Max Planck Institute for Security and Privacy

Implementing Polynomial Multiplications for Lattice-Based Cryptography on Microcontrollers

Vincent Hwang

January 6th

Today's Plan

1. Lattice-based cryptography.
   - Well-balanced size and computational efficiency.
   - Popular (well-studied).
   - Computational bottleneck:
     - Polynomial multiplications.
     - Cryptographic hash functions.
2. Microcontrollers.
   - Resource-constraint devices.
   - Low-level implementation is more preferred.
   - Optimizing compilers are not very "optimizing."
   - Susceptible to implementation attacks.
3. Polynomial multiplications.
   - Modular arithmetic.
   - Memory optimization of fast homomorphisms.
   - Choices of fast homomorphisms.

Lattice-Based Cryptography

- Cryptography relying on **lattice** hard problems.
- Lattice hard problems:
  - LWE, R-LWE, M-LWE,
  - SIS, R-SIS, M-SIS, NTRU,
  - and more...

Unstructured Learning-With-Errors

1. Generate public A (uniform), secret s, e (small support).
2. Compute public $As + e$.
3. Challenge: find (s, e) from $(A, As + e)$.

- A, s, e: large matrices
- FrodoKEM (NIST round 3 alternate candidate).
- Computational bottleneck: As (dimension $n \times n$).
  - $n = 640 \sim 1344$.
  - $O(n^3)$ or $O\left(n^{\log_2 7}\right)$.
  - Too slow in practice.

Structured Learning-With-Errors

1. Generate public A (uniform), secret s, e (small support).
2. Compute public $As + e$.
3. Challenge: find (s, e) from $(A, As + e)$.

- Ring-learning-with-errors (R-LWE).
  - A, s, e: elements in **a large polynomial ring**.
  - NewHope (NIST round 2 candidate).
- Module-learning-with-errors (M-LWE).
  - A: small matrix over **a large polynomial ring**.
  - s, e: small vectors over **a large polynomial ring**.
  - More flexible on the parameter choices.
  - ML-KEM, ML-DSA (NIST standards).
- We need fast polynomial multiplications!
  - LWE: $O\left(n^3\right)$ or $O\left(n^{\log_2 7}\right)$.
  - R-LWE, M-LWE (with small module dimension): $O\left(n^2\right)$ or $O\left(n \log n\right)$.

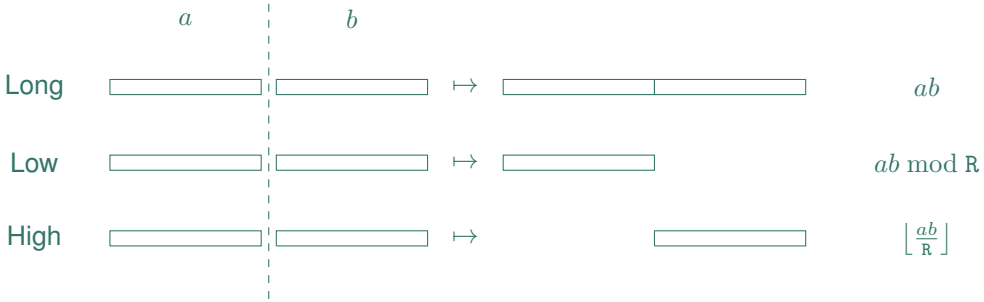
Polynomial Multiplication

Compute $\boldsymbol{a}(x)\boldsymbol{b}(x)$ in $R[x]/\langle \boldsymbol{g}(x) \rangle$, $R = \mathbb{Z}_m := \mathbb{Z}/m\mathbb{Z}$ for a positive integer m.

1. Modular arithmetic.
   - How efficient the ring arithmetic in R can be?
2. Fast homomorphisms.
   - Which asymptotically faster (faster than $O\left(n^2\right)$) approaches we should choose?
   - Very complicated. Takes many tries.
3. Memory operations.
   - Usually invisible from the math description.

Multiplication Instructions

$\mathtt{R} = 2^{16}, 2^{32}$. Signed arithmetic throughout this talk.



- How to map the modular arithmetic to multiplication instructions?
- How do the multiplication instructions perform on a particular processor?

Montgomery Multiplication

Fixed modulus q, precision $\mathtt{R}$ (power-of-two), precomputed $-q^{-1} \bmod \mathtt{R}$.
Given a, b, compute

$$\frac{ab + \left(-abq^{-1} \bmod \mathtt{R}\right) q}{\mathtt{R}} \equiv ab\mathtt{R}^{-1} \pmod{q}.$$

Division and modulo reduction by R are cheap, free if nicely aligned with the architecture.

- Result derived from the high parts of the long products.
- Result is scaled by $\mathtt{R}^{-1} \bmod q$.
  - Additional Montgomery multiplication with $\mathtt{R}^2 \bmod q \longrightarrow \equiv ab \pmod{q}$.
  - Replace b by $b\mathtt{R} \bmod q$ (precomputation required).

Cost of Montgomery Multiplication (with Precomputation)

$$\frac{a\,(b\mathtt{R} \bmod q) + \left(-a\,(b\mathtt{R} \bmod q)\,q^{-1} \bmod \mathtt{R}\right) q}{\mathtt{R}} \equiv ab \pmod q$$

1. $a, (b\mathtt{R} \bmod q) \mapsto a\,(b\mathtt{R} \bmod q)$: long mul.
2. $a\,(b\mathtt{R} \bmod q) \mapsto -a\,(b\mathtt{R} \bmod q)\,q^{-1} \bmod \mathtt{R}$: low mul. (assuming $\bmod \mathtt{R}$ is free).
3. $-a\,(b\mathtt{R} \bmod q)\,q^{-1} \bmod \mathtt{R} \mapsto \left(-a\,(b\mathtt{R} \bmod q)\,q^{-1} \bmod \mathtt{R}\right) q$: long mul.
4. Add.
5. Divide by $\mathtt{R}$.

Summary: 2 long mul. + 1 low mul.

Aligning the Arithmetic with Low-Level Instructions

- Armv7-M (32-bit ISA): native 32-bit low (w/ acc.), long mul. (w/ acc.) instructions.
- Processors:
  - Cortex-M3.
    - Low takes 1 cycle, acc. one takes 2 cycles.
    - Long mul takes $3 \sim 7$ cycles.
  - Cortex-M4.
    - DSP extension: high multiplication instructions.
    - Each multiplication instruction takes 1 cycle.

How many cycles does 32-bit Montgomery multiplication take?

- $9 \sim 16$ cycles on Cortex-M3.
- 3 cycles on Cortex-M4.

Alternatives?

- Can we do better than 32-bit Montgomery mul. on Cortex-M3?
  - Yes, 32-bit Barrett mul is faster.
- Can we do better than 32-bit Montgomery mul. on Cortex-M4?
  - No other known approach outperforming it.
  - Other approaches perform the same, but with more memory for the precomputed values.

Barrett Multiplication

Given a, b, we have

$$ab \bmod q = ab - \left\lfloor \frac{ab}{q} \right\rceil q$$

by definition.

Barrett mul. approximates $\left\lfloor \frac{ab}{q} \right\rceil$ up to a small integral perturbation. Various formulations ($b\mathtt{R}/q$ precomputed).

- $\left\lfloor \frac{a \lfloor b\mathtt{R}/q \rceil}{\mathtt{R}} \right\rceil$.
- $\left\lfloor \frac{a \lfloor b\mathtt{R}/q \rceil}{\mathtt{R}} \right\rceil$.
- $\left[\!\left[\frac{a \lfloor b\mathtt{R}/q \rceil}{\mathtt{R}} \right]\!\right]$, advanced rounding technique $[\![]\!]$.

Cost of Barrett Multiplication

Typically, $ab - \left\lfloor \frac{a\, b\mathtt{R}/q}{\mathtt{R}} \right\rceil q \in \left[-\frac{\mathtt{R}}{2}, \frac{\mathtt{R}}{2}\right)$ so

$$ab - \left\lfloor \frac{a \left\lfloor b\mathtt{R}/q \right\rceil}{\mathtt{R}} \right\rceil q = (ab \bmod \mathtt{R}) - \left(\left\lfloor \frac{a \left\lfloor b\mathtt{R}/q \right\rceil}{\mathtt{R}} \right\rceil q \bmod \mathtt{R} \right).$$

1. $a, b \mapsto ab \bmod \mathtt{R}$: low mul.
2. $a, \left\lfloor b\mathtt{R}/q \right\rceil \mapsto \left\lfloor \frac{a \left\lfloor b\mathtt{R}/q \right\rceil}{\mathtt{R}} \right\rceil$: high mul.
3. $\left\lfloor \frac{a \left\lfloor b\mathtt{R}/q \right\rceil}{\mathtt{R}} \right\rceil, q \mapsto \left\lfloor \frac{a \left\lfloor b\mathtt{R}/q \right\rceil}{\mathtt{R}} \right\rceil q \bmod \mathtt{R}$: low mul.
4. Subtract.

Summary: 2 low mul. + 1 high mul.

- Cortex-M3: $6 \sim 10$ cycles.
- Cortex-M4: 3 cycles.

Comparisons

- Cortex-M3:
  - Montgomery: $9 \sim 16$ cycles.
  - (Preferred) Barrett: $6 \sim 10$ cycles.
- Cortex-M4:
  - (Preferred) Montgomery: 3 cycles, precomputed $b\mathtt{R} \bmod q$ (we don't need b anymore).
  - Barrett: 3 cycles, precomputed $\left\lfloor \frac{b\mathtt{R}}{q} \right\rceil$ (we still need b).
- Know your instruction set architecture.
- Know your platform.
- Constant-time?
  - Advanced rounding technique on Cortex-M3.
  - Solution: see this paper: `https://tches.iacr.org/index.php/TCHES/article/view/11926/11785`.

Fast Homomorphisms

$\deg \boldsymbol{g} = n, \deg \boldsymbol{h}_i < n.$

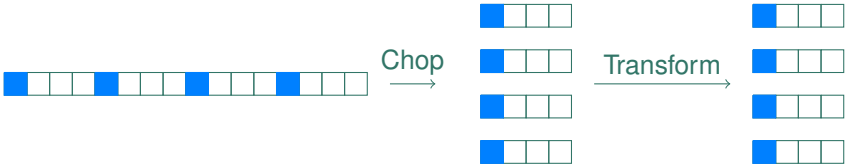
$$R[x]/\langle \boldsymbol{g} \rangle \xrightarrow{\text{Fast homomorphism } f.} \prod_i R[x]/\langle \boldsymbol{h}_i \rangle$$

$$\boldsymbol{a}(x)\boldsymbol{b}(x) = f^{-1}\left(f\left(\boldsymbol{a}(x)\right) f\left(\boldsymbol{b}(x)\right) \in R[x]/\langle \boldsymbol{g} \rangle\right.$$

- Large polynomial ring $\rightarrow$ several small polynomial rings.
- Toom–k: $O\left(n^{\log_k(2k-1)}\right)$.
- Number-theoretic transforms: $O\left(n \log_2 n\right)$.
- Number-theoretic transforms (with poly. ring extensions): $O\left(n \log_2 n \log_2 \log_2 n\right)$.
- A lot of ways.

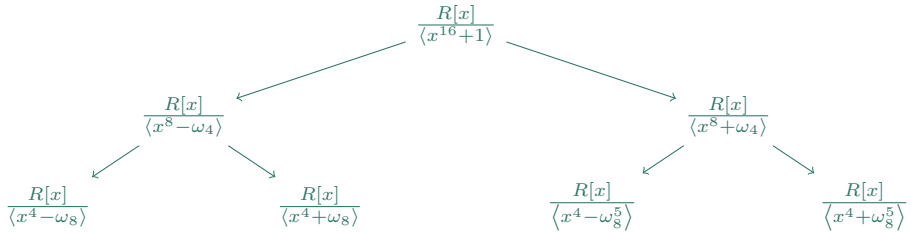
Frequently Seen Patterns

Component-wise arithmetic of sub-polynomials.



- Chop phase is free (for illustration only).
- Transform phase costs.
  - Arithmetic cost can be inferred from the math.
  - Memory cost is usually invisible from the math.

Number-Theoretic Transform



Poly. mul. in $R[x]/\langle x^{16}+1\rangle \rightarrow$ four poly. mul. of the form $R[x]/\langle x^4-\alpha\rangle$.

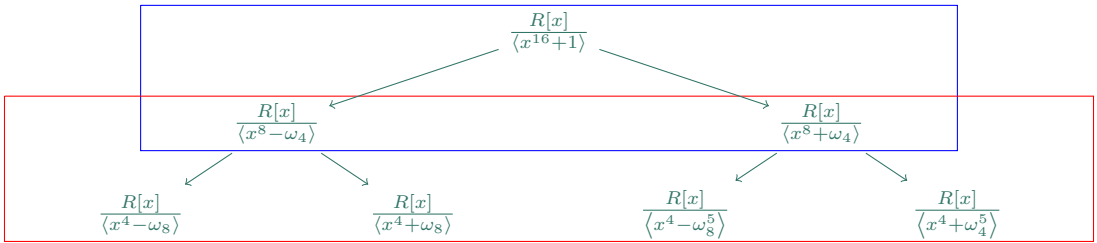
Memory and Registers

- Registers:
  - Fast memory inside the processor.
  - # register is ISA-fixed.
  - Bit-size of register is ISA-fixed.
- Memory:
  - Access though memory bus: incurring overhead.
  - Size of memory depends on what the platform designers want.

How to minimize transfers between memory and registers?

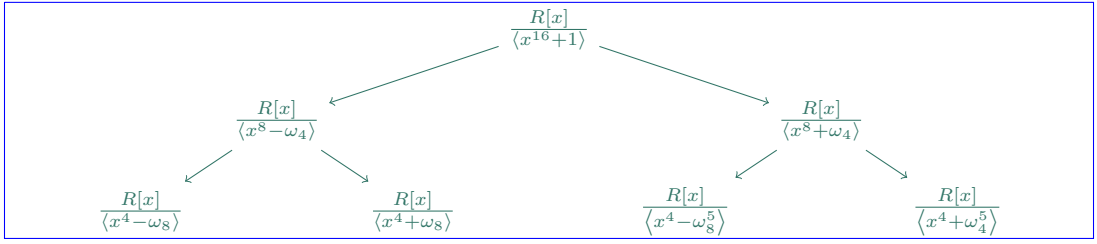
- Issue as much arithmetic as possible while dependent data are in register.
- Restructure the computation flow if needed.

Memory Operation Cost (First Try)



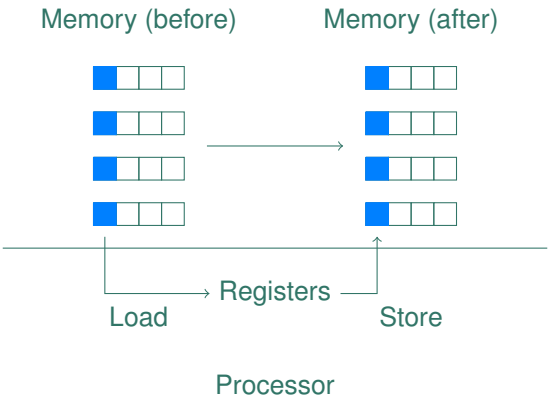
Two load-store pairs for each entry. First and Second.

Memory Operation Cost (Optimized): Layer-Merging

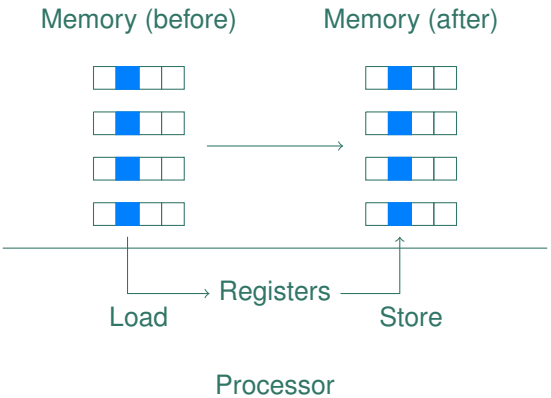


One load-store pair for each entry.

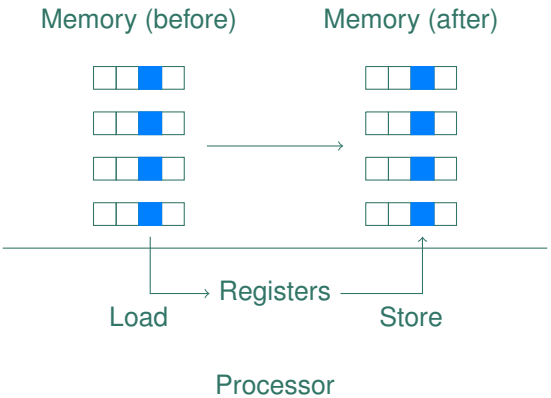
Memory Operation Cost (Optimized, Concretely)



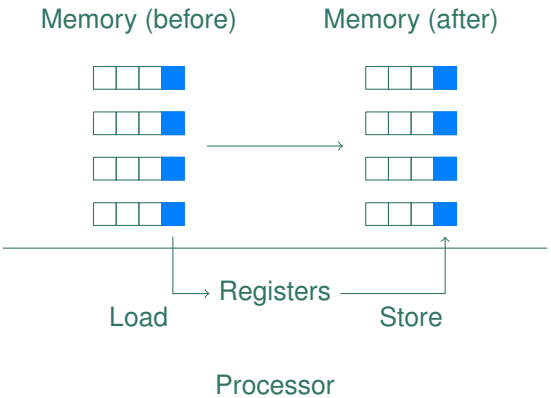
Memory Operation Cost (Optimized, Concretely)



Memory Operation Cost (Optimized, Concretely)



Memory Operation Cost (Optimized, Concretely)



Choosing a Fast Homomorphism

$\deg \boldsymbol{g} = n, \deg \boldsymbol{h}_i < n.$

$$R[x]/\langle \boldsymbol{g} \rangle \xrightarrow{\text{Fast homomorphism } f.} \prod_i R[x]/\langle \boldsymbol{h}_i \rangle$$

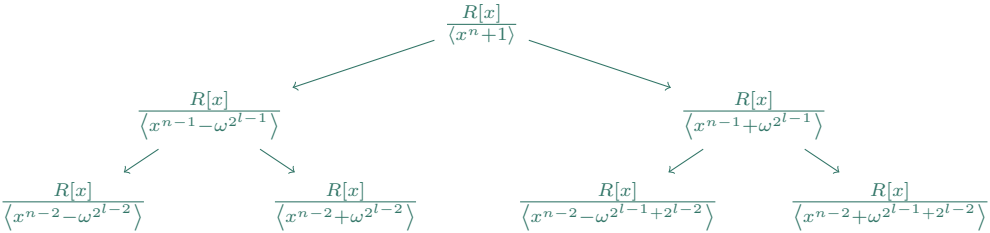
$$\boldsymbol{a}(x)\boldsymbol{b}(x) = f^{-1}\left(f\left(\boldsymbol{a}(x)\right) f\left(\boldsymbol{b}(x)\right)\right) \in R[x]/\langle \boldsymbol{g} \rangle$$

- Necessary conditions of the homomorphisms.
- Algebraic properties of the polynomial rings.
- Performance characteristics of the platforms:
  - Multiplication instructions.
  - Vectorization (for high-performance processors).

Power-of-Two Cyclotomic Polynomial Rings I

For $\forall l \leq n \in \mathbb{Z}$, $n = 2^m$, we require $\exists w^{2^l} = -1 \in R$ and $\exists 2^{-l} \in R$ for the Cooley–Tukey FFT

$$\frac{R[x]}{\langle x^n + 1 \rangle} \cong \prod \frac{R[x]}{\langle x^{n-1} \pm w^{2^{l-1}} \rangle} \cong \cdots \cong \prod_i \frac{R[x]}{\langle x^{n-l} - w^{\mathtt{bitrev}_{2^l}(2i+1)} \rangle}.$$



Power-of-Two Cyclotomic Polynomial Rings II

$R = \mathbb{Z}_q$ in lattices. $\exists \omega^{2^l} = -1 \in R$? $\exists 2^{-l} \in R$?

Scheme	q	l
ML-KEM	$3329 = 13 \cdot 256 + 1$	≤ 7
ML-DSA	$8380417 = 2^{23} - 2^{13} + 1$	≤ 12
Saber (round 3 candidate)	$8192 = 2^{13}$	0

- ML-KEM: $\mathbb{Z}_{3329}[x]/\langle x^{256}+1\rangle \cong \prod_i \mathbb{Z}_{3329}[x]/\langle x^2 - \alpha_i\rangle$.
- ML-DSA: $\mathbb{Z}_{8380417}[x]/\langle x^{256}+1\rangle \cong \prod_i \mathbb{Z}_{8380417}[x]/\langle x - \alpha_i\rangle$.
- Saber: $\mathbb{Z}_{8192}[x]/\langle x^{256}+1\rangle$?

Approach One: Choose a New Coefficient Ring

In lattices, one of $\boldsymbol{a}, \boldsymbol{b}$ has small coefficients.

- Coeff. of $\boldsymbol{a} < q$.
- Coeff. of $\boldsymbol{b} <$ a small constant μ.
- Coeff. of $\boldsymbol{ab}$ in $\mathbb{Z} < q \cdot n \cdot \mu$.
- Compute over $\mathbb{Z}_{q'}$ for an odd $q' \geq q \cdot n \cdot \mu$.
- Reduce to $\mathbb{Z}_q$ at the end of poly. mul.
- CRT: $\mathbb{Z}_{q_1} \times \mathbb{Z}_{q_2}$ with $q_1 q_2 \geq q \cdot n \cdot \mu$.
- Signed arithmetic relaxed to $\geq q \cdot n \cdot \mu/2$.

Approach Two: Polynomial Ring Extension

For $\forall l \leq n \in \mathbb{Z}$, we require $\exists w^{2^l} = -1, 2^{-l} \in R$ for

$$\frac{R[x]}{\langle x^n + 1 \rangle} \cong \prod \frac{R[x]}{\langle x^{n-1} \pm w^{2^{l-1}} \rangle} \cong \cdots \cong \prod_i \frac{R[x]}{\langle x^{n-l} - w^{\mathtt{bitrev}_{2^l}(2i+1)} \rangle}.$$

Craft an $\boldsymbol{\omega}$.

- Introduce $y \sim x^{16}$, $\mathcal{R} = \mathbb{Z}_q[y]/\langle y^{16} + 1 \rangle$, $\boldsymbol{\omega} = y$.
- $\hookrightarrow$: Extension with zero-padding.
- We have

$$\frac{\mathbb{Z}_q[x]}{\langle x^{256} + 1 \rangle} \cong \frac{\mathcal{R}[x]}{\langle x^{16} - y \rangle} \hookrightarrow \frac{\mathcal{R}[x]}{\langle x^{32} - 1 \rangle} \cong \prod_i \frac{\mathcal{R}[x]}{\langle x - \boldsymbol{\omega}^{\mathtt{bitrev}_{32}(i)} \rangle}.$$

Craft the inverses of powers of two.

- $2^{-1}\mathcal{R}$: localization of $\mathcal{R}$ at $\{1, 2, 4, \cdots\}$.
- Practically, compute $2^l \boldsymbol{ab} \in 2^l\mathcal{R}$, then shift right by l bits.

Performance

- NTT, $\mathbb{Z}_{q'}[x]/\langle x^{256}+1\rangle$.
  - Modular arithmetic in $\mathbb{Z}_{q'}$.
  - Multiplication-heavy hom. + fast polymul. with small dim.
- NTT, $(\mathbb{Z}_{q_1} \times \mathbb{Z}_{q_2})[x]/\langle x^{256}+1\rangle$.
  - Modular arithmetic in $\mathbb{Z}_{q_1} \times \mathbb{Z}_{q_2}$.
  - Multiplication-heavy hom. + fast polymul. with small dim.
- Polynomial ring extension, $2^{-1}\mathcal{R}[x]/\langle x^{32}-1\rangle$:
  - Arithmetic in $\mathbb{Z}_{8192} \longrightarrow$ plain arithmetic in registers.
  - Addition-heavy hom. + large # fast polymul. with small dim.

Processor	NTT, $\mathbb{Z}_{q'}$	NTT, $\mathbb{Z}_{q_1} \times \mathbb{Z}_{q_2}$	Polynomial ring extension
Cortex-M3	98 213	69 187	**45 797**
Cortex-M4	**23 107**	37 161	Close to Cortex-M3

Fastest approaches on high-performance processors.

- Processors implementing Armv8-A Neon: NTT, $\mathbb{Z}_{q'}$.
- Processors implementing x86 AVX2: NTT, $\mathbb{Z}_{q_1} \times \mathbb{Z}_{q_2}$.

Summary I

Optimize polynomial multiplication on microcontrollers.

1. Pick a modular arithmetic.
2. Pick a fast homomorphism.
   - A lot more to say. See this survey: `https://cic.iacr.org/p/1/2/1`.
3. Optimize the memory operations.

In reality, we jump between each steps.

- A particular modular arithmetic might be the fastest one for a homomorphism on a certain platform, BUT NOT ON OTHER PLATFORMS.
- A particular homomorphism might be best applied to the polynomial ring on a certain platform, BUT NOT ON OTHER PLATFORMS.

Summary II

Know your platforms.

- Instruction set architecture (ISA):
  - Armv7E-M: `https://developer.arm.com/documentation/ddi0403/ed`.
  - Armv8-M: `https://developer.arm.com/documentation/ddi0553/latest`.
  - Armv7-A: `https://developer.arm.com/documentation/ddi0406/cb`.
  - Armv8-A: `https://developer.arm.com/documentation/ddi0487/gb/?lang=en`.
  - Armv9-A: `https://developer.arm.com/documentation/ddi0487/latest/`.
- Processors.
  - One for each processor.
    - Cortex-M: search “Technical Reference Manual.”
    - Cortex-A: search “Software Optimization Guide.”
  - If unlucky, no document for the target processor.
    - Benchmark the instructions following the ISA.
  - If unlucky, undocumented instructions.
    - AMX coprocessor in Apple M1, M2, M3: See here: `https://github.com/corsix/amx`.
    - Guess how instructions are encoded.
    - Benchmark the undocumented instructions.

Summary III

Know the fundamentals.

- Fast homomorphisms are conditioned on several things.
- Fundamentals in algebra help crafting the desired algebraic structures.

Various algebraic techniques that were found useful in the literature for lattices.

- FFTs:
  - Radix-2 Cooley–Tukey.
  - Non-radix-2/mixed-radix Cooley–Tukey.
  - Good–Thomas (tensor product of algebras).
  - (Truncated)Rader's.
  - Schönhage, Nussbaumer (polynomial ring extension).
- Toeplitz matrix-vector product (algebraic dual of hom., vector instructions).

Summary IV

Homomorphisms in ML-KEM and ML-DSA are fixed.

ML-KEM (16-bit $\mathbb{Z}_q$)			
Processor	ISA	Modular arith.	Layer-merging
Cortex-M3	Armv7-M	Plantard	3
Cortex-M4	Armv7E-M	Plantard	4
Apple M1	Armv8-A	Barrett	4
Haswell	x86 + AVX2	Montgomery	3
ML-DSA (32-bit $\mathbb{Z}_q$)			
Processor	ISA	Modular arith.	Layer-merging
Cortex-M3	Armv7-M	Barrett	3
Cortex-M4	Armv7E-M	Montgomery	4
Apple M1	Armv8-A	Barrett	4
Haswell	x86 + AVX2	Montgomery	3

Summary V

Saber ($\mathbb{Z}_{8192}$)				
Processor	ISA	Modular arith.	Hom.	Layer-merging
Cortex-M3	Armv7-M	Plain	Poly. ring ext.	2
Cortex-M4	Armv7E-M	Montgomery	NTT, $\mathbb{Z}_{q'}$	4
Apple M1	Armv8-A	Barrett	NTT, $\mathbb{Z}_{q'}$	4
Haswell	x86 + AVX2	Montgomery	NTT, $\mathbb{Z}_{q_1} \times \mathbb{Z}_{q_2}$	3

Thank you for your attention

Slides: https://vincentvbh.github.io/slides/UK_MMU-UTAR_PQC_2026_01_06.pdf