Optimizations on ECDH25519 and ML-KEM

David Wu

Cryptography Engineering Final Project

December 16, 2025

ML-KEM(Kyber)

- Selected by NIST as the primary Post-Quantum Cryptography (PQC) standard for general-purpose key encapsulation.
- Resistant to quantum computer attacks
- Security is based on the Module Learning With Errors (M-LWE) problem in structured lattices.
- Operations are performed over polynomial rings $Z_q[x]/x^n + 1$ where $n = 256$ and $q = 3329$

Open Source Implementation

An update on Keccak performance on ARMv7-M

- Author : Alexandre Adomnicai
- Faster Keccak Implementation in Assembly with two versions
  - **Balanced** (My choice)
  - Fast –Lazy rotations for all rounds which gains 2% performance improvement but with 50% extra code size
- Grouping memory accesses together as much as possible for pipelining
- 32-bit interleaving
- Lazy rotations for 3/4 of the rounds

Optimization Strategies 1 – Polynomial Operations

Small Optimizations

- Replaces dynamic indexed shifting with incremental shifting to reduce arithmetic overhead within **cbd2**.
- Implement the whole base multiplication directly to reduce calling overhead

DSP Instructions

- **poly_add** and **poly_sub** is suitable for DSP operations.
- **uadd16 / usub16** can reach 2x speedup for these functions.

Compression / Decompression

- Intermediate bit-packing entirely within registers to eliminate redundant memory access
- Barrel Shifter to embed bitwise shifts into logical instructions
- Merging scaling factors into multiplication constants with post-increment addressing for pointer updates.

Optimization Strategies 2 – NTT

NTT Optimization

- Performance Bottleneck : Load / Store Operations
  - Loop Unrolling
  - Layer Fusing – $7 = 2 + 2 + 2 + 1$
  - Inverse NTT with Cooley-Tukey Butterfly to achieve lazy reduction
  - Merge twist factors and scaling factors for each element
- Performance Improvement : Two at a time
  - The elements for NTT are 16-bit, but the registers are 32-bit.
  - **ldr / str** to load and store neighboring elements together
  - DSP instructions to achieve SIMD arithmetic operations – **uadd16 / usub16**
  - Specialized Montgomery Reduction for two elements from Faster Kyber and Dilithium on the Cortex-M4
  - Authors : Amin Abdulrahman / Vincent Hwang / Matthias J. Kannwischer / Amber Sprenkels

Optimization Results 1 – ML-KEM

C Reference Implementation

- **poly_ntt** : 35830 Cycles
- **Keypair Generation** : 1325464 Cycles
- **Encapsulation** : 1481614 Cycles
- **Decapsulation** : 1741181 Cycles

Until 12 / 15

- **poly_ntt** : 19746 Cycles (44.89% Speedup)
- **Keypair Generation** : 772394 Cycles(41.73% Speedup)
- **Encapsulation** : 823690 Cycles(44.41% Speedup)
- **Decapsulation** : 960378 Cycles(44.84% Speedup)

Optimization Results 2 – Specific Functions and Code Size

Individual Functions

- **NTT** : 23546 $\rightarrow$ 14566 (38.1% Reduction)
- **Inverse NTT** : 32376 $\rightarrow$ 15469 (52.2% Reduction)
- **poly_add** : 2056 $\rightarrow$ 1045 (49.2% Reduction)
- **poly_sub** : 2343 $\rightarrow$ 1045 (55.4% Reduction)

NTT code size in **ntt.S**

- Overall 2028 bytes
- However, it includes base multiplication + NTT + Inverse NTT
- Each of them $\leq$ 1024 Bytes

Next Steps

- Find possibilities to employ FPRs to reduce memory operations
- More aggressive layer fusing in NTT within code size constraint
- Implement functions with loops in assembly to reduce calling overhead with possible unrolling
- Optimizing redundant **mov**
- Register allocation and pipelining

ECDH25519

- An Elliptic Curve Diffie-Hellman key exchange protocol using Curve25519.
- Allows two parties to securely agree on a shared secret over an insecure channel.
- 128-bit security level with prime $p = 2^{255} - 19$, allowing fast arithmetic on CPUs.

Let G be the standard base point on the curve.

1. **Alice**: Generates private a, sends public $A = a \cdot G$.
2. **Bob**: Generates private b, sends public $B = b \cdot G$.
3. **Shared Secret**: Both compute $S = a \cdot B = b \cdot A = (a \cdot b) \cdot G$.

Optimization Strategies 1

Basic Side-Channel Prevention

- Constant time double-and-add instead of variable scalar multiplication with secret-dependent branch
- Branchless cmov and cswap helper

Finite Field Arithmetic

- Radix-2^8 for all operations $\rightarrow$ Radix $2^{25.5}$
  - Before : 1024 Multiplications and additions without overflow risk. However, register is 32-bit wide.
  - Now : Repacking without freezing in the middle of multiplication(squaring). Only about 100 multiplications needed. (90% improvement)
- Multiplication and addition $\rightarrow$ **smlal** and **smull** in assembly
- Nested Loop Operand Scanning $\rightarrow$ Product Scanning
- Specialized Squaring to reduce repeated multiplications(results later)

Optimization Strategies 2

Scalar Multiplication

- Fixed-base Window Method with precomputed table($w = 4$)
- Conditional move to achieve constant-time loading
- Calculate base $G, 2^w G, 2^{2w} G, \ldots$
- $table[i][j] = j \times 2^{iw} G$
- Reduce 255 doubling + 128 additions to 64 additions only
- Large size : 512 KB (within 1MB flash)

Optimization Results – ECDH25519

C Reference with constant-time Implementation

- **crypto_scalarmult_base** : 54552132 Cycles
- **crypto_scalarmult** : 51772998 Cycles

Until 12 / 15

- **crypto_scalarmult_base** : 3141710 Cycles (94.24% Speedup)
- **crypto_scalarmult** : 12454872 Cycles (75.94% Speedup)

More Specifically...

- **fe25519_mul** : 9695045 $\rightarrow$ 1812045 Cycles (81.31% Speedup)
- **fe25519_sqr** : 9695078 $\rightarrow$ 1431045 Cycles (85.24% Speedup)

Next Step

- Larger radix in other functions
- FPRs usage to reduce stack usage
- Pipelining and instruction selection
- Be aware of secret-dependent operations

References I

[Abd+22] Amin Abdulrahman, Vincent Hwang, Matthias J. Kannwischer, and Amber Sprenkels. *Faster Kyber and Dilithium on the Cortex-M4*. Cryptology ePrint Archive, Paper 2022/112. 2022. URL: https://eprint.iacr.org/2022/112.

[Ado23] Alexandre Adomnicai. *An update on Keccak performance on ARMv7-M*. Cryptology ePrint Archive, Paper 2023/773. 2023. URL: https://eprint.iacr.org/2023/773.