



Optimized implementation of SMAUG-T on resource-constrained 16-bit MSP430 MCU

MinGi Kim ^a, DongHyun Shin ^a, WooHyung Ko ^a, YoungBeom Kim ^b, Seog Chung Seo ^a,*

^a Department of Cybersecurity, Kookmin University, Seoul, South Korea

^b Department of Financial Information Security, Kookmin University, Seoul, South Korea

ARTICLE INFO

Keywords:

Post-quantum Cryptography
SMAUG-T
Polynomial multiplication
NTT
MSP430

ABSTRACT

In this paper, we present an optimized implementation of SMAUG-T, one of Round 2 Key Encapsulation Mechanism algorithms in Korean Post-quantum Cryptography Competition, on a widely used 16-bit MSP430 MCU. To achieve performance efficiency of polynomial multiplication, one of the most time-consuming operations in SMAUG-T, we find the optimal method by investigating several latest algorithms such as the Toom–Cook method and the Number-Theoretic Transform (NTT)-based methods (32-bit single moduli version and 16-bit multi-moduli version). Through the investigation, we found that 32-bit single moduli version is the best approach for polynomial multiplication in SMAUG-T on 16-bit MSP430 MCU. To enhance the performance of NTT-based polynomial multiplication, we proposed an improved 32-bit signed Montgomery multiplication method with a newly found Montgomery prime (0x250001) and the intrinsic hardware multiplier. We also apply the state-of-the-art techniques for NTT and inverse NTT (iNTT) such as the layer merging, CT butterfly by tuning them proper to the target device. As a result, our NTT implementation achieves around 35% of improved performance compared to the previous best result of 32-bit single moduli version implementation proposed for Dilithium on 16-bit MSP430 MCU. Finally, our SMAUG-T implementation with the proposed NTT implementation provides 43%–63%, 92%–99%, and 85%–95% of improved performance for key generation, encapsulation, and decapsulation compared to the reference implementation, respectively.

1. Introduction

With the advancement of quantum computing, the transition from traditional public-key cryptography to Post-Quantum Cryptography (PQC) has become essential [1]. In response, NIST recently selected four algorithms as PQC standards. Among these, Kyber [2], Dilithium [3], and Falcon [4] are lattice-based cryptographic schemes. The core operation of these schemes is polynomial multiplication, and significant optimization efforts have been made for mid-range embedded devices such as 32-bit Cortex-M3/M4 [5–7]. However, research on 8/16-bit processors remains insufficient. As highlighted in the Third PQC Standardization Conference [8] and recent surveys [9], large-scale sensor networks, such as those used in military applications or wild-fire detection systems, require low-cost and low-power architectures. Consequently, the need for the implementation of PQC in low-end devices is becoming increasingly critical [10]. However, research on low-end devices remains scarce compared to that on high-performance devices, and optimization strategies differ significantly. Therefore, it is necessary to reassess current implementation methodologies for these

constrained environments. For example, [11] redesigned state-of-the-art optimization strategies for Dilithium to adapt them to the MSP430 environment. They optimized the 32-bit signed Montgomery reduction by leveraging the MSP430's hardware multiplier.

South Korea is also hosting its own PQC competition (KpQC), currently in Round 2. At this juncture, we emphasize the necessity of PQC implementation on low-end devices and advocate for performance evaluation of KpQC algorithms in constrained environments. Among the KpQC Round 2 KEM algorithms, SMAUG-T demonstrates $1.6 \times$ (Encapsulation) and $1.4 \times$ (Decapsulation) faster performance compared to Kyber in x86/64 environments [12]. However, our performance evaluation on 16-bit low-end devices reveals that SMAUG-T performs slower than Kyber (refer to Table 4). This outcome indicates that the ordinary implementation of SMAUG-T does not provide satisfactory performance on resource-constrained IoT devices where typically multiplication instructions are computationally expensive. This is because SMAUG-T uses NTT-unfriendly ring, which cannot perform NTT-based

* Corresponding author.

E-mail addresses: mingi1684@kookmin.ac.kr (M. Kim), dkldkl13@kookmin.ac.kr (D. Shin), ohnb2974@kookmin.ac.kr (W. Ko), darania@kookmin.ac.kr (Y. Kim), scseo@kookmin.ac.kr (S.C. Seo).

<https://doi.org/10.1016/j.ict.2025.07.007>

Received 13 January 2025; Received in revised form 26 May 2025; Accepted 21 July 2025

Available online 5 August 2025

2405-9595/© 2025 The Authors. Published by Elsevier B.V. on behalf of The Korean Institute of Communications and Information Sciences. This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

multiplication and thus requires a number of modular multiplications compared to Kyber.

Previous studies [6,13] have shown that NTT-based multiplication outperforms traditional multiplication methods for NTT-unfriendly algorithms on mid-range embedded devices. Specifically, a 32-bit single-moduli NTT was found to be optimal for Cortex-M4, while a 16-bit multi-moduli NTT was optimal for AVX2 platforms. These results were achieved by carefully considering the platform-specific characteristics, the cost of multiplication instructions, and the number of multiplications required. In this work, we revisit these methodologies from the perspective of the MSP430 to apply NTT to SMAUG-T.

Contributions.

Platform-optimal NTT selection. Leveraging the fact that 32-bit multiplication on a 16-bit MSP430 incurs twice the computational cost compared to 16-bit multiplication, we propose that a 32-bit single-moduli NTT is optimal for this platform. When the computational overhead of a 16-bit NTT is considered as 1, a 16-bit multi-moduli NTT requires at least twice the overhead due to the additional cost of the Chinese Remainder Theorem (CRT) operation [6]. Unlike previous studies, we further demonstrate that a complete NTT approach is the optimal scenario in this context.

Prime selection for modular multiplication. We propose a prime selection method to optimize modular multiplication, a core operation in NTT. Using our proposed prime, 0x250001, the clock cycles for signed Montgomery reduction are reduced from 62 to 49, resulting in a 13-cycle improvement, which contributes to a 35% faster performance in NTT, compared to the general implementation in [11].

MSP430-Specific NTT Customization. We customize state-of-the-art NTT implementation techniques for the MSP430 environment. We develop an optimized NTT layer with a merging technique suited for the MSP430 architecture. In addition, we adopt a Cooley–Tukey (CT) butterfly method in the iNTT to further improve efficiency.

Applying these methodologies to SMAUG-T, we demonstrate a 143% performance improvement in polynomial multiplication compared to reference implementation using the Toom–Cook and Karatsuba algorithms. Ultimately, our approach achieves performance improvements of up to 43%–63%, 92%–99%, and 85%–95% in the key generation, encapsulation, and decapsulation processes of SMAUG-T.

2. Preliminary

2.1. SMAUG-T

SMAUG-T [12] is a lattice-based key encapsulation mechanism based on the Module Learning With Errors (M-LWE) and Module Learning With Rounding (M-LWR) problems. Fig. 1 illustrates the public key encryption (PKE) scheme of SMAUG-T. The polynomial ring used within SMAUG-T is $R_q = \mathbb{Z}_q[x]/(X^n + 1)$ with $q = 2^{10}$ in SMAUG-T128 and $q = 2^{11}$ in SMAUG-T{192, 256}. The dimension n is 256 for all parameter sets. The public matrix $\mathbf{A}$ is structured as an $l \times l$ matrix, while the secret vectors $\mathbf{s}$ and $\mathbf{r}$ are $l \times 1$ column vectors. The secret distribution parameter η determines the coefficient bounds of the secret vectors $\mathbf{s}$ and $\mathbf{r}$. The parameter sets for each security level of SMAUG-T are summarized in Table 1. As the modulus q is a power of two, which is not prime, NTT-based multiplication cannot be applied. Consequently, the most time-consuming operation, polynomial multiplication, is performed using a combination of Toom–Cook and Karatsuba multiplication. There are two matrix–vector multiplication of polynomials $-\mathbf{A}^\top \cdot \mathbf{s}$ and $\mathbf{A} \cdot \mathbf{r}$, and two inner product $\langle \mathbf{b}, \mathbf{r} \rangle$ and $\langle \mathbf{c}_1, \mathbf{s} \rangle$.

```

KeyGen( $1^\lambda$ ) :
1 :  $\mathbf{A} \leftarrow \mathcal{R}_q^{k \times k}, \mathbf{s} \leftarrow \text{HWT}_{h_s} \in S_\eta^k$ 
2 :  $\mathbf{e} \leftarrow \widetilde{D}_\sigma \in \mathcal{R}^k$ 
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( $\text{pk}, \mu$ ) :  $\triangleright \text{pk} = (\mathbf{A}, \mathbf{b}), \mu \in \mathcal{R}_t$ 
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( $\text{sk}, \text{ct}$ ) :  $\triangleright \text{sk} = \mathbf{s}, \text{ct} = (\mathbf{c}_1, c_2)$ 
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  $\mu'$ 

```

Fig. 1. Description of SMAUG-T.PKE.

Table 1
SMAUG-T parameter sets.

Name	l	n	q	η
SMAUG-T128	2	256	2^{10}	1
SMAUG-T192	3	256	2^{11}	1
SMAUG-T256	4	256	2^{11}	1

2.2. Number theoretic transform (NTT)

NTT is a specific form of the Discrete Fourier Transform (DFT) that defines an isomorphism over a finite field. Performing NTT in the finite field $\mathbb{Z}_q/f(x)$ requires q to be a prime number. Additionally, when $f(x) = X^n + 1$, n must satisfy the existence of a primitive $2n$ -th root of unity ψ in $\mathbb{Z}_q$, which exists if and only if $2n \mid (q - 1)$. Using the Chinese Remainder Theorem (CRT), an algebraic isomorphism can be established as follows:

$$\frac{R[x]}{\langle x^n - \psi^n \rangle} \cong \prod \frac{R[x]}{\langle x^{\frac{n}{2}} \pm \psi^{\frac{n}{2}} \rangle} \cong \cdots \cong \prod_{i=0}^{n-1} \frac{R[x]}{\langle x - \psi^{2brv(i)+1} \rangle}.$$

The computational complexity of NTT is $\mathcal{O}(n \cdot \log n)$, as point-wise multiplication of the transformed polynomials involves n operations. When the divided subrings are linear (monomials), this is referred to as the complete NTT. Otherwise, it is called the incomplete NTT.

2.3. MSP430

MSP430 is an ultra-low power MCU based on a 16-bit RISC CPU by Texas Instruments [14]. It has 16 registers numbered R0–R15, where R4–R15 serve as general-purpose registers (gpr). Operations are performed in 16-bit units, matching the register size. MSP430 features a hardware multiplier that executes multiplications instantly upon writing to dedicated registers, without additional cycles. Algorithm 1 and Algorithm 2 illustrate 16-bit and 32-bit multiplication, respectively, showing that 32-bit operations take twice the clock cycles. While hardware-specific registers can be accessed like general-purpose ones, this incurs extra cycles.

3. Efficient polynomial multiplication methods

Research has been conducted on applying NTT-based multiplication to efficiently perform polynomial multiplication in NTT-unfriendly algorithms [6,13,15–17]. The superior performance of NTT has already

Algorithm 1 16×16-bit multiplication on MSP430

Require: 16-bit a, b
Ensure: r4, r5 = $a \cdot b$
1: mov a_0 , &MPY
2: mov b_0 , &OP2 ▷ $a \cdot b$ in multiplier
3: mov &RESLO, r4
4: mov &RESHI, r5 ▷ move the multiplier to gpr

Algorithm 2 32 × 32-bit multiplication on MSP430

Require: 32-bit $a = (a_1 || a_0)$, $b = (b_1 || b_0)$
Ensure: r4 . . . r7 = $a \cdot b$
1: mov a_0 , &MPY32L
2: mov a_1 , &MPY32H
3: mov b_0 , &OP2L
4: mov b_1 , &OP2H ▷ $a \cdot b$ in multiplier
5: mov &RES0, r4
6: mov &RES1, r5
7: mov &RES2, r6
8: mov &RES3, r7 ▷ move the multiplier to gpr

been demonstrated, replacing traditional methods like Toom–Cook and Karatsuba multiplication. However, implementing NTT in a 16-bit environment requires careful optimization due to the resource-constrained nature of such systems. Therefore, we first compare previously studied NTT methodologies, focusing on SMAUG-T and MSP430, to determine the most efficient approach. From this point forward, we denote the new modulus prime as q' .

3.1. Approach for NTT-unfriendly algorithms

To apply NTT in an NTT-unfriendly algorithm, new parameters must be selected to expand to an NTT-friendly ring. When choosing a new prime, there are two main approaches [18]. The first approach selects a single large prime q' and performs a single NTT over $Z_{q'}/(X^n \pm 1)$. We refer to this approach as the large-prime method. Another approach involves selecting a large prime q' and then choosing multiple primes $q'_1, q'_2, \dots, q'_i$ that satisfy $q'_1 \cdot q'_2 \cdots q'_i > q'$. NTT is then performed in each ring, and the results are recovered to the original ring using the Chinese Remainder Theorem (CRT). This is referred to as the multi-moduli method. The following illustrates polynomial multiplication $c = a \cdot b$ using the large-prime and multi-moduli methods:

Large-prime method:

$$c = iNTT_{q'}(NTT_{q'}(a) \cdot NTT_{q'}(b))$$

Multi-moduli method:

$$c = CRT \left(iNTT_{q'_1}(NTT_{q'_1}(a) \cdot NTT_{q'_1}(b)), \right. \\ \left. iNTT_{q'_2}(NTT_{q'_2}(a) \cdot NTT_{q'_2}(b)) \right)$$

In [6], the two approaches introduced above were efficiently implemented for Saber, an NTT-unfriendly algorithm, considering the characteristics of each device. On Cortex-M4, the large-prime method was applied using a single NTT-friendly prime. This method was chosen considering the identical cost of 16-bit and 32-bit multiplication on Cortex-M4. In contrast, the AVX2 platform employed a multi-moduli approach using two primes, as the AVX2 instruction set lacks a high-only half-product instruction for 32-bit multiplication. These examples demonstrate how efficient NTT methodologies vary depending on the instruction set and characteristics of the target platform.

3.2. NTT in 16-bit MSP430

As discussed in Section 2.3, the cost of 32-bit multiplication on the MSP430 is twice that of 16-bit multiplication, depending on the number of mov instructions required. The large-prime NTT is based on 32-bit multiplications, while the multi-moduli NTT is based on 16-bit multiplications. However, since the multi-moduli method requires two NTT executions, both approaches incur the same number of multiplication cycles. Despite this, the multi-moduli method incurs additional cost due to the CRT process needed to combine the rings. Therefore, we determine that the large-prime method is the more suitable NTT for the MSP430.

4. Prime selection for efficient NTT

To extend the polynomial ring for NTT, selecting a new NTT-friendly prime is essential. Previous studies [6,13] simply selected the smallest value from the possible range of primes. In our approach, we carefully chose the prime to optimize both NTT/iNTT and modular reduction specifically for the MSP430 platform. This selection facilitates more efficient Montgomery multiplication, significantly improving the performance of NTT/iNTT and pointwise-multiplication.

4.1. NTT-friendly prime for SMAUG-T

The simplest condition for selecting a new modulus q' for NTT is $q' > n \cdot q^2$, which represents a naive upper bound considering the multiplication range in $Z_{q'}$. However, in SMAUG-T, all polynomial multiplications involve one operand being a secret vector, whose coefficients are bounded by the parameter η . Additionally, considering the order l of the public matrix $\mathbf{A}$ in the modulus range, the accumulated values in the NTT domain reduce the number of iNTT operations required. Thus, considering the signed representation, we determine the new modulus range as $q' > 2 \cdot n \cdot \frac{q}{2} \cdot \frac{\eta}{2} \cdot l$.

4.2. Montgomery-friendly prime

We select q' with its lower 16 bits set to 0x0001 to optimize Montgomery multiplication. According to Claim 1, when the lower 16 bits of q' are 0x0001, the lower 16 bits of its modular inverse q'^{-1} over 2^{32} are also 0x0001. This property can reduce intermediate calculations in Montgomery multiplication. Details of this optimization are discussed in the next section.

Claim 1. Given 16-bit $q' < 32$ -bit and the lower 16 bits of q' are 0x0001, the lower 16 bits of the modular inverse q'^{-1} over 2^{32} are also 0x0001.

Proof. Let q' and q'^{-1} be expressed as $q' = a_1 \cdot 2^{16} + a_0$ and $q'^{-1} = b_1 \cdot 2^{16} + b_0$, respectively. The product $q' \cdot q'^{-1}$ can be expanded as:

$$q' \cdot q'^{-1} = a_1 \cdot b_1 \cdot 2^{32} + (a_0 \cdot b_1 + a_1 \cdot b_0) \cdot 2^{16} + a_0 \cdot b_0.$$

To satisfy $q' \cdot q'^{-1} \equiv 1 \pmod{2^{32}}$, it must hold that:

$$a_0 \cdot b_0 \equiv 1 \pmod{2^{16}}.$$

Since a_0 is given as 0x0001, it follows that b_0 must also be 0x0001 to satisfy the condition. $\square$

In this paper, we select $q' = 2424833 (=0x250001)$, aligning the parameters specified in Section 4.1 with the highest security level of SMAUG-T and use it across all security levels.

Table 2

Comparison of costs in 32-bit NTT for SMAUG-T (Mont: Montgomery multiplication, A,M: 32/64-bit Addition/Multiplication).

	NTT	Pointwise-multiplication	iNTT
7-layer (Incomplete NTT)	7·(128 Mont + 256A)	128·(4Mont + 1M64 + 1A64 + 1A32)	7·(128Mont + 256A)
8-layer (Complete NTT)	8·(128 Mont + 256A)	256 · 1Mont	8·(128 Mont + 256A)

Algorithm 3 Signed Montgomery Reduction [20]

Require: $0 < q < \frac{\beta}{2}$ odd, $-\frac{\beta}{2}q \leq a = a_1\beta + a_0 < \frac{\beta}{2}q$ where $0 \leq a_0 < \beta$
Ensure: $r' \equiv \beta^{-1}a \pmod{q}$, $-q < r' < q$
1: $m \leftarrow a_0q^{-1} \pmod{\pm\beta}$ $\triangleright$ signed low product, q^{-1} precomputed
2: $t_1 \leftarrow \left\lfloor \frac{mq}{\beta} \right\rfloor$ $\triangleright$ signed high product
3: $r' \leftarrow a_1 - t_1$

4.3. Montgomery multiplication with q'

Montgomery reduction is an algorithm that performs modular operations efficiently without division instructions. By applying Montgomery reduction to the product $a \cdot b$, Montgomery multiplication [19] can be performed.

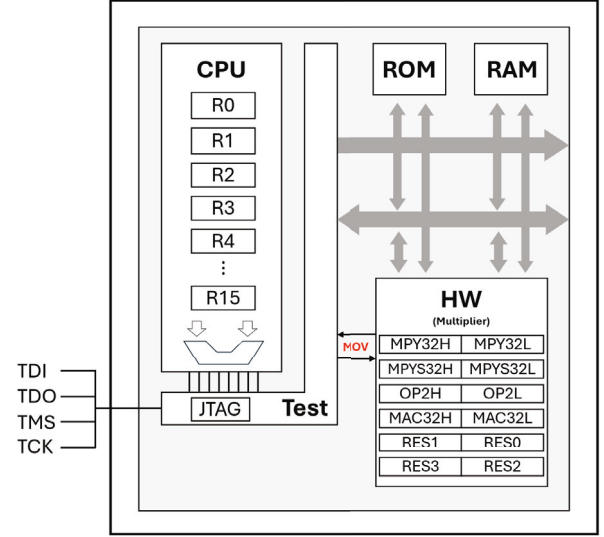
Algorithm 3 illustrates the signed Montgomery reduction proposed in [20]. Here, we first need to calculate the low product, which is the product of the lower 32 bits of a and q^{-1} . Typically, this is performed using 32-bit multiplication. However, by selecting a prime q' with lower 16 bits set to 0x0001, we can handle this with a single 16-bit multiplication and an add instruction instead of 32-bit multiplication. This reduces the number of mov instructions, providing a notable advantage of the chosen prime.

When calculating the high product, as demonstrated in [20, Section 3], signed Montgomery reduction computes only the signed high product, differing from conventional Montgomery reduction. The result is efficiently obtained through subtraction between high parts. The high product can be calculated by left-shifting after 32-bit multiplication. However, since the MSP430 lacks a barrel shifter for simultaneous multi-bit shifts, performing a 32-bit shift requires additional registers and 32 shift instructions. Instead, we directly utilize the RES2 and RES3 registers, where the multiplication result is stored, to avoid shift instructions.

Algorithm 4 32-bit Signed Montgomery Multiplication on MSP430 Using q'

Require: 32-bit $a = (a_1 || a_0)$, $b = (b_1 || b_0)$
Ensure: $c = abR^{-1} \pmod{q}$, where $R = 2^{32}$
1: mov a_0 , &MPYS32L
2: mov a_1 , &MPYS32H
3: mov b_0 , &OP2L
4: mov b_1 , &OP2H $\triangleright$ multiply a, b
5: mov &RES0, &MPYS
6: mov &RES1, r9
7: mov &RES2, c_0
8: mov &RES3, c_1 $\triangleright$ store multiplication result
9: mov #0xFFDB, &OP2
10: add &RESL0, r9 $\triangleright$ signed low product
11: mov r9, &MPYS32H
12: mov #0x0001, &OP2L
13: mov #0x0025, &OP2H $\triangleright$ signed high product
14: sub &RES2, c_0
15: subc &RES3, c_1
16: return $c = (c_1 || c_0)$

Algorithm 4 demonstrates the implementation of Montgomery multiplication on the MSP430, leveraging the property that the lower 16 bits of the chosen q' are set to 0x0001. This property reduces the low

**Fig. 2.** Diagram of MSP430 architecture.

product computation to 16-bit multiplication and addition, avoiding the need for 32-bit multiplication. Fig. 2 presents the architecture diagram of the MSP430. On this platform, multiplication is performed by moving the operands into the hardware multiplier registers MPY and OP2 using the mov instruction. Reducing one 32-bit multiplication is effectively equivalent to removing four mov instructions to the hardware multiplier. This optimization reduces the clock cycles per Montgomery multiplication from 62 to 49, resulting in a 13-cycle reduction compared to [11]. Given that numerous multiplications occur in NTT, iNTT, and pointwise-multiplication, this reduction significantly enhances the overall computational efficiency.

5. NTT optimization for MSP430

This section discusses techniques for optimizing NTT implementations. We customize existing the state-of-the-art optimization strategies for the 16-bit MSP430 platform as follows.

5.1. Efficient NTT layer

In 32-bit environments, [6,13] found a 6-layer incomplete NTT to be optimal for SABER, which has parameters similar to SMAUG-T. However, our experiments reveal different results for the MSP430.

Complete NTT and incomplete NTT differ in computational requirements during the transform and pointwise-multiplication processes. As more NTT layers are applied, the cycle count for NTT increases, while the cycle count for pointwise-multiplication decreases. Table 2 shows the computational cost of each function according to the number of layers. For a 7-layer NTT, pointwise-multiplication involves performing School-book multiplication for degree-1 polynomials. By applying lazy reduction [6] to these degree-1 polynomials, one Mont operation can be replaced with M64.

If values are accumulated in the NTT domain during matrix-vector multiplication, NTT is performed $l^2 + l$ times, pointwise-multiplication

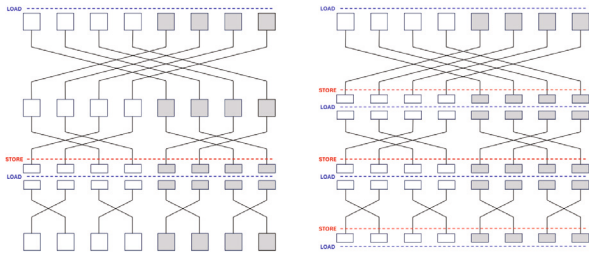


Fig. 3. Merging technique in NTT/iNTT.

l^2 times, and iNTT l times. Considering the computational cost in Table 2 and the matrix order l , 8-layer complete NTT is always more efficient than 7-layer incomplete NTT when $l > 2$. 6-layer NTT requires higher-degree multiplications during pointwise-multiplication. Due to register limitations, this induces additional cycles and is therefore excluded from the comparison. Additionally, in SMAUG-T encryption, the sparse randomness vector r in ComputeC1, matrix-vector multiplication $A \cdot r$, can be retained in the NTT domain. This removes the need for an additional NTT in ComputeC2, inner product (b, r) , further reducing the cycle count.

5.2. NTT layer merging

For hand-assembled NTT implementations, the merging technique is employed to minimize memory accesses during NTT [21]. Identifying the optimal merged layers is crucial, as it depends on the number of registers and the size of coefficients for each embedded device. Since q' is 22 bits, storing a single coefficient on the MSP430 requires two registers. With 4 registers dedicated to NTT, 8 registers remain available for merging, making the merging of two layers the optimal choice. Based on this, we applied 8–7, 6–5, 4–3, and 2–1 layer merging, which is similarly applied to the iNTT. Fig. 3 illustrates a high-level overview of the merging strategy. The left side of Fig. 3 shows the merging-based approach, while the right side depicts the general approach. By merging two layers, we are able to significantly reduce the number of intermediate STORE and LOAD instructions required during NTT and iNTT.

5.3. Using CT-butterfly in iNTT

Typically, the CT-butterfly algorithm is used in NTT, while the GS-butterfly algorithm is applied in iNTT. However, as shown in [15], selecting the CT-butterfly algorithm for iNTT can provide additional computational advantages. When the twiddle factor is 1, the butterfly operation can be performed without multiplication. In SMAUG-T, this approach removes a total of $2^8 - 1$ multiplication operations during iNTT, reducing the cycle count.

6. Result

6.1. Benchmarking setup

We measured performance using the clock cycle register provided by IAR EW for MSP430. All code was tested with the high(speed) compiler optimization option on MSP430F6779. Prior to the full benchmarking, we evaluated the performance of the 32-bit signed Montgomery multiplication implementation used in Dilithium [11] on the MSP430. We then compare the reference Toom–Cook implementation in C with a C implementation where only the multiplications were replaced with NTT. Additionally, we performed comparisons with our proposed techniques implemented in assembly language. Our proposed NTT was implemented entirely in hand-assembled code, while higher-level functions were written in C.

Table 3

Clock cycles for Montgomery multiplication, NTT, iNTT in MSP430 environment (Mont_mul : Montgomery multiplication).

	Mont_mul	NTT	iNTT
[11] (asm)	62 (–)	90736 (–)	114026 (–)
This work (asm)	49 (+27%)	71027 (+28%)	74402 (+53%)

6.2. Performance analysis

Table 3 presents a performance comparison with [11]. While [11] employs the prime 0x7FE001, our implementation selects the Montgomery-friendly prime 0x250001, achieving a 13-cycle reduction in Montgomery multiplication. This improvement directly impacts both NTT and iNTT performance. Furthermore, while [11] uses the GS-butterfly for iNTT, we apply the CT-butterfly, which results in a 53% improvement. Since SMAUG-T, unlike Dilithium, applies the NTT to the public matrix, improving NTT efficiency becomes a key factor in optimizing the overall performance of the algorithm.

Table 4 summarizes the performance of the entire process. The “1-module Mul” represents the cycle count for multiplying two polynomials, while the NTT performance combines the cycles for two NTTs, one pointwise multiplication, and one iNTT. Replacing the Toom–Cook multiplication with NTT-based multiplication in the C implementation significantly improves performance while also reducing stack usage. This is because NTT-based multiplication has lower computational complexity and requires fewer multiplication operations compared to the Toom–Cook multiplication. Furthermore, our assembly implementation, incorporating the proposed optimization techniques, achieves a 143% performance improvement in polynomial multiplication compared to the reference. This gain is mainly attributed to the efficient use of registers in our assembly implementation, particularly for layer merging and Montgomery multiplication. However, this optimization comes at the cost of slightly increased code size compared to the original C implementation. As polynomial multiplication constitutes a major performance bottleneck in LWE-based schemes, this improvement significantly contributes to the overall performance enhancement. Applied to SMAUG-T, our approach improves key generation, encapsulation, and decapsulation by 22%–39%, 44%–57%, and 38%–52% in C, and by 43%–63%, 92%–99%, and 85%–95% in assembly, respectively, compared to the reference. Our work achieves faster performance than the Kyber [22] in the encapsulation and decapsulation processes through the proposed optimizations.

6.3. Security consideration

Side-channel resistance constitutes a critical security consideration for cryptographic implementations in embedded systems, where non-constant-time implementation exposes vulnerabilities to timing attacks capable of leaking secret information. [12] revisited the original SMAUG-T proposal and introduced a timing-secure version that mitigates known timing-based side-channel attacks. In this work, we use the timing-secure reference implementation from [12] and optimize only the polynomial multiplication using hand-written assembly. As our assembly code does not contain any control-flow instructions dependent on secret data, such as conditional branches or data-dependent loops, our implementation maintains constant-time execution. For all other cryptographic operations beyond polynomial multiplication, we use pure reference code. Therefore, our full implementation maintains the side-channel security characteristics of the timing-secure SMAUG-T reference.

Table 4

Performance comparison of SMAUG-T in MSP430 environment (cc: clock cycles, K: 1,000 cc, B: bytes).

Software	1-module Mul cc		SMAUG-T128		SMAUG-T192		SMAUG-T256	
			cc	stack	cc	stack	cc	stack
Reference [12]	569K (–)	K	5 375K (–)	9 296B	11 055K (–)	12 496B	18 141K (–)	16 848B
		E	5 177K (–)	11 028B	10 817K (–)	14 900B	17 762K (–)	19 796B
		D	6 982K (–)	11 640B	13 477K (–)	15 768B	21 301K (–)	20 984B
This work (C)	474K (+20%)	K	4 416K (+22%)	7 396B	8 433K (+31%)	11 108B	13 069K (+39%)	15 972B
		E	3 607K (+44%)	8 682B	7 197K (+50%)	13 516B	11 302K (+57%)	18 924B
		D	5 074K (+38%)	9 744B	9 263K (+45%)	14 384B	13 995K (+52%)	20 112B
This work (asm)	234K (+143%)	K	3 768K (+43%)	7 396B	7 209K (+53%)	11 108B	11 101K (+63%)	15 972B
		E	2 701K (+92%)	8 682B	5 630K (+92%)	13 516B s	8 906K (+99%)	18 924B
		D	3 781K (+85%)	9 744B	7 161K (+88%)	14 384B	10 914K (+95%)	20 112B
Software	–		Kyber512		Kyber768		Kyber1024	
	–		cc	stack	cc	stack	cc	stack
Kyber [22]	–	K	3 325K	6 228B	5 635K	10 258B	8 866K	15 378B
		E	4 468K	8 890B	7 424K	13 430B	11 088K	19 070B
		D	4 362K	9 666B	7 287K	14 526B	10 880K	20 646B

Table 5

Cycle counts and estimated energy consumption of Kyber768 and SMAUG-T192 KEM implementations on low-resource embedded platforms. (K: 1,000cc, energy in mJ).

Work	Platform Board		Kyber768	
			cc	energy
[25] (asm)	32-bit Cortex-M3	K	842K	1.40
		E	1 017K	1.69
		D	958K	1.59
[23] (asm)	8-bit AVR	K	8 608K	5.16
		E	11 169K	6.70
		D	10 250K	6.15
Work	Platform Board		SMAUG-T192	
			cc	energy
This Work (asm)	16-bit MSP430	K	7 209K	2.09
		E	5 630K	1.63
		D	7 161K	2.07

6.4. Clock cycle and energy comparison on low-resource platforms

Table 5 summarizes the clock cycles and estimated energy usage for Key Generation, Encapsulation, and Decapsulation procedures across the three platforms. While cycle counts are influenced by implementation-specific optimization and instruction set architectures, we include them here for reference.

To enable consistent energy comparison, we normalized all energy estimates to a common frequency of 25 MHz. The MSP430F6779, which natively supports up to 25 MHz operation and utilizes a 32-bit hardware multiplier, served as the baseline for extrapolation. We roughly measured power consumption based on publicly available specifications, following the energy consumption estimation approach described in [23]. According to TI's datasheet [24], the power consumption of the MSP430 at 25 MHz is 7.25 mW. Similarly, power figures for the AVR and Cortex-M3 platforms were derived from their respective datasheets or prior literature [23,25], assuming linear scaling with frequency.

Notably, our SMAUG-T192 implementation achieves consistently lower energy usage than AVR-based Kyber768, and remains competitive with the more capable 32-bit Cortex-M3 platform. These results further support the suitability of SMAUG-T192 for deployment on ultra-low-power microcontrollers.

CRediT authorship contribution statement

MinGi Kim: Writing – original draft. **DongHyun Shin:** Writing – review & editing. **WooHyung Ko:** Writing – review & editing. **Young-Beom Kim:** Writing – review & editing. **Seog Chung Seo:** Writing – review & editing.

Declaration of competing interest

The authors declare no conflict of interest.

Acknowledgments

This work was partly supported by the Institute of Information and Communications Technology Planning and 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) and the National Research Foundation of Korea (NRF) grant funded by the MSIT (No. 2022R1C1C1013368).

References

- [1] P.W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, *SIAM Rev.* 41 (2) (1999) 303–332.
- [2] J. Bos, et al., CRYSTALS-kyber: a CCA-secure module-lattice-based KEM, in: 2018 IEEE European Symposium on Security and Privacy, EuroS&P, IEEE, 2018, pp. 353–367.
- [3] L. Ducas, et al., Crystals-dilithium: A lattice-based digital signature scheme, *IACR Trans. Cryptogr. Hardw. Embed. Syst.* (2018) 238–268.
- [4] T. Prest, et al., Falcon, Post-Quantum Cryptogr. Proj. NIST (2020).
- [5] D.O. Greconici, et al., Compact dilithium implementations on cortex-M3 and cortex-M4, *IACR Trans. Cryptogr. Hardw. Embed. Syst.* (2021) 1–24.
- [6] C.-M.M. Chung, et al., 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) 159–188.
- [7] V. Hwang, et al., Multiplying polynomials without powerful multiplication instructions, *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2025 (1) (2025) 160–202.
- [8] D. Atkins, Requirements for post-quantum cryptography on embedded devices in the IoT, in: Third PQC Standardization Conference, 2021.
- [9] M. Pursche, et al., SoK: The engineer's guide to post-quantum cryptography for embedded devices, *Cryptol. EPrint Arch.* (2024).
- [10] Y. Kim, et al., Efficient implementation of ARX-based block ciphers on 8-bit AVR microcontrollers, *Mathematics* 8 (2020).
- [11] D. Shin, et al., Optimizing crystals-dilithium implementation in 16-bit MSP430 environment utilizing hardware multiplier, *ICT Express* (2024).
- [12] J.H. Cheon, et al., SMAUG (-T), revisited: Timing-secure, more compact, less failure, *IEEE Access* (2024).
- [13] J. Zhang, et al., Time-memory trade-offs for saber+ on memory-constrained RISC-V platform, *IEEE Trans. Comput.* 71 (11) (2022) 2996–3007.
- [14] Texas Instruments, MSP430x5xx and MSP430x6xx Family User's Guide, Dallas, TX, USA, 2021, Available at: <https://www.ti.com/lit/ug/slau208>.
- [15] A. Abdulrahman, et al., Multi-moduli NTTs for saber on cortex-M3 and cortex-M4, *IACR Trans. Cryptogr. Hardw. Embed. Syst.* (2022) 127–151.
- [16] T. Fritzmann, et al., RISQ-V: Tightly coupled RISC-V accelerators for post-quantum cryptography, *IACR Trans. Cryptogr. Hardw. Embed. Syst.* (2020) 239–280.
- [17] T. Fritzmann, et al., Masked accelerators and instruction set extensions for post-quantum cryptography, *IACR Trans. Cryptogr. Hardw. Embed. Syst.* (2022) 414–460.

- [18] Z. Liang, et al., Number theoretic transform and its applications in lattice-based cryptosystems: A survey, 2022, arXiv Preprint.
- [19] P.L. Montgomery, Modular multiplication without trial division, *Math. Comp.* 44 (170) (1985) 519–521.
- [20] G. Seiler, Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography, *Cryptol. EPrint Arch.* (2018).
- [21] E. Alkim, et al., Cortex-M4 optimizations for $\{R, M\}$ LWE schemes, *IACR Trans. Cryptogr. Hardw. Embed. Syst.* (2020) 336–357.
- [22] M.J. Kannwischer, et al., Improving software quality in cryptography standardization projects, in: 2022 IEEE European Symposium on Security and Privacy Workshops, EuroS&PW, IEEE, 2022, pp. 19–30.
- [23] Y. Kim, S.C. Seo, An optimized instantiation of post-quantum MQTT protocol on 8-bit AVR sensor nodes, *Cryptol. EPrint Arch.* (2025).
- [24] T. Instruments, MSP430f6779 datasheet, 2018, <https://www.ti.com/product/MSP430F6779>, SLAS768E, Rev. E.
- [25] J. Huang, H. Zhao, J. Zhang, W. Dai, L. Zhou, R.C. Cheung, C.K. Koc, D. Chen, Yet another improvement of plantard arithmetic for faster kyber on low-end 32-bit IoT devices, *IEEE Trans. Inf. Forensics Secur.* (2024).