2024 International Conference on Consumer Electronics - Taiwan (ICCE-Taiwan) | 979-8-3503-8684-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICCE-Taiwan62264.2024.10674226

Optimizing CRYSTALS-Dilithium in Rust: Radix-4 NTT and Assembly-level Comparison with Official C Implementation

Shunri Kudo*, Yasuyuki Nogami*, Samsul Huda†, and Yuta Kodera*
*Graduate School of Environmental, Life, Natural Science and Technology, Okayama University, Japan
†Green Innovation Center, Okayama University, Japan
shunri_kudo@s.okayama-u.ac.jp {yasuyuki.nogami, shuda, yuta_kodera}@okayama-u.ac.jp

***Abstract*—Post-Quantum Cryptography (PQC) is becoming popular since it offers protection against quantum attacks. CRYSTALS-Dilithium is being considered as a promising NIST PQC standard candidate. It utilizes the Radix-2 Number Theoretic Transform (NTT). Its reliance on lattice-based cryptography and number-theoretic principles positions CRYSTALS-Dilithium as a strong contender for providing robust security in the era of quantum computing. Besides, the Radix-4 NTT divides polynomials into more parts than the Radix-2 NTT, making calculations more efficient by reusing values. In this paper, we improve the computational efficiency of CRYSTALS-Dilithium by implementing the Radix-4 NTT algorithm in both C and Rust programming languages. We verified the effectiveness through experiments in terms of signature and verification speeds. Additionally, we analyzed the assembly code generated by both languages to understand the performance differences.**

***Index Terms*—CRYSTALS-Dilithium, PQC, NTT**

I. Introduction

The emergence of quantum computing poses significant challenges to conventional cryptographic methods like RSA and elliptic curve cryptography. These methods are now at risk from sophisticated algorithms such as Shor's algorithm, which can efficiently break their security measures in polynomial time. In response, PQC offers solutions resistant to quantum attacks [1].

CRYSTALS-Dilithium, also referred to as Dilithium, is emerging as a notable candidate for the NIST Post-Quantum Cryptography (PQC) standard. This lattice-based system demonstrates promise as a PQC candidate by utilizing number-theoretic transforms (NTT) [1]. CRYSTALS-Dilithium utilizes Radix-2 NTT. Its reliance on lattice-based cryptography and number-theoretic principles positions CRYSTALS-Dilithium as a strong contender for providing robust security in the era of quantum computing. Additionally, the Radix-4 NTT sorts polynomials differently from the Radix-2 NTT. It divides them into more parts, making calculations more efficient by reusing values.

In this paper, we enhance the computational efficiency of CRYSTALS-Dilithium by implementing the Radix-4 NTT algorithm in C and Rust. We choose Rust for its memory safety and efficient parallel processing. We evaluate the effectiveness of each implementation and analyze the generated assembly code to understand the performance differences.

Then, we conduct experiments to evaluate the effectiveness of each implementation. Lastly, we analyze the assembly code generated by both languages to understand the performance differences.

II. Preliminaries

In this section, we provide an overview of CRYSTALS-Dilithium and explore the Radix-4 NTT, which is the main focus of our paper.

A. CRYSTALS-Dilithium

CRYSTALS-Dilithium is one of the digital signature schemes selected for standardization in the NIST's project on quantum-resistant cryptography. It is a lattice-based cryptographic scheme that adopts the Module-LWE and SelfTargetMSIS problems as the basis of its security. One of the key features of CRYSTALS-Dilithium, compared to other lattice-based digital signature schemes, is the ease of secure implementation. Specifically, schemes like FALCON and SPHINCS+ require random number generation using the discrete Gaussian distribution, which, if not implemented properly, could pose security risks. However, CRYSTALS-Dilithium uses random number generation based on a uniform distribution, offering an advantage in achieving secure implementations more readily.

1) Module-LWE Problem: The Module-LWE problem is a variant of the learning problem, extending the Learning With Errors (LWE) problem to a modular structure. The LWE problem posits that recovering the original secret information is difficult when small noise is added to random linear equations. In Module-LWE, this concept is expanded to higher dimensions using vectors and matrices.

2) SelfTargetMSIS Problem: The SelfTargetMSIS problem involves finding a solution to a specific product of polynomials computationally difficult.

B. Negative Wrapped Convolution

In CRYSTALS-Dilithium, polynomial multiplication utilizes the modulus polynomial $f(x) = x^n + 1$, enabling the use of a specific method called Negative Wrapped Convolution (NWC). This method allows avoiding redundant reductions. Specifically, NWC operations are defined as follows, where ϕ_{2n} is a primitive $2n$-th root of unity in $\mathbb{Z}_p$, n divides $p-1$:

$$\begin{aligned}
\text{Preprocessing:} \quad & \bar{a}_i = a_i \cdot \phi_{2n}^{i}, \quad \bar{b}_i = b_i \cdot \phi_{2n}^{i} \\
\text{Multiplication:} \quad & \bar{c}_i = \text{INTT}(\text{NTT}(\bar{a}_i) \cdot \text{NTT}(\bar{b}_i)) \\
\text{Postprocessing:} \quad & c_i = \bar{c}_i \cdot \phi_{2n}^{-i}
\end{aligned}$$

Using NWC allows for efficient computation of polynomial ring multiplication. Additionally, the procedures of NWC can be integrated into the preprocessing and postprocessing of transformation methods like Radix-2 NTT or Radix-4 NTT/INTT (Inverse Number Theoretic Transform).

C. Radix-4 NTT

The Radix-4 NTT [2] is a form of Number Theoretic Transform that divides polynomials based on the remainder of their degrees by four. This method increases the number of divisions compared to Radix-2 NTT, enabling the reuse of the same computational values and improving memory and operational efficiency. The transformation formula for Radix-4 NTT, applying NWC, is as follows:

$$\begin{aligned}
A_i &= (F_0 + \phi_{2n}^{2(2i+1)} F_2) + (\phi_{2n}^{2i+1} F_1 + \phi_{2n}^{3(2i+1)} F_3), \\
A_{i+2n/4} &= (F_0 + \phi_{2n}^{2(2i+1)} F_2) - (\phi_{2n}^{2i+1} F_1 + \phi_{2n}^{3(2i+1)} F_3), \\
A_{i+n/4} &= (F_0 - \phi_{2n}^{2(2i+1)} F_2) + \phi_{2n}^{n/2} (\phi_{2n}^{2i+1} F_1 - \phi_{2n}^{3(2i+1)} F_3), \\
A_{i+3n/4} &= (F_0 - \phi_{2n}^{2(2i+1)} F_2) - \phi_{2n}^{n/2} (\phi_{2n}^{2i+1} F_1 - \phi_{2n}^{3(2i+1)} F_3)
\end{aligned}$$

where F_0, F_1, F_2, F_3 are defined as:

$$F_0 = \sum_{j=0}^{n/4-1} a_{4j} \phi_{2n}^{4j(2i+1)}, \quad F_1 = \sum_{j=0}^{n/4-1} a_{4j+1} \phi_{2n}^{4j(2i+1)},$$

$$F_2 = \sum_{j=0}^{n/4-1} a_{4j+2} \phi_{2n}^{4j(2i+1)}, \quad F_3 = \sum_{j=0}^{n/4-1} a_{4j+3} \phi_{2n}^{4j(2i+1)}.$$

III. Experimental Setup and Design

In this research, Docker was employed to facilitate the comparison between C and Rust languages for the construction of the experimental environment. The specifications of the experimental setup are presented in Table I.

TABLE I
EXPERIMENTAL ENVIRONMENT

CPU	Intel(R) Core(TM) i9-11900K @ 3.50GHz
OS	Ubuntu 22.04 (WSL)
Memory	64.0 GB
Container Environment	
OS	Alpine 3.16.8
gcc	11.2.1 (-O3)
cargo	1.74.1
rustup	1.26.0

IV. Experimental Results

The performance comparison between C and Rust implementations of the Radix-4 NTT algorithm highlights the efficiency and speed improvements achieved through Rust. Dilithium offers three security levels, denoted as mode2, mode3, and mode5, which provide increasing levels of security at the cost of increased computational complexity and key/signature sizes. Table II details the signature and verification speeds for these different configurations of the Dilithium algorithm, based on 10,000 iterations of each operation. Table III shows the percentage improvement in performance when transitioning from C to Rust. Upon examining the assembly output for the NTT processing part, two significant factors were identified:

TABLE II
ALGORITHM PERFORMANCE IN C, RUST (RADIX-4 NTT)

Configuration	Signature Speed [μsec]		Verification Speed [μsec]	
	C	Rust	C	Rust
mode2	1.78×10^2	1.50×10^2	0.55×10^2	0.59×10^2
mode3	2.87×10^2	2.40×10^2	0.90×10^2	0.93×10^2
mode5	3.71×10^2	3.13×10^2	1.44×10^2	1.50×10^2

TABLE III
PERFORMANCE IMPROVEMENT FROM C TO RUST (RADIX-4 NTT)

Configuration	Percentage of Reduction Compared to C implementation	
	Signature Speed [%]	Verification Speed [%]
mode2	15.8	-7.6
mode3	16.1	-3.2
mode5	15.7	-3.8

Memory Access: The frequency of ldur (load unscaled register) operations in Rust was about half that of C. This reduction in memory access operations within loop constructs potentially contributes to Rust's improved performance.

Code Volume: Rust's NTT implementation binary was approximately 17,365 bytes, while C's was 27,023 bytes. This 35.7% reduction in code volume reflects Rust's efficiency in compiling more optimized binaries and indicates a more concise expression of the algorithm.

These observations suggest that the Rust language's optimizations, particularly in reducing memory access and generating less code, are key factors in the enhanced performance of the Radix-4 NTT implementation.

V. Conclusion

In this work, the authors implemented Radix-4 NTT algorithm into CRYSTALS-Dilithium to improve the computational efficiency. Then, we evaluated the effectiveness of proposal in both C and Rust programming languages. The results showed an average improvement of 15.9% in signature speeds across all security levels. However, there was a slight decrease in verification speed, averaging 4.8%. Further, a meticulous examination of the assembly output shed light on Rust's advantages, particularly in reducing memory access and the amount of generated code. As future works, our focus will be on enhancing the verification process.

VI. Acknowledgment

This work is a part of cooperate research with the Japan Research Institute Limited.

References

[1] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Dilithium: Algorithm Specifications and Supporting Documentation (Version 3.1)," NIST Post-Quantum Cryptography Standardization Round 3, 2021.

[2] T. -H. Nguyen, B. Kieu-Do-Nguyen, C. -K. Pham and T. -T. Hoang, "High-Speed NTT Accelerator for CRYSTAL-Kyber and CRYSTAL-Dilithium," in IEEE Access, vol. 12, pp. 34918-34930, 2024, doi: 10.1109/ACCESS.2024.3371581.