Montgomery Reduction Algorithm for Modular Multiplication Using Low-Weight Polynomial Form Integers

Jaewook Chung* and M. Anwar Hasan
j4chung@uwaterloo.ca
ahasan@secure.uwaterloo.ca

Department of Electrical and Computer Engineering
and
Centre for Applied Cryptographic Research
University of Waterloo
Ontario, Canada
August 3, 2006

Abstract. We extend low-weight polynomial form integers (LWPFIs) presented in [5]. An LWPFI p is an integer expressed as a degree-l, monic polynomial such that $p = t^l + f_{l-1}t^l + \cdots + f_1 t + f_0$, where t can be any positive integer. In [5], f_i's are limited to 0 and ± 1, but here we let $|f_i| \leq \xi$ for some small positive integer ξ. In modular multiplication based on LWPFI, elements in $\mathbb{Z}_p$ are expressed in polynomial in t and multiplication is performed in $\mathbb{Z}[t]/f(t)$. The coefficients must be reduced for subsequent modular multiplications. In [5], a coefficient reduction algorithm based on a division algorithm derived from the Barrett reduction algorithm is presented. In this report, we present a coefficient reduction algorithm based on the Montgomery reduction algorithm and its detailed analysis results. Bounds on the input and output of our coefficient reduction algorithm is carefully analyzed. We give conditions for eliminating the final subtractions at the end of the Montgomery reduction algorithm. In addition, we present efficient modular addition and subtraction methods using LWPFI moduli.

1 Introduction

In [5], LWPFIs are defined as integers that can be represented in a degree-l monic polynomial form, $f(t) = t^l + f_{l-1}t^{l-1} + \cdots + f_1 t + f_0$, where $|f_i| \leq 1$. Modular multiplication using an LWPFI is performed in two phases. First, integers represented in polynomial form are multiplied in the polynomial ring $\mathbb{Z}[t]/f(t)$. Then the coefficients of the resulting polynomial are reduced. In [5], the coefficient reduction algorithms are based on a division algorithm derived from the Barrett reduction algorithm. In this report, we present a new coefficient reduction algorithm based on the Montgomery reduction algorithm. Moreover, we further generalize LWPFIs by removing the restriction on f_i's. We analyze the performance of the new coefficient reduction algorithm using this general framework.

First, we present the definition of our extended LWPFIs in Section 2. Then, in Section 3, we present our new coefficient reduction method based on the Montgomery reduction algorithm. We give detailed analysis and consider some interesting special cases. In Section 4, we show conditions for parameters for which our new coefficient reduction algorithm can be performed without requiring any final subtractions. In Section 5, we show methods for performing additions and subtractions modulo an LWPFI. In Section 6, we discuss how the performance of our coefficient reduction algorithm is

* This research was funded by Bell Canada Enterprises through the Bell 125th Anniversary Scholarship.

compared to that of the Montgomery reduction algorithm. In Section 7, we consider the applications of LWPFIs. In Section 8, we consider applying our coefficient reduction method to modular number systems [3,4].

2 Low-Weight Polynomial Form Integers Redefined

In [5], LWPFIs are defined as integers expressed in low-weight, monic polynomial form: $p = f(t) = t^l + f_{l-1}t^{l-1} + \cdots + f_1 t + f_0$, where $l \geq 2$, $f_i \in \{0, \pm 1\}$ and $t > 2(2^{2l+1} - 1)(2^l - 1)$.

Here we loosen the restriction on f_i's so that $|f_i| \leq \xi$ for some small positive integer $\xi < t$. The condition $t > 2(2^{2l+1}-1)(2^l-1) \approx 2^{3l+2}$ is applied in [5] due to the use of coefficient reduction based on a division algorithm. However, such a condition is not needed in our improved coefficient reduction presented here. In this report, we work in this general framework and narrow down conditions on parameters that allow efficient implementation of modular arithmetic modulo an LWPFI.

Definition 1 (LWPFI Redefined). *For a degree-l, monic polynomial $f(t) = t^l + f_{l-1}t^{l-1} + \cdots + f_1 t + f_0$, where t is a positive integer and $|f_i| \leq \xi$ for some small positive integer $\xi < t$, $p = f(t)$ is a* **low-weight polynomial form integer**.

In modular arithmetic based on LWPFI moduli, we express elements of $\mathbb{Z}_p$ as polynomials in $\mathbb{Z}[t]/f(t)$. Such a representation always exists for any element in $\mathbb{Z}_p$ using coefficients at most $(t+\xi)/2$ in magnitude.

Proposition 1. *For any integer $x \in \mathbb{Z}_p$, there exists a degree-$(l-1)$ polynomial $x(t) = \sum_{i=0}^{l-1} x_i t^i$ such that $x \equiv x(t) \pmod{p}$ and $|x_i| \leq \psi$, if $\psi \geq (t+\xi)/2$.*

Proof. Let $p_{\max} = t^l + \xi t^{l-1} + \cdots + \xi t + \xi$. Then $p_{\max}$ is the maximum possible LWPFI of the form $f(t) = t^l + \sum_{i=0}^{l-1} f_i t^i$, where $|f_i| \leq \xi$. Let $x(t) = \sum_{i=0}^{l-1} x_i t^i$. If $\max(x(t)) - \min(x(t)) \geq p_{\max}$ holds, then $x(t)$ can represent any element in $\mathbb{Z}_{f(t)}$. It is straightforward that

$$\max(x(t)) = \sum_{i=0}^{l-1} \psi t^i = -\min(x(t)). \tag{1}$$

It follows that

$$\begin{aligned} &\max(x(t)) - \min(x(t)) \geq p_{\max} \\ \iff\ &(2\psi - \xi) \cdot \frac{t^l - 1}{t - 1} \geq t^l. \end{aligned} \tag{2}$$

It is easy to see that $2\psi - \xi = t - 1$ does not satisfy the above inequality, but $2\psi - \xi \geq t$ does. Therefore $\psi \geq (t + \xi)/2$.

We let $\psi_{\min} = (t + \xi)/2$. However, in practice, the magnitudes of the coefficients do not have to be limited to ψ_{min}. To find a polynomial that corresponds to a given integer, Algorithm 1 can be used. The resulting polynomial has coefficients that are at most $(t/2+\xi)$ in magnitude. Since $t/2+\xi > \psi_{\min}$, Algorithm 1 results in a slightly redundant representation.

Algorithm 1 Conversion to Polynomial Form

Require: an integer $0 \leq x < p$, where $p = f(t) = t^l + f_{l-1}t^{l-1} + \cdots + f_1 t + f_0$.
Ensure: a polynomial $x(t) = \sum_{i=0}^{l-1} x_i t^i$, such that $x \equiv x(t) \pmod{p}$, where $|x_i| \leq t/2 + \xi$.
1: $c_{-1} \leftarrow x$.
2: **for** i from 0 to $l-1$ **do**
3: Find c_i and x_i such that $c_{i-1} = c_i t + x_i$, where $-t/2 \leq x_i < t/2$.
4: **end for**
5: **for** i from 0 to $l-1$ **do**
6: $x_i \leftarrow x_i - f_i \cdot c_{l-1}$. (Note: $|c_{l-1}| \leq 1$)
7: **end for**
8: **return** $x(t) = \sum_{i=0}^{l-1} x_i t^i$.

3 Modular Multiplication Using LWPFI moduli

In this section, we present an efficient modular multiplication scheme using LWPFI moduli. The modular multiplication using LWPFI moduli is performed in the following steps.

1. POLY-MULT: $\hat{z}(t) = x(t) \cdot y(t)$.
2. POLY-REDC: $z'(t) = \hat{z}(t) \bmod f(t)$.
3. COEFF-REDC: coefficient reduction of $z'(t)$.

The above modular multiplication scheme is called the *LWPFI modular multiplication*. POLY-MULT step can be performed by at most l^2 multiplications of coefficients using the schoolbook method. Sub-quadratic multiplication algorithms may be applied to achieve better performance [10,15,6,13]. POLY-REDC step requires at most $(l-1)\tau$ constant multiplications by integers at most ξ in magnitude, where τ is the number of non-zero f_i's. The range of f_i we use here is larger than that in [5]. Note that, due to this extended range for f_i's, our POLY-REDC step is potentially slower than that in [5]. However, we will not go over the details on POLY-REDC and focus only on the establishment of a new coefficient reduction algorithm based on the Montgomery reduction algorithm. For fixed $f(t)$, one may consider combining POLY-MULT and POLY-REDC steps for better performance as we propose in [5].

Suppose that the coefficients of $x(t)$ and $y(t)$ are at most ψ in magnitude. It easily follows that the result of POLY-REDC has coefficients that are at most $\psi^2((\xi+1)^l - 1)/\xi$ in magnitude as shown in Proposition 2. Throughout this report, we will use λ to denote $((\xi+1)^l - 1)/\xi$.

Proposition 2. $|z'_i| \leq \lambda\psi^2$.

Proof. Let $x(t)$ and $y(t)$ be the polynomials whose coefficients are at most ψ in magnitude. Let $\hat{z}(t) = (\hat{z}_{2l-2}, \ldots, \hat{z}_1, \hat{z}_0) = x(t) \cdot y(t)$. It follows that $|\hat{z}_i| \leq (i+1)\psi^2$ for $i = 0, \ldots, l-1$ and $|\hat{z}_i| \leq (2l-1-i)\psi^2$ for $i = l, \ldots, 2l-2$. The magnitudes of coefficients in $z'(t) = \hat{z}(t) \bmod f(t)$ are maximum when $f(t) = t^l \pm \xi\sum_{i=0}^{l-1} t^i$. In both cases, $\max(|z'_i|) = ((\xi+1)^{l-1} + (\xi+1)^{l-2} + \cdots + 1) \cdot \psi^2$. Therefore $|z'_i| \leq ((\xi+1)^l - 1)/\xi \cdot \psi^2$.

In Section 4, we discuss how the value ψ is related to other parameters, t, ξ and l. In [5], $\psi = t + 2^{l+2} - 2$ is fixed and a division algorithm derived from the Barrett reduction algorithm is used to perform COEFF-REDC step. In this work, we apply the Montgomery reduction algorithm to perform COEFF-REDC step and determine appropriate value ψ.

Algorithm 2 Montgomery Algorithm for Integers Reduction (MAIR)

Require: integers T and $m = (m_{k-1} \cdots m_1 m_0)_b$, such that $R = b^q$, $0 \leq T < mR$ and $\gcd(b, m) = 1$.
Ensure: $T \cdot b^{-q} \bmod m$.
1: $T_0 \leftarrow T$.
2: **for** i from 0 to $q - 1$ **do**
3: $\quad u_i \leftarrow -m^{-1} \cdot T_i \bmod b$.
4: $\quad T_{i+1} \leftarrow (T_i + u_i \cdot m)/b$.
5: **end for**
6: **if** $T_q \geq m$ **then**
7: $\quad T_q \leftarrow T_q - m$.
8: **end if**
9: **return** T_q.

Note that the output of our COEFF-REDC based on the Montgomery reduction algorithm (MONT-COEFF-REDC) is different from the output from Algorithm 5 in [5]. In [5], Algorithm 5 computes $z(t)$ such that $z(t) \equiv x(t) \cdot y(t) \pmod{p}$. However, the MONT-COEFF-REDC presented here outputs $z(t) \equiv x(t) \cdot y(t) \cdot b^{-q} \pmod{p}$, where b is the radix used to represent coefficients of polynomials in $\mathbb{Z}[t]/f(t)$ and q is a positive integer. Consider two integers $\bar{x}(t) \equiv x(t) \cdot b^q \pmod{p}$ and $\bar{y}(t) \equiv y(t) \cdot b^q \pmod{p}$. These are the transformation of $x(t)$ and $y(t)$ to the so-called *the Montgomery domain*. The direct product of $\bar{x}(t)$ and $\bar{y}(t)$ in $\mathbb{Z}[t]/f(t)$ results in $\bar{x}(t)\bar{y}(t) \equiv x(t)y(t) \cdot b^{2q} \pmod{p}$. Applying our new coefficient reduction algorithm results in $\bar{z}(t) \equiv x(t)y(t) \cdot b^q \pmod{p}$, whose coefficients are at most ψ. Note that the result is the transformation of $x(t)y(t)$ to the Montgomery domain. We discuss the relationship between the value q and other parameters of LWPFI in Section 4.

3.1 Montgomery Reduction Algorithm

The Montgomery algorithm performs modular reduction without using any division instruction of the underlying processor [12]. Let m be a modulus, and T be a positive integer which is to be reduced. We choose an integer R such that $R > m$, $\gcd(m, R) = 1$ and $0 \leq T < mR$.

Algorithm 2 computes $T \cdot b^{-q} \bmod m$, given an integer $0 \leq T < mR$, where $R = b^q$. In each iteration of Algorithm 2, a multiple of the modulus M is added to T_i such that the least significant digit becomes zero. Then, the division of T_{i+1} by b can be performed simply by shifting all digits of T by one place to the right. If q is chosen to be the digit length of T, then it can be easily shown that $T_q \in [0, 2m)$. Therefore, one final subtraction by m may be required to output an integer within $[0, m)$. Some researchers have proposed ways to eliminate this final subtraction to avoid timing attacks [11,14,18]. Walter proposed using q such that $2m < b^{q-1}$ [16]. Hachez and Quisquater improved this condition to $m < b^{q-1}$ for $b = 2$ [9]. Walter improved this condition again to $4m < b^q$ [17]. Line 3 of MAIR requires one single-precision multiplication and line 4 requires k single-precision multiplications. Therefore, MAIR requires a total of $q(k + 1)$ single-precision multiplications.

3.2 COEFF-REDC based on Montgomery Reduction Algorithm

Here, we construct a new coefficient reduction algorithm which is similar to Algorithm 2. Given an input polynomial $z'(t)$ of degree $(l - 1)$, our new algorithm computes a polynomial whose evaluation at t is congruent to $z'(t) \cdot b^{-q} \bmod p$.

Before, we begin the description of a new coefficient reduction algorithm, we clarify notations that we use in this report. Let $\boldsymbol{u}$ and $\boldsymbol{v}$ be the column vectors in $\mathbb{Z}^l$ such that the following condition is satisfied:

$$[t^{l-1}, \ldots, t, 1] \cdot \boldsymbol{u} \equiv [t^{l-1}, \ldots, t, 1] \cdot \boldsymbol{v} \pmod{p}. \tag{3}$$

Then we say $\boldsymbol{u}$ *is congruent to* $\boldsymbol{v}$ *modulo* p and write as $\boldsymbol{u} \cong_p \boldsymbol{v}$. We slightly abuse this notation and write as $\boldsymbol{u} \cong_b v$ for some integer v satisfying $[t^{l-1}, \ldots, t, 1] \cdot \boldsymbol{u} \equiv v \pmod{b}$. We also say $\boldsymbol{u}$ *is congruent to* v *modulo* b, if $\boldsymbol{u} \cong_b v$. We use '$\equiv$', to express element-wise congruence relation, i.e., $\boldsymbol{u} \equiv \boldsymbol{v} \pmod{b}$. In "$\boldsymbol{u} \bmod b$", modulo operation applies to each element of $\boldsymbol{u}$.

Let $x(t) = (x_{l-1}, \ldots, x_1, x_0)_t$ be the result of POLY-REDC step and b be the radix used for representing x_i's. When performing multiplication in $GF(p)[t]/f(t)$, we can apply Algorithm 2 individually to each coefficient to reduce them modulo p. However, individual reduction of coefficients is not possible with arithmetic in $\mathbb{Z}[t]/f(t)$. To reduce coefficients in $\mathbb{Z}[t]/f(t)$, we must apply the Montgomery reduction algorithm to all coefficients simultaneously.

The coefficient reduction is closely related to the *closest vector problem* from lattice theory. A lattice $\mathcal{L}$ is a discrete subgroup of $\mathbb{R}^l$. Let $\boldsymbol{V} = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{d-1}, \boldsymbol{v}_d\}$ be a set of linearly independent vectors in $\mathbb{R}^l$. The lattice $\mathcal{L} = \mathcal{L}(\boldsymbol{V})$ is a set of all integral combination of $\boldsymbol{v_i}$'s. The set $\boldsymbol{V}$ is called the basis of the lattice $\mathcal{L}(\boldsymbol{V})$. If $d = l$, $\mathcal{L}$ is called a full-rank lattice. If $\boldsymbol{v}_i \in \mathbb{Z}^l$ for all i, then $\mathcal{L}$ is called an integral lattice. For our purpose, we assume that $\mathcal{L}$ is a full-rank, integral lattice.

Suppose $\boldsymbol{v}_i \cong_p 0 \pmod{p}$ for all $i = 1, \ldots, l$. Then all the lattice points in $\mathcal{L}$ represent 0 modulo p. Let $\boldsymbol{x}$ be a vector whose elements are the coefficients of $x(t)$. Suppose $\boldsymbol{y} \in \mathcal{L}(\boldsymbol{V})$ is the closest lattice point (with respect to L_∞ norm) to $\boldsymbol{x}$, then $\boldsymbol{z} = \boldsymbol{x} - \boldsymbol{y}$ belongs to the fundamental domain of $\mathcal{L}$. The coordinate values of $\boldsymbol{z}$ forms a polynomial $z(t)$ such that $z(t) \equiv x(t) \pmod{p}$ and it has only reasonably small coefficients. However, closest vector problem is believed to be an NP-hard problem. There are polynomial time algorithms that give approximate solutions [1], but they require arithmetic using floating point or rational numbers and are too cumbersome to use for our purposes.

Rather than solving the closest vector problem, we search for $\boldsymbol{z'}$ such that $\boldsymbol{x} \cong_p \boldsymbol{z'} \cdot b^q \pmod{p}$ and the elements of $\boldsymbol{z'}$ are reasonably small. Below we show how to find such a vector $\boldsymbol{z'}$ using a method similar to the Montgomery reduction algorithm. This approach requires only simple integer arithmetic and enjoys good features of the Montgomery reduction algorithm for integers.

Algorithm 3 shows our Montgomery reduction algorithm adapted to perform COEFF-REDC step. Note that we have used $\boldsymbol{x}_q^{(i)}$ to denote the element of $\boldsymbol{x}_q$ at the i-th row in Algorithm 3. Moreover, F is an $l \times l$ integral matrix such that the following holds for any column vectors $\boldsymbol{x}$ and $\boldsymbol{u} \in \mathbb{Z}^l$:

$$\boldsymbol{x} + F \cdot \boldsymbol{u} \cong_p \boldsymbol{x}. \tag{4}$$

A non-trivial matrix F that satisfies (4) can be constructed by collecting l column vectors that are congruent to 0 modulo p. Such a matrix F must be invertible modulo b, since we need $F' = -F^{-1} \bmod b$ in line 3 of Algorithm 3. The invertibility of F modulo b can be verified by checking if $\det F \neq 0$ and the determinant has no common factor with b, i.e., $\gcd(\det F, b) = 1$.

Theorem 1. *Algorithm 3 returns* $z(t) \equiv x(t) \cdot b^{-q} \pmod{p}$.

Proof. It is easily seen that each iteration of Algorithm 3 computes the following:

$$\boldsymbol{x}_{i+1} \leftarrow \frac{\boldsymbol{x}_i + F \cdot (-F^{-1} \cdot \boldsymbol{x}_i \bmod b)}{b}. \tag{5}$$

Algorithm 3 MONT-COEFF-REDC

Require: $x(t) = (x_{l-1}, \ldots, x_1, x_0)_t$, a matrix F and $F' = -F^{-1} \bmod b$, where $\det F \neq 0$ and $\gcd(\det F, b) = 1$.
Ensure: $z(t) \equiv x(t) \cdot b^{-q} \pmod{p}$.
1: $\boldsymbol{x_0} \leftarrow [x_{l-1}, x_{l-2}, \ldots, x_0]^T$.
2: **for** i from 0 to $q-1$ **do**
3: $\quad \boldsymbol{u_i} \leftarrow F' \cdot \boldsymbol{x_i} \bmod b$.
4: $\quad \boldsymbol{x_{i+1}} \leftarrow (\boldsymbol{x_i} + F \cdot \boldsymbol{u_i})/b$.
5: **end for**
6: Perform final subtractions if necessary.
7: **return** $z(t) = \sum_{i=0}^{l-1} z_i t^i$, where $z_i = \boldsymbol{x_q}^{(i)}$.

Since F is a collection of column vectors that are congruent to 0 modulo p, adding any integral linear combination of the column vectors in F to $\boldsymbol{x}_i$ does not change its value in $\mathbb{Z}_p$. Hence, $\boldsymbol{x}_{i+1} \cong_p (\boldsymbol{x}_i + F \cdot (-F^{-1} \cdot \boldsymbol{x}_i \bmod b)) \cdot b^{-1}$. The division by b in (5) is exact and requires no division, since

$$\begin{aligned} \boldsymbol{x} + F \cdot \boldsymbol{u} &= \boldsymbol{x} + F \cdot (-F^{-1} \cdot \boldsymbol{x} \bmod b) \\ &\equiv [0, \ldots, 0, 0]^T \pmod{b}. \end{aligned} \tag{6}$$

Therefore, $\boldsymbol{x}_{i+1} \cong_p \boldsymbol{x}_i \cdot b^{-1}$. In Algorithm 3, the process (5) is performed iteratively q times starting with $\boldsymbol{x}_0 = \boldsymbol{x}$ resulting in $\boldsymbol{x}_q \equiv \boldsymbol{x} \cdot b^{-q} \pmod{p}$. This is quite similar to the original Montgomery reduction algorithm. The only difference is that Algorithm 3 uses vectors and matrix, while the original Montgomery reduction algorithm deals with integers.

At this point, a number of questions arise: what are the conditions for q such that $\boldsymbol{x}_q$ are sufficiently reduced, so that the result can be used as input to the subsequent LWPFI modular multiplications? How do we construct the matrix F? Is Algorithm 3 efficient? We answer these questions in the following.

3.3 Construction of F and Analysis of Algorithm 3

For $p = f(t) = t^l + f_{l-1}t^{l-1} + \cdots f_1 t + f_0$, where $|f_i| \leq \xi$, consider the following $l \times l$ matrix F:

$$F = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & -t - f_{l-1} \\ -t & 1 & \cdots & 0 & 0 & -f_{l-2} \\ 0 & -t & \cdots & 0 & 0 & -f_{l-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & -t & 1 & -f_1 \\ 0 & 0 & \cdots & 0 & -t & -f_0 \end{bmatrix}. \tag{7}$$

We have constructed the matrix F such that the column vectors of F are congruent to 0 modulo p, i.e., $F \cong_p [0, \ldots, 0, 0]$. It remains to verify whether F has its inverse modulo b. The invertibility of F modulo b can be easily checked as shown in Proposition 3.

Proposition 3. *The $l \times l$ matrix F as shown in* (7) *is invertible modulo b if and only if $\gcd(p = f(t), b) = 1$ and $f(t) \neq 0$.*

Proof. We perform some elementary row operations on both sides of $I_l \cdot F = F$, where I_l is an $l \times l$ identity matrix, to obtain

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0\,0 \\ t & 1 & \cdots & 0 & 0\,0 \\ t^2 & t & \cdots & 0 & 0\,0 \\ \vdots & \vdots & \vdots & \ddots & \vdots\,\vdots \\ t^{l-2} & t^{l-3} & \cdots & t & 1\,0 \\ t^{l-1} & t^{l-2} & \cdots & t^2 & t\,1 \end{bmatrix} \cdot F = \begin{bmatrix} 1\,0 \cdots 0\,0 & -C_{l-1} \\ 0\,1 \cdots 0\,0 & -C_{l-2} \\ 0\,0 \cdots 0\,0 & -C_{l-3} \\ \vdots\,\vdots\,\ddots\,\vdots\,\vdots & \vdots \\ 0\,0 \cdots 0\,1 & -C_1 \\ 0\,0 \cdots 0\,0 & -C_0 \end{bmatrix}, \tag{8}$$

where $C_i = (t^l + \sum_{j=i}^{l-1} f_j t^j)/t^i$. Using the fact that the determinant of a triangular matrix is the product of all diagonal entries, we easily obtain that $\det(F) = -C_0 = -f(t)$ and F is invertible modulo b if and only if $\gcd(f(t), b) = 1$ and $f(t) \neq 0$. We remark that this invertibility condition of F modulo b is always satisfied when $p = f(t)$ is an odd number, for an even radix b.

We analyze the performance of Algorithm 3 in terms of the number of single-precision multiplications and single-precision additions/subtractions. The overhead caused by additions and subtractions shall not be ignored. Additions and subtractions are ignored in many literature, however, the difference between addition/subtraction and multiplication is not significant in many modern microprocessors. The latency of `add` and `sub` instructions is only one clock cycle on Intel Pentium 4 Family 4 processors. However, when long integer addition operation is performed, they are used only when adding or subtracting the least significant digits. The rest of the digits are added or subtracted with slow `adc` (add with carry) and `sbb` (subtract with borrow) instructions, whose latency is 10 clock cycles. These instructions are only 9% faster than `mul` instruction, whose latency is 11 clock cycles [8].

For convenience, we use Intel x86 instructions `mul`, `add` and `adc` to denote the following operations:

- `mul`: single-precision multiplication,
- `add`: addition/subtraction without carry/borrow,
- `adc`: addition/subtraction with carry/borrow.

When multiplying n-digit integer with a single-digit integer, it is clear that n `mul` instructions are required. The numbers of required `add` and `adc` instructions are 1 and $(n-1)$, respectively. When adding i-digit and j-digit integers, the required number of `add` and `adc` instructions are one and $\min(i, j)$, respectively, assuming that carry does not propagate more than one digit place above the most significant digit of the shorter operand. The probability of having carry above the most significant digit place of the shorter integer is $1/2$. The probability that the carry will propagate one more digit place is only $1/b$. Similar argument holds for subtracting two long integers.

Straightforward computation of $\boldsymbol{u}_i = -F^{-1} \cdot \boldsymbol{x}_i \bmod b$ requires l^2 `mul` and $(l^2 - l)$ `add` instructions. However, exploiting the special structure of F, we can compute $\boldsymbol{u}_i$ using only $(2l-1)$ `mul` and $2(l-1)$ `add` instructions, provided that we are allowed to have l-digit pre-computed values that depend on the coefficients of $f(t)$ and the value t.

Theorem 2. *The computation $\boldsymbol{u}_i = -F^{-1} \cdot \boldsymbol{x}_i \bmod b$ can be performed using only $(2l-1)$* `mul` *and $2(l-1)$* `add` *instructions, using l-digit pre-computed values that depend only on the coefficients of $f(t)$ and the value t.*

Proof. Further row operations from (8) easily reveals the exact form of $F' = -F^{-1}$ as follows:

$$F' = \frac{-1}{C_0} \begin{bmatrix} C_0 - C_{l-1}t^{l-1} & -C_{l-1}t^{l-2} & \cdots & -C_{l-1}t^2 & -C_{l-1}t & -C_{l-1} \\ tC_0 - C_{l-2}t^{l-1} & C_0 - C_{l-2}t^{l-2} & \cdots & -C_{l-2}t^2 & -C_{l-2}t^1 & -C_{l-2} \\ t^2C_0 - C_{l-3}t^{l-1} & tC_0 - C_{l-3}t^{l-2} & \cdots & -C_{l-3}t^2 & -C_{l-3}t^1 & -C_{l-3} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ t^{l-2}C_0 - C_1t^{l-1} & t^{l-3}C_0 - C_1t^{l-2} & \cdots & tC_0 - C_1t^2 & C_0 - C_1t & -C_1 \\ -t^{l-1} & -t^{l-2} & \cdots & -t^2 & -t & -1 \end{bmatrix}, \tag{9}$$

where $C_i = (t^l + \sum_{j=i}^{l-1} f_j t^j)/t^i$. Now, we can express F' as follows,

$$F' = F'_1 - F'_2, \tag{10}$$

where,

$$F'_1 = \begin{bmatrix} \frac{C_{l-1}}{C_0}\boldsymbol{v} \\ \frac{C_{l-2}}{C_0}\boldsymbol{v} \\ \frac{C_{l-3}}{C_0}\boldsymbol{v} \\ \vdots \\ \frac{C_1}{C_0}\boldsymbol{v} \\ \frac{1}{C_0}\boldsymbol{v} \end{bmatrix}, \qquad F'_2 = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & 0 \\ t & 1 & \cdots & 0 & 0 & 0 \\ t^2 & t & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ t^{l-2} & t^{l-3} & \cdots & t & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 \end{bmatrix}, \qquad \boldsymbol{v} = [t^{l-1}, \ldots, t^1, t, 1]. \tag{11}$$

The matrix-vector product $F'_2 \cdot \boldsymbol{x}_i \bmod b$ can be computed using Horner's rule, and it requires only $(l-2)$ single-precision multiplications and $(l-2)$ single-precision additions. The vector product $\boldsymbol{v} \cdot \boldsymbol{x}_i$ can be computed by multiplying $(t \bmod b)$ to the $(l-1)$-th entry of $F'_2 \cdot \boldsymbol{x}_i \bmod b$, and then adding $(f_0 \bmod b)$ to the result. Assuming that $1/C_0 \bmod b$ and $C_i/C_0 \bmod b$ for $i = 1, \ldots, l-1$ are precomputed, computing $F'_1 \cdot \boldsymbol{x}_i \bmod b$ requires only l single-precision multiplications. It only remains to compute $F' \cdot \boldsymbol{x}_i = F'_1 \cdot \boldsymbol{x}_i - F'_2 \cdot \boldsymbol{x}_i$ using l single-precision subtractions.

Algorithm 4 explicitly shows how $\boldsymbol{u}_i$ is computed using $(2l-1)$ `mul` and $2(l-1)$ `add` instructions. Since $l \geq 2$, Algorithm 4 always performs better than the straightforward matrix-vector product, which requires l^2 `mul` and $(l^2 - l)$ `add` instructions.

We analyze the line 4 of Algorithm 3. Since each row of F contains only one t and f_i, the matrix-vector product $F \cdot \boldsymbol{u}_i$ requires l multiplications of t and f_i's by a 1-digit integer, and some additions/subtractions. Let n and k $(\leq n)$ be the digit length of t and f_i, respectively and let τ be the number of non-zero f_i's in $f(t)$. Then the number of `mul` required in line 4 is $(ln + \tau k)$. If f_i's are small powers of 2 or integers with very small Hamming weight, multiplications by f_i's can be efficiently computed, replacing τk multiplications with τ bit shifts.

We now count the numbers of `add` and `adc` instructions in line 4. There are l multiplications of t with single-digit integers from $\boldsymbol{u_i}$, and the total numbers of `add` and `adc` instructions required in this computation are l and $l(n-1)$, respectively. There are τ multiplications of f_i with one digit integer from $\boldsymbol{u_i}$, and the total numbers of `add` and `adc` instructions are τ and $\tau(k-1)$, respectively. The matrix vector product $F \cdot \boldsymbol{u_i}$ involves $(l-1)$ additions/subtractions of $(n+1)$-digit integer and a single digit integer. This can be computed with $(l-1)$ `add` and `adc` instructions. There are τ additions/subtractions of an $(n+1)$-digit integer with a $(k+1)$-digit integer. Since $k \leq n$ by definition, this computation requires τ `add` and $\tau(k+1)$ `adc` instructions. So far, the numbers of `add` and `adc`

Algorithm 4 Computing $F' \cdot \boldsymbol{x} \bmod b$

Require: $f(t) = t^l + f_{l-1}t^{l-1} + \cdots f_1 t + f_0$, $\boldsymbol{x} = [x_{l-1}, \ldots, x_1, x_0]$ and pre-computed values $C_i/C_0 \bmod b$ for $i = 1, \ldots, l-1$ and $1/C_0 \bmod b$, where $C_i = (t^l + \sum_{j=i}^{l-1} f_j t^j)/t^i$.
Ensure: $F' \cdot \boldsymbol{x}^T = [u_{l-1}, \ldots, u_1, u_0]^T$.
1: $v_{-1} \leftarrow 0$.
2: **for** i from 0 to $l-1$ **do**
3: $\quad v_i \leftarrow v_{i-1} \cdot t + x_{l-1-i} \bmod b$ ($l-1$ `mul`, $l-1$ `add`)
4: **end for**
5: **for** i from 0 to $l-2$ **do**
6: $\quad u_i \leftarrow v_{l-1} \cdot C_i/C_0 \bmod b$. ($l-1$ `mul`)
7: **end for**
8: $u_{l-1} \leftarrow v_{l-1}/C_0 \bmod b$. (1 `mul`)
9: **for** i from 0 to $l-2$ **do**
10: $\quad u_i \leftarrow u_i - v_i \bmod b$. ($l-1$ `add`)
11: **end for**
12: **return** $[u_{l-2}, \ldots, u_1, u_0]^T$.

instructions in $F \cdot \boldsymbol{u}_i$ have been counted. It only remains to add $F \cdot \boldsymbol{u}_i$ to $\boldsymbol{x}_i$. This computation requires l `add` and $l(n+1)$ `adc` instructions. In total, the numbers of `add` and `adc` instructions required in line 4 are $3l + 2\tau - 1$ and $l(2n+1) + 2\tau k - 1$, respectively.

The total number of `mul`, `add` and `adc` instructions required in Algorithm 3, not considering the final subtraction step, is summarized as follows:

$$\begin{aligned} \#\texttt{mul} &= q(l(n+2) + \tau k - 1), \\ \#\texttt{add} &= q(5l + 2\tau - 3), \\ \#\texttt{adc} &= q(l(2n+1) + 2\tau k - 1). \end{aligned}$$

3.4 Conversions to and from the Montgomery Domain

To perform modular multiplication using Algorithm 3 as a coefficient reduction algorithm, we must transform operands to the Montgomery domain. For $x(t) \in \mathbb{Z}[t]/f(t)$, we compute $\overline{x}(t) \equiv x(t) \cdot b^q \pmod{p}$. This computation can be easily achieved by multiplying two polynomials $x(t)$ and $y(t) \equiv b^{2q} \bmod p$, and then reduce coefficients using Algorithm 3. The result will be $\overline{x}(t) \equiv x(t) \cdot b^q \bmod p$, as desired. It is convenient to have $y(t) \equiv b^{2q} \bmod p$ pre-computed for each p. Conversion from the Montgomery domain can be performed by directly applying Algorithm 3 on $\overline{x}(t)$. The result is $\overline{x}(t) \cdot b^{-q} \equiv x(t) \cdot b^q \cdot b^{-q} \equiv x(t) \pmod{p}$, as desired.

3.5 Interesting Implementation Options

We consider some special cases for which Algorithm 3 can speed up.

– Special Case I: $t \equiv 0 \pmod{b}$.
In such a case, it can be shown that

$$F' = -F^{-1} = \begin{bmatrix} -1 & 0 & 0 & \cdots & 0 & f_{l-1}/f_0 \\ 0 & -1 & 0 & \cdots & 0 & f_{l-2}/f_0 \\ 0 & 0 & -1 & \cdots & 0 & f_{l-3}/f_0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -1 & f_1/f_0 \\ 0 & 0 & 0 & \cdots & 0 & 1/f_0 \end{bmatrix} \bmod b.$$

Note that F is invertible if and only if $f_0 \neq 0$ and $\gcd(f_0, b) = 1$. Then, $\boldsymbol{u}_i = F' \cdot \boldsymbol{x}_i \bmod b$ can be computed with only τ `mul` and $(l-1)$ `add` instructions, provided that $f_i/f_0 \bmod b$ for $i = 1, \ldots, l-1$ and $1/f_0 \bmod b$ are pre-computed. Hence, compared to the general case, we save $(2l - \tau - 1)$ `mul` and $(l-1)$ `add` instructions in line 3 of Algorithm 3.
When computing $F \cdot \boldsymbol{u}_i$, we can save l `mul` instructions and one `adc` instruction, since the least significant digit of t is zero. The total number of saved instructions is given as follows:

$$\begin{aligned} \#\texttt{mul}_{save} &= q(3l - \tau - 1), \\ \#\texttt{add}_{save} &= q(l-1), \\ \#\texttt{adc}_{save} &= q. \end{aligned}$$

– Special Case II: f_i's are powers of 2.
In such a case, multiplication by f_i's can be simply performed by bit shifts. In line 3, there is no speed up. The number of instructions we can save in line 4 is given below.

$$\begin{aligned} \#\texttt{mul}_{save} &= q\tau k, \\ \#\texttt{add}_{save} &= q\tau, \\ \#\texttt{adc}_{save} &= q\tau(k-1). \end{aligned}$$

Note that Algorithm 3 requires ql bit shift instructions in exchange for the above saved instructions.

4 Modular Multiplication Stability

In this section, we carefully analyze the bounds on the input and output of Algorithm 3. The conditions on the parameters for which we can eliminate the final subtractions in Algorithm 3 are determined.

4.1 Montgomery Reduction with Final Subtractions

Suppose that the number of iterations q in Algorithm 3 is the same as the digit size of t, i.e., $q = n$. Suppose that $||\boldsymbol{x}_0||_\infty \leq \lambda\psi^2$, where $||\cdot||_\infty$ is the maximum norm of $\boldsymbol{x}$ defined as $||[x_l - 1, \ldots, x_1, x_0]||_\infty = \max(|x_{l-1}|, \ldots, |x_1|, |x_0|)$. This is the case when the coefficients of input polynomial to the LWPFI

modular multiplication are at most ψ in magnitude. Since $\psi = t - 1$ leads to a maximally redundant signed-digit representation, we assume that $\psi < t$. The bound on the output of Algorithm 3 is determined as follows:

$$\begin{aligned} ||\boldsymbol{x}_n||_\infty &= \left|\left| \frac{\boldsymbol{x}_0 + F \cdot (\sum_{i=0}^{n-1} \boldsymbol{u_i} b^i)}{b^n} \right|\right|_\infty \\ &< \frac{\lambda t^2 + b^n \cdot (t + \xi)}{b^n} \\ &< (\lambda + 1)t + \xi. \end{aligned}$$

We used the fact that the magnitude of the sum of elements in each row of F is maximum at the l-th row and it is $(t + \xi)$. Since the magnitude of the output coefficients may be greater than ψ, we must perform final subtractions to reduce the coefficients. To perform final subtractions we find integers h_i and s_i such that $x_i = h_i t + s_i$, where $-t/2 \leq s_i < t/2$, for each coefficient x_i. Then compute $x_i \leftarrow s_i + h_{i-1} - h_l \cdot f_i$ for $i = 0, \ldots, l - 1$. Then the new coefficients are at most $t/2 + (\lambda + 2)(\xi + 1)$ in magnitude. This method is fast when λ is very small. In such a case, h_i and s_i can be obtained by additions and subtractions. However it may require long integer division if λ is large.

4.2 Montgomery Reduction without Final Subtractions

Under some conditions with a $q \geq n$, we show that it is possible to avoid the final subtractions. We use an approach similar to [17]. Suppose that $||\boldsymbol{x}_0||_\infty \leq \lambda \Psi^2$ in magnitude, where Ψ is a positive integer. This is the case when the coefficients of the input polynomials to the LWPFI modular multiplication are at most Ψ in magnitude. To maintain input/output stability, we have to ensure that

$$||\boldsymbol{x}_q||_\infty = \left|\left| \frac{\boldsymbol{x}_0 + F \cdot (\sum_{i=0}^{q-1} \boldsymbol{u_i} b^i)}{b^q} \right|\right|_\infty < \frac{\lambda \Psi^2}{b^q} + t + \xi \leq \Psi. \tag{12}$$

Solving (12) for Ψ, we obtain

$$\frac{b^q - \sqrt{b^{2q} - 4\lambda(t + \xi)b^q}}{2\lambda} \leq \Psi \leq \frac{b^q + \sqrt{b^{2q} - 4\lambda(t + \xi)b^q}}{2\lambda}. \tag{13}$$

The above solution has real roots when $t \leq b^q/(4\lambda) - \xi$. However, this condition only assures that there exist a real solution for Ψ, whereas Ψ must be an integral value. A sufficient condition for the existence of at least one integral value of Ψ is given as follows:

$$\sqrt{b^{2q} - 4\lambda(t + \xi)b^q} \geq \lambda. \tag{14}$$

It follows that

$$t \leq \frac{b^{2q} - \lambda^2}{4\lambda b^q} - \xi = \frac{b^q}{4\lambda} - \frac{\lambda}{4b^q} - \xi. \tag{15}$$

For t satisfying (15), there exists at least one integral value for Ψ. Since the width of the interval (13) is at least one, the choice $\Psi = \lfloor b^q/(4\lambda) \rceil$, where $\lfloor x \rceil$ denotes the nearest integer from x, is always within the interval. For this value of Ψ, it always holds that $t + \xi \leq \Psi$.

Suppose that $t+\xi$ is an n'-bit integer, i.e., $2^{n'-1} \leq t+\xi < 2^{n'}$, It can be easily verified that an n'-bit integer $t+\xi$ satisfies (15) if

$$\lambda < \frac{b^q}{4 \cdot 2^{n'}}. \tag{16}$$

Note that $\lambda = ((\xi+1)^l - 1)/\xi \geq 3$, since $\xi \geq 1$ and $l \geq 2$. Let w be the bit length of a digit, i.e., $b = 2^w$. It follows that ξ and q must be chosen such that the following holds:

$$3 \leq ((\xi+1)^l - 1)/\xi < 2^{qw-n'-2}. \tag{17}$$

For efficient implementation, the number of iterations q must be as small as possible, but not smaller than n. For instance, $q = n$ or $q = n+1$, where n is the digit length of $t+\xi$, would be the most interesting cases for implementation. If $n' = nw - \rho$, where $4 \leq \rho < w$, and ξ is chosen such that $3 \leq \lambda < 2^{\rho-2}$, then $q = n$ satisfies (16). If $n' = nw - \rho$, where $0 \leq \rho < w$, and ξ is chosen such that $3 \leq \lambda < 2^{w+\rho-2}$, then $q = n+1$ satisfies (16).

One may consider using $f(t)$ such that only f_0 and f_1 are non-zero, but $f_i = 0$ for $2 \leq i \leq l-1$. In such a case, it can be proven that the coefficients of the output of POLY-REDC step are at most $(l-1)(|f_0| + |f_1|)\Psi^2$. Then we can choose l, f_0 and f_1 such that $(l-1)(|f_0| + |f_1|) < 2^{qw-n'-2}$. Using this method, we can choose larger coefficients f_0 and f_1 than we can with the condition (17). Note that such an $f(t)$ is also used in [4].

To convert $\overline{x}(t) = x(t) \cdot b^q$ from Montgomery domain to the usual domain, we can simply apply Algorithm 3 on $\overline{x}(t)$. In such a case, coefficients of the input are bounded by Ψ. The output $\boldsymbol{x_q}$ of Algorithm 3 is bounded as follows:

$$||\boldsymbol{x_q}||_\infty \leq \frac{\Psi + (b^q - 1)(t+\xi)}{b^q} \leq \frac{\Psi + (b^q-1)\Psi}{b^q} = \Psi. \tag{18}$$

Therefore, the final subtractions are not required even after the final conversion.

Another interesting stability condition is to make the output bounded by b^{q-1}. This may be useful to prevent any potential side channel threat that may exploit the probability on the digit length of the output. Considering that the magnitude of input to Algorithm 3 is bounded by λb^{2q-2}, we obtain the following stability condition:

$$||X_q||_\infty < \frac{\lambda b^{2q-2}}{b^q} + t + \xi \leq b^{q-1}. \tag{19}$$

It follows that $t \leq b^{q-2}(b-\lambda) - \xi$. Of course, we must ensure that $b > \lambda$, since otherwise we will have a negative t. It is easily seen that any $t < b^{q-1}$. If t is chosen as above, the output of Algorithm 3 will be strictly within b^{q-1} bound. When converting back to usual domain, we have that

$$||X_q||_\infty < \frac{b^{q-1} + (b^q-1)(t+\xi)}{b^q} \leq t + \xi. \tag{20}$$

5 Additions and Subtractions

In this section, we study additions and subtractions modulo an LWPFI. It is well-known that redundant signed-digit representation allows carry/borrow free additions/subtractions [2]. Here we derive similar methods for additions and subtractions modulo an LWPFI.

Algorithm 5 Short Coefficient Reduction

Require: $a(t) = (a_{l-1}, \ldots, a_1, a_0)_t$.
Ensure: $c = (c_{l-1}, \ldots, c_1, c_0)_t \equiv a(t) \pmod{p}$, where $c_i \leq \psi$.
1: $t_0 \leftarrow 0$.
2: **for** i from 0 to $l-1$ **do**
3: (Transfer Digit) $t_{i+1} \leftarrow C(a_i)$.
4: (Reduction) $w_i \leftarrow p_i - t_{i+1} \cdot t$.
5: (Sum) $s_i \leftarrow w_i + t_i$.
6: **end for**
7: **for** i from 0 to $l-1$ **do**
8: (Reduction modulo $f(t)$) $c_i = s_i - t_l \cdot f_i$.
9: **end for**
10: **return** $c = (c_{l-1}, \ldots, c_1, c_0)_t$.

To perform an addition or a subtraction of two numbers $x(t)$ and $y(t)$, we first compute coefficient-wise additions and subtractions as follows:

$$z(t) = x(t) \pm y(t) = \sum_{i=0}^{l-1} (x_i \pm y_i) t^i,$$

where $x(t)$ and $y(t)$ are polynomials of degree $(l-1)$ and the coefficients are at most ψ in magnitude. The coefficients of $z(t)$ will be at most 2ψ in magnitude. Algorithm 5 efficiently reduces the coefficients of $z(t)$ so that the coefficients of the resulting polynomial are at most ψ in magnitude. We call it the *short coefficient reduction*.

Algorithm 5 is based mostly on the carry-free addition or borrow-free subtraction algorithm widely used in redundant signed-digit arithmetic [2]. The only difference is the inclusion of reduction modulo $f(t)$. The reduction modulo $f(t)$ is necessary, since the result must be represented with $(l-1)$ coefficients. The function $C(\cdot)$, depending on the value of its input, outputs an integer in $[-t_{\max}, t_{\max}]$ for some positive integer $t_{\max}$, We will determine the exact behavior of $C(\cdot)$ in the following.

Note that the c_i's computed in Algorithm 5 must satisfy the condition, $-\psi \leq c_i \leq \psi$. Since $-t_{\max} \leq C(a_i) \leq t_{\max}$ for some positive integer $t_{\max}$ and $|f_i| \leq \xi$, $w_i + t_i$'s at line 5 must satisfy

$$-\psi + t_{\max} \cdot \xi \leq w_i + t_i \leq \psi - t_{\max} \cdot \xi. \tag{21}$$

Substituting $w_i = p_i - t_{i+1} \cdot t$ into (21) results in

$$\frac{p_i - (\psi - t_{\max} \cdot \xi - t_i)}{t} \leq t_{i+1} \leq \frac{p_i + (\psi - t_{\max} \cdot \xi + t_i)}{t}.$$

Suppose that $-\mu + t_{\max} \cdot \xi \leq t_i \leq \mu - t_{\max} \cdot \xi$, where $\mu = (\xi + 1) \cdot t_{\max}$. In the worst case, the range of t_{i+1} is restricted by the following inequality:

$$\frac{p_i - (\psi - \mu)}{t} \leq t_{i+1} \leq \frac{p_i + (\psi - \mu)}{t}.$$

We consider two interesting cases.

- Case I: $t/2 + \xi + 1 \leq \psi \leq t - (\xi + 1)$.
- Case II: $t + 2(\xi + 1) \leq \psi \leq 2t - 2(\xi + 1)$.

5.1 Case I: $t/2 + \xi + 1 \leq \psi \leq t - (\xi + 1)$

We assume that ψ satisfies the following:

$$\frac{t}{2} + \xi + 1 \leq \psi \leq t - (\xi + 1). \tag{22}$$

We plot a graph of t_{i+1} versus p_i in Figure 1. Note that $t_{i+1} \in \mathbb{Z}$ must be chosen within the parallelogram ABCD. It is easy to show that there always exists an integral value of a transfer digit t_{i+1} for all p_i in $[-2\psi, 2\psi]$, since the following inequalities are immediate from (22).

1. $1 \geq (\psi + \mu)/t$: this is the reason why we can choose $t_{\max} = 1$.
2. $\psi - \mu \geq t - (\psi - \mu)$.

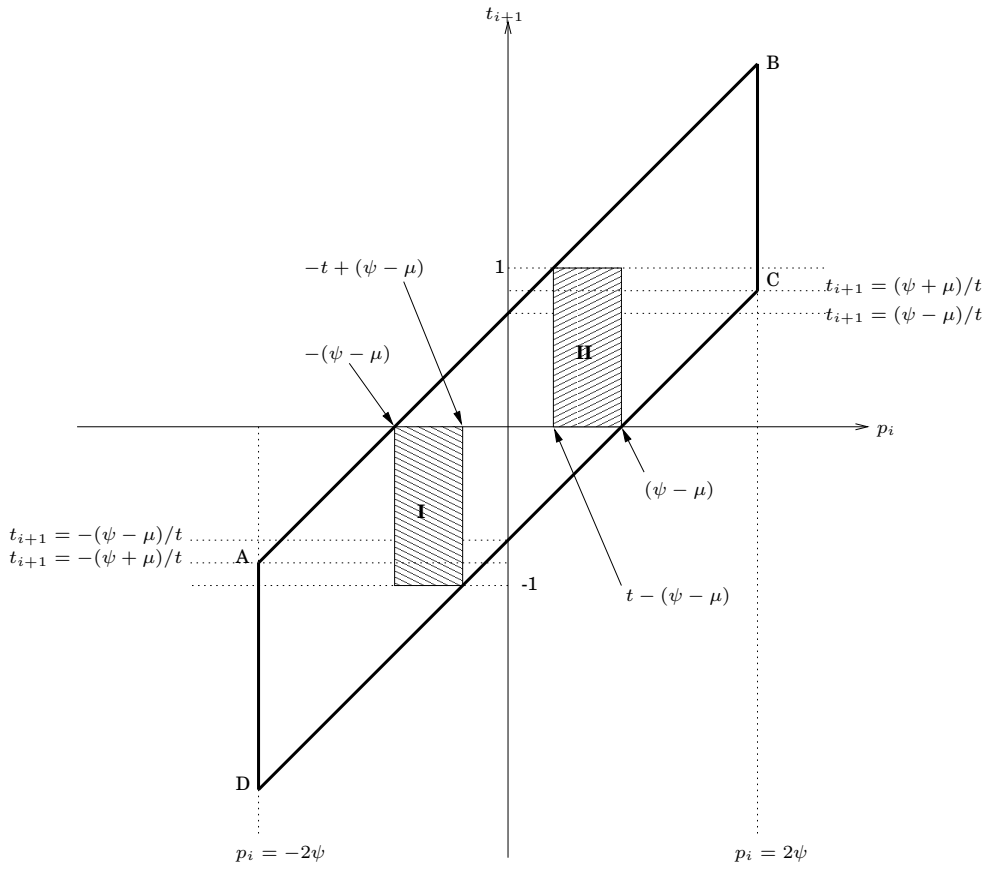

Fig. 1. Range of Transfer Digit t_{i+1} ($\mu = \xi + 1$)

Now we can define $C(\cdot)$ as follows:

$$C(x) = \begin{cases} -1 & (\text{if } \leq x < -\mathcal{C}), \\ 0 & (\text{if } -\mathcal{C} \leq x < \mathcal{C}), \\ 1 & (\text{if } \mathcal{C} \leq x), \end{cases} \tag{23}$$

where $\mathcal{C}$ is any positive integer in the range $[t - (\psi - \mu), \psi - \mu]$.

Even though any $\psi > t/2 + \xi$ can be used to represent any element in $\mathbb{Z}_p$ as shown in Proposition 1, only ψ that satisfies (22) allows carry/borrow free addition/subtraction. If ψ is chosen in the range (22), one can use Algorithm 5 to perform final subtractions required at the end of Algorithm 3. In Section 4.1, we have observed that $||\boldsymbol{x_n}||_\infty < (\lambda + 1)t + \xi$. Therefore, at most $\lfloor((\lambda + 1)t + \xi)/\psi\rfloor$ executions of Algorithm 5 are required to reduce the result to $[-\psi, \psi]$. It is advantageous to choose $\psi = t - (\xi + 1)$, to reduce the number of executions of Algorithm 5.

5.2 Case II: $t + 2(\xi + 1) \leq \psi \leq 2t - 2(\xi + 1)$

The conditions required to perform Algorithm 3 without final subtractions that we have discussed in Section 4.2 result in $t + \xi \leq \Psi$. Hence, this condition results in an overly redundant signed-digit representation. For completeness, we study the conditions to perform carry/borrow free additions/subtractions in overly redundant signed-digit representation.

Suppose $\psi = \Psi$ is an integer such that the following holds:

$$t + 2(\xi + 1) \leq \Psi \leq 2t - 2(\xi + 1). \tag{24}$$

We plot a graph of t_{i+1} versus p_i in Figure 2. Note that $t_{i+1} \in \mathbb{Z}$ must be chosen within the parallelogram ABCD. It is easy to show that there always exists an integral value of a transfer digit t_{i+1} for all p_i in $[-2\Psi, 2\Psi]$, since the following inequalities are immediate from (24).

1. $(\Psi + \mu)/t \leq 2$.
2. $1 \leq (\Psi - \mu)/t$: due to this, $t_{\max} = 1$ cannot be chosen.
3. $\Psi - \mu \geq t$.
4. $2t - (\Psi - \mu) \leq t$.
5. $-t + (\Psi - \mu) \geq 0$.

Since it is better to have smaller set of digits for $C(\cdot)$, we have chosen $t_{\max} = 2$. Observe that t is always within the range $2t - (\Psi - \mu) \leq t \leq \Psi - \mu$. Therefore, $C(\cdot)$ can be defined as follows:

$$C(x) = \begin{cases} -2 & (\text{if } x < -t), \\ 0 & (\text{if } -t \leq x < t), \\ 2 & (\text{if } t \leq x). \end{cases} \tag{25}$$

Note that $C(\cdot)$ defined in (25) does not require a pre-determined constant, unlike (23) which uses a constant $\mathcal{C}$. Moreover, $C(\cdot)$ does not output ± 1. Hence, computing $C(\cdot)$ as shown in (25) is as fast as computing $C(\cdot)$ as in (23).

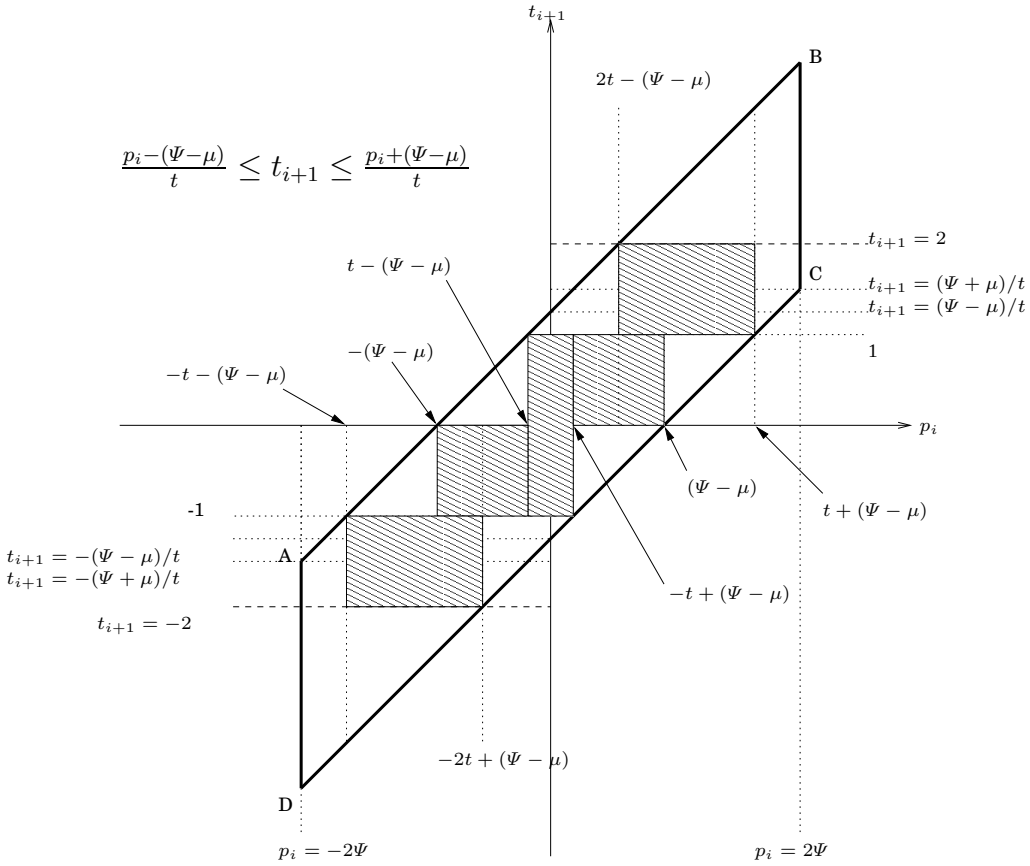

Fig. 2. Range of Transfer Digit t_{i+1} ($\mu = 2(\xi + 1)$)

6 Comparisons

In this section, we compare our new coefficient reduction algorithm with Algorithm 2. For fairness of comparison, we let the modulus m used in Algorithm 2 be an nl-digit integer. Note that a degree-l polynomial $f(t)$ with n-digit t generates an nl-digit LWPFI.

We use the same technique that we use in Section 3.3 to analyze Algorithm 2. In line 3, only one `mul` is required. In line 4, the computation of $u_i \cdot m$ requires nl `mul`, 1 `add` and $(nl-1)$ `adc` instructions. Adding $u_i \cdot m$, which is at most $(nl+1)$ digits long, to T_i requires 1 `add` and $(nl+1)$ `adc` instructions. Therefore, Algorithm 2 requires the following number of instructions, not considering the final subtraction:

$$\begin{aligned} \#\texttt{mul} &= q(nl+1), \\ \#\texttt{add} &= 2q, \\ \#\texttt{adc} &= 2qnl. \end{aligned}$$

Note that q in Algorithm 2 is not the same as the one used in Algorithm 3. Final subtractions in Algorithm 2 can be avoided in the case $b \geq 4$ by simply letting $q = nl+1$. In Algorithm 3, we suppose $q = n+1$ eliminates the necessity of final subtractions. Note that such a value for q can be chosen if ξ is reasonably small. The details on this have been discussed at the end of Section 4.2.

Table 1. Comparison of Algorithm 2 and Algorithm 3

Instruction	Algorithm 2	Algorithm 3
`mul`	$(nl+1)^2$	$ln^2 + (\tau k + 3l - 1)n + 2l + \tau k - 1$
`add`	$2(nl+1)$	$(n+1)(5l + 2\tau - 3)$
`adc`	$2n^2l^2 + 2nl$	$2ln^2 + (2\tau k + 3l - 1)n + 2\tau k + l - 1$

In Table 1, we clearly observe that Algorithm 2 requires $O(n^2l^2)$ operations, whereas Algorithm 3 requires $O(ln^2)$ operations. Hence, Algorithm 3 does have better asymptotic behavior than Algorithm 2. However, this does not mean that Algorithm 3 is always faster than Algorithm 2. If actual values for parameters n, l, τ and k are substituted in Table 1, the required number of operations for Algorithm 3 may be larger. However, it is clear that the larger n and l, the better Algorithm 3 will perform.

7 Applications of LWPFI Modular Multiplications

Many cryptosystems rely on the ability to perform modular arithmetic modulo large integers. Among the modular arithmetic, modular multiplication is the most frequently used operation. In most cases, the modulus has to be a prime number. One can randomly try t until $f(t)$ is a prime to use it in cryptosystems requiring modular multiplications. One may find t such that $f(t)$ has a large enough prime factor suitable for certain cryptosystems. We denote such a factor p'. In this case, we can embed any modular arithmetic modulo p' in slightly larger ring $\mathbb{Z}_{f(t)}$, where we can use efficient modular multiplications using LWPFI moduli. Note, however, this method is faster only if the modular multiplication using LWPFI is faster than the modular multiplication modulo p' using usual integer

arithmetic. After all computations have been performed, the result must be converted to the usual representation of integers and be taken modulo p'.

The idea of embedding arithmetic into a larger ring, where computations are easy, is not at all new. The similar technique is used for efficient multiplication in finite fields [19] [7].

8 Application to Modular Number Systems?

In this section, we investigate if Algorithm 3 can be applied also in modular number systems (MNS) proposed in [3,4] by Bajard et. al.

Definition 2 (Modular Number System). *A Modular Number System (MNS) $\mathcal{B}$, is a quadruple p, l, γ, ρ), such that all positive integers $0 \leq x < p$ satisfy*

$$x = \sum_{i=0}^{l-1} x_i \gamma^i \bmod p, \text{ with } \gamma > 1 \text{ and } |x_i| < \rho \tag{26}$$

The MNS has some similarities with LWPFIs. For instance, numbers are represented in polynomials and the steps for modular multiplications are quite similar: multiplication is first performed in $\mathbb{Z}[t]/f(t)$ and then the coefficients are reduced by a coefficient reduction algorithm. However, the coefficient reduction method used in MNS is different from the one used in LWPFI modular multiplication. The difference is mainly due to the fact that γ can lie in much wider range than t. In particular, $0 \leq \gamma < p$, while the size of t used in LWPFI is approximately that of $p^{1/l}$.

As a special case of MNS, adapted modular number system (AMNS) is proposed in [3]. AMNS is an MNS where $\gamma^l \bmod p = c$ for some small integer c. Each iteration of the coefficient reduction algorithm (Algorithm CR in [3]) reduces $\lceil 3s/2 \rceil$ bits to $s + 1$ bits. Some repetition of CR produces polynomial whose coefficients are at most $s + 1$ bits long. Modular multiplication in AMNS is very efficient, however, it appears that the suitable sets of parameters that allows efficient computation in AMNS are quite difficult to find and are scarce.

In a recent paper [4], Bajard et. al. have proposed another special case of MNS called the polynomial modular number system (PMNS).

Theorem 3 (Fundamental theorem of an MNS). *Let $p, l > 1$. Also define $E(X) = X^l + \alpha X + \beta$, with $\alpha, \beta \in \mathbb{Z}$, such that $E(\gamma) \equiv 0 \pmod{p}$, and E irreducible in $\mathbb{Z}[X]$. Then, we can define a modular number system $\mathcal{B} = MNS(p, l, \gamma, \rho)$ provided that*

$$\rho \geq (|\alpha| + |\beta|)p^{1/l}. \tag{27}$$

Definition 3. *An MNS, $\mathcal{B} = MNS(p, l, \gamma, \rho)$, which satisfies the conditions of Theorem 3 is called a Polynomial Modular Number System (PMNS).*

The coefficient reduction algorithms presented in [4] reduces one bit at a time in each iteration using a look-up table.

To perform the Montgomery reduction algorithm for MNS , we need an $l \times l$ matrix having similar properties as the matrix F used in Algorithm 3. However, we have not been able to find such a matrix for MNS. We have considered two candidate matrices, but they do not lead to satisfactory results.

Consider a matrix A as follows.

$$A = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & 0 \\ -\gamma & 1 & \cdots & 0 & 0 & 0 \\ 0 & -\gamma & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & 0 \\ 0 & 0 & \cdots & -\gamma & 1 & 0 \\ 0 & 0 & \cdots & 0 & -\gamma & p \end{bmatrix} \tag{28}$$

The matrix A is constructed similarly as (7) except for the last column. The column vectors in A are all congruent to 0 modulo p and A has inverse modulo b if and only if $\gcd(p, b) = 1$ and $p \neq 0$. Hence, the matrix A is a good candidate for the matrix F in Algorithm 3, to compute $x(t) \cdot b^{-q} \bmod p$. Unfortunately, the entries in A are too large ($0 \leq \gamma < p$) to compute line 4 of Algorithm 3 efficiently. Due to the same reason, it is clear that Algorithm 3 will not reduce the coefficients of input polynomial if F is given as (28), regardless of the choice of q.

Let $\mathcal{L}(A)$ denote the lattice generated by the column vectors of A. Let v be a vector in $\mathcal{L}(A)$ such that $||\boldsymbol{v}||_\infty \leq p^{1/l}$. Note that there is always a vector $\boldsymbol{v}$ such that $||\boldsymbol{v}||_\infty \leq \sqrt[l]{\det A}$ in any lattice $\mathcal{L}(A)$. We construct another lattice $\mathcal{L}(B)$ generated by B, where

$$B = \begin{bmatrix} v_{l-1} & v_{l-2} & \cdots & (v_0 - \alpha v_{l-1}) \\ \vdots & \vdots & \vdots & \vdots \\ v_2 & v_1 & \cdots & (-\beta v_3 - \alpha v_2) \\ v_1 & v_0 - \alpha v_{l-1} & \cdots & (-\beta v_2 - \alpha v_1) \\ v_0 & -\beta v_{l-1} & \cdots & -\beta v_1 \end{bmatrix}. \tag{29}$$

Let $\boldsymbol{b_i}$ denote the i-th column vector of B. Then, $\boldsymbol{b_i} = \gamma^i \boldsymbol{v} \bmod (\gamma^l + \alpha\gamma + \beta)$. It is easily seen that $\max(||\boldsymbol{b_i}||_\infty) \leq (|\alpha| + |\beta|)p^{1/l}$. Clearly, $L(B) \subseteq L(F)$. Therefore, the matrix B is a good candidate for the matrix F in Algorithm 3. If $F = B$ in Algorithm 3, the algorithm will reduce the output to a certain extent. However efficient coefficient reduction is still not achievable since computing $B \cdot \boldsymbol{u_i}$ (at line 4 of Algorithm 3) is not any faster than computing general matrix-vector product.

9 Conclusions

In this report, we have extended LWPFIs presented in [5], and have proposed a new coefficient reduction reduction based on the Montgomery reduction algorithm. Our new coefficient reduction algorithm have been analyzed using the extended definition of LWPFIs. Performance have been analyzed in terms of the number of digit-level multiplications, additions/subtractions and additions/subtractions with carry. Bounds on input and output of the new coefficient reduction algorithm have been carefully analyzed to eliminate the final subtractions in the new coefficient reduction algorithm. As a side result, we have presented methods for performing additions and subtractions modulo an LWPFI in a carry/borrow-free manner. We have also considered applying our coefficient reduction algorithm to modular number systems proposed by Bajard et. al. but have not been successful in finding a good F that can lead to efficient coefficient reduction.

References

1. Erik Agrell, Thomas Eriksson, Alexander Vardy, and Kenneth Zeger. Closest point search in lattices. *IEEE Transactions on Information Theory*, 48(8):2201–2214, August 2002.
2. A. Avizienis. Signed-digit number representation for fast parallel arithmetic. *IRE Transaction on Computers*, EC-10:389–400, 1961.
3. Jean-Claude Bajard, Laurent Imbert, and Thomas Plantard. Modular number systems: Beyond the Mersenne family. In *Selected Areas in Cryptography 2004*, LNCS 3357, pages 159–169. Springer-Verlag, 2004.
4. Jean-Claude Bajard, Laurent Imbert, and Thomas Plantard. Arithmetic operations in the polynomial modular number system. In *Proceedings of the 17th IEEE Symposium on Computer Arithmetic*, ARITH'05, pages 206–213, 2005.
5. Jaewook Chung and M. Anwar Hasan. Low-weight polynomial form integers for efficient modular multiplication, 2006. To appear in IEEE Transactions on Computers. Available at `http://vlsi.uwaterloo.ca/~ahasan/web_papers/technical_reports/web_lwpfi.pdf`.
6. S. A. Cook. *On the Minimum Computation Time of Functions*. PhD thesis, Harvard University, May 1966.
7. Germain Drolet. A new representation of elements of finite fields $GF(2^m)$ yielding small complexity arithmetic circuits. *IEEE Transactions on Computers*, 47(9), 1998.
8. Torbjörn Granlund. Instruction latencies and throughput for AMD and Intel x86 processors, 2005. Available at `http://swox.com/doc/x86-timing.pdf`.
9. Gaël Hachez and Jean-Jacques Quisquater. Montgomery exponentiation with no final subtractions: Improved results. In *Cryptographic Hardware and Embedded Systems - CHES 2000*, LNCS 1965, pages 293–301. Springer-Verlag, 2000.
10. A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Doklady (English translation)*, 7(7):595–596, 1963.
11. P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology - CRYPTO '96*, LNCS 1109, pages 104–113. Springer-Verlag, 1996.
12. Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.
13. Peter L. Montgomery. Five, six, and seven-term Karatsuba-like formulae. *IEEE Transaction on Computers*, 54(3):362–369, 2005.
14. Werner Schindler. A timing attack against RSA with the Chinese remainder theorem. In *Cryptographic Hardware and Embedded Systems - CHES 2000*, LNCS 1965, pages 109–124. Springer-Verlag, 2000.
15. A. L. Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. *Soviet Math*, 3:714–716, 1963.
16. Colin D. Walter. Montgomery exponentiation needs no final subtractions. *Electronics Letters*, 35(21):1831–1832, 1999.
17. Colin D. Walter. Precise bounds for Montgomery modular multiplication and some potentially insecure RSA moduli. In *Topics in Cryptology - CT-RSA 2002*, LNCS 2271, pages 30–39. Springer-Verlag, 2002.
18. Colin D. Walter and Susan Thompson. Distinguishing exponent digits by observing modular subtractions. In *Progress in Cryptology - CT-RSA 2001*, LNCS 2020, pages 192–207. Springer-Verlag, 2001.
19. Huapeng Wu, M. Anwar Hasan, Ian F. Blake, and Shuhong Gao. Finite field multiplier using redundant representation. *IEEE Transactions on Computers*, 51(11):1306– 1316, 2002.