

Accelerating SMAUG-T using the number theoretic transform on modern high-end CPUs

WooHyung Ko¹  | YoungBeom Kim² | Seog Chung Seo¹

¹Department of Cyber Security, Kookmin University, Seoul, Republic of Korea

²Department of Financial Information Security, Kookmin University, Seoul, Republic of Korea

Correspondence

Seog Chung Seo, Department of Cyber Security, Kookmin University, Seoul, Republic of Korea.

Email: scseo@kookmin.ac.kr

Funding information

Institute of Information & Communications Technology Planning & Evaluation (IITP), Grant/Award Number: RS-2025-02306395

Abstract

We enhance the performance of SMAUG-T, the winning algorithm of South Korea's Post-Quantum Cryptography competition, by applying an NTT-friendly ring homomorphism tailored to x86/64 and AVX2/AVX-512. This paper makes two main contributions: (i) We integrate the NTT into the reference C(x86/64) implementation of SMAUG-T and apply a more efficient NTT-friendly ring homomorphism optimized for parallel processing. Previous attempts have also been made to apply the NTT to SMAUG-T on AVX2; however, they have not addressed the methodology for selecting optimal primes in close conjunction with implementation optimization. (ii) We redesign several state-of-the-art PQC optimization techniques, including layer merging, lazy reduction, shuffle, and negative wrapping convolution, to suit SMAUG-T in parallel environments. Our C-based x86/64 implementation achieves a 209%–287% performance improvement on matrix-vector multiplication compared with the reference, whereas the AVX-512 assembly implementation yields a 75%–78% improvement over AVX2 in the same operation.

KEYWORDS

AVX-512, NTT, polynomial multiplication, post-quantum cryptography, SMAUG-T

1 | INTRODUCTION

With recent advancements in quantum computing and the increasing feasibility of Shor's algorithm, traditional public-key cryptosystems based on the discrete logarithm problem, elliptic curve cryptography, and integer factorization are facing significant threats [1]. In particular, the "Harvest Now, Decrypt Later" strategy underscores the urgent need for quantum-safe security across all layers of network infrastructure. In response, countries around the world are accelerating the institutional adoption of post-

quantum cryptography (PQC). Notably, the United States and Canada have initiated PQC standardization through the National Institute of Standards and Technology (NIST), and similarly, South Korea launched the Korean Post-Quantum Cryptography (KpqC) competition, spearheaded by the National PQC Research Group.

The core requirements for transitioning to quantum-safe cryptography can be summarized into two main factors: (i) *Theoretical security of PQC*: PQC schemes must maintain their security against both quantum and classical adversaries, without succumbing to attacks feasible

within polynomial time. (ii) *Efficiency of PQC*: PQC implementations must be practical for deployment in existing computing environments, and the performance degradation compared with traditional public-key systems must not be excessive.

Following the conclusion of the NIST PQC competition, implementation optimization research on various PQC schemes including Kyber [2] and Dilithium [3] has been pursued actively [4–9]; however, research on KpqC algorithms is relatively lacking. In particular, although SMAUG-T was the winning key encapsulation mechanism (KEM) in the KpqC competition [10], optimization studies targeting SMAUG-T are rare [11, 12]. In this paper, we propose a novel ring homomorphism specialized for the number theoretic transform (NTT) of SMAUG-T and present optimization techniques that utilize modern single instruction, multiple data (SIMD) instruction sets, such as AVX2/AVX-512. Through empirical evaluations, we demonstrate that the proposed method effectively improves the performance of SMAUG-T.

1.1 | Contributions

This paper presents the first AVX-512 implementation of SMAUG-T and explores performance optimizations in C(x86/64) and SIMD environments. The main contributions of this paper can be summarized as follows:

- We demonstrate that superior performance in the matrix-vector multiplication of SMAUG-T on C(x86/64) can be achieved through NTT without relying on Toom-Cook and Karatsuba, contrary to the results of a previous paper [10].
- In SIMD-based optimization, while applying multi-moduli in the same manner as the existing AVX2, we propose a new multi-moduli based on a set of smaller primes more suitable for SMAUG-T.
- We experimentally confirm that the proposed multi-moduli in AVX2 is superior to the existing method and extend it to AVX-512 to present the first AVX-512 implementation of SMAUG-T.
- By considering the architectural characteristics of C(x86/64) and AVX-512, we maximize computational efficiency by applying NTT optimization techniques such as lazy reduction, shuffle, layer merging, and negative wrapping convolution (NWC), together with optimizations considering SMAUG-T parameters.

A detailed evaluation of these contributions is presented in Section 5.2. Our source code is available at <https://github.com/woohyunggo/AVX-512>.

2 | PRELIMINARIES

In this section, we present the preliminaries required to understand our optimization methodologies, which are detailed in Sections 3 and 4. Currently, AVX2 instructions integrated into Intel and AMD CPUs are widely used to optimize cryptographic operations. AVX2 provides 256-bit, 16 SIMD vector registers [13], enabling parallel processing of 8-bit, 16-bit, 32-bit, and 64-bit data elements in a single instruction.

In contrast, AVX-512 provides 512-bit, 32 vector registers [13] along with an extended instruction set, allowing for greater data-level parallelism and computational throughput. These additional instructions enable certain operations to be implemented more simply and efficiently compared with AVX2. The core operation of SMAUG-T is matrix-vector multiplication, where the elements are polynomials. Table 1 presents a brief summary of the AVX2/AVX-512 instructions and functionalities used in this paper for polynomial multiplication optimization.

2.1 | SMAUG-T

SMAUG-T is a lattice-based cryptosystem that was selected as the final algorithm in the PKE and KEM categories of the KpqC competition [10]. This scheme is based on module learning with errors (M-LWE) for public keys and module learning with rounding (M-LWR) for ciphertexts and operates over the ring $R_q := \mathbb{Z}_q[x]/(x^n + 1)$ with modulus q . The parameter sets for each security level are presented in Table 2.

TABLE 1 AVX2/AVX-512 notations [13].

Symbol	Description
<code>vpshld</code>	Logical left shift of packed 32-bit
<code>vpaddw</code>	Adds packed 16-bit
<code>vpsubw</code>	Subtracts packed 16-bit
<code>vpmuldq</code>	Multiplies even-indexed 32-bit to 64-bit
<code>vpmullw</code>	Multiplies packed 16-bit (multiply-low)
<code>vpmulhw</code>	Multiplies packed 16-bit (multiply-high)
<code>vpblendd</code>	Blends packed 32-bit
<code>vmovshdup</code>	Duplicates upper 32-bit floats of 64-bit
<code>vshufi32x4</code>	Shuffles 128-bit lanes from two vectors
<code>vpermq</code>	Permutates 64-bit integers in a vector
<code>vpblendmq</code>	Blends 64-bit integers from two vectors
<code>vpshufd</code>	Shuffles 32-bit integers within a vector
<code>vpblendmw</code>	Blends 16-bit integers from two vectors

TABLE 2 Parameters of SMAUG-T.

Scheme	Security level	q	n	k
SMAUG-T	1	2^{10}	256	2
(NTT-unfriendly ring)	3	2^{11}	256	3
(using TC4 and Kara)	5	2^{11}	256	4

KeyGen(1^λ) :

1 : $\mathbf{A} \leftarrow \mathcal{R}_q^{k \times k}, \mathbf{s} \leftarrow \text{HWT}_{h_s} \in S_\eta^k$

2 : $\mathbf{e} \leftarrow \tilde{D}_\sigma \in \mathcal{R}^k$

3 : $\mathbf{b} \leftarrow -\mathbf{A}^\top \cdot \mathbf{s} + \mathbf{e} \in \mathcal{R}_q^k$

4 : **return** $\text{pk} = (\mathbf{A}, \mathbf{b}), \text{sk} = \mathbf{s}$

Enc(pk, μ) :

$\triangleright \text{pk} = (\mathbf{A}, \mathbf{b}), \mu \in \mathcal{R}_t$

1 : $r \leftarrow \text{spCBD}_r \in S_\eta^k$

2 : $\mathbf{c}_1 = \lfloor p/q \cdot \mathbf{A} \cdot \mathbf{r} \rfloor \in \mathcal{R}_p^k$

3 : $c_2 = \lfloor p'/q \cdot \langle \mathbf{b}, \mathbf{r} \rangle + p'/t \cdot \mu \rfloor \in \mathcal{R}_{p'}$

4 : **return** $\text{ct} = (\mathbf{c}_1, c_2)$

Dec($\text{sk}, \mathbf{c}$) :

$\triangleright \text{sk} = \mathbf{s}, \mathbf{c} = (\mathbf{c}_1, c_2)$

1 : $\mu' = \lfloor t/p \cdot \langle \mathbf{c}_1, \mathbf{s} \rangle + t/p' \cdot c_2 \rfloor \in \mathcal{R}_t$

2 : **return** μ'

FIGURE 1 Description of SMAUG-T.PKE.

TABLE 3 Notation used in this paper.

Symbol	Description
q	Prime modulus used in the reference [10]
q'	Prime modulus used in our methodology
n	Degree of polynomials
k	Dimension of the public matrix $\mathbf{A}$
TC4	4-way Toom-Cook algorithm
Kara	Karatsuba algorithm
NTT	Number theoretic transform
SHAKE $\times 4$	AVX2-based 4-way parallel SHAKE
SHAKE $\times 8$	AVX-512-based 8-way parallel SHAKE

The core operation of SMAUG-T is matrix-vector multiplication between the matrix $\mathbf{A}$ and vectors $\mathbf{s}$ and $\mathbf{r}$ (cf. Figure 1). Here, $\mathbf{A}$ is a square matrix of size $k \times k$, and both $\mathbf{s}$ and $\mathbf{r}$ are vectors of size $k \times 1$. SMAUG-T uses a modulus q of the form 2^k , which enables efficient modular reduction through bitwise operations. However, a modulus of the form 2^k inherently precludes the use of NTT. Table 3 summarizes the notation used in this paper.

2.2 | Number theoretic transform

We refer to three operations of the NTT as NTT, component-wise multiplication (denoted as basemul) ($\circ$), and Inverse NTT (INTT), respectively. The multiplication of two polynomials a and b over $\mathcal{R}_q$ is performed in the NTT domain as $ab = \text{INTT}(\text{NTT}(a) \circ \text{NTT}(b))$.

When $\mathcal{R}_q$ is defined over a cyclotomic polynomial of the form $x^n - 1$, the modulus q must be prime, and a primitive n -th root of unity must exist in $\mathbb{Z}_q$. To satisfy this condition, $q \equiv 1 \pmod{n}$. When $\mathcal{R}_q$ is defined over the cyclotomic polynomial $x^n + 1$, and if q is prime and a primitive $2n$ -th root of unity exists in $\mathbb{Z}_q$, meaning $\omega_{2n}^n \equiv -1 \pmod{q}$ holds. Using this property, $x^n + 1$ can be rewritten as $x^n - \omega_{2n}^n$. By applying the Chinese remainder theorem (CRT), it is possible to construct the following homomorphism [14], where $\text{brv}_n(i)$ denotes the bit reversal of i over $\log_2 n$ bits.

$$\frac{\mathbb{Z}_q[x]}{x^n + 1} \cong \prod_{i=0}^{n-1} \frac{\mathbb{Z}_q[x]}{(x - \omega_{2n}^{2\text{brv}_n(i)+1})}. \quad (1)$$

The two main algorithms for computing radix-2 butterflies are the Cooley–Tukey (CT) and Gentleman–Sande (GS) butterfly [15, 16]. The CT butterfly is used in NTT, where ω is the twiddle factor that maps a pair (a, b) to $(a + b\omega, a - b\omega)$. In contrast, the GS butterfly is used in INTT to map (a, b) and ω^{-1} to $(a + b, (a - b)\omega^{-1})$.

2.3 | NTT multiplication for NTT-unfriendly rings

For lattice-based cryptographic schemes operating over NTT-unfriendly rings, various approaches have been proposed to enable the NTT [4, 8]. For example, when using a non-cyclotomic polynomial or a modulus q that is not a prime number, it is impossible to apply the NTT, and such a ring is called an NTT-unfriendly ring. In SMAUG-T, the ring $\mathcal{R}_q := \mathbb{Z}_q[x]/(x^n + 1)$ with $q = 2^k$ is used to improve the efficiency of polynomial operations. In particular, when the modulus is constrained to $q = 2^k$, fast modular operations can be achieved, but the NTT cannot be applied. In such cases, when the

constraint on q makes it impossible to apply the NTT , it is possible to find an NTT -friendly prime q' and perform the NTT over the ring $R_{q'} = \mathbb{Z}_{q'}[x]/(x^n + 1)$. The condition for an NTT -friendly q' is

$$q' > 2 \cdot n \cdot \frac{q}{2} \cdot k \cdot \frac{\eta}{2}. \quad (2)$$

$2 \cdot n \cdot q/2 \cdot k \cdot \eta/2$ represents the maximum value that can occur in the matrix-vector multiplication process of M-LWE lattice-based algorithms. By selecting an NTT -friendly prime q' greater than this maximum value, even if the matrix-vector multiplication is performed using the NTT over the ring $R_{q'} = \mathbb{Z}_{q'}[x]/(x^n + 1)$, the result remains smaller than q' , which is identical to the result of matrix-vector multiplication performed in the integer domain without modular reduction. Therefore, if the result is ultimately reduced by the original modulus q , the same result as performing over the ring $R_q := \mathbb{Z}_q[x]/(x^n + 1)$ can be obtained.

From an implementation perspective, the polynomial elements in $R_{q'}$ may require significantly larger data representations than those in R_q , which can impose constraints on implementation efficiency depending on the supported instruction sets and processing strategies of the target architecture. In such cases, an alternative approach is to define the ring as $R_{q'_1 q'_2} = \mathbb{Z}_{q'_1 q'_2}[x]/(x^n + 1)$ and utilize the following ring isomorphism. Similar to the conditions mentioned above, $q'_1 q'_2$ must satisfy $q'_1 q'_2 > 2 \cdot n \cdot q/2 \cdot k \cdot \eta/2$, and q'_1 and q'_2 must be coprime or distinct prime numbers. The use of two arbitrary integers is not allowed.

$$\frac{\mathbb{Z}_q[x]}{x^n + 1} \subseteq \frac{\mathbb{Z}_{q'_1 q'_2}[x]}{x^n + 1} \cong \frac{\mathbb{Z}_{q'_1}[x]}{x^n + 1} \times \frac{\mathbb{Z}_{q'_2}[x]}{x^n + 1}. \quad (3)$$

This approach is called multi-moduli [4, 8], where the NTT is first performed over q'_1 and q'_2 . Then, the results are combined using the explicit CRT over $q'_1 q'_2$, and finally, a modular reduction by q is performed to recover the original polynomial [4, 8, 17].

2.4 | Complete versus incomplete NTT

When the base polynomial is $x^n + 1$ and the modulus q is a prime with a primitive $2n$ -th root of unity, a polynomial ring of degree n can be completely decomposed into linear polynomials through $\log_2 n$ layers. This structure is referred to as a complete NTT . However, even if q does not have a primitive $2n$ -th root of unity, partial decomposition is still possible, which is called incomplete NTT . In this case, decomposition can proceed to the maximum

m -layer for which a primitive $2 \cdot 2^m$ -th root of unity exists. As the number of layers decreases, the computational cost of the NTT /INTT is reduced. However, the degree of the polynomials that must be handled in the basemul increases. Specifically, when a ring is decomposed into m layers, basemul performs multiplication of polynomials of degree $2^{(\log_2 n) - m} - 1$.

Therefore, in terms of total computational cost, whether complete NTT or incomplete NTT is more efficient may vary depending on the execution environment, and it is critical to select an architecture-optimized method through profiling.

2.5 | Signed montgomery multiplication

The NTT requires modular multiplication, which is typically performed using Montgomery multiplication to ensure constant-time execution. In SIMD environments such as AVX2/AVX-512, maintaining coefficient packing within vector registers is crucial for achieving optimal performance. However, conventional unsigned Montgomery multiplication requires the storage of a full-precision product, which degrades SIMD packing efficiency. To address this limitation, signed Montgomery multiplication [18] has been proposed, which leverages multiply-high and multiply-low operations to perform modular multiplication while preserving coefficient packing within vector registers. The details of this process are provided in Algorithm 1.

Algorithm 1. Signed Montgomery multiplication [18].

Inputs: a, b . $\triangleright R = 2^k, q^{-1}$ precomputed
Output: $r \equiv abR^{-1} \bmod^{\pm} q$.

- 1: $m \leftarrow bq^{-1} \bmod^{\pm} R$ $\triangleright$ signed multiply-low.
 - 2: $t \leftarrow am \bmod^{\pm} R$ $\triangleright$ signed multiply-low.
 - 3: $u \leftarrow \left\lfloor \frac{ab}{R} \right\rfloor$ $\triangleright$ signed multiply-high.
 - 4: $v \leftarrow \left\lfloor \frac{tq}{R} \right\rfloor$ $\triangleright$ signed multiply-high.
 - 5: $r \leftarrow u - v$ $\triangleright$ one-word subtraction.
-

2.6 | Lazy reduction

In each layer of the NTT /INTT, the CT and GS butterflies include modular multiplication, addition, and subtraction. Therefore, as the layers progress, the coefficients accumulate, and their magnitudes may gradually increase. Lazy reduction to prevent overflow

while minimizing computational overhead does not perform modular reduction at every operation but instead accumulates as many coefficients as possible and then applies only the minimum necessary modular reductions.

In the signed Montgomery-based *NTT*, when processing m layers, the coefficients can increase to $q + m \cdot q$ in the CT butterfly and up to $2^m \cdot q$ in the GS butterfly [19]. This difference originates from the order in which modular multiplication is performed in the two butterfly structures. In the CT butterfly, modular multiplication is performed first, and then addition and subtraction are applied to the modularly reduced result. In contrast, the GS butterfly performs addition and subtraction first, and modular multiplication is applied only to the subtraction result.

2.7 | NWC-based INTT

In the polynomial ring $R_q := \mathbb{Z}_q[x]/(f(x))$, when $f(x) = x^n + 1$ and the *NTT* is applicable, the CT butterfly is generally used in the *NTT*, and only the GS butterfly is used in the *INTT*. In contrast, when $f(x) = x^n - 1$, both the *NTT* and *INTT* can use the CT butterfly [14]. However, by applying the NWC technique, the CT butterfly can be used in the *INTT* even when $f(x) = x^n + 1$. To this end, we let $\psi_n = \omega_{2n}^2 \bmod q$, and define $\Omega_n^{-1} = (1, \omega_{2n}^{-1}, \omega_{2n}^{-2}, \dots, \omega_{2n}^{-(n-1)})$. Then, the following relationship holds:

$$\text{INTT}_{\omega_{2n}}(a) = \Omega_n^{-1} \circ \text{INTT}_{\psi_n}(a). \quad (4)$$

In other words, by treating the structure as if $f(x) = x^n - 1$, performing INTT_{ψ_n} based on ψ_n , and then multiplying the result element-wise by the vector Ω_n^{-1} , one can obtain the same result as the original $\text{INTT}_{\omega_{2n}}$.

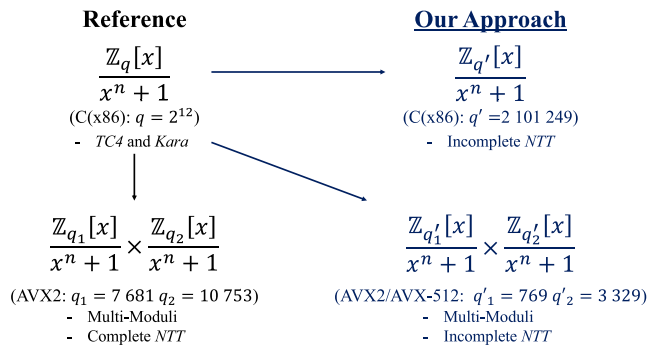


FIGURE 2 Our *NTT* approach.

Therefore, through NWC, even when $f(x) = x^n + 1$, the GS butterfly in the *INTT* can be replaced with the CT butterfly. As described in Section 2.6, this approach mitigates the overhead caused by modular reductions.

Algorithm 2. AVX2/AVX-512 16-bit CT butterfly.

Inputs: a, b . $\triangleright \omega q^{-1}$ precomputed

Output: $a, b = (a + \omega b, a - \omega b)$.

- 1: `vpmullw  $\omega q^{-1}, b, t0$` $\triangleright$ signed multiply-low
- 2: `vpmulhw  $\omega, b, t1$` $\triangleright$ signed multiply-high
- 3: `vpmulhw  $q, t0, t0$` $\triangleright$ signed multiply-high
- 4: `vpsubw  $t1, t0, t0$` $\triangleright$ one-word subtraction
- 5: `vpsubw  $t0, a, b$`
- 6: `vpaddw  $t0, a, a$`

3 | POLYNOMIAL MULTIPLICATION STRATEGY IN SMAUG-T

We apply the following two methods to optimize polynomial multiplication in SMAUG-T for x86/64 and AVX-512:

- Based on the prime $q' = 2^{101} 249$, we replace the original *TC4* and *Kara* used in [10] with *NTT* in our x86/64 implementation (cf. Figure 2).
- We propose a multi-moduli using the new primes $q'_1 = 769$ and $q'_2 = 3329$, and present the first AVX-512 implementation of SMAUG-T (cf. Figure 2). The prime $q'_2 = 3329$ is the same modulus used in Kyber.

AVX-512, like AVX2, does not provide instructions to compute the multiply-high result of a 32-bit multiplication. Therefore, when implementing Signed Montgomery multiplication (cf. Algorithm 1), additional operations are required to obtain the multiply-high, as illustrated in Figure 3. However, AVX2/AVX-512 supports the `vpmulhw` instruction, which can efficiently compute the multiply-high result of 16-bit multiplication (cf. Figure 4). Accordingly, it has been reported that the multi-moduli approach using two 16-bit primes achieves better performance in AVX2 [8]. In this paper, we extend this approach to AVX-512. Algorithm 2 represents a 16-bit CT butterfly implemented using Signed Montgomery multiplication on AVX2/AVX-512.

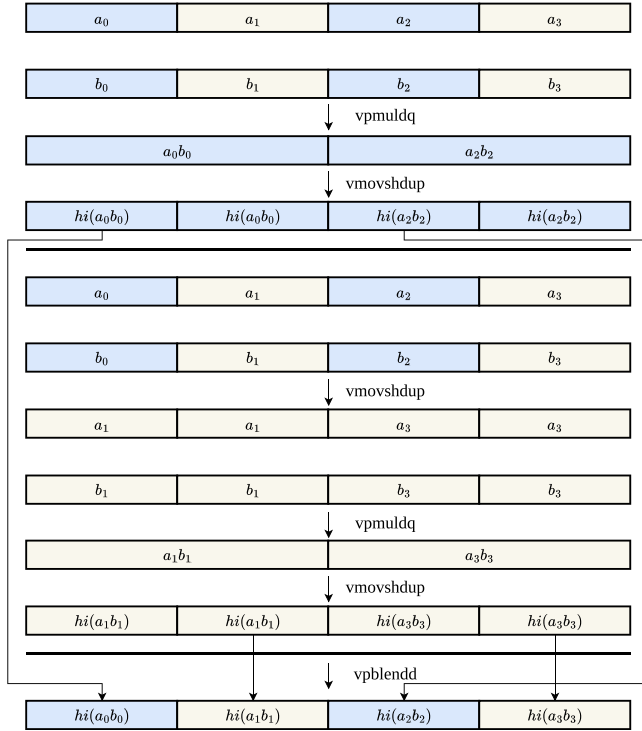


FIGURE 3 Emulating 32-bit multiply-high in AVX2 (illustrated using 128-bit lanes for simplicity).

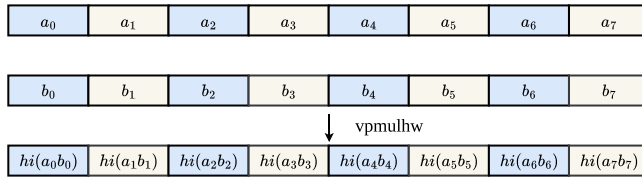


FIGURE 4 Obtaining 16-bit multiply-high in AVX2 (illustrated using 128-bit lanes for simplicity).

3.1 | Prime selection strategy for efficient NTT

Primes are selected by considering the SMAUG-T256 parameters such that the same primes can be used across all security levels, and from the perspective of lazy reduction, modular reduction does not occur in NTT/INTT.

$$q' \text{ and } q'_1 \times q'_2 > 2 \times 256 \times \frac{2048}{2} \times 4 \times \frac{2}{2}. \quad (5)$$

In particular, in the M-LWE-based *NTT*-unfriendly ring, matrix-vector multiplication requires $k^2 + k$ instances of the *NTT*, k instances of the *INTT*, and k^2 instances of basemul. Therefore, eliminating modular reductions in the *NTT*/*INTT* is critical for reducing the overall *NTT* cost.

3.2 | Selection of an incomplete NTT for x86/64

Based on the results of [8], we optimize the polynomial multiplication of SMAUG-T by applying a seven-layer incomplete *NTT* using the prime $q' = 2\,101\,249$ (22-bit). Although a primitive 512-th root of unity $\omega_{512} = 14\,119$ exists for q' , allowing for a complete *NTT*, explicit register control is not possible on the x86/64 architecture, making layer merging inapplicable. Therefore, computing the additional layer introduces unnecessary memory load and store operations for all 256 coefficients, resulting in performance degradation. Consequently, in x86/64-based SMAUG-T, the seven-layer incomplete *NTT* is structurally more efficient than the complete *NTT*, which is confirmed through experimental results. Additionally, in the proposed seven-layer incomplete *NTT* using the prime q' , the accumulated coefficients can grow up to $q' + 7 \cdot q'$ (25-bit) in the CT butterfly-based *NTT* and up to $2^7 \cdot q'$ (29-bit) in the GS butterfly-based *INTT*. Thus, from the perspective of lazy reduction, no modular reduction is required.

3.3 | Selection of optimal primes in multi-moduli for AVX2/AVX-512

The AVX2 implementation of SMAUG-T adopts the multi-moduli using the same two primes as those used in the AVX2 implementation of Saber [20], which was proposed in [8], namely, $q_1 = 7681$ and $q_2 = 10\,753$. For each prime q_1 and q_2 , a primitive 512-th root of unity ω_{512} exists. Therefore, complete *NTT* is performed. On AVX2, complete *NTT* is known to provide better performance than incomplete *NTT* [8]. Additionally, both SMAUG-T and Saber are based on the same polynomial ring structure of the form $\mathbb{Z}_q[x]/(x^{256} + 1)$, and since the maximum coefficient value arising from the matrix-vector multiplication in Saber is greater than that in SMAUG-T, the multi-moduli used in Saber can also be applied to SMAUG-T without theoretical issues. However, as shown in Section 2.6, using smaller primes in *NTT*/*INTT* has clear advantages from the perspective of lazy reduction. In this regard, considering that the maximum coefficient value arising from the matrix-vector multiplication in SMAUG-T is smaller than that in Saber, a new multi-moduli using smaller primes can be sufficiently explored.

For the *NTT*/*INTT* with respect to the primes q_1 and q_2 , from the perspective of lazy reduction, because the CT butterfly is used in the *NTT*, and considering $q_1 + 4 \cdot q_1$ (16-bit) and $q_2 + 3 \cdot q_2$ (16-bit), modular reduction is required for every three layers of q_1 and every two layers of q_2 . In contrast, in *INTT*, because the GS butterfly is

used, and considering $2^3 \cdot q_1$ (16-bit) and $2^2 \cdot q_2$ (16-bit), modular reduction is required every two layers for q_1 and every layer for q_2 . In Saber, at the point where modular reduction is first required, it is not performed directly; rather, the coefficients are twisted into the form $\mathbb{Z}_q[x]/(x^n - 1)$ through modular multiplication, thereby reducing the frequency of modular multiplications and reductions needed in the subsequent layers. However, modular reduction cannot be eliminated completely in the NTT/INTT [8].

To eliminate the performance overhead of modular reductions in the conventional multi-moduli, we propose a new multi-moduli using small primes $q'_1 = 769$ and $q_2 = 3329$. These primes do not have a primitive 512-th root of unity required for a complete NTT, but they do have primitive $2 \cdot 2^7$ -th roots of unity, specifically $\omega_{256} = 7$ for $q'_1 = 769$ and $\omega_{256} = 17$ for $q_2 = 3329$, making a seven-layer incomplete NTT possible. In this case, the accumulated coefficients in the NTT using the CT butterfly are limited to at most $q'_1 + 7 \cdot q'_1$ (13-bit) and $q'_2 + 7 \cdot q'_2$ (15-bit), respectively, meaning modular reductions are not required, even when coefficients are accumulated across all layers. Additionally, for the two primes q'_1 and q'_2 , modular reductions are required in the INTT when using the GS butterfly. However, by applying the NWC technique to use the CT butterfly in the INTT, modular reductions can be completely eliminated in both the NTT and INTT for q'_1 and q'_2 , unlike in the conventional multi-moduli. We confirmed that the multi-moduli based on q'_1 and q'_2 in the AVX2 implementation of SMAUG-T outperforms the conventional multi-moduli, and based on this result, we extend the new multi-moduli to AVX-512.

4 | ADDITIONAL OPTIMIZATION STRATEGIES ON AVX-512

4.1 | Layer merging

In each layer of the NTT/INTT, memory loads and coefficient stores are repeatedly performed to execute butterfly operations, which increases unnecessary memory access in the overall computation process. To reduce this overhead, the layer merging technique [6] reduces unnecessary memory accesses by keeping related coefficients in registers and processing them consecutively, without repeatedly storing/loading the coefficients for each layer.

4.1.1 | AVX2

In the NTT/INTT, eight out of the 16 AVX2 vector registers are used to load and store coefficients. Because each

vector register can store sixteen 16-bit coefficients, using eight registers allows a total of $128 = 2^7$ coefficients to be stored. Therefore, after the first layer, all of the remaining seven layers can be merged for each polynomial ring divided into $\mathbb{Z}_q[x]/(x^{128} - \omega)$ and $\mathbb{Z}_q[x]/(x^{128} + \omega)$. In the existing complete NTT, 1–7 layer merging is applied, which merges seven out of the total eight layers excluding the first [10]. In contrast, for an incomplete NTT with respect to q'_1 and q'_2 , 1–6 layer merging is applied to the NTT/INTT, which consists of seven layers.

4.1.2 | AVX-512

In AVX-512, a single vector register can store 32 16-bit coefficients, meaning eight vector registers are sufficient to store a total of 256 coefficients. Because all 256 coefficients can be stored using only eight registers, full layer merging can be applied to all seven layers of the NTT/INTT for the proposed primes q'_1 and q'_2 . Additionally, because AVX-512 provides twice as many vector registers as AVX2, a total of 512 coefficients for each of q'_1 and q'_2 can be stored using only 16 vector registers, which is the same number of registers required to store 256 coefficients with 32-bit precision, enabling the NTT/INTT for both primes to be performed in parallel within a single computational flow. This approach mitigates the structural drawback of conventional multi-moduli, where separate NTT/INTT executions are required for each prime.

4.2 | Matrix transpose (MT) shuffle for the parallel AVX-512 NTT

In the case of AVX-512, because a single vector register contains 32 16-bit coefficients, butterfly operations are required among the coefficients within the vector register from the fourth layer onward. However, because operations in a SIMD environment are performed on a per-vector basis, there are constraints on performing direct operations between coefficients located at different positions within the same register.

Therefore, to maintain parallelism across all NTT/INTT layers during computation, it is necessary to rearrange the coefficients across the vector registers, which is referred to as a shuffle. In this paper, we adopt the MT-shuffle technique proposed in previous studies [9, 18] and extend it to AVX-512. In AVX-512, 256 coefficients are stored using eight registers, which can be regarded as an 8×32 matrix $\mathbf{M}$. Over the course of five steps, in the NTT, the coefficients across vector registers are rearranged as $\mathbf{M} \rightarrow \mathbf{M}^T$, whereas in the INTT, an inverse operation is performed to rearrange the coefficients as $\mathbf{M}^T \rightarrow \mathbf{M}$. In

this manner, efficient computation is achieved while maintaining parallelism across all layers of the NTT/INTT. The detailed operation of the MT-shuffle is presented in Appendix A (please see Figure A1).

4.3 | Layer-specific simplification of CT butterfly

In SMAUG-T, **s** and **r** are ternary polynomials. Therefore, in the first layer of the NTT for **s** and **r**, the CT butterfly does not require modular multiplication. To exploit this property, in the CT butterfly of the first NTT layer, the existing modular multiplication (Algorithm 1) is replaced with a single `vpnullw` instruction. Consequently, the 128 modular multiplications in the first NTT layer are simplified.

Additionally, we optimize the CT butterfly in the INTT using the NWC technique. In the NTT/INTT over $R_q := \mathbb{Z}_q[x]/(x^{256} - 1)$, the structural property ensures that the first twiddle factor of each layer is always 1. Therefore, in the INTT, a CT butterfly with the twiddle factor 1 is executed using only addition and subtraction. Because the CT butterfly is performed in vector units, this optimization is applied only to the first three layers. Consequently, in the matrix-vector multiplication of the multi-moduli approach based on q'_1 and q'_2 , in the NTT, a total of $2 \cdot (k^2 + k) \cdot 128$ modular multiplications are simplified, and in the INTT, $2 \cdot k \cdot 224$ modular multiplications are eliminated.

4.4 | Hashing

To enhance the computational efficiency of the Keccak-based hash function, we integrated the 4-way and 8-way AVX-512 parallel implementations from the XKCP library [21] into SMAUG-T. Because the sampling procedures of SMAUG-T (Uniform, Hamming weight [HWT], centered binomial distribution [CBD], and Gaussian) require output buffers proportional to the sizes of the corresponding matrices and vectors, we integrated $SHAKE \times 4$ and $SHAKE \times 8$ as shown in Table 4. In the existing AVX2 implementation, HWT sampling does not have a direct comparison instruction for unsigned 16-bit integers (`uint16_t`). Therefore, conversion to 32-bit integers (`uint32_t`) and additional masking are required. Consequently, the packing efficiency inside the vector registers is reduced by half and approximately 12 additional instructions are generated. We employed AVX-512 intrinsic `_mm512_mask_cmpgt_epu16_mask()` [22] to perform comparison and masking efficiently while maintaining packing inside the vector registers without 32-bit widening. Additionally, in the

TABLE 4 SMAUG-T SHAKE parallelism: SHAKE-128 is used for sampling **A**, and SHAKE-256 is used for **s**, **e**, and **r**.

Level	k	A	s	e	r
128	2	4-way	4-way	4-way	4-way
192	3	8-way + 1-way	4-way	4-way	4-way
256	4	2×8 -way	4-way	4-way	4-way

existing AVX2 implementation, the CBD sampling of SMAUG-T192 is not parallelized; however, we performed AVX-512-based parallel implementation for all security levels. Through these optimizations, we observed up to 77%, 14%, and 200% performance improvements in Uniform, HWT, and CBD sampling, respectively, compared with the existing AVX2 performance.

4.5 | Parallel packing/unpacking

In the existing AVX2 implementation of SMAUG-T, parallelization for packing and unpacking is applied only to `pk`. Consequently, the packing and unpacking of `sk` and `ct` still rely on a C-based implementation, leaving room for optimization, so we parallelized `sk` and `ct`. Because packing/unpacking compresses or restores consecutive coefficients at the byte level, we apply the MT-shuffle, just as in the NTT/INTT, when necessary, depending on the packing method. Consequently, `sk` achieved up to a 460% performance improvement, and `ct` achieved up to a 67% improvement compared with the existing implementation.

5 | EXPERIMENTAL RESULTS

5.1 | Benchmarking setup

Performance was measured using an Intel i9-11900 K 3.50 GHz CPU and the gcc 11.4.0 compiler with the `-O3` optimization flag. Hyper-Threading and Turbo Boost were disabled during our experiments to ensure consistent performance measurements.

5.2 | Performance analysis

We replaced the original polynomial multiplication over an *NTT*-unfriendly ring with an *NTT*, optimizing the core operation of SMAUG-T and improving overall computational efficiency. Specifically, by applying an incomplete *NTT* based on the prime $q' = 2101249$ on x86/64, we achieved a 209%–287% performance improvement for matrix-vector multiplication, resulting in performance

TABLE 5 Operation costs (in cycles) for SMAUG-T on C(x86/64) and AVX2/AVX-512 (avg. over 10 000 iterations).

	SMAUG-T128			SMAUG-T192			SMAUG-T256		
	[10] (C) [1024]	Our (C) [2101249]	Speedup	[10] (C) [2048]	Our (C) [2101249]	Speedup	[10] (C) [2048]	Our (C) [2 101 249]	Speedup
$-A^T s$	132 752	43 022	+209%	297 804	82 934	+259%	527 632	138 304	+282%
Ar	131 848	42 218	+212%	296 330	82 506	+259%	525 968	135 838	+287%
br	66 712	17 974	+271%	99 816	24 436	+308%	132 626	30 834	+330%
$c_1 s$	66 712	25 388	+163%	99 816	35 164	+184%	132 626	45 288	+193%
$-A^T s$	[10] (AVX2) [7681 × 10 753]	Our (AVX2) [769 × 3329]	Speedup	[10] (AVX2) [7681 × 10 753]	Our (AVX2) [769 × 3329]	Speedup	[10] (AVX2) [7681 × 10 753]	Our (AVX2) [769 × 3329]	Speedup
$-A^T s$	4 962	4584	+8%	9474	8716	+9%	15 284	14 140	+8%
Ar	4892	4528	+8%	9322	8680	+7%	15 004	14 114	+6%
br	1906	1830	+4%	2566	2440	+5%	3200	3074	+4%
$c_1 s$	2986	2712	+10%	4242	3762	+13%	5404	4814	+12%
$-A^T s$	[10] (AVX2) [7681 × 10 753]	Our (AVX-512) [769 × 3329]	Speedup	[10] (AVX2) [7681 × 10 753]	Our (AVX-512) [769 × 3329]	Speedup	[10] (AVX2) [7681 × 10 753]	Our (AVX-512) [769 × 3329]	Speedup
$-A^T s$	4962	2788	+78%	9474	5328	+78%	15 284	8660	+76%
Ar	4892	2796	+75%	9322	5326	+75%	15 004	8598	+75%
br	1906	1138	+67%	2566	1586	+62%	3200	2042	+57%
$c_1 s$	2986	1676	+78%	4242	2388	+78%	5404	3096	+75%

Note: Brackets “[.]” denote the modulus q used in each operation.

improvements of 85% – 128%, 138% – 168%, and 130% – 158% on KeyGen, Encaps, and Decaps, respectively (cf. Tables 5 and 6). We also designed a new multi-moduli based on the primes $q'_1 = 769, q'_2 = 3329$ considering the characteristics of SMAUG-T, and conducted further optimization by combining the latest *NTT* techniques with optimizations reflecting the parameters of SMAUG-T. As a result, the proposed AVX-512 implementation achieved a 75% – 78% performance improvement in matrix-vector multiplication compared with the existing AVX2 implementation. Additionally, the AVX2 implementation with the proposed primes exhibited a 6% – 9% improvement over the original AVX2, confirming the validity of the proposed approach (cf. Table 5).

This paper represents the first implementation of SMAUG-T on AVX-512, integrating optimized parallel hash functions from the XKCP library to significantly reduce computational cost. We also extended parallelization to the packing and unpacking of sk and ct , thereby expanding the parallel optimization previously applied only to pk for the entire SMAUG-T. Through these optimizations, we refined SMAUG-T for AVX-512 and achieved performance improvements of 18% – 40%, 38% – 63%, and 50% – 66% on KeyGen, Encaps, and Decaps, respectively (cf. Table 6).

5.3 | Discussion

The existing C(x86/64) implementation of SMAUG-T showed a lower performance than the C(x86/64) implementation of Kyber. However, in this paper, we replace the *TC4*- and *Kara*-based multiplication with *NTT*,

thereby improving the C(x86/64) implementation of SMAUG-T to approach the performance of Kyber C(x86/64) (cf. Table 6).

In AVX2, SMAUG-T also performs *NTT*-based multiplication through multi-moduli, but Kyber still achieves superior performance (cf. Table 6). Although these two algorithms are structurally different, if we examine only the commonly included matrix-vector multiplication, the operation counts are as follows. Kyber is an *NTT*-friendly algorithm based on a single 16-bit prime, and it is assumed that the public matrix $\mathbf{A}$ is located in the *NTT* domain. In particular, for matrix-vector multiplication, Kyber requires k instances of *NTT*, k instances of *INTT*, and k^2 instances of *basemul*. However, because the 32-bit *NTT* is inefficient in AVX2/AVX-512 for SMAUG-T, a multi-moduli approach utilizing two 16-bit primes is applied. Consequently, for matrix-vector multiplication, SMAUG-T requires $2 \cdot (k^2 + k)$ instances of *NTT*, $2 \cdot k$ instances of *INTT*, $2 \cdot k^2$ instances of *basemul*, and k instances of *CRT*, which requires more operations compared with Kyber (cf. Table 7).

Because an AVX-512 implementation in Kyber does not yet exist, an equivalent comparison is difficult.

TABLE 7 Comparison of the number of *NTT*, *INTT*, *basemul*, and *CRT* operations required for matrix-vector multiplication in Kyber and SMAUG-T.

Scheme	<i>NTT</i>	<i>INTT</i>	<i>basemul</i>	<i>CRT</i>
Kyber	k	k	k^2	-
SMAUG-T	$2 \cdot (k^2 + k)$	$2 \cdot k$	$2 \cdot k^2$	k

TABLE 6 SMAUG-T (C(x86/64), AVX2/AVX-512) and Kyber (C(x86/64), AVX2) cycles (avg. over 10 000 iterations).

	SMAUG-T128			SMAUG-T192			SMAUG-T256		
	[10] (C)	Our (C)	Speedup	[10] (C)	Our (C)	Speedup	[10] (C)	Our (C)	Speedup
K	196 432	106 150	+85%	415 470	199 900	+108%	704 914	309 216	+128%
E	23 9874	100 794	+138%	480 362	189 006	+154%	793 000	295 398	+168%
D	320 682	139694	+130%	602 314	245 928	+145%	955 734	369 750	+158%
	[10] (AVX2)	Our (AVX-512)	Speedup	[10] (AVX2)	Our (AVX-512)	Speedup	[10] (AVX2)	Our (AVX-512)	Speedup
	[10] (C)	[2] (AVX2)	Speedup	[2] (C)	[2] (AVX2)	Speedup	[2] (C)	[2] (AVX2)	Speedup
K	29 462	24 894	+18%	46 522	34 958	+33%	61 084	43 736	+40%
E	21 416	15 530	+38%	41 166	25 818	+59%	55 058	33 706	+63%
D	32 676	21 782	+50%	57 220	35 144	+63%	76 806	46 178	+66%
	Kyber512			Kyber768			Kyber1024		
	[2] (C)	[2] (AVX2)	Speedup	[2] (C)	[2] (AVX2)	Speedup	[2] (C)	[2] (AVX2)	Speedup
K	105 306	20 342	+418%	176 556	33 524	+427%	302 596	44 808	+575%
E	122 824	21 654	+467%	192 708	32 986	+484%	321 490	45 418	+608%
D	161 746	23 386	+592%	243 878	35 680	+584%	395 434	49 686	+696%

Regardless, the purpose of this paper is to demonstrate that, by considering the structural constraints of SMAUG-T, optimization can achieve a certain level of performance. In this context, the SMAUG-T AVX-512 implementation presented in this paper achieved superior performance on Encaps and Decaps compared with Kyber AVX2 at all security levels. At security level 5, it also achieved faster results on KeyGen.

6 | CONCLUSION

This paper presented the first implementation of SMAUG-T on AVX-512 and highlighted improved performance based on NTT optimizations, including the proposed modulus design and parameter-based refinements. The proposed optimizations are not limited to a specific environment and can also be applied to various embedded and SIMD platforms, such as the Cortex-A series and Armv8 NEON. In particular, because SMAUG-T was the winning algorithm at the KpqC competition and has high potential for future use, continuous optimization research is necessary. Future work will involve applying these techniques to various devices and evaluating their performance to assess the generality of the proposed approach.

ACKNOWLEDGMENTS

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government, Ministry of Science and ICT (MSIT) (No. RS-2025-02306395, Development and Demonstration of PQC-Based Joint Certificate PKI Technology).

CONFLICT OF INTEREST STATEMENT

The authors declare no conflicts of interest.

ORCID

WooHyung Ko  <https://orcid.org/0009-0006-6245-7665>

REFERENCES

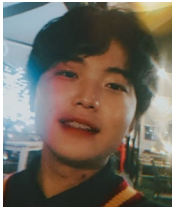
1. P. W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM Review **41** (1999), no. 2, 303–332.
2. J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, *CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM*, (2018 IEEE European Symposium on Security and Privacy (EuroS&P), London, UK), 2018, pp. 353–367.
3. L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, *CRYSTALS-Dilithium: a lattice-based digital signature scheme*, IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018** (2018), 238–268.
4. A. Abdulrahman, J. P. Chen, Y. J. Chen, V. Hwang, M. J. Kannwischer, and B. Y. Yang, *Multi-moduli NTTs for Saber on Cortex-M3 and Cortex-M4*, IACR Trans. Cryptogr. Hardw. Embed. Syst. **2022** (2022), 127–151.
5. A. Abdulrahman, V. Hwang, M. J. Kannwischer, and A. Sprenkels, *Faster Kyber and Dilithium on the Cortex-M4*, (International Conference on Applied Cryptography and Network Security), 2022, pp. 853–871.
6. E. Alkim, Y. A. Bilgin, M. Cenk, and F. Gérard, *Cortex-M4 optimizations for $\{R, M\}$ LWE schemes*, IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020** (2020), 336–357.
7. H. Becker, V. Hwang, M. J. Kannwischer, B.-Y. Yang, and S.-Y. Yang, *Neon NTT: faster Dilithium, Kyber, and Saber on Cortex-A72 and Apple M1*, Cryptology ePrint Archive (2021).
8. C.-M. M. Chung, V. Hwang, M. J. Kannwischer, G. Seiler, C. J. Shih, and B. Y. Yang, *NTT multiplication for NTT-unfriendly rings: new speed records for Saber and NTRU on Cortex-M4 and AVX2*, IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021** (2021), 159–188.
9. R. Xu, D. He, M. Luo, C. Peng, and X. Zeng, *Optimizing Dilithium implementation with AVX2/-512*, ACM Trans. Embed. Comput. Syst. **23** (2024), no. 6, 1–30.
10. J. H. Cheon, H. Choe, J. Seo, and H. Seong, *SMAUG (-T), revisited: timing-secure, more compact, less failure*, IEEE Access **12** (2024), 188386–188397.
11. M. Kim, D. Shin, W. Ko, Y. Kim, and S. C. Seo, *Optimized implementation of SMAUG-T on resource-constrained 16-bit MSP430 MCU*, ICT Express **11** (2025), no. 5, 851–857.
12. O. Narlı, M. K. Pehlivanoglu, and S. Akleyek, *A new optimized implementation of SMAUG-T for lightweight devices*, (Nordic Conference on Secure It Systems, Tartu, Estonia), 2025, pp. 77–90.
13. Intel® 64 and IA-32 architectures software developer's manual, 2025. <https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html>
14. Z. Liang and Y. Zhao, *Number theoretic transform and its applications in lattice-based cryptosystems: a survey*, arXiv preprint (2022). DOI [10.48550/arXiv.2211.13546](https://doi.org/10.48550/arXiv.2211.13546)
15. J. W. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. Comput. **19** (1965), no. 90, 297–301.
16. W. M. Gentleman and G. Sande, *Fast Fourier transforms: for fun and profit*, (Proc. Fall Joint Comput. Conf., San Francisco, CA, USA), 1966, pp. 563–578.
17. D. Bernstein and J. Sorenson, *Modular exponentiation via the explicit Chinese remainder theorem*, Math. Comput. **76** (2007), no. 257, 443–454.
18. G. Seiler, *Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography*, Cryptology ePrint Archive, 2018.
19. J. Huang, J. Zhang, H. Zhao, Z. Liu, R. C. C. Cheung, C. K. Koç, and D. Chen, *Improved Plantard arithmetic for lattice-based cryptography*, IACR Trans. Cryptogr. Hardw. Embed. Syst. **4** (2022), 614–636.
20. J.-P. D'Anvers, A. Karmakar, S. Sinha Roy, and F. Vercauteren, *Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM*, (Marrakesh, Morocco, International Conference on Cryptology in Africa, Marrakesh, Morocco), 2018, pp. 282–305.

21. G. Bertoni, J. Daemen, S. Hoffert, M. Peeters, G. V. Assche, and R. V. Keer, *XKCP: eXtended Keccak Code Package*, 2024. <https://github.com/XKCP/XKCP>
22. Intel Corporation, Intel® Intrinsic Guide, 2024. <https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html>

AUTHOR BIOGRAPHIES



WooHyung Ko received his BS degree from the Department of Information Security, Cryptology, and Mathematics at Kookmin University, Seoul, Republic of Korea, in 2021. He is currently pursuing an MS degree in the Department of Cyber Security at Kookmin University. His research interests include the optimization of cryptographic algorithms, the implementation of post-quantum cryptography, and module validation programs.



YoungBeom Kim received his BS degree from the Department of Information Security, Cryptology, and Mathematics at Kookmin University, Seoul, Republic of Korea, in 2021, and his MS degree in Financial Information Security from the same university in 2023. He is currently pursuing a PhD at Kookmin University. His research interests include the optimization of cryptographic algorithms, post-quantum cryptography implementation, module validation programs, and SW/HW co-design for IoT devices.



Seog Chung Seo received the BS degree in Information and Computer Engineering from Ajou University, Suwon, Republic of Korea, and the MS degree in Information and Communications from the Gwangju Institute of Science and Technology (GIST), Gwangju, Republic of Korea, in 2005 and 2007, respectively. He received the PhD degree from Korea University, Seoul, Republic of Korea, in 2011. He worked as a research staff member at SAIT (Samsung Advanced Institute of Technology) and the Samsung DMC Research and Development Center from September 2011 to April 2014. He was a senior research member at the Affiliated Institute of ETRI, Republic of Korea, from 2014 to 2018. Currently, he works as an associate professor at Kookmin University, Republic of Korea. His research interests include public-key cryptography, its efficient implementation on various IT devices, the cryptographic module validation program, network security, and data authentication algorithms.

How to cite this article: W. Ko, Y. Kim, and S. Chung Seo, *Accelerating SMAUG-T using the number theoretic transform on modern high-end CPUs*, ETRI Journal (2026), 1–13, DOI [10.4218/etrij.2025-0418](https://doi.org/10.4218/etrij.2025-0418).

APPENDIX A

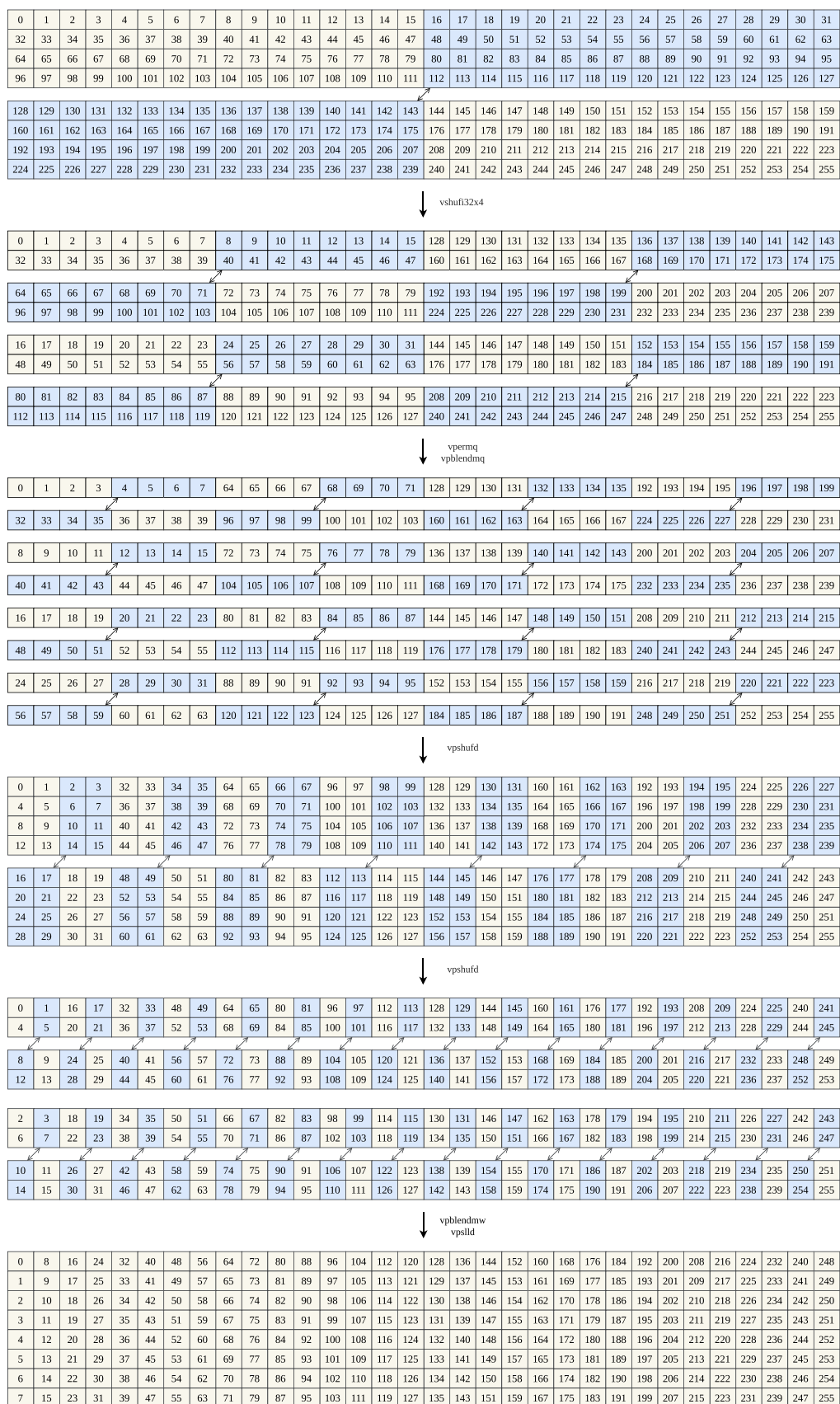


FIGURE A1 16-bit MT-shuffle process on AVX-512.