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.
   - 3.1 Modular arithmetic.
   - 3.2 Fast homomorphisms (more on memory).

# 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 for microcontrollers.

# 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

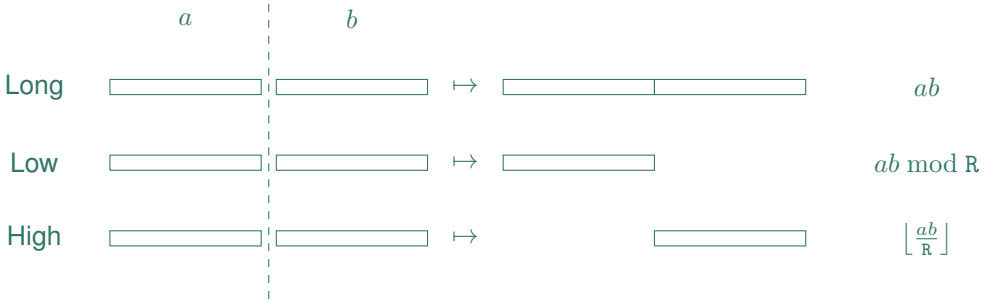

Compute $\boldsymbol{a}(x)\boldsymbol{b}(x)$ in $R[x]/\langle \boldsymbol{g}(x) \rangle$, $R = \mathbb{Z}_m := \mathbb{Z}/m\mathbb{Z}$ for a positive integer $m$.

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. Decide at the end.
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, $ab - \left\lfloor \frac{a\, b\mathtt{R}/q}{\mathtt{R}} \right\rceil q \in [-\frac{\mathtt{R}}{2}, \frac{\mathtt{R}}{2})$ so

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

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


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

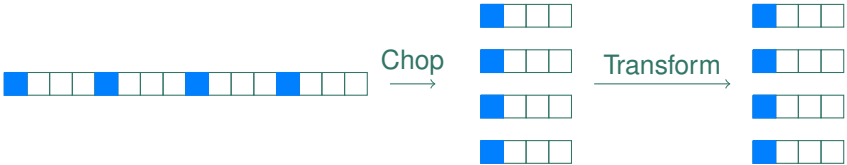
$$R[x]/\langle \boldsymbol{g} \rangle \xrightarrow{\text{Fast homomorphism } f.} \prod_i R[x]/\langle \boldsymbol{h}_i \rangle$$

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

- 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

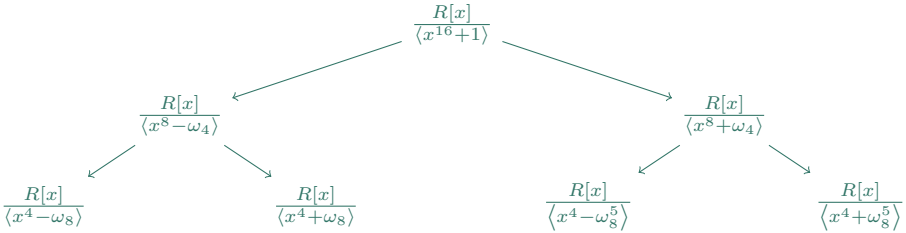

Component-wise arithmetic of sub-polynomials.




- 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

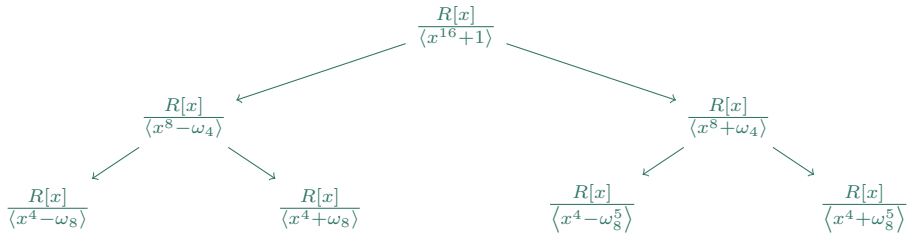





- Poly. mul. in $R[x]/\langle x^{16}-1\rangle \rightarrow$ four poly. mul. of the form $R[x]/\langle x^4-\alpha\rangle$.
- $R[x]/\langle x^{16}+1\rangle$ : 256 mul. + 240 add.
- Four $R[x]/\langle x^4-\alpha\rangle$ : 64 mul. + 48 add.

# Arithmetic Cost






1. $R[x]/\langle x^{16}+1\rangle \to R[x]/\langle x^8-\omega_4\rangle \times R[x]/\langle x^8+\omega_4\rangle$: $8$ mul., $16$ add./sub.
2. $R[x]/\langle x^8-\omega_4\rangle \to R[x]/\langle x^4-\omega_8\rangle \times R[x]/\langle x^4+\omega_8\rangle$: $4$ mul., $8$ add./sub.
3. $R[x]/\langle x^8+\omega_4\rangle \to R[x]/\langle x^4-\omega_8^5\rangle \times R[x]/\langle x^4+\omega_8^5\rangle$: $4$ mul., $8$ add./sub.

- Transformation cost: $16$ mul. and $32$ add.
- Three transformations for poly. mul: $48$ mul. and $96$ add.
- Total: $112$ mul. $+$ $144$ add. (previously 256 mul. + 240 add.).

# Memory and Registers



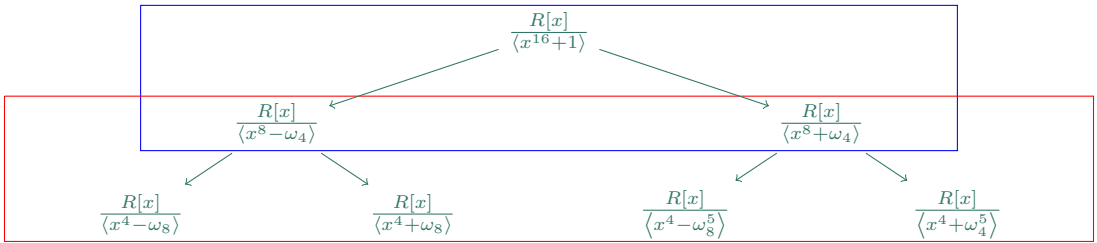

- Registers:
  - Fast memory inside the processor.
  - # register is ISA-fixed.
  - Bit-size of register is ISA-fixed.
- Memory:
  - Access though memory bus: incurring overhead.
  - Size of memory depends on what the platform designers want.

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)

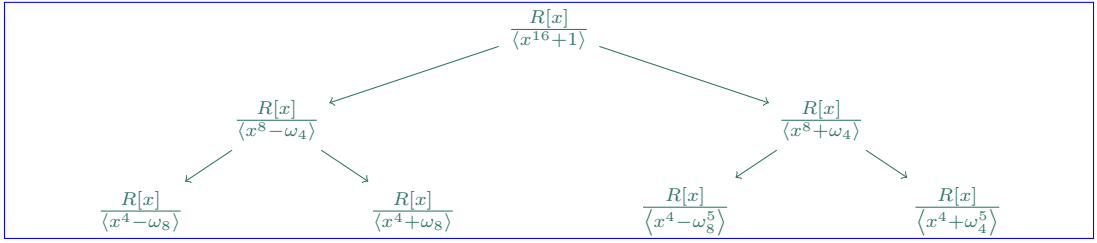





Two load-store pairs for each entry. First and Second.

# Memory Operation Cost (Optimized): Layer-Merging

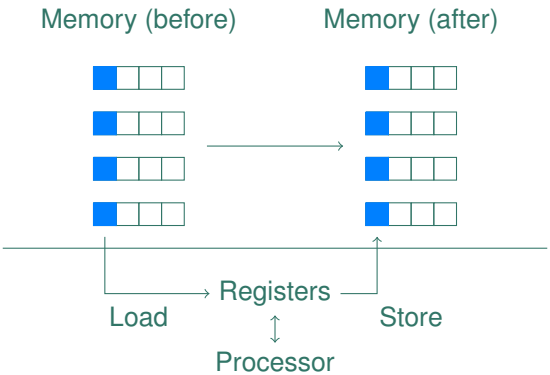





One load-store pair for each entry.

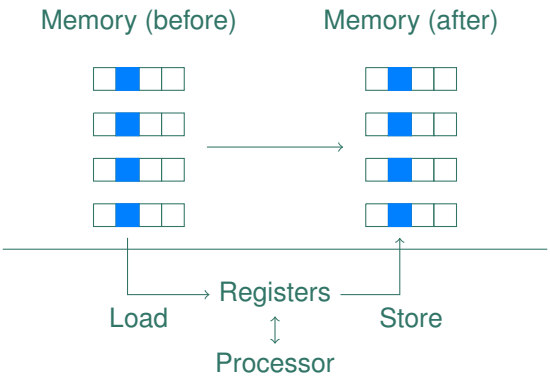
# Memory Operation Cost (Optimized, Concretely)





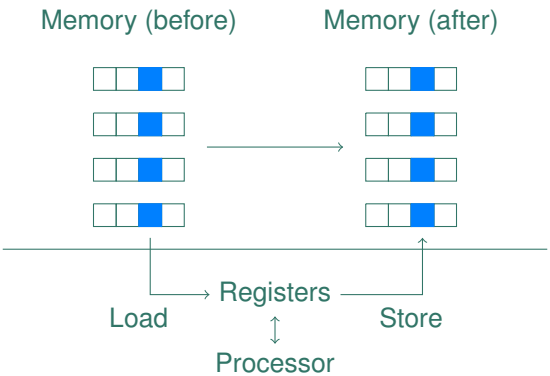
# Memory Operation Cost (Optimized, Concretely)





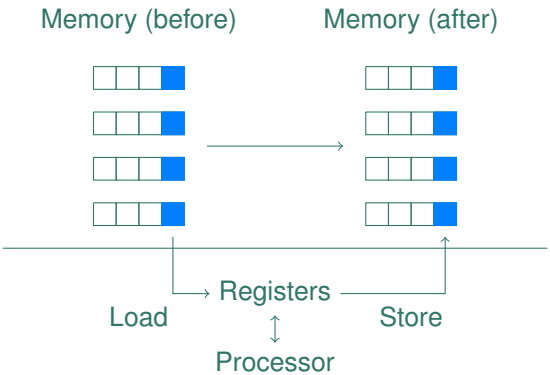
# Memory Operation Cost (Optimized, Concretely)





# Memory Operation Cost (Optimized, Concretely)





# 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 platform.

- 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.
  - If unlucky, no document for the target processor.
    - Benchmark the instructions following the ISA.
  - If unlucky, undocumented instructions.
    - AMX in Apple M1, M2, M3: See here: `https://github.com/corsix/amx`.
    - Guess how instructions are encoded.
    - Benchmark the undocumented instructions.

# Summary III



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

| ML-KEM | | | |
|---|---|---|---|
| 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 | | | |
|---|---|---|---|
| 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 |

# Thank you for your attention

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