab}{q} \right\rceil \approx \left\lfloor \frac{ab'}{\mathtt{R}} \right\rceil, b' = \left\lfloor \frac{b\mathtt{R}}{q} \right\rceil$.

Originally, compute $c = ab - \left\lfloor \frac{ab'}{\mathtt{R}} \right\rceil q$ with $\left\lfloor \frac{ab'}{\mathtt{R}} \right\rceil$ as:



Instead, compute $c = ab - \left\llbracket \frac{ab'}{\mathtt{R}} \right\rrbracket q$ with $\left\llbracket \frac{ab'}{\mathtt{R}} \right\rrbracket$ as:



Theory

- For a $\delta > 0$, δ-integer approximation $[\![]\!]$: $\forall r \in \mathbb{R}, |r - [\![r]\!]| \le \delta$.
- For a b, write $b_l + b_h\sqrt{\mathrm{R}} = \left\lfloor \frac{b\mathrm{R}}{q} \right\rceil$. Define $[\![]\!]_b$ as

$$\forall r \in \mathbb{R}, [\![r]\!]_b := \left\lfloor \frac{a_l b_h}{\sqrt{\mathrm{R}}} \right\rceil + \left\lfloor \frac{a_h b_l}{\sqrt{\mathrm{R}}} \right\rceil + a_h b_h$$

  where $a_l + a_h\sqrt{\mathrm{R}} = \frac{r\mathrm{R}}{\left\lfloor \frac{b\mathrm{R}}{q} \right\rceil}, b_l + b_h\sqrt{\mathrm{R}} = \left\lfloor \frac{b\mathrm{R}}{q} \right\rceil$.
- Obviously, $|[\![r]\!]_b - r| < 3$ (see previous slide).
- $|[\![r]\!]_b - \lfloor r \rceil| < 3$ (see paper).
- $\left| \left(ab - \left[\!\!\left[\frac{ab'}{\mathrm{R}}\right]\!\!\right]_b q\right) - ab \bmod q \right| \le 3q$.

Results

- $1.92\times$ faster modular multiplication.
  - $\approx$ as a long multiplication.
  - One of the operands must be a constant.
  - An extra precomputed constant.
  - No assumptions on the modulus.
  - No assumptions on the constants.
  - Above conditions are usually met in lattice-based cryptosystems.
- $1.51\times$ faster NTT (for poly. mul.) in $\mathbb{Z}_q[x]/\langle x^{256}+1\rangle$.

Table: Cycle count on Cortex-M3.

Long	11
Montgomery	23
Barrett	12

Optimize **a modular multiplication** instead of **a long multiplication + a modular reduction**.

Formal Verification of Emulated Floating-Point Arithmetic in Falcon

Vincent Hwang

- Paper: `https://link.springer.com/chapter/10.1007/978-981-97-7737-2_7`.
- Artifact: `https://github.com/vincentvbh/Float_formal`.

Where Are We?

Recommendations

- Which parameters?
  - Module dimension
  - Polynomial dimension
  - Coefficient ring
- How to compute?
  - FFT
  - Float
  - NTT
  - Modular arith.
- Which hashes?
  - SHA-2
  - SHA-3
- Randomness sampling?

High-level descriptions

Formal verification

Programs

Implementations

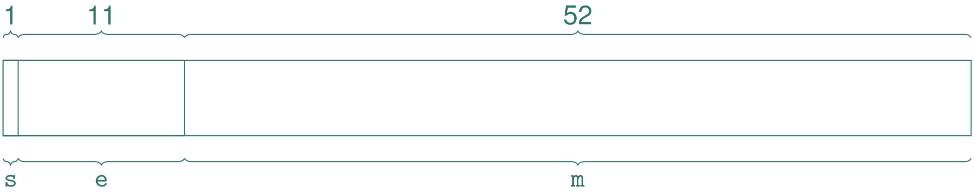
- Timing side-channels
  - Conditional jumps
  - Table lookups
  - Variable-time inst.
    - Emulation
- Power side-channels
- Performance (hardware)
  - Area
  - Latency
- Performance (software)
  - Low-level arithmetic
  - Register pressure
  - Memory access
  - Vectorization

Falcon

- `FIPS206`, FN-DSA. Drafting.
- Lattice-based digital signature scheme.
- Hash-and-sign.
- Double-precision floating-point arithmetic in signing.
- Concerns of floating-point arithmetic.
  - Not always constant-time.
  - Double-precision FPU does not even exist on some microcontrollers!
    - Cortex-M3 (Armv7-M).
    - Cortex-M4F (Armv7E-M + single-precision).
  - Emulating with integer and logical arithmetic.

Floating-Point Arithmetic

Double-precision in this talk.



- $0 < \mathsf{e} < 2047$ (normal values):

$$(-1)^{\mathsf{s}}\, 2^{\mathsf{e}-1075} \left(2^{52} + \mathtt{m}\right) .$$

- Zeros: $\mathsf{e} = \mathtt{m} = 0$.
- Other values: $\mathsf{e} = 0 \wedge \mathtt{m} \neq 0$, $\mathsf{e} = 2047$.

Motivations of Formal Verification for Floats

Incorrect zeroization in floating-point multiplication (FP mul.).

- Emulated floating-point arithmetic (by Falcon submitters) in C and Armv7-M assembly.
- Zeroizing and rounding in the wrong order.
- $\exists$ **normal non-zero** floating-point products $\approx \pm 2^{-1074}$ being **zeroized**.
- $692/2048$ (34%) float constants in FFT admit such float products!

Questions.

- $\exists$ such floats in Falcon? Experimentally no, but no proofs.
- What are the programs actually doing?

Formal Verification

- $\exists$ such float products in Falcon? Range checking.
  - No in FFTs of Falcon.
  - With **input conditions**, $\neg\exists$ floats with incorrect zeroizations.
  - If FFT inputs are integers drawn from $\left[-2^{15}, 2^{15}\right)$, every floats in $\left[2^{-476}, 2^{27}(2^{52} + 6051824482945 68)\right]$ ✓.
  - 0.3 seconds for an FP addition.
  - 2.6 seconds for an FP multiplication.
- What are the programs actually doing? Functional equivalence.
  - $==$ more readable programs.
  - Armv7-M assembly $\longleftrightarrow$ CryptoLine $\longleftrightarrow$ Jasmin.
  - ≈ 1 minute for each FP addition $\longleftrightarrow$.
  - $5 \sim 60$ seconds for each FP multiplication $\longleftrightarrow$.

More Published Works I

Recommendations

- Which parameters?
  - Module dimension
  - Polynomial dimension
  - Coefficient ring
- How to compute?
  - FFT
  - Float
  - NTT
  - Modular arith.
- Which hashes?
  - SHA-2
  - SHA-3
- Randomness sampling?

High-level descriptions

Formal verification

Programs

Armv7-M

Implementations

- Timing side-channels
  - Conditional jumps
  - Table lookups
  - Variable-time inst.
- Power side-channels
- Performance (hardware)
  - Area
  - Latency
- Performance (software)
  - Low-level arithmetic
  - Register pressure
  - Memory access
  - Vectorization

More Published Works II

Recommendations
- Which parameters?
  - Module dimension
  - Polynomial dimension
  - Coefficient ring
- How to compute?
  - FFT
  - Float
  - NTT
  - Modular arith.
- Which hashes?
  - SHA-2
  - SHA-3
- Randomness sampling?

High-level descriptions

Formal verification

Programs

Armv8-A Neon

Implementations
- Timing side-channels
  - Conditional jumps
  - Table lookups
  - Variable-time inst.
- Power side-channels
- Performance (hardware)
  - Area
  - Latency
- Performance (software)
  - Low-level arithmetic
  - Register pressure
  - Memory access
  - Vectorization

More Published Works III

Recommendations
- Which parameters?
  - Module dimension
  - Polynomial dimension
  - Coefficient ring
- How to compute?
  - FFT
  - Float
  - NTT
  - Modular arith.
- Which hashes?
  - SHA-2
  - SHA-3
- Randomness sampling?

High-level descriptions

Formal verification

Programs

Armv8-A Neon

Implementations
- Timing side-channels
  - Conditional jumps
  - Table lookups
  - Variable-time inst.
- Power side-channels
- Performance (hardware)
  - Area
  - Latency
- Performance (software)
  - Low-level arithmetic
  - Register pressure
  - Memory access
  - Vectorization

More Published Works IV

Recommendations

- Which parameters?
  - Module dimension
  - Polynomial dimension
  - Coefficient ring
- How to compute?
  - FFT
  - Float
  - NTT
  - Modular arith.
- Which hashes?
  - SHA-2
  - SHA-3
- Randomness sampling?

High-level descriptions

Formal verification

Programs

Armv8-A Neon, x86 AVX2

Implementations

- Timing side-channels
  - Conditional jumps
  - Table lookups
  - Variable-time inst.
- Power side-channels
- Performance (hardware)
  - Area
  - Latency
- Performance (software)
  - Low-level arithmetic
  - Register pressure
  - Memory access
  - Vectorization

Publications

Authors in alphabetic order.

ACCEHHLNSWY20 CHKSSY21 ACCHKY21

Microcontroller opt.

AABEHOPSS25

Formal verification

AHKS22 HKS24 BHKPY22

H24b

Survey of opt.

HLSSTYY22

BHKYY21 AHY22 CCHY24 HLY24 H24a

H24c

Mid-tier/high-end processor opt.

Ongoing Works

- Deniable authenticated KEM.
  - USENIX 2026, rebuttal phase.
  - Feasibility results in practice (mid-tier/high-end processors).
  - Coauthors: theory.
  - My job: implementation.
- Elliptic-curve discrete logarithm on GPUs.
  - A random curve over a generic 131-bit prime.
  - H100 GPU.
  - Projection: $66,000$ H100-days.
  - Low-level optimizations (inline `ptx`).
  - Memory access.

Future Works

- Optimizations.
  - More architectures: Armv9-A and AVX-512.
  - More cryptosystems: Falcon.
- Formal verification of Falcon.
  - Transformation from high-dim. to one-dim. samplers.
    - Functional correctness.
    - Range properties.
  - Security arguments for the samplers.
    - Rényi divergence.
    - $\exists$ math. proofs for the one-dimensional. case in literature.
    - $\neg\exists$ math. proofs for the high-dimensional case.
- Cryptanalysis on GPUs.
  - Elliptic-curve method in the general number field sieve for `IF`.
    - Random curves over random ~ 130-bit integers.
  - Lattice sieving for shortest vector problem.
    - GPU-memory-friendly bucketing.
    - Inner products.

Thank you for attention