HAL
open science

An Alternative Approach to Polynomial Modular Number System Internal Reduction

Nicolas Méloni

To cite this version:

Nicolas Méloni. An Alternative Approach to Polynomial Modular Number System Internal Reduction. 2022. hal-03635347v1

HAL Id: hal-03635347

https://univ-tln.hal.science/hal-03635347v1

Preprint submitted on 8 Apr 2022 (v1), last revised 8 Jun 2022 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A New Approach to Polynomial Modular Number System Internal Reduction

Nicolas Méloni
Université de Toulon
Institut de Mathématiques de Toulon
Toulon, France
nicolas.meloni@univ-tln.fr

***Abstract*—The Polynomial Modular Number System (PMNS) is an alternative to the binary multi-precision representation that allows to transport the arithmetic of a finite field to a polynomial ring. The most important operation in that system is the internal reduction that follows any arithmetic operation. All recent works on the subject use the same algorithm derived from Montgomery's modular multiplications to perform this internal reduction. This paper designs and analyzes two new algorithms to perform the internal reduction, both based on Babai's Closest Vector algorithms. It allows to significantly reduce the number of additions needed to perform this operation. A comprehensive experimental analysis shows that one of those algorithms is also faster in practice. For that matter, a C code generation tool has been developed in order to produce implementations for any prime number field.**

***Index Terms*—Finite field arithmetic, Polynomial Modular Number System, Integer Lattice.**

I. Introduction

Field arithmetic is a key component of many modern cryptosystems. The implementation of prime field in particular has received a lot of attention for the past decades. The most common way to represent a field element is to use a 2^w-ary representation. In such a positional number system, given a prime p, an element a of $\mathbb{Z}/p\mathbb{Z}$ is represented as a $n = \lceil \log(p) \rceil + 1$ array $(a_0, \ldots, a_{n-1})$ corresponding to the integer

$$a = \sum_{i=0}^{n-1} a_i 2^{wi}$$

with $0 \leq a_i \leq 2^w$ or $-2^{w-1} \leq a_i \leq 2^{w-1} - 1$. All arithmetic operations are then performed modulo p. Among those operations modular multiplication has become the main focus of investigation as most cryptosystems heavily rely on it. It is usually performed as a standard integer multiplication followed by a modular reduction. Many of approaches have been proposed to speed up modular reduction modulo a prime number. Two main cases are usually considered, depending on whether the prime can be freely chosen or not. A typical example of the former case is that of a Mersenne prime, that is a prime number of the form $2^k - 1$. In that case a modular reduction can be easily computed as a shift-and-add operation. Generalizations of Mersenne primes have been proposed [7], [15] in order to extend the range of possible candidates. Those methods usually provide extremely fast modular reductions but are limited to a small number of primes. In the latter case, when the cryptosystem does not allow enough freedom to chose a generalized Mersenne number, more generic algorithms must be used [5], [13].

The Polynomial Modular Number System (PMNS) is an alternative to the standard radix 2^w multi-precision representation. It was introduced by Bajard, Imbert and Plantard [2] in order to speed up modular arithmetic when special primes are not available. A field element is represented as a polynomial of bounded degree and field arithmetic is performed using polynomial arithmetic. An element a of $\mathbb{Z}/p\mathbb{Z}$ is represented as a polynomial

$$A(X) = \sum_{i=0}^{n-1} a_i X^i \mod E(X)$$

where E is a degree n polynomial and $|a_i|\rho$ for some ρ. One key point is that, even if p has no special form, the polynomial E can in general be chosen so that polynomial reduction modulo E is fast (for instance if $E = X^n - 1$). In that case, the modular reduction that occurs in positional number system is replaced by a fast polynomial reduction (usually called external reduction) followed by a coefficient reduction (called internal reduction). The latter operation is the critical point. Two approaches have been proposed to perform it based either on Barrett modular reduction algorithm [3] or Montgomery's one [14]. During the past few years improvements have been made on the implementation [10], generation [4], [8], randomization [9] and generalization [6], [11] of PMNS in various contexts. One interesting common feature between all those works is that they all perform the internal reduction step using the Montgomery-like approach.

In this paper we propose two new approaches to perform the internal reduction based on two algorithms initially proposed by Babai to solve the Closest Vector Problem for integer lattices [1]. Those two algorithms have been mentioned before in the literature in order to obtain theoretical bounds on the size of the coefficients [4] but discarded as a practical tools due their high computational complexities [9]. We show that it is actually possible to design competitive algorithms based on Babai's ideas achieving faster internal reduction in many cases.

The rest of the paper is structured as follows. Section II is a review of PMNS and Babai's Closest Vector algorithms. In Section III we propose and study two new internal reduction algorithms and two optimized versions of those. In Section IV we give implementation details about our code generation tool targeting 64-bit architectures and we illustrate the efficiency of our approach with experimental results and comparisons with a similar tool based on Montgomery reduction [10].

II. Mathematical background

A. Lattices

A lattice is a sub-group of $\mathbb{R}^n$. It can be seen as the set of all linear combinations with integer coefficients of any set of linearly independent vectors. Typically, a lattice can be seen as the set

$$\Lambda = \left\{ \sum_{i=0}^{n-1} a_i \mathbf{b}_i \mid a_i \in \mathbb{Z} \right\}$$

where $(\mathbf{b}_0, \ldots, \mathbf{b}_{n-1})$ is a basis of $\mathbb{R}^n$.

Volume of a lattice.: Any two basis of a lattice Λ can be obtained from one another using a integral change-of-basis matrix of determinant ± 1. Thus the absolute value of the determinant of any basis only depends on Λ. That determinant is called the *volume* of the lattice and is noted $\det(\Lambda)$.

Gram-Schmidt orthogonalization.: Let $\mathbf{B} = (\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_{n-1})$ a basis. Its *Gram-Schmidt orthogonalization* (GSO) $\tilde{\mathbf{B}} = (\tilde{\mathbf{b}}_0, \tilde{\mathbf{b}}_1, \ldots, \tilde{\mathbf{b}}_{n-1})$ is the orthogonal family obtained using the recursive process:

1) $\tilde{\mathbf{b}}_0 = \mathbf{b}_0$
2) $\forall i \geq 1, \tilde{\mathbf{b}}_i = \mathbf{b}_i - \sum_{j=0}^{i-1} \mu_{ij} \tilde{\mathbf{b}}_\mathbf{j}$

where $\mu_{ij} = \frac{\langle \mathbf{b}_i, \tilde{\mathbf{b}}_j \rangle}{\|\tilde{\mathbf{b}}_j\|_2^2}$. Let Λ be the lattice spanned by $\mathbf{B}$, a classical result is that it satisfies

$$\det(\Lambda) = \prod_{i=0}^{n-1} \|\tilde{\mathbf{b}}_i\|_2.$$

LLL algorithm.: The *LLL* algorithm [12] is a lattice basis reduction process that computes an *LLL*-reduced basis in polynomial time. For a given basis $\mathbf{B}$ and a real number $\frac{1}{4} < \delta < 1$, $\mathbf{B}$ is *LLL*-reduced if

- for $0 \leq j \leq i \leq n-1 : |\mu_{ij}| \leq 0.5$
- for $1 \leq i \leq n-1 : \delta \|\tilde{\mathbf{b}}_{i-1}\|_2^2 \leq \|\tilde{\mathbf{b}}_i\|_2^2 + \mu_{i,i-1} \|\tilde{\mathbf{b}}_{i-1}\|_2^2$.

In practice it means that the basis has short vectors and that they are nearly orthogonal.

B. PMNS

Let p be a prime number, E a polynomial of degree n and γ a root of E modulo p. The general idea is to represent integers as elements of the polynomial quotient ring $\mathbb{Z}[X]/(E)$. Any arithmetic operation can then be performed using standard arithmetic over the polynomials followed by a euclidean division by E.

Definition 1: A Polynomial Modular Number System (PMNS) $\mathcal{B}$ is defined by a tuple $(p, n, \gamma, \rho, E, l) \in \mathbb{Z}^4 \times \mathbb{Z}[X] \times \mathbb{Z} \cup \{\infty\}$, such that $E(\gamma) \equiv 0 \mod p$ and for every integer $0 \leq x < p$, there exists a polynomial $V(X)$ in $\mathbb{Z}[X]/(E)$ (or equivalently a vector $V = (v_0, ..., v_{n-1}) \in \mathbb{Z}^n$) such that:

$$V(\gamma) \equiv \sum_{i=0}^{n-1} v_i \gamma^i \equiv x \mod p \text{ and } \|V\|_l < \rho,$$

where $\|.\|_l$ is the l-norm of $\mathbb{R}^n$.

Such a polynomial (or vector) is said to be a representative of x and we notate $V \equiv_\mathcal{B} x$. Their might be several representatives for a given x. If V' is another polynomial such that $V' \equiv_\mathcal{B} x$ we say that V and V' are equivalent an notate $V \equiv_\mathcal{B} V'$.

Remark 1: In all previous work, the only norm considered was the infinity norm. In other words the parameter l was fixed to ∞ so that $\|V\|_l = \|V\|_\infty = \max(|v_i|)$. However in this paper the 1 and 2-norm will also be considered.

Remark 2: In [2] and [3], a naming distinction is made based on the shape of the polynomial E. The name PMNS is reserved to the case where $E(X) = X^n - \alpha X - \lambda$. When $\alpha = 0$ the system is called Adapted Modular Number System (AMNS) and simply MNS otherwise. We make no such distinction in this paper and only use the concept of PMNS.

Example 1: Let $p = 1048573, n = 5, \gamma = 238019, \rho = 36, E = X^5 - 2$ then $\mathcal{B} = (p, n, \gamma, \rho, E, \infty)$ is a PMNS. Let us consider two polynomials $A = -9X^4 + 13X^3 - 35X^2 - 25$ and $B = 24X^4 - 32X^3 + 21X^2 - 2X - 16$, their modular multiplication in PMNS is performed in 3 steps:

1) a standard polynomial multiplication

$$\begin{aligned} AB = &-216X^8 + 600X^7 - 1445X^6 + 1411X^5 \\ &- 1217X^4 + 662X^3 + 35X^2 + 50X + 400 \end{aligned}$$

2) a degree reduction (external reduction)

$$\begin{aligned} C =& AB \mod E \\ =& -1217X^4 + 230X^3 + 1235X^2 - 2840X + 3222 \end{aligned}$$

3) a coefficient reduction (internal reduction) to ensure the coefficients (c'_i) of the results satisfy the bound $|c'_i| < \rho$

$$C \equiv_\mathcal{B} C' = 3X^4 + 2X^3 + 8X^2 - X - 1.$$

One can verify that $A(\gamma)B(\gamma) \equiv C(\gamma) \equiv C'(\gamma) \mod p$. The first two steps are usually performed using standard algorithms and often merge as one operation. The third step is the most difficult from a computational point of view. The general method is to find a polynomial R, close to C (depending on the chosen norm), such that $R(\gamma) \equiv 0 \mod p$, in which case $C' = C - R \equiv_\mathcal{B} C$ is and $\|C'\|$ is supposed to be small. This problem can be interpreted as finding a close vector to the vector C in some integer lattice as shown next.

Definition 2: Let $\mathcal{B} = (p, n, \gamma, \rho, E, l)$ be a PMNS. The lattice $\mathcal{L}(\mathcal{B})$ associated to $\mathcal{B}$ is the set

$$\mathcal{L}(\mathcal{B}) = \left\{ (v_0, \ldots, v_{n-1}) \in \mathbb{Z}^n : \sum_{i=0}^{n-1} v_i \gamma^i \equiv 0 \mod p \right\}.$$

$\mathcal{L}(\mathcal{B})$ can also be seen as the set of all polynomials $V \in \mathbb{Z}[X]$ of degree less than n such that $V(\gamma) \equiv 0 \mod p$. This set is introduced in [2] and it is proven that it is a lattice of dimension n, generated by the row of the matrix

$$G = \begin{pmatrix} p & 0 & 0 & \dots & 0 \\ -\gamma & 1 & 0 & \dots & 0 \\ -\gamma^2 & 0 & 1 & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ -\gamma^{n-2} & 0 & \dots & 1 & 0 \\ -\gamma^{n-1} & 0 & \dots & 0 & 1 \end{pmatrix}.$$

Let $(\mathbf{b}_0, \dots, \mathbf{b}_{n-1})$ be any basis of $\mathcal{L}(\mathcal{B})$ and $C = (c_0, \dots, c_{n-1}) \in \mathbb{Z}^n$. The coefficient reduction problem consists in finding a close vector to C that is a linear combination of the $\mathbf{b}_i$'s. We define the PMNS reduction problem as follow.

Definition 3: **PMNS reduction problem**

Given a PMNS $\mathcal{B} = (p, n, \gamma, \rho, E, l)$ and a vector $v \in \mathbb{Z}^n$, find s in $\mathcal{L}(\mathcal{B})$ such that

$$\|v - s\|_l < \rho.$$

This is obviously linked to the Closest Vector Problem (CVP) that is known to be NP-hard for the ∞-norm [17]. Thankfully we have to deal with a much weaker version of that problem: n is small in practice and we only look for a *close enough* vector to v.

C. Montgomery PMNS Reduction

The most efficient method to solve the *PMNS reduction problem* is a Montgomery-like algorithm proposed by Nègre and Plantard [14]. It actually solves a slightly different version of the problem and requires the existence of a polynomial M with certain properties as studied in [10]. The method is summarized by Algorithm 1.

Algorithm 1: RedCoeff ([14])

Data: $\mathcal{B} = (p, n, \gamma, \rho, E, \infty), V \in \mathbb{Z}_n[X], M \in \mathcal{B}$ such that, $M(\gamma) \equiv 0 \mod p$, $\phi \in \mathbb{N} - \{0\}$ and $M' = -M^{-1} \mod (E, \phi)$.
Result: $S(\gamma) = V(\gamma)\phi^{-1} \mod p$
$Q \leftarrow V \times M' \mod (E, \phi)$;
$T \leftarrow Q \times M \mod E$;
$S \leftarrow (V + T)/\phi$;
return S;

D. Babai's Nearest Plane algorithm

Let $\mathbf{B} = (\mathbf{b}_0, \dots, \mathbf{b}_{n-1})$ a lattice basis and $\tilde{\mathbf{B}}$ its GSO. Recall that $\tilde{\mathbf{B}}$ satisfies

$$\tilde{\mathbf{b}}_i \perp \mathbf{span}(\mathbf{b}_0, \dots, \mathbf{b}_{i-1})$$

and

$$\mathbf{span}(\tilde{\mathbf{b}}_0, \dots, \tilde{\mathbf{b}}_i) = \mathbf{span}(\mathbf{b}_0, \dots, \mathbf{b}_i)$$

Babai's nearest hyperplane algorithm [1] can be seen as a way to reduce each coordinate of v in the Gram-Schmidt basis one by one, starting from coefficient $n-1$ to 0.

Suppose we have a semi-reduced vector v' whose coefficient $i+1$ to $n-1$ are reduced in the Gram-Schmidt basis. Reducing coefficient i is done by computing the integer c_i such that $v' - c_i\mathbf{b}_i$ has the smallest i-th coefficient possible in the Gram-Schmidt basis. In other terms c_i must minimize

$$< v', \tilde{\mathbf{b}}_i > - c_i < \mathbf{b}_i, \tilde{\mathbf{b}}_i >$$

which leads to

$$c_i = \left\lfloor \frac{< v', \tilde{\mathbf{b}}_i >}{\|\tilde{\mathbf{b}}_i\|_2^2} \right\rceil.$$

Babai's Nearest Plane algorithm is then straightforward as shown by Algorithm 2.

Algorithm 2: Babai's Nearest Plane algorithm

Data: $\mathcal{L}$ a lattice, $v \in \mathbb{Z}^n$, $B = (\mathbf{b}_0, \dots, \mathbf{b}_{n-1})$ a basis and $\tilde{B} = (\tilde{\mathbf{b}}_0, \tilde{\mathbf{b}}_1, \dots, \tilde{\mathbf{b}}_{n-1})$ its Gram-Schmidt orthogonalization
Result: $s \in \mathbb{Z}^n$ such that $v' = v - s \in \mathcal{L}$ and v' is close to v
$s \leftarrow v$;
for $i = n-1 \dots 0$ **do**
  $c \leftarrow \left\lfloor \frac{<s, \tilde{\mathbf{b}}_i>}{\|\tilde{\mathbf{b}}_i\|_2^2} \right\rceil$;
  $s \leftarrow s - c \times \mathbf{b}_i$
return s;

The output vector s satisfies

$$s = \sum_{i=0}^{n-1} s_i \tilde{\mathbf{b}}_i \text{ with } |s_i| \leq 1/2$$

from which we can deduce that

$$\|s\|_2 \leq \frac{1}{2}\sqrt{\sum_{i=0}^{n-1} \|\tilde{\mathbf{b}}_i\|_2^2}.$$

That last inequality gives, for a given PMNS, a maximum bound for ρ that depends on the basis of $\mathcal{L}(\mathcal{B})$. In particular, the smaller $\sum_{i=0}^{n-1} \|\tilde{\mathbf{b}}_i\|_2^2$ is , the better. As $\det(\mathcal{L}) = \prod \|\tilde{\mathbf{b}}_i\| = p$ is a constant of the lattice, a *good* basis should be one whose vectors satisfies $\|\mathbf{b_i}\| \simeq p^{1/n}$. This is typically the kind of basis that the *LLL* algorithm provides when the dimension n is small.

E. Babai's Rounding algorithm

Let $\mathbf{B} = (\mathbf{b}_0, \dots, \mathbf{b}_{n-1})$ a lattice basis and v a vector. Babai's Rounding algorithm [1] consists of computing the rational coordinates of v in $\mathbf{B}$, so that $v = q_0\mathbf{b}_0 + \dots + q_{n-1}\mathbf{b}_{n-1}$, and choose $c = \lfloor q_0 \rceil \mathbf{b}_0 + \dots + \lfloor q_{n-1} \rceil \mathbf{b}_{n-1}$ as a close vector. The rational coordinates can be obtained by precomputing the inverse matrix $\mathbf{B}^{-1} = (\mathbf{b'}_0, \dots, \mathbf{b'}_{n-1})$ so that

$c = v\mathbf{B}^{-1}$. The whole procedure can be reduced to a simple equation:

$$s = v - \lfloor v\mathbf{B}^{-1} \rceil \mathbf{B}.$$

Babai's Rounding algorithm is described in Algorithm 3.

Algorithm 3: Babai's Rounding algorithm

Data: $\mathcal{L}$ a lattice, $v \in \mathbb{Z}^n$, $\mathbf{B} = (\mathbf{b}_0, \ldots, \mathbf{b}_{n-1})$ a basis and $\mathbf{B^{-1}} = (\mathbf{b'}_0, \mathbf{b'}_1, \ldots, \mathbf{b'}_{n-1})$ its inverse
Result: $s \in \mathbb{Z}^n$ such that $v' = v - s \in \mathcal{L}$ and v' is close to v

```
s ← v;
for i = 0...n−1 do
    r ← ⌊< v, b'_i >⌉;
    s ← s − r b_i;
return s;
```

The output vector $s = (q_0 - \lfloor q_0 \rceil)\mathbf{b}_0 + \cdots + (q_{n-1} - \lfloor q_{n-1} \rceil)\mathbf{b}_{n-1}$ satisfies

$$s = \sum_{i=0}^{n-1} s_i \mathbf{b}_i \text{ with } |s_i| \leq 1/2$$

from which we can deduce that

$$\|s\|_1 \leq \frac{1}{2} \sum_{i=0}^{n-1} \|\mathbf{b}_i\|_1.$$

The last inequality gives, for a given PMNS, a maximum bound for ρ that depends on the basis of $\mathcal{L}(\mathcal{B})$. In particular, the smaller $\sum_{i=0}^{n-1} \|\mathbf{b}_i\|_1$ is , the better.

III. Babai PMNS reduction algorithms

Babai's CVP algorithms can not be used as such to perform the internal reduction of a modular reduction because of the size of their operands. The coefficients of the GSO and the inverse matrix are fractional numbers whose numerators and denominators can be as large as the prime number p. This section describes two algorithms based on Babai's Closet Vector algorithms adapted to the PMNS reduction problem.

A. Nearest Plane PMNS internal reduction algorithm

At its core, Algorithm 2 consists of computing a good integer approximation of $\frac{<s, \tilde{\mathbf{b}}_i>}{\|\tilde{\mathbf{b}}_i\|_2^2}$. Our goal is to compute a good enough approximation of these quantities using coefficients that fit in the registers of a target processor.

We first remark that $\left\| \frac{\tilde{\mathbf{b}}_i}{\|\tilde{\mathbf{b}}_i\|_2^2} \right\|_2 = \|\tilde{\mathbf{b}}_i\|_2^{-1}$. If the basis $\mathbf{B}$ is LLL-reduced, it is expected that, for all i, $\|\tilde{\mathbf{b}}_i\|_2 \simeq p^{1/n}$. This justifies that there should be an integer h_1 so that $\left\lfloor \frac{2^{h_1} \tilde{\mathbf{b}}_{i,j}}{\|\tilde{\mathbf{b}}_i\|^2} \right\rceil$ fit in a w-bit register of all i, j.

Let us suppose that there are two integers h_1 and h_2 such that $\left\lfloor \frac{2^{h_1} \tilde{\mathbf{b}}_{i,j}}{\|\tilde{\mathbf{b}}_i\|^2} \right\rceil$ and $\left\lfloor \frac{s_j}{2^{h_2}} \right\rceil$ are w-bit integers for all i and j. In that case

$$\begin{aligned} < s, \frac{\tilde{b}_i}{\|\tilde{b}_i\|^2} > &= \sum_j s_j \frac{\tilde{b}_{i,j}}{\|\tilde{b}_i\|^2} \\ &= \frac{1}{2^{h_1 - h_2}} \sum_j \frac{s_j}{2^{h_2}} \times \frac{2^{h_1} \tilde{b}_{i,j}}{\|\tilde{b}_i\|^2} \\ &\simeq \left\lfloor \frac{\sum_j \left\lfloor \frac{s_j}{2^{h_2}} \right\rceil \times \left\lfloor \frac{2^{h_1} \tilde{b}_{i,j}}{\|\tilde{b}_i\|^2} \right\rceil}{2^{h_1 - h_2}} \right\rceil \end{aligned}$$

In other words, given h_1 and h_2, it is possible to approximate the coefficients in Babai's Nearest Plane algorithm through register-size integer arithmetic. Algorithm 4 describe a complete reduction procedure.

Algorithm 4: Nearest Plane Reduction Algorithm

Data: $\mathcal{L}$ a lattice defined by a basis $(\mathbf{b}_0, \ldots, \mathbf{b}_{n-1})$, $(\tilde{\mathbf{b}}_0, \ldots, \tilde{\mathbf{b}}_{n-1})$ its GSO, $G_{i,j} = \left\lfloor \frac{2^{h_1} \tilde{\mathbf{b}}_{i,j}}{\|\tilde{\mathbf{b}}_i\|^2} \right\rceil$ for $0 \leq i, j < n$,
Input: $v \in \mathbb{Z}^n$
Result: s equivalent to v with "*small*" coefficients

```
s ← v;
for i = n−1...0 do
    c ← 0;
    for j = 0...n−1 do
        r ← s_j ≫ h2;
        c ← c + r × G_ij
    r ← c ≫ (h_1 − h_2);
    s ← s − r × b_i;
return s;
```

The output $s = (s_0, \ldots, s_{n-1})$ satisfies

$$s = \sum_{i=0}^{n-1} s_i \tilde{\mathbf{b}}_i$$

with each s_i satisfying $|s_i| \leq e_i$ for some e_i depending on the approximation error made during the computation of c.

Complexity analysis: One can remark that $\left\lfloor \frac{2^{h_1} \tilde{\mathbf{b}}_{i,j}}{\|\tilde{\mathbf{b}}_i\|^2} \right\rceil$ does not depend on the input vector v and thus can be pre-computed for all i, j. Let $\mathbf{add}_w$ and $\mathbf{mult}_w$ represent the cost a one addition and one multiplication of two w-bit integers and **shft** that of a shift. For each iteration of the main loop, the inner loop computes $n\,\mathbf{shft}$, $n\,\mathbf{mult}_w$ and $(n-1)\,\mathbf{add}_{2w} = (2n-2)\,\mathbf{add}_w$. It is followed by a addition **shft** and $n\,\mathbf{mult}_w$ and $n\,\mathbf{add}_{2w}$. The total computational cost is

$$2n^2\,\mathbf{mult}_w + (4n^2 - 2n)\,\mathbf{add}_w + (n^2 + n)\,\mathbf{shft}\,.$$

It is possible to save a few additions from the observation that the last vector subtraction of the last iteration ($s \leftarrow s - r \times \mathbf{b_0}$) produces a small vector, that is one with coefficients fitting in one register. It means that the upper part of each subtraction can be discarded, saving $n\,\mathbf{add}_w$. The complexity of Algorithm 4 thus becomes

$$2n^2\,\mathbf{mult}_w + (4n^2-3n)\,\mathbf{add}_w + (n^2+n)\,\mathbf{shft}\,.$$

Coefficient reduction: We need to prove that Algorithm 4 efficiently solves the PMNS reduction problem. To do so, we must evaluate the error made during the various approximations.

Lemma 1: Let $s^{(n-1)}$ be the initial value of variable s and $s^{(n-k-1)}$ its value at the beginning of the k-th iteration of Algorithm 4. Let $S = \sum_{i=0}^{n-1} S_i \tilde{\mathbf{b}}_i$ be the output of Algorithm 4. Then for all i there exists a constant K_i that does not depend on the input v such that

$$|S_i| \leq e_i = \frac{\|s^{(i)}\|_1}{2^{h_1+1}} + K_i.$$

Proof: First, we know that $\tilde{\mathbf{b}}_i \perp \mathbf{span}(\mathbf{b}_0, \ldots, \mathbf{b}_{i-1})$. It means that, instruction the $s \leftarrow s - r \times \mathbf{b}_i$ do not modify the value of S_k for any $k > i$. In other terms, S_i only depends on the value of the vector $s^{(i)}$.

Set

$$R_i = \left\lfloor \frac{\sum_j \left\lfloor \frac{s_j^{(i)}}{2^{h_2}} \right\rfloor \times \left\lfloor \frac{2^{h_1}\tilde{\mathbf{b}}_{ij}}{\|\tilde{\mathbf{b}}_i\|^2} \right\rceil}{2^{h_1-h_2}} \right\rceil.$$

Let e_i be the approximation error:

$$e_i = \left| \sum_j s_j^{(i)} \frac{\tilde{\mathbf{b}}_{i,j}}{\|\tilde{\mathbf{b}}_i\|^2} - R_i \right|.$$

We have

$$\begin{aligned}
R_i &= \frac{1}{2^{h_1-h_2}} \sum_j \left(\frac{s_j^{(i)}}{2^{h_2}} - \varepsilon_j \right) \left(\frac{2^{h_1}\tilde{\mathbf{b}}_{ij}}{\|\tilde{b}_i\|^2} - \varepsilon'_j \right) + \varepsilon \\
&= \frac{1}{2^{h_1-h_2}} \sum_j \left(\frac{s_j^{(i)}}{2^{h_2}} \times \frac{2^{h_1}\tilde{\mathbf{b}}_{ij}}{\|\tilde{\mathbf{b}}_i\|^2} \right) \\
&\quad - \frac{1}{2^{h_1-h_2}} \sum_j \left(\varepsilon_j \times \frac{2^{h_1}\tilde{\mathbf{b}}_{ij}}{\|\tilde{\mathbf{b}}_i\|^2} \right) \\
&\quad - \frac{1}{2^{h_1-h_2}} \sum_j \left(\varepsilon'_j \times \frac{s_j^{(i)}}{2^{h_2}} \right) \\
&\quad + \frac{1}{2^{h_1-h_2}} \sum_j \varepsilon_j \varepsilon'_j + \varepsilon
\end{aligned}$$

with $0 \leq \varepsilon, \varepsilon_j < 1$ and $-1/2 \leq \varepsilon'_j < 1/2$. Remark that

$$\begin{aligned}
\left| \sum_j \left(\varepsilon'_j \times \frac{s_j^{(i)}}{2^{h_2}} \right) \right| &\leq \frac{\sum_j |s_j^{(i)}|}{2^{h_2+1}} \\
&\leq \frac{\|s^{(i)}\|_1}{2^{h_2+1}}
\end{aligned}$$

and

$$\left| \sum_j \varepsilon_j \varepsilon'_j \right| \leq \frac{n}{2}$$

we obtain

$$e_i \leq \frac{\|s^{(i)}\|_1}{2^{h_1+1}} + \frac{2^{h_2}}{\|\tilde{b}_i\|^2} \left| \sum_j \tilde{b}_{i,j} \right| + \frac{n}{2^{h1-h2+1}} + 1.$$

Setting $K_i = \frac{2^{h_2}}{\|\tilde{\mathbf{b}}_i\|^2} \left| \sum_j \tilde{\mathbf{b}}_{i,j} \right| + \frac{n}{2^{h1-h2+1}} + 1$ gives the expected result. ■

Despite the fact that the output vector has a small 2-norm, nothing guaranties that the intermediate values of s do not grow. It would be the case if the basis $\mathbf{B}$ was orthogonal as the norm of the vector s would get lower after each iteration.

In our context, the basis is LLL-reduced which means that it is close to being orthogonal and the norm of s is expected not to grow too much. Yet obtaining an effective bound remains an open problem.

For the rest of our analysis we simply remark that, once ρ and $\mathbf{B}$ are fixed there exists an absolute constant α such that $\max_i(\|s^{(i)}\|_1) \leq \alpha\|v\|_1$. All our experiments tend to show that, in our context, $\alpha \leq 2$, however we could not give a formal proof of this result.

Proposition 1: Let $\mu_2 = \sqrt{\sum \|\tilde{\mathbf{b}}_i\|_2^2}$. There exist two constants α and K such that

$$\|S\|_2 \leq \left(\frac{\alpha\|v\|_1}{2^{h_1+1}} + K \right) \mu_2.$$

Proof: We have seen that there exists α such that $\max_i(\|s^{(i)}\|_1) \leq \alpha\|v\|_1$. Then for all i

$$e_i \leq \frac{\alpha\|v\|_1}{2^{h_1+1}} + K.$$

where $K = \max K_i$. As the output satisfies $S = \sum_{i=0}^{n-1} s_i \tilde{\mathbf{b}}_i$ with $|s_i| \leq e_i$ we get that

$$\|S\|_2 \leq \left\| \sum_{i=0}^{n-1} \left(\frac{\alpha\|v\|_1}{2^{h_1+1}} + K \right) \tilde{\mathbf{b}}_i \right\|_2 \leq \left(\frac{\alpha\|v\|_1}{2^{h_1+1}} + K \right) \mu_2.$$

■

Corollary 1: Let $(p, n, \gamma, \rho, E, 2)$ be a PMNS, α and K be given by Proposition 1. If v satisfies

$$\|v\|_1 \leq \frac{2^{h_1+1}}{\alpha} \left(\frac{\rho}{\mu_2} - K \right)$$

then Algorithm 4 solves the PMNS reduction problem.

Proof: Algorithm 4 solves the PMNS reduction problem if the output S satisfies $\|S\|_2 \leq \rho$. Combining that condition with Proposition 1 gives the expected result. ■

B. Rounding PMNS reduction algorithm

Our approach is quite similar to that of the previous algorithm. Babai's Rounding method consists of computing an integer approximation of $v\mathbf{B}^{-1}$. The main idea is to approximate the rational coordinates of v in $\mathbf{B}$ using register-size approximations of the coefficients of the inverse matrix $\mathbf{B}^{-1} = (\mathbf{b}'_0, \ldots, \mathbf{b}'_{n-1})$. We introduce the same set of parameters, h_1 and h_2, so that $\lfloor 2^{h_1}\mathbf{b}'_{i,j} \rceil$ and $\lfloor \frac{v_j}{2^{h_2}} \rfloor$ are register-size integers. It is then possible to compute a short vector s based on the following equation

$$s = v - \left\lfloor \frac{\lfloor \frac{v}{2^{h_2}} \rfloor \lfloor 2^{h_1}\mathbf{B}^{-1} \rceil}{2^{h_1-h_2}} \right\rfloor \mathbf{B}.$$

Algorithm 5 describes the complete procedure.

Algorithm 5: Rounding Reduction Algorithm

Data: $\mathcal{L}$ a lattice, $\mathbf{B}$ a basis saw as a row matrix, $\mathbf{B}^{-1}$ its inverse and for $0 \leq i, j < n$, $B'_{ij} = \lfloor 2^{h_1}\mathbf{b}'_{ij} \rceil$

Input: $v \in \mathbb{Z}^n$

Result: s equivalent to v with "*small*" coefficients

```
s ← v;
v' ← (v_0 ≫ h_2, ..., v_{n-1} ≫ h_2);
for i = 0 ... n-1 do
    r ← 0;
    for j = 0 ... n-1 do
        r ← r + v'_j × B'_{i,j}
    r ← r ≫ (h_1 - h_2);
    s ← s - r × b_i;
return s;
```

Complexity analysis: The algorithm performs a succession of dot products. A similar analysis to that of Algorithm 4 shows that the computational cost is

$$2n^2\,\mathbf{mult}_w + (4n^2 - 2n)\,\mathbf{add}_w + 2n\,\mathbf{shft}\,.$$

It is possible to save many additions from the observation the coefficients of final result are supposed to fit in a single register. The instruction $s \leftarrow s - r \times \mathbf{b}_i$ can be replaced by $s \leftarrow (s - r \times \mathbf{b}_i) \mod 2^w$ saving n additions per iteration. Algorithm 5 can thus be performed in

$$2n^2\,\mathbf{mult}_w + (3n^2 - 2n)\,\mathbf{add}_w + 2n\,\mathbf{shft}\,.$$

Lemma 2: Let $S = \sum_{i=0}^{n-1} S_i\mathbf{b}_i$ be the output of Algorithm 5 then for all $0 \leq i \leq n-1$ there exists a constant K_i that does not depend on the input v such that

$$|S_i| \leq \frac{\|v\|_1}{2^{h_1+1}} + K_i.$$

Proof: The output S satisfies

$$\begin{aligned} S &= v - \left\lfloor \frac{\lfloor \frac{v}{2^{h_2}} \rfloor \lfloor 2^{h_1}\mathbf{B}^{-1} \rceil}{2^{h_1-h_2}} \right\rfloor \mathbf{B} \\ &= v\mathbf{B}^{-1}\mathbf{B} - \left\lfloor \frac{\lfloor \frac{v}{2^{h_2}} \rfloor \lfloor 2^{h_1}\mathbf{B}^{-1} \rceil}{2^{h_1-h_2}} \right\rfloor \mathbf{B} \\ &= \left(v\mathbf{B}^{-1} - \left\lfloor \frac{\lfloor \frac{v}{2^{h_2}} \rfloor \lfloor 2^{h_1}\mathbf{B}^{-1} \rceil}{2^{h_1-h_2}} \right\rfloor \right) \mathbf{B} \end{aligned}$$

On top of that, for a given i, consider the i-th coordinate R_i of the vector $\left\lfloor \frac{\lfloor \frac{v}{2^{h_2}} \rfloor \lfloor 2^{h_1}\mathbf{B}^{-1} \rceil}{2^{h_1-h_2}} \right\rfloor$

$$\begin{aligned} R_i &= \left(\left\lfloor \frac{\lfloor \frac{v}{2^{h_2}} \rfloor \lfloor 2^{h_1}\mathbf{B}^{-1} \rceil}{2^{h_1-h_2}} \right\rfloor \right)_i \\ &= \left\lfloor \frac{< \lfloor \frac{v}{2^{h_2}} \rfloor, \lfloor 2^{h_1}\mathbf{b}'_{\mathbf{i}} \rceil >}{2^{h_1-h_2}} \right\rfloor \\ &= \left\lfloor \frac{\sum_j^{n-1} \lfloor \frac{v_j}{2^{h_2}} \rfloor \lfloor 2^{h_1}\mathbf{b}'_{i,j} \rceil}{2^{h_1-h_2}} \right\rfloor \\ &= \frac{1}{2^{h_1-h_2}} \sum_j^{n-1} \left\lfloor \frac{v_j}{2^{h_2}} \right\rfloor \lfloor 2^{h_1}\mathbf{b}'_{i,j} \rceil - \varepsilon \\ &= \frac{1}{2^{h_1-h_2}} \sum_j^{n-1} \left(\frac{v_j}{2^{h_2}} - \varepsilon_i \right) \left(2^{h_1}\mathbf{b}'_{i,j} - \varepsilon'_j \right) - \varepsilon \\ &= \sum_j^{n-1} v_j\mathbf{b}'_{i,j} - \sum_j^{n-1} \varepsilon'_j \frac{v_j}{2^{h_1}} - \sum_j^{n-1} \varepsilon_j 2^{h_2}\mathbf{b}'_{i,j} \\ &\quad + \frac{1}{2^{h_1-h_2}} \sum_{j=0}^{n-1} \varepsilon_j\varepsilon'_j - \varepsilon \end{aligned}$$

with $0 \leq \varepsilon, \varepsilon_j < 1$ and $-1/2 \leq \varepsilon'_j < 1/2$. We deduce that

$$\begin{aligned} |S_i| &= \left| \sum_j^{n-1} v_j\mathbf{b}'_{i,j} - \left\lfloor \frac{< \lfloor \frac{v}{2^{h_2}} \rfloor, \lfloor 2^{h_1}\mathbf{b}'_{\mathbf{i}} \rceil >}{2^{h_1-h_2}} \right\rfloor \right| \\ &\leq \sum_j^{n-1} \frac{|v_j|}{2^{h_1+1}} + 2^{h_2} \left| \sum_j^{n-1} \mathbf{b}'_{i,j} \right| + \frac{n}{2^{h_1-h_2+1}} + 1 \\ &\leq \frac{\|v\|_1}{2^{h_1+1}} + K_i \end{aligned}$$

■

Proposition 2: Let $\mu_1 = \sum_i \|\mathbf{b}_i\|_1$. There is a constant K such that

$$\|S\|_1 \leq (\frac{\|v\|_1}{2^{h_1+1}} + K)\mu_1.$$

Proof:

From the previous lemma we have that $S = \sum_i S_i\mathbf{b}_i$ with $|S_i| \leq \frac{\|v\|_1}{2^{h_1+1}} + K_i$. Set $K = \max_i(K_i)$ we obtain that desired result. ■

Corollary 2: Let $(p, n, \gamma, \rho, E, 1)$ be a PMNS and K be given by Proposition 2. If v satisfies

$$\|v\|_1 \leq 2^{h_1+1}\left(\frac{\rho}{\mu_1} - K\right)$$

then Algorithm 5 solves the PMNS reduction problem.

C. Error/Speed trade-off

It is possible to speed-up both reduction algorithms at the cost of an increase of the error. In some situations, the error remains small enough so that the algorithms still solve the PMNS problem. It is based on the observation that, generally speaking,

$$\sum_i a_i \simeq \left\lfloor \frac{\sum_i \lfloor a_i 2^h \rfloor}{2^h} \right\rfloor \simeq \sum_i \left\lfloor \frac{\lfloor a_i 2^h \rfloor}{2^h} \right\rfloor .$$

The first approximation is more accurate but the second one perform additions on smaller integers at the cost of additional shifts. However if h can be chosen to be equal to w, the register size of the target architecture, those shifts can be considered as virtually free. Algorithms 6 and 7 describe two optimized versions of algorithms 4 and 5. In both cases $n^2 - n$ $\mathbf{add}_{2w}$ are replaced by $n^2 - n$ $\mathbf{add}_w$ at the cost of $n^2 - n$ (virtually free) **shft**. The $\mathbf{shft}_w$ notation is added to represent such w-bit right shifts. The respective complexities of Algorithm 6 and 7 are

$$2n^2\,\mathbf{mult}_w + (3n^2 - 2n)\,\mathbf{add}_w + n^2\,\mathbf{shft} + n^2\,\mathbf{shft}_w$$

and

$$2n^2\,\mathbf{mult}_w + (2n^2 - n)\,\mathbf{add}_w + n\,\mathbf{shft} + n^2\,\mathbf{shft}_w .$$

On the other hand for both algorithms the value of K_i from lemma 1 and 2 is modified to $K_i + n - 1$.

Algorithm 6: Optimized Nearest P.

```
s ← v;
for i = n − 1 ... 0 do
    c ← 0;
    for j = 0 ... n − 1 do
        r ← s_j ≫ h2;
        c ← c + ((r × G_ij) ≫ (h_1 − h_2))
    s ← s − r × b_i;
return s;
```

Table I summarizes the complexities of the different internal reduction algorithms. The optimized version of the Rounding method appears to be more efficient that all other methods and performs 33% less additions than Montgomery's algorithm. Even the standard version should perform slightly faster. The efficiency of the Nearest Plane method is more dependent on the target architecture and the relative cost between additions and shifts.

Algorithm 7: Optimized Rounding

```
s ← v;
v' ← (v_0 ≫ h_2, ..., v_{n−1} ≫ h_2);
for i = 0 ... n − 1 do
    r ← 0;
    for j = 0 ... n − 1 do
        r ← r + ((v'_j × B'_ij) ≫ (h_1 − h_2))
    s ← s − r × b_i;
return s;
```

Alg.	$\mathbf{mult}_w$	$\mathbf{add}_w$	**shft**	$\mathbf{shft}_w$
Montgomery [10]	$2n^2$	$3n^2 - n$	0	n
Nearest P.	$2n^2$	$4n^2 - 3n$	n^2	n
Nearest P. (opt)	$2n^2$	$3n^2 - 2n$	n^2	n^2
Rounding	$2n^2$	$3n^2 - 2n$	n	n
Rounding (opt)	$2n^2$	$2n^2 - n$	n	n^2

TABLE I
THEORETICAL COMPLEXITIES OF THE INTERNAL REDUCTION ALGORITHMS

IV. IMPLEMENTATION

This section is dedicated to implementing the previous algorithms on a 64-bit architecture in C. A code generation tool has been developed along with a parameter search tool in order to simplify any implementation of our algorithms on a given field. The software was build using `SageMath` [16] and available in the following **Git** repository
`https://plmlab.math.cnrs.fr/melon359/pmns`

A. Choice of the the polynomial E

The effectiveness of the different methods studied here depends on the value of the 1-norm of the input vector V as shown in Corollaries 1 and 2. Given two polynomials A and B such that $V = AB \mod E$, $\|V\|_1$ highly depends on the choice of the polynomial E. Indeed, the column vector V can be expressed as a matrix-vector product

$$\begin{pmatrix} v_0 \\ \vdots \\ v_{n-1} \end{pmatrix} = \begin{pmatrix} I_n & E_{n-1} \end{pmatrix} (A^C) \begin{pmatrix} b_0 \\ \vdots \\ b_{n-1} \end{pmatrix} = \mathcal{R}_{A,E} \begin{pmatrix} b_0 \\ \vdots \\ b_{n-1} \end{pmatrix}$$

where I_n is the $n \times n$ identity matrix , E_{n-1} is the $n \times (n-1)$ matrix whose column i corresponds to the vector $X^{n+i} \mod E(X)$ in the base $(1, X, \ldots, X^{n-1})$ and A^C is the $(2n-1) \times n$ matrix whose column i correspond to the vector $a_i X^i$ in the base $(1, X, \ldots, X^{2n-2})$.

For a matrix M, let

$$\|M\|_l = \sup_{\|x\|_l = 1} \|Mx\|_l$$

be the matrix norm induced by the vector norm $\|.\|_l$. Such norm satisfies $\forall x \in \mathbb{R}^n, \|Mx\|_l \leq \|M\|_l \|x\|_l$. For instance, the 1-matrix norm is the maximum absolute column sum of the matrix, so that $\|(I_n E_{n-1})\|_1 = \max(\|I_n\|_1, \|E_{n-1}\|_1) = \|E_{n-1}\|_1$. In our case we obtain the following inequalities

$$\begin{aligned}\|V\|_1 &\leq \|\mathcal{R}_{A,E}\|_1\|B\|_1 &(1)\\ &\leq \|(I_nE_{n-1})\|_1\|(A^C)\|_1\|B\|_1 &(2)\\ &\leq \|E_{n-1}\|_1\|A\|_1\|B\|_1 &(3)\\ &\leq n\times\|E_{n-1}\|_1\|A\|_2\|B\|_2. &(4)\end{aligned}$$

Example 2: Consider $A = a_0 + a_1X + a_2X^2$, $B = b_0 + b_1X + b_2X^2$ and $E = X^3 - \lambda$. Then $X^4 \mod E = \lambda$ and $X^5 \mod E = \lambda X$. Considered as column vector in base $(1, X, X^2)$ we get that $E_2 = \begin{pmatrix}\lambda & 0\\ 0 & \lambda\\ 0 & 0\end{pmatrix}$

Let $V = AB \mod E$ we have

$$\begin{aligned}\begin{pmatrix}v_0\\ v_1\\ v_2\end{pmatrix} &= \begin{pmatrix}1 & 0 & 0 & \lambda & 0\\ 0 & 1 & 0 & 0 & \lambda\\ 0 & 0 & 1 & 0 & 0\end{pmatrix}\begin{pmatrix}a_0 & 0 & 0\\ a_1 & a_0 & 0\\ a_2 & a_1 & a_0\\ 0 & a_2 & a_1\\ 0 & 0 & a_2\end{pmatrix}\begin{pmatrix}b_0\\ b_1\\ b_2\end{pmatrix}\\ &= \begin{pmatrix}a_0 & \lambda a_2 & \lambda a_1\\ a_1 & a_0 & \lambda a_2\\ a_2 & a_1 & a_0\end{pmatrix}\begin{pmatrix}b_0\\ b_1\\ b_2\end{pmatrix}\end{aligned}$$

and $\|V\|_1 \leq |\lambda|\|A\|_1\|B\|_1$.

Inequality (1) usually provides tighter bounds than (3) but is also harder to evaluate. In both cases E should be chosen so that the associated matrix E_{n-1} has the smallest norm. The rest of this work focuses on polynomials of the form $X^n - \lambda$ with $|\lambda|$ as small as possible in order to compare the algorithms presented here to that used in [9]. In that case the following theorem holds.

Theorem 1: Let $(p, n, \gamma, \rho, X^n - \lambda, l)$ be a PMNS. Let A, B and V be three degree $n-1$ polynomial and $V = AB \mod E$ with $\|A\|_l \leq \rho$ and $\|B\|_l \leq \rho$.

1) If $l = 1$, let K be given by Proposition 2. If
$$|\lambda|K\mu_1^2 \leq 2^{h_1-1}$$
then Algorithm 5 solves the PMNS reduction problem.
2) If $l = 2$, let α and K be given by Proposition 1. If
$$n\alpha|\lambda|K\mu_2^2 \leq 2^{h_1-1}$$
then Algorithm 4 solves the PMNS reduction problem.

Proof: As $E = X^n - \lambda$, we have

$$E_{n-1} = \lambda\times\begin{pmatrix} & I_{n-1} & \\ 0 & \dots & 0\end{pmatrix} = \begin{pmatrix}\lambda & 0 & \dots & 0\\ 0 & \lambda & \ddots & \vdots\\ \vdots & \ddots & \ddots & 0\\ 0 & \dots & 0 & \lambda\\ 0 & \dots & \dots & 0\end{pmatrix}$$

so that $\|E_{n-1}\|_1 = |\lambda|$.

- If $l = 1$, combining Corollary 2 with the inequality 3 gives that Algorithm 5 solves the PMNS problem if
$$\left(\frac{|\lambda|\mu_1}{2^{h_1+1}}\right)\rho^2 - \rho + K\mu_1 \leq 0$$
is satisfied. The real function $x \mapsto ax^2 - x + c$ reaches its minimum when $x = 1/2a$. We deduce that the previous inequality is satisfied as soon as
$$\left(\frac{|\lambda|\mu_1}{2^{h_1+1}}\right)\left(\frac{2^{h_1}}{|\lambda|\mu_1}\right)^2 - \left(\frac{2^{h_1}}{|\lambda|\mu_1}\right) + K\mu_1 \leq 0$$
that is
$$|\lambda|K\mu_1^2 \leq 2^{h_1-1}.$$
- If $l = 2$, combining Corollary 1 with the inequality 4 gives that Algorithm 4 solves the PMNS problem if
$$\left(\frac{n\alpha|\lambda|\mu_2}{2^{h_1+1}}\right)\rho^2 - \rho + K\mu_1 \leq 0$$
which is satisfied as soon as
$$n\alpha|\lambda|K\mu_2^2 \leq 2^{h_1-1}$$

■

B. Parameter generation

Let p be a prime number. Generating the parameters for p is done in the following sequence:

1) chose n the degree of the reduction polynomial E. Targeting a 64-bit architecture, n must satisfy $64 \times n \geq \lceil\log_2(p)\rceil$;
2) find λ such that $E = X^n \pm \lambda$ has a root γ in $\mathbb{F}_p$;
3) compute the lattice associated to the polynomial E and a LLL-reduced basis $\mathbf{B}$;
4) compute the GSO $(\tilde{\mathbf{b}}_\mathbf{0}, \dots, \tilde{\mathbf{b}}_\mathbf{n-1})$ and inverse matrix $\mathbf{B}^{-1} = (\mathbf{b}'_0, \dots, \mathbf{b}'_{n-1})$ of $\mathbf{B}$;
5) for each algorithm (Nearest plane and Rounding) find a pair of integer (h_1, h_2) such that
   - for all i, j $\left\lfloor\frac{2^{h_1}\tilde{\mathbf{b}}_{ij}}{\|\tilde{\mathbf{b}}_i\|^2}\right\rceil$ or $\lfloor 2^{h_1}\mathbf{b}'_{ij}\rceil$ fit in a 64-bit register;
   - every coordinates of $\lfloor v/2^{h_2}\rfloor$ fit in a 64-bit register for any valid input vector v;
   - there exists a ρ satisfying Theorem 1;
6) If no parameters can be found, go to step 2 and change λ or to step 1 and increase n.

Remark 3: The search for h_1 and h_2 is performed through exhaustive search with some conditions to speed up the process. Set $M_1 = \max_{i,j}\left|\frac{\tilde{\mathbf{b}}_{ij}}{\|\tilde{\mathbf{b}}_i\|^2}\right|$ and $M_2 = \max_{i,j}|\mathbf{b}'_{ij}|$, then h_1 must satisfy $2^{h_1}M_k < 2^{63} - 1/2$ which gives a maximum bound for h_1.

Similarly, $\lfloor\|v\|_\infty/2^{h_2}\rfloor$ must fit in a 64-bit register and we know from the inequality that $\|v\|_\infty \leq \|v\|_1 \leq |\lambda|\|A\|_1\|B\|_1$ which gives a minimum bound on h_2.

Remark 4: When verifying that Theorem 1 holds it is possible to compute the exact values of the K_i from Lemmas 1 and 2 thus obtaining tighter bounds on the norm of the output vector.

C. Comparisons

In [10] the authors have provided numerous C implementations of Montgomery PMNS reduction algorithms for many key size relevant for elliptic curve cryptography (i.e., 192, 224, 256, 384 and 521 bits). The same prime numbers have been used to generate parameters for our reduction algorithms. For each prime p, we have generated 10000 pairs of elements of $\mathbb{Z}/p\mathbb{Z}$, converted then into the relevant PMNS representation and performed a polynomial multiplication followed by an external reduction. Then, for each algorithm, we have measured the number of clock cycles required to perform the internal reduction as well as the total number of instructions performed by the CPU. The experiment was performed on a Linux system powered by an *Intel(R) Core(TM) i5-10500 CPU @ 3.10GHz* and the code was compiled with *gcc 9.3.0* using the *-O3* option. The benchmark test results are shown in Table II and III.

p size	192	224		256	384		512	
degree n	4	4	5	5	7	8	10	11
Montgomery [10]	60	63	86	83	155	206	300	357
Nearest P.	83	83	143	133	265	331	494	606
Nearest P. (opt)	70	70	108	117	196	248	405	494
Rounding	50	52	76	74	155	205	321	362
Rounding (opt)	45	45	64	70	148	199	291	344

TABLE II
NUMBER OF CPU CLOCK CYCLES TO PERFORM AN INTERNAL REDUCTION USING A POLYNOMIAL OF THE FORM $X^n - \lambda$

p size	192	224		256	384		512	
degree n	4	4	5	5	7	8	10	11
Montgomery [10]	191	188	255	254	496	649	966	1122
Nearest P.	228	214	388	357	793	1021	1624	2014
Nearest P. (opt)	194	199	298	304	538	743	1218	1486
Rounding	181	178	261	253	468	644	987	1021
Rounding (opt)	159	156	221	220	381	493	735	879

TABLE III
NUMBER OF CPU INSTRUCTIONS TO PERFORM AN INTERNAL REDUCTION USING A POLYNOMIAL OF THE FORM $X^n - \lambda$

First let us say that CPU clock cycle measurements should be taken with caution especially when numbers are that small. However, the measurements shown in II are very consistent from one execution to another with a standard deviation of less than one cycle.

Generally speaking, the results are consistent with the theoretical complexities from Table I. The Rounding and Optimized Rounding algorithms are faster than Montgomery reduction in most cases. On top of that the optimized version of the Rounding method executes 10 to 20% less instructions than any other approaches. The Nearest Plane algorithms however do not appear to be that competitive at the moment. These algorithms are significantly slower as shown in both tables and it clearly highlights that the numerous shifts, whose cost is often overlooked, have a great impact on the performances of core operations such as modular multiplications.

Another phenomenon is surprising: as the size of p increases, so does the degree of the polynomial E and the difference between the Rounding algorithm and the Montgomery-like algorithm tends to decrease. For large primes, Montgomery reduction becomes even faster than the standard Rounding method. The analysis of the instruction count is interesting for that matter. For instance, with $n = 11$, Table III confirms that the Rounding method executes less operations but Montgomery reduction ends up being faster. One explanation is that the super-scalar nature of modern processors combine with instruction pipelining and compiler optimizations can introduce a significant level of variability between theoretical analysis and practical results on a given platform. This is confirmed by the fact that compiling the same code without any optimization (*-O0* option of *gcc*) reverses the results with the Rounding algorithm being faster than Montgomery's.

Even more surprising is the fact that, for $n = 10$, the Rounding algorithm executes more instructions than the Montgomery-like counter-part. An analysis of the assembly code shows that the number of additions or multiplications are consistent with the theory but a large number of *move* instructions slows down the Rounding method. It appears that, in some cases, the compiler has a hard time optimizing memory allocation thus impacting the instruction count and execution speed.

V. CONCLUSIONS

The Polynomial Modular Number System have received a lot of interest recently. Various approaches and sets of parameters have been studied but the internal reduction step of all these works remains the same and is always based on a Montgomery-like algorithm originally proposed in [14]. In this work we have proposed two completely different internal reduction algorithms based Babai's Closest Vector algorithms for integer lattices. Our algorithms are both provided in two versions: one standard that aim at minimizing the size of the output vector and one optimized version where an error/speed trade-off is made.

Both algorithms are competitive with the previous works. In particular, algorithms based on Babai's rounding method are theoretically faster, the optimized version performing 33% less additions than Montgomery's. Experimental results comparing our implementations to those in [10] confirms the efficiency of our approach.

All our work has been carried in the most general case possible, with a prime p with no particular shape. Adapting our algorithms to special primes is one potential field for future studies. On top of that, our C code generation tool could be optimized depending on the targeted platform.

REFERENCES

[1] L. Babai. On lovász' lattice reduction and the nearest lattice point problem (shortened version). *Combinatorica*, 1986.

[2] Jean-Claude Bajard, Laurent Imbert, and Thomas Plantard. Arithmetic Operations in the Polynomial Modular Number System. In *ARITH'05: 17th IEEE Symposium on Computer Arithmetic*, pages 206–213, USA, 2005. IEEE computer society.

[3] Jean-Claude Bajard, Laurent Imbert, and Thomas Plantard. Modular Number Systems: Beyond the Mersenne Family. In *SAC'04: 11th International Workshop on Selected Areas in Cryptography*, number 3357 in Lecture Notes in Computer Science, pages 159–169, University of Waterloo, Ontario (Canada), August 2005. Springer Verlag.

[4] Jean-Claude Bajard, Jérémy Marrez, Thomas Plantard, and Pascal Véron. On Polynomial Modular Number Systems over Z/pZ. working paper or preprint, June 2020.

[5] Paul Barrett. Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor. In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 311–323. Springer, 1986.

[6] Cyril Bouvier and Laurent Imbert. An Alternative Approach for SIDH Arithmetic. working paper or preprint, April 2021.

[7] Jaewook Chung and Anwar Hasan. More generalized mersenne numbers. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography*, pages 335–347, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[8] Titouan Coladon, Philippe Elbaz-Vincent, and Cyril Hugounenq. MPHELL: A fast and robust library with unified and versatile arithmetics for elliptic curves cryptography (extended version). In *ARITH 2021*, Transactions on Emerging Topics in Computing, Torino, Italy, June 2021.

[9] Laurent-Stéphane Didier, Fangan-Yssouf Dosso, Nadia El Mrabet, Jérémy Marrez, and Pascal Véron. Randomization of Arithmetic over Polynomial Modular Number System. In *26th IEEE International Symposium on Computer Arithmetic*, volume 1 of *Proceedings of the 2019 IEEE 26th Symposium on Computer Arithmetic*, pages 199–206, Kyoto, Japan, June 2019. IEEE Computer Society.

[10] Laurent-Stéphane Didier, Fangan-Yssouf Dosso, and Pascal Véron. Efficient modular operations using the adapted modular number system. *Journal of Cryptographic Engineering*, January 2020.

[11] Nadia El Mrabet and Nicolas Gama. Efficient Multiplication over Extension Fields. In *WAIFI 2012*, Ghent, Belgium, July 2012.

[12] H.W. jr. Lenstra, A.K. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

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

[14] Christophe Negre and Thomas Plantard. Efficient modular arithmetic in adapted modular number system using lagrange representation. In Yi Mu, Willy Susilo, and Jennifer Seberry, editors, *Information Security and Privacy*, pages 463–477, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[15] J.A. Solinas. Generalized Mersenne Number. Technical report, Center for Applied Cryptographic Research, University of Waterloo, ON, 1999.

[16] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.3)*, 2021. `https://www.sagemath.org`.

[17] Peter van Emde Boas. Another np-complete problem and the complexity of computing short vectors in a lattice. Technical report, University of Amsterdam, Department of Mathematics,Netherlands, 1981.

Google
Google

Nicolas Méloni Nicolas Méloni received the MS degree in mathematics in 2004 and the PhD degree in computer science in 2007 from the Université Montpellier 2, France. He did postdoctoral work with the Centre for Applied Cryptographic Research at the University of Waterloo, Canada from 2008 to 2010. Since September 2010, he is an assistant professor at Université de Toulon, France. His research interests are in computer arithmetic and cryptography.