Cryptographic Engineering 2025 – Final Project

September 2025

Submission Deadline (Code & Report): December 08, 2025, 23:59 (Taipei time)
Presentation Slides Deadline: December 15, 2025, 23:59 (Taipei time)
Final Presentations: December 16, 2025
Submission via NTU COOL!
Submission in **groups of two or individual work**!

Assignment

Optimize cryptographic algorithms for Arm Cortex-M4 microcontrollers.
Download the starter code repository from: `https://github.com/mkannwischer/ce2025`

Grading & Submission

Grading Breakdown

Final project = 70% of course grade:

- 50%: Implementation & report
- 20%: Final presentation

Submission Requirements

Each submission must consist of:

1. An archive (`zip`, `tar.bz2`) containing your complete code
2. A `pdf` report (max 20 pages) describing optimization techniques and results

Hardware

Each student will receive an STM32F407 development board for testing their optimized implementations on real hardware. To be eligible for a development board, you must successfully complete Homework 0, which demonstrates that your development environment is working. The development board must be returned at the end of the term.

Important Notes

- All tests must pass on both QEMU and STM32F407 hardware
- Do not modify the provided tests in `test.c` (you may add additional benchmarks or tests)
- Only one person per group should submit (include all names/student IDs)
- Final presentations will include questions about your implementation

Project Overview

The repository contains four cryptographic reference implementations organized into two parts:

Part A: Classical Cryptography

- **SHAKE256**: Extensible-output hash function based on the Keccak permutation (FIPS202)
- **ECDH25519**: Elliptic curve Diffie-Hellman key exchange using Curve25519

Part B: Post-Quantum Cryptography

- **ML-KEM**: Module-Lattice-based Key Encapsulation Mechanism (FIPS203)
- **ML-DSA**: Module-Lattice-based Digital Signature Algorithm (FIPS204)

Completion Requirements: Groups of 2 complete all 4 projects; individual work completes 1 from Part A + 1 from Part B.

Code Requirements

In order to get a passing grade for the Cryptographic Engineering course, the code you submit must fulfill **all the following minimal requirements**:

1. It must not have any secret-dependent branches or access to memory at secret-dependent locations (timing attack resistance).
2. The submitted software must offer the same functionality as the reference implementations, i.e., all tests must pass.
3. The assembly-optimized functions must be faster than the C reference implementations.
4. Meet the specific minimum requirements for each project listed below.

SHAKE256

- Write the Keccak-f[1600] permutation in assembly
- Permutation (binary) code size limit: 4 096 bytes (no round unrolling)
- Must be faster than C reference implementation

ECDH25519

- $<$ 35000000 cycles for ECDH `scalarmult_base` and $<$ 32000000 cycles for ECDH `scalarmult`

ML-KEM

- Write the Number Theoretic Transform (NTT) in assembly
- NTT code size limit: 1 024 bytes (excluding twiddle factors)
- Write at least two other polynomial functions in assembly
- Each assembly function must achieve speedup over C reference

ML-DSA

- Optimize total stack usage of ML-DSA-65 to at most 32 KiB for all operations
- Write at least 2 polynomial functions in assembly
- Each assembly function must achieve speedup over C reference

Evaluation Criteria

Each project has different optimization priorities for achieving a good grade:
SHAKE256

- Primary: Speed (performance improvement over C reference)
- Constraint: Must stay under 4,096 byte code size limit
- Best grades: Maximum speed while remaining under code size limit

ECDH25519

- Primary: Speed
- Best grades: Fastest implementations

ML-KEM

- NTT: Minimize code size + achieve speedup
- Other functions: Speed (performance improvement over C reference)
- Best grades: Compact NTT with good performance + fast other operations

ML-DSA

- NTT: Speed
- Primary: Minimize stack usage
- Speed can be sacrificed for lower stack usage
- Best grades: Lowest stack usage

To obtain a good grade, a comparison with the state-of-the-art will be required for the completed parts.

AI Tools Policy

AI tools are **permitted** with transparency requirements:

- Document exactly how AI was used
- AI-generated code must be understood, tested, and validated by you
- You must be able to explain every part of your code
- Direct copy-paste without understanding is prohibited

Optimization Hints

SHAKE256

- Focus on the Keccak permutation – this is the performance bottleneck
- Consider bit-interleaved representation for efficient rotation operations
- The permutation does not need to be performed in-place
- Avoid unrolling multiple rounds to stay within code size limits

ECDH25519

Before optimizing, investigate and eliminate timing leaks (secret-dependent branches/memory access). Optimization opportunities include:

- Optimized field arithmetic using larger radix (e.g., radix-$2^{25.5}$)
- Specialized base-point scalar multiplication
- Efficient group arithmetic and point representations
- Montgomery ladder for scalar multiplication

ML-KEM

- NTT and Keccak are the most critical function for performance
- Consider polynomial multiplication, addition/subtraction operations
- Look at compression/decompression functions for additional speedup
- Profile the code to identify other bottlenecks
- You may want to use a fast Keccak implementation available in the literature (no code size constraints)

ML-DSA

- Use stack measurement tools to identify high stack usage functions
- Consider in-place operations and buffer reuse
- Optimize polynomial operations for both speed and stack usage
- The NTT and polynomial arithmetic are good assembly targets
- You may want to use a fast Keccak implementation available in the literature (no code size constraints)