PMNS revisited for consistent redundancy and equality test

Fangan Yssouf Dosso[1*], Alexandre Berzati[2], Nadia El Mrabet[1], Julien Proy[2]

[1]SAS laboratory, École des Mines de Saint-Étienne, Gardanne, France.
[2]Thales DIS, Meyreuil, France.

*Corresponding author. E-mail: fanganyssouf.dosso@emse.fr;
Contributing authors: alexandre.berzati@thalesgroup.com; nadia.el-mrabet@emse.fr; julien.proy@thalesgroup.com;

Abstract

The Polynomial Modular Number System (PMNS) is a non-positional number system for modular arithmetic. A PMNS is defined by a tuple $\boldsymbol{(p, n, \gamma, \rho, E)}$, where $\boldsymbol{p}$, $\boldsymbol{n}$, $\boldsymbol{\gamma}$ and $\boldsymbol{\rho}$ are positive non-zero integers and $\boldsymbol{E} \in \mathbb{Z}_{\boldsymbol{n}}[\boldsymbol{X}]$ is a monic polynomial such that $\boldsymbol{E(\gamma) \equiv 0 \pmod{p}}$. The PMNS is a redundant number system. In [1], Didier et al. used this redundancy property to randomise the data during the Elliptic Curve Scalar Multiplication (ECSM). In this paper, we refine the results on redundancy and propose several new results on PMNS. More precisely, we study a generalisation of the Montgomery-like internal reduction method proposed in [2], along with some improvements on parameter bounds for smaller memory cost to represent elements in this system. We also show how to perform equality test in the PMNS.

Keywords: Modular arithmetic, Polynomial modular number system, Internal reduction, Euclidean lattices, Redundancy, Equality test

1 Introduction

Modular arithmetic is one of the key points for efficient and secure cryptographic applications, such as Elliptic Curve Cryptography (ECC) [3, 4], RSA [5] and some post-quantum schemes such as Crystal-Dilithium [6]. Many approaches have been proposed to perform it both efficiently and safely. When the modulus can be chosen freely, special numbers such as Mersenne and pseudo-Mersenne numbers [7, 8] can be used. They are very efficient for modular reduction. In many applications, such as pairing-based cryptography [9], the modulus cannot be chosen arbitrarily and does not have a shape that would allow speeding up modular arithmetic. For such moduli, there are general modular reduction methods, such as those proposed by Barrett [10] and Montgomery [11]. These methods are based on the classical binary number system. To further improve this modular arithmetic, alternative number systems have been proposed. A famous one is the Residue Number System (RNS) [12], which has been widely studied and implemented. It is an efficient number system which has a very high parallelisation capability. In 2004, Bajard et al. proposed the Polynomial Modular Number System (PMNS) [13] for modular arithmetic modulo a prime integer p. In this system, elements are polynomials. More precisely, we have the following definition.

Definition 1.1. A polynomial modular number system (PMNS) is defined by a tuple $\mathcal{B} = (p, n, \gamma, \rho, E)$, such that for each integer $y \in$

$\mathbb{Z}/p\mathbb{Z}$ there exists a polynomial $V(X) = v_0 + v_1.X + \cdots + v_{n-1}.X^{n-1}$ such that:

$$y = V(\gamma) \pmod p ,$$

where $|v_i| < \rho$ and $2 \leqslant \rho, \gamma < p$. In this case, we say that $V(X)$ (or equivalently the vector $V = (v_0, \ldots, v_{n-1})$) is a representation of y in $\mathcal{B}$.
The parameter E is a monic polynomial in $\mathbb{Z}[X]$ of degree n, having γ as a root modulo p, i.e. $E(\gamma) \equiv 0 \pmod p$.

In the PMNS, polynomial multiplications are performed modulo E. The reduction modulo E is called **external reduction** and E is called *external reduction polynomial.*

After an addition or multiplication (modulo E), the result might not be in the PMNS, since its coefficients are likely to be greater than ρ, especially after a multiplication. To keep the coefficients of the result bounded by ρ, an operation called **internal reduction** is performed. It is somehow equivalent to the modular reduction in classical binary representation. In Section 2.5, we describe in detail the Montgomery-like internal reduction method proposed by Negre and Plantard [2].

Like the RNS, the PMNS offers a very high parallelisation capability, due to the polynomial structure of its elements. In [14–16], it is shown that the PMNS can be an efficient alternative to the classical representation, for integer sizes used in ECC. It is also shown that operations in PMNS can be vectorised very efficiently. In [17], sequential and vectorised implementations of PMNS and RNS are compared, for small to large moduli.

The PMNS is a redundant number system. Indeed, an element $a \in \mathbb{Z}/p\mathbb{Z}$ can have more than one representation. For example, in the PMNS $\mathcal{B} = (19, 3, 7, 2, X^3 - 1)$, the polynomials $(-X - 1)$ and X^2 are representations of 11.
In [1], Didier et al. use this redundancy property to randomise arithmetic and conversion operations in order to randomise the data during the Elliptic Curve Scalar Multiplication (ECSM). This redundancy property of PMNS has however a drawback. It makes the equality check non-trivial. Indeed, since elements can have more than one representation, a simple comparison of two polynomials is not sufficient to conclude that they represent two different elements in $\mathbb{Z}/p\mathbb{Z}$. Until now, the best way to determine if two polynomials $A, B \in \mathbb{Z}_{n-1}[X]$ represent the same element $a \in \mathbb{Z}/p\mathbb{Z}$ is to first compute $C = A - B$, then evaluate C modulo p to check if $C(\gamma) \equiv 0 \pmod p$. So, a conversion out of the PMNS (the evaluation of C) is required, which is not appropriate for some cryptographic use cases where equality testing is for instance done during the process, to detect fault attacks [18, 19] or for signature verification [20].

Contributions

This paper is a comprehensive study of how elements are represented in the PMNS and how they behave with respect to arithmetic and conversion operations. We aim to provide a complete understanding of this system. To do so, we introduce a generalised Montgomery-like internal reduction method, with a number of properties. This is the first major (and rather theoretical) contribution of this paper. It is a toolkit of results that will serve as the basis for our three major (and practical) contributions, which are:

- Improved bounds to represent PMNS elements, for smaller memory cost.
- Precise study of redundancy in PMNS, explaining the relation between representations.
- Equality test within the PMNS.

These contributions make the PMNS a more complete system and allows to compute comparison of PMNS element and remove the burden of conversion to original system.

Paper organisation

This paper is structured as follows. Section 2 gives some background on PMNS. Section 3 introduces **GMont-like**, a generalised Montgomery-like internal reduction method with a number of properties. In Section 4, we present a set of properties for coefficient reductions in some special domains, which are interesting for both memory requirement and redundancy in the PMNS. These two theoretical sections constitute the toolkit for the next three sections, which respectively focus on memory improvement, redundancy and equality test in the PMNS. Before concluding, Section 8 addresses the related computational operation costs.

2 Background on PMNS

For consistency, we assume that $p \geqslant 3$ and $n \geqslant 2$. Let us start with the notations we will be using throughout this paper.

2.1 Some notations and conventions

- Let $\mathcal{A}$ be a set, $\#\mathcal{A}$ denotes the number of elements of $\mathcal{A}$.
- $\mathbb{Z}_{n-1}[X]$ is the set of polynomials in $\mathbb{Z}[X]$ with degrees lower than or equal to $n-1$: $\mathbb{Z}_{n-1}[X] = \{C \in \mathbb{Z}[X] \mid \deg(C) \leqslant n-1\}$.
- If $A \in \mathbb{Z}_{n-1}[X]$, we assume that $A(X) = a_0 + a_1X + \cdots + a_{n-1}X^{n-1}$ can equivalently be represented as the vector $A = (a_0, \ldots, a_{n-1}) \in \mathbb{Z}^n$.
- Let $a \in \mathbb{Z}/p\mathbb{Z}$. If a polynomial $A \in \mathbb{Z}_{n-1}[X]$ is such that $A(\gamma) \equiv a \pmod p$, then we say that A is a representation of a in $\mathcal{B}$.
- If $\alpha = (\alpha_0, \ldots, \alpha_{n-1})$, then $\alpha \pmod \phi = (\alpha_0 \pmod \phi), \ldots, \alpha_{n-1} \pmod \phi))$.
- Let $C \in \mathbb{Z}[X]$. $C \bmod (E, \phi)$ is the polynomial reduction $C \bmod E$, with the coefficients of the result are computed modulo ϕ.
- Unless otherwise specified, the symbol $\mathcal{B}$ will always designate a PMNS (p, n, γ, ρ, E).
- Let $A \in \mathbb{Z}^n$ be a vector. We have: $\|A\|_\infty = \max\limits_{0 \leqslant i \leqslant n-1} |a_i|$.
- Let $\mathcal{W} \in \mathbb{Z}^{n \times n}$ be a matrix. We have: $\|\mathcal{W}\|_1 = \max\limits_{0 \leqslant j \leqslant n-1} \sum_{i=0}^{n-1} |w_{ij}|$.

2.2 PMNS and euclidean lattices

Given a tuple $\mathcal{B} = (p,\, n,\, \gamma,\, \rho,\, E)$, one can build the n-dimensional full-rank Euclidean lattice $\mathcal{L}_\mathcal{B}$ defined as follows:

$$\mathcal{L}_\mathcal{B} = \{A \in \mathbb{Z}_{n-1}[X] \quad A(\gamma) \equiv 0 \pmod p\}. \tag{1}$$

It is the set of polynomials with degrees strictly less than n and having γ as a root modulo p. A basis of $\mathcal{L}_\mathcal{B}$ is the $n \times n$ matrix $\mathbf{B}$, defined as follows:

$$\mathbf{B} = \begin{array}{c@{}l} \begin{pmatrix} p & 0 & 0 & \ldots & 0 & 0 \\ t_1 & 1 & 0 & \ldots & 0 & 0 \\ t_2 & 0 & 1 & \ldots & 0 & 0 \\ \vdots & & & \ddots & & \vdots \\ t_{n-2} & 0 & 0 & \ldots & 1 & 0 \\ t_{n-1} & 0 & 0 & \ldots & 0 & 1 \end{pmatrix} & \begin{array}{l} \leftarrow p \\ \leftarrow X + t_1 \\ \leftarrow X^2 + t_2 \\ \\ \leftarrow X^{n-2} + t_{n-2} \\ \leftarrow X^{n-1} + t_{n-1} \end{array} \end{array}, \tag{2}$$

where $t_i = (-\gamma^i) \bmod p$.

The following result, from [21], gives a condition on ρ for $\mathcal{B}$ to be a PMNS.

Theorem 2.1. *Let $p \geqslant 3$, $n \geqslant 1$ be two integers, $\gamma \in \mathbb{Z}/p\mathbb{Z} \setminus \{\bar{0}\}$ and $E \in \mathbb{Z}_n[X]$ a monic polynomial such that $E(\gamma) \equiv 0 \pmod p$. Let $\mathcal{W}$ be any basis of the lattice $\mathcal{L}_\mathcal{B}$. A tuple $\mathcal{B} = (p,\, n,\, \gamma,\, \rho,\, E)$ defines a PMNS if:*

$$\rho > \frac{1}{2}\|\mathcal{W}\|_1 .$$

Remark 2.1. Theorem 2.1 works with any basis of $\mathcal{L}_\mathcal{B}$ regardless of its quality, defined here as its 1-norm. In practice however, we want ρ to be as small as possible for small memory cost. So, one should take $\mathcal{W}$ as a reduced basis of $\mathcal{L}_\mathcal{B}$ (the smaller the better). Such a basis can be obtained with algorithms like LLL [22], BKZ[23] or HKZ[24], applied to the basis $\mathbf{B}$ (Equation 2). This results in a parameter ρ such that $\rho \approx \sqrt[n]{p}$.

2.3 A property on $\mathcal{L}_\mathcal{B}$ sublattices

Let $\mathcal{L}$ be a sublattice of $\mathcal{L}_\mathcal{B}$, having $\mathcal{G}$ as a basis. Assuming that $\mathcal{G}_i$ is the i^{th} row of $\mathcal{G}$, let us consider the fundamental regions $\mathcal{H}$ and $\mathcal{H}'$ defined as follows:

$$\mathcal{H} = \{t \in \mathbb{R}^n \mid t = \sum_{i=0}^{n-1} \mu_i \mathcal{G}_i \text{ and } 0 \leqslant \mu_i < 1\}, \tag{3}$$

$$\mathcal{H}' = \{t \in \mathbb{R}^n \mid t = \sum_{i=0}^{n-1} \mu_i \mathcal{G}_i \text{ and } -\frac{1}{2} \leqslant \mu_i < \frac{1}{2}\}. \tag{4}$$

Lemma 2.1. *Let $a \in \mathbb{Z}/p\mathbb{Z}$. There exists $A \in \mathcal{H} \cap \mathbb{Z}^n$ (resp. $A \in \mathcal{H}' \cap \mathbb{Z}^n$), such that: $a = A(\gamma) \pmod p$.*

Proof. Let $V = (a,\, 0,\, \ldots,\, 0)$. From lattice theory, we know that there exists a vector $T \in \mathcal{L}$ such that $V + T \in \mathcal{H}$ (resp. $V + T \in \mathcal{H}'$). Let $A = V + T$. Since $\mathcal{L} \subseteq \mathcal{L}_\mathcal{B} \subset \mathbb{Z}^n$, $T \in \mathbb{Z}^n$. So, $A \in \mathbb{Z}^n$. Additionally, $T(\gamma) \equiv 0 \pmod p$, since $T \in \mathcal{L}$. Thus, $A(\gamma) \equiv V(\gamma) \pmod p$. □

This lemma says that every element $a \in \mathbb{Z}/p\mathbb{Z}$ has at least one representation in $\mathcal{H} \cap \mathbb{Z}^n$ (resp. $\mathcal{H}' \cap \mathbb{Z}^n$).

Remark 2.2. Let $A \in \mathbb{Z}^n$.

- If $A \in \mathcal{H}$, then $\|A\|_\infty < \|\mathcal{G}\|_1$.
- If $A \in \mathcal{H}'$, then $\|A\|_\infty \leqslant \frac{1}{2}\|\mathcal{G}\|_1$.

2.4 External reduction

Let $\mathcal{B} = (p, n, \gamma, \rho, E)$ be a PMNS, where $E \in \mathbb{Z}_n[X]$ is a monic polynomial. As explained in the introduction, polynomial multiplications in PMNS are done modulo E. The reduction modulo E is called **external reduction**.

When $E(X) = X^n - \lambda$, with $\lambda \in \mathbb{Z}\backslash\{0\}$ and $|\lambda|$ small, this operation can be done very efficiently, as explained in [13].

Let us assume that $E(X) = X^n + e_{n-1}X^{n-1} + \cdots + e_1 X + e_0$. To optimise the general case for E, the authors in [16] introduce the **external reduction matrix** $\mathcal{E}$, which is defined as follows:

$$\mathcal{E} = \begin{pmatrix} -e_0 & -e_1 & \dots & -e_{n-1} \\ \dots & \dots & \dots & \dots \\ \vdots & \vdots & & \vdots \\ \dots & \dots & \dots & \dots \end{pmatrix} \begin{array}{l} \leftarrow X^n \bmod E \\ \leftarrow X^{n+1} \bmod E \\ \\ \leftarrow X^{2n-2} \bmod E \end{array} . \quad (5)$$

It is a $(n-1) \times n$ matrix where each row contains the coefficients of the polynomial $X^{n+i} \pmod E$, for $i = 0, \dots, n-2$.

Let $A, B \in \mathbb{Z}_{n-1}[X]$ be two polynomials. Let $C = A \times B = c_0 + c_1 X + \cdots + c_{2n-2}X^{2n-2}$ and $R = C \bmod E$. In [16], it is shown that:

$$R = (c_0, \dots, c_{n-1}) + (c_n, \dots, c_{2n-2})\mathcal{E}. \quad (6)$$

Moreover, with $\mathcal{E}'$ being the $(n-1) \times n$ matrix such that ${\mathcal{E}'}_{ij} = |\mathcal{E}_{ij}|$, it is also shown that:

$$\|R\|_\infty \leqslant w \|A\|_\infty \|B\|_\infty \,, \quad (7)$$

with:

$$w = \|(1, 2, \dots, n) + (n-1, n-2, \dots, 1)\mathcal{E}'\|_\infty \,. \quad (8)$$

In [16] (Table 1), a set of polynomials E minimising w, and building very sparse and friendly matrices $\mathcal{E}$ are given.

2.5 Internal reduction

Many approaches have been proposed to perform the internal reduction [2, 13, 15, 25]. In this paper, we focus on the Montgomery-like method proposed in [2]. It introduces three parameters: an integer $\phi \geqslant 2$ and two polynomials $M, M' \in \mathbb{Z}_{n-1}[X]$ such that: $M(\gamma) \equiv 0 \pmod p$ and $M' = -M^{-1} \bmod (E, \phi)$. Algorithm 1 describes this method. Notice that this method introduces a factor ϕ^{-1} on the output. So, a conversion in Montgomery domain should be done, as explained in [14](Section 4.4).

Algorithm 1 Coefficients reduction [2]

```
Require: $V \in \mathbb{Z}_{n-1}[X]$, $M \in \mathbb{Z}_{n-1}[X]$ such that
    $M(\gamma) \equiv 0 \pmod p$, $\phi \in \mathbb{N} \setminus \{0, 1\}$ and $M' = -M^{-1} \bmod (E, \phi)$.
Ensure: $S(\gamma) = V(\gamma)\phi^{-1} \pmod p$, with $S \in \mathbb{Z}_{n-1}[X]$
1: $Q \leftarrow V \times M' \bmod (E,\, \phi)$
2: $T \leftarrow Q \times M \bmod E$
3: $S \leftarrow (V + T)/\phi$
4: return $S$
```

In [14], focusing on the case $E(X) = X^n - \lambda$, the authors show that this approach is at least as efficient as the classical Montgomery modular reduction method. In [16], the authors extend that work to any monic polynomial $E \in \mathbb{Z}_n[X]$. To do that efficiently, regardless E shape, they introduce the matrices $\mathcal{M}$ and $\mathcal{M}'$ defined below. With these matrices, Algorithm 1 is rewritten as Algorithm 2.

$$\mathcal{M} = \begin{pmatrix} m_0 & m_1 & \dots & m_{n-1} \\ \dots & \dots & \dots & \dots \\ \vdots & \vdots & & \vdots \\ \dots & \dots & \dots & \dots \end{pmatrix} \begin{array}{l} \leftarrow M \\ \leftarrow X.M \bmod E \\ \\ \leftarrow X^{n-1}.M \bmod E \end{array} , \quad (9)$$

$$\mathcal{M}' = \begin{pmatrix} m'_0 & m'_1 & \dots & m'_{n-1} \\ \dots & \dots & \dots & \dots \\ \vdots & \vdots & & \vdots \\ \dots & \dots & \dots & \dots \end{pmatrix} \begin{array}{l} \leftarrow M' \\ \leftarrow X.M' \bmod (E, \phi) \\ \\ \leftarrow X^{n-1}.M' \bmod (E, \phi) \end{array} .$$

2.6 Arithmetic and conversion operations

In this section, we briefly remind the main arithmetic and conversion operations in the PMNS.

Algorithm 2 Coefficients reduction for PMNS (**RedCoeff**)

```
Require: $V \in \mathbb{Z}_{n-1}[X]$, the matrices $\mathcal{M}, \mathcal{M}'$ and $\phi \in \mathbb{N} \setminus \{0, 1\}$.
Ensure: $S(\gamma) = V(\gamma)\phi^{-1} \pmod{p}$, with $S \in \mathbb{Z}_{n-1}[X]$
1: $Q = (v_0, \ldots, v_{n-1})\mathcal{M}' \pmod{\phi}$
2: $T = (q_0, \ldots, q_{n-1})\mathcal{M}$
3: $S \leftarrow (V + T)/\phi$
4: return $S$
```

2.6.1 Addition and subtraction

Let $A, B \in \mathcal{B}$ be two polynomials. The addition (resp. subtraction) is a simple polynomial addition (resp. subtraction); i.e. the operation $C = A + B$ (resp. $C = A - B$). Although $\deg(C) < n$, C might not be in $\mathcal{B}$, since we only have $\|C\|_\infty < 2\rho$. So, an internal reduction should be done on C to guarantee the result in the PMNS. However, this reduction (see Algorithm 2) is too expensive compared to the polynomial addition or subtraction.

To avoid this need for coefficient reductions, a parameter δ has been introduced in [14]. It is the maximum number of consecutive additions of elements in $\mathcal{B}$ that we want to compute before doing a modular multiplication.

If C is the result of such successive additions, then $\|C\|_\infty < (\delta + 1)\rho$. In Section 2.6.4, we remind the bounds proposed in [16] to ensure complete coefficient reductions of the product of such polynomials.

2.6.2 Modular multiplication

Multiplication is a simple polynomial multiplication, followed by the external reduction and then the internal reduction, see Algorithm 3.

Algorithm 3 Modular multiplication for PMNS

```
Require: $A, B \in \mathbb{Z}_{n-1}[X]$, the matrices $\mathcal{M}, \mathcal{M}'$, $\mathcal{E}$ and $\phi \in \mathbb{N} \setminus \{0, 1\}$.
Ensure: $S(\gamma) = A \cdot B(\gamma)\phi^{-1} \pmod{p}$, with $S \in \mathbb{Z}_{n-1}[X]$
1: $C = A \times B$
2: $V = (c_0, \ldots, c_{n-1}) + (c_n, \ldots, c_{2n-2})\mathcal{E}$
3: $S \leftarrow \mathrm{RedCoeff}(V)$
4: return $S$
```

2.6.3 Conversion operations

The use of PMNS requires conversions from and to the classical binary representation. These operations are described in detail in [13](Section 4.4) and [16](Section 6.1). The conversion from PMNS is a polynomial evaluation modulo p. Here, we briefly present the conversion process to PMNS.

Algorithms 4 and 5 describe two ways to convert an element from $\mathbb{Z}/p\mathbb{Z}$ to $\mathcal{B}$. Algorithm 5 is fast (slightly faster than a modular multiplication), but requires precomputed polynomials P_i, such that $P_i(\gamma) \equiv (\rho^i\phi^2) \pmod{p}$, for $i = 0 \ldots n-1$. Algorithm 4 is relatively slower (it makes n calls to RedCoeff, whereas Algorithm 5 makes only one), but only requires to precompute $\tau = \phi^n \bmod p$. In practice, Algorithm 4 is used to precompute the polynomials P_i and then Algorithm 5 is used for fast conversion.

Algorithm 4 Exact conversion from binary to PMNS

```
Require: $a \in \mathbb{Z}/p\mathbb{Z}$, $\mathcal{B} = (p, n, \gamma, \rho, E)$ and $\tau = \phi^n \bmod p$
Ensure: $A \in \mathcal{B}$, such that $A \equiv a_{\mathcal{B}}$
1: $\alpha = a \times \tau \pmod{p}$
2: $A = (\alpha, 0, \ldots, 0)$ # a polynomial of degree 0
3: for $i = 0 \ldots n-1$ do
4:     $A \leftarrow \mathrm{RedCoeff}(A)$
5: end for
6: return $A$
```

Algorithm 5 Fast conversion from classical representation to PMNS

```
Require: $a \in \mathbb{Z}/p\mathbb{Z}$ and $P_i$ such that $P_i(\gamma) \equiv (\rho^i\phi^2) \pmod{p}$
Ensure: $A \in \mathcal{B}$, such that $A \equiv (a\phi)_{\mathcal{B}}$
1: $t = (t_{n-1}, \ldots, t_0)_\rho$ # radix-$\rho$ decomposition of $a$
2: $U \leftarrow \sum_{i=0}^{n-1} t_i P_i(X)$
3: $A \leftarrow \mathrm{RedCoeff}(U)$
4: return $A$
```

In Algorithm 5 (line 1), a radix-ρ decomposition is done. For this operation to be fast, ρ must be a power of two, which is always possible (see Section 2.6.4).

2.6.4 Bounds for consistency

In [16], the authors give these bounds on ρ and ϕ to ensure that the arithmetic and conversion operations in the PMNS are consistent:

$$\rho \geqslant 2\|\mathcal{M}\|_1 \text{ and } \phi \geqslant 2w\rho(\delta+1)^2 .$$

Consistent here means that the conversion algorithms to PMNS outputs are in PMNS and also that Algorithm 3 applied to inputs A, B such that $\|A\|_\infty, \|B\|_\infty < (\delta+1)\rho$, returns a result in the PMNS. In [16], the authors suggest using $\rho = 2^{\lceil \log_2(2\|\mathcal{M}\|_1)\rceil}$, since we want it to be a power of two for fast radix-ρ decomposition.

2.7 PMNS generation

Given a modulo p, one chooses the parameter n. Since $\rho \approx \sqrt[n]{p}$, this gives the approximate size of element coefficients in the PMNS that will be generated. We do not deal with PMNS generation in this paper, this is detailed in [14] for $E(X) = X^n - \lambda$, where $\lambda \in \mathbb{Z} \setminus \{0\}$, and extended in [16] to any monic polynomial $E \in \mathbb{Z}_n[X]$.

In the next two sections, we introduce **GMont-like**, with a set of properties based on it. We will then use these results in Section 5 and subsequent sections, to reduce memory requirement and perform equality test in the PMNS.

3 Montgomery-like internal reduction generalised

The matrix $\mathcal{M}$ (Equation 9) generates the sublattice $\mathcal{L} = \{ZM \bmod E \mid Z \in \mathbb{Z}_{n-1}[X]\}$. As a consequence, Algorithm 2 performs the internal reduction using the points of this sublattice. In this section, we extend this algorithm to **any** sublattice $\mathcal{L}$ of $\mathcal{L}_\mathcal{B}$. This generalised method (we call **GMont-like**) has many properties. We discuss them in this section. Before presenting **GMont-like**, we need to introduce a number of results.

Let $\mathcal{G}$ be a basis of a sublattice $\mathcal{L}$ of $\mathcal{L}_\mathcal{B}$. Let us start with the following remark.

Remark 3.1. Since $\mathcal{L}$ is a sublattice of $\mathcal{L}_\mathcal{B}$, there exists a matrix $\mathbf{T} \in \mathbb{Z}^{n\times n}$ such that $\mathcal{G} = \mathbf{T} \times \mathbf{B}$. We have $\det(\mathbf{B}) = p$ (see Equation 2), so:

$$\det(\mathcal{G}) = tp, \text{ with } t = \det(\mathbf{T}) \in \mathbb{Z} \setminus \{0\}.$$

It is known that $\mathcal{L} = \mathcal{L}_\mathcal{B}$ if and only if $\mathbf{T}$ is an unimodular matrix. As a consequence,

$$\mathcal{L} = \mathcal{L}_\mathcal{B} \iff \det(\mathcal{G}) = \pm p. \qquad (10)$$

For instance, to know if the matrix $\mathcal{M}$ generates $\mathcal{L}_\mathcal{B}$, that is $\mathcal{L}_\mathcal{B} = \{ZM \bmod E \mid Z \in \mathbb{Z}_{n-1}[X]\}$, it is sufficient to check if $\det(\mathcal{M}) = \pm p$.

In the PMNS, computations are performed on polynomials in $\mathbb{Z}_{n-1}[X]$ (or equivalently on vectors in $\mathbb{Z}^n$). As a first step to introduce **GMont-like**, we need to characterise $\mathbb{Z}^n$ elements with respect to $\mathcal{G}$.

3.1 Characterising $\mathbb{Z}^n$ elements

Let $V \in \mathbb{R}^n$. The matrix $\mathcal{G}$ is a basis of $\mathbb{R}^n$ (seen as a vector space over $\mathbb{R}$). So, there exists $(\alpha_0, \alpha_1, \dots, \alpha_{n-1}) \in \mathbb{R}^n$, such that $V = (\alpha_0, \alpha_1, \dots, \alpha_{n-1})\mathcal{G}$. If $V \in \mathbb{Z}^n$, the coefficients α_i have a special shape, which is given in the proposition below. This shape will be a core element to prove the correctness of **GMont-like** and many other properties.

Proposition 3.1. *Let $V \in \mathbb{Z}^n$. There exists $(k_0, k_1, \dots, k_{n-1}) \in \mathbb{Z}^n$ such that:*

$$V = (\frac{k_0}{d}, \frac{k_1}{d}, \dots, \frac{k_{n-1}}{d}).\mathcal{G} ,$$

where $d = |\det(\mathcal{G})|$.

Proof. The matrix $\mathcal{G}$ is also a basis of $\mathbb{R}^n$ (seen as a vector space over $\mathbb{R}$). So, there exists $\beta \in \mathbb{R}^n$, such that $V = \beta.\mathcal{G}$, as $\mathbb{Z}^n \subset \mathbb{R}^n$. Thus, $\beta = V.\mathcal{G}^{-1}$. On the other hand, $\mathcal{G}^{-1} = \frac{1}{\det(\mathcal{G})}\mathbf{G}$, where $\mathbf{G} \in \mathbb{Z}^{n\times n}$ is the adjugate matrix of $\mathcal{G}$. So, $\mathcal{G}^{-1} = \frac{1}{d}\mathbf{J}$, where $\mathbf{J} = \pm\mathbf{G}$ according to the sign of $\det(\mathcal{G})$. Then, $\beta = V.(\frac{1}{d}\mathbf{J}) = \frac{1}{d}K$, where $K = V.\mathbf{J}$. We have $K \in \mathbb{Z}^n$, because $V \in \mathbb{Z}^n$ and $J \in \mathbb{Z}^{n\times n}$. $\square$

From this proposition, we derive the following result.

Corollary 3.1. *Let $V \in \mathbb{Z}^n$. There exist $(\eta_0, \eta_1, \dots, \eta_{n-1}) \in \mathbb{Z}^n$ and $(\beta_0, \beta_1, \dots, \beta_{n-1}) \in \mathbb{N}^n$ such that:*

$$\begin{aligned} V &= (\eta_0 + \frac{\beta_0}{d}, \eta_1 + \frac{\beta_1}{d}, \dots, \eta_{n-1} + \frac{\beta_{n-1}}{d})\mathcal{G} , \\ &= J + S . \end{aligned}$$

where $\begin{cases} J = (\eta_0, \eta_1, \ldots, \eta_{n-1})\mathcal{G} \in \mathcal{L}\,, \\ S = (\frac{\beta_0}{d}, \frac{\beta_1}{d}, \ldots, \frac{\beta_{n-1}}{d})\mathcal{G} \in \mathcal{H}\,. \end{cases}$

Proof. η_i and β_i are respectively the quotient and the remainder of the Euclidean division of k_i by d. Since $\eta_i \in \mathbb{Z}$, we have $J \in \mathcal{L}$. That is, $J(\gamma) \equiv 0 \pmod{p}$; i.e. $V(\gamma) \equiv S(\gamma) \pmod{p}$. β_i is the remainder of the Euclidean division of k_i by d, so $0 \leqslant \beta_i < d$. Thus, $0 \leqslant \frac{\beta_i}{d} < 1$. Hence, $S \in \mathcal{H}$. □

Remark 3.2. From such a polynomial S, one can easily compute $S' \in \mathcal{H}'$, such that $S'(\gamma) \equiv S(\gamma) \pmod{p}$. In fact, it is sufficient to take $S' = (\alpha_0, \alpha_1, \ldots, \alpha_{n-1})\mathcal{G}$,
with $\alpha_i = \begin{cases} \frac{\beta_i}{d} \text{ if } \beta_i < d/2\,, \\ \frac{\beta_i}{d} - 1 \text{ if not.} \end{cases}$
The inverse transformation from $\mathcal{H}'$ to $\mathcal{H}$ is also obvious. Thus, $\#(\mathcal{H} \cap \mathbb{Z}^n) = \#(\mathcal{H}' \cap \mathbb{Z}^n)$.

3.2 Montgomery-like internal reduction generalised

This section presents **GMont-like**. As explained in Remark 3.1, every basis $\mathcal{G}$ of $\mathcal{L}$ is such that $\det(\mathcal{G}) = tp$, where $t \neq 0$ an integer. Lemma 3.1 guarantees the existence of an essential parameter for **GMont-like**.

Lemma 3.1. *Let $\phi \geqslant 2$ be an integer.*
If $\gcd(tp, \phi) = 1$, then $\mathcal{G}' = -\mathcal{G}^{-1} \pmod{\phi}$ exists.

Proof. We have $\mathcal{G}^{-1} = \frac{1}{\det(\mathcal{G})}\mathbf{G} = \frac{1}{tp}\mathbf{G}$, where $\mathbf{G} \in \mathbb{Z}^{n\times n}$ is the adjugate matrix of $\mathcal{G}$. So, $\mathbf{G}^{-1} \pmod{\phi}$ exists. Since $\gcd(tp, \phi) = 1$, $\frac{1}{tp} \pmod{\phi}$ exists as well. □

Algorithm 6 describes **GMont-like**. We assume the parameter ϕ such that $\gcd(tp, \phi) = 1$.

Algorithm 6 Coefficients reduction for PMNS (**GMont-like**)

```
Require: $C \in \mathbb{Z}^n$, $\phi \in \mathbb{N}\backslash\{0, 1\}$, and the matrices
    $\mathcal{G}$ and $\mathcal{G}'$.
Ensure: $S(\gamma) = C(\gamma)\phi^{-1} \pmod{p}$, with $S \in \mathbb{Z}^n$
1: $Q = (c_0, \ldots, c_{n-1})\mathcal{G}' \pmod{\phi}$
2: $T = (q_0, \ldots, q_{n-1})\mathcal{G}$
3: $S \leftarrow (C + T)/\phi$
4: return $S$
```

According to Proposition 3.1, the input C of Algorithm 6 is such that $C = \alpha\mathcal{G}$, with $\alpha_i = \frac{k_i}{|tp|}$ where $k_i \in \mathbb{Z}$. We have $\gcd(tp, \phi) = 1$, so $\alpha_i \pmod{\phi}$ exists. Thus, the vector $(-\alpha) \bmod \phi = ((-\alpha_0) \bmod \phi, \ldots, (-\alpha_{n-1}) \bmod \phi)$ exists.
Additionally, at line 1 of this algorithm, we have:

$$\begin{aligned} Q &= C\mathcal{G}' \pmod{\phi}\,, \\ &= (\alpha\mathcal{G})(-\mathcal{G}^{-1}) \pmod{\phi}\,, \\ &= (-\alpha)(\mathcal{G} \times \mathcal{G}^{-1}) \pmod{\phi}\,, \\ &= (-\alpha) \bmod \phi\,. \end{aligned}$$

As a consequence, the output S is such that:

$$S = \frac{\alpha + ((-\alpha) \bmod \phi)}{\phi}\mathcal{G}\,. \qquad (11)$$

The divisions at line 3 in **GMont-like** are thus exact. For these divisions to be fast, one must take ϕ as a power of 2. So, from Lemma 3.1, we need $\det(\mathcal{G}) \equiv 1 \pmod{2}$; that is, tp should be odd.

A consequence of Equation 11 is the following property. It tells us that if an element is in the fundamental domain $\mathcal{H}$ (see Equation 3), then it remains in $\mathcal{H}$ when **GMont-like** is applied to it.

Property 3.1. *Let $C \in \mathbb{Z}^n$.*
*If $C \in \mathcal{H}$, then **GMont-like**$(C) \in \mathcal{H}$.*

Proof. Assume $C = \alpha\mathcal{G}$. Then, according to Equation 11, **GMont-like**$(C) = \beta\mathcal{G}$, with $\beta_i = \frac{\alpha_i + ((-\alpha_i) \bmod \phi)}{\phi}$.
Remember that $0 \leqslant (-\alpha_i) \bmod \phi < \phi$. Since $C \in \mathcal{H}$, $0 \leqslant \alpha_i < 1$. Thus, $0 \leqslant \beta_i < 1$. Hence, the result. □

Remark 3.3. Using the idea in [16], with Algorithm 6, the bounds given in Section 2.6.4 become:
$$\rho \geqslant 2\|\mathcal{G}\|_1 \text{ and } \phi \geqslant 2w\rho(\delta + 1)^2\,.$$
However, these bounds can be improved. We will see in Section 5 how this can be done.

3.3 GMont-like and PMNS generation

This generalised Montgomery-like internal reduction greatly simplifies the PMNS generation process. Indeed, the process proposed in [16] to find a suitable polynomial M (which is used to compute $\mathcal{M}$) requires a search in a space of size 2^n. This search is very efficient when n is small, which is most of the time the case for modulo sizes in elliptic curve cryptography. However, as shown in [17], n grows very fast as p becomes large. This makes

the search for a suitable polynomial M very expensive. As seen above, **GMont-like** works with any basis $\mathcal{G}$ of any sublattice of $\mathcal{L}_\mathcal{B}$. In particular, it works with any (reduced) basis $\mathcal{G}$ of $\mathcal{L}_\mathcal{B}$.
As mentioned in Remark 2.1, such a basis can be obtained with algorithms like LLL [22], BKZ[23] or HKZ[24], applied to the basis $\mathbf{B}$ (Equation 2). Remember that $\det(\mathbf{B}) = p$, so $\det(\mathcal{G}) = \pm p$. Thus, in this case, ϕ can be taken as a power of 2, assuming p is odd. With this choice, it is no longer necessary to search for a suitable polynomial M.

More generally, being able to choose $\mathcal{G}$ as a basis of any sublattice $\mathcal{L}$ gives the freedom to decide which subset of the polynomials representing zero we want to use to perform coefficient reductions. This might be used to have a 'specialised' (and more efficient) internal reduction method, which would be used after addition and subtraction for instance. It remains an open problem.

A possible disadvantage of not using $\mathcal{M}$ for internal reduction could be the memory required to store PMNS parameters. Indeed, the polynomial M is sufficient to compute the parameters $\mathcal{M}$ and $\mathcal{M}'$ since E is known, whereas in the general case the matrix $\mathcal{G}$ is needed for both $\mathcal{G}'$ computation and **GMont-like**. We have $M \in \mathbb{Z}^n$, whereas $\mathcal{G} \in \mathbb{Z}^{n\times n}$.

3.4 GMont-like and lattice points

Remember that if $J \in \mathcal{L}$, then $J = \alpha\mathcal{G}$, with $\alpha \in \mathbb{Z}^n$. **GMont-like** behaviour on lattice points is quite interesting. It has many properties we discuss in this section.

Let us begin with the invariants. Some elements of $\mathcal{L}$ are invariant for **GMont-like**. That is, applying **GMont-like** on them returns the input. The following property highlights them.

Property 3.2. *Let $J \in \mathcal{L}$, with $J = \alpha\mathcal{G}$.*

$$\textit{\textbf{GMont-like}}(J) = J \iff \forall i,\ \alpha_i \in \{0,1\}\,.$$

Proof. According to Equation 11, **GMont-like**$(J) = (\beta_0, \dots, \beta_{n-1})\mathcal{G}$, where $\beta_i = \frac{\alpha_i + ((-\alpha_i) \bmod \phi)}{\phi}$. So,

$$\begin{aligned}\textbf{GMont-like}(J) = J &\iff \beta_i = \alpha_i\,,\\ &\iff \frac{\alpha_i + ((-\alpha_i) \bmod \phi)}{\phi} = \alpha_i\,,\\ &\iff (\phi-1)\alpha_i = ((-\alpha_i) \bmod \phi)\,,\\ &\iff \alpha_i \in \{0,1\}\,,\end{aligned}$$

since $0 \leqslant (-\alpha_i \bmod \phi) < \phi$. This allows to conclude. □

Notice that a non-lattice point cannot be invariant for **GMont-like**. In fact, such a point represents a non-zero element of $\mathbb{Z}/p\mathbb{Z}$, whereas **GMont-like** induces a factor ϕ^{-1} on the output. In other words: $J \notin \mathcal{L} \Rightarrow$ **GMont-like**$(J) \neq J$.
Property 3.2 leads us to define the **canonical set of zeros**.

Definition 3.1. Given a basis $\mathcal{G}$ of $\mathcal{L}$, the **canonical set of zeros** $\mathcal{O}$ is defined as follows:

$$\mathcal{O} = \{(\alpha_0, \dots, \alpha_{n-1})\mathcal{G} \mid \alpha_i \in \{0,1\}\}\,. \qquad (12)$$

$\mathcal{O}$ elements correspond to the points on the edges of the fundamental parallelepiped $\mathcal{H}$. For instance, if $n = 2$, its elements are the points with coordinates $(0,0)$, $(1,0)$, $(0,1)$ and $(1,1)$ as shown in Figure 1.

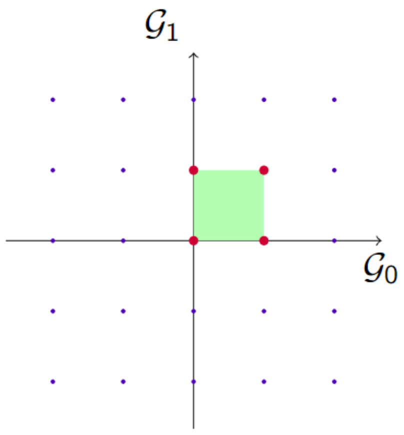

Fig. 1: $\mathcal{O} = \mathcal{H}$ edges

Properties 3.1 and 3.2 complement each other quite elegantly. Indeed, one tells us that points on the edges of $\mathcal{H}$ do not change when **GMont-like** is applied to them, while the other tells us that points inside $\mathcal{H}$ remain inside $\mathcal{H}$ when **GMont-like** is called on them.

Example 3.1. Let $p = 291791$, a 19-bit prime integer. Let $\mathcal{B} = (p, n, \gamma, \rho, E) =$

$(p,\ 2,\ 11810,\ 841,\ X^2-2)$ be a PMNS, with:

$$\mathcal{G} = \begin{pmatrix} 247 & 420 \\ -593 & 173 \end{pmatrix} .$$

Then, $\mathcal{O} = \{(0,0),\ (247,420),\ (-593,173),\ (-346,593)\}$.
Note: To avoid constant reminding of parameters, this PMNS will be used in all examples in this paper. In addition, we will avoid showing unnecessary intermediate computations in the examples. Section 8.2 provides a GitHub link where one can find implementations to check all the examples, as well as the intermediate computations.

Let us now study the effect of **GMont-like** on the output.

Proposition 3.2. *Let $J \in \mathcal{L}$, with $J = (\alpha_0, \dots, \alpha_{n-1})\mathcal{G}$. If $S =$ **GMont-like**(J), then $S = (\beta_0, \dots, \beta_{n-1})\mathcal{G}$, with:*

$$\beta_i = \lceil \frac{\alpha_i}{\phi} \rceil .$$

Proof. Remember that $\alpha_i \in \mathbb{Z}$. According to Equation 11, **GMont-like**$(J) = (\beta_0, \dots, \beta_{n-1})\mathcal{G}$, with:

$$\beta_i = \frac{\alpha_i + ((-\alpha_i) \bmod \phi)}{\phi} .$$

The Euclidean division of α_i by ϕ computes q_i and r_i, such that $\alpha_i = q_i\phi + r_i$, with $0 \leqslant r_i < \phi$ and $q_i = \lfloor \frac{\alpha_i}{\phi} \rfloor$.
Thus, $\beta_i = q_i + \frac{r_i + ((-r_i) \bmod \phi)}{\phi}$.
As a consequence,

- If $\alpha_i \equiv 0 \bmod \phi$, then $r_i = 0$. So, $\beta_i = q_i = \frac{\alpha_i}{\phi} = \lceil \frac{\alpha_i}{\phi} \rceil$.
- If $\alpha_i \not\equiv 0 \bmod \phi$, then $\frac{r_i + ((-r_i) \bmod \phi)}{\phi} = 1$. Thus, $\beta_i = q_i + 1 = \lceil \frac{\alpha_i}{\phi} \rceil$.

□

A direct consequence of this proposition is a condition for a reduction to $\mathcal{O}$.

Corollary 3.2. *Let $J \in \mathcal{L}$, with $J = (\alpha_0, \dots, \alpha_{n-1})\mathcal{G}$. If $\forall i \in \{0, ..., n-1\}$, $-\phi < \alpha_i \leqslant \phi$, then:*

$$\textit{\textbf{GMont-like}}(J) \in \mathcal{O} .$$

Proof. According to Proposition 3.2, we have two cases:

- If $-\phi < \alpha_i \leqslant 0$, then $\beta_i = 0$.
- If $0 < \alpha_i \leqslant \phi$, then $\beta_i = 1$.

So, we always have $\beta_i \in \{0, 1\}$. □

Another consequence of Proposition 3.2 is the following result, which gives a condition for the internal reduction to return the null polynomial. The equality test we present in Section 7 is based on this result.

Corollary 3.3. *Let $J \in \mathcal{L}$, with $J = (\alpha_0, \dots, \alpha_{n-1})\mathcal{G}$. If $\forall i \in \{0, ..., n-1\}$, $-\phi < \alpha_i \leqslant 0$, then:*

$$\textit{\textbf{GMont-like}}(J) = 0 .$$

Proof. Let us assume that $(\beta_0, \dots, \beta_{n-1})\mathcal{G} =$ **GMont-like**(J). According to Proposition 3.2, we have $\beta_i = 0$, $\forall i \in \{0, ..., n-1\}$. So, **GMont-like**$(J) = (0, \dots, 0)\mathcal{G} = 0$. □

Proposition 3.2 tells us how **GMont-like** reduces input coefficients. This reduction is such that a number of successive calls to **GMont-like** on an input will result in an element of $\mathcal{O}$. The following result highlights this number.

Proposition 3.3. *Let $J \in \mathcal{L}$, with $J = (\alpha_0, \dots, \alpha_{n-1})\mathcal{G}$. Let $k \geqslant 0$ be the smaller integer such that $\phi^k > \max_i |\alpha_i|$.*

*If $\dot{J} =$ **GMont-like**$^k(J)$, then $\dot{J} \in \mathcal{O}$.*
Additionally, $\dot{J} = (\beta_0, \dots, \beta_{n-1})\mathcal{G}$, with:

$$\beta_i = \begin{cases} 0 \ \textit{if}\ \alpha_i \leqslant 0 , \\ 1 \ \textit{if}\ \alpha_i \geqslant 1 . \end{cases} \tag{13}$$

Proof. Let us assume that $S = (t_0, \dots, t_{n-1})\mathcal{G} =$ **GMont-like**$^{k-1}(J)$.
Let f be the application defined as follows:

$$\begin{aligned} f : \mathbb{Z} &\to \ \mathbb{Z} , \\ u &\mapsto \lceil \tfrac{u}{\phi} \rceil . \end{aligned}$$

Thus, $t_i = f^{k-1}(\alpha_i)$, according to Proposition 3.2. This application is the composition of the applications $\lceil \cdot \rceil$ and $\frac{\cdot}{\phi}$, which are increasing. In particular, if $u > r\phi$, with $r \in \mathbb{Z}$, then $f(u) > f(r\phi) = r$. Thus, since $\phi^k > \max_i |\alpha_i|$, we have $f^{k-1}(-\phi^k) < t_i \leqslant f^{k-1}(\phi^k)$. That is:

$$-\phi < t_i \leqslant \phi .$$

Additionally,

- if $\alpha_i \leqslant 0$, then $t_i \leqslant 0$, because $f^{k-1}(\alpha_i) \leqslant f^{k-1}(0) = 0$.
- if $\alpha_i \geqslant 1$, then $t_i \geqslant 1$, because $f^{k-1}(\alpha_i) \geqslant f^{k-1}(1) = 1$.

We have $\dot{J} = (\beta_0, \ldots, \beta_{n-1})\mathcal{G} = \textbf{GMont-like}(S)$. So, $\beta_i = f(t_i) = \lceil \frac{t_i}{\phi} \rceil$. As a consequence,

- if $\alpha_i \leqslant 0$, then $-\phi < t_i \leqslant 0$, so $\beta_i = 0$,
- if $\alpha_i \geqslant 1$, then $1 \leqslant t_i \leqslant \phi$, so $\beta_i = 1$.

Hence, $\dot{J} \in \mathcal{O}$ and Equation 13. □

According to Property 3.2, the vector $\dot{J}$ computed in Proposition 3.3 is invariant for **GMont-like**. We define this vector as the **canonical representation** of J.

This proposition provides two ways to compute the canonical representation $\dot{J}$ of a lattice point $J = \alpha\mathcal{G}$. Indeed, if α is known, Equation 13 gives a very simple way to compute $\dot{J}$. However, if only a bound on $\|\alpha\|_\infty$ (equals to $\max_i |\alpha_i|$) is known, i.e. smaller than an integer w, this proposition tells us that:

$$\dot{J} = \textbf{GMont-like}^k(J)\,,$$

with $k = \lceil \log_\phi(w) \rceil$.

With **GMont-like** and some of the properties presented in this section, it is possible to further reduce the coefficients of elements in the PMNS. The next section introduces additional properties that we will use in Section 5 to perform internal reduction to some special domains that allow to control the redundancy and the reduce memory requirement in the PMNS.

4 Reductions to some special domains

As seen in Lemma 2.1, each element $a \in \mathbb{Z}/p\mathbb{Z}$ has at least one representation in the domains $\mathcal{H}$ and $\mathcal{H}'$ (see Equations 3 and 4). With Remarks 2.2 and 3.3, it can be observed that reducing PMNS elements to these domains would further reduce the bound on the parameter ρ. Additionally, we will need the reduction to $\mathcal{H}$ to study the redundancy.

In this section, based on **GMont-like**, we present required properties for coefficient reductions in $\mathcal{H}$ and $\mathcal{H}'$. Before that, we need to introduce the domain $\mathcal{D}_j$, a set that will help us both to improve coefficients bound and to control the redundancy in the PMNS.

4.1 The domain $\mathcal{D}_j$

It is a subset of $\mathbb{R}^n$ that can be seen as an extension of the fundamental region $\mathcal{H}'$.

Definition 4.1. Let $j \geqslant 1$ be an integer. Given a basis $\mathcal{G}$, the domain $\mathcal{D}_j$ is defined as follows:

$$\mathcal{D}_j = \{\sum_{i=0}^{n-1} \mu_i \mathcal{G}_i \;\mid\; -j \leqslant \mu_i < j\,,\ \text{with } \mu_i \in \mathbb{R}\}\,. \tag{14}$$

Two direct consequences of this definition are the following properties.

Property 4.1.

- *If $i < j$, then $\mathcal{D}_i \subset \mathcal{D}_j$.*
- *If $A \in \mathcal{D}_i$ and $B \in \mathcal{D}_j$, then $A + B \in \mathcal{D}_{i+j}$.*

Example 4.1. Let us take $n = 2$. Figure 2 shows the domain $\mathcal{D}_1$ inside the domain $\mathcal{D}_2$.

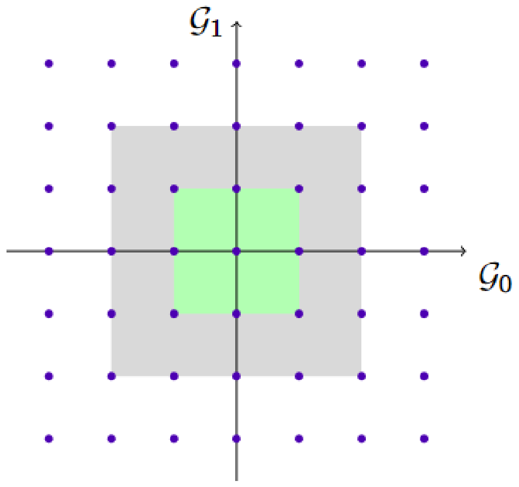

Fig. 2: Domain $\mathcal{D}_1$ inside $\mathcal{D}_2$

Notice that if $A \in \mathcal{D}_j$, then $\|A\|_\infty \leqslant j\|\mathcal{G}\|_1$. In particular, if $A \in \mathcal{D}_1$, then $\|A\|_\infty \leqslant \|\mathcal{G}\|_1$. With the bounds given in Remark 3.3, **GMont-like** performs reductions to a domain $\mathcal{D}_j$ with $j \geqslant 2$, since $\rho \geqslant 2\|\mathcal{G}\|_1$. If we could do the reduction to $\mathcal{D}_1$, it would allow to take $\rho = \|\mathcal{G}\|_1 + 1$, thus reducing the memory required to represent PMNS elements. This section explains how to do this.

Let us first see Lemma 4.1 which ensures that this value for ρ allows to define a PMNS. Then, we explain how to perform the reduction to $\mathcal{D}_1$.

This straightforward result extends Theorem 2.1 to any sublattice of $\mathcal{L}_\mathcal{B}$.

Lemma 4.1. *Let $\mathcal{L}$ be a sublattice of $\mathcal{L}_\mathcal{B}$, having $\mathcal{G}$ as a basis. A tuple $\mathcal{B} = (p, n, \gamma, \rho)$ defines a*

PMNS if:

$$\rho > \frac{1}{2}\|\mathcal{G}\|_1 \,.$$

Proof. According to Lemma 2.1, each element $a \in \mathbb{Z}/p\mathbb{Z}$ has at least one representation $A \in \mathcal{H}'$. Additionally, $A \in \mathcal{H}'$ implies that $\|A\|_\infty \leqslant \frac{1}{2}\|\mathcal{G}\|_1$ (see Remark 2.2). □

According to this lemma, taking $\rho = \frac{1}{2}\|\mathcal{G}\|_1 + 1$ allows to build a PMNS. Since coefficients in this system can be negative, it contains at most $(\|\mathcal{G}\|_1 + 1)^n$ polynomials. Each element $a \in \mathbb{Z}/p\mathbb{Z}$ has at least one representation in the PMNS, thus one must have $p \leqslant (\|\mathcal{G}\|_1 + 1)^n$. The following property, which is a direct consequence of Hadamard's inequality, shows that this is the case.

Property 4.2.

$$\det(\mathcal{G}) \leqslant (\|\mathcal{G}\|_1)^n \,.$$

Proof. Let $c_0, c_1, \ldots, c_{n-1}$ be the columns of $\mathcal{G}$. According to Hadamard's inequality, we have:

$$|\det(\mathcal{G})| \leqslant \prod_{i=0}^{n-1} \|c_i\|_2 \,.$$

Moreover, we have $\|c_i\|_2 \leqslant \|c_i\|_1 \leqslant \|\mathcal{G}\|_1$. □

As mentioned in Remark 3.1, $\det(\mathcal{G}) = tp$, with $t \in \mathbb{Z} \setminus \{0\}$, since it is a basis of the sublattice $\mathcal{L}$. So, $p \leqslant |\det(\mathcal{G})|$. Thus,

$$p \leqslant |\det(\mathcal{G})| \leqslant (\|\mathcal{G}\|_1)^n < (\|\mathcal{G}\|_1 + 1)^n \,.$$

Now, let us consider the following result which gives a condition for a reduction to $\mathcal{D}_1$.

Proposition 4.1. *Let $A \in \mathbb{Z}^n$, with $A = (\alpha_0, \ldots, \alpha_{n-1})\mathcal{G}$. If $\forall i \in \{0, ..., n-1\}$, $-\phi \leqslant \alpha_i \leqslant 0$, then:*

$$\textbf{\textit{GMont-like}}(A) \in \mathcal{D}_1 \,.$$

Proof. Assume $S = (\beta_0, \ldots, \beta_{n-1})\mathcal{G} =$ **GMont-like**(J). According to Equation 11, we have:

$$\beta_i = \frac{\alpha_i + ((-\alpha_i) \bmod \phi)}{\phi} \,.$$

Since $0 \leqslant (-\alpha_i) \bmod \phi < \phi$ and $-\phi \leqslant \alpha_i \leqslant 0$,

$$-\phi \leqslant \alpha_i + ((-\alpha_i) \bmod \phi) < \phi \,.$$

Thus, $-1 \leqslant \beta_i < 1$. Hence, the result. □

Note that Proposition 4.1 requires that the vector α be such that $-\phi \leqslant \alpha_i \leqslant 0, \forall i \in \{0, ..., n-1\}$. To satisfy this requirement, we introduce the **translation vector**.

4.2 The translation vector

Let $\mathcal{B} = (p, n, \gamma, \rho, E)$ be a PMNS. This section first introduces the translation vector $\mathcal{T}$ along with some of its properties.

As mentioned above, the goal is to satisfy the requirement (given in Proposition 4.1) on the input for an internal reduction in $\mathcal{D}_1$, in order to have PMNS elements in $\mathcal{D}_1$. To do this, we will use $\mathcal{T}$ to translate some $\mathbb{Z}^n$ elements to a region in $\mathbb{R}^n$ where they have negative coordinates with respect to the basis $\mathcal{G}$, without changing the values which are represented. This translation is meant to be done in the conversions (to PMNS) and the modular multiplication for outputs in $\mathcal{D}_1$. So, we define the translation vector $\mathcal{T}$ with respect to these operations.

Let us first consider the modular multiplication. Let $A, B \in \mathbb{Z}^n$ be the results of adding at most δ elements of $\mathcal{B}$; so, we have $\|A\|_\infty, \|B\|_\infty \leqslant (\delta + 1)(\rho - 1)$.

Property 4.3. *Let $C = A \times B \bmod E$. Then, $C = \alpha\mathcal{G}$, with $\alpha \in \mathbb{R}^n$ such that:*

$$\|\alpha\|_\infty \leqslant w(\delta + 1)^2(\rho - 1)^2\|\mathcal{G}^{-1}\|_1 \,.$$

Proof. The matrix $\mathcal{G}$ is also a basis of $\mathbb{R}^n$ (seen as a vector space over $\mathbb{R}$). So, there exists $\alpha \in \mathbb{R}^n$ such that $C = \alpha\mathcal{G}$. On the one hand, according to Equation 7, we have:

$$\|C\|_\infty \leqslant w(\delta + 1)^2(\rho - 1)^2 \,.$$

On the other hand, we have $\alpha = C\mathcal{G}^{-1}$. So,

$$\begin{aligned} \|\alpha\|_\infty &= \|C\mathcal{G}^{-1}\|_\infty \,, \\ &\leqslant \|C\|_\infty \|\mathcal{G}^{-1}\|_1 \,, \\ &\leqslant w(\delta + 1)^2(\rho - 1)^2\|\mathcal{G}^{-1}\|_1 \,. \end{aligned}$$

□

Let us now focus on the conversion process, more precisely the fast conversion to PMNS (Algorithm 5). This algorithm requires the polynomials P_i, where each P_i represents $\rho^i\phi^2$. It performs a radix-ρ decomposition, where ρ is assumed to be a power of two for efficiency. Since we want PMNS elements in $\mathcal{D}_1$, we take $\rho = \|\mathcal{G}\|_1 + 1$. Such ρ is unlikely to be a power of two. So, to keep the conversion fast, we need to redefine the polynomials P_i.

Remember that p is the modulo for which the PMNS is built. Let k and β be two integers such that:

$$k = \left\lceil \frac{\log_2(p)}{n} \right\rceil \quad \text{and} \quad \beta = 2^k. \tag{15}$$

Then, $p \leqslant \beta^n$. As a consequence, a radix-β decomposition on n coefficients of any $a \in \mathbb{Z}/p\mathbb{Z}$ is always possible. The polynomials P_i are now computed as representations of $\beta^i \phi^2$, i.e.:

$$P_i(\gamma) \equiv (\beta^i \phi^2) \pmod{p}. \tag{16}$$

As a consequence, the polynomial U computed at line 2 of Algorithm 5 is such that:

$$\|U\|_\infty \leqslant n(\beta - 1)(\rho - 1)\,.$$

As with Property 4.3 for modular multiplication, it implies that:

$$U = \alpha\mathcal{G} \;\text{ with }\; \|\alpha\|_\infty \leqslant n(\beta-1)(\rho-1)\|\mathcal{G}^{-1}\|_1\,. \tag{17}$$

We can now define the translation vector $\mathcal{T}$.

Definition 4.2. Let m and u be two integers such that:

$$m = \max(n(\beta - 1)\,,\, w(\delta+1)^2(\rho - 1))\,,$$

and

$$u = \lceil m(\rho - 1)\|\mathcal{G}^{-1}\|_1 \rceil\,. \tag{18}$$

The **translation vector** $\mathcal{T}$ is defined as follows:

$$\mathcal{T} = (-u, \ldots, -u)\mathcal{G}\,. \tag{19}$$

Example 4.2. In our example of PMNS, $\rho = 841$. The parameter $E(X) = X^2 - 2$. So, $w = 3$.

$$\mathcal{G}^{-1} = \frac{1}{291791}\begin{pmatrix} 173 & -420 \\ 593 & 247 \end{pmatrix}.$$

With $\delta = 0$, we have $u = 5557$.
So, $\mathcal{T} = (-u, -u)\mathcal{G} = (1922722, -3295301)$.

A consequence of Definition 4.2 is the following result.

Property 4.4. *Let $V \in \mathbb{Z}^n$ be a vector. If V is such that:*

$$\|V\|_\infty \leqslant (\rho - 1)\max(n(\beta-1)\,,\, w(\delta+1)^2(\rho-1))\,.$$

Then,

$$V + \mathcal{T} = (\beta_0, \beta_1, \ldots, \beta_{n-1})\mathcal{G}\,,$$

with $-2u \leqslant \beta_i \leqslant 0$.

Proof. This directly comes from how $\mathcal{T}$ is computed. Indeed, as seen with Property 4.3 and Equation 17, we have $V = \alpha\mathcal{G}$, where $\|\alpha\|_\infty \leqslant m(\rho-1)\|\mathcal{G}^{-1}\|_1 \leqslant u$. That is, $-u \leqslant \alpha_i \leqslant u$. Since $\beta_i = \alpha_i - u$, one can conclude. ☐

Notice that $\mathcal{T} \in \mathcal{L}$, since $u \in \mathbb{N}$. So, $\mathcal{T}(\gamma) \equiv 0 \bmod p$. Thus, V and $V + \mathcal{T}$ represent the same element. That is, the translation with $\mathcal{T}$ does not change the value which is represented.

Example 4.3. Let us take $n = 2$ and $u = 2$. Figure 3 shows $\mathcal{T}$ effect on elements in $\mathcal{D}_2$.

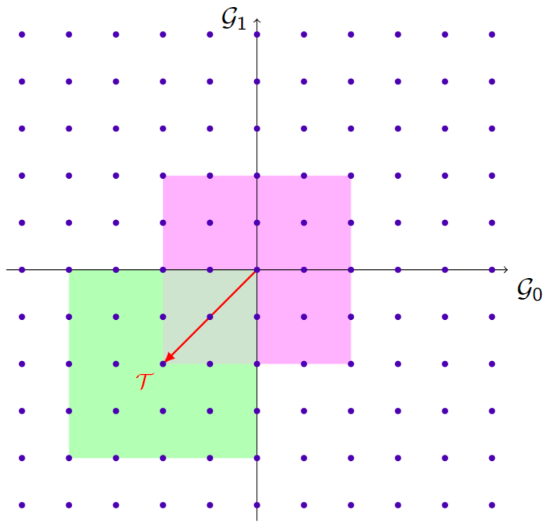

Fig. 3: Domain translation with $\mathcal{T}$

With the translation vector $\mathcal{T}$, we can now discuss the reductions in $\mathcal{D}_1$, $\mathcal{H}$ and then in $\mathcal{H}'$.

4.3 Reduction to $\mathcal{D}_1$

Proposition 4.1 requires that the input A coordinates be negative, with respect to the base $\mathcal{G}$. Thanks to the translation vector and the right bound on ϕ, we can relax this constraint to allow positive coordinates.

Proposition 4.2. *Let $A \in \mathbb{Z}^n$, with $A = (\alpha_0, \dots, \alpha_{n-1})\mathcal{G}$. Let us assume that $\phi \geqslant 2u$. If $\forall i \in \{0, ..., n-1\}$, $-u \leqslant \alpha_i \leqslant u$, then:*

$$\textbf{\textit{GMont-like}}(A + \mathcal{T}) \in \mathcal{D}_1 \,.$$

Proof. We have $A + \mathcal{T} = \beta\mathcal{G}$, with $\phi \leqslant -2u \leqslant \beta_i \leqslant 0$, since $\phi \geqslant 2u$. Proposition 4.1 allows to conclude. $\square$

This proposition has the following consequence.

Corollary 4.1. *Let us assume that $\phi \geqslant 2u$. If $V \in \mathbb{Z}^n$ is a vector such that:*

$$\|V\|_\infty \leqslant (\rho - 1)\max(n(\beta - 1)\,,\, w(\delta + 1)^2(\rho - 1))\,.$$

Then,

$$\textbf{\textit{GMont-like}}(V + \mathcal{T}) \in \mathcal{D}_1 \,.$$

Proof. From Equation 18, $V = (\alpha_0, \dots, \alpha_{n-1})\mathcal{G}$, with $-u \leqslant \alpha_i \leqslant u$. Proposition 4.2 allows to conclude. $\square$

This corollary gives a condition on the input in order the conversion and multiplication outputs in $\mathcal{D}_1$. In Section 5.1, we will see how to modify the corresponding algorithms to take advantage of this result. Let us now see how to perform the reduction to the fundamental regions $\mathcal{H}$ and $\mathcal{H}'$.

4.4 Reduction to $\mathcal{H}$

Let us remind that:

$$\mathcal{H} = \{t \in \mathbb{R}^n \mid t = \sum_{i=0}^{n-1} \mu_i \mathcal{G}_i \ \text{ and}\, 0 \leqslant \mu_i < 1\}\,.$$

The reduction to $\mathcal{H}$ is done on an element already in $\mathcal{D}_1$. So, we assume in this section that:

$$\phi \geqslant 2u\,.$$

The reduction in $\mathcal{H}$ is based on the following result.

Proposition 4.3. *Let $A \in \mathcal{D}_1 \cap \mathbb{Z}^n$. If k is the smallest integer such that $\phi^k > \det(\mathcal{G})$, then:*

$$\textbf{\textit{GMont-like}}^k(A) \in \mathcal{H}\,.$$

Proof. Let $d = |\det(\mathcal{G})|$. Let us assume that $A = \alpha\mathcal{G}$ and consider the following notation:

$$A^{(j+1)} = \textbf{GMont-like}(A^{(j)})\,.$$

So, $A^{(j)} = \textbf{GMont-like}^{(j)}(A)$, with $A = A^{(0)}$. As already mentioned, we have $A^{(j+1)} = \alpha^{(j+1)}\mathcal{G}$, with:

$$\alpha_i^{(j+1)} = \frac{\alpha_i^{(j)} + ((-\alpha_i^{(j)}) \bmod \phi)}{\phi}\,. \tag{20}$$

Notice that if $\alpha_i^{(j)} \in [0, 1[$, then $\alpha_i^{(j+1)} \in [0, 1[$. Since $A \in \mathcal{D}_1$, we have $-1 \leqslant \alpha_i^{(j)} < 1$. Thus, if $\alpha_i^{(j)} \not\equiv 0 \pmod{\phi}$, then $\alpha_i^{(j+1)} \in [0, 1[$, according to Equation 20.

We have $A^{(0)} = A \in \mathbb{Z}^n$, so $\alpha_i^{(0)} = \frac{k_i^{(0)}}{d}$, with $k_i^{(0)} \in \mathbb{Z}$, according to Proposition 3.1. Remember that $\gcd(d, \phi) = 1$ for **GMont-like** (see Section 3.2). Thus, if $\alpha_i^{(j)} \equiv 0 \pmod{\phi}$, then $k_i^{(j)} \equiv 0 \pmod{\phi}$ and $\alpha_i^{(j+1)} = \frac{\alpha_i^{(j)}}{\phi}$. Since $\phi^k > d$, it implies that $-\phi^k < k_i^{(0)} < \phi^k$. As a consequence $\forall i \in \{0, \dots, n-1\}$, there exists a positive integer $t_i < k$, such that $\alpha_i^{(t_i)} \not\equiv 0 \pmod{\phi}$. With Equation 20, it leads to $\alpha_i^{(t_i+1)} \in [0, 1[$. So, $\alpha_i^k \in [0, 1[$, since $k \geqslant t_i + 1$. $\square$

A consequence of this proposition is the following result, which is due to u expression.

Corollary 4.2. *Let $A \in \mathcal{D}_1 \cap \mathbb{Z}^n$. Then,*

$$\textbf{\textit{GMont-like}}^n(A) \in \mathcal{H}\,.$$

Proof. According to Property 4.2, $\det(\mathcal{G}) \leqslant (\|\mathcal{G}\|_1)^n$. Since $\phi > \rho > \|\mathcal{G}\|_1$, we have $\phi^n > \det(\mathcal{G})$. Additionally, if $A \in \mathcal{H}$, then $\textbf{GMont-like}(A) \in \mathcal{H}$. So, one can conclude using Proposition 4.3. $\square$

4.5 Reduction to $\mathcal{H}'$

Remember that:

$$\mathcal{H}' = \{t \in \mathbb{R}^n \mid t = \sum_{i=0}^{n-1} \mu_i \mathcal{G}_i \ \text{ and } -\frac{1}{2} \leqslant \mu_i < \frac{1}{2}\}\,.$$

The main interest of this reduction is to further reduce the memory required to represent PMNS elements, since $A \in \mathcal{H}'$ implies that $\|A\|_\infty \leqslant$

$\frac{1}{2}\|\mathcal{G}\|_1$. This means that the theoretical optimal bound is obtained (see Lemma 4.1).

The reduction to $\mathcal{H}'$ is done on an element already in $\mathcal{H}$. So, we also assume in this section that:

$$\phi \geqslant 2u\,.$$

The main idea of this reduction is the following. Let $A \in \mathcal{H}$, with $A = (\alpha_0, \ldots, \alpha_{n-1})\mathcal{G}$. The polynomial $A' \in \mathcal{H}'$ representing the same element is such that: $A' = (\alpha'_0, \ldots, \alpha'_{n-1})\mathcal{G}$, where

$$\alpha'_i = \begin{cases} \alpha_i & \text{if } \alpha_i < 1/2\,, \\ \alpha_i - 1 & \text{else}\,. \end{cases}$$

GMont-like can be slightly modified to directly apply this idea during the internal reduction. As mentioned before, even if it is not necessary, the parameter ϕ should be taken as a power of two for efficiency. Below, we present an algorithm that needs to check whether an integer $q \in \mathbb{Z}/\phi\mathbb{Z}$ is greater than $\phi/2$ or not. To perform this operation efficiently without conditional branching, we assume that $\phi = 2^h$, with $h \geqslant 1$. In this case, $q < \phi/2 \iff (q >> (h-1)) = 0$.

Algorithm 7 Conditional coeff. reduction (**CMont-like**)

Require: $A \in \mathbb{Z}^n$, $\phi = 2^h$ and the matrices $\mathcal{G}$, $\mathcal{G}'$.
Ensure: $S(\gamma) = A(\gamma)\phi^{-1} \pmod p$, with $S \in \mathbb{Z}^n$

```
1: $Q = (a_0, \ldots, a_{n-1})\mathcal{G}' \pmod \phi$
2: $U \leftarrow Q >> (h-1)$    # so $u_i \in \{0,1\}$
3: $T \leftarrow (q_0, \ldots, q_{n-1})\mathcal{G}$
4: $V \leftarrow (u_0, \ldots, u_{n-1})\mathcal{G}$
5: $R \leftarrow (A + T)/\phi$
6: $S \leftarrow R - V$
7: return $S$
```

Notice that the vector U (line 2, Algorithm 7) is such that $u_i \in \{0, 1\}$. So, the computation of V (line 4) does not involve any multiplication, but only the addition of some rows of $\mathcal{G}$.

Property 4.5. *If $A \in \mathcal{H} \cap \mathbb{Z}^n$, then **CMont-like**$(A) \in \mathcal{H}'$.*

Proof. Remember that we take $\phi = 2^h$. Let us assume that $A = \alpha\mathcal{G}$. As seen in Section 3.2, the vector Q at line 1 is such that $q_i = (-\alpha_i) \bmod \phi$. So, the output S is such that $S = \beta\mathcal{G}$, with:

$$\beta_i = \frac{\alpha_i + ((-\alpha_i) \bmod \phi)}{\phi} - ((-\alpha_i) \bmod \phi >> (h-1))\,.$$

Since $A \in \mathcal{H}$, we have $\alpha_i \in [0, 1[$. As a consequence:

- If $(-\alpha_i) \bmod \phi < \frac{\phi}{2}$, then $(-\alpha_i) \bmod \phi >> (h-1) = 0$. So, $0 \leqslant \beta_i < \frac{1+((-\alpha_i) \bmod \phi)}{\phi}$. Thus, $0 \leqslant \beta_i < \frac{1}{2}$.
- If $(-\alpha_i) \bmod \phi \geqslant \frac{\phi}{2}$, then $(-\alpha_i) \bmod \phi >> (h-1) = 1$. So, $\beta_i = \frac{\alpha_i+((-\alpha_i) \bmod \phi)}{\phi} - 1$. Thus, $-\frac{1}{2} < \beta_i < 0$.

Hence, $S \in \mathcal{H}'$, with $\|S\|_\infty < \frac{1}{2}\|\mathcal{G}\|_1$. □

In the previous section, we saw how to compute a representation $\mathcal{H}$. With Property 4.5, this gives the following result.

Proposition 4.4. *Let $A \in \mathcal{D}_1 \cap \mathbb{Z}^n$. Then,*

$$\textbf{\textit{CMont-like}}(\textbf{\textit{GMont-like}}^n(A)) \in \mathcal{H}'\,.$$

Proof. Let $R =$ **GMont-like**$^n(A)$. According to Corollary 4.2, $R \in \mathcal{H}$. Hence, from Property 4.5, **CMont-like**$(R) \in \mathcal{H}'$. □

With Sections 3 and 4, we introduced a number of theoretical tools to better understand the behaviour of elements in the PMNS. In the following sections, we explain how to use these tools:

- for smaller coefficients size (Section 5),
- to control the redundancy (Section 6),
- to perform the equality test in the PMNS (Section 7).

5 Internal reduction in $\mathcal{D}_1$, $\mathcal{H}$ and $\mathcal{H}'$

In the previous section, we introduced the domain $\mathcal{D}_j$. We explained that the reduction to $\mathcal{D}_1$ would allow the bound on ρ to be reduced from $2\|\mathcal{G}\|_1$ to $\|\mathcal{G}\|_1$, thus allowing to take $\rho = \|\mathcal{G}\|_1 + 1$. We also presented results to do so. In addition, we introduced properties for reduction in the domains $\mathcal{H}$ and $\mathcal{H}'$. The main interest of these domains is to study redundancy in the PMNS, as we will see in Section 6.

To have PMNS elements in $\mathcal{D}_1$, we need the conversion (to PMNS) and the multiplication algorithms to output the results in $\mathcal{D}_1$. In this section, we first see how to do this. Then, given $A \in \mathcal{B}$, we show how to compute an equivalent representation of A in $\mathcal{H}$ and $\mathcal{H}'$.

5.1 Internal reduction to $\mathcal{D}_1$

In order to have PMNS elements in $\mathcal{D}_1$, the internal reduction to $\mathcal{D}_1$ must to be done during the modular multiplication (Algorithm 3) and the conversions in the PMNS (Algorithms 4 and 5). Bellow, we present a slight modification of these algorithms, using the translation vector $\mathcal{T}$.

Algorithm 9 requires the polynomials $P_i \in \mathcal{D}_1$ which are representations of $\beta^i\phi^2$ (see Equation 15 for β). These polynomials are computed using Algorithm 8. Proposition 5.1 ensures an output in $\mathcal{D}_1$ for Algorithms 8 and 9. Notice that Algorithm 8 is also faster than Algorithm 4, since it makes one less call to **GMont-like**. Remark that the precomputed data τ is $\phi^{n-1} \bmod p$, instead of $\phi^n \bmod p$. We remind that **GMont-like** is Algorithm 6, described in Section 3.2.

Algorithm 8 Exact conversion to PMNS with translation

```
Require: $a \in \mathbb{Z}/p\mathbb{Z}$, $\mathcal{B} = (p, n, \gamma, \rho, E)$, $\tau = \phi^{n-1} \bmod p$, the matrices $\mathcal{G}$, $\mathcal{G}'$ and the translation vector $\mathcal{T}$.
Ensure: $A \in \mathcal{D}_1$, such that $A \equiv a_{\mathcal{B}}$
1: $\alpha = a \times \tau \pmod p$
2: $U = (\alpha, 0, \ldots, 0)$ # a polynomial of degree 0
3: for $i = 0 \ldots n-3$ do
4:     $U \leftarrow$ GMont-like$(U)$
5: end for
6: $V \leftarrow U + \mathcal{T}$
7: $A \leftarrow$ GMont-like$(V)$
8: return $A$
```

Proposition 5.1. *If ρ and ϕ are such that:*

$$\rho > \|\mathcal{G}\|_1 \quad \textit{and} \quad \phi \geqslant 2u,$$

then, with $a \in \mathbb{Z}/p\mathbb{Z}$ as input, Algorithms 8 and 9 output a polynomial $A \in \mathcal{D}_1$.

Proof. Let us start with Algorithm 8. Let $\beta \in \mathbb{R}^n$ be the vector such that $U = \beta\mathcal{G}$ (line 2). We have $\beta = U\mathcal{G}^{-1}$, so $\|\beta\|_\infty \leqslant \|U\|_\infty\|\mathcal{G}^{-1}\|_1 < p\|\mathcal{G}^{-1}\|_1 \leqslant (\|\mathcal{G}\|_1)^n\|\mathcal{G}^{-1}\|_1$, since $\alpha < p$ (line 1) and $p \leqslant (\|\mathcal{G}\|_1)^n$ according to Property 4.2.
Also, since $\delta \geqslant 0$, $\rho > \|\mathcal{G}\|_1$ and $w \geqslant n \geqslant 2$, we have $\phi \geqslant 4(\|\mathcal{G}\|_1)^2\|\mathcal{G}^{-1}\|_1$. So, the vector U after the first round of the loop at line 3 is such that $U = \beta\mathcal{G}$, with $\|\beta\|_\infty < (\|\mathcal{G}\|_1)^{n-2}$; notice that this loop is executed only if $n \geqslant 3$.
Remember that $\|\mathcal{G}\|_1\|\mathcal{G}^{-1}\|_1 \geqslant 1$, so $\phi \geqslant 4\|\mathcal{G}\|_1$. Thus, after the next $n-3$ rounds of this loop, U will be such that $\|\beta\|_\infty < \|\mathcal{G}\|_1 < u$. So, according to Property 4.4, the vector V (at line 6) is such that $V = (\nu_0, \nu_1, \ldots, \nu_{n-1})\mathcal{G}$, with $-2u \leqslant \nu_i \leqslant 0$. Since $\phi \geqslant 2u$, we have $-\phi \leqslant \nu_i \leqslant 0$. Thus, Proposition 4.1 ensures that the output $A \in \mathcal{D}_1$.
Let us now consider Algorithm 9. According to Equation 17, the polynomial U (line 2) is such that $\|U\|_\infty \leqslant n(\beta-1)(\rho-1)$. Since $\phi \geqslant 2u$, the output $A \in \mathcal{D}_1$, according to Corollary 4.1. □

Algorithm 9 Fast conversion to PMNS with translation

```
Require: $a \in \mathbb{Z}/p\mathbb{Z}$, $P_i \in \mathcal{D}_1$ such that $P_i(\gamma) \equiv (\beta^i\phi^2) \pmod p$ and the translation vector $\mathcal{T}$.
Ensure: $A \in \mathcal{D}_1$, such that $A \equiv (a\phi)_{\mathcal{B}}$
1: $t = (t_{n-1}, ..., t_0)_\beta$ # radix-$\beta$ decomposition of $a$
2: $U \leftarrow \sum_{i=0}^{n-1} t_i P_i(X)$
3: $W \leftarrow U + \mathcal{T}$
4: $A \leftarrow$ GMont-like$(W)$
5: return $A$
```

Algorithm 10 Modular multiplication with translation (**TransMult**)

```
Require: $A, B \in \mathbb{Z}_{n-1}[X]$, the matrices $\mathcal{G}$, $\mathcal{G}'$, $\mathcal{E}$, $\phi \in \mathbb{N} \setminus \{0,1\}$ and the translation vector $\mathcal{T}$.
Ensure: $S(\gamma) = A \cdot B(\gamma)\phi^{-1} \pmod p$, with $S \in \mathbb{Z}_{n-1}[X]$
1: $C = A \times B$
2: $V = (c_0, \ldots, c_{n-1}) + (c_n, \ldots, c_{2n-2})\mathcal{E}$
3: $R = V + \mathcal{T}$
4: $S \leftarrow$ GMont-like$(R)$
5: return $S$
```

Modular multiplication is applied to inputs that are the result of adding at most δ elements of $\mathcal{B}$. The following proposition gives the bounds for the output to be in $\mathcal{D}_1$.

Proposition 5.2. *Let $A, B \in \mathbb{Z}_{n-1}[X]$ be two polynomials, such that: $\|A\|_\infty, \|B\|_\infty \leqslant (\delta+1)(\rho-1)$. If ϕ is such that:*

$$\phi \geqslant 2u \textit{ (see Equation 18)},$$

then, with A and B as inputs, Algorithm 10 outputs a polynomial $S \in \mathcal{D}_1$.

Proof. The vector V computed at line 2 of this algorithm is such that $V = A \times B \bmod E$. So, $\|V\|_\infty \leqslant w(\delta+1)^2(\rho-1)^2$. Since $\phi \geqslant 2u$, Corollary 4.1 ensures that $S \in \mathcal{D}_1$. □

Remark 5.1. Some observations about these reductions to $\mathcal{D}_1$.

- Since the conversions only require that $\rho > \|\mathcal{G}\|_1$, we will take:

$$\rho = \|\mathcal{G}\|_1 + 1 \quad \text{and} \quad \phi \geqslant 2u, \tag{21}$$

- The additional cost of reducing in $\mathcal{D}_1$ is the addition of the translation vector $\mathcal{T}$. This cost is very small compared to that of **GMont-like**; see Section 8, where operation costs are discussed.

With these algorithms and bounds, $\mathcal{B}$ elements will be in $\mathcal{D}_1$. Thus, to study the redundancy in $\mathcal{B}$, we will study the redundancy in $\mathcal{D}_1$. Before studying this redundancy, let us see how to compute a representation of a given element in the fundamental regions $\mathcal{H}$ and $\mathcal{H}'$.

5.2 Internal reduction to $\mathcal{H}$

Let us remind that:

$$\mathcal{H} = \{t \in \mathbb{R}^n \mid t = \sum_{i=0}^{n-1} \mu_i \mathcal{G}_i \ \text{and}\ 0 \leqslant \mu_i < 1\},$$

We consider the bound on ρ and ϕ, given in Remark 5.1. Corollary 4.2 requires a polynomial $A \in \mathcal{D}_1$. Also, remember that **GMont-like** introduces a factor ϕ^{-1} on the output. So, to compute a representation in $\mathcal{H}$ of an element $A \in \mathcal{B}$, we use a polynomial $\nu \in \mathcal{D}_1$ which represents ϕ^{n+1}, i.e. $\nu(\gamma) \equiv \phi^{n+1} \pmod p$. It can be precomputed using Algorithm 8 or 9. With such a polynomial, a representation in $\mathcal{H}$ can be computed using Algorithm 11.

According to Proposition 5.2, the vector S (line 1, Algorithm 11) is such that $S \in \mathcal{D}_1$. Thus, at the end of the loop (line 4), we have $S \in \mathcal{H}$, according to Corollary 4.2. We have $\mathcal{T} \in \mathcal{L}$, i.e. $\mathcal{T}(\gamma) \equiv 0 \pmod p$. Since **GMont-like** induces a factor ϕ^{-1} on the output and $\nu(\gamma) \equiv \phi^{n+1} \pmod p$, it is clear that $A(\gamma) \equiv S(\gamma) \pmod p$.

Algorithm 11 Representation computation in $\mathcal{H}$

```
Require: $A \in \mathcal{D}_1$, the polynomial $\nu \in \mathcal{D}_1$, such
    that $\nu(\gamma) \equiv \phi^{n+1} \pmod p$.
Ensure: $S \in \mathcal{H}$ and $A(\gamma) \equiv S(\gamma) \pmod p$
1: $S \leftarrow$ TransMult$(A, \nu)$ # Algorithm 10
2: for $i = 0 \ldots n-1$ do
3:     $S \leftarrow$ GMont-like$(S)$
4: end for
5: return $S$
```

Finally, to compute a representation of an element $a \in \mathbb{Z}/p\mathbb{Z}$ in $\mathcal{H}$, one first computes a representation $A \in \mathcal{D}_1$ (using Algorithm 8 or 9) and then apply Algorithm 11 on A.

5.3 Internal reduction to $\mathcal{H}'$

Let us remind that:

$$\mathcal{H}' = \{t \in \mathbb{R}^n \mid t = \sum_{i=0}^{n-1} \mu_i \mathcal{G}_i \ \text{and}\ -\frac{1}{2} \leqslant \mu_i < \frac{1}{2}\}.$$

As mentioned in Section 4.5, the main interest of this reduction is to further reduce the memory required to represent PMNS elements, since $A \in \mathcal{H}'$ implies that $\|A\|_\infty \leqslant \frac{1}{2}\|\mathcal{G}\|_1$. That is, the theoretical optimal bound (given in Lemma 4.1) is obtained. Here also, we consider the bound on ρ and ϕ, given in Remark 5.1. Additionally, remember that for **CMont-like** (Algorithm 7, Section 4.5), we assume that ϕ is a power of two such that $\phi = 2^h$, which is the best case for efficiency.

Proposition 4.4 requires a polynomial $A \in \mathcal{D}_1$. Like **GMont-like**, **CMont-like** induces a factor ϕ^{-1} on the output. To compute a representation in $\mathcal{H}'$ of an element $A \in \mathcal{B}$, we first precompute the polynomial $\xi \in \mathcal{B}$ representing ϕ^{n+2}. Then, Algorithm 12 can be applied.

Algorithm 12 Representation computation in $\mathcal{H}'$

```
Require: $A \in \mathcal{D}_1$, the polynomial $\xi \in \mathcal{D}_1$, such
    that $\xi(\gamma) \equiv \phi^{n+2} \pmod p$.
Ensure: $S \in \mathcal{H}'$ and $A(\gamma) \equiv S(\gamma) \pmod p$
1: $R \leftarrow$ TransMult$(A, \nu)$ # Algorithm 10
2: for $i = 0 \ldots n-1$ do
3:     $R \leftarrow$ GMont-like$(R)$
4: end for
5: $S \leftarrow$ CMont-like$(R)$
6: return $S$
```

According to Proposition 5.2, the vector R (line 1, Algorithm 11) is such that $R \in \mathcal{D}_1$. Hence, $S \in \mathcal{H}'$, according to Proposition 4.4. We have $\mathcal{T}(\gamma) \equiv 0 \pmod{p}$. Since **GMont-like** and **GMont-like** induce a factor ϕ^{-1} on the output and $\xi(\gamma) \equiv \phi^{n+2} \pmod{p}$, it is clear that $A(\gamma) \equiv S(\gamma) \pmod{p}$.

Finally, to compute a representation of an element $a \in \mathbb{Z}/p\mathbb{Z}$ in $\mathcal{H}'$, one first computes a representation $A \in \mathcal{D}_1$ (using Algorithm 8 or 9) and then apply Algorithm 12 on A.

To summarise this section, we have explained how to compute a representation in $\mathcal{D}_1$, $\mathcal{H}$ or $\mathcal{H}'$ of an element $a \in \mathbb{Z}/p\mathbb{Z}$. We also saw how to do modular multiplication for an output in $\mathcal{D}_1$. Let us now study the redundancy in these sets.

6 Redundancy in the PMNS

This section is about the redundancy in $\mathcal{H}$, $\mathcal{H}'$ and $\mathcal{D}_j$ (which includes $\mathcal{D}_1$). We first discuss the number of representations in these sets. Then, we explain how to compute these representations.

6.1 Number of representations

Some elements in $\mathbb{Z}/p\mathbb{Z}$ can have more than one representation in $\mathcal{H} \cap \mathbb{Z}^n$ (or $\mathcal{H}' \cap \mathbb{Z}^n$). Indeed, according to Lemma 2.1, every element $a \in \mathbb{Z}/p\mathbb{Z}$ has **at least** one representation in these sets. This is not precise enough for an exact study of redundancy. The following result gives the condition for representation uniqueness in these sets.

Property 6.1. *If $\mathcal{L} = \mathcal{L}_\mathcal{B}$, then each $a \in \mathbb{Z}/p\mathbb{Z}$ has exactly one representation in $\mathcal{H} \cap \mathbb{Z}^n$, and $\#(\mathcal{H} \cap \mathbb{Z}^n) = p$.*

Proof. From Lemma 2.1, we know that a has at least one representation in $\mathcal{H} \cap \mathbb{Z}^n$. Let $A, A' \in \mathcal{H} \cap \mathbb{Z}^n$ such that: $a = A(\gamma) \pmod{p} = A'(\gamma) \pmod{p}$. So, $(A - A')(\gamma) \equiv 0 \pmod{p}$. That is, $A - A' \in \mathcal{L}_\mathcal{B}$. Since $A, A' \in \mathcal{H}$, we have $A - A' = (\nu_0, \ldots, \nu_{n-1})\mathcal{G}$, with $-1 < \nu_i < 1$ and $\nu_i \in \mathbb{N}$ as $A - A' \in \mathcal{L}_\mathcal{B}$. Thus, $\forall i$, $\nu_i = 0$. As a consequence, $A - A' = 0$, i.e. $A = A'$. So, each $a \in \mathbb{Z}/p\mathbb{Z}$ has exactly one representation in $\mathcal{H} \cap \mathbb{Z}^n$. We have $\#(\mathcal{H} \cap \mathbb{Z}^n) \geqslant p$. If $\#(\mathcal{H} \cap \mathbb{Z}^n) > p$, then at least one element $a \in \mathbb{Z}/p\mathbb{Z}$ would necessarily have more than one representation, which is impossible. Hence, $\#(\mathcal{H} \cap \mathbb{Z}^n) = p$. □

A consequence of this property is also the uniqueness of representation $\mathcal{H}' \cap \mathbb{Z}^n$, with $\#(\mathcal{H}' \cap \mathbb{Z}^n) = p$, if $\mathcal{L} = \mathcal{L}_\mathcal{B}$. This comes from Remark 3.2.

Remark 6.1. $\mathcal{L} = \mathcal{L}_\mathcal{B}$ means that $\mathcal{G}$ is a basis of $\mathcal{L}_\mathcal{B}$. As mentioned in Remark 3.1, this is equivalent to have $\det(\mathcal{G}) = \pm p$. For example, for the classical Montgomery-like internal reduction method (Algorithm 2), one has to check if $\det(\mathcal{M}) = \pm p$ (see Equation 9 for $\mathcal{M}$).

Let us now discuss the redundancy in the domain $\mathcal{D}_j$. This set has an interesting connection with the fundamental region $\mathcal{H}$. This is highlighted in the following result.

Property 6.2. *Let $j \geqslant 1$ be an integer.*
The set $\mathcal{D}_j$ contains exactly $(2j)^n$ times the set $\mathcal{H}$.

Proof. Let $\mathcal{G}_i$ be the i-th row of $\mathcal{G}$. We respectively define $\mathcal{H}^{(i)}$ and $\mathcal{D}_j^{(i)}$ as the fundamental region and the domain j with respect to the i-th coordinate, that is:

$$\mathcal{D}_j^{(i)} = \{\mu\mathcal{G}_i \;\;|\;\; -j \leqslant \mu < j\,,\;\; \text{with } \mu \in \mathbb{R}\}\,,$$

and

$$\mathcal{H}^{(i)} = \{\mu\mathcal{G}_i \;\;|\;\; 0 \leqslant \mu < 1\,,\;\; \text{with } \mu \in \mathbb{R}\}\,.$$

Thus, we have:

$$\mathcal{D}_j^{(i)} = \bigcup_{t \in \mathbb{Z} \cap [-j, j[} \{t\mathcal{G}_i + \mathcal{H}^{(i)}\}\,.$$

That is, $\mathcal{D}_j^{(i)}$ is the union of $\mathcal{H}^{(i)}$ translations by the vectors $t\mathcal{G}_i$, where $t \in \mathbb{Z} \cap [-j, j[$.
If $t_1 \neq t_2$, with $t_1, t_2 \in \mathbb{Z}$, then $\{t_1\mathcal{G}_i + \mathcal{H}^{(i)}\} \cap \{t_2\mathcal{G}_i + \mathcal{H}^{(i)}\} = \emptyset$. So, each domain $\mathcal{D}_j^{(i)}$ contains exactly $2j$ times $\mathcal{H}^{(i)}$.
$\mathcal{D}_j$ is a Cartesian product of $\mathcal{D}_j^{(i)}$; more precisely, we have:

$$\mathcal{D}_j = \mathcal{D}_j^{(0)} \times \mathcal{D}_j^{(1)} \times \cdots \times \mathcal{D}_j^{(n-1)}\,,$$

Also,

$$\mathcal{H} = \mathcal{H}^{(0)} \times \mathcal{H}^{(1)} \times \cdots \times \mathcal{H}^{(n-1)}\,.$$

Thus, $\mathcal{D}_j$ contains exactly $\underbrace{(2j) * (2j) * \cdots * (2j)}_{n \text{ times}}$ times the fundamental domain $\mathcal{H}$. □

Given $a \in \mathbb{Z}/p\mathbb{Z}$, we define $\mathcal{R}_j(a)$ as the set of polynomials that represent a in $\mathcal{D}_j$:

$$\mathcal{R}_j(a) = \{A \in \mathcal{D}_j \cap \mathbb{Z}^n \mid a = A(\gamma) \pmod p\}. \tag{22}$$

The set $\mathcal{R}_j(0)$ is quite easy to compute. Indeed, $\mathcal{R}_j(0) \subset \mathcal{L}_{\mathcal{B}}$. Since $\mathcal{R}_j(0) \subset \mathcal{D}_j$, we have:

$$\mathcal{R}_j(0) = \{(\alpha_0, \dots, \alpha_{n-1})\mathcal{G}, \text{ with } \alpha_i \in \mathbb{Z} \cap [-j, j[\,\}. \tag{23}$$

The following result gives $\mathcal{R}_j(a)$ cardinality and defines it with respect to $\mathcal{R}_j(0)$.

Proposition 6.1. *Let us assume that $\mathcal{L} = \mathcal{L}_{\mathcal{B}}$. Let $a \in \mathbb{Z}/p\mathbb{Z}$. If A is its unique representation in $\mathcal{H} \cap \mathbb{Z}^n$, then:*

$$\mathcal{R}_j(a) = \{A + J \mid J \in \mathcal{R}_j(0)\}, \tag{24}$$

and

$$\#\mathcal{R}_j(a) = (2j)^n. \tag{25}$$

Proof. According to Property 6.1, a has exactly one representation in $\mathcal{H}$. Since $\mathcal{D}_j$ contains exactly $(2j)^n$ times the set $\mathcal{H}$ (according to Property 6.2), we deduce that $\#\mathcal{R}_j(a) = (2j)^n$. In particular, $\#\mathcal{R}_j(0) = (2j)^n$. So, the set $\mathcal{K} = \{A + J \mid J \in \mathcal{R}_j(0)\}$ is such that $\#\mathcal{K} = (2j)^n$. We have $\mathcal{K} \subset \mathcal{D}_j$. Since $B \in \mathcal{K} \Rightarrow a = B(\gamma) \pmod p$, it implies that $\mathcal{K} \subset \mathcal{R}_j(a)$. As a consequence, $\mathcal{R}_j(a) = \mathcal{K}$, because $\#\mathcal{K} = \#\mathcal{R}_j(a) = (2j)^n$. □

According to this proposition, **if $\mathcal{L} = \mathcal{L}_{\mathcal{B}}$, then each element $a \in \mathbb{Z}/p\mathbb{Z}$ has exactly $(2j)^n$ representations in $\mathcal{D}_j$.**
So, each element $a \in \mathbb{Z}/p\mathbb{Z}$ has exactly 2^n representations in the PMNS, with the internal reduction in $\mathcal{D}_1$ (discussed in Section 5.1). In comparison, using the approach presented in [14, 16] where the internal is done in $\mathcal{D}_j$ with $j \geqslant 2$ (see Section 4.1), each element $a \in \mathbb{Z}/p\mathbb{Z}$ has at least 4^n representations in the PMNS.

6.2 Representation computations

Let us now see how to compute $\mathcal{R}_j(a)$, the set of all the representations in $\mathcal{D}_j$ of an element $a \in \mathbb{Z}/p\mathbb{Z}$.

We assume that $\mathcal{L} = \mathcal{L}_{\mathcal{B}}$. With Equation 23, it is easy to pre-compute $\mathcal{R}_j(0)$. To compute the representations of a in $\mathcal{D}_j$, one first computes its unique representation in $\mathcal{H}$, using Algorithm 11. Then, $\mathcal{R}_j(a)$ can be computed using Equation 24.

The following example shows how to compute all the representations of an element in $\mathcal{D}_1$.

Example 6.1. Let us go back to our example of PMNS. We have $p = 291791 = \det(\mathcal{G})$, so $\mathcal{L} = \mathcal{L}_{\mathcal{B}}$.

$$\mathcal{R}_1(0) = \{-593X+346, -173X+593, -420X-247, 0\}$$

The unique representation of $a = 122706$ in $\mathcal{H}$ is $A(X) = 381X - 39$, with:

$$(-39, 381) = (\frac{219186}{291791}, \frac{110487}{291791})\mathcal{G}\,.$$

So, $\mathcal{R}_1(a) = \{-212X + 307,\ 208X + 554,\ -39X - 286,\ 381X - 39\}$. Its unique representation in $\mathcal{H}'$ is $-39X - 286$, with:

$$(-286, -39) = (\frac{-72605}{291791}, \frac{110487}{291791})\mathcal{G}\,.$$

We have now studied redundancy in the PMNS, with a process to compute the representations in $\mathcal{D}_j$ of a given element in $\mathbb{Z}/p\mathbb{Z}$. The next section discusses equality test in the PMNS.

7 Equality test in the PMNS

Many cryptographic applications [18–20, 26] require to perform arithmetic equality test. This operation is however non-trivial in the PMNS, due to its redundancy property. Indeed, since elements can have more than one representation, a simple comparison of two polynomials is not sufficient to conclude that they represent two different elements in $\mathbb{Z}/p\mathbb{Z}$. Until now, the best way to know if two polynomials $A, B \in \mathbb{Z}_{n-1}[X]$ represent the same element, i.e. $A(\gamma) \equiv B(\gamma) \pmod p$, is to first compute $C = A - B$, then evaluate C modulo p to check if $C(\gamma) \equiv 0 \pmod p$. So, a conversion out of the PMNS (the evaluation of C) is required. Such a conversion is not appropriate for some cryptographic implementation, for instance when comparison are used to protect against fault attacks [18]. In this section, we present a method to perform equality test on elements in the PMNS. This method does not require any evaluation. Its cost is discussed in Section 8.1.

The equality test is based on Corollary 3.3. This corollary requires the coordinates of J, with respect to the basis $\mathcal{G}$, to be negative. To fulfil this requirement, we use the translation vector $\mathcal{T}$. Note that having A and B represent the same element is equivalent to having $A - B \in \mathcal{L}$.

Let $l \in \mathbb{N}$ be an integer, such that:

$$l = \frac{1}{2} w(\delta + 1)^2 (\rho - 1)^2 . \qquad (26)$$

Algorithm 13 performs the equality test between two polynomials $A, B \in \mathbb{Z}_{n-1}[X]$ such that $\|A\|_\infty, \|B\|_\infty < l$. The bounds on ρ and ϕ remain the same as those given in Remark 5.1.

Algorithm 13 Equality test in the PMNS

```
Require: $A, B \in \mathbb{Z}_{n-1}[X]$ with $\|A\|_\infty, \|B\|_\infty < l$, and the translation vector $\mathcal{T}$.
Ensure: $S = 0$ if and only if $A(\gamma) \equiv B(\gamma) \pmod p$
1: $C = (A - B) + \mathcal{T}$
2: $S \leftarrow$ GMont-like($C$)
3: return $S$
```

Proposition 7.1. *Let $A, B \in \mathbb{Z}_{n-1}[X]$ be two polynomials, such that $\|A\|_\infty, \|B\|_\infty < l$.*
Then, Algorithm 13 outputs the null polynomial if and only if $A - B \in \mathcal{L}$.

Proof. Notice that the output S is such that $S =$ **GMont-like**$(A - B + \mathcal{T})$.
Let us first assume that $S = 0$. Since $\mathcal{T} \in \mathcal{L}$, we thus have **GMont-like**$(A - B) \equiv 0$. That is, $\frac{A-B}{\phi} \in \mathcal{L}$. Hence, $A - B \in \mathcal{L}$.
Let us now assume that $A - B \in \mathcal{L}$. So, $A - B + \mathcal{T} \in \mathcal{L}$. Since $\|A\|_\infty, \|B\|_\infty < \frac{1}{2} w(\delta + 1)^2 (\rho - 1)^2$, we have $\|A - B\|_\infty < w(\delta + 1)^2 (\rho - 1)^2$. So, as seen in Property 4.3 proof, $A - B = \alpha \mathcal{G}$, with $\|\alpha\|_\infty < w(\delta + 1)^2 (\rho - 1)^2 \|\mathcal{G}^{-1}\|_1 \leqslant u$. Thus, $A - B + \mathcal{T} = \beta \mathcal{G}$, with $-2u < \beta_i < 2u$. Since $\phi \geqslant 2u$, one can conclude that **GMont-like**$(A - B + \mathcal{T}) = 0$, according to Corollary 3.3. $\square$

The bound l on the infinity norms of the inputs is quite large. Indeed, since $w \geqslant 2$ and $\delta \geqslant 0$, we have $l \geqslant (\rho - 1)^2$. Remember that if $A \in \mathcal{B}$, then $\|A\|_\infty \leqslant (\rho - 1)$. Thus, Algorithm 13 applies to elements in the PMNS.

Example 7.1. As seen in the previous example, $u = 5557$ and $\mathcal{T} = (1922722, -3295301)$. Let us take $\phi = 2^{16}$, thus:

$$\mathcal{G}' = \begin{pmatrix} 59709 & 63772 \\ 61473 & 7591 \end{pmatrix} .$$

Let $A(X) = 50X + 623$ and $B(X) = -197X - 217$, be two representations of 7541. Let $C(X) = 55X - 3$, be a representation of 65965.
One can check that **GMont-like**$(A - B + \mathcal{T}) = 0$, whereas **GMont-like**$(A - C + \mathcal{T}) = 372X - 178$. The code provided in Section 8.2 can be used to check these results.

8 Operation costs

In this section, we discuss the theoretical cost of the main algorithms presented in this paper. We also provide a link to a SageMath implementation to generate PMNS, to study the redundancy and to check the equality test.

8.1 Theoretical costs

This section is about the costs of the main algorithms presented in this paper. We use the notations and conventions adopted in [16]. We assume a k-bit processor architecture. So, the basic arithmetic computations are performed on k-bit words. We also assume that the algorithms inputs belong to a PMNS $\mathcal{B}$, such that $\phi = 2^h$, where $h \in \mathbb{N}$ and $1 \leqslant h \leqslant k$. Thus, an element in $\mathcal{B}$ requires nk bits to be represented.

The symbols $\mathcal{M}$ and $\mathcal{A}$ respectively denote the multiplication and the addition of two k-bit integers. Also, $\mathcal{S}_l^i$ and $\mathcal{S}_r^i$ respectively denote a left shift and a right shift of i bits.

We have $\|\mathcal{T}\|_\infty \leqslant u\|\mathcal{G}\|_1 < \phi^2 \leqslant 2^{2k}$. So, adding $\mathcal{T}$ to an element costs $2n\mathcal{A}$.

Table 1 gives the costs of the main algorithms. It is based on the costs given in [16] (Section 8.1). The modular multiplication method costs are based on the efficient polynomials E presented in Table 1 of that paper. For the fast conversion algorithms, we do not take into account the radix-β decomposition of the input. Assuming that β is a power of two (as we suggested, see Equation 15), this decomposition can be done very efficiently.

It can be observed in this table that the modular multiplication costs are similar to the one of the classical Montgomery modular multiplication. In [14] (Section 6.4) and [16] (Section

Algorithm	Cost
Mont-like (Alg. 2) [16]	$2n^2\mathcal{M} + (3n^2 - n)\mathcal{A} + n\mathcal{S}_r^h$
MulMod (Alg. 3) [16]	$3n^2\mathcal{M} + (5n^2 - n - 2)\mathcal{A} + n\mathcal{S}_r^h$
FastConv (Alg. 5) [16]	$3n^2\mathcal{M} + (5n^2 - 3n)\mathcal{A} + n\mathcal{S}_r^h$
GMont-like (Alg. 6)	$2n^2\mathcal{M} + (3n^2 - n)\mathcal{A} + n\mathcal{S}_r^h$
CMont-like (Alg. 7)	$2n^2\mathcal{M} + (4n^2 - n)\mathcal{A} + n\mathcal{S}_r^h + n\mathcal{S}_r^{h-1}$
Mult to $\mathcal{D}_1$ (Alg. 10)	$3n^2\mathcal{M} + (5n^2 + n - 2)\mathcal{A} + n\mathcal{S}_r^h$
Conv to $\mathcal{D}_1$ (Alg. 9)	$3n^2\mathcal{M} + (5n^2 - n)\mathcal{A} + n\mathcal{S}_r^h$
Equ. test (Alg. 13)	$2n^2\mathcal{M} + (3n^2 + 3n)\mathcal{A} + n\mathcal{S}_r^h$

Table 1: Main algorithm costs

8.2), C implementations of PMNS are compared with implementations of the classical Montgomery modular multiplication in GMP [27] and OpenSSL [28] libraries, for both performance and memory requirement. They appear to perform similarly. These PMNS implementations did not take advantage of the high parallelisation capability of this system. Finally, it can be seen that an equality test in the PMNS costs almost an internal reduction.

8.2 Some codes for testing

Using SageMath library [29], we have implemented the methods and algorithms presented in this paper. These implementations are available on GitHub:

https://github.com/arithPMNS/PMNS-and-redundancy

Given a prime p, one can generate PMNS, with the choice of parameters such as n, δ and z. Given a PMNS, this code allows to compute the representation we want for any $a \in \mathbb{Z}/p\mathbb{Z}$. It also allows to perform equality check. This code can be used to check the examples presented in this paper.

9 Conclusion

In this paper, we have presented **GMont-like**, a generalised Montgomery-like internal reduction method. We discussed a number of properties of this method. With **GMont-like**, we also improved the memory requirement to represent PMNS elements. In addition, we explained how to compute element representations in some special domains. This allowed us to make a precise study of redundancy in the PMNS, with a process to compute the desired representation of any element $a \in \mathbb{Z}/p\mathbb{Z}$. Based on some properties of **GMont-like**, we explained how to perform equality test within the PMNS. This for instance removes the need for conversion to the original system in order to apply cryptographic countermeasures that require an arithmetic equality test. Finally, we provided implementations to generate PMNS and to check the work presented in this paper. This work makes the PMNS a more complete system.

References

[1] Didier, L., Dosso, F., Mrabet, N. E., Marrez, J. & Véron, P. Randomization of arithmetic over polynomial modular number system. 26th IEEE Symposium on Computer Arithmetic, ARITH 2019, 199–206.

[2] Negre, C. & Plantard, T. Efficient modular arithmetic in adapted modular number system using lagrange representation (2008). 463–477 (Springer Berlin Heidelberg, 2008).

[3] Koblitz, N. Elliptic curve cryptosystems. *Mathematics of computation* **48** (1987).

[4] Miller, V. S. Use of elliptic curves in cryptography (1985). 417–426 (Springer, 1985).

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

[6] Ducas, L. *et al.* Crystals-dilithium: A lattice-based digital signature scheme. *IACR TCHES* **2018**, 238–268 (2018).

[7] Crandall, R. E. Method and apparatus for public key exchange in a cryptographic system (1992). US Patent 5,159,632.

[8] Solinas, J. A. *Generalized mersenne numbers* (Citeseer, 1999).

[9] El Mrabet, N. & Joye, M. *Guide to pairing-based cryptography* (CRC Press, 2017).

[10] Barrett, P. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. 311– 323 (Springer, Berlin, Heidelberg, 1987).

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

[12] Garner, H. L. The residue number system. *IRE Transactions on Electronic Computers* **EL 8**, 140–147 (1959).

[13] Bajard, J.-C., Imbert, L. & Plantard, T. Modular number systems: Beyond the mersenne family. 159–169 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2005).

[14] Didier, L.-S., Dosso, F. Y. & Véron, P. Efficient modular operations using the Adapted Modular Number System. *Journal of Cryptographic Engineering* 1–23 (2020).

[15] Méloni, N. An Alternative Approach to Polynomial Modular Number System Internal Reduction. *IEEE Transactions on Emerging Topics in Computing* (2022).

[16] Dosso, F. Y., Robert, J.-M. & Véron, P. PMNS for efficient arithmetic and small memory cost. *IEEE Transactions on Emerging Topics in Computing* **10**, 1263–1277 (2022). URL https://hal.science/hal-03768546v1/file/TETC3187786.pdf.

[17] Didier, L., Robert, J., Dosso, F. & Mrabet, N. E. A software comparison of RNS and PMNS. 29th IEEE Symposium on Computer Arithmetic, ARITH 2022.

[18] Blömer, J., Otto, M. & Seifert, J.-P. Sign change fault attacks on elliptic curve cryptosystems. Vol. 4236 of Lecture Notes in Computer Science, 36–52 (Springer, 2006).

[19] Joye, M. Protecting ECC against fault attacks: The ring extension method revisited. Cryptology ePrint Archive, Paper 2019/495 (2019).

[20] Fu, D. E. & Solinas, J. IKE and IKEv2 Authentication Using the Elliptic Curve Digital Signature Algorithm. RFC 4754 (2007).

[21] Bajard, J.-C., Marrez, J., Plantard, T. & Véron, P. On polynomial modular number systems over $\mathbb{Z}/p\mathbb{Z}$. *Advances in Mathematics of Communications* **0**, – (2022).

[22] Lenstra, A. K., Lenstra, H. W. & Lovász, L. Factoring polynomials with rational coefficients. *Mathematische annalen* **261**, 515–534 (1982).

[23] Schnorr, C. & Euchner, M. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Vol. 529 of Lecture Notes in Computer Science, 68–85 (Springer, 1991).

[24] Korkine, A. & Zolotareff, G. Sur les formes quadratiques positives. *Mathematische Annalen* **11**, 242–292 (1877).

[25] Bajard, J., Imbert, L. & Plantard, T. Arithmetic operations in the polynomial modular number system. 206–213, 17th IEEE Symposium on Computer Arithmetic (ARITH-17 2005), 27-29.

[26] Abarzúa, R., Valencia, C. & López, J. Survey for performance and security problems of passive side-channel attacks countermeasures in ECC. Cryptology ePrint Archive, Report 2019/010 (2019).

[27] Granlund, T. & al. GNU multiple precision arithmetic library 6.1.2. https://gmplib.org/.

[28] Project, T. O. Openssl. https://www.openssl.org/.

[29] Stein, W. & al. Sagemath. http://www.sagemath.org/index.html.