Arithmetic Operations in the Polynomial Modular Number System

Jean-Claude Bajard[1], Laurent Imbert[1,2], and Thomas Plantard[1]

[1] CNRS LIRMM UMR 5506
161 rue Ada, 34392 Montpellier cedex 5, France

[2] ATIPS, CISaC, University of Calgary
2500 University drive NW, Calgary, AB, T2N 1N4, Canada

Abstract

We propose a new number representation and arithmetic for the elements of the ring of integers modulo p. The so-called Polynomial Modular Number System (PMNS) allows for fast polynomial arithmetic and easy parallelization. The most important contribution of this paper is the fundamental theorem of a Modular Number System, which provides a bound for the coefficients of the polynomials used to represent the set $\mathbb{Z}_p$. However, we also propose a complete set of algorithms to perform the arithmetic operations over a PMNS, which make this system of practical interest for people concerned about efficient implementation of modular arithmetic.

Keywords: Number system, Modular arithmetic, Lattice theory, Table-based methods

1. Introduction

Efficient implementation of modular arithmetic is an important prerequisite in today's public-key cryptography [10]. The celebrated RSA algorithm [13], and the cryptosystems based on the discrete logarithm problem, such as Diffie-Hellman key exchange [6], need fast arithmetic modulo integers of size 1024 to roughly 15000 bits. For the same level of security, elliptic curves defined over prime fields, require operations modulo prime numbers whose size range approximately from 160 to 500 bits [8].

Classic implementations use multiprecision arithmetic, where long integers are represented in a predefined high-radix (usually a power of two depending on the word size of the targeted architecture). Arithmetic operations, namely modular reduction and multiplication, are performed using efficient algorithms, such as as Montgomery [12], or Barrett [3]. (For more details, see [10], chapter 14.) These general algorithms do not require the divisor, also called modulus, to be of special form. When this is the case, however, modular multiplication and reduction can be accelerated considerably. Mersenne numbers, of the form $2^m - 1$, are the most common examples. Pseudo-Mersenne numbers [5], generalized Mersenne numbers [14], and their extension [4] are other examples of numbers allowing fast modular arithmetic.

In a recent paper [2], we have defined the so-called Modular Number Systems (MNS) and Adapted Modular Number Systems (AMNS) to speed up the arithmetic operations for moduli which do not belong to any of the previous classes. In this paper, we propose a new representation, and the corresponding arithmetic operations for the elements of $\mathbb{Z}_p$, the ring of integers modulo p. (The integer p does not have to be a prime, although it is very likely to be prime for practical cryptographic applications.) We define the Polynomial Modular Number System (PMNS), over which integers are represented as polynomials. Compared to the classical (binary) representation, polynomial arithmetic offers the advantages of no carry propagation and easiest parallelization. The main contribution of this paper is the fundamental theorem of a MNS, which provides a bound for the coefficients of the polynomials used to represent the elements of $\mathbb{Z}_p$. This theorem is presented in Section 3. It uses results from lattice reduction theory [9, 11]. The second half of the paper focuses on the arithmetic operations; in Section 4, we propose algorithms for the basic operations – addition, multiplication, conversions – which all require a final step, called coefficient reduction, that we present in details in Section 5. A numerical example is provided in Section 6.

2. Modular number systems

In classic positional number systems, every non-negative integer, x, is uniquely represented in radix r as

$$x = \sum_{i=0}^{n-1} x_i \, r^i, \quad \text{where } x_i \in \{0, \ldots, r-1\}. \qquad (1)$$

If $x_{n-1} \neq 0$, x is said to be a n-digit radix-r number.

In most public-key cryptographic applications, computations have to be done over finite rings or fields. In prime fields $GF(p)$, we deal with representatives of equivalence classes modulo p (for simplicity we generally use the set of positive integers $\{0, 1, \ldots, p-1\}$), and the arithmetic operations – addition and multiplication – are performed modulo p. In order to represent the set of integers modulo p, we define a *Modular number system*, by extending the Definition (1) of positional number systems.

Definition 1 (MNS) *A Modular Number System, $\mathcal{B}$, is a quadruple (p, n, γ, ρ), such that every positive integers, $0 \leq x < p$, satisfy*

$$x = \sum_{i=0}^{n-1} x_i \, \gamma^i \bmod p, \quad \textit{with } \gamma > 1 \textit{ and } |x_i| < \rho. \qquad (2)$$

The vector $(x_0, \ldots, x_{n-1})_{\mathcal{B}}$ denotes a representation of x in $\mathcal{B} = MNS(p, n, \gamma, \rho)$.

In the rest of the paper, we shall omit the subscript $(.)_{\mathcal{B}}$ when it is clear from the context. We shall represent the integer, a, either as the vector, $\boldsymbol{a}$, or the polynomial, A, without distinction. We shall use a_i to represent both for the ith element of $\boldsymbol{a}$, and the ith coefficient of A. (Note that we use a left-to-right notation; i.e., a_0, the left-most coefficient of A, is the constant term.) Hence, depending on the context, we shall use $\|\boldsymbol{a}\| = \|A\|$, to refer to the norm of the vector, or the corresponding polynomial. We shall also use the notation $\boldsymbol{a}_i$ to refer to the ith vector within a set of vectors or a matrix.

Example 1 *Let us consider a MNS defined with $p = 17, n = 3, \gamma = 7, \rho = 2$. Over this system, we represent the elements of $\mathbb{Z}_{17}$ as polynomials in γ, of degree at most 2, with coefficients in $\{-1, 0, 1\}$ (cf. table 1).*

1	2	3	4
1	$-\gamma^2$	$1-\gamma^2$	$-1+\gamma+\gamma^2$
5	6	7	8
$\gamma+\gamma^2$	$-1+\gamma$	γ	$1+\gamma$
9	10	11	12
$-1-\gamma$	$-\gamma$	$1-\gamma$	$-\gamma-\gamma^2$
13	14	15	16
$1-\gamma-\gamma^2$	$-1+\gamma^2$	γ^2	$1+\gamma^2$

Table 1. The elements of $\mathbb{Z}_{17}^*$ in the MNS defined as $\mathcal{B} = MNS(17, 3, 7, 2)$

In example 1, we remark that the number of polynomials of degree 2, with coefficients in $\{-1, 0, 1\}$ is equal to $3^3 = 27$. Since we only have to represent 17 values, the system is clearly redundant. For example, we have $6 = 1+\gamma+\gamma^2 = -1+\gamma$, or $9 = 1-\gamma+\gamma^2 = -1-\gamma$. The level of redundancy depends on the parameters of the MNS. Note yet that, in this paper, we shall take advantage of the redundancy only by considering different representations of zero.

In a MNS, every integer, $0 \leq x < p$, is thus represented as a polynomial in γ. But; what do we know about the coefficients of those polynomials? Are they bounded by some value which depends on the parameters of the MNS? In other words, given the integers p and n, are we able we determine ρ and construct a MNS? We answer these questions in the next section. We prove the fundamental theorem of a MNS, using results from lattice reduction theory, and we introduce the concept of Polynomial Modular Number System (PMNS).

3. Polynomial Modular Number Systems

In this section, we consider special cases of modular number systems, where γ is a root (modulo p) of a given polynomial E. In the following fundamental theorem of a MNS, we prove that if ρ is greater than a certain bound, then it is always possible to define a valid MNS. Roughly speaking, Theorem 1 says that there exitst a MNS, $\mathcal{B} = MNS(p, n, \gamma, \rho)$, where one can represent every integer less than p, as a polynomial of degree at most $n-1$, with coefficients all less than $C \times p^{1/n}$, where C is a small constant.

Theorem 1 (Fundamental theorem of a MNS) *Let us define $p, n > 1$, and a polynomial $E(X) = X^n + \alpha X + \beta$, with $\alpha, \beta \in \mathbb{Z}$, such that $E(\gamma) \equiv 0 \pmod p$, and E irreducible in $\mathbb{Z}[X]$. If*

$$\rho \geq (|\alpha| + |\beta|) \, p^{1/n}, \qquad (3)$$

then, the parameters (p, n, γ, ρ) define a modular number system, $\mathcal{B} = MNS(p, n, \gamma, \rho)$.

Sketch of proof: (A complete, detailed, proof can be found in [1].)

The proof is based on the theory of lattice reduction [9, 11]. A *lattice* $\mathcal{L}$ is a discrete sub-group of $\mathbb{R}^n$, or equivalently the set of all the integral combinations of $d \leq n$ linearly independent vectors over $\mathbb{R}$:

$$\begin{aligned}\mathcal{L} = \mathcal{L}(\boldsymbol{A}) &= \mathbb{Z}\,\boldsymbol{a}_1 + \cdots + \mathbb{Z}\,\boldsymbol{a}_d \\ &= \{\lambda_1 \boldsymbol{a}_1 + \cdots + \lambda_n \boldsymbol{a}_d\,;\, \lambda_i \in \mathbb{Z}\}.\end{aligned}$$

The matrix $\boldsymbol{A} = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_d)$ is called a *basis* of $\mathcal{L}$. It is known that, every vector over $\mathbb{R}$ can be reduced, modulo the lattice, within the *fundamental domain* of $\mathcal{L}$, given by

$$\mathcal{H} = \{x \in \mathbb{R}^n\,;\, x = \sum_{i=1}^{d} x_i\,\boldsymbol{a}_i,\ 0 \leq x_i < 1\}.$$

In order to prove Theorem 1, we first define the lattice, $\mathcal{L} = \mathcal{L}(\boldsymbol{A})$, over $\mathbb{Z}^n$, of all the multiples of p in $\mathcal{B}$; or equivalently, the set of vector of $\mathbb{Z}^n$ defined by

$$\begin{aligned}\mathcal{L} = \mathcal{L}(\boldsymbol{A}) = \big\{&(x_0, \ldots, x_{n-1})\,; \\ &x_0 + x_1\gamma + \cdots + x_{n-1}\gamma^{n-1} \equiv 0 \pmod{p}\big\}. \quad (4)\end{aligned}$$

From Minkowski's theorem [9, 7], and because we have $|\det \boldsymbol{A}| = p$, we prove that there exists a vector $\boldsymbol{v} \in \mathcal{L}$, such that $\|\boldsymbol{v}\|_\infty \leq p^{1/n}$. We then define a second lattice, $\mathcal{L}' = \mathcal{L}'(\boldsymbol{B}) \subseteq \mathcal{L}$, of dimension n, with $\boldsymbol{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$, such that

$$\|\boldsymbol{b}_i\|_\infty \leq (|\alpha| + |\beta|)\, p^{1/n}. \quad (5)$$

To conclude the proof, we simply remark that every integer, $a \in \mathbb{N}$, can be first associated with the vector $\boldsymbol{a} = (a, 0, \ldots, 0)$, and reduced modulo $\mathcal{L}'$ to a vector $\boldsymbol{a'}$, which belongs to the fundamental domain $\mathcal{H}'$ of $\mathcal{L}'$. Since $\mathcal{H}'$ can be overlapped by spheres of radius $(|\alpha| + |\beta|)\ p^{1/n}$, and centers the vertices of $\mathcal{H}'$, and because all the points of a lattice are equivalent, we conclude that $\|\boldsymbol{a'}\|_\infty \leq (|\alpha| + |\beta|)\, p^{1/n}$. □

Definition 2 (PMNS) *A modular number system $\mathcal{B} = MNS(p, n, \gamma, \rho)$ which satisfies the conditions of Theorem 1 is called a Polynomial Modular Number System (PMNS). We shall denote $\mathcal{B} = PMNS(p, n, \gamma, \rho, E)$.*

In practice, we shall define the polynomial E with α and β as small as possible.

Example 2 *We define the PMNS with $p = 23$, $n = 3$, $\rho = 2$, $E(X) = X^3 - X + 1$ ($\alpha = -1, \beta = 1$). We easily check that $\gamma = 13$ is a root of E in $\mathbb{Z}_{23}$, and E is irreducible in $\mathbb{Z}[X]$. We represent the elements of $\mathbb{Z}_{23}$ as polynomials of degree at most 2, with coefficients in $\{-1, 0, 1\}$.*

4. PMNS arithmetic

In this section, we propose algorithms for the classical operations in $GF(p)$, namely addition and multiplication modulo p, when the operands are represented in a PMNS. We give solutions for the conversion from a classical number system (binary) to PMNS, and back. For simplicity, we assume that $\rho = 2^k$ and all the operands are represented in $\mathcal{B} = PMNS(p, n, \gamma, 2^k, E)$, with $E = X^n + \alpha X + \beta$.

At this point, it is very important to understand that the value(s) k we must consider, depend on the way we implement the coefficient reduction(s) presented in Section 5. From Theorem 1, we know that every integer $a < p$ can be represented in a PMNS with coefficients $|a_i| \leq (|\alpha| + |\beta|)\, p^{1/n}$, i.e., of size at most

$$\left\lceil \log_2 (|\alpha| + |\beta|) + \frac{1}{n}\log_2(p) \right\rceil \quad \text{bits.} \quad (6)$$

Algorithms designed for common problems in the lattice's world, such as CVP_∞, can be used to reach this bound (see [11] for details). However, they are unpractical from an arithmetic point of view. In Section 5, we propose algorithms which can be seen as approximation algorithms in the case of our specific lattices. For each proposed solution, we determine, k, the number of bits required in order to represent the coefficients of the polynomials in the definition of a PMNS.

4.1. Addition, subtraction

Let a, b be two integers less than p, given in their PMNS representation. We want to compute the sum $s = a + b \bmod p$. Because of the polynomial nature of the PMNS representation, additions can be carried out independently, in parallel, on each coefficients. Let $A, B \in \mathbb{Z}[X]$ be the polynomial representations of a and b respectively. We have $A(\gamma) \equiv a \pmod{p}$, and $B(\gamma) \equiv b \pmod{p}$, and we compute $C = A + B$, such that

$$C(\gamma) \equiv A(\gamma) + B(\gamma) \pmod{p}, \quad (7)$$

or equivalently $c \equiv a + b \pmod{p}$.

Yet, the result of (7) is a polynomial of degree less than n, but whose coefficients c_i can be larger than $\rho = 2^k$. It is clear however that $\|C\|_\infty < 2^{k+1}$. In order to obtain c in a valid PMNS form, we thus need to reduce its coefficients. We propose different approaches to this problem in Section 5.

4.2. Multiplication

As for the addition, we use the polynomial forms of a and b to compute the product $r = ab \bmod p$. The details are presented in Algorithm 1.

Algorithm 1 [PMNS Modular Multiplication]

Input: $A = (a_0, \ldots, a_{n-1})_{\mathcal{B}}$, $B = (b_0, \ldots, b_{n-1})_{\mathcal{B}}$, with $|a_i|, |b_i| < 2^k$

Output: $R = (r_0, ..., r_{n-1})_{\mathcal{B}}$ such that $R(\gamma) \equiv ab \bmod p$, with $|r_i| < 2^k$

1: Polynomial multiplication in $\mathbb{Z}[X]$: $C \leftarrow A \times B$
2: Polynomial reduction: $C' \leftarrow C \bmod E$
3: Coefficient reduction: $R \leftarrow CTCR(C')$

Given $A, B \in \mathbb{Z}[X]$, with $A(\gamma) \equiv a \pmod p$, and $B(\gamma) \equiv b \pmod p$, we first evaluate the product $C = A\,B$. Clearly, we have

$$C(\gamma) \equiv A(\gamma)\,B(\gamma) \pmod p. \tag{8}$$

Since $\deg A, \deg B \leq n-1$, their polynomial product, C, satisfies $\deg C \leq 2n-2$. Step 2 of Algorithm 1 consists in the reduction modulo E, which reduces C to a polynomial of degree less than n. Since $E(\gamma) \equiv 0 \pmod p$, the polynomial $C' = C \bmod E$ corresponds to the same value than C in $\mathcal{B}$. In other words, there exists $K \in \mathbb{Z}[X]$ such that

$$C'(\gamma) = C(\gamma) - K(\gamma)\,E(\gamma) \equiv C(\gamma) \pmod p,$$

with $\deg C' < \deg E = n$. Note that the special form of $E(X) = X^n + \alpha X + \beta$ nicely simplifies the reduction modulo E. As shown in Fig. 1, we first compute the product $C = (c_0, \ldots, c_{2n-2})_{\mathcal{B}}$ (see Fig. 1(a)), and we reduce the terms of order $\geq n$ using the congruence $X^n \equiv -\alpha X - \beta \pmod E$ (see Fig. 1(b)).

If A, B have coefficients such that $|a_i|, |b_i| < 2^k$, then, we can see in Fig. 1(b) that the coefficients of the polynomial C' reduced modulo E, satisfy

$$|c'_i| < n\,(|\alpha| + |\beta|)\,2^{2k}, \quad \forall i = 0, \ldots, n-1, \tag{9}$$

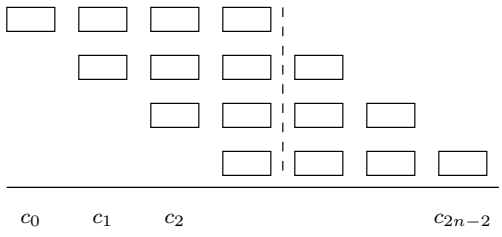
To be a little more precise, we can remark that $|c'_i| < ((n-1)|\alpha| + (n-2)|\beta| + 2)\,2^{2k}$; this upper bound being given by c'_1. As for the addition, the final step in the multiplication algorithm consists in a coefficient reduction, detailed in Section 5.

4.3. Conversions

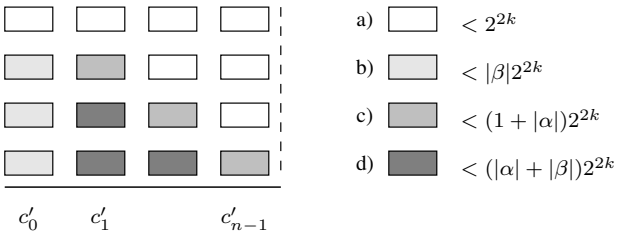
We briefly propose methods for the conversions from binary to PMNS, and from PMNS to binary, that can be easily implemented at low memory cost.

4.3.1 Binary to PMNS

Given the integer, $0 \leq a < p$, we want to define a polynomial A with coefficients less than 2^k, which satisfy



(a) The polynomial $C = AB$



(b) The polynomial $C' = C \bmod E$

Figure 1. Reduction modulo $E(X) = X^n + \alpha X + \beta$

$A(\gamma) \equiv a \pmod p$. We first represent a in radix 2^k as

$$a = \sum_{i=0}^{n-1} d_i\,(2^k)^i, \quad \text{with } 0 \leq d_i < 2^k. \tag{10}$$

Our approach requires the precomputation of n values T_i, corresponding to the polynomial representations of 2^{ki} for $i = 0, \ldots, n-1$. We have

$$T_i(\gamma) \equiv 2^{ki} \pmod p, \quad \text{for } i = 0, \ldots, n-1.$$

We store those polynomials in a reduced form; i.e., with coefficients less than $(|\alpha| + |\beta|)2^k$. Thus, (10) rewrites

$$A = \sum_{i=0}^{n-1} d_i\,T_i, \quad \text{with } 0 \leq d_i < 2^k.$$

The polynomial A satisfies $\|A\|_\infty < n(|\alpha| + |\beta|)2^{2k}$, and can be reduced using one of the coefficient reduction algorithms presented in Section 5.

4.3.2 PMNS to binary

Given the polynomial A, we want to recover the corresponding integer $a = A(\gamma) \pmod p$. In the PMNS representation, we have

$$A(\gamma) = \sum_{i=0}^{n-1} a_i\,\gamma^i.$$

We use n precomputed integers g_i, corresponding to powers of γ:

$$g_i = \gamma^i \bmod p, \quad \text{for } i = 0, \ldots, n-1.$$

Hence, a can be computed by

$$a = \sum_{i=0}^{n-1} a_i \, g_i \bmod p.$$

5. Coefficient reduction

In this section, we propose solutions for the coefficient reduction, based on lookup-tables. As explained in the previous sections, we must be able to reduce the coefficients resulting from the different arithmetic operations presented in Section 4, particularly the polynomial multiplication in Section 4.2.

Let $C \in \mathbb{Z}[X]$ be a polynomial of degree less than n, with coefficients $|c_i| < 2^{k+l}$, where l is the number of bits to reduce. We want to define a polynomial R, such that $R(\gamma) \equiv C(\gamma) \pmod p$, and $\|R\|_\infty < 2^k$. We decompose C into two polynomials, L, and H, such that

$$C = L + 2^{k-1} H, \quad \text{with } |l_i| < 2^{k-1} \text{ and } |h_i| < 2^{l+1}, \tag{11}$$

as shown in Figure 2.

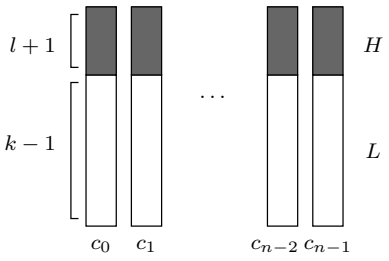


Figure 2. The decomposition of C into its lower part L, and higher part H

A polynomial H', such that $\|H'\|_\infty < 2^{k-1}$, is deduced from one or more lookup tables plus, if necessary, a few number of additions, such that $R = L + H'$ satisfies $\|R\|_\infty \leq 2^k$. In Algorithm 2 below, Step 2 ($H' \leftarrow \texttt{TREAD}[H]$), must be seen as a virtual instruction, allowing many degrees of freedom in the implementation, as we shall see in the next sections.

Algorithm 2 [TCR: Table-based Coefficient Reduction]

Input: $C = (c_0, \ldots, c_{n-1})_{\mathcal{B}}$, with $|c_i| < 2^{k+l}$

Output: $C' = (c'_0, ..., c'_{n-1})_{\mathcal{B}}$ such that $c' \equiv c \pmod p$, and $|c'_i| < 2^k$ for $i = 0, \ldots, n-1$

1: $C = L + 2^{k-1} H$
2: $H' \leftarrow \texttt{TREAD}[H]$
3: $C' \leftarrow L + H'$

The main idea of all the possible variants we shall suggest, is that the precomputed polynomials stored in lookup table(s) have their coefficients as small as possible, i.e., reaching the bound given in Theorem 1. For simplicity in the notations, we define

$$\tau = \left\lceil \log_2 (|\alpha| + |\beta|) + \frac{1}{n} \log_2(p) \right\rceil, \tag{12}$$

as the number of bits required per coefficient of the precomputed polynomials.

If we perform the reduction with a single table lookup, we have $\|H'\|_\infty \leq 2^\tau$, and since we want $\|L + H'\|_\infty \leq 2^k$, choosing $k \geq 1 + \tau$, gives the expected result. However, for large values of l (the number of bits to reduce), this solution becomes unpractical in terms of memory requirements. Actually, it would require a table of $2^{(l+2)n}$ input bits (one extra bit is necessary for the sign of each coefficient), with values on τn bits. As an example, the coefficients of the result of a polynomial multiplication are less than $n\,(|\alpha| + |\beta|)\, 2^{2k}$ (see Section 4.2). It corresponds to $l = k + \log_2(n) + \log_2(|\alpha| + |\beta|)$, which is purely impossible with a single table!

A first, straightforward solution is to reduce a smaller number of bits at once using table(s), and to perform several iterations until we get a complete reduction. Algorithm 3 is used to reduce the coefficients of a polynomial C, with $\|C\|_\infty < 2^{k+t}$, to a polynomial R such that $\|R\|_\infty < 2^k$. It performs several iterations of Algorithm TCR (cf. Algorithm 2), which reduce the coefficients from $k + l$ to k bits using lookup table(s).

Algorithm 3 [CTCR: Complete Table-based Coefficient Reduction]

Input: $C = (c_0, \ldots, c_{n-1})_{\mathcal{B}}$, with $|c_i| < 2^{k+t}$

Output: $R = (r_0, ..., r_{n-1})_{\mathcal{B}}$ such that $r \equiv c \pmod p$, and $|r_i| < 2^k$ for $i = 0, \ldots, n-1$

1: $R \leftarrow C$
2: **for** $i = t - l$ **downto** 0 **by** l **do**
3: $\quad R = \underline{R} + 2^i \overline{R}$ with $\|\underline{R}\|_\infty < 2^i$ and $\|\overline{R}\|_\infty < 2^{k+l}$
4: $\quad R \leftarrow \underline{R} + 2^i \, TCR(\overline{R})$
5: **end for**

Yet, the size of the lookup table, even for small values of

l, can be unpractical. For example, with $n = 6$, $k = 32$, with Algorithm TCR reducing a single bit, i.e., with $l = 1$, we need about $6MB$ of memory! In order to reduce the amount of required memory, we propose different variants which decompose the table into several, smaller tables, at the cost of a few additions for the reconstruction of the lookup value.

5.1. Horizontal Splitting

We decompose H into s, horizontal slices, of height h, such that

$$H = H_0 + 2^h H_1 + \cdots + 2^{(s-1)h} H_{s-1}, \qquad (13)$$

where $h = \lceil (l+1)/s \rceil$, (as shown in Figure 3 in the case $s = 3$).

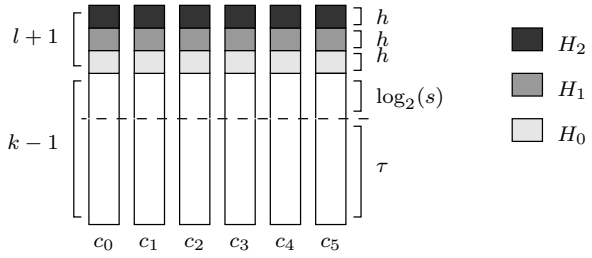


Figure 3. An horizontal decomposition of H, with $n = 6$, and $s = 3$

Using this decomposition, we implement step 2 of Algorithm 2, i.e. the instruction $H' \leftarrow \texttt{TREAD}[H]$, as

$$H' \leftarrow \texttt{TAB}_0[H_0] + \cdots + \texttt{TAB}_{s-1}[H_{s-1}], \qquad (14)$$

where $\texttt{TAB}_i[H_i]$ returns a reduced polynomial (with coefficients on τ bits) equivalent to $2^{k-1+ih} H_i$. Since H' is obtained from the addition of s polynomials, we need $\log_2(s)$ extra bits to compensate the carry generation due to the sum (14); and because we want $\|H'\|_\infty < 2^{k-1}$, we must take $k \geq \lceil \tau + \log_2(s) + 1 \rceil$.

In terms of memory requirements, we remark that, since H_i is a polynomial with coefficients on $h+1$ bits (one extra bit is necessary for the sign of each coefficient), i.e., representing values in $\{-2^h+1, \ldots, 2^h-1\}$, we need s tables, of $(2^{h+1}-1)^n - 1$ input bits each (there is no need to store the null polynomial), with polynomials stored on $n(\tau+1)$ bits (one extra bit is necessary for the sign of each coefficient). Thus, the total cost of the horizontal approach is

$$s[(2^{h+1}-1)^n - 1] \times n(\tau+1) \text{ bits.}$$

For example, with $k = 32$, $n = 6$, and $s = l + 1 = 2$ ($h = 1$), we have $\tau = 30$, and the total memory cost is $33KB$.

5.2. Vertical Splitting

In order to further reduce the size of the tables, we can consider a decomposition of H into r vertical blocs, of width w, such that

$$H = H_0 + H_1 X^w + \cdots + H_{r-1} X^{(r-1)w}, \qquad (15)$$

where $w = \lceil n/r \rceil$, (as shown in Figure 4 in the case $r = 3$).

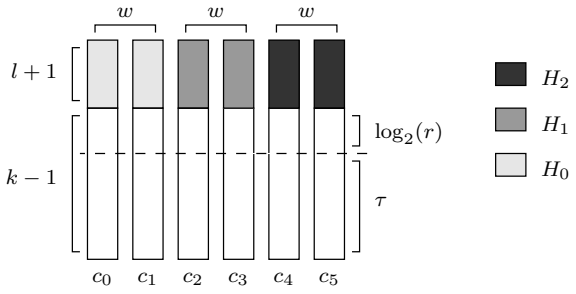


Figure 4. A vertical decomposition of H, with $n = 6$, $r = 3$, and $w = 2$

Thus, step 2 of Algorithm 2, i.e., the virtual instruction, $H' \leftarrow \texttt{TREAD}[H]$, thus rewrites

$$H' \leftarrow \texttt{TAB}_0[H_0] + \cdots + \texttt{TAB}_{r-1}[H_{r-1}], \qquad (16)$$

where $\texttt{TAB}_i[H_i] = 2^{k-1} H_i X^{id}$. Since H' is obtained by the addition of r polynomials with coefficients on τ bits, we need $\log_2(r)$ extra bits to compensate the carry generation due to the sum (16); and because we need $\|H'\|_\infty < 2^{k-1}$, we must take $k \geq \lceil \tau + \log_2(r) + 1 \rceil$.

In terms of size, this approach requires r tables, of $2^{w(l+2)}$ input bits (one extra bit is necessary for the sign of each coefficient), with polynomials stored on $n(\tau+1)$ bits each; a total cost of

$$r\, 2^{w(l+2)} \times n(\tau+1) \text{ bits.}$$

For example, with $k = 32$, $n = 6$, $r = 3$, $w = 2$, and $l = 1$, we have $\tau = 30$, and the total memory cost is less than $4.4KB$.

If further reduction is required, we remark that we can express (16) with only one table, TAB, plus some low-cost operations, as

$$H' \leftarrow \texttt{TAB}[H_0] + (\texttt{TAB}[H_1]\, X^w \bmod E) + \ldots + (\texttt{TAB}[H_{r-1}]\, X^{w(r-1)} \bmod E). \qquad (17)$$

The computation of the terms $(\texttt{TAB}[H_i]\, X^{wi} \bmod E)$, is performed at very low cost, with right shifts for the product, followed by a reduction modulo E. In this case, we still have r polynomials to add up, in order to get H', but

their coefficients have $\log_2(|\alpha| + |\beta|)$ extra bits, because of the reduction modulo E (see Figure 1). Thus, we must take $k \geq \lceil \tau + \log_2(|\alpha| + |\beta|) + \log_2(r) + 1 \rceil$. The memory cost is divided by r, which, using the same set of parameters as above, gives $1.45KB$.

5.3. Combined Vertical-Horizontal Splitting

An ultimate optimization is to consider a combination of the horizontal and vertical splittings. We decompose H in r vertical blocs, such that

$$H = H_0 + H_1 X^w + \cdots + H_{r-1} X^{(r-1)w}, \qquad (18)$$

where $w = \lceil n/r \rceil$, and we split each bloc H_i, into s horizontal slices, yielding

$$H_i = H_{i,0} + 2^h H_{i,1} + \cdots + 2^{(s-1)h} H_{i,s-1}, \qquad (19)$$

where $h = \lceil (l+1)/s \rceil$. In this case, the virtual instruction, $H' \leftarrow \texttt{TREAD}[H]$, rewrites

$$H' \leftarrow \sum_{i=0}^{r-1} \sum_{j=0}^{s-1} \texttt{TAB}_{\texttt{i},\texttt{j}}[H_{i,j}]. \qquad (20)$$

Since we add up rs reduced polynomials, we must take $k \geq \lceil \tau + \log_2(rs) + 1 \rceil$. We have rs small tables, with 2^{hw+1} input bits each (one extra bit is required for the sign of each coefficient), with values on $n(\tau+1)$ bits. Using the parameters ($k = 32$, $n = 6$, $r = 3$, $w = 2$, $s = 2$, $h = 1$), we have $\tau = 29$, and a total memory size of about $1KB$. Further reduction is possible if we consider the vertical optimization (see (17)). We divide the size by r, but, again, the number of extra bits per coefficient increases by $\log_2(|\alpha| + |\beta|)$.

6. Example

In this section, we propose an example to illustrate some of the algorithms proposed in the previous sections. Given a prime number p, an integer, $n > 4$, and a polynomial, E, we first define a Polynmial Modular Number System, $\mathcal{B} = PMNS(p, n, \gamma, \rho, E)$, where $\rho = 2^k$ is deduced from the bound given for the horizontal splitting approach. Then, we convert two integers a, b into $\mathcal{B}$, and we perform the multiplication $a\,b \bmod p$, using the PMNS modular multiplication presented in Algorithm 1. For the coefficient reduction, we use Algorithm CTCR (cf. Algorithm 3), with $l = 1$, and with TCR implemented using the horizontal splitting variant, presented in Section 5.1, with parameters $s = 2$, and $h = 1$ (two slices of height 1 bit).

Definition of the PMNS: Let us define $p = 123456789120001$. We have $|p| = \lceil \log_2(p) \rceil = 47$ bits. We also define $E(X) = X^4 + 1$ ($n = 4, \alpha = 0, \beta = 1$). E is irreducible in $\mathbb{Z}$, and $\gamma = 46988594033438$ is a root of E modulo p. Since we shall use the horizontal splitting, with $s = 2$, we define

$$k = \left\lceil \log_2(|\alpha| + |\beta|) + \frac{1}{n}\log_2(p) + 2 \right\rceil = 14.$$

Binary to PMNS: Let $a = 111111111111111, b = 22222222222222$. The polynomials A, B, below, are given by Algorithm 3 with inputs $(a, 0, 0, 0)$ and $(b, 0, 0, 0)$ respectively. We have

$$A = 4466 + 6362\,X - 6906\,X^2 - 2934\,X^3$$
$$B = -2835 - 1844\,X - 2252\,X^2 - 7482\,X^3$$

It is easy to check that $a = A(\gamma) \bmod p$, and $b = B(\gamma) \bmod p$.

Modular multiplication: We use Algorithm 1 for the multiplication modulo E, and Algorithm 3, with the horizontal splitting variant, for the coefficient reduction. We have

$$C(X) = A(X)\,B(X) = -12661110 - 26271574\,X - 2210450\,X^2 - 26689282\,X^3 - 26637876\,X^4 + 58278060\,X^5 + 21952188\,X^6,$$

and

$$C'(X) = C(X) \bmod E(X) = 13976766 - 84549634\,X - 24162638\,X^2 - 26689282\,X^3.$$

The iterations of Algorithm 3 are given in Table 2. We remark that the coefficients of $R = R_0$ satisfy $|r_i| < 2^k = 2^{14} = 16384$.

PMNS to binary: Finally, we can convert R in binary. We obtain $r = 76459417066083$, and we easily check that this is the correct result $r = a\,b \bmod p$.

7. Conclusions

In this paper, we have proposed a new representation for the elements of $\mathbb{Z}_p$, the ring of integers modulo p, called Polynomial Modular Number Systems. In this system, integers are represented as polynomials in γ, of degree less than n, with coefficients bounded by $(|\alpha| + |\beta|)p^{1/n}$, where α, β are very small integers. Since $p^{1/n}$ is a minimum value,

R_i	$r_{i,0}$	$r_{i,1}$	$r_{i,2}$	$r_{i,3}$
R_{13}	13976766	-84549634	-24162638	-26689282
R_{12}	14488766	-24305666	-20345166	-32534274
R_{11}	13759678	-9254914	-620878	-17989378
R_{10}	4661438	-2222082	1705650	-1809154
R_9	1237182	-2060802	1175730	-1774850
R_8	1237182	-2060802	1175730	-1774850
R_7	323390	-895874	-54222	-975362
R_6	247870	-310274	-70670	-395842
R_5	210110	-17474	-78894	-106082
R_4	103102	-12434	-95454	-105010
R_3	46214	-4626	-24166	-55938
R_2	7958	-6282	-26850	-22402
R_1	7130	-7624	-10082	-3274
R_0	6095	-7557	-3394	-3589

Table 2. The iterations performed by the CTCR Algorithm 3

only a few extra bits are required for each coefficient. Compared to the classic multiprecision representation, the polynomial nature of PMNS allows for no-carry propagation, and efficient polynomials arithmetic. The algorithms presented in this paper for the arithmetic operations must be seen as a first step in doing the arithmetic over this new representation. Many improvements are still to come...

Acknowledgements

This work was done during L. Imbert leave of absence at the university of Calgary, with the *Centre for Information Security and Cryptography (CISaC)* and the *Advanced Technology Information Processing Systems (ATIPS)* Labaratory. It was also partly supported by the French ministry of education and research under the grant *ACI Sécurité Informatique 2003, Opérateurs Cryptographiques et Arithmétique Matérielle (OCAM)*.

References

[1] J.-C. Bajard, L. Imbert, and T. Plantard. Arithmetic operations in the polynomial modular number system. Research Report 04030, LIRMM – CNRS, 161 rue Ada, 34392 Montpellier cedex 5, France, Sept. 2004. Available electronicaly at `http://www.lirmm.fr/~imbert`.

[2] J.-C. Bajard, L. Imbert, and T. Plantard. Modular number systems: Beyond the Mersenne family. In *Selected Areas in Cryptography: 11th International Workshop, SAC 2004*, volume 3357 of *LNCS*, pages 159–169. Springer-Verlag, Jan. 2005.

[3] P. Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In A. M. Odlyzko, editor, *Advances in Cryptology – CRYPTO '86*, volume 263 of *LNCS*, pages 311–326. Springer-Verlag, 1986.

[4] J. Chung and A. Hasan. More generalized mersenne numbers. In M. Matsui and R. Zuccherato, editors, *Selected Areas in Cryptography – SAC 2003*, volume 3006 of *LNCS*, pages 335–347. Springer-Verlag, 2004.

[5] R. Crandall. Method and apparatus for public key exchange in a cryptographic system. U.S. Patent number 5159632, 1992.

[6] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[7] C. Dwork. Lattices and their applications to cryptography. Lecture notes, Stanford University, 1998.

[8] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag, 2004.

[9] L. Lovász. *An Algorithmic Theory of Numbers, Graphs and Convexity*, volume 50 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM Publications, 1986.

[10] A. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC Press, 1997.

[11] D. Micciancio and S. Goldwasser. *Complexity of Lattice Problems, A Cryptographic Perspective*. Kluwer Academic Publishers, 2002.

[12] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, Apr. 1985.

[13] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[14] J. Solinas. Generalized mersenne numbers. Research Report CORR-99-39, Center for Applied Cryptographic Research, University of Waterloo, Waterloo, ON, Canada, 1999.