On Modular Algorithms and Butterfly Operations in Number Theoretic Transform

Yanze Yang[*], Yiran Jia[†], Guangwu Xu[‡]

Abstract

Number theoretic transform (NTT) has been a very useful tool in computations for number theory, algebra and cryptography. Its performance affects some post-quantum cryptosystems. In this paper, we discuss the butterfly operation of NTT. This basic module of NTT requires heavy modular arithmetics. Montgomery reduction is commonly used in this setting. Recently several variants of Montgomery algorithm have been proposed for the purpose of speeding up NTT. We observe that the Chinese remainder theorem (CRT) can be involved in this type of algorithms in nature and transparent ways. In this paper, a framework of using CRT to model Montgomery type algorithms is described. The derivation of these algorithms as well as their correctness are all treated in the CRT framework. Under our approach, some problems of a modular reduction algorithm (published in IACR Transactions on Cryptographic Hardware and Embedded Systems, doi:10.46586/tches.v2022.i4.614-636) are identified, and a counterexample is generated to show that the algorithm is incorrect.

Key words: Number theoretic transform, Butterfly operation, Modular algorithm, Chinese remainder theorem.

1 Introduction

The Fourier transform is a deep mathematical theory with a wide range of applications and reflects a perfect duality in mathematics. Its early motivation can be traced to solving wave equation and heat equation. We remark that the Fourier transform can be derived from the Chinese remainder theorem [13]. In fact, a finite Fourier transform is indeed a special case of the ring theoretic version of the Chinese remainder theorem, see [12]. In the modern information age, the finite Fourier transform is getting so important and becoming a basic mathematical tool for many fields. In theory, the Fourier transform mapping a problem from time domain to frequency domain, so a problem may be solved more easily in another framework. However, one needs to perform the Fourier transform back and forth between the frameworks and the computation of the transform itself is required to be efficient. This makes the fast computation of a Fourier transform a topic of great significance. The nice symmetry of the nth root of unity provides the possibility of such a fast calculation. In 1965, Cooley and Tukey [1] (re)discovered a general algorithm for fast Fourier transform (since Gauss had some idea of fast Fourier transform in his

[*] SCST, Shandong University, China, e-mail: yyannze99@mail.sdu.edu.cn.

[†] SCST, Shandong University, China, e-mail: kawyon@mail.sdu.edu.cn.

[‡] SCST, Shandong University, China, e-mail: gxu4sdq@sdu.edu.cn(Corresponding author).

unpublished note of 1805, as mentioned in the literature). For the parameter n, fast Fourier transform (FFT) takes time of $O(n \log n)$ compared to that of $O(n^2)$ by a naïve method. Fast Fourier transform is regarded as one of the ten most important algorithms on 20th century. Obviously, it is still an extremely important algorithm today.

In traditional fast Fourier transform, complex nth roots of unity involve in the computation. As they are mostly irrational numbers with no exact values can be stored in computer, one has to use a good approximations which is time consuming. The efforts for reducing such a side effect includes the Nussbaumer's trick[6], an idea of constructing a quotient ring of a polynomial ring, so the (coset of) indeterminant for the polynomial can be treated as a root of unity to reduce or eliminate the computation of multiplication. Another way of avoiding complex nth roots of unity is to perform Fourier transform over finite fields (or certain finite rings) using integer modular operation. In such a case, nth roots of unity are integers. This is what we now called the Number Theoretic Transform (NTT).

In many cryptographic applications, NTT has become a common operation so that it affects the performance of the corresponding crypto systems. As we shall see, a basic module of FFT (or NTT) is the butterfly operation. This module consists of integer addition, subtraction, multiplication, as well as modular operation for the case of NTT. It is noted that the NTT has been an important computation tool for many post-quantum algorithms and homomorphism encryptions, optimization of the butterfly operation and modular operation is getting greater recent attention.

Montgomery's seminal paper [4] of 1985 cleverly uses a power of 2 to the modular operation with odd modulus as multiplication, division and modular operation with a power of 2 take much less time. Recently, many variants of Montgomery's reduction algorithm has been proposed and applied in speed up the butterfly operation in NTT, for examples, [10, 7, 5].

In the literature, Montgomery reduction algorithm is regarded as a generalization of Hensel method for computing an inverse of 2-adic number.

We observe that, in the construction of Montgomery type algorithms there are two coprime moduli naturally involve in. One is a power of 2, the other is an odd modulus. What are processed in the algorithms are the remainders with respect to these two moduli. That means that the Chinese remainder theorem (CRT) might be involved in certain manner. In this paper, we establish a framework of using CRT to model Montgomery type algorithms, and to explain how these algorithms are derived as well as how their correctness are proved. Detailed treatments for Montgomery reduction algorithm [4] and signed-Montgomery reduction algorithm [10] are described. Under our approach, problems of the modular reduction algorithm in [5] are identified, and a counterexample is generated to show that the algorithm is incorrect.

The rest of the paper is arranged as follows. The preparation materials are given in sections 2. Our main results are presented in section 3. We conclude the paper in section 4.

2 Preparation and Literature Review

In this section, we introduce some related materials. Some discussions and commentations for certain topics are given. Some expressions and derivations presented in this section might be of independent interest.

2.1 Fast Fourier Transform

We shall give a brief introduction of FFT by setting the case to be $n = 2^m$. In our discussion of finite Fourier transform, an object to be transformed is usually a polynomial of degree less than n. If we identify an polynomial with its coefficients, the finite Fourier transform is also applies to n-dimensional vector. Among these two equivalent approaches, we choose to use polynomials. Given an nth primitive root ω, the Fourier transform of a polynomial f with $\deg(f) < n$ is the following vector

$$\widehat{f} = (f(1), f(\omega), f(\omega^2), \cdots, f(\omega^{n-1})).$$

An efficient way of computing $\widehat{f}$ is to use divide and conquer and to turn the operations on polynomials f with $\deg(f) < n$ to the operations on polynomials g with $\deg(g) < \frac{n}{2}$.

We list two ways of making such a reduction to polynomials of lower degrees.

Let $f(x) = \sum_{j=0}^{n-1} a_j x^j$. This polynomial can be rewritten as $f(x)f(x) = f_e(x) + x f_o(x)$ where

$$f_e(x) = \sum_{j=0}^{\frac{n}{2}-1} a_{2j} x^{2j}, f_o(x) = \sum_{j=0}^{\frac{n}{2}-1} a_{2j+1} x^{2j}.$$

If we set $y = x^2$, then f_o, f_e are polynomials of degree less than $\frac{n}{2} - 1$ in y. Thus the evaluations of f at $1, \omega, \cdots, \omega^{n-1}$ can be accomplished by evaluating polynomials with degree less than $\frac{n}{2} - 1$, so a reclusive divide and conquer procedure applies:

$$\begin{array}{ll}
f(1) = f_e(1) + 1 \cdot f_o(1), & f(\omega) = f_e(\omega^2) + \omega f_o(\omega^2), \\
f(\omega^2) = f_e(\omega^4) + \omega^2 \cdot f_o(1\omega^4), & f(\omega^3) = f_e(\omega^6) + \omega^3 f_o(\omega^6), \\
\cdots & \\
f(\omega^{n-2}) = f_e(\omega^{2(n-2)}) + \omega^{n-2} f_o(\omega^{2(n-2)}), & f(\omega^{n-1}) = f_e(\omega^{2(n-1)}) + \omega^{n-1} f_o(\omega^{2(n-1)}).
\end{array} \quad (1)$$

Now let us examine another decomposition of evaluation of $f(x) = \sum_{j=0}^{n-1} a_j x^j$. Let

$$f_p(x) = \sum_{j=0}^{\frac{n}{2}-1} (a_j + a_{j+\frac{n}{2}}) x^j, f_m(x) = \sum_{j=0}^{\frac{n}{2}-1} (a_j - a_{j+\frac{n}{2}}) \omega^j x^j,$$

then we can check that evaluation of f can be achieved by evaluation of two polynomials of degree less than

$$\begin{array}{ll} f(1) = f_p(1), & f(\omega) = f_m(1), \\ f(\omega^2) = f_p(\omega^2), & f(\omega^3) = f_m(\omega^2), \\ \cdots & \\ f(\omega^{n-2}) = f_p(\omega^{n-2}), & f(\omega^{n-1}) = f_m(\omega^{n-2}). \end{array} \tag{2}$$

Corresponding(1)and (2)we get two fast Fourier transforms respectively.

Algorithm 2.1 Fast Fourier Transform-(1)

```
Require: Positive integer$n = 2^m$nth roots of unity $\omega, \cdots, \omega^{n-1}$, polynomial $f$ with deg($f$) $< n$
Ensure: Vector $(f(1), f(\omega), \cdots, f(\omega^{n-1}))$.
 1: function FFT1($f, \omega, n$)
 2:     if $n = 1$ then
 3:         return $(f(1))$
 4:     end if
 5:     $(f_e(1), f_e(\omega^2), \cdots, f_e(\omega^{n-2})) \leftarrow$ FFT1$(f_e, \omega^2, \frac{n}{2})$
 6:     $(f_o(1), f_o(\omega^2), \cdots, f_o(\omega^{n-2})) \leftarrow$ FFT1$(f_o, \omega^2, \frac{n}{2})$
 7:     for $j = 0$ to $\frac{n}{2} - 1$ do
 8:         $f(\omega^j) \leftarrow f_e(\omega^{2j}) + \omega^j f_o(\omega^{2j})$
 9:         $f(\omega^{j+\frac{n}{2}}) \leftarrow f_e(\omega^{2j}) - \omega^j f_o(\omega^{2j})$
10:     end for
11:     return $(f(1), f(\omega), \cdots, f(\omega^{n-1}))$
12: end function
```

Algorithm 2.2 Fast Fourier Transform-(2)

```
Require: Positive integer$n = 2^m$nth roots of unity $\omega, \cdots, \omega^{n-1}$, polynomial $f$ with deg($f$) $< n$
Ensure: Vector $(f(1), f(\omega), \cdots, f(\omega^{n-1}))$.
function FFT2($f, \omega, n$)
    if $n = 1$ then
        return $(f(1))$
    end if
    $(f_p(1), f_p(\omega^2), \cdots, f_p(\omega^{n-2})) \leftarrow$ FFT2$(f_p, \omega^2, \frac{n}{2})$
    $(f_m(1), f_m(\omega^2), \cdots, f_m(\omega^{n-2})) \leftarrow$ FFT2$(f_m, \omega^2, \frac{n}{2})$
    return $(f_p(1), f_m(1), f_p(\omega^2), f_m(\omega^2), \cdots, f_p(\omega^{n-2}), f_m(\omega^{n-2}))$
end function
```

We remark that the operations in lines 8 and 9 of the Fast Fourier Transform-(1) are like $X + WY, X - WY$. While the functions f_p, f_m used in the Fast Fourier Transform-(2) need the basic operations $(a_j + a_{j+\frac{n}{2}})$ and $(a_j - a_{j+\frac{n}{2}})\omega^j$, they are like $X + Y, (X - Y)W$. These two structures are called butterfly structure. We shall pay attention to the latter.

2.2 Modular Operation

In this subsection, we describe several known algorithms for computing modular reduction that are used in NTT.

2.2.1 Classical Montgomery Reduction Algorithm

The Montgomery reduction algorithm is probably the most successful modular reduction methods that are useful in cryptography. It is based the Montgomery representation. Given an odd number $p > 1$, we select an $R = 2^n > p$. It is well known that $\pmod R$ is extremely straightforward and can be neglected.

The idea of using such convenient R to help computation of modulo p appeared in a paper of 1985 by Montgomery [4], where a clever method of representing elements in $\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$ is described. Under such a representation, arithmetic operations, especially multiplication, achieve greater efficiency. Let $a \in [0, p-1]$ be an integer, we call $\check{a} = aR \pmod p$ the Montgomery representation of a. For integer $T \in [0, Rp-1]$, its Montgomery reduction is the number $Redc(T) = TR^{-1} \pmod n$. The importance of the Montgomery reduction is that the reduction of the product of two Montgomery representations is still a Montgomery representation.

Let us first (pre-)compute $k = R - p^{-1} \pmod R$. The following algorithm as well as its proof of correctness is described in [4].

Algorithm 2.3 Montgomery Reduction

```
Require: Positive integers $T < p^2$
Ensure: $TR^{-1} \pmod p$.
function REDC($T$)
    $m \leftarrow (T \pmod R)k \pmod R$
    $t \leftarrow \frac{T+mp}{R}$
    if ($t > p$) then
        $t \leftarrow t - p$
    end if
    return $t$
end function
```

Division operation is eliminated in Montgomery algorithm. The Montgomery performs especially well for modulus being fixed. Montgomery algorithm and its variants are used in the butterfly operations we mentioned in the next subsection.

In processing an integer consisting of many words, Montgomery develops a modular multiplication method which alternating subtraction and multiplication. In this so called Montgomery alternating algorithm, R is taken to be $R = \beta^m$ with $\beta = 2^n$ being a word. The method replaces modulo and division by R with that by β, reduces number of registers used and is suitable for parallelization. see [4] for more details.

2.2.2 Signed-Montgomery Reduction

Recently, Seiler proposed a variation Montgomery reduction. The modulus is restricted in the size of a word and negative residue is allowed. The odd modulus p satisfies $2p < \beta$. The input is a product

WT of two Montgomery representations. We do not use $T < p^2$ to denote input because in the latter we shall consider butterfly operation and handle terms like $WX, W(X-Y)$. We shall also use signed modular operation $\pmod{^{\pm}N}$ to indicate that a residue should be in $[-\frac{N}{2}, \frac{N}{2})$. The description of the Seiler algorithm is next.

Algorithm 2.4 Signed-Montgomery Reduction

```
Require: Integer Product $-\frac{p\beta}{2} < WT = a_1\beta + a_0 < \frac{p\beta}{2},\ 0 \le a_0 < \beta$
Ensure: $r_1 = (WT)\beta^{-1} \pmod{p}$, $-p < r_1 < p$.
function SigRedc($WT$)
    $m \leftarrow a_0 p^{-1} \pmod{^{\pm}\beta}$
    $t \leftarrow \left\lfloor \frac{mp}{\beta} \right\rfloor$
    $t \leftarrow a_1 - t$
    return $t$
end function
```

2.2.3 Plantard Reduction Algorithm

In 2021, Plantard designed a new modular reduction algorithm [7]. The method utilizes the special property of doing modular operation with in a word size. It exhibits better performance for several cryptographic schemes. To be more precise, let $\phi = \frac{1+\sqrt{5}}{2}\beta = 2^n$ with n being the word size If the modulus$p$ satisfies $p < \frac{2^n}{\phi}$then the following algorithm works.

Algorithm 2.5 Plantard Reduction Algorithm

```
Require: $p < \frac{2^n}{\phi}, 0 \le W, T \le p \quad \mu = p^{-1} \pmod{2^{2n}}$
Ensure: $r \equiv WT(-2^{-2n}) \pmod{p}, 0 \le r < p$.
function PRedc($W, T, p, \mu$)
    $r \leftarrow \left\lfloor \frac{\left(\left\lfloor \frac{WT\mu \pmod{2^{2n}}}{2^n} \right\rfloor + 1\right)p}{2^n} \right\rfloor$
    if $(r = p)$ then
        return 0
    end if
    return $r$
end function
```

2.2.4 Signed Plantard Reduction Algorithm

To ease the situation in each level of NTT one has to do reduction on the coefficients of a polynomial and enlarge the allowed range of the input while reduce the range of output, Huang et al introduced an algorithm that processes signed integer [5]. In the algorithm, the odd modulus p satisfies $p < 2^{n-\alpha-1}$, where n is the size of a word and $\alpha \ge 0$ is an integer parameter. The description of the algorithm is as follows.

According to [5], this algorithm improves Plantard reduction algorithm, and is more reflexible.

Algorithm 2.6 Signed Plantard Reduction Algorithm

```
Require: $\alpha \geq 0, p < 2^{n-\alpha-1}, -p2^\alpha \leq W, T \leq p2^\alpha, \mu = p^{-1} \pmod{^\pm 2^{2n}}$
Ensure: $r \equiv WT(-2^{-2n}) \pmod{^\pm p}, -\frac{p}{2} < r < \frac{p}{2}$.
function SPRedc($W, T, p, \mu, \alpha$)
    $r \leftarrow \left\lfloor \frac{\left(\left\lfloor \frac{WT\mu \pmod{^\pm 2^{2n}}}{2^n} \right\rfloor + 2^\alpha\right)p}{2^n} \right\rfloor$
    return $r$
end function
```

However, we find that Algorithm 2.6 does not always produce the required result $WT(-2^{-2n})$ $\pmod{^\pm p}$ and hence is incorrect. In its proof of correctness (see [5], Theorem 1), the meaning of $\left\lfloor \frac{WT\mu \pmod{^\pm 2^{2n}}}{2^n} \right\rfloor$ is mistakenly modified. In our unified approach of Montgomery algorithm and its variants by using the Chinese remainder theorem in next section, differences between $WT(-2^{-2n})$ $\pmod{^\pm p}$ and $\left\lfloor \frac{\left(\left\lfloor \frac{WT\mu \pmod{^\pm 2^{2n}}}{2^n} \right\rfloor + 2^\alpha\right)p}{2^n} \right\rfloor$ shall be revealed and counter examples shall be generated. We also remark that the proof of Theorem 1 of [5] is erroneous in checking r to be in the right range, due to a misuse of floor and ceiling functions.

2.3 Butterfly Structure and Its Optimization in NTT

2.3.1 Butterfly Operation in NTL

The C++ library NTL developed by Shoup [11] provides powerful methods and tools for number theoretic and algebraic computations, also for cryptographic computations. In the butterfly operation for NTT, uses pre-estimation to calculate quotient to get efficiency. Let $\beta = 2^n$ with n being word size, the following method computes the map $(X, Y) \to (X + Y, W(X - Y))$.

2.3.2 Harvey Butterfly Operation

Recently in [3], Harvey discussed the NTL implementation of butterfly module and provides its proof of correctness. Harvey also designed an improved butterfly algorithm. Two strategies are used to in enhancing efficiency. The first one is to perform pre-estimation of quotient, and the second is to reduce the number of if clauses through lazy reduction. More precisely, the its improvement of modular reduction, signed Montgomery reduction that is similar to [10] is used. Combining the butterfly operation, $W' = \lfloor \frac{W\beta}{p} \rfloor$ is replaced by $W' = W\beta \bmod p$ which is the Montgomery representation of W. The optimized Algorithm 2.8 is

Algorithm 2.7 Butterfly Structure in NTL

```
Require: $p < \frac{\beta}{2}, 0 < W < p, W' = \lfloor \frac{W\beta}{p} \rfloor, 0 < W' < \beta, 0 \le X < p, 0 \le Y < p$
Ensure: $X' = X + Y \pmod{p}, 0 \le X' < p, Y' = W(X - Y) \pmod{p}, 0 \le Y' < p.$
function NTL-butterfly($X, Y, W, W', p$)
    $X' \leftarrow X + Y$
    if ($X' \ge p$) then
        $X' \leftarrow X' - p$
    end if
    $T \leftarrow X - Y$
    if ($T < 0$) then
        $T \leftarrow T + p$
    end if
    $Q \leftarrow \left\lfloor \frac{W'T}{\beta} \right\rfloor$
    $Y' \leftarrow (WT - Qp) \pmod{\beta}$
    if ($Y' \ge p$) then
        $Y' \leftarrow Y' - p$
    end if
    return $X', Y'$
end function
```

Algorithm 2.8 Harvey Butterfly Algorithm

```
Require: $p < \frac{\beta}{4}, 0 < W < p, W' = W\beta \pmod{p}, 0 < W' < p, \mu = p^{-1} \pmod{\beta}, 0 \le X < 2p, 0 \le Y < 2p$
Ensure: $X' = X + Y \pmod{p}, 0 \le X' < 2p, Y' = W(X - Y) \pmod{p}, 0 \le Y' < 2p.$
function Harvey-butterfly($X, Y, W', \mu, p$)
    $X' \leftarrow X + Y$
    if ($X' \ge 2p$) then
        $X' \leftarrow X' - 2p$
    end if
    $T \leftarrow X - Y + 2p$
    $R_1\beta + R_0 \leftarrow W'T$
    $Q \leftarrow \mu R_0 \pmod{\beta}$
    $H \leftarrow \lfloor \frac{Qp}{\beta} \rfloor$
    $Y' \leftarrow R_1 - H + p$
    return $X', Y'$
end function
```

2.3.3 Scott Butterfly Operation

In 2017, Scott[8] pointed out that in the current descriptions and improvements of NTT, side channel attacks were not considered. To deal with the threat of side channel attacks, Scott proposed an improved butterfly structure. The following algorithm of Scott uses lazy reduction and enlarges redundance, and extra reduction step is introduced at a suitable position.

3 Our Results

In this section, we described a unified framework to treat Montgomery reduction algorithm and its variants. The famous Montgomery algorithm is usually thought as a generalization of Hensel computation

Algorithm 2.9 Scott Butterfly Algorithm

```
Require: $p < \frac{\beta L}{4N}, 0 < W < p, \mu = -p^{-1} \pmod{\beta}, 0 \leq X < \frac{N}{L}p, 0 \leq Y < \frac{N}{L}p$
Ensure: $X' = X + Y \pmod{p}, 0 \leq X' < \frac{N}{L}p, Y' = W(X - Y) \pmod{p}, 0 \leq Y' < 2p$.
function Scott-butterfly($X, Y, W, \mu, p$)
    if ($m < L$ and $j < k + \frac{L}{2m}$) then
        $X \leftarrow X \pmod{p}$
        $Y \leftarrow Y \pmod{p}$
    end if
    $X' \leftarrow X + Y$
    $T \leftarrow X - Y + \frac{N}{L}p$
    $Q \leftarrow \mu(WT \pmod{\beta}) \pmod{\beta}$
    $Y' \leftarrow \frac{WT+Qp}{\beta}$
    return $X', Y'$
end function
```

of the inverse of a 2-adic number. We observed that the expressions of Montgomery reduction algorithm and a solution to the Chinese remainder theorem are identical in some sense, so they have a natural relation. Further more, the latter variants of Montgomery reduction algorithm also have Chinese remainder theorem interpretation. So the Chinese remainder theorem approach provides a unified, natural and transparent treatment to this family of algorithms.

The CRT is a well know method for solving a system of modular equation. It is also called Sun Tzu Theorem as it was described in a very ancient Chinese book "Sun Tzu Suan Jing". In his "Mathematical Treatise in Nine Sections" of 1247 [9], Qin described the Chinese remainder theorem with great detail and generality.

In [9], Qin discussed the concept of 'positive use', it can be summarized into the following equality (3) for the case of two moduli. We remark that although its proof is straightforward, the fact that Qin paid special attention to this expression in his derivation of CRT is very interesting.

Proposition 3.1. *Let $p, R > 1$ be two coprime integers. Denote $p^{-1} = p^{-1} \pmod{R}$, $R^{-1} = R^{-1} \pmod{p}$. Then*

$$p^{-1}p + R^{-1}R = 1 + pR. \tag{3}$$

Proof. By definition there exists positive integer ℓ such that

$$p^{-1}p = 1 + \ell R.$$

Since $1 \leq p^{-1} < R$, we see that $\ell < p$. Furthermore $\ell R \equiv -1 \pmod{p}$, so $p - \ell = R^{-1}$. Therefore,

$$p^{-1}p + R^{-1}R = p^{-1}p + (p - \ell)R = 1 + pR.$$

□

For the case of two moduli, we follow an approach in [2]. Using the solution x_0 of the system of modular equations $\left\{\begin{array}{l} x \equiv r_1 \pmod{p} \\ x \equiv r_2 \pmod{R} \end{array}\right.$ to multiply both sides of (3)we get $x_0 p^{-1} p + x_0 R^{-1} R = x_0 + x_0 pR$. Repalcing x_0 by r_2, r_1 respective on the left hand side, we see the formula of the Chinese remainder theorem:

$$r_2 p^{-1} p + r_1 R^{-1} R \equiv x_0 \pmod{pR}. \tag{4}$$

3.1 CRT Interpretation of Montgomery algorithm

Let us recall Montgomery reduction algorithm. Let odd number $p > 1$ and $R = 2^n > p$We need to compute $TR^{-1} \pmod{p}$ for positive integer $T < p^2$.

Denote $T_R = T \pmod{R}, T_p = T \pmod{p}$. The method of Montgomery reduction is: first precompute $k = R - p^{-1} \pmod{R}$; then compute $t = \frac{T + (T_R k \pmod{R})p}{R}$. The number t satisfies $t \equiv TR^{-1} \pmod{p}$ and is in $(0, 2p)$ thus t or $t - p$ is desired.

The following discussion indicates that a natural motivation of the Montgomery reduction algorithm is CRT.

Denote $T_R = T \pmod{R}, T_p = T \pmod{p}$, then the solution of

$$\left\{\begin{array}{ll} x \equiv T_R & \pmod{R}, \\ x \equiv T_p & \pmod{p} \end{array}\right.$$

is $X = T$.

Now T is given, T_R can be easily computed. For the present case, (4) becomes

$$T_R p^{-1} p + T_p R^{-1} R \equiv T \pmod{pR}.$$

We hope to get $TR^{-1} \pmod{p}$ through the above formula to investigate $T_p = T \pmod{p}$. A key observation is that $TR^{-1}R \pmod{p}$ is a term in the formula!

Now we use CRT to give another proof of the correctness of Algorithm 2.3.

Proposition 3.2. *Algorithm2.3 is correct.*

Proof. In order to get the same expression as in Montgomery reduction algorithm, we use (3). We get

$$Tp^{-1}p + TR^{-1}R = T + TpR.$$

Manipulating the terms above, and combining terms with Rp, then there is an integer a such that

$$\begin{aligned} TR^{-1}R &= T + Tkn = T + T_R kp + (T - T_R)kp \\ &= T + (T_R k \pmod{R})\, p + aRp \end{aligned}$$

From this, we see that $\big(T + (T_R k \pmod R)p\big)$ is divisible by R, so the result in Montgomery reduction algorithm is obtained.

$$TR^{-1} \equiv \frac{T + (T_R k \pmod R)\, p}{R} \mod p.$$

□

We shall not discuss the verification of $t < 2p$ as it is a routine checking. We would like to point out that the discussion above also suggests how the Montgomery reduction algorithm is derived.

3.2 CRT Interpretation of Signed Montgomery algorithm

Next we present the derivation of Algorithm 2.4, together with a proof of its correctness. In this algorithm, numbers proceeded are supposed to be within a word is of length n. Set $\beta = 2^n$ and assume the odd modulus $p < \frac{\beta}{2}$. We use A to denote the input WT, thus $A \in (-\frac{p\beta}{2}, \frac{p\beta}{2})$ and $A = a_1\beta + a_0$ with $0 \le a_0 < \beta$. We need to find an integer $r' \in (-p, p)$ such that $r' \equiv A\beta^{-1} \pmod p$.

From Proposition3.1, we have

$$\beta^{-1}\beta + p^{-1}p = 1 + \beta p.$$

Note that here β^{-1}, p^{-1} are all positive as in the setup by Qin [9]. So

$$A = A\beta^{-1}\beta + Ap^{-1}p - A\beta p$$

Note that in Algorithm 2.4, $m = a_0 p^{-1} \pmod{^{\pm}\beta}$, so

$$A = A\beta^{-1}\beta + Ap^{-1}p - A\beta p = A\beta^{-1}\beta + (a_0 + a_1\beta)p^{-1}p - A\beta p \equiv A\beta^{-1}\beta + mp \pmod{\beta p}$$

This means that $A - mp$ is divisible by β. Using $A = a_1\beta + a_0$, we have

$$\frac{a_0 + a_1\beta - mp}{\beta} \equiv A\beta^{-1} \pmod p.$$

As $\frac{a_0+a_1\beta-mp}{\beta} = a_1 - \lfloor\frac{mp}{\beta}\rfloor + \Big(\frac{a_0}{\beta} - \frac{mp}{\beta} + \lfloor\frac{mp}{\beta}\rfloor\Big)$ and $0 \le \frac{a_0}{\beta} < 1$, $-1 < -\frac{mp}{\beta} + \lfloor\frac{mp}{\beta}\rfloor \le 0$, we get

$$\begin{aligned} \tfrac{a_0+a_1\beta-mp}{\beta} &= a_1 - \lfloor\tfrac{mp}{\beta}\rfloor, \\ \tfrac{a_0}{\beta} - \tfrac{mp}{\beta} + \lfloor\tfrac{mp}{\beta}\rfloor &= 0. \end{aligned}$$

So

$$A\beta^{-1} \pmod p = a_1 - \lfloor\frac{mp}{\beta}\rfloor.$$

Letting $r' = a_1 - \lfloor\frac{mp}{\beta}\rfloor$, we need to check that $r' \in (-p, p)$. This is an easy argument since $r' = \frac{a_0+a_1\beta-mp}{\beta}$ and

$$\begin{aligned} -\tfrac{\beta p}{2} &\le mp < \tfrac{\beta p}{2}, \\ -\tfrac{\beta p}{2} &< A = a_0 + a_1\beta < \tfrac{\beta p}{2}. \end{aligned}$$

So we have proved

Proposition 3.3. *Algorithm2.4 is correct.*

It is remarked that the above discussion conveys some other information, such as $\frac{a_0}{\beta} - \frac{mp}{\beta}$ is an integer.

3.3 Discussion of Algorithm 2.6

This subsection is to discuss the correctness of Algorithm 2.6.

Some steps of proving the correctness of Plantard Reduction Algorithm can be inferred from here, so we will leave the treatment of Plantard Reduction Algorithm in the framework of the CRT.

As we indicated, the original proof of correctness of Algorithm 2.6 contains misuse of concepts. We shall perform an analysis using CRT to identify its problem and generate a counter example to show that it is incorrect.

In Algorithm 2.6, we set $R = 2^{2n}$. The odd modulus p satisfies $p < 2^{n-\alpha-1}$, where the integer parameter α satisfies $0 \le \alpha < n-1$. The number $\mu = p^{-1} \pmod{{}^{\pm}R}$ can be pre-computed.

With integer inputs $W, T \in [-p2^{\alpha}, p2^{\alpha}]$, the algorithm outputs $r \equiv WT(-R^{-1}) \pmod{{}^{\pm}p}, -\frac{p}{2} < r < \frac{p}{2}$. The number r computed by the algorithm is

$$r = \left\lfloor \frac{\left(\left\lfloor \frac{WT\mu \bmod {}^{\pm}2^{2n}}{2^n} \right\rfloor + 2^{\alpha}\right) p}{2^n} \right\rfloor .$$

Let $A = WT$, then $A \in [-2^{2\alpha}p^2, 2^{2\alpha}p^2]$. Multiply both sides of $p^{-1}p + R^{-1}R = 1 + pR$ by A, (here p^{-1} is $p^{-1} \pmod{R}$ a positive integer), we get a form of the CRT

$$Ap^{-1}p + AR^{-1}R = A + ApR.$$

Since $\mu = \begin{cases} p^{-1} & \text{if } p^{-1} < \frac{R}{2} \\ p^{-1} - R & \text{if } p^{-1} \ge \frac{R}{2} \end{cases}$, we have

$$A\mu p + AR^{-1}R = A + \delta ApR,$$

where $\delta = \begin{cases} 1 & \text{if } p^{-1} < \frac{R}{2} \\ 0 & \text{if } p^{-1} \ge \frac{R}{2} \end{cases}$. Choose $\ell \in \mathbb{Z}$ such that $A\mu - A\mu \pmod{{}^{\pm}R} = \ell R$, we see that

$$\big(A\mu \pmod{{}^{\pm}R}\big)p + AR^{-1}R = A + (\delta A - \ell)pR.$$

This indicates that $\big(A\mu \pmod{{}^{\pm}R}\big)p - A$ is divisible by R and

$$\frac{\big(A\mu \pmod{{}^{\pm}R}\big)p - A}{R} = A(-R^{-1}) + (\delta A - \ell)p.$$

Therefore,

$$A(-R^{-1}) \pmod{{}^{\pm}p} = \frac{\big(A\mu \pmod{{}^{\pm}R}\big)p - A}{R} \pmod{{}^{\pm}p}.$$

Denote $h = A\mu \pmod{^\pm R}, K = \frac{hp-A}{R}$. Since $-\frac{R}{2} \le h < \frac{R}{2}, -2^{2\alpha}p^2 \le A \le 2^{2\alpha}p^2 2^{2\alpha}p^2 < 2^{2n-2} = \frac{R}{4}$, we get $-\left(\frac{p}{2}+\frac{1}{4}\right) < K < \left(\frac{p}{2}+\frac{1}{4}\right)$. Notice that K is an integer and p is odd, we have the following

$$-\frac{p}{2} < K < \frac{p}{2}.$$

If Algorithm 2.6 were correct, the next equality would be true.

$$K = \left\lfloor \frac{\left(\left\lfloor \frac{h}{2^n} \right\rfloor + 2^\alpha\right)p}{2^n} \right\rfloor.$$

It would imply

$$\frac{\left(\left\lfloor \frac{h}{2^n} \right\rfloor + 2^\alpha\right)p}{2^n} - 1 < K = \frac{hp-A}{R} = \frac{\frac{h}{2^n}p - \frac{A}{2^n}}{2^n} \le \frac{\left(\left\lfloor \frac{h}{2^n} \right\rfloor + 2^\alpha\right)p}{2^n}.$$

However, the above relation does not always hold, we construct series of counterexamples based on this.

Here we present a simple example to show that Algorithm 2.6 is incorrect.

Example 3.1. *Let $n = 6, p = 31, \alpha = 0$. So $R = 2^{2n} = 4096$ and p satisfies $p < 2^{n-\alpha-1} = 32$.*

It is calculated that $\mu = p^{-1} \pmod{^\pm R} = -1057,\ R \pmod{^\pm p} = 8$. Now we take inputs $W = 19, T = -5$, then $A = -95$. So

$$A(-R^{-1}) \pmod{^\pm p} = 760 \pmod{^\pm 31} = -15.$$

On the other hand, $h = A\mu \pmod{^\pm R} = -1985$, we see that $\left\lfloor \frac{h}{2^n} \right\rfloor = \left\lfloor \frac{-1985}{64} \right\rfloor = -32$. This gives

$$\left\lfloor \frac{\left(\left\lfloor \frac{h}{2^n} \right\rfloor + 2^\alpha\right)p}{2^n} \right\rfloor = \left\lfloor \frac{(-32+1)31}{64} \right\rfloor = -16.$$

which is the output r by the algorithm, and is different from $A(-R^{-1}) \pmod{^\pm p}$.

4 Conclusion

In this paper, we setup a CRT framework to treat Montgomery reduction and its variants. Under this approach, the derivation of these algorithms can be revealed more transparently, their proof of correctness can be processed naturally. Using this approach, some problems of the modular reduction algorithm in [5] are identified, a counterexample is generated to show that the algorithm is incorrect.

References

[1] Cooley J. W., Tukey, J. W. An algorithm for the machine calculation of complex Fourier series Math. Comput., 19(1965), 297-301.

[2] G. Davida, B. Litow , and G. Xu, Fast arithmetics using Chinese Remaindering, *Information Processing Letters*, 109(2009), 660-662.

[3] Harvey D., Faster arithmetic for number-theoretic transforms, Journal of Symbolic Computation, 60(2014),113-119.

[4] P. L. Montgomery Modular multiplication without trial division Mathematics of Computation, 44(1985), 519C521.

[5] J. Huang, J. Zhang, H. Zhao et al., Improved Plantard arithmetic for lattice-based cryptography, IACR Transactions on Cryptographic Hardware and Embedded Systems, 2022, 2022(4): 614-636.

[6] H. J. Nussbaumer, Fast polynomial transform algorithms for digital convolution, IEEE Trans. Acoust. Speech Signal Process., 28 (1980), 205-215.

[7] T. Plantard, Efficient word size modular arithmetic, IEEE Transactions on Emerging Topics in Computing, 2021, 9(3): 1506-1518.

[8] M. Scott, A note on the implementation of the number theoretic transform. Cryptography and Coding: 16th IMA International Conference, 2017: 247-258.

[9] J. Qin, Mathematical Treatise in Nine Sections, 1247..

[10] G. Seiler, Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography, https://eprint.iacr.org/2018/039.

[11] V. Shoup, NTL: a library for doing number theory (Version5.5.2), http://www.shoup.net/ntl/.

[12] J. von zur Gathen, J. Gerhard, Modern Computer Algebra (3rd ed.), Cambridge Press, 2013

[13] G. Xu. A Remark on Fourier Transform, https://arxiv.org/abs/1807.05829, 2018