Efficient Word Size Modular Multiplication over Signed Integers

Daichi Aoki
Secure System Platform Research Labs.
NEC Corporation
Japan
daichi_aoki@nec.com

Kazuhiko Minematsu
Secure System Platform Research Labs.
NEC Corporation
Japan
k-minematsu@nec.com

Toshihiko Okamura
Secure System Platform Research Labs.
NEC Corporation
Japan
t_okamura@nec.com

Tsuyoshi Takagi
Department of Mathematical Informatics
University of Tokyo
Japan
takagi@mist.i.u-tokyo.ac.jp

***Abstract*—As an efficient multiplication method for polynomial rings, Number Theoretic Transform (NTT) is a fundamental algorithm that is both practically useful and theoretically established. Chung et al. proposed a method to perform NTT-based polynomial multiplication for NTT-unfriendly rings that do not have suitable primitive roots. They applied their proposal to lattice-based cryptography using NTT-unfriendly rings and speeded up several schemes. At ARITH 2021, Plantard proposed a modular multiplication algorithm that improves the speed of NTT if moduli are not large (a few dozen of bits), which is the case for typical lattice-based cryptography. It is natural to expect that Plantard's method improves Chung et al.'s NTT when applied to them, however, this is not possible as Chung et al. requires the use of signed integers while Plantard's method assumes unsigned integers. A simple fix would cause a slowdown and a non-constant-time operation. To overcome this problem, we propose an efficient method for calculating the modular multiplication for signed integers based on Plantard's method. Our proposal generally incurs no overhead from the original and works in a constant-time fashion. To show the effectiveness of our proposal, we provide experimental implementation results on a lattice-based cryptographic scheme Saber. Currently, NIST is selecting candidates for standardization of post-quantum cryptography in preparation for the compromise of current public key cryptography by quantum computers, and has completed the selection of the final candidates. Saber is one of the finalists for the NIST standardization project.**

***Index Terms*—Modular Multiplication, Polynomial Multiplication, Number Theoretic Transform, Post-Quantum Cryptography, Saber.**

I. Introduction

Various efficient multiplication techniques for large integers or polynomials are known, including Karatsuba [1], Toom-Cook [2], [3] and Number Theoretic Transform [4]. Number Theoretic Transform (NTT) is a practically useful and theoretically established fundamental algorithm as a polynomial multiplication method. Chung et al. [5] proposed an effective way of performing NTT-based polynomial multiplication for NTT-unfriendly rings that do not have suitable primitive roots of unity. They applied their proposal to several lattice-based cryptographic schemes using NTT-unfriendly rings and significantly improved their speed in ARM Microcontrollers and Intel platforms.

The efficiency of the modular arithmetic is also important for speeding up lattice-based cryptography. This is also true for RSA [6] and Elliptic Curve Cryptography (ECC) [7], which have been actively studied for a long time. Popular techniques, such as Montgomery reduction [8] and Barrett reduction [9], perform modular arithmetic and are particularly suitable for large moduli, say hundreds or larger. Since lattice-based cryptography typically uses smaller moduli, such as a few dozen bits, these techniques are not necessarily the fastest way to calculate the modular arithmetic.

At ARITH 2021, Plantard [10] proposed a novel algorithm to compute modular multiplications for one word size moduli, i.e. 32 or 64bits[1]. Plantard's algorithm increases the speed of NTTs with word-size moduli by reducing the number of multiplications. It is natural to expect that Chung et al.'s NTT [5] could improve the overall performance by using [10] as the internal modular arithmetic routine. Unfortunately, this is not possible as [5] requires the use of signed integers while [10] assumes unsigned integers. A simple fix, such as converting a signed integer to an unsigned integer and then using Plantard's technique, would cause a slowdown and a non-constant-time operation. Constant-time implementations are one of the major countermeasures against timing attacks, which extract secret information by measuring the differences in execution time that depend on secret elements. Since the timing attack on RSA [11] was proposed, research has been active and various timing attacks on modular arithmetic and lattice-based schemes have been proposed [12]–[18].

To overcome this problem, we propose a modular multiplication algorithm for signed integers based on [10]. We provide experimental results on a lattice-based cryptography Saber [19]. In 2016, the U.S. National Institute of Standards

[1]Reference [10] was first published at ARITH 2021 and later its journal version was published at IEEE TDSC.

and Technology (NIST) announced the standardization project for post-quantum cryptography in preparation for the compromise of current public key cryptography. NIST called for candidates and 82 schemes were submitted in 2017. Then, 15 schemes (7 finalists and 8 alternates) were selected in the 3rd round started in July 2020, and Saber was one of the finalists. Saber's security is based on the hardness of the Module Learning With Rounding (MLWR) problem [20]. NIST rated Saber as a highly efficient scheme and suitable for general-purpose applications [21].

As mentioned earlier, we propose an efficient algorithm to quickly compute modular multiplication for signed integers based on [10], which is originally defined over unsigned integers. The number of multiplications in our proposal is the same as [10], and the total number of instructions is almost the same. The instructions used in our method are only basic operations, as Plantard's one. Since our algorithm does not have a branching based on a secret value, we expect that constant-time implementation is possible (if constant-time implementation of [10] is possible). We provide experimental implementation results on a lattice-based cryptographic scheme Saber [19]. The experimental results show that NTT-based polynomial multiplication using our technique is faster than that using Montgomery multiplication.

Organization of the paper: In Section II, we introduce notations on Montgomery reduction, NTT and Chung et al.'s method. In Section III, we describe Plantard's modular multiplication algorithm. In Section IV, we present our new modular multiplication for signed integers. In Section V, we show experimental results compared to Montgomery multiplication. Section VI will conclude this work.

II. Preliminaries

A. Rounding

We define two operations to round real numbers to integers.

Definition 1 (Rounding)**.** For a real number x, we denote by $\lfloor x \rfloor$ the integer n satisfying $n \le x < n+1$. We also denote by $\lfloor x \rceil$ the integer n satisfying $n - \frac{1}{2} \le x < n + \frac{1}{2}$.

For any real number x and any integer N, $\lfloor x + N \rfloor = \lfloor x \rfloor + N$, $\lfloor x + N \rceil = \lfloor x \rceil + N$ and $\lfloor x + \frac{1}{2} \rfloor = \lfloor x \rceil$ hold.

For an integer $N > 0$, we denote the unsigned interval $U_N := \{0, 1, \ldots, N-1\}$ and the signed interval $S_N := \{-\lfloor \frac{N}{2} \rfloor, -\lfloor \frac{N}{2} \rfloor + 1, \ldots, \lfloor \frac{N-1}{2} \rfloor - 1, \lfloor \frac{N-1}{2} \rfloor\}$. For $z \in \mathbb{Z}$, we denote $z \bmod N \in U_N$ and $z \bmod^{\pm} N \in S_N$ the unique representatives in U_N and S_N respectively. The following equations hold;

$$z \bmod N = z - N \left\lfloor \frac{z}{N} \right\rfloor, \; z \bmod^{\pm} N = z - N \left\lfloor \frac{z}{N} \right\rceil \quad (1)$$

Each of them follows from $0 \le \frac{z \bmod N}{N} < 1$, $-\frac{1}{2} \le \frac{z \bmod^{\pm} N}{N} < \frac{1}{2}$ respectively.

B. Montgomery Multiplication

The calculation of modular multiplication $AB \bmod N$ can be done by using (1), but it is very costly to perform division by general integers on a computer. On the other hand, division by powers of two can be implemented using the shift operation, which is less expensive. Therefore, Montgomery multiplication [8] is often used, which can compute modular multiplications using division by a power of two. The calculation procedure of Montgomery multiplication is shown in Algorithm 1.

We briefly check that the algorithm is correct. First, the following equation holds;

$$\begin{aligned}
&(AB + P \cdot (ABR \bmod 2^n)) \bmod 2^n \\
=&(AB + P \cdot (AB(-P^{-1}) \bmod 2^n)) \bmod 2^n \\
=&(AB - AB) \bmod 2^n \\
=&0.
\end{aligned}$$

Therefore, there is an integer k exists such that

$$AB + P \cdot (ABR \bmod 2^n) = k2^n$$

holds. Then we get $AB2^{-n} \bmod P = k \bmod P$. The range of value of k is

$$0 \le k = \frac{AB + P \cdot (ABR \bmod 2^n)}{2^n} < \frac{P^2 + P2^n}{2^n} < 2P.$$

Finally, the correction step (line 3-5 of Algorithm 1) yields an output that satisfies $0 \le C < P$ and $C = AB2^{-n} \bmod P$.

If $P < 2^{n-2}$, then the correction step can be omitted. Precisely, $0 \le C < 2^{n-1}$ holds for $0 \le A, B < 2^{n-1}$ without correction. This is because

$$\begin{aligned}
0 \le k &= \frac{AB + P \cdot (ABR \bmod 2^n)}{2^n} \\
&< \frac{(2^{n-1})^2 + P2^n}{2^n} \\
&< 2^{n-2} + 2^{n-2} = 2^{n-1}.
\end{aligned}$$

This redundancy is often used in Montgomery multiplication.

Algorithm 1 Montgomery Multiplication [8]

```
Input: $A, B, P, R, n$ with odd modulus $P < 2^n$, $0 \le A, B < P$ and $R := (-P^{-1}) \bmod 2^n$
Output: $C = AB2^{-n} \bmod P$ with $0 \le C < P$
1: $t \leftarrow AB$
2: $C \leftarrow (t + P \cdot (tR \bmod 2^n))/2^n$
3: if $C \ge P$ then
4:     return $C = C - P$
5: end if
6: return $C$
```

Here we consider the case where signed integers $-\frac{P-1}{2} \le A, B < \frac{P-1}{2}$ are input to Algorithm 1. In this case, there still exists an integer k such that $AB + P \cdot (ABR \bmod 2^n) = k2^n$. However, the range of value of k is different;

$$-\frac{P}{4} < -\frac{|AB|}{2^n} \le k \le \frac{|AB| + P2^n}{2^n} < \frac{P}{4} + P.$$

Algorithm 2 Signed Montgomery Multiplication [8]

```
Input: $A, B, P, R, n$ with odd modulus $P < 2^{n-1}$, $|A|, |B| \leq 2^{n-1}$ and $R := (-P^{-1}) \bmod^{\pm} 2^n$
Output: $C = AB2^{-n} \bmod^{\pm} P$ with $|C| \leq 2^{n-1}$
1: $t \leftarrow AB$
2: return $C = (t + P \cdot (tR \bmod^{\pm} 2^n))/2^n$
```

Therefore, in order for output C to fulfill $|C| \leq \frac{P-1}{2}$, we need to modify the correction step;

$$\textbf{if } C \geq P/2 \textbf{ then return } C \leftarrow C - P.$$

As unsigned version, if $P < 2^{n-2}$, then $|C| \leq \frac{|AB|+P2^n}{2^n} < 2^{n-1}$ for $|A|, |B| < 2^{n-1}$ without correction.

However, by changing $\bmod$ to $\bmod^{\pm}$ we can use a larger P, namely, $P < 2^{n-1}$. Montgomery multiplication with signed integers as input and output is shown in Algorithm 2. We can check the correctness of Algorithm 2 with almost the same logic as Algorithm 1. For $R = (-P^{-1}) \bmod^{\pm} P$,

$$\begin{aligned}&(AB + P \cdot (ABR \bmod^{\pm} 2^n)) \bmod^{\pm} 2^n \\ =&(AB - AB) \bmod^{\pm} 2^n \\ =&0.\end{aligned}$$

Therefore, an integer k such that $AB+P\cdot(ABR \bmod^{\pm} 2^n) = k2^n$ exists and $AB2^{-n} \bmod^{\pm} P = k \bmod^{\pm} P$ holds. In this case, if $P < 2^{n-1}$, the range of value of k is

$$\begin{aligned}|k| &= \frac{|AB + P \cdot (AB(-P^{-1}) \bmod^{\pm} 2^n)|}{2^n} \\ &\leq \frac{|AB| + P2^{n-1}}{2^n} \\ &< \frac{2^{2n-2} + 2^{n-1}2^{n-1}}{2^n} = 2^{n-1}.\end{aligned}$$

Therefore, $|C| \leq 2^{n-1}$ holds for $|A|, |B| \leq 2^{n-1}$ without correction step.

C. NTT-based Multiplication

Number Theoretic Transform (NTT) is well known as a useful algorithm for efficiently computing polynomial multiplication.

Let be $n \geq 1$ an integer and p be prime. Suppose that the quotient ring $R := \mathbb{Z}/p\mathbb{Z}$ has a n-th root of unity ω and an invertible element $\zeta \in R$. We define $\mathsf{NTT}_{n:\zeta:\omega}$ as follows

$$\mathsf{NTT}_{n:\zeta:\omega} : \begin{cases} R[X]/(X^n - \zeta^n) \to \prod_{i=0}^{n-1} R[X]/(X - \zeta\omega^i) \\ f(X) \mapsto (f(\zeta), f(\zeta\omega), \ldots, f(\zeta\omega^{n-1})) \end{cases}$$

and its inverse as follows

$$\mathsf{NTT}^{-1}_{n:\zeta:\omega} : \begin{cases} \prod_{i=0}^{n-1} R[X]/(X - \zeta\omega^i) \to R[X]/(X^n - \zeta^n) \\ (\hat{f}_0, \hat{f}_1, \ldots, \hat{f}_{n-1}) \mapsto \frac{1}{n} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (\zeta^{-1}\omega^{-i})^j \hat{f}_i \end{cases}.$$

The multiplication of polynomials $f, g \in R[X]/(X^n - \zeta^n)$ can be computed as follows (where $\circ$ is the element-wise multiplication)

$$f(x)g(x) = \mathsf{NTT}^{-1}_{n:\zeta:\omega}(\mathsf{NTT}_{n:\zeta:\omega}(f) \circ \mathsf{NTT}_{n:\zeta:\omega}(g)).$$

Let n be a power of two. By using Cooley-Tukey butterfly [22] $\mathsf{CT} : ((f_0, f_1), \omega) \mapsto (f_0 + \omega f_1, f_0 - \omega f_1)$ and Gentleman-Sande butterfly [23] $\mathsf{GS} : ((f_0, f_1), \omega) \mapsto (f_0 + f_1, (f_0 - f_1)\omega)$ recursively, NTT and inverse NTT can be computed efficiently. CT and GS are inverse transformations of each other except for the doubling factor; $\mathsf{GS}(\mathsf{CT}(f_0, f_1, \omega), \omega^{-1}) = 2(f_0, f_1)$.

NTT's calculation method using CT is shown in Algorithm 3, where $rev(i)$ is n-bit reverse of i.

Algorithm 3 Number Theoretic Transform

```
Input: $N = 2^n$, $f \in (\mathbb{Z}/p\mathbb{Z})[X]/(X^N - \zeta^N)$ and $N$-th primitive root of unity $\omega$
Output: $\hat{f} = \{f(\zeta\omega^{rev(i)})\}_{0 \leq i < N}$
1: For each $i$, $\hat{f}_i \leftarrow \zeta^i f_i$
2: $c \leftarrow 1$
3: for $i = 0$ to $n - 1$ do
4:     for $j = 0$ to $2^i - 1$ do
5:         for $k = 0$ to $2^{n-1-i} - 1$ do
6:             $s \leftarrow j2^{n-i} + k$
7:             $t \leftarrow j2^{n-i} + k + 2^{n-1-i}$
8:             $tmp \leftarrow \hat{f}_t\omega^{rev(c)} \bmod p$
9:             $\hat{f}_t = (\hat{f}_s - tmp) \bmod p$
10:            $\hat{f}_s = (\hat{f}_s + tmp) \bmod p$
11:            $c \leftarrow c + 1$
12:        end for
13:    end for
14: end for
```

D. NTT Multiplication for NTT-unfriendly Rings

Chung et al. [5] proposed a method to perform NTT-based multiplication for NTT-unfriendly rings. The lattice-based cryptographic scheme Saber [19] uses an NTT-unfriendly polynomial ring

$$R = (\mathbb{Z}/q\mathbb{Z})[X]/(X^n + 1),\ q = 2^{13},\ n = 256.$$

The reference implementation of Saber performs polynomial multiplication by Toom-Cook and Karatsuba methods. According to [5], performing Saber's polynomial multiplications by NTT-based multiplication improved the computation speed by about 20% on Cortex-M4.

Algorithm 4 NTT-based multiplication for Saber [5]

```
Input: f[256], g[256] for $f, g \in R := (\mathbb{Z}/q\mathbb{Z})[X]/(X^n+1)$,
    $q = 2^{13}$, $n = 256$ and prime $p = 25231359$
Output: h[256] for $h = fg \in R$
1: Compute $\mathsf{NTT}_{256}(\mathsf{f})$ and $\mathsf{NTT}_{256}(\mathsf{g})$ over $\mathbb{Z}/p\mathbb{Z}$
2: For each $i$, compute $\mathsf{h}[i] = \mathsf{f}[i] \cdot \mathsf{g}[i] \bmod^{\pm} p$
3: Compute $\mathsf{NTT}_{256}^{-1}(\mathsf{h})$
4: For each $i$, $\mathsf{h}[i] \leftarrow \mathsf{h}[i]$ & $(q-1)$
```

The following is a brief description of the method used by [5]. For integer polynomials f, g with coefficients in $[-q/2, q/2)$, the magnitude of each coefficient of the convolution $fg \bmod (X^n+1)$ is at most $nq^2/4$. Then, if we choose a prime p such that $p > nq^2/2$ and $2n \mid (p-1)$ are satisfied, then

$$fg \bmod^{\pm} (p, X^n+1) = fg \bmod (X^n+1) \tag{2}$$

holds, where $fg \bmod^{\pm} (p, X^n+1)$ denotes the convolution $fg \bmod (X^n+1)$ with all coefficients in $[-p/2, p/2)$. We can compute the left-hand side of (2) by NTT and recover the correct result $fg \bmod (q, X^n+1)$. The methods described above are shown in Algorithm 4.

III. Plantard Method

Plantard [10] proposed an improved version of Montgomery multiplication that efficiently computes n-bit modular multiplications when operating on a $2n$-bit word (Algorithm 5). This algorithm uses the fact that multiplication on a $2n$-bit word is equivalent to a modular multiplication modulo 2^{2n}. The major advantage of this algorithm is that the number of multiplications is less than Montgomery multiplication if $BR \bmod 2^{2n}$ can be precomputed, where NTT fits in this case. According to [10], NTT was about 10% faster by changing modular multiplications from Montgomery multiplication to Plantard's method.

Algorithm 5 Plantard Multiplication [10]

```
Input: $A, B, P, R, n$ with $0 \le A, B \le P$ and $R = P^{-1} \bmod$
    $2^{2n}$
Output: $C$ with $0 \le C < P$ and $C = AB(-2^{-2n}) \bmod P$
1: $C \leftarrow \left\lfloor \left( \left\lfloor ABR \bmod 2^{2n}/2^n \right\rfloor + 1 \right) P/2^n \right\rfloor$
2: if $C = P$ then
3:     return $C - P$
4: end if
5: return $C$
```

The following theorem holds with respect to the size of P.

Theorem 2 ([10, Theorem 1])**.** Let P be an odd modulo with $P < 2^n/\phi$ and $\phi = \frac{1+\sqrt{5}}{2}$, then Algorithm 5 is correct.

We check the key points of the proof of this theorem. We denote $Q := ABP^{-1} \bmod 2^{2n}$ and obtain

$$AB(-2^{-2n}) \bmod P = \frac{QP - AB}{2^{2n}}, \tag{3}$$

as did Montgomery multiplication. We further transform the right-hand of (3) and bring together all the ABs. Let $Q_1 := \lfloor Q/2^n \rfloor$ and $Q_0 := Q - Q_1 2^n = Q \bmod 2^n$. If $P < 2^n/\phi$, then

$$0 < \frac{P2^n - Q_0 P + AB}{2^{2n}} < 1. \tag{4}$$

Consequently, we obtain

$$\begin{aligned}
\frac{QP - AB}{2^{2n}} &= \left\lfloor \frac{QP - AB}{2^{2n}} + \frac{P2^n - Q_0 P + AB}{2^{2n}} \right\rfloor && (5) \\
&= \left\lfloor \frac{(Q - Q_0 + 2^n)P}{2^{2n}} \right\rfloor && (6) \\
&= \left\lfloor \frac{(Q_1 2^n + 2^n)P}{2^{2n}} \right\rfloor \\
&= \left\lfloor \frac{(Q_1 + 1)P}{2^n} \right\rfloor \\
&= \left\lfloor \frac{\left(\left\lfloor \frac{ABP^{-1} \bmod 2^{2n}}{2^n} \right\rfloor + 1 \right) P}{2^n} \right\rfloor .
\end{aligned}$$

This transformation of equation from the sum of products in Montgomery multiplication to the form of a product by an integer approximation function is the core idea of Plantard multiplication.

When we use Algorithm 5 to compute the modular multiplication in NTT (8th line of Algorithm 3), we can precompute $\omega^{rev(c)} p^{-1} \bmod 2^{2n}$. Therefore, to compute modular multiplication $\hat{f}_t \omega^{rev(c)} \bmod p$, Plantard multiplication requires two single-word size multiplications (MUL), one single-word size addition (ADD) and two shift operations (SHIFT). On the other hand, Montgomery multiplication without correction requires three MUL, one ADD, one SHIFT and one logical AND operation. Plantard method reduces the number of MUL operations by one. Plantard multiplication needs one more SHIFT operation than Montgomery multiplication, however, this does not affect efficiency as n-bit shifts on a $2n$-bit word are generally cheap.

IV. Proposed Method

Plantard's Algorithm 5 computes the modular Multiplications in NTTs more efficiently than Montgomery Multiplication. However, to use Algorithm 5 with Chung et al.'s method described in Section II-D, it is necessary to change the inputs and outputs from unsigned integers to signed integers. Unlike Montgomery multiplication (Algorithm 1 and 2), simply changing mod to $\bmod^{\pm}$ does not work well when the inputs are signed integers. This is because (4) does not necessarily hold if the inputs A, B are signed integers.

Another naïve way is the following approach. First, we convert input signed integers to unsigned integers. Then we process the first line of Algorithm 5. Finally, we convert from unsigned integers to signed integers just before output. It basically works, however the conversions require additional instructions. The conversions include operations to determine whether the given integer is positive or negative, which causes a branching by an **if** statement. This branching makes constant-time implementation difficult.

In this study, we propose Algorithm 6, dubbed Signed Plantard Multiplication, in which the inputs and outputs of Algorithm 5 are changed from unsigned integers to signed integers. This algorithm handles signed integers without branching while maintaining efficiency. Since there is no branching by **if** statements, a constant-time implementation of this algorithm is possible. The number of required multiplications is the same as in Algorithm 5, and the number of additions, shifts, and logical AND operations is almost the same.

Algorithm 6 Signed Plantard Multiplication

```
Input: $A, B, P, R, n$ with $|A|, |B| \leq 2^{n-1}$ and $R = P^{-1} \bmod^{\pm} 2^{2n}$
Output: $C = AB(-2^{-2n}) \bmod^{\pm} P$
1: $C \leftarrow \lfloor \left( \lfloor (ABR \bmod^{\pm} 2^{2n})/2^n \rceil \right) P/2^n \rceil$
2: return $C$
```

To make Plantard multiplication work well when the inputs are signed integers, we need to modify (4). The $(-Q_0P+AB)$ part of (4) plays an important role in the transformation (6). If we define Q_0 as $Q \bmod^{\pm} 2^{2n}$, $(-Q_0P + AB)$ is zero-centered. To take advantage of this property, we assume $-\frac{1}{2} \leq \frac{-Q_0P+AB}{2^{2n}} < \frac{1}{2}$ and use the round function $\lfloor \cdot \rceil$, not the floor function $\lfloor \cdot \rfloor$ at (5) (see the proof of Theorem 3 for details).

Theorem 3. Let P be an odd modulo with $P < 2^{n-1}$. Then Algorithm 6 is correct.

Proof of Theorem 3: We check the output of Algorithm 6 $C = AB(-2^{-2n}) \bmod^{\pm} P$, i.e.,

$$\left\lfloor \frac{\left\lfloor \frac{ABP^{-1} \bmod^{\pm} 2^{2n}}{2^n} \right\rceil \cdot P}{2^n} \right\rceil = AB(-2^{-2n}) \bmod^{\pm} P.$$

As P is odd, there exists a $Q := ABP^{-1} \bmod^{\pm} 2^{2n}$. Therefore, we have

$$\begin{aligned}(QP - AB) \bmod^{\pm} 2^{2n} &= (ABP^{-1}P - AB) \bmod^{\pm} 2^{2n} \\ &= 0.\end{aligned}$$

We observe that there exists an integer k satisfying

$$QP - AB = k2^{2n} \tag{7}$$

and therefore

$$\begin{aligned}(7) &\Rightarrow (-AB) \bmod^{\pm} P = k2^{2n} \bmod^{\pm} P \\ &\Rightarrow AB(-2^{-2n}) \bmod^{\pm} P = k \bmod^{\pm} P.\end{aligned}$$

Now, we analyze the size of k, knowing $|A|, |B| \leq 2^{n-1}$, then we get

$$\begin{aligned}|k| &= \left| \frac{QP - AB}{2^{2n}} \right| \\ &\leq \frac{|QP| + |AB|}{2^{2n}} \\ &\leq \frac{2^{2n-1} \cdot P + 2^{2n-2}}{2^{2n}} \\ &= \frac{P}{2} + \frac{1}{4}.\end{aligned}$$

Furthermore, since k is an integer and P is an odd integer, we have $-\frac{P-1}{2} \leq k \leq \frac{P-1}{2}$. Consequently, we obtain $k \bmod^{\pm} P = k$ and

$$AB(-2^{-2n}) \bmod^{\pm} P = k = \frac{QP - AB}{2^{2n}}.$$

Let $Q_0 := Q \bmod^{\pm} 2^n$ and $Q_1 := \lfloor Q/2^n \rceil = (Q - Q_0)/2^n$, and we assume

$$-\frac{1}{2} \leq \frac{AB - Q_0P}{2^{2n}} < \frac{1}{2}. \tag{8}$$

We show at the end of the proof that (8) holds. From the assumption (8) and the fact that $QP - AB$ is divisible by 2^{2n}, the following holds;

$$\begin{aligned}\frac{QP - AB}{2^{2n}} &= \left\lfloor \frac{QP - AB}{2^{2n}} + \frac{AB - Q_0P}{2^{2n}} \right\rceil \\ &= \left\lfloor \frac{(Q - Q_0)P}{2^{2n}} \right\rceil \\ &= \left\lfloor \frac{Q_1P}{2^n} \right\rceil.\end{aligned}$$

As $Q_1 = \lfloor (ABP^{-1} \bmod^{\pm} 2^{2n})/2^n \rceil$, we obtain that

$$\left\lfloor \frac{\left\lfloor \frac{ABP^{-1} \bmod^{\pm} 2^{2n}}{2^n} \right\rceil \cdot P}{2^n} \right\rceil = AB(-2^{-2n}) \bmod^{\pm} P.$$

To finish this proof, we show that assumption (8) holds if $P < 2^{n-1}$;

$$\begin{aligned}|AB - Q_0P| &\leq |AB| + |Q_0P| \\ &< 2^{n-1} \cdot 2^{n-1} + 2^{n-1} \cdot 2^{n-1} \\ &= 2^{2n-1}.\end{aligned}$$

■

By tightening the constraints on inputs A and B, the range of P can be increased.

Corollary 4. Let P be an odd modulo with $P < 2^n \left(\sqrt{3 - \frac{1}{2^{n-1}}} - 1 \right) + 1$. Then Algorithm 6 is correct for inputs A, B with $|A|, |B| \leq \frac{P-1}{2}$.

Proof of Corollary 4: We check that the assumption (8) in the proof is satisfied.

$$\begin{aligned}
|AB - Q_0P| &\leq |AB| + |Q_0P| \\
&\leq \left(\frac{P-1}{2}\right)^2 + 2^{n-1}P \\
&= \left(\frac{P-1}{2} + 2^{n-1}\right)^2 - 2^{2n-2} + 2^{n-1} \\
&< \left(2^{n-1}\left(\sqrt{3 - \frac{1}{2^{n-1}}} - 1\right) + 2^{n-1}\right)^2 \\
&\quad - 2^{2n-2} + 2^{n-1} \\
&= 2^{2n-2}\left(3 - \frac{1}{2^{n-1}}\right) - 2^{2n-2} + 2^{n-1} \\
&= 2^{2n-1}
\end{aligned}$$

■

Remark 5. In the proposed Algorithm 6, the output range is already $[-\frac{P-1}{2}, \frac{P-1}{2}]$, so there is no need to use **if** statements. We show that this is also true for Plantard's algorithm 5.

We check that in the following inequality

$$\left\lfloor \frac{\left(\left\lfloor \frac{ABP^{-1} \bmod 2^{2n}}{2^n} \right\rfloor + 1\right) P}{2^n} \right\rfloor \leq \frac{((2^n - 1) + 1)P}{2^n} = P,$$

the quality condition

$$\left\lfloor \frac{ABP^{-1} \bmod 2^{2n}}{2^n} \right\rfloor = 2^n - 1 \tag{9}$$

does not hold. Assuming that equality (9) holds, we derive a contradiction.

$$\begin{aligned}
(9) &\Rightarrow 2^n - 1 \leq \frac{ABP^{-1} \bmod 2^{2n}}{2^n} < 2^n \\
&\Rightarrow ABP^{-1} \bmod 2^{2n} = 2^{2n} - r \text{ for } {}^{\exists}r \in \mathbb{Z},\ 0 < r \leq 2^n \\
&\Rightarrow AB \bmod 2^{2n} = (2^{2n} - r)P \bmod 2^{2n} \\
&\Rightarrow AB \bmod 2^{2n} = (-rP) \bmod 2^{2n} \\
&\Rightarrow AB = 2^{2n} - rP \\
&\qquad (\because 0 \leq AB < 2^{2n},\ 0 < 2^{2n} - rP < 2^{2n}) \\
&\Rightarrow AB + rP = 2^{2n}.
\end{aligned} \tag{10}$$

Now we evaluate the size of $AB + rP$.

$$\begin{aligned}
|AB + rP| &\leq P^2 + 2^nP \\
&= (P + 2^{n-1})^2 - 2^{2n-2} \\
&< (2^n/\phi + 2^{n-1})^2 - 2^{2n-2} \\
&= 2^{2n-2}(2/\phi + 1)^2 - 2^{2n-2} \\
&= 2^{2n-2}(4/\phi^2 + 4/\phi) \\
&= 2^{2n}\frac{1+\phi}{\phi^2} \\
&= 2^{2n} \quad (\because \phi^2 - \phi - 1 = 0)
\end{aligned}$$

From the above, equation (10) does not hold, hence the contradiction is revealed.

Source code 1
SIGNED PLANTARD MULTIPLICATION

```
int64_t signedPlantardMul(int64_t A, int64_t B) {
   return (((A*B + 0x80000000)>>32)*P + 0x80000000)>>32;
}
```

Source code 2
SIGNED MONTGOMERY MULTIPLICATION

```
int signedMontgomeryMul(int A, int B) {
   int64_t t = (int64_t) A*B;
   return (t+(((t & 0xFFFFFFFF)*R) & 0xFFFFFFFF)*P)>>32;
}
```

V. EXPERIMENTAL RESULTS

In this section, we compare the performance of Signed Plantard Multiplication (Algorithm 6) with that of signed Montgomery multiplication (Algorithm 2). We also present a comparison between the NTT-based multiplication using signed Plantard multiplication and that using signed Montgomery multiplication.

A. C implementations

We implemented NTT-based multiplication for Saber [19] by Chung et al.'s method [5], i.e. size-256 NTT and inverse NTT over a polynomial ring $R' = (\mathbb{Z}/p\mathbb{Z})[X]/(X^{256} + 1)$, $p = 25231361 = 49280 \cdot 512 + 1$ (see Algorithm 4). In [5], Chung et al. implemented their NTT-based polynomial multiplication with 25-bit modulus on ARM 32-bit Microcontrollers. On a 64-bit processor, however, we can use signed Plantard multiplication instead of Montgomery multiplication for modular multiplication with 25-bit modulus.

The C implementations of signed Plantard multiplication and signed Montgomery multiplication for the NTT-based multiplication are shown in source code 1 and 2. The inputs A, B satisfy $-2^{32} \leq A, B < 2^{32}$. The parameters $P = 25231361$, $n = 32$ and $R = (-P^{-1}) \bmod^{\pm} 2^n = 25231359$ for signed Montgomery multiplication. In signed Plantard multiplication, the product $BR \bmod^{\pm} 2^{2n}$ for $R = P^{-1} \bmod^{\pm} 2^{2n}$ is precomputed and again denoted as B. The rounding $\lfloor X/2^n \rceil$ is calculated as $\lfloor (X + 2^{n-1})/2^n \rfloor$. The instructions used in both source codes and the number of times they are used are summarized in the TABLE I. Since our method used one less instruction each for MUL and the total number of instructions, the proposed method is considered to be more advantageous for platforms in particular, when multiplication takes multiple cycles.

B. Benchmarking for K210

We measured performance on the Sipeed Maixduino[2] development board based on the Kendryte SoC K210[3] (64-bit

[2]https://www.seeedstudio.com/Sipeed-Maixduino-Kit-for-RISC-V-AI-IoT-p-4047.html

[3]https://canaan.io/product/kendryteai

TABLE I
COMPARISON OF NUMBER OF INSTRUCTIONS BETWEEN OUR METHOD AND MONTGOMERY MULTIPLICATION.

Instruction	Our method (Alg. 6)	Montgomery (Alg. 2)
MUL ($\star$)	2	3
ADD (+)	2	1
SHIFT ($\gg$)	2	1
AND (&)	0	2
Total	6	7

RISC-V RV64GC 400 MHz). Maixduino is a popular RISC-V-based MCU and was used in the third-party evaluation of NIST Lightweight cryptography by Renner et al. [24][4]. We compiled our implementations with RISC-V GCC toolchain for Kendryte 210 toolchain-kendryte210[5] by PlatformIO with `-O3` options.

The portable C implementation of Saber is available at the official web page[6]. An almost identical implementation is provided by PQClean project[7], an OSS project that provides portable C implementations of the post-quantum schemes in the NIST PQC project. These implementations use Karatsuba and Toom-Cook multiplication for polynomial multiplications. Toom-Cook algorithm splits each of the two given polynomials of degree n into $d+1$ polynomials of degree $n/(d+1)$, evaluates the values at $2d+1$ different points, computes pointwise multiplication, performs polynomial interpolation, and finally reconstructs the product of the given polynomials. When $d=1$, this algorithm is called Karatsuba multiplication. In PQClean's Toom-Cook, $d=3$ and pointwise multiplications of degree 64 are calculated by Karatsuba multiplication. In Kendryte K210, we compare PQClean's Toom-Cook multiplication and our C implementations of NTT-based polynomial multiplication using Montgomery multiplication and signed Plantard multiplication, respectively.

TABLE II presents the measured CPU cycles for signed Plantard multiplication (Source code 1) and signed Montgomery multiplication (Source code 2). The values in the table are the minimum of 10,000 tests. In Kendryte K210, our method was about 23% faster than signed Montgomery multiplication. Since no assembly analysis was performed in this study and the K210 datasheet does not provide information on instruction cycles, an accurate analysis is difficult. However, after some experiments on instruction cycles, we presume the reduced number of MUL could be the main contribution of this gain. In this C implementation, the round function was implemented with a floor function, which increased the number of ADD. In some environments, it is possible that instructions corresponding to the round function could be used, and the execution cycles could be even smaller.

TABLE III shows the cycle results for PQClean's Toom-Cook multiplication vs. our NTT-based polynomial multiplication using Montgomery multiplication and signed Plantard multiplication, respectively. The values in the table are the minimum of 10,000 tests. Our method was about 16% faster when used within NTT-based multiplication. Changing the algorithm for calculating modular multiplications from signed Montgomery multiplication to signed Plantard multiplication improves the efficiency of computing of modular multiplication by a constant, especially in the butterfly operation in NTT (line 8 of Algorithm 3). As can be seen from TABLE I, the calculation of $\hat{f}_t \omega^{rev(c)} \bmod^{\pm} p$ with signed Plantard multiplication saves one MUL instruction. Since we need to perform the butterfly operations $n \log_2 n$ times in Algorithm 4, the impact of this gain on the overall efficiency is significant.

[4]https://lwc.las3.de/
[5]https://registry.platformio.org/tools/platformio/toolchain-kendryte210
[6]https://www.esat.kuleuven.be/cosic/pqcrypto/saber/
[7]https://github.com/PQClean/PQClean

TABLE II
COMPARISON BETWEEN OUR AND MONTGOMERY'S MODULAR MULTIPLICATION.

Algorithm	CPU cycle
Montgomery (Alg. 2)	38
Our method (Alg. 6)	31

TABLE III
COMPARISON OF PQCLEAN TOOM-COOK MULTIPLICATION AND NTT-BASED POLYNOMIAL MULTIPLICATION (NTT-MUL) WITH OUR METHOD AND MONTGOMERY MULTIPLICATION FOR SABER.

Algorithm	CPU cycle
PQClean Toom-Cook	184,919
NTT-Mul with Montgomery (Alg. 4 + Alg. 2)	115,050
NTT-Mul with our method (Alg. 4 + Alg. 6)	98,952

VI. CONCLUSION

In this work, we proposed an efficient method for computing the modular multiplication of signed integers when operating on a single word with reference to Plantard's method [10], which is originally defined over unsigned integers. The number of operations of our method is almost the same as that of [10], and the efficiency is considered to be maintained. In addition, since there is no branching in the algorithm, constant-time implementation is possible. This features would be lost if we apply a conventional conversion of Plantard method so that it works on signed integers. A promising application of our proposal is lattice-based cryptography using Number Theoretic Transform defined over signed integers.

To demonstrate the merit of our proposal, we implemented our method for NTT-based multiplication in Saber [19] by Chung et al.'s method [5] and achieved a speedup of about 16% on a RISC-V CPU K210 compared to using signed Montgomery multiplication.

We expect our method improves the performance of other lattice-based cryptography schemes that benefit from NTT. For example, CRYSTALS-Kyber [25], one of NIST PQC finalists, uses NTT-based polynomial multiplication with 12-bit modulus. Implementing Kyber on a 32-bit CPU with our proposal is interesting, especially considering that Kyber is now selected as one of the winners of NIST PQC. We can apply NTT with our algorithm to lattice-based schemes defined over NTT-unfriendly rings by Chung et al.'s technique as well. Such examples are NTRU [26], another NIST PQC finalist, and LAC [27], the award winner of the PQC competition hosted by Chinese Association for Cryptologic Research (also a NIST PQC round 2 candidate). Implementation of NTRU and LAC using our method is an interesting future work.

References

[1] A. Karatsuba, "Multiplication of multidigit numbers on automata," in *Soviet physics doklady*, vol. 7, 1963, pp. 595–596.

[2] A. L. Toom, "The complexity of a scheme of functional elements realizing the multiplication of integers," in *Soviet Mathematics Doklady*, vol. 3, no. 4, 1963, pp. 714–716.

[3] S. A. Cook and S. O. Aanderaa, "On the minimum computation time of functions," *Transactions of the American Mathematical Society*, vol. 142, pp. 291–314, 1969. [Online]. Available: http://www.jstor.org/stable/1995359

[4] R. Agarwal and C. Burrus, "Fast convolution using fermat number transforms with applications to digital filtering," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 22, no. 2, pp. 87–97, 1974.

[5] C.-M. M. Chung, V. Hwang, M. J. Kannwischer, G. Seiler, C.-J. Shih, and B.-Y. Yang, "NTT multiplication for NTT-unfriendly rings," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2021, no. 2, pp. 159–188, 2021, https://tches.iacr.org/index.php/TCHES/article/view/8791.

[6] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the Association for Computing Machinery*, vol. 21, no. 2, pp. 120–126, 1978.

[7] V. S. Miller, "Use of elliptic curves in cryptography," in *Advances in Cryptology – CRYPTO'85*, ser. Lecture Notes in Computer Science, H. C. Williams, Ed., vol. 218. Springer, Heidelberg, Aug. 1986, pp. 417–426.

[8] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.

[9] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor," in *Advances in Cryptology – CRYPTO'86*, ser. Lecture Notes in Computer Science, A. M. Odlyzko, Ed., vol. 263. Springer, Heidelberg, Aug. 1987, pp. 311–323.

[10] T. Plantard, "Efficient word size modular arithmetic," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1506–1518, 2021.

[11] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Advances in Cryptology – CRYPTO'96*, ser. Lecture Notes in Computer Science, N. Koblitz, Ed., vol. 1109. Springer, Heidelberg, Aug. 1996, pp. 104–113.

[12] Y. Sakai and K. Sakurai, "Timing attack against implementation of a parallel algorithm for modular exponentiation," in *ACNS 03: 1st International Conference on Applied Cryptography and Network Security*, ser. Lecture Notes in Computer Science, J. Zhou, M. Yung, and Y. Han, Eds., vol. 2846. Springer, Heidelberg, Oct. 2003, pp. 319–330.

[13] B. Bakhshi and B. Sadeghiyan, "A timing attack on Blakley's modular multiplication algorithm, and applications to DSA," in *ACNS 07: 5th International Conference on Applied Cryptography and Network Security*, ser. Lecture Notes in Computer Science, J. Katz and M. Yung, Eds., vol. 4521. Springer, Heidelberg, Jun. 2007, pp. 129–140.

[17] M. Tibouchi and A. Wallet, "One bit is all it takes: A devastating timing attack on BLISS's non-constant time sign flips," Cryptology ePrint Archive, Report 2019/898, 2019, https://eprint.iacr.org/2019/898.

[14] M. Vielhaber, "Reduce-by-feedback: Timing resistant and DPA-aware modular multiplication plus: How to break RSA by DPA," in *Cryptographic Hardware and Embedded Systems – CHES 2012*, ser. Lecture Notes in Computer Science, E. Prouff and P. Schaumont, Eds., vol. 7428. Springer, Heidelberg, Sep. 2012, pp. 463–475.

[15] J. Mittmann and W. Schindler, "Timing attacks and local timing attacks against barrett's modular multiplication algorithm," Cryptology ePrint Archive, Report 2020/946, 2020, https://eprint.iacr.org/2020/946.

[16] J. H. Silverman and W. Whyte, "Timing attacks on NTRUEncrypt via variation in the number of hash calls," in *Topics in Cryptology – CT-RSA 2007*, ser. Lecture Notes in Computer Science, M. Abe, Ed., vol. 4377. Springer, Heidelberg, Feb. 2007, pp. 208–224.

[18] Q. Guo, T. Johansson, and A. Nilsson, "A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM," in *Advances in Cryptology – CRYPTO 2020, Part II*, ser. Lecture Notes in Computer Science, D. Micciancio and T. Ristenpart, Eds., vol. 12171. Springer, Heidelberg, Aug. 2020, pp. 359–386.

[19] J.-P. D'Anvers, A. Karmakar, S. S. Roy, F. Vercauteren, J. M. B. Mera, M. V. Beirendonck, and A. Basso, "SABER," National Institute of Standards and Technology, Tech. Rep., 2020, available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[20] J.-P. D'Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren, "Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM," in *AFRICACRYPT 18: 10th International Conference on Cryptology in Africa*, ser. Lecture Notes in Computer Science, A. Joux, A. Nitaj, and T. Rachidi, Eds., vol. 10831. Springer, Heidelberg, May 2018, pp. 282–305.

[21] D. Moody, G. Alagic, D. Apon, D. Cooper, Q. Dang, J. Kelsey, Y.-K. Liu, C. Miller, R. Peralta, R. Perlner, A. Robinson, D. Smith-Tone, and J. Alperin-Sheriff, "Status report on the second round of the nist post-quantum cryptography standardization process," 2020-07-22 2020.

[22] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[23] W. M. Gentleman and G. Sande, "Fast fourier transforms: for fun and profit," in *Proceedings of the November 7-10, 1966, fall joint computer conference*, 1966, pp. 563–578.

[24] S. Renner, E. Pozzobon, and J. Mottok, "A hardware in the loop benchmark suite to evaluate NIST LWC ciphers on microcontrollers," in *ICICS 20: 22nd International Conference on Information and Communication Security*, ser. Lecture Notes in Computer Science, W. Meng, D. Gollmann, C. D. Jensen, and J. Zhou, Eds., vol. 11999. Springer, Heidelberg, Aug. 2020, pp. 495–509.

[25] P. Schwabe, R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, G. Seiler, and D. Stehlé, "CRYSTALS-KYBER," National Institute of Standards and Technology, Tech. Rep., 2020, available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[26] C. Chen, O. Danba, J. Hoffstein, A. Hulsing, J. Rijneveld, J. M. Schanck, P. Schwabe, W. Whyte, Z. Zhang, T. Saito, T. Yamakawa, and K. Xagawa, "NTRU," National Institute of Standards and Technology, Tech. Rep., 2020, available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[27] X. Lu, Y. Liu, D. Jia, H. Xue, J. He, Z. Zhang, Z. Liu, H. Yang, B. Li, and K. Wang, "LAC," National Institute of Standards and Technology, Tech. Rep., 2019, available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.