Received November 17, 2021, accepted December 8, 2021, date of publication December 27, 2021, date of current version January 7, 2022.

Digital Object Identifier 10.1109/ACCESS.2021.3138807

A Survey of Polynomial Multiplication With RSA-ECC Coprocessors and Implementations of NIST PQC Round3 KEM Algorithms in Exynos2100

JONG-YEON PARK[1], YONG-HYUK MOON[2,3], (Member, IEEE), WONIL LEE[1], SUNG-HYUN KIM[1], AND KOUICHI SAKURAI[4], (Member, IEEE)

[1]Samsung Electronics System LSI, Hwaseong-si, Gyeonggi-do 16677, South Korea
[2]Department of Computer Software, University of Science and Technology, Daejeon 34113, Republic of Korea
[3]Electronics and Telecommunications Research Institute, Daejeon 34129, Republic of Korea
[4]Department of Informatics, Faculty of Information Science and Electrical Engineering, Kyushu University, Fukuoka 819-0395, Japan

Corresponding author: Yong-Hyuk Moon (yhmoon@etri.re.kr)

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT, Ministry of Science and ICT) (No. 2021-0-00907, Development of Adaptive and Lightweight Edge-Collaborative Analysis Technology for Enabling Proactively Immediate Response and Rapid Learning).

ABSTRACT Polynomial multiplication is one of the heaviest operations for a lattice-based public key algorithm in Post-Quantum Cryptography (PQC). Many studies have been done to accelerate polynomial multiplication with newly developed hardware accelerators or special CPU instructions. However, another method utilizes previously implemented and commercial hardware accelerators for RSA/elliptic curve cryptography (ECC). Reusing an existing hardware accelerator is advantageous, not only for the cost benefit but also for the improvement in performance. In this case, the developer should adopt the most efficient implementation method for the functions provided by a given legacy hardware accelerator. It is difficult to find an optimized implementation for a given hardware accelerator because there are a variety of methods, and each method depends on the functions provided by the given accelerator. In order to solve the problem, we survey methods for polynomial multiplication using RSA/ECC coprocessors and their application for Learning With Error (LWE)-based KEM algorithms of National Institute of Standards and Technology (NIST) PQC round 3 candidates. We implement all known methods for polynomial multiplication with RSA/ECC coprocessors in a platform, commercial mobile system-on-chip (SoC), the Exynos2100 Smart Secure Platform (SSP). We present and analyze the simulation results for various legacy hardware accelerators and give guidance for optimized implementation.

INDEX TERMS Polynomial multiplication, RSA/ECC coprocessor, Kronecker substitution, PQC, post-quantum cryptography, Lattice-based cryptosystem.

I. INTRODUCTION

The emergence of quantum computers affects widely commercialized encryption algorithms such as RSA and Elliptic curve cryptography (ECC) algorithms, which have been used for decades. Shor's research [1] revealed that RSA and ECC are no longer secure in quantum computing environments. To prepare for these changes, the National Institute of Standards and Technology (NIST) started Post-Quantum Cryptography (PQC) standardization in 2016, and the candidates for the third round were selected in 2020 [3], [4].

One of the most-studied areas in which PQC candidate algorithms are created is lattice-based cryptography. Among these algorithms, Learning With Error (LWE)-based algorithms are often studied due to their efficiency [2]. Polynomial multiplication is one of the most important operations that constitute an LWE-based algorithm. Therefore, many researchers are studying high-speed methods that use

The associate editor coordinating the review of this manuscript and approving it for publication was Gustavo Olague.

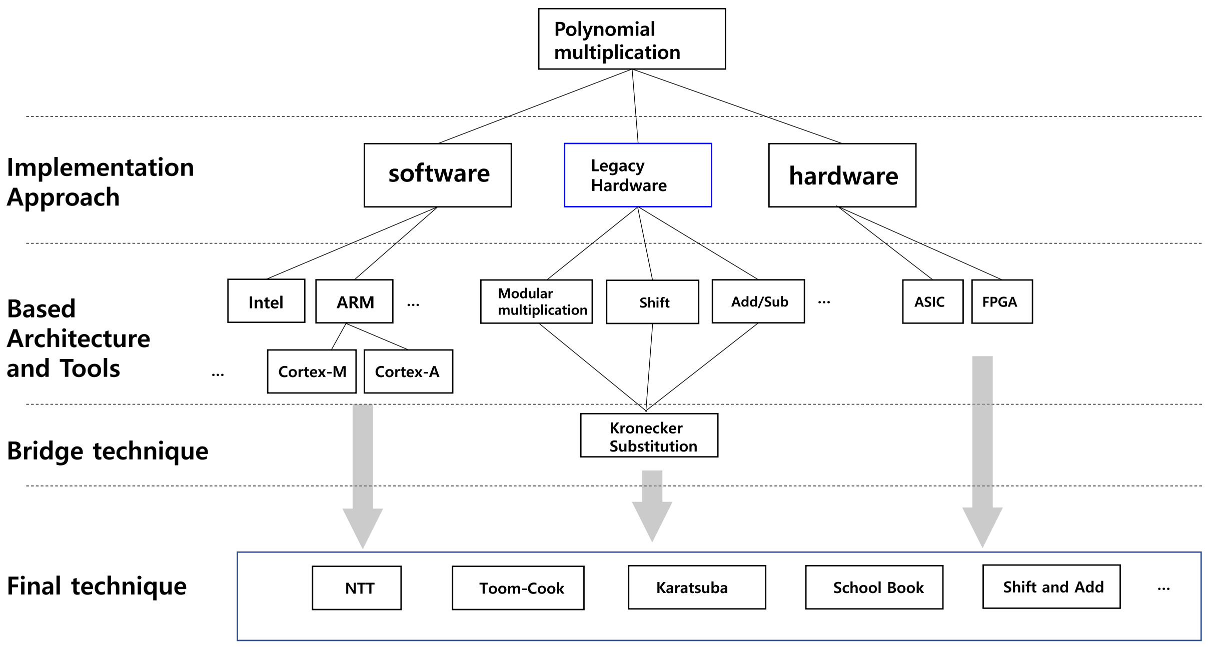


FIGURE 1. Technical overview of researches related to polynomial multiplication.

TABLE 1. Comparison of three approaches for PQC implementation.

Approach	Advantages	Disadvantages
Software only	Flexible	Slow
PQC-dedicated H/W	Fast	Expansive
Legacy Hardware	Reusable	Limited application

hardware or software to speed up an LWE-based algorithm with various implementation techniques; see the software and hardware implementation approach in Figure 1. The hardware implementation platform that should be considered for improving performance is ASIC or the field-programmable gate array (FPGA). For software implementation, a processor based on ARM or Intel with instructions related to the selected architecture are considered. After that, the best combination is determined by selecting from among the final techniques in Figure 1, which are to be discussed in this paper.

However, considering the long transition time to the new PQC algorithms, utilizing the existing legacy hardware accelerator for current public key cryptography algorithms is a reasonable approach due to its flexibility and ease of deployment: see the legacy hardware implementation approach in Figure 1. There are many devices that have a large number of modular multiplication hardware accelerators for the RSA/ECC algorithm because RSA and ECC are the most popular algorithms for public key cryptography. We can compare three approaches to implementing PQC algorithms; see Table 1. Software implementation is a common and easy way to realize algorithms. However, it is usually not proper for embedded devices because it is very slow compared with hardware-based implementation. We can also achieve a high speed by making a specific PQC algorithm-dedicated hardware accelerator. In this case, however, we have to spend a considerable amount of time and money. In Table 1, legacy hardware refers to the big number modular multiplication hardware accelerators that are intended for the RSA/ECC algorithm, and this approach is the main topic of this study. Note that legacy hardware is used in many industries, including the smart card business. Thus, the results of studies for reusing legacy hardware have been published by research engineers at companies and organizations in the smart card business.

In addition to the advantages of this approach, there are many things that we need to consider. The first is that it is slower than using PQC dedicated hardware, but it is generally faster than implementing the PQC algorithm with Software alone. Therefore, it is considered quite appropriate to use this approach if the speed with RSA/ECC coprocessor is sufficiently high. The second is that we need to know it is difficult to use legacy hardware to implement all kinds of PQC candidate algorithms. However, we can investigate which algorithm are better to use. The third is that there are many kinds of commercial legacy hardware, and a developer should use a specific legacy hardware. The reason for mentioning this is that the developer should adopt the most efficient implementation method using the functions provided by the chosen hardware. The performance differs according to the legacy hardware in the actual implementation. The fourth is that there are several well-known techniques for polynomial multiplication using RSA/ECC coprocessors. In this paper, we discuss and research these four points, and we describe our own implementation result and comparison using one platform, the commercial mobile System-On-Chip (SoC) Exynos2100 SSP [35]. In addition, we present and analyze simulation results for various legacy hardware accelerators, and we offer guidance for optimized implementation.

As we described, there are various legacy hardware accelerators for RSA/ECC implementation, and software developers should adopt the most efficient implementation method within the functions provided by the given hardware. Possible functions provided by a given legacy hardware are integer operations, for example, multiplication, addition, subtraction, shift, and reduction of big number, see category of legacy hardware in Figure 1.

However, the functions that can be provided on a device vary depending on the size of chips and scope of use, so all commercial legacy hardware is different. The most time-consuming operation for RSA/ECC operations is modular integer multiplication with modular reduction, and it is generally the most widely mounted function. To maximize the computational speed of RSA/ECC, a manufacturer implements modular multiplication with a hardware pipeline that is as fast as possible. If the area allows for further improvement in performance, addition, subtraction, shift operations are provided optionally. The maximum size of the calculation value provided also varies depending on the legacy hardware. Therefore, the implementation method varies depending on the specification of the provided size, even if the same kind of functions can be used.

In order to reuse commercial legacy hardware, a key problem that must be solved is that it must be possible to perform polynomial modular multiplication using integer modular multiplication. To solve the problem, it should be possible to perform polynomial modular multiplication by moving the space of the polynomial operation to the integer operation. A mathematical approach called Kronecker substitution (KS) [26], [59], which is a method for polynomial multiplication using integer computational multiplication is suggested. This is the bridge technique in Figure 1. Variations of KS have been studied by Harvey [5], Yaman *et al.* [6], Fateman [10], and Fateman [10].

We intend to compare actual implementations according to various techniques on a single platform and to present the results. In addition, by simulating various legacy hardware and comparing the results, we would like to help developers choose the optimal method.

- Can a PQC candidate algorithm be implemented using an RSA/ECC hardware accelerator?
- Which operations constituting the PQC candidate algorithm can be implemented through the RSA/ECC hardware accelerator?
- Which of the current PQC candidate algorithms can be implemented using the RSA/ECC hardware accelerator?
- What would be the best way for developers who are already using a commercial RSA/ECC hardware accelerator to implement a PQC candidate algorithm?

To answer the above questions, this paper consists of the following: in Section II, the relation between polynomial multiplication and integer multiplication is described, and KS, a method of substituting polynomial multiplication with integer multiplication, is described. In Section III, we explain RSA/ECC Coprocessors for integer operations. In Section IV, we describe Saber and Cryptographic Suite for Algebraic Lattices (CRYSTALS) Kyber, which this study focuses on, and we discuss the characteristics of the polynomial multiplication used in these two PQC candidate algorithms. In Section V, we describe additional consideration for the KS implementation. In Section VI, we describe well-known techniques that are modifications of KS. Section VII shows the performance results of the well-known techniques implemented using the RSA/ECC hardware accelerator in Exynos 2100. In Section VIII, the simulation results according to various types of legacy hardware are shown. In Section IX, the conclusions are described and future tasks are summarized.

II. RELATION BETWEEN POLYNOMIAL MULTIPLICATION AND INTEGER MULTIPLICATION

A. MATHEMATICAL RELATIONS

Polynomial multiplication is basically different from integer multiplication. The main difference is the carry-by-addition of coefficients. The KS method [26] was introduced as a technique for reducing problems concerning multivariate polynomials to the case of univariate polynomials by evaluating the polynomial. By this technique, the multiplication in $\mathbb{Z}[x]$ can be changed to multiplication in $\mathbb{Z}$ by evaluation. Schönhage also introduced this property in his research [59]. KS is a common way to perform modular multiplication in integer arithmetic for polynomial multiplication. The main property is described as below,

$$\phi : \mathbb{Z}[y] \rightarrow \mathbb{Z}[y]/(B - y) \tag{1}$$

A well-chosen $B \in \mathbb{Z}$ in (1) makes $\mathbb{Z}[y]/(B - y)$ isomorphic to $\mathbb{Z}$ and $f(y) \in \mathbb{Z}[y]$ is transformed to $f(B)$. This means a polynomial is transformed to an integer. As a result, the relation between polynomial multiplication and integer multiplication is an evaluation with a well-chosen parameter B. This transformation is called "base-B clumping," and is also known as an evaluation by B. In the opposite direction, it is called "lifting(ϕ^{-1})," as suggested by Bernstein [22]. We use these terms in later sections. The term "lifting" is not just defined as the opposite direction of "clumping"; it actually means the transformation to a larger space that includes the original space. This concept is based on evaluation homomorphism, which is a basic concept of abstract algebra [34].

B. BASIC METHOD OF KS

Harvey introduced KS and its variant in [5]. For example, given $f(x) = 12x + 34$ and $g(x) = 56x + 78$, we want to know the polynomial product $f(x) \times g(x) = h(x)$. We know the answer is $h(x) = 672x^2 + 2840x + 2652$. To use integer multiplication instead of polynomial operation, we set

$B = 10^4$. Thus, the result of clumping is $\phi(f(x)) = f(10^4) = 120034$ and $\phi(g(x)) = g(10^4) = 560078$. By equation (1), $\phi(f(x) \times g(x)) = \phi(f(x)) \times \phi(g(x)) = f(10^4) \times g(10^4) = 120034 \times 560078 = 67228402652 = h(10^4)$, "672 || 2840 || 2652." Finally, by lifting, which is a reversal of ϕ, the number in $\mathbb{Z}$ is transformed to $h(x) = 672x^2 + 2840x + 2652$. This concept illustrates how to use an integer operator for polynomial operation. An evaluation by "power of 10" is not an arithmetic operation—indeed, there is no operation for this, except extracting and storing the values on a proper buffer by radix 10^4. For this operation in computers, we will use the radix "power of 2." The value of B is important for the lifting. If B is not adequately large, this transformation is not invertible. For example, $B = 10^2, f(10^2) \times g(10^2) = h(10^2) = 7006652$ is not directly converted to 67228402652. So, we need to choose a double size of maximum coefficient for B, which guarantees ϕ is invertible.

C. A KS VARIANT: KS2

An integer that is too large causes difficulties in the real-world implementation of PQC. For example, in Saber, the polynomial degree of a given ring is 256 and the coefficients of the multiplicands are 13 bits and 3 bits. Thus, the output of each operand by clumping is at least 6656 bits(256 × 13). The scale of an RSA/ECC coprocessor does not normally use 6656 bits, because the RSA key size generally ranges to 4096 bits, and a 2048-bit RSA is recommended until the year 2030 [27]–[29]. Indeed, many products have coprocessors that support multiplication under about 4096 bits [23]–[25]. Thus, we need to reduce the size of the operation for PQC polynomial multiplication. A simple example is given in Harvey [5], called KS variant, e.g., "KS2." There are several division techniques, such as Number Theoretic Transform (NTT) or Nussbaumer [30], [31] as shown below:

$$h(x) = xh_0(x^2) + h_1(x^2) \tag{2}$$

As we remember from the previous example, we need $h(10^4)$. In addition, $h_0(10^4)$ and $h_1(10^4)$ can be directly converted to $h(10^4)$. In other words, h_1 and h_2 are different representations of h. By the equation (2), we obtain $h(10^2) = 10h_0(10^4) + h_1(10^4)$ and $h(-10^2) = -10^2h_0(10^4) + h_1(10^4)$. From the two previous equations, we derive

$$h_1(10^4) = \frac{h(10^2) + h(-10^2)}{2} \tag{3}$$

and

$$h_0(10^4) = \frac{h(10^2) - h(-10^2)}{20} \tag{4}$$

By (3) and (4), $h(10^4)$ can be computed, and the final lifting step is the same as the general technique that has already been described in section II-B. The size is $B = 10^2$ instead of $B = 10^4$. Therefore, the size of the integer for multiplication after clumping is reduced to half. Finally, we can compute the polynomial multiplication for Saber, which has 6656 bits, using a 4096 bits multiplicator, because the size of each polynomial is $6656/2 = 3328$ bits.

III. RSA/ECC COPROCESSORS: HARDWARE FOR INTEGER OPERATIONS

For devices that have limited processing power, such as embedded secure elements, cryptographic operations need a hardware accelerator to meet application and service transaction performance requirements, for example, the transaction flow of VISA card processing should be under 300 ms [17]. In order to meet these requirements more securely, symmetric key algorithms are usually fully hardware based; we can verify that most devices for financial purposes have symmetric algorithms with a semiconductor intellectual property core (SIP core), denoted IP; see the common criteria security targets of each device [18]. RSA or Elliptic Curve Cryptography (ECC) are used for the digital signature in many applications, but the algorithms are too large to be fully implemented on hardware. However, software implementations have performance limitations even with a lightweight ECC [58]. Therefore, only the most time consuming and critical operations use the hardware accelerator, such as big number arithmetic in the integer domain.

Because of the trade-off between chip size and performance, the accelerator usually supports only core functions, which are modular multiplication, addition, subtraction, and barrel shifter, and so on. Generally speaking, addition and subtraction require a small number of operations. Shift needs a few more operations. Multiplication is a high-cost operation. One category of division operations, such as modular reduction, is a very high-cost integer operation. In a hardware accelerator, modular reduction and other operations can be combined. Modular addition is the addition and then reduction of two integers. Modular addition and subtraction are not high-cost operations because only one subtraction or addition is required in the worst case of reduction. In contrast, modular multiplication is a relatively more complex operation; modular multiplication is not a simple combination of multiplication and division. The operational cost of reduction is much greater than that of multiplication.

Many hardware security chips that support RSA/ECC, i.e., NXP [23], Infineon [24], Espressif [25] and Samsung [18], already have the accelerator for the modular multiplication. Those devices have optional adders and shifters. In addition, the pros and cons of the implementation are different depending on the type of hardware the device has. We organize the basic hardware that is helpful for the existing public key operation into the following categories and analyze the performance of the application with the latest technique using KS which is described in section VI.

- "Modular multiplication" is utilized in devices as default.
- "Add/Subtraction" is optional.
- "Shift" is optional.

Therefore, there are four combinations of hardware application. However, an ECC-dedicated architecture also exists for Transport Layger Security(TLS), called the TLS coprocessor [40], [41]. The relevant instruction is modular addition, subtraction, and multiplication by prime number p up

TABLE 2. (Algorithm 1): Kyber.cpa.enc (Pseudo Code, Structure overview).

```
Input : $pk_{CPA} = (t, \rho), m \in \mathbf{M}, r \leftarrow \{0,1\}^{256}$
Output : Cipher $c$.
1. $t \leftarrow Decompress_q(t, d_t)$
2. $A \leftarrow R_q^{k\times k}$
3. $(r, e_1, e_2) \leftarrow \chi_n^k \times \chi_n^k \times \chi_n$
4. $u \leftarrow Compress_q(A^T r + e_1, d_u)$
5. $v \leftarrow Compress_q(t, r + e_2 +, \lceil \frac{q}{2} \rfloor)$
6. $c = (u, v)$
```

TABLE 3. (Algorithm 2): Kyber.cpa.enc (Pseudo Code, submitted algorithm in Round 3).

```
Input : $pk_{CPA} = (t, \rho), m \in \mathbf{M}, r \leftarrow \{0,1\}^{256}$
Output : Cipher $c$.
1. $t \leftarrow Decompress_q(t, d_t)$
2. $A \leftarrow R_q^{k\times k}$ ($A$ is already in NTT domain)
3. $(r, e_1, e_2) \leftarrow \chi_n^k \times \chi_n^k \times \chi_n$
4. $r \leftarrow NTT(r)$
5. $u \leftarrow NTT^{-1}(A^T r) + e_1$
6. $v \leftarrow NTT^{-1}(t^T r) + e_2 + Decompress_q(m, 1)$
6. $c = (u, v)$
```

TABLE 4. (Algorithm 3): Saber.cpa.enc (Pseudo Code).

```
Input : $pk = (b, seed_A), m \in \mathbf{M}$
Output : Cipher $c$.
1. $A \leftarrow R_q^{l\times l}$
2. $s' \leftarrow \chi_n^l$
3. $b' \leftarrow ((As' + h) \bmod q) >> (\epsilon_q - \epsilon_p) \in R_p^{l\times 1}$
4. $v' \leftarrow b^T(s' \bmod p) \in R_p$
6. $c_m \leftarrow (v' + h_1 - 2^{\epsilon_p - 1} m \bmod p) >> (\epsilon_p - \epsilon_T) \in R_T$
5.c=$(c_m, b')$
```

to 256 bits, SHA2-256 [47], and AES-128,256 [46]. We can find some implementations in Banerjee *et al.* [39] for CRYSTALS Kyber [14], FrodoKEM [43], ThreeBears [44], SPHINCS+ [45] and SIKE [42] with ECC coprocessor. In particular, the ECC-based algorithm SIKE is well-adapted to the ECC coprocessor. However, the limitations of the legacy-only ECC coprocessor is that the size of multiplication is small, so the big arithmetic of ring LWE-based candidates is not suitable. Thus, we do not deal with ECC dedicated coprocessors in this study.

IV. LWE-BASED PQC ALGORITHMS

A. CRYSTALS KYBER

Kyber is part of CRYSTALS along with the signature scheme Dilithium [14], [16]. Kyber is an IND-CCA2-secure Key Encapsulation Mechanism (KEM) whose mathematics basis is the LWE problem in the module lattices (Module-LWE) problem by Langlois *et al.* [12]). The mathematical carrier is the polynomial ring $Z_q[X]/(X^{256}+1)$ with $q = 3229$ and a 256 coefficient polynomial. Kyber512 uses $k = 2$, Kyber768 uses k = 3, and Kyber1024 uses $k = 4$, which are the module sizes for each security level, as shown in Algorithm 1. The most expensive operations are multiplication over a vector or a matrix of polynomials, for example matrix $A\times$ vector s. This algorithm uses NTT for the polynomial multiplication [15]. There are several back and forth transformations between domains, and the calculation complexity and the size trade-offs have to be considered carefully in the algorithm, for example, matrix A is directly sampled in the NTT domain, and public key $t = (A \times s + e)$ is in the NTT domain.

Algorithm 1 in Table 2, is pseudocode of the Kyber encryption algorithm. We refer to the algorithm that is based on Albrecht *et al.* [7] without the NTT domain. χ_n is a Centered Binomial Distribution (CBD) with an n-degree polynomial shape. The output bit of CBD depends on parameters η_1, η_2. The value of η that is the maxium absolute value of χ is derived from the standard deviation where 0 is the mean. According to the parameters of Round 3 Kyber, η_1 is 3 or 2, and η_2 is always 2. Thus, the output of the CBD is at most 3 bits. The specification of Kyber submitted to Round 3 of NIST does not exactly follow Algorithm 1 due to the adoption of NTT, as we explained above.

Algorithm 2 in Table 3 is pseudocode of Kyber with an encryption algorithm based on NTT. This is the actual algorithm that was submitted to NIST Round 3. The main advantage is that the polynomial multiplication can operate in $O(n(logn))$. Moreover, to avoid conversion to the NTT domain, matrix A is set in NTT domain coefficients after random uniform sampling; see step 2. The operation is well-defined because any coefficient has a one-to-one correspondence with the normal domain coefficient. In other words, the NTT domain has the same sampling space as the normal domain. However, the algorithm must fix the twiddle factors for the NTT (or inverse NTT) operation, which costs additional memory space. Moreover, the algorithm for A matrix generation reduces flexibility and may cause a lack of compatibility in future implementations.

B. SABER

The Saber is based on the Module Learning With Rounding (MLWR) problem, which is a variant of the LWE problem [13]. The algorithm utilizes a module structure, as introduced by Langlois *et al.* [12]. The polynomial ring is $Z_q[X]/(X^{256}+1)$ with $q = 2^{13}$. We do not need modular arithmetic, as modular reduction can be done by simple shifting because it uses a 2^n modulus, and the rounding operation is also easily done by chopping. LightSaber, Saber, and FireSaber consist of 2×2, 3×3, and 4×4 modules (poly-matrix) with a 256 coefficient polynomial for each module.

Algorithm 3 in Table 4, is a pseudo code of Saber encryption algorithm. We refer to the algorithm from the official Saber documentation [13]. l can be 2, 3 or 4, which is the module size of each security level. The main difference between Saber and CRYSTALS Kyber is shown below.

- There is no error addition presented in e_1, e_2 of steps 4 and 5 in Algorithm 1 and steps 5 and 6 in

Algorithm 2. Instead of modification of the value by error, Saber uses shift (rounding) of the variables in step 5 and 7 in Algorithm 3. The advantage of this property does not need to be mentioned in this paper.

- This algorithm is not an NTT-based scheme. Saber uses pure polynomial multiplication, which helps the algorithm utilize the RSA/ECC coprocessor directly, and this leads us to various efficient methods that can be used for polynomial operations, in contrast to Kyber and Dilithium, which propose algorithms with NTT for the polynomial multiplication.

V. ADDITIONAL CONSIDERATIONS FOR KS IMPLEMENTATIONS

A. GENERAL PRE-PROCESSING

To apply polynomial arithmetic with the RSA/ECC coprocessor, additional preprocessing is inevitable. In the above example $f(x) = 12x + 34$ is stored in two memory spaces, e.g., $F_0 = 12$ and $F_1 = 34$. However, after clumping, the integer is 120034. Even this simple transformation, like producing 120034 from F_0 and F_1, requires additional operations that cannot be overlooked e.g., the save, store, and bit shift operations. KS2 needs more precomputing than KS1, and each method has its own preprocessing. In real-world implementation, these additional "non-computing steps" should be counted.

B. PREPROCESSING FOR INTEGER REDUCTION

Montgomery or Barrett methods are used in general for efficient modular reduction [19], [20]. Each method requires precomputing for multiplication, e.g., $R = b^m \bmod n$ in the Montgomery method to convert to the Montgomery domain, $\frac{b^m}{n}$ where b is the radix of operation [21], which is normally a power of 2. As the modular value for PQC is fixed and m is a value where $b^m > n$, we do not need to consider general precomputing methods for unknown modular values. In addition, some RSA/ECC coprocessors offer precomputing by hardware. Montgomery multiplication is conducted as below,

$$Mont(A, B) \leftarrow ABR^{-1} \tag{5}$$

Thus, if we need the value $A \times B$, there are two choices: computing AR first and then performing (5), or (5) and then $Mont(AR^{-1}, R^2)$. AR is also computed using $Mont(A, R^2)$. Furthermore, one additional multiplication is required as a default when Montgomery multiplication is used.

C. GENERAL POST-PROCESSING

After integer multiplication, the value must be recovered in the polynomial form. In general KS, final lifting is required. For example, 67228402652 is not the desired form. Each coefficient of the polynomial should be extracted, such as $h(x) = 672x^2 + 2840x + 2652$. One of the differences between polynomial multiplication and integer multiplication is the way negative values are dealt with. Coefficients in polynomials have their own independent signs. However, in the integer world, there is no independent coefficient, so we have to recover the real signed values for each coefficient. For example, $f(x) = 5x^3 - 2x^2 + 3x + 1$ and $f(10) = 5 \times 1000 - 2 \times 100 + 3 \times 10 + 1$. The output is represented as 4831, but the desired result is $5\hat{2}31$ (the hat denotes a negative number). Therefore, we need a reference number to decide whether a number is negative or positive. For example, if $(x > 5)$ then $x \leftarrow (10 - x)$. The 8 is changed to 2, the borrow from 4 will be added, and finally 4 becomes 5. Thus, one can know that 4831 is actually $5\hat{2}31$. This post processing impacts the overall performance because one must check the size of all coefficients, change values, and add carry bits.

VI. KNOWN METHODS FOR POLYNOMIAL MULTIPLICATION WITH RSA/ECC COPROCESSORS

In this section, we explain three techniques that use different strategies for polynomial multiplication based on KS. To the best of our knowledge, these three techniques cover for most of the related studies until now. The techniques are verified by real implementation and simulation.

A. DIVISION AND MULTIPLICATION (DM)

This section describes the research presented by Albrecht *et al.* [7]. The main contribution of this paper is using KS combined with low-degree polynomials, using Karatsuba-based polynomial multiplication and KS1 and KS2 on SLE78 [24]. The asymmetric coprocessor on SLE78 has an operator of approximately 2048 bits. The main target algorithm is Kyber-768, which originally included polynomial multiplication by NTT [31], [48] in submitted algorithm in NIST round 3. Kyber does not use pure polynomial multiplication, which has already been explained in SectionIV-A. The size of the polynomial for KS1 is at least 5376 bits for multiplication in $\mathbb{F}_q[x]/(x^{256} + 1)$. Thus, we need the polynomial to use a small operator hardware accelerator. In order to divide the polynomial, Schönhage's method is used. This technique is quite useful, as many theories are based on it e.g., KS2, NTT, Nussbaumer, and many other methods stated in D.J. Bernstein [22].

1) DIVISION POLYNOMIAL BY SCHÖNHAGE'S TECHNIQUE

Schönhage's technique is a method for separating polynomials using the transformation shown below,

$$\Psi : \mathbb{Z}[X]/(X^n + 1) \rightarrow (\mathbb{Z}[Y]/(Y^{n/t} + 1)[X]/(Y - X^t) \tag{6}$$

for example, $f(x) = 12 + 34x + 56x^2 + 78x^3 \in \mathbb{Z}[X]/(X^4 + 1)$ is transformed with $t = 2, y = x^2, f(x, y) = (12 + 56y) + (34 + 78y)x$ in $(\mathbb{Z}[X]/(Y^2 + 1)[x]/(Y - X^2)$. Thus, the multiplication of the polynomial is divided into a two-parts coefficient polynomial about y in $\mathbb{Z}[Y]/(Y^2 + 1)$. The number of multiplications for 2-polynomials is $2^2 = 4$; in general, the number of multiplications is t^2.

2) REDUCTION IN INTEGER DOMAIN

The original KS method is based on general polynomial multiplication. However, polynomial operation in ring LWE includes modular reduction by a given polynomial. The candidate algorithms in NIST Round 3 use $x^{256} + 1$ as the polynomial for reduction. Indeed, one could easily compute the polynomial reduction after getting output of polynomial degree 512, which is $h'(x) \leftarrow f(x) \times g(x)$in $\mathbb{Z}[x]$, and then, $h(x) \leftarrow h'(x) \bmod (x^{256} + 1)$. The cost of this operation can be considered not to be high because the polynomial $x^{256} + 1$ is so efficient, as only 256 small coefficient subtractions are required. However, subtraction by Software can have very high-cost compared with that of a hardware coprocessor. Albrecht *et al.* [7] use modular multiplication using an RSA coprocessor by evaluating 2^l of KS $f(2^l) \times g(2^l) \bmod (2^{l\times 256} + 1)$.

3) COMBINATION WITH KARATSUBA TECHNIQUE

After polynomial separation, we can use the Karatsuba multiplication quoted in D.J. Bernstein [22]. It is a well-known technique with a complexity according to the number of multiplications $3^{log_2 t}$. $(Ax + B) \times (Cx + D) = ACx^2 + (AD + BC)x + BD$, concluding with four multiplications and three additions. In order to reduce the number of multiplications, we compute $A+B$ and $C+D$ first, and $P = (A+B)\times(C+D) = AC + BC + AD + BD$. Finally, $(AD + BC)$ is computed by P with one subtraction, so only three multiplications, which are AC, BD and P, are required and two additional additions are needed. This process can be done recursively with $t > 2$, and then the number of multiplications is converged to $3^{log_2 t}$.

4) COMBINATION WITH TOOM COOK TECHNIQUE

Toom-Cook is one of the most widely used algorithm for polynomial or integer multiplication [11]. Depending on the depth of the operation, the Toom–Cook method can be generalized from Toom–Cook 2 to Toom–Cook (n). Regardless of the size of n, polynomial multiplication is calculated using the following three steps. $f(x) \times g(x) = h(x)$,

1) Evaluation—The polynomials $f(x)$, $g(x)$ are converted into a constant by evaluating $g(1), g(-1), f(-1), g(-1)$, and so on with other integers, which is easily computed and is denoted by $f(b_i)$, $g(b_i)$, where i is an index.
2) Multiplication—By multiplying $f(b_i) \times g(b_i) = h(b_i)$.
3) Interpolation—$h(x)$ is finally recovered with $h(b_i)$.

In order to recover $h(x)$ with $h(b_i)$ by interpolation efficiently, one can use matrix arithmetic. Because interpolation is the reverse of evaluation, the matrix for interpolation is computed using the inverse of the evaluation matrix. The input of the interpolation matrix is $h(b_i)$ and the output of the matrix is all coefficients of h(x). Toom–Cook 2 divides $f(x)$ and $g(x)$ into two large polynomials, and also uses the Ψ function of Schönhage's technique; see Equation(6). $F(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{n-1}x^{n-1}$, can be separated into $F(x) = (a_1 + a_3x^2 + a_5x^4 \ldots + a_{n-2}x^{n-2})x + (a_0 + a_2x^2 + a_4x^4 + \ldots + a_{n-2}x^{n-2})$, and then changed by $F(x, y) = (a_1 + a_3y + a_5y^2 \ldots + a_{n-2}y^{(n-2)/2})x + (a_0 + a_2y + \ldots + a_{n-2}y^{(n-2)/2})$ where $x^2 = y$. In this way, given $F(x)$ and $G(x)$, we can get $F(x, y) = xf_1(y) + f_0(y)$ and $G(x, y) = xg_1(y)+g_0(y)$ where f_0 and g_0 are separated polynomials with even indexed coefficients, and f_1 and g_1 have an odd index. If two polynomials $F(x, y)$ and $G(x, y)$ for x with degree 1 are multiplied, the product $H(x, y)$ becomes a cubic polynomial for x. Finally, we obtain $H(x, y) = f_0g_0 + (f_1g_0 + f_0g_1)x + (f_1g_1)x^2$. In the same way, where $x^4 = y$, Toom–Cook 4 is applied. For example, in order to obtain multiplication of the two 255-degree polynomials, 255-degree polynomial is divided into four 63-degree polynomial, and the operation of 3-degree polynomials with coefficient which is a 63-degrees polynomial is performed. The product of two cubic polynomials is a sixth-order polynomial, so there are a total of seven coefficients. There must be seven output coefficients. Therefore, Toom–Cook 4 requires 7 evaluations and multiplications for each polynomial. Toom–Cook 2 requires three polynomial operations with degree $m/2$, where the original polynomial degree is m. Toom–Cook 3 requires 5 polynomial operations with degree $m/3$, Toom–Cook 4 requires 7 polynomial operations with degree $m/4$ and Toom–Cook(n) requires $2n - 1$ polynomial operations with degree m/n.

5) REAL-WORLD IMPLEMENTATION OF DIVISION AND MULTIPLICATION (DM)

From the beginning to the end, until we obtain the output of polynomial multiplication, the operation steps are as shown below,

(A) Polynomial Separation of k polynomials
(B) Evaluation by 2^l for KS
(C) Polynomial multiplication by hardware
(D) Post-Processing

Step (A) is regarded not to be an even operation, and Albrecht *et al.* [7] calls step (B) SNORT, and step(D) SNEEZE. In a real-world implementation, the process in detail is as below,

(A) Polynomial separation of k polynomials

- Changing the order of polynomial coefficients using equation (6). There is no arithmetic operation in this step.

(B) Evaluation by 2^l for KS

- This step can be ignored in the operation if each coefficient is stored in a 32 bit array. This is regarded as the output the evaluation $l = 2^{32}$. By the same logic, l is chosen according to the power of 2 that is the basic size of the computation, such as 2^8 or 2^{16}, which are an unsigned char and an unsigned short.

(C) Polynomial multiplication by hardware

- If Karatsuba is used, addition operations are required before the evaluation. The first number moves to the Montgomery domain that is described in Section V-B, $f(2^{32})R \leftarrow f(2^{32})$.

- Multiplication $h(2^{32}) \leftarrow Mont(f(2^{32})R, g(2^{32}))$: This step must be performed by the hardware.

- The number of multiplications depend on DS, Karatsuba, or Toom–Cook.

(D) Post-Processing

- Addition and subtraction of multiplication outputs. The number of addition and subtraction operations depends on the method, e.g., Karatsuba or normal DM.

- Composite array of addition outputs for single $h(x)$. This step is the reverse of (A).

- Final reduction by prime field (in Kyber and Dilithium only).

B. Kronecker+

Kronecker+ is introduced by Bos *et al.* [8]. This method focuses on minimizing the number of separations with a generalization of KS2.

1) GENERALIZATION OF KS2

According to VI-A, the main purpose of these techniques is to reduce the size of the polynomial. However, a smaller polynomial causes a greater number of multiplications. The original variant of KS, which is KS2 (see II-B), is a smart way to reduce the size of the multiplication by evaluating a smaller number, but KS2 does not reduce the size of the polynomial recursively. In other words, KS2 has no more separation compared with DM which can separate polynomial more, but Kronecker+ suggests a method or reducing the multiplicand to use multiplication by generalization, which is of the techniques of KS2 [8]. On the other hand, KS2 is based on 2-way division, as shown below,

$$f(x) = f_0(x^2) + xf_1(x^2) \tag{7}$$

$$f(10) = f_0(10^2) + 10 \times f_1(10^2) \tag{8}$$

If one more split is required, it is possible with $\pm\sqrt{10}$ and $\pm\sqrt{10i}$ of f_0 and f_1. However, when $t = 4$, equation (7) expands equation (9) by Schönhage's technique. The background will be explained in the next section.

$$f(x) = f_0(x^4) + xf_1(x^4) + x^2f_2(x^4) + x^3f(x^4) \tag{9}$$

Then the evaluation of the polynomial by '10' in the previous example for KS2 is,

$$f(10) = f_0(10^4) + 10f_1(10^4) + 10^2f_2(10^4) + 10^3f(10^4) \tag{10}$$

To use KS2, there are four different polynomials for pointwise multiplication. Two polynomials, $f(10)$ and $f(-10)$, obviously exist as below,

$$f(10) = f_0(10^4) + 10f_1(10^4) + 10^2f_2(10^4) + 10^3f(10^4) \tag{11}$$

$$\begin{aligned} f(-10) &= f_0((-10)^4) + -10f_1((-10)^4) \\ &\quad +((-10)^2)f_2((-10)^4) + ((-10)^3)f((-10)^4) \\ &= f_0(10^4) - 10f_1(10^4) + 10^2f_2(10^4) - 10^3f(-10^4) \end{aligned} \tag{12}$$

We would like to know $f(10^4)$, so we need two more equations. Thus, we need four values for $x = 10^4$, which are $10, -10, 10i, -10i$. However, we do not have i in $\mathbb{Z}$. In order to compute the evaluation of i, Bos *et al.* [8] used Nussbaumer's technique, which is quoted in Bernstein [22]; this technique is called Kronecker+. By this technique, we can split a polynomial into smaller segments than with KS2. KS2 is a 2-way example of Kronecker+, so Kronecker+ is generalization of KS2.

2) NUSSBAUMER

For the description of Nussbaumer's technique, we recall equation (6).

$$\Psi : \mathbb{Z}[X]/(X^n + 1) \rightarrow (\mathbb{Z}[Y]/(Y^{n/t} + 1)[X]/(Y - X^t) \tag{13}$$

In CRYSTALS Kyber, Dilithium, and Saber, $\mathbb{Z}[X]/(X^{256} + 1)$ is used for reduction. It can be rewritten as $\mathbb{Z}[Y]/(Y^{256/t} + 1)[X]/(Y - X^t)$. This means that the degree of the equation for Y is less than $256/t$, and the degree of the equation for X is less than t. The degree of the multiplication of two polynomials in $\mathbb{Z}[Y]/(Y^{256/t} + 1)[X]/(Y - X^t)$ is less than $2t$ about x. Equation (13) can be naturally lifted to

$$\mathbb{Z}[Y]/(Y^{256/t} + 1)[X]/(X^{2t} - 1) \tag{14}$$

If we need to conduct one multiplication of two polynomials in $\mathbb{Z}[Y]/(Y^{n/t} + 1)[X]$, the space can be lifted again $(Y^{256/t} + 1)[X]$ without the reduction part. By the definition of general homomorphism [34], we can do an evaluation by x_i for $\Psi(f(x)) \times \Psi(g(x)) = \Psi(h(x))$, such as $\Psi(f(x_i)) \times \Psi(g(x_i)) = \Psi(h(x_i))$ where $x_i \in \mathbb{Z}[Y]/(Y^{n/t} + 1)[X]$, by the $2t$ evaluation of $\Psi(g(x_i))$ and $\Psi(f(x_i))$ and pointwise multiplications. One can make $2t$ equations, such as $\{\Psi(h(x_0)), \Psi(h(x_1)), \Psi(h(x_2)), \ldots, \Psi(h(x_{2t-1}))\}$. Now, by linear transformation of the system of equations, one can compute $\Psi(h(x))$. On the other hand, to simplify the system of equations, the value of the evaluation is chosen by the primitive $2t$-th root of unity with the property $x^n = -1$ and $y^{n/t} = -1$. We know $(y^{n/t^2})^{2t} = 1$, so y^{n/t^2} is the primitive $2t$-th root of unity, denoted $y^{n/t^2} = \zeta_{2t}$, ${\zeta_{2t}}^{2t} = 1$ and ${\zeta_{2t}}^{t} = -1$, where ζ_n is primitive n-th root of unity.

The evaluation of $\Psi(f)$ and $\Psi(g)$ with the root of unity ζ^i can be carried out using the Cooley-Tukey butterfly algorithm [31]. The total number of operations for one polynomial multiplication by Nussbaumer is $2t$ for polynomials of degree (n/t), initial evaluation and final evaluation that is the same algorithm to initial evaluation. By the evaluation of the primitive root (which is also polynomial with y) can split a given ring.

For example, $n = 8, t = 2, y = x^2, \Psi(f(x)) = y^3 - 3y - 3 + (-3y^3 - 2y^2 - y)x$, $\Psi(f(\zeta_4)) = \Psi(f(y^2)) = y^3 - 3y - 3 + (-3y^3 - 2y^2 - y)y^2 = y^3 - 3y - 3 + (-3y^5 - 2y^4 - y^3)$. $\Psi(f(\zeta_4)) = y^3 - 3y - 3 + (3y + 2 - y^3) = -1$ By $y^4 = -1$. It is a very lucky case. The output is in $\mathbb{Z}[Y]/(Y^{n/t} + 1)[X]$, so this reduces the original polynomial to half degree.

3) COMBINATION OF NUSSBAUMER AND GENERALIZATION OF KS2

Equation (14) expands the space of the actual area; that is $\mathbb{Z}[X]/(X^n+1) \cong \mathbb{Z}[Y]/(Y^{n/t}+1)[X]/(X^t-1)$. If there are t elements $\mathbb{Z}[Y]/(Y^{n/t}+1)$ that satisfy$X^t+1=0$, t equations evaluated by the elements can be utilized in t point-wise multiplications, like NTT. However, no element exists in $\mathbb{Z}[Y]/(Y^{n/t}+1)$ for $X^t-1=0$. To solve this limitation, Kronecker+ does not evaluate the polynomial using the coefficients in $\mathbb{Z}[Y]/(Y^{n/t}+1)$. In other words, an evaluation by the primitive wt root of unity, $({\zeta_{2t}}^0, {\zeta_{2t}}^1, {\zeta_{2t}}^2, \ldots, {\zeta_{2t}}^{2t-1})$, is utilized for Nussbaumer, but the primitive t root of unity, $({\zeta_{2t}}^0, {\zeta_t}^1, {\zeta_t}^2, \ldots, {\zeta_t}^{t-1})$, is finally used to evaluate the polynomial in Kronecker+. Those elements all satisfy $X^{t/2}=-1$ due to the multiplication by x.

We can describe KS2 in a different way. $f(x)=f_0(x^2)+xf_1(x^2)$ is re-written by Schönhage $f(x)=f_0(y)+xf_1(y) \in (Y^{n/2}+1)[X]/(X^2-Y)$, and $\zeta_t=y^{n/2}$ is primitive 2-nd root of unity. Thus, the elements for evaluation are $(x{\zeta_2}^0, x{\zeta_2}^1)$. Therefore $f(x{\zeta_2}^0)=f(x)$ and $f(x{\zeta_2}^1)=f(-x)$ are required for KS2. Now, we can use KS alone, which is exactly KS2 (See Section II-B).

For one more split from KS2, denote For 4-way separation, the domain of operation is $(Y^{n/4}+1)[X]/(X^4-Y)$. A shape of element is above equation (9), and the primitive 4-th root of unity is $\zeta_4=y^{n/16}$. We evaluate $f(x)$ using $(x, x{\zeta_4}^1, x{\zeta_4}^2, x{\zeta_4}^3)$. We obtain four outputs, $f(x)$,$f(x{\zeta_4}^1)$,$f(-x)$ and $f(x{\zeta_4}^3)$. Now ${\zeta_4}^2=-1$, so ζ_4 is regarded as imaginary number i, and consider the example of evaluation by 10. Equation (8) has four results of evaluation, e.g., $f(10)$,$f(10\zeta_4)$,$f(-10)$ and $f(-10\zeta_4)$. $f(10\zeta_4)=f_0(10^4)+10\zeta_4f_1(10^4)-10^2f_2(10^4)-10^3{\zeta_4}^3f_3(10^4)$ by $x=y^4$. Note that the evaluation in $(Y^{n/4}+1)[X]/(X^4-Y)$ by mapping $X \leftarrow 10\zeta_4$ is different from the evaluation in $\mathbb{Z}[X]/(X^n+1)$.

The final mapping is $X \leftarrow 10y^{n/16}$, $Y \leftarrow 10^4$. $f(10\zeta_4)=f(10y^{n/16})=f_0(10^4)+10\times 10^{4n/16}f_1(10^4)-10^2\times f_2(10^4)-10^3\times(10^{4n/16})^3f_3(10^4)$. Note that $(10^{4n/16})^2=-1$, so $(10^{4n/16})^2+1=0$. Thus, if an RSA/ECC coprocessor with modular reduction is used for the algorithm, one can use reduction by $(10^{4n/16})^2+1$. This means the evaluation of the polynomial can be done with $\mathbb{Z}[X]/(X^n+1)$.

TABLE 5. **(Algorithm 4): Pseudo-algorithm of Kronecker+.**

```
Input : $f, g \in \mathbb{Z}/(X^n+1)$ for a positive integer $n$, the Kronecker parameter $l$ and a positive integer $t$ such that $t|l$ and $t|n$, and $M_i = 2^{2iln/t^2 \cdot 2^{l/t}}$ for $0 \le i < t$.
Output : $h = \sum_{i=0}^{n-1} h_i X^i = f \cdot g \bmod X^n + 1$.
1. Compute $f(M_i)$ and $g(M_i)$ for $i = 0, ..., t-1$.
2. Compute $h(M_i) = f(M_i) \cdot g(M_i) \bmod 2^{ln/t} + 1$ for $i = 0, ...t-1$.
3. Compute $h^{(i)} = \frac{\sum_{j=0}^{t-1} 2^{2i(t-j)ln/t} h(M_j)}{t \cdot 2^{il/t}} \bmod 2^{ln/t} + 1$ for $i = 0, ..., t-1$.
4. Recover $h_{i+tj}$ as the $j$-th $l$-bit limb of $h^{(i)}$ for $0 \le i < t$ and $0 \le j < n/t$
```

4) REAL-WORLD IMPLEMENTATION OF Kronecker+

Table 5 describes how to use Kronecker+ from the beginning to the end, until we obtain the output of polynomial multiplication. The operation steps are below.

(A) Evaluation (SNORT in [7]). This is step 1 in Algorithm 4.

(B) Polynomial multiplication by hardware. This is step 2 in Algorithm 4.

(C) Post-processing. This is steps 3 and 4 in Algorithm 4.

In a real-world implementation, the process in detail would be as shown below,

(A) Evaluation by 2^l for KS

- This is also considered evaluation by 2^l, but unlike DM [7], it needs t different type of evaluations. It is not neglect able time-consumption. Additional consideration of Kronecker+ is the size after polynomial evaluation. This is because the Bos *et al.* [8] use evaluation by $M_i=2^{2iln/t^2}\cdot 2^{l/t}$ where $f(x)=\sum_{i=0}^{n-1} f_iX^i$.

This is also considered evaluation by 2^l, but unlike DM [7], it needs t different types of evaluations. The time consumption is not negligible. An additional consideration of Kronecker+ is the size after polynomial evaluation. This is because Bos *et al.* [8] used evaluation by $M_i=2^{2iln/t^2}\cdot 2^{l/t}$ where $f(x)=\sum_{i=0}^{n-1} f_iX^i$. - Therefore, additional reduction is required. One reduction is replaced by one subtraction.

(B) Polynomial multiplication by hardware

- The first number moves to the Montgomery domain, as described in Section V-B. $f(M_i)R \leftarrow f(M_i)$.

- Pointwise multiplication, $h(M_i) \leftarrow Mont(f(M_i)R, g(M_i))$. This is the same as DM.

(C) Post-Processing

- Recombination for $h(x)$. This is step 3 of Algorithm 4. Every element in the summation $\sum_{j=0}^{t-1} 2^{2i(t-j)ln/t}h(M_j)$ is multiple of $t\cdot 2^{il/t}$, which seems quite complicated compared with KS2. The troublesome point is that it also causes many shift operations. If numerator is calculated at the first and we start to divide by the denominator, then the total number of operations is extremely high. Therefore, to avoid this situation, before each addition of numerator, we shift by $t\cdot 2^{il/t}$ and add with modular reduction by $2^{ln/t}+1$.

- Processing negative values for the final output.

- Final reduction by the prime field (in Kyber and Dilithium only)

C. SMALL COEFFICIENT MULTIPLICATION

Greuet *et al.* [9] proposed two specific optimizations of polynomial multiplication when one of the operands has coefficients close to 0, namely Kronecker Substitution Variant using small coefficients (KSV) and Shift&Add. In particular, Greuet *et al.* [9] proposed optimization for embedded devices that have an RSA/ECC coprocessor that provides efficient large-integer arithmetic.

1) KSV

Suppose there are two polynomials, $f(x)$ and $g(x)$. Let $g(x)=g_2x^2+g_1x+g_0$. Here, we focus on the case where g_i is a very small coefficient. Then,$f(x)g(x) = f(x)g_2x^2 + f(x)g_1x + f(x)g_0$. Let $f(x) = 8x^2 + 3x + 2$ and $g(x) = 5x^2 + 4x + 1$. Then, $f(10^2)g(10^2) = f(10^2)g_2(10^2)^2 + f(10^2)g_1(10^2) + f(10^2)g_0 = f(10^2) \times 5 \times (10^2)^2 + f(10^2) \times 4 \times (10^2) + f(10^2) \times 1 = 80302 \times 5 \times (10^2)^2 + 80302 \times 4 \times (10^2) + 80302 \times 1 = 4015100000 + 32120800 + 80302 = 4047301102$. The resulting polynomial is recovered with radix conversion like in classical KS. The multiplication of two large integers is replaced with three multiplications of a large integer by a small coefficient and some additions and shifts. This technique can be used when we consider the multiplication of a polynomial with small coefficients. Such multiplications are used in some lattice-based KEM, CRYSTALS Kyber and Saber.

2) SHIFT&ADD

There is no need to perform multiplication in Shift&Add. It relies only on additions and left-shifts. This technique is of interest when one of the operands has small coefficients. The basic idea is explained in the following example.

Let $f(x) = 9x^2 + 8x + 3$ and $g(x) = g_2x^2 + g_1x + g_0 = 2x^2 - 1$. Let $r = 0$. The computation of $f(x)g(x)$ is done as follow: Step 1. Evaluate $f : f(10^3) = 9008003$ Step 2. Since $g_2 = 2$:

1. $r \leftarrow r + f(10^3) \times (10^3)^2$
2. $r \leftarrow r + f(10^3) \times (10^3)^2$

Step 3. Since $g_1 = 0$, do nothing;
Step 4. Since $g_0 = -1$, $r \leftarrow r - f(10^3) \times (10^3)^0$;
This leads to

$$\begin{aligned} f(10^3)g(10^3) &= 2f(10^3)(10^3)^2 - f(10^3) \\ &= 2(9008003 \times (10^3)^2) - 9008003 \\ &= 18015996991997 \end{aligned}$$

By radix conversion,

$$\begin{aligned} f(x)g(x) &= 18x^4 + (15+1)x^3 - (10^3 - (996+1))x^2 \\ &\quad -(10^3 - (991+1))x - (10^3 - 997) \\ &= 18x^4 + 16x^3 - 3x^2 - 8x - 3. \end{aligned}$$

Greuet *et al.* [9] also show practical results to compare the above two methods using Kyber and Saber. Between KS variants and Shift&Add, we cannot say one is always more efficient that the others. However, in the case of Kyber512R1, KSV is more efficient than Shift&Add. In the case of Kyber1024R1 and KyberR2, Shift&Add is more efficient than KSV. We give only the simulation results for the hardware barrel shifter and adder. We do not implement polynomial multiplication with this method for the reasons given below.

1) In our environment, the multiplication of small variables and large variables is equal to the multiplication of two large variables. Therefore, KSV offers no advantage.

2) There is no barrel shifter in our environment, so we cannot take advantage of the "shift and add" technique. Actually, the shift operation is very slow for big number arithmetic. Nevertheless, this technique can be applicable to an environment that has a hardware barrel shifter and adder, but no hardware modular multiplication operator.

VII. IMPLEMENTATION OF MULTIPLICATION ON Exynos2100 SSP

Exynos2100 is a SoC for Mobile phones [36]. It contains a security module for cryptographic operations, which is called Strong Box [35] and SSP [56], [57]. It has functions for asymmetric cryptography RSA, ECC, which is a modular multiplication based on Montgomery reduction up to 4096 bits.

In this environment, we use hardware modules such as modular multiplication by Montgomery reduction and addition, and subtraction up to 512 bits only for ECC. The addition and subtraction hardware does not help big operations because hardware operations need extra data relay for hardware control. It doesn't have a barrel shifter. Therefore, we use only modular multiplication with 4096 bits maximum.

Finally, there are three meaningful hardware conditions for our research, the first is the big addition and subtraction operator, the second is the barrel shift operator, and the third is the modular multiplication operator. Our environment is satisfactory with only the third condition, and we also discuss other conditions in the remaining sections. We implement division and multiplication (DM), Karatsuba as quoted in Albrecht *et al.* [7], and Kronecker+ by Bos *et al.* [8] by 2-way separation and 4-way separation. In our environment, a 2-way split works sufficiently, but we give an additional prospect of more splitting with the result of 4-way separation.

More separations such as 8-way and 16-way are not practical in any environment for two reasons. Firstly, as of today, we expect that commercial devices with RSA/ECC coprocessors have a modular multiplication operator of more than 2048 bits. Secondly, if an operator for less than 2048 bits multiplication is used in some environments, there is no remarkable advantage compared to the full implementation of Software multiplication. The compiler is ARM compiler version 6, and optimization -O0 because speed optimization removes the side channel attack countermeasures such as dummy operations, duplication code, etc. Furthermore, we do not choose code optimization by compiler, as countermeasures should not be deleted unintentionally.

The number of clock cycles in the following experiments is based on the external clock, so it is not estimated by clock cycles of operation in Exynos2100 SSP directly. The operation clock for the multiplication is secret hardware IP information. It gives a relative estimation of the speed of operations.

A. ADDITIONAL CONSIDERATIONS OF IMPLEMENTATION

1) PREPROCESSING FOR MONTGOMERY REDUCTION

In our environment, the hardware uses Montgomery multiplication. For Montgomery reduction, the inputs are moved to the Montgomery domain. The constant for the Montgomery domain, R, is a form of 2^m, and m is a constant that depends on

TABLE 6. Number of clock cycles for DM [7] by 2-way separation(Exynos2100).

Sub-Operation	Number of clock cycles	ratio in one-multiplication
Polynomial Separation × 2	1422	8.2%
Pointwise Multiplication × 4	13480	77.9%
Polynomial Addition × 2	1250	7.2%
Polynomial shift	475	2.7%
Polynomial re-combination	688	4.0%
Summation	17315	100%

TABLE 7. Number of clock cycles of each steps in a "4096 bits Pointwise multiplication"(Exynos2100).

Sub-Operation	Number of clock cycles	ratio in one-multiplication
Data copy for inputs ×2	287	8.7%
Move to Montgomery domain	511	15.4%
Main multiplication H/W	1520	45.9%
Processing Negative sign	711	21.5%
Data copy for outputs ×2	274	8.3%
Summation	3303	100%

TABLE 8. Number of clock cycles of each steps for Karatsuba [7] by 2-way separation(Exynos2100).

Sub-Operation	Number of clock cycles	ratio in one-multiplication
Polynomial Separation × 2	1422	9.4%
Pointwise Multiplication × 3	10010	66.5%
Polynomial Addition × 3	1875	12.3%
Polynomial Subtraction	625	4.1%
Polynomial shift	475	3.1%
Polynomial Addition	624	4.1%
Polynomial re-combination	688	4.5%
Summation	15212	100%

TABLE 9. Number of clock cycles of each steps for DM [7] by 4-way separation(Exynos2100 SSP).

Sub-Operation	Number of clock cycles	ratio in one-multiplication
Polynomial Separation × 4	1422	5.4%
Pointwise Multiplication × 16	20400	77.2%
Polynomial Addition × 9	2813	10.6%
Polynomial shift × 2	475	1.8%
Polynomial Addition × 2	625	2.4%
Polynomial re-combination	688	2.6%
Summation	26423	100%

the hardware specification. Thus, for KS, $A \times 2^m \bmod 2^n + 1$, where $A < 2^n + 1$, is carried out by shift and subtraction. If m is a multiple of l, which is the Kronecker parameter for evaluation (see Algorithm 4), the rotation of the index is replaced by shift. For example, the shift of 32bits does not need to be calculated with 'unsigned int' variable. This can be done with index-change of the array.

2) MODULAR REDUCTION OF COEFFICIENTS

For the implementation of the PQC algorithm, the domain of each operation is different for each algorithm. In Saber, modular reduction by 2^13 is removable because the computer operates variables over a power of 2, for example 16bits(short) or 32bits(int). However, Kyber needs a final reduction by q of Kyber parameters. This additional operation affects the overall operation time, but it is implemented only by software, we do not cover its implementation.

B. RESULT OF MULTIPLICATION

In this section, we present the result of DM in Section VI-A and Kronecker+ in Section VI-B. We have explained the reason why we do not implement small coefficient multiplication (see Section VI-C.).

Table 6 shows the result for the number of clock cycles of DM [7] by 2-way separation. The Kronecker parameter l is set to 32. Theoretically, we can choose the parameter $l > 26$ in Saber and Kyber. However, a parameter with 32 bits is small enough to operate a variable that is less than 4096 bits with 2-way separation. If we choose $l = 26$, this also requires 2-way separation without the advantage for operation complexity. There is one more advantage: the evaluation by 2^{32} actually needs no operation, as it's the basic data size (unsigned int), and the polynomial shift of this result does not require a barrel shifter because changing the index is substituted for shifting by 32 bits. Pointwise multiplication in Table 6 means polynomial multiplication by KS. In this case, there are four polynomial multiplications to be executed.

Table 7 shows a detailed analysis of pointwise multiplication in Table 6. This result does not contain the number of clock cycles for the function call. For multiplication, we need an additional data copy operation as well as hardware multiplication in $\mathbb{Z}$ itself. It's over 15% in pointwise multiplication. Our environment does not have a Direct Memory Access block (DMA) [37]. If there is a DMA in the environment, the cost will be reduced. However, DMA involves many security concerns and leakages [38], so it may not be an advantage. The move to the Montgomery domain also occupies over 15% in pointwise multiplication. For optimization of the operations, we apply the technique described in Section VII-A1. Processing negative signs is also a main post-processing after KS, so we explain this in Section V-C. In summary, the pure multiplication consumes only approximately half of the whole computational time.

Table 8 is the number of clock cycles for Karatsuba [7] using 2-way separation. By Karatsuba, there are three multiplications and more additions required, instead of four multiplications with additional memory. If the implementation environment has enough memory for the intermediate result of addition, Karatsuba is efficient solution for reducing the operation.

Tables 9 and 10 present the 4-way separation by the DM and Karatsuba methods. Because of the number of polynomial multiplications t^2 in DM, the operation time is almost twice that of the 2-way separation. Even though the polynomial is half the size, pointwise multiplication is not divided by

TABLE 10. Number of clock cycles of each steps for Karatsuba [7] by 4-way separation(Exynos2100 SSP).

Sub-Operation	Number of clock cycles	ratio in one-multiplication
Polynomial Separation × 4	1422	8.1%
Pointwise Multiplication × 9	11475	65.6%
Polynomial Addition(divided by 4-length) × 6	1875	10.7%
Polynomial Subtraction(divided by 4-length) × 3	938	5.4%
Polynomial shift(comment)	475	2.7%
Polynomial Addition(divided by 2-length)	625	3.6%
Polynomial re-combination	688	3.9%
Summation	17498	100%

TABLE 11. Number of clock cycles of each steps for Kronecker+ [8] by 2-way separation(Exynos2100 SSP).

Sub-Operation	Number of clock cycles	ratio in 1-multiplication
Evaluation 1	650	3.6%
Evaluation 2	650	3.6%
Number Reduction	625	3.5%
Pointwise multiplication × 2	5318	19.7%
Number Shift × 4	3552	19.8%
Number Reduction × 6	3750	20.9%
Number Addition × 2	1250	7.0%
Negative Processing	711	3.9%
Negative Processing	711	3.9%
Polynomial re-combination	688	3.8%
Summation	17905	100%

TABLE 12. Number of clock cycles of each steps for Kronecker+ [8] by 4-way separation(Exynos2100 SSP).

Sub-Operation	Number of clock cycles	ratio in 1-multiplication
Evaluation × 4	1300	5.7%
Number Reduction	1250	5.4%
Pointwise multiplication	3678	16.0%
Number Shift × 16	7104	31.0%
Number Addition × 12	3750	16.3%
Number Reduction × 12	3570	16.3%
Negative Processing × 4	1422	6.2%
Polynomial re-combination	688	3.0%
Summation	22942	100%

a quadratic number. As you can see in Table 7, multiplication by hardware is under 50% in pointwise multiplication, so the number of clock cycles is not reduced with $1/t^2$. The shift of polynomial coefficients in Karatsuba and DM is different from the shift of number in Kronecker+ (see Table 11). Polynomial shift is just array shifting by changing the index of the buffers; in contrast, the shift of number is quite complicated in the Software implementation. Therefore, the polynomial shift is more advantageous than Kronecker+'s shift of number.

Tables 11 and 12 present the clock cycles of the implementation of Kronecker+ by 2-way separation and 4-way separation, respectively. An interesting point is that the performance is indeed not better than that of Karatsuba (see Tables 8 and 10). As we mentioned above, the main reason

TABLE 13. Number of clock cycles of each steps for Toom-Cook 4 with RSA/ECC Coprocessor(Exynos2100 SSP).

Sub-Operation	Number of clock cycles	ratio in 1-multiplication
Interpolation and Evaluation	13166	59.5%
Pointwise Multiplication × 7	8925	40.4%
Summation	22091	100%

for the low performance is the integer bit shifter. The performance enhancement will be discussed in Section VIII.

There are many reductions due to Step 3 in Algorithm 4(Kronecker+). The size of each intermediate result of the summation causes big size addition, but if we reduce each intermediate result in the summation, we can reduce the size of the next addition. Actually, the whole operation is almost the same (or big size addition and just one reduction may be somewhat better). In order to increase the credibility of the simulation, we just unify only operations of the same size.

Table 13 shows the result of Toom–Cook 4 with an RSA/ECC coprocessor. Toom–Cook 4 is Saber's application method for reference and optimized assembly code. In Toom–Cook 4, there are seven multiplications, which are also called pointwise multiplication. The degree of multiplication is 64, so this is the same as the application when Karatsuba or DM uses 4-way separation. Karatsuba needs nine multiplications, but Toom–Cook 4 with KS requires 2 fewer multiplications. However, the final result is worse than that of Karatsuba because the interpolation and evaluation step need many more operations, which account for 59.5% in Toom-Cook 4.

Table 14 summarizes all implementation results without the number of each operation. The best performance of 'C' implementation in Exynos2100 is Karatsuba ($k = 2$), because the proportion of the remaining operations, except for pointwise multiplication, is relatively low, and the number of pointwise multiplications is also low. Thus, we mainly use Karatsuba($k = 2$) in the Saber implementation for comparison with the result of only using software.

C. APPLICATION TO 3 ROUND KEM BASED ON LWE

In our experiments, we implement CRYSTALS Kyber and Saber. The main difference between these two candidates is whether NTT is used or not. As our purpose in this study is to analyze and compare each KS and variant implementation, we refer to CRYSTALS Kyber clean code [14] and Saber reference code [13]. We do not use AVX2, Cortex-M4 optimization, or PQC variants that utilize SHA2 and AES because,

- The instructions for each ARM architecture are all different. The available optimized code based on Cortex M4 in the NIST submission mainly uses DSP instructions [54], which have parallel operations and also known as SIMD(Single Instruction Multiple Data). However, in our environment, DSP is not available.
- AVX2 is only for Intel CPU [55], so we do not use optimization with AVX2.

- In our environment, hardware for AES [46] acceleration exists, but we see the main submissions are consistent with SHA3. There are PQC variants with sampling using AES. M.R. Albrecht *et al.* [7] shows KS with an AES sampler. It can achieve high performance due to the hardware. However, we do not use AES hardware for the implementation of the actual submitted algorithms.

Our strategy for applying KS is Karatsuba $k = 2$, because this achieved the best performance of our experimental results in the previous section. Other results of KS variants will be discussed in the simulation section. We implement only algorithms with level 5 security, e.g., Kyber1024 and FireSaber, because level 5 algorithms show more characteristics of the impacts of each configuration

1) SABER

Saber can directly utilize KS techniques because pure polynomial multiplication is used in Algorithm 3. The technique in the reference code is Toom–Cook 4, quoted in Bernstein [22]. Table 15 shows a comparison between Toom–Cook 4 and KS with an RSA/ECC coprocessor by Karatsuba $k = 2$, which is the same result an in Table 8 and in the 'C' implementation of Table 21. The relative operation ratio in the Tables means the relative ratio of the third columns (in this case, Karatsuba k = 2) to the second column (in this case, Toom–Cook).

Only 15.4% of the time (clocks) is required for polynomial multiplication with KS compared with Toom–Cook using only Software. The result of Saber is shown in Table 16. The result using KS is approximately 3× faster compared with using Software. Surprisingly, the performance is quite enhanced by changing the polynomial multiplication to the KS technique.

2) CRYSTALS KYBER

CRYSTALS Kyber uses NTT, which is similar to Saber, which uses Toom–Cook 4 for ring multiplication. There are three steps for polynomial multiplication in NTT, such as forward NTT, inverse NTT, and pointwise multiplication. All of these three steps must be sequentially combined for one multiplication. Table 17 shows a comparison between NTT and Karatsuba $k = 2$ with KS. The total number of clock cycles of NTT for one polynomial multiplication is 54644, which is a very high cost compared with Karatsuba $k = 2$, when NTT domain changes happen.

Every variable for multiplication in the original submitted algorithm, described in Algorithm 2, are in represented in the NTT domain. As we introduced in Algorithm 1, the basic structure is based on multiplication of ring $\mathbb{Z}_q[x]/(x^{256}+1)$. Although NTT reduces the cost of operations from $O(n^2)$ to $O(n(log(n)))$, the sequence from NTT to iNTT (inverse NTT) is less than 2× faster than Toom–Cook 4. Additionally, Saber does not have coefficient reduction by q. Thus, to speed up the algorithm, Kyber uses NTT domain representation directly for the interface. This dramatically reduces the number of total NTT and iNTT in matrix A in Algorithm 1 and Algorithm 2. We call Algorithm 2 the Kyber NTT domain, denoted Kyber NTTD, and Algorithm 1 Kyber NTT multiplication, denoted Kyber NTTM without NTT domain parameters.

Thus, the final submission for the parameters of the public key and private key with Kyber NTTD is even faster than with Kyber NTTM (see Table 18). To apply KS, Kyber NTTD requires many inverse NTT operations for normal domain operations by matrix A in Algorithm 2. Therefore, the result of Karatsuba k = 2 incurs greater operation cost than does NTT with only software. Karatsuba 2-way separation is compared with Kyber NTTD in Table 19 and Kyber NTTM in Table 18.

Indeed, Kyber NTTM is more appropriate for KS. Table 20 shows the estimated number of clock cycles for CRYSTALS Kyber Algorithm 1 (Kyber NTTM), software alone vs. Kyber with the KS variant, Karatsuba $k = 2$. There is an enhancement with KS, unlike the application to Kyber NTTD.

In conclusion, CRYSTALS Kyber is not suitable for modular multiplication with integer arithmetic due to its NTT domain representation. The only way it is meaningful for KS is if the algorithm is changed to an NTTM-like scheme, which is Algorithm 1.

3) APPLICATION TO OTHER ALGORITHMS IN NIST PQC ROUND 3 CANDIDATES

- NTRU [49] is a candidate of NIST Round 3 candidate KEM. This algorithm also has polynomial multiplications and reductions. The coefficient of reduction is a form of 2^k or 3. The degree of the polynomial is approximately from 500 to 900, so many more separations are required. NTRU is not dedicated to using NTT. It may be more suitable to use RSA/ECC coprocessors, because polynomial multiplication is more than 98% in the encapsulation of NTRU algorithms. The disadvantage of KS is the operation cost by splitting; we already showed this property in Section VII-B. This implementation will be our future work.
- CRYSTALS Dilithium [16] has almost the same property as CRYSTALS Kyber. In order to enhance the performance of the algorithm, signature generation is the main target, because there are many rejected steps that cause unpredictable redundancy. Matrix A is generated and stored in an NTT representation like Kyber. The size of the matrix in Dilithium 5 that is submitted with security parameters that satisfy security level 5 is 8×7, so 56 inverse NTT operations are required by default to utilize KS. Moreover, the coefficient in Dilithium is $\mathbb{Z}_q$ where $q = 2^{23} - 2^{13} + 1$. The minimum size after evaluation by 2^l is 14080 bits, which requires at least 4-way separation. As a result, Round 3 Dilithium is not suitable for KS. As a Kyber example, we can suggest a modified specification, but it does not achieve better performance compared with an NTT-based scheme.
- FALCON [50] is a signature algorithm in the NIST Round 3 candidates. Most operations are based on floating-point operations. There is no RSA/ECC

TABLE 14. Number of clock cycles and operation portion where DM [7] ($k = 2, k = 4$), Karatsuba [7] ($k = 2, k = 4$), Kronecker+ [8]($k = 2, k = 4$) and Toom-Cook 4 with Kronecker Substitution - estimated by 'C' implementation optimized level 0. The empty space(-) means that the operation is not applied for the method(Exynos2100 SSP).

Sub-operations	DM [7]		Karatsuba [7]		Kronecker+ [8]		Toom-Cook 4
	DM $k = 2$	DM [7] $k = 4$	Karatsuba [7] $k = 2$	Karatsuba [7] $k = 4$	Kronecker+ [8] $k = 2$	Kronecker+ [8] $k = 4$	
Pointwise Multiplication	13480	20400	10010	11475	5318	3678	8925
Polynomial separation	1422	1422	1422	1422	-	-	-
Polynomial addition	1250	3438	2499	2500	-	-	-
Polynomial subtraction	-	-	675	938	-	-	-
Polynomial shift	475	475	475	475	-	-	-
Polynomial re-combination	688	688	688	688	688	688	-
Evaluation	-	-	-	-	1300	1300	-
Number reduction	-	-	-	-	4375	5000	-
Number shift	-	-	-	-	3352	7104	-
Number addition	-	-	-	-	1250	3750	-
Negative Processing	-	-	-	-	1422	1422	-
Interpolation and Evaluation	-	-	-	-	-	-	13166
Total clock cycles	17315	26423	15769	17498	17905	22942	22091
Pointwise Multiplication	77.9%	77.2%	63.5%	65.6%	29.7%	16.0%	40.4%
Polynomial separation	8.2%	5.4%	9.0%	8.1%	-	-	-
Polynomial addition	7.2%	13.0%	15.8%	14.3%	-	-	-
Polynomial subtraction	-	-	4.3%	5.4%	-	-	-
Polynomial shift	2.7%	1.8%	3.0%	2.7%	-	-	-
Polynomial re-combination	4.0%	2.6%	4.4%	3.9%	3.8%	3.0%	-
Evaluation	-	-	-	-	7.3%	5.7%	-
Number reduction	-	-	-	-	24.4%	21.8%	-
Number shift	-	-	-	-	19.8%	31.0%	-
Number addition	-	-	-	-	7.0%	16.3%	-
Negative Processing	-	-	-	-	7.9%	6.2%	-
Interpolation and Evaluation	-	-	-	-	-	-	59.6%
Total portion	100%	100%	100%	100%	100%	100%	100%

TABLE 15. Estimated Number of clock cycles for Toom-Cook 4(Software only) vs Karatsuba k = 2(with RSA/ECC co-processor) in Exynos2100 SSP.

	Toom-Cook 4	Karatsuba 2-way separation	performance improvement
clocks	98356	15195	6.47×

TABLE 16. Estimated Number of clock cycles for Saber(Software only) vs Saber with Karatsuba k = 2(KS with RSA/ECC Coprocessor) in Exynos2100 SSP.

	Software only(Toom-Cook 4)	Karatsuba 2-way separation	performance improvement
KeyGen	1954943	634613	3.08×
Enc	2426347	778390	3.11×
Dec	2767131	790786	3.49×

TABLE 17. Estimated Number of clock cycles for NTT(Software only) vs Karatsuba k = 2(KS with RSA/ECC Coprocessor) in Exynos2100 SSP.

NTT+iNTT+Pointwise Multiplication	Karatsuba 2-way separation(KS)	performance improvement
54644	15195	3.59×

TABLE 18. Kyber performance, Round 3 Submission(Algorithm 2, Kyber NTTD, Software only) vs Karatsuba 2-way separation(Algorithm 1, Kyber NTTM with RSA-ECC Coprocessor) in Exynos2100 SSP.

	Referece	Karatsuba 2-way separation	performance improvement
Keygen	748885	1168651	0.64×
Enc	924737	1308096	0.70×
Dec	1008004	1468035	0.68×

TABLE 19. Estimated Number of clock cycles for Crystals Algorithm 2(Kyber NTTD), Kyber Software only vs Kyber with Karatsuba k=2(KS with RSA-ECC Coprocessor) in Exynos2100 SSP.

	Reference	Karatsuba 2-way separation	performance improvement
Keygen	748885	1329017	0.56×
Enc	924737	1431882	0.64×
Dec	1008004	1555241	0.64×

TABLE 20. Estimated Number of clock cycles for Crystals Kyber Algorithm 1(Kyber NTTM), Kyber Software only vs Kyber with Karatsuba k=2(KS with RSA-ECC Coprocessor) in Exynos2100 SSP.

	Reference (modified)	Karatsuba 2-way separation(KS)	performance improvement
Keygen	1168651	723360	1.61×
Enc	1308096	866490	1.50×
Dec	1468035	876770	1.67×

coprocessors that support floating operations. Recently, an integer-based structure was suggested, the so called, ZALCON [52], with which we can find a way to use RSA/ECC coprocessors with integer-based schemes.

- Rainbow [53] is also one of the three NIST post-quantum signature finalists. The dominant operation of rainbow is matrix operation. Each element is in Galois Field such as $GF(4)$, $GF(16)$, or $GF(256)$, which is not directly related to polynomial operation. This is also unsuitable for utilizing RSA/ECC coprocessors directly.

TABLE 21. Comparison of multiplication among environments with reference H/W + Software(Assembly) - clock cycles(smaller is faster).

	C implementation(optimized level 0)					Assembly Optimized				
Final techniques	MM only (Exynos)	+Add(Sub)	+BS	+Add(Sub) +BS	Add(Sub) BS Only	MM only (Exynos)	+Add(Sub)	+BS	+Add(Sub) +BS	Add(Sub) BS Only
DM(t=2)	17315	16085	17315	16085	∞	10882	10574	10789	10481	∞
Karatsuba(t=2)	**15195**	12735	15195	12735	N/A	8747	8131	8654	8038	∞
Kronecker+(t=2)	17905	**12370**	**14393**	**7308**	∞	**8335**	**6949**	**7327**	5941	∞
DM(t=4)	26423	23040	26423	23040	∞	15228	14831	15135	14288	∞
Karatsuba(t=4)	17498	14115	17498	14115	∞	9268	8421	9175	8328	∞
Kronecker+(t=4)	22942	14332	15918	8858	∞	8683	6527	8047	**4511**	∞
Shift&Add	∞	∞	∞	15360	**15360**	∞	∞	∞	15360	**15360**
Toom-Cook4(KS)	22091	22091	22091	22091	∞	11185	11185	11185	11185	∞
Toom-Cook4(Software only)	98356	98356	98356	98356	98356	34292	34292	34292	34292	34292
Multi-Moduli NTT(Software only) [60]	11533	11533	11533	11533	11533	11533	11533	11533	11533	11533

- Classic McEliece [51] has matrix multiplications in finite extension fields($GF(2^m)$) dominantly. It is also not directly related to polynomial operation of integers.
- Other PQC algorithms based on Ring-LWE also can be applied for KS not in NIST round 3 candidates.They have the same properties to CRYSTALS Kyber or Saber.

VIII. SIMULATION BY HARDWARE PERFORMANCE VARIATION

We show the result of simulations in different environments with three methods. In our Exynos2100 SSP environment [56], only modular multiplication is supported by hardware; other operations must be implemented by Software. However, there are several environments that have different hardware IP. Table 21 shows the result of simulating the clock cycles of various IP hardware, such as Modular Multiplication only (MM only), modular multiplication + Barrel Shifter (+BS), modular multiplication + Addition(+Add(Sub)), modular multiplication + barrel shifter + addition (+BS +Add(Sub)). Table 21 shows the results of implementation by 'C' and assembly optimized implementations, which have the same environment as the experimental result of Exynos2100. Here are some considerations of the simulation results. The hardware performance is referred to as relative speed in Table 22.

- Clock cycles of reduction by modular $2^{nl}+1$ is the same as clock cycles of subtraction. Thus, the hardware clock cycles of subtraction, addition, and reduction are equivalent.
- All hardware operates in $\mathbb{Z}$, so polynomial addition and subtraction are also not applied directly as in modular multiplication. However, after converting the numbers in $\mathbb{Z}$ to polynomials, which is called negative processing, addition and subtraction can be carried out with hardware. Thus, every polynomial addition and subtraction is changed to operations in $\mathbb{Z}$.
- We cannot know the exact performance of addition and shift. This depends on how the register transistor location is designed and the size of the hardware. One thing that is known is that both need few operations. We set

TABLE 22. Estimated ratio of each operations on H/W VS Software for 4096 bits.

Operation	H/W	Software
Modular Multiplication	1	N/A
Number Addition	0.0066	0.4112
Number Subtraction	0.0066	0.4112
Number Shifter	0.0066	0.5842

operation time of addition and shifter are 0.0066 both comparing with the time operation of the modular multiplication equals 1. More important than the absolute performance is the relative performance with software. Even if the performance is very fast compared with software, the result in Table 21 should be similar.
- The software performance of modular multiplication is not useful information for us. It is definitely extremely high. It also even slower than Software NTT or Toom–Cook.
- The result of the Shift&Add algorithm is valuable when both shifter and adder exist, because one of them is used hundreds of times. Using Software in one of the operations is too slow. In addition, this method is affected by the absolute performance of addition and shift hardware. Thus, one needs clocks of real-world performance of different hardware if the method is considered one's own product.

According to the result in Table 21, if addition or/and shift hardware is added to MM, Kronecker+ produces better efficiency. However, in the real world, we should note that hardware for addition that supports more than 1024 bits is not necessary for RSA and ECC. For similar reasons, shift hardware is also not essential. Therefore, the algorithms in Table 21 that is most appropriate should be determined according to each situation.

The assembly code is implemented in an ARM Cortex M35P-based Exynos2100 environment, which does not have SIMD (DSP). Therefore, assembly optimized code is indeed not really different from 'C' code with an optimized level -O3 (or -Ofast). However, the 'C' code result is compiled with optimized level -O0, so the result of the assembly

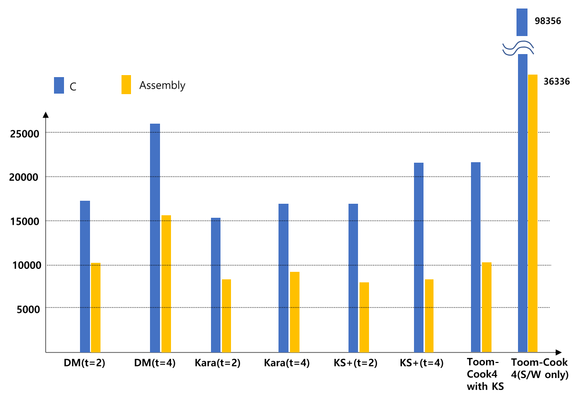


FIGURE 2. Performance Comparison in Exynos2100 with Modular multiplication(Assembly Optimization).

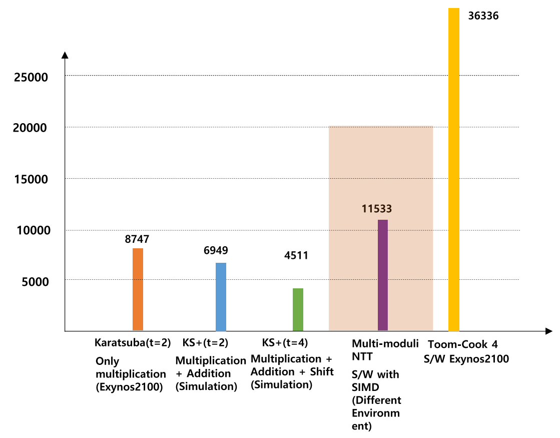


FIGURE 3. Performance Comparison of "best case" in each environment(Table 21 and Multi-moduli NTT [60]).

code is much faster than that of 'C' code. The infinity symbol (∞) means two things. The first is not applicable, for example, if a given device supports only Add(Sub) and Shift, as DM($t = 2$) cannot be implemented. The second is that the expected performance must be extremely slow, for example, the Shift&Add method can be implemented without hardware "Shifter and Adder" but it is meaningless to estimate.

Figure 2 shows a comparison of the results in Exynos2100 only, with the results in blue color ('C' code) and in yellow (Assembly) in Table 21. Each result represents the result of the column on the far left side of the table.

Figure 3 shows the results of the best performance for each hardware set. Additionally, one more result is added in Figure 3, namely multi-moduli NTT, a method proposed by A. Abdularhan *et al.* [60]. This method achieves the best results as far as we know. The study implemented Saber using NTT in Cortex M3, and in Cortex M4 using SIMD. Our environment is similar to Cortex M3. This is because "no floating-point registers" and "no SIMD instructions" are the same in our environment. The clock cycles of multi-moduli NTT are converted to our timer clocks for the comparison, so the estimated clock cycles is different from the number in the referenced paper.

TABLE 23. Comparison between Saber Software vs Saber with Kronecker+ k = 2(simulation).

	Reference	Kronecker+ (simulation)	performance improvement
KeyGen	1954943	509393	3.83×
Enc	2426347	622098	3.90×
Dec	2767131	603350	4.58×

Finally, Table 23 is the result of Saber simulation with Kronecker+, which is an environment with hardware addition and shifter using 'c' code implementation. Indeed, this is the best case of our simulation. The result makes Saber performance approximately four times faster without any hardware support for the sampler, for example, AES or SHA of hardware.

IX. CONCLUSION

Recent studies on the optimization of PQC have mainly been conducted to develop new dedicated hardware or Software with special instructions, such as SIMD. However, for rapid commercialization, if a legacy hardware accelerator is used, it can make a meaningful contribution. This paper shows the value of reusing legacy RSA/ECC coprocessors and its implementation result using various specific methods. Developers can determine the best way by referring to these implementation results.

In this paper, we showed and explained the following findings:

- We explained the method of using existing legacy hardware in the implementation of the PQC candidate algorithm and various limitations to be considered when implementing it.
- We also showed that it actually works by implementing Karatsuba, Toom–Cook, Kronecker+, and a variety of variants of KS using legacy RSA/ECC coprocessors on an Exynos2100 platform, which is currently commercialized and used in mobile phones.
- We subdivided the operations of polynomial multiplication for each method into several steps and expressed the ratio of time-consumption to analyze the computational characteristics of each method and facilitate performance prediction when a specific step is replaced with hardware.
- The implementation results and predictive performance for various methods in several legacy hardware setups were simulated and shown.
- Through comparison of the performance for polynomial multiplication with the fastest results [60] recently implemented based on NTT, this paper also showed that the performance of legacy RSA/ECC hardware accelerators was better.

In the future, we plan to conduct the following studies:

- Research on more efficient methods based on KS.
- A study to apply legacy RSA/ECC hardware accelerators to other PQC candidate algorithms that is based on not only LWE lattice cryptography but also on other mathematical foundations.

REFERENCES

[1] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proc. 35th Annu. Symp. Found. Comput. Sci.*, Nov. 1994, pp. 124–134.

[2] W. Beullens, J. D'Anvers, A. Hülsing, T. Lange, L. Panny, C. Guilhem, and N. P. Smart. (May 3, 2021). Post quantum cryptography: Current state and quantum mitigation. Enisa. [Online]. Available: https://www.enisa.europa.eu/publications/post-quantum-cryptography-current-state-and-quantum-mitigation

[3] NIST. *Announcing Request for Nominations for Public-Key Post-Quantum Cryptographic Algorithms*. Accessed: Dec. 2016. [Online]. Available: https://federalregister.gov/a/2016-30615

[4] NIST. *NIST 3rd Round Finalist and Alternate Candidates*. Accessed: Oct. 2020. [Online]. Available: https://csrc.nist.gov/News/2020/pqc-third-round-candidate-announcement

[5] D. Harvey, "Faster polynomial multiplication via multipoint Kronecker substitution," *J. Symbolic Comput.*, vol. 44, no. 10, pp. 1502–1510, Oct. 2009.

[6] F. Yaman, A. C. Mert, E. Öztürk, and E. Savas, "A hardware accelerator for polynomial multiplication operation of CRYSTALS-KYBER PQC scheme," in *Proc. DATE*, Feb. 2021, pp. 1020–1025.

[7] M. R. Albrecht, C. Hanser, A. Hoeller, T. Pöppelmann, F. Virdia, and A. Wallner, "Implementing RLWE-based schemes using an RSA co-processor," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2019, no. 1, pp. 169–208, Nov. 2018.

[8] J. W. Bos, J. Renes, and C. van Vredendaal, "Post-quantum cryptography with contemporary coprocessors beyond Kronecker, Schönhage–Strassen & Nussbaumer," Cryptol. ePrint Arch. Rep. 2020/1303. [Online]. Available: https://ia.cr/2020/1303

[9] A. Greuet, S. Montoya, and G. Renault, "On using RSA/ECC coprocessor for ideal lattice-based key exchange," in *Proc. COSADE*, Oct. 2021, pp. 205–227.

[10] R. J. Fateman, "Can you save time in multiplying polynomials by encoding them as integers?" Univ. California Berkeley, Berkeley, CA, USA, Tech. Rep. CA 947220-1776, Aug. 2010. [Online]. Available: http://people.eecs.berkeley.edu/~fateman/papers/polysbyGMP.pdf

[11] D. E. Knuth, *The Art of Computer Programming*, vol. 2. Reading, MA, USA: Addison-Wesley. 1981.

[12] A. Langlois and D. Stehlé, "Worst-case to average-case reductions for module lattices," *Des., Codes Cryptogr.*, vol. 75, no. 3, pp. 565–599, Jun. 2015, doi: 10.1007/s10623-014-9938-4.

[13] SABER. (Oct. 21, 2020). *SABER: Mod-LWR Based KEM(Round 3 Submission), NIST PQC Round 3 Submission*. [Online]. Available: https://www.esat.kuleuven.be/cosic/pqcrypto/saber/

[14] National Institute of Standards and Technology. (2019). *CRYSTALS-KYBER*. [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions

[15] T. Pöppelmann and T. Güneysu, "Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware," in *Progress in Cryptology—LATINCRYPT 2012* (Lecture Notes in Computer Science), vol. 7533, A. Hevia and G. Neven, Eds. Berlin, Germany: Springer, 2012, doi: 10.1007/978-3-642-33481-8_8.

[16] *CRYSTALS-Dilithium—Submission to Round 3 of the NIST Post-Quantum Project. Specification Document (Part of the Submission Package)*. Accessed: Oct. 1, 2020. [Online]. Available: https://pq-crystals.org/

[17] EMVco. (Feb. 2016). *Contactless Specifications for Payment System, Book C-2 Kernel 2 Specification, Version 2.6*. [Online]. Available: https://www.emvco.com

[18] Common Criteria. *Samsung SSP01 of S5E9830 With Specific IC Dedicated Software Revision 1.1*. Accessed: Mar. 18, 2018. [Online]. Available: https://www.commoncriteriaportal.org/products/#IC

[19] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor," in *Advances in Cryptology—CRYPTO'86* (Lecture Notes in Computer Science), vol. 263. Berlin, Germany: Springer, pp. 311–323.

[20] P. L. Montgomery, "Modular multiplication without trial division," *Math. Comput.*, vol. 44, no. 170, pp. 519–521, Jan. 1985.

[21] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, FL, USA: CRC Press, 1996.

[22] D. J. Bernstein. (1997). *Multidigit Multiplication for Mathematicians*. [Online]. Available: https://cr.yp.to/papers/m3.pdf

[23] NXP. *NXP Secure Microcontroller SmartMX P71D321*. [Online]. Available: https://www.nxp.com/docs/en/fact-sheet/P71D321.pdf

[24] Infineon. *SLE 78CAFX1M1SPHM*. [Online]. Available: https://www.infineon.com/cms/en/product/security-smart-card-solutions/security-controllers/sle-78/sle-78cafx1m1sphm

[25] Espressif Systems. *ESP32 Technical Reference Manual*. v4.6. Accessed: Nov. 30, 2021. [Online]. Available: https://www.espressif.com/sites/default/files/

[26] L. Kronecker, "Grundzüge einer arithmetischen theorie der algebraischen Grössen," *J. die Reine und Angewandte Math.*, vol. 92, pp. 1–122, 1882.

[27] *Recommendation for Key Management*, Standard NIST SP 800-57, Revision 5, May 2020.

[28] *Mécanismes Cryptographiques—Règles et Recommandations*, ANSSI, Paris, France, Revision 2.03, Feb. 2014.

[29] *Cryptographic Mechanisms: Recommendations and Key Lengths*, Standard BSI TR-02102-1, Version 2020-01, Mar. 2020.

[30] H. Nussbaumer, "Fast polynomial transform algorithms for digital convolution," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-28, no. 2, pp. 205–215, Apr. 1980, doi: 10.1109/TASSP.1980.1163372.

[31] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, no. 90, pp. 297–301, Apr. 1965.

[32] NIST. (Apr. 16, 2015). *SHA-3 Standardization*. [Online]. Available: https://csrc.nist.gov/Projects/Hash-Functions/SHA-3-Project/SHA-3-Standardization

[33] Intel. (Aug. 20, 2013). *AVX-512 Instructions*. [Online]. Available: https://software.intel.com/content/www/us/en/develop/articles/intel-avx-512-instructions.html

[34] J. B. Fraleigh, *A First Course in Abstract Algebra*, 7th ed., 2003.

[35] SAMSUNG. *Exynos2100: Exynos on Official Reply*. Accessed: Jan. 12, 2021. [Online]. Available: https://www.youtube.com/watch?v=qcBqg6Y_cnw

[36] SAMSUNG. *Galaxy S21 5G, S21 + 5G*. [Online]. Available: https://www.samsung.com/us/smartphones/galaxy-s21-5g/

[37] O. Adam, *An Introduction to Microcomputers: Basic Concepts*, vol. 1, 2nd ed. New York, NY, USA: McGraw-Hill, Apr. 1982, pp. 5–64.

[38] Microsoft. (Mar. 4, 2011). *Blocking the SBP-2 Driver to Reduce 1394 DMA Threats to BitLocker*. [Online]. Available: https://support.microsoft.com/en-us/topic/blocking-the-sbp-2-driver-and-thunderbolt-controllers-to-reduce-1394-dma-and-thunderbolt-dma-threats-to-bitlocker-bf0ef10b-f563-5cfc-9740-8340b1d86a0c

[39] U. Banerjee, S. Das, and A. P. Chandrakasan, "Accelerating post-quantum cryptography using an energy-efficient TLS crypto-processor," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Oct. 2020, pp. 1–5, doi: 10.1109/ISCAS45731.2020.9180550.

[40] U. Banerjee, C. Juvekar, A. Wright, Arvind, and A. P. Chandrakasan, "An energy-efficient reconfigurable DTLS cryptographic engine for end-to-end security in IoT applications," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 42–44, doi: 10.1109/ISSCC.2018.8310174.

[41] U. Banerjee, A. Wright, C. Juvekar, M. Waller, Arvind, and A. P. Chandrakasan, "An energy-efficient reconfigurable DTLS cryptographic engine for securing Internet-of-Things applications," *IEEE J. Solid-State Circuits*, vol. 54, no. 8, pp. 2339–2352, Aug. 2019, doi: 10.1109/JSSC.2019.2915203.

[42] R. Azarderakhsh, M. Campagna, C. Costello, L. D. Feo, B. Hess, A. Jalali, D. Jao, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, G. Pereira, J. Renes, V. Soukharev, and D. Urbanik, "Supersingular isogeny key encapsulation," NIST's Post-Quantum Cryptogr. Standardization Process, Round 2, 2019. [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions/SIKE.zip.

[43] Frodo. *FrodoKEM Learning With Errors Key Encapsulation Algorithm Specifications and Supporting Documentation*. Accessed: Mar. 25, 2020. [Online]. Available: https://frodokem.org/files/FrodoKEM-specification-20171130.pdf

[44] ThreeBears. *Post-Quantum Cryptography Proposal: ThreeBears*. Accessed: Nov. 27, 2017. [Online]. Available: https://www.shiftleft.org/papers/threebears/nist-submission.pdf

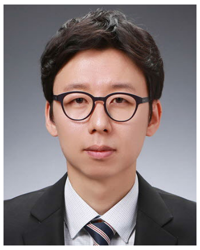
[45] SPHINCS. *PHINCS + Submission to the NIST Post-Quantum Project*. Accessed: Oct. 1, 2020. [Online]. Available: https://frodokem.org/files/FrodoKEM-specification-20171130.pdf

[46] *Announcing the Advanced Encryption Standard (AES)*, Federal Information Processing Standards Publication 197, NIST, Nov. 2001.

[47] *Announcing Approval of FIPS Publication 180-2, Federal Register Notice 02-21599*, NIST, Gaithersburg, MD, USA, Oct. 2008.

[48] D. J. Bernstein, "Fast multiplication and its applications," in *Algorithmic Number Theory*, vol. 44. Berkeley, CA, USA: MSRI Publications, 2008.
[49] NIST. *A Submission to the NIST Post-Quantum Standardization Effort, NIST Round 3 Candidate*. [Online]. Available: https://ntru.org/
[50] FALCON. *Fast-Fourier Lattice-Based Compact Signatires Over NTRU, NIST Round 3 Candidate*. Accessed: Oct. 1, 2020. [Online]. Available: https://falcon-sign.info/
[51] *Classic McEliece, NIST Round 3 Candidate*. Accessed: Oct. 1, 2020. [Online]. Available: https://classic.mceliece.org/
[52] P. Fouque, F. Gérard, M. Rossi, and Y. Yu, "Zalcon: An alternative FPA-free NTRU sampler for Falcon," in *Proc. 3rd NIST PQC Workshop*, Jun. 2021, pp. 1–23.
[53] RAINBOW. *One of the Three NIST Post-Quantum Signature Finalists*. Accessed: Oct. 1, 2020. [Online]. Available: https://www.pqcrainbow.org/
[54] J. Yiu, "Blending DSP and ML features into a low-power general-purpose processor—How far can we go?" ARM, Cambridge, U.K., White Paper, 2020. [Online]. Available: https://armkeil.blob.core.windows.net/developer/Files/pdf/white-paper/blending-dsp-and-ml-features-into-a-low-power-general-purpose-processor.pdf
[55] G. Lento, "Optimizing performance with Intel advanced vector extensions," AVX, Fountain Inn, SC, USA, White Paper, Sep. 2014. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documentSoftwarehite-papers/performance-xeon-e5-v3-advanced-vector-extensions-paper.pdf
[56] NIST. *CAVP Samsung Smart Secure Platform*. Accessed: Oct. 26, 2020. [Online]. Available: https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/details?product=13329
[57] ETSI. *Smart Secure Platform (SSP); Part 1: General Characteristics (Release 15)*. Accessed: Mar. 1, 2019. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/103600_103699/10366601/15.00.00_60/ts_10366601v150000p.pdf
[58] H. Seo, "Compact implementations of curve Ed448 on low-end IoT platforms," *ETRI J.*, vol. 41, no. 6, pp. 863–872, Dec. 2019.
[59] A. Schönhage, "Asymptotically fast algorithms for the numerical muitiplication and division of polynomials with complex coefficients," in *Proc. EUROCAM*, in Lecture Notes in Computer Science, vol. 144, Apr. 1982, pp. 3–15.
[60] A. Abdulrahman, J. Chen, Y. Che, V. Hwang, M. J. Kannwischer, and B. Yang, "Multi-moduli NTTs for Saber on Cortex-M3 and Cortex-M4," Cryptol. ePrint Arch. Rep. 2021/995. [Online]. Available: https://ia.cr/2021/995

JONG-YEON PARK received the master's degree in mathematics from Kookmin University, in 2012. He was a Researcher with the Electronics and Telecommunications Research Institute (ETRI), Daejeon, South Korea, from 2012 to 2014. He was also a Research Engineer with Korea Telecom (KT) Convergence Laboratory, Seoul, South Korea, from 2015 to 2017. He is currently a Staff Engineer with Samsung Electronics, System LSI. His research interests include most of cryptographer's topics, especially mathematical structures related in secure algorithms and SCA.



YONG-HYUK MOON (Member, IEEE) received the M.S. and Ph.D. degrees in information and communication engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2006 and 2013, respectively. He has been a Senior Researcher with the Artificial Intelligence Laboratory, Electronics and Telecommunications Research Institute (ETRI), Daejeon, since 2006. He also started working as an Associate Professor with the Computer Software Department, University of Science and Technology (UST), Daejeon, in September 2021. His research interests include automated machine learning, deep learning, optimization, device security, and edge computing.

WONIL LEE received the B.S., M.S., and Ph.D. degrees in mathematics from Korea University, in 1998, 2000, and 2004, respectively. From 2004 to 2005, he was a Researcher studying provable security of cryptographic hash function with Kyushu University, Fukuoka, Japan. Since 2005, he has been working as a Principal Engineer with Smart Card and Security Industry, Samsung Electronic Ltd. His research interests include security IC silicon security, near field communication technology, and system security for mobile and the IoT.

SUNG-HYUN KIM received the B.S. degree in electronic engineering and the M.S. degree in parallel processing computer from Kyungpook National University, in 1987 and 1989, respectively. From 1989 to 1996, he engaged in research and development with Automated Fingerprint Identification System for Korea National Police Agency. Since 1996, he has been working with Smart Card and Security Industry, Samsung Electronic Ltd. He currently works as an Architect and a Principal Engineer of advanced system security and future security and cryptography technology. His research interests include security IC silicon security, chip operating systems (COS), and system security for mobile, the IoT, and cloud.

KOUICHI SAKURAI (Member, IEEE) received the B.S. degree in mathematics from the Faculty of Science, Kyushu University, in 1986, and the M.S. degree in applied science and the Ph.D. degree in engineering from the Faculty of Engineering, Kyushu University, in 1988 and 1993, respectively. From 1988 to 1994, he was engaged in research and development on cryptography and information security with the Computer and Information Systems Laboratory, Mitsubishi Electric Corporation. Since 1994, he has been working with the Department of Computer Science, Kyushu University, as an Associate Professor, where he became a Full Professor, in 2002. From 2005 to 2006, he was successful in generating such co-operation between Japan, China, and South Korea, for security technologies, as a Leader of the Cooperative International Research Project supported by the National Institute of Information and Communications Technology (NICT). Moreover, in March 2006, he established research co-operations under a Memorandum of Understanding in the field of information security with Prof. Bimal Kumar Roy, the first time Japan has partnered with The Cryptology Research Society of India (CRSI). He is currently working with the Institute of Systems, Information Technologies and Nanotechnologies, as the Chief of the Information Security Laboratory, for promoting research co-operations among the industry, university, and government under the theme enhancing IT-security in social systems. He also directs the Laboratory for Information Technology and Multimedia Security and is working with the Cybersecurity Center, Kyushu University. He is also with the Department of Advanced Security, Advanced Telecommunications Research Institute International and involved in a NEDO-SIP-Project on supply chain security. He has published about 400 academic articles in cryptography and cybersecurity.