# Friendly primes for efficient modular arithmetic using the Polynomial Modular Number System

Fangan Yssouf Dosso<sup>1</sup>, Nadia El Mrabet<sup>1</sup>, Nicolas Méloni<sup>2</sup>, François Palma<sup>2</sup>, Pascal Véron<sup>2</sup>

<sup>1</sup>SAS laboratory, École des Mines de Saint-Étienne, Gardanne, France.
<sup>2</sup>Université de Toulon, Institut de Mathématiques, Toulon, France.
fanganyssouf.dosso@emse.fr, nadia.el-mrabet@emse.fr, nicolas.meloni@univ-tln.fr,
francois.palma@univ-tln.fr, pascal.veron@univ-tln.fr

Abstract. The Polynomial Modular Number System (PMNS) is a non-positional number system designed for modular arithmetic. Its efficiency, both in software and hardware, has been demonstrated for integers commonly used in Elliptic Curve Cryptography [11,26]. In [9,7], the authors introduce specific prime forms that are particularly well-suited for PMNS arithmetic. In this work, we extend their results to a broader class of prime numbers. In practice, our approach yields performance that is competitive with, and in some cases superior to, Pseudo-Mersenne arithmetic. As a result, we expand the set of prime numbers that are well-suited for modular arithmetic. Furthermore, we contribute a database of proof of concept Elliptic Curves constructed with those primes that verify the Brainpool Standard.

Keywords: Modular arithmetic  $\cdot$  Polynomial modular number system  $\cdot$  Internal reduction  $\cdot$  Mersenne primes  $\cdot$  Pseudo-Mersenne primes  $\cdot$  Fermat primes

# 1 Introduction

This work presents some methods to build very efficient PMNS for several classes of (prime) moduli. In so doing, we extend the set of friendly primes for modular arithmetic.

Mersenne primes, of the shape  $p = 2^q - 1$ , are a notable number class due to the fast modular reduction they afford. More precisely, the modular reduction can be done in linear time with regard to the number of multi-precision coefficients. As such, they are used in cryptosystems that allow them with the pre-eminent example being the E-521 curve in Elliptic Curve Cryptography [3].

Unfortunately, the density of such primes leaves much to be desired which is why Pseudo-Mersenne [10], of the shape  $p = 2^n - c$  with small c, and Generalized Mersenne [30] primes, of the shape  $p = 2^{n_1} \pm 2^{n_2} \pm \cdots \pm 2^{n_i} \pm 1$ , are often used instead such as for Curve25519 [6] and secp256k1 [1]. These also allow for linear-time modular reduction.

PMNS were introduced by Bajard et al. [4] as yet another extension of Mersenne numbers. Unfortunately, in the general case, for any prime p, the modular reduction has an at best sub-quadratic time complexity. PMNS-friendly primes that allow for linear-time reduction are known to exist and have been utilized in the context of Elliptic Curve Cryptography [9] as well as for Isogeny-based Cryptography [7]. The main motivation behind this work is to construct a class of primes which includes the ones used in previous works as well as to extend it thanks to new constructions [27]. The result is a prime class with speed in the same ballpark as Pseudo and Generalized Mersenne primes while being much denser. More explicitly, our new class includes any prime of the shape  $p = \frac{\alpha \gamma^n - \lambda}{k}$  with no need for any of those parameters to be powers of 2 or close to powers of 2. Restrictions on each parameter are detailed further in the rest of the paper.

Section 2 gives a background on PMNS while Section 3 explains more recent additions to PMNS usage as well as improved bounds for parameter consistency and corresponding conversion algorithms. Section 4 introduces the new prime class which branches off into two distinct constructions and explains how to convert between the two. Those constructions are detailed further in Sections 5 and 6. In Section 7, we present the cost analysis and benchmarks allowed by this construction. Finally, we conclude in Section 8.

#### $\mathbf{2}$ **Background on PMNS**

The Polynomial Modular Number System (PMNS) is a non-positional number system for modular arithmetic. A PMNS is defined by a tuple  $(p, n, \gamma, \rho, E)$ , where p, n,  $\gamma$  and  $\rho$  are positive non-zero integers and  $E \in \mathbb{Z}[X]$  is a monic polynomial such that  $E(\gamma) \equiv 0 \pmod{p}$ . For consistency, we assume that  $p \ge 3$  and  $n \ge 2$ . Let us start with the notations we will be using throughout this paper.

#### $\mathbf{2.1}$ Some notations and conventions

 $-\mathbb{Z}_k[X]$  is the set of polynomials in  $\mathbb{Z}[X]$  with degrees lower than or equal to k:

$$\mathbb{Z}_k[X] = \{ C \in \mathbb{Z}[X] \mid \deg(C) \leq k \}.$$

- If  $A \in \mathbb{Z}_k[X]$ , we assume that  $A(X) = a_0 + a_1X + \cdots + a_kX^k$  can equivalently be represented as the vector  $A = (a_0, \ldots, a_k) \in \mathbb{Z}^{k+1}$ .
- If  $A = (a_0, ..., a_k)$ , then  $A \pmod{\phi} = (a_0 \pmod{\phi}, ..., a_k \pmod{\phi})$ .
- Let  $A \in \mathbb{Z}^k$  be a vector. The supremum norm of A is defined as:  $||A||_{\infty} = \max_{0 \leq i \leq k-1} |a_i|.$ - Let  $\mathcal{W} \in \mathcal{M}_k(\mathbb{Z})$  be a matrix. The 1-norm of  $\mathcal{W}$  is defined as:
- $\|\mathcal{W}\|_1 = \max_{0 \le j \le k-1} \sum_{i=0}^{k-1} |w_{ij}|.$

#### $\mathbf{2.2}$ PMNS and euclidean lattices

**Definition 1.** Let  $p \ge 3$ ,  $n \ge 2$ ,  $\gamma \in [1, p-1]$  and  $\rho \in [1, p-1]$ . Let  $E \in \mathbb{Z}[X]$  a monic polynomial of degree n, such that  $E(\gamma) \equiv 0 \pmod{p}$ . A PMNS is a set  $\mathcal{B} \subset \mathbb{Z}[X]$  such that : 1.  $\forall A \in \mathcal{B}, \deg(A) < n$ ,

- 2.  $\forall A(X) = \sum_{i=0}^{n-1} a_i X^i \in \mathcal{B}, \ -\rho < a_i < \rho \text{ for all } i,$ 3.  $\forall a \in \mathbb{Z}/p\mathbb{Z}, \ \exists A \in \mathcal{B} \text{ such that } A(\gamma) \equiv a \pmod{p}.$

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:

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

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

$$\mathbf{B} = \begin{pmatrix} p & 0 & 0 & \dots & 0 & 0 \\ t_1 & 1 & 0 & \dots & 0 & 0 \\ t_2 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \ddots & \vdots \\ t_{n-2} & 0 & 0 & \dots & 1 & 0 \\ t_{n-1} & 0 & 0 & \dots & 0 & 1 \end{pmatrix} \xleftarrow{\leftarrow p} \\ \xleftarrow{} X + t_1 \\ \xleftarrow{} X^2 + t_2 \\ \xleftarrow{} X^{2} + t_2 \\ \xleftarrow{} X^{n-2} + t_{n-2} \\ \xleftarrow{} X^{n-1} + t_{n-1} \end{cases}$$
(2)

where  $t_i = (-\gamma^i) \mod p_i$ 

Theorem 1 (from [5]) gives a condition on  $\rho$  for  $\mathcal{B}$  to be a PMNS given p, n and E.

**Theorem 1.** Let  $p \ge 3$ ,  $n \ge 1$  be two integers,  $\gamma \in \mathbb{Z}/p\mathbb{Z} \setminus \{0\}$  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:

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

This theorem applies on any basis  $\mathcal{W}$  of  $\mathcal{L}_{\mathcal{B}}$ . However, for optimization purposes to represent PMNS elements, we want  $\rho$  to be as small as possible. So, in practice,  $\mathcal{W}$  is taken as a reduced basis of  $\mathcal{L}_{\mathcal{B}}$  (the smaller the better). Such a basis can be obtained with algorithms like LLL [21], BKZ [28] or HKZ [20], applied to the basis **B** (Equation 2). Since det(**B**) = p, this results in a parameter  $\rho \approx \sqrt{p}$  [25]. In [12, Lemma 4.1], this theorem is extended to any basis of any sub-lattice of  $\mathcal{L}_{\mathcal{B}}$ .

# 2.3 Arithmetic of PMNS

Let p = 1048573, n = 5,  $\gamma = 238019$ ,  $\rho = 36$  and  $E(X) = X^5 - 2$ . Constructing the basis **B** as described in Equation 2 and reducing it with LLL produces the reduced basis

$$\mathcal{W} = \begin{pmatrix} 5 & 6 & 9 & -2 & -9 \\ 5 & 7 & -3 & 12 & -5 \\ -10 & 5 & 7 & -3 & 12 \\ -13 & -7 & 2 & 0 & -9 \\ 1 & -12 & 14 & 4 & 0 \end{pmatrix}$$

One can verify that  $\|\mathcal{W}\|_1 = 37$  so that  $\rho > \frac{1}{2} \|\mathcal{W}\|_1$  and thus  $\mathcal{B} = (p, n, \gamma, \rho, E)$  is a PMNS. Now let us consider two polynomials in  $\mathcal{B}: A = -9X^4 + 13X^3 - 35X^2 - 25$  and  $B = 24X^4 - 32X^3 + 21X^2 - 2X - 16$ . Their modular multiplication is performed in three steps:

1. a standard polynomial multiplication

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

2. a degree reduction (called external reduction)

$$C = AB \mod E$$
  
= -1217X<sup>4</sup> + 230X<sup>3</sup> + 1235X<sup>2</sup> - 2840X + 3222

Since  $E(\gamma) \equiv 0 \mod p$ , we have  $C(\gamma) \equiv A(\gamma)B(\gamma) \mod p$ . 3. a coefficient reduction (called internal reduction)

$$S = C - (-1211X^{4} + 228X^{3} + 1225X^{2} - 2832X + 3236)$$
  
=  $-6X^{4} + 2X^{3} + 10X^{2} - 8X - 14$   
(with  $-1211\gamma^{4} + 228\gamma^{3} + 1225\gamma^{2} - 2832\gamma + 3236 \equiv 0 \mod p$ )

At the end of the process, with have  $S(\gamma) \equiv A(\gamma)B(\gamma) \mod p$  and  $||S||_{\infty} = 10 < \rho$  so that S is in the PMNS.

More formally, the goal of external reduction is to ensure that the degree of the product of two elements of the PMNS remains less than or equal to n-1 so that the first property of Definition 1 holds. Polynomial division guarantees that there exists a polynomial  $Q(X) \in \mathbb{Z}[X]$  such that A(X)B(X) = Q(X)E(X) + R(X) with deg R(X) < n. Since  $E(\gamma) \equiv 0 \pmod{p}$ , it follows that  $R(\gamma) \equiv A(\gamma)B(\gamma) \pmod{p}$ . (mod p). External reduction involves computing  $A(X)B(X) \mod E(X)$ . In Section 3, we provide a general framework for this operation, which encompasses the methods presented in [13] for  $E(X) = X^n + e_{n-1}X^{n-1} + \cdots + e_1X + e_0$ , and [27] for  $E(X) = \alpha X^n - \lambda$ .

The goal of the internal reduction is to ensure the absolute value of the coefficients of the product of two elements of the PMNS remains smaller than  $\rho$  so that the second property of Definition 1 holds. A general approach is to find a polynomial C' in the lattice  $\mathcal{W}$  close to the reduced product  $C = AB \mod E$  so that the coefficients of C - C' are small. Since  $C' \in \mathcal{W}$  we have  $C'(\gamma) \equiv 0 \mod p$  and  $C(\gamma) - C'(\gamma) \equiv C(\gamma) \mod p$ . In the next section, we focus on another approach that is commonly used to perform internal reduction.

# 2.4 Internal reduction

This paper focuses on the Montgomery-like internal reduction method introduced in [24] by Christophe Negre and Thomas Plantard. The main idea is to add to C a multiple of a polynomial  $M \in \mathcal{W}$  such that all the coefficients of C + QM are divisible by the same integer  $\phi$ . This method (see Algorithm 1) requires three parameters chosen so that  $||C + QM||_{\infty} < \phi\rho$ .

- An integer  $\phi \ge 2$ .

- A polynomial  $M \in \mathbb{Z}_{n-1}[X]$  such that:  $M(\gamma) \equiv 0 \pmod{p}$ .
- A polynomial  $M' \in \mathbb{Z}_{n-1}[X]$  such that:  $M' = -M^{-1} \mod (E, \phi)$ .

Algorithm 1 Coefficient reduction [24]

**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  $M' = -M^{-1} \mod(E, \phi)$ . **Ensure:**  $S(\gamma) = V(\gamma)\phi^{-1} \pmod{p}$ , with  $S \in \mathbb{Z}_{n-1}[X]$ 1:  $Q \leftarrow V \times M' \mod(E, \phi)$ 2:  $T \leftarrow Q \times M \mod E$ 3:  $S \leftarrow (V+T)/\phi$ 4: return S

With  $\phi$  and the polynomials M and M', the authors in [13] introduce the matrices  $\mathcal{M}$  and  $\mathcal{M}'$  (see Equations 3 and 4) in order to make the internal reduction's cost independent of the shape of E. With these matrices, Algorithm 1 is rewritten as Algorithm 2. Notice that  $\mathcal{M}$  generates a sub-lattice of  $\mathcal{L}_{\mathcal{B}}$ .

$$\mathcal{M} = \begin{pmatrix} m_0 \ m_1 \ \dots \ m_{n-1} \\ \vdots \ \vdots \ \vdots \\ \vdots \ \vdots \ \vdots \\ \dots \ \dots \ \dots \end{pmatrix} \overset{\leftarrow}{} \begin{array}{l} M \\ \leftarrow X.M \ \text{mod} \ E \\ \leftarrow X^{n-1}.M \ \text{mod} \ E \end{cases}$$
(3)

$$\mathcal{M}' = \begin{pmatrix} m'_0 \ m'_1 \ \dots \ m'_{n-1} \\ \dots \ \dots \ \dots \\ \vdots \ \vdots \ \vdots \ \vdots \\ \dots \ \dots \ \dots \ \dots \end{pmatrix} \stackrel{\leftarrow}{\leftarrow} M' \\ \leftarrow X.M' \ \text{mod} \ (E, \phi) \\ \leftarrow X^{n-1}.M' \ \text{mod} \ (E, \phi)$$
(4)

Algorithm 2 Coefficients reduction for PMNS (RedCoeff) [13]

**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

# 2.5 Generalized internal reduction and bounds for $\rho$ and $\phi$

In two recent (and independent) works, the above Montgomery-like internal reduction method has been generalized. In [22], the authors show that the matrix  $\mathcal{M}$  in Algorithm 2 can be replaced by any (reduced) basis  $\mathcal{W}$  of  $\mathcal{L}_{\mathcal{B}}$ , with  $\mathcal{M}' = -\mathcal{W}^{-1} \mod \phi$ . In [12], the authors go even further by doing a precise study of the redundancy in the PMNS and showing that  $\mathcal{M}$  can be replaced by any (reduced) basis  $\mathcal{G}$  of any sub-lattice of  $\mathcal{L}_{\mathcal{B}}$ , with  $\mathcal{M}' = -\mathcal{G}^{-1} \mod \phi$ . With this generalization, they also propose a simple way to perform the equality test within the PMNS.

Let  $\mathcal{L}$  be a sub-lattice of  $\mathcal{L}_{\mathcal{B}}$ , having  $\mathcal{G}$  as a basis. As explained in [12], det $(\mathcal{G}) = k \times p$ . If  $gcd(\phi, det(\mathcal{G})) = 1$ , then  $\mathcal{G}' = -\mathcal{G}^{-1} \mod \phi$  exists. Thus, the Montgomery-like internal reduction method can be generalized using any basis  $\mathcal{G}$  of any sub-lattice  $\mathcal{L}$  (see Algorithm 3).

Let A and B be two elements of  $\mathcal{B}$  and  $V = A \times B \mod E$ . In [12], the authors give bounds on  $\rho$  and  $\phi$  to ensure consistency of operations and to guarantee that **GMont-like** (Algorithm 3) outputs from V an element of  $\mathcal{B}$ . They propose to take these parameters such that:

$$\rho = \|\mathcal{G}\|_{1} + 1 \quad \text{and} \quad \phi \ge 2[w(\delta + 1)^{2}\|\mathcal{G}^{-1}\|_{1}\|\mathcal{G}\|_{1}^{2}]$$
(5)

Algorithm 3 Coefficients reduction for PMNS (GMont-like) [12]

**Require:**  $V \in \mathbb{Z}_{n-1}[X], \phi \in \mathbb{N} \setminus \{0, 1\}$ , a (reduced) basis  $\mathcal{G}$  of a sub-lattice of  $\mathcal{L}_{\mathcal{B}}$  such that  $gcd(\phi, det(\mathcal{G})) = 1$ , and  $\mathcal{G}' = -\mathcal{G}^{-1} \mod \phi$ . **Ensure:**  $S(\gamma) = V(\gamma)\phi^{-1} \pmod{p}$ , with  $S \in \mathbb{Z}_{n-1}[X]$ 1:  $Q = (v_0, \ldots, v_{n-1})\mathcal{G}' \pmod{\phi}$ 2:  $T = (q_0, \ldots, q_{n-1})\mathcal{G}$ 3:  $S \leftarrow (V+T)/\phi$ 4: return S

The parameter  $\delta$  has been introduced in [11], designating the maximum number of consecutive additions of elements in  $\mathcal{B}$  that we want to compute before doing a multiplication and an internal reduction. Introduced in [13], the parameter w represents the increase in the coefficients of the result due to the external reduction.

# **3** PMNS with non-monic external reduction polynomials

Let us first recall in this Section a classical result on the product of two polynomials U(X) and T(X) in any ring  $\mathcal{R}[X]$ . Suppose  $U(X) = \sum_{i=0}^{d} u_i X^i$  is a polynomial of degree d and T(X) is a constant polynomial of degree d. Then

$$U(X)T(X) = \sum_{i=0}^{a} u_i X^i T(X) \,.$$

Thus, each coefficient of the polynomial U(X)T(X) can be obtained by computing the following vectormatrix product :

$$(u_0, u_1, \ldots, u_{d-1}, u_d) \mathcal{E}$$
.

where  $\mathcal{E}$  is a precomputed matrix whose rows are composed of the coefficients of  $X^i T(X)$  in  $\mathcal{R}[X]$ , for  $i = 0 \dots, d$ .

Let A(X) and B(X) two elements of  $\mathbb{Z}_{n-1}[X]$  and let

$$C(X) = A(X)B(X)$$
  
=  $c_0 + c_1X + \dots + c_{n-1}X^{n-1} + (c_n + c_{n-1}X + \dots + c_{n-2}X^{n-2})X^n$   
=  $L(X) + U(X)X^n$ .

Then

$$C(X) \mod E(X) = L(X) + (U(X)X^n \mod E(X))$$

As previously detailed the coefficients of  $(U(X)X^n \mod E(X))$  can be computed using the following vector-matrix product

$$(c_n, c_{n-1}, \ldots, c_{n-2})\mathcal{E},$$

where  $\mathcal{E}$  is the matrix whose rows are the coefficients of  $X^{n+i}$  in  $\mathbb{Z}[X]/E(X)$ . Notice that these coefficients are in  $\mathbb{Z}$  when E(X) is a monic polynomial. The whole computation of  $C(X) \mod E(X)$  boils down to computing (as described in [13])

$$(c_0,\ldots,c_{n-1})+(c_n,c_{n-1},\ldots,c_{n-2})\mathcal{E}.$$

# 3.1 Extending the parameters for non-monic E

Let  $E(X) = \alpha X^n + e_{n-1}X^{n-1} + \cdots + e_1X + e_0$  be a non-monic polynomial (i.e.  $\alpha > 1$ ), there is no guarantee that  $X^{n+i} \mod E(X)$  belongs to  $\mathbb{Z}[X]$ . Now, since

$$C(X) \mod E(X) = L(X) + (U(X)X^n \mod E(X)),$$

then

$$\alpha C(X) \mod E(X) = \alpha L(X) + (U(X)\alpha X^n \mod E(X)).$$

Let us consider the matrix  $\mathcal{E}_{\alpha}$  whose rows are the coefficients of  $\alpha X^{n+i} \mod E(X)$ .

$$\mathcal{E}_{\alpha} = \begin{pmatrix} -e_0 & -e_1 & \dots & -e_{n-1} \\ \dots & \dots & \dots & \dots \\ \vdots & \vdots & \vdots & \vdots \\ \dots & \dots & \dots & \dots \end{pmatrix} \xleftarrow{\leftarrow \alpha X^{n+1} \mod E}_{\leftarrow \alpha X^{2n-2} \mod E}.$$
(6)

Since  $\alpha X^n \equiv -(e_{n-1}X^{n-1} + \dots + e_1X + e_0) \mod E(X)$  then

$$\alpha X^{n+1} \equiv -(e_{n-1}X^n + \dots + e_1X^2 + e_0X) \mod E(X),$$

and

$$-e_{n-1}X^{n} \equiv \frac{e_{n-1}}{\alpha}(e_{n-1}X^{n-1} + \dots + e_{1}X + e_{0}).$$

Hence to keep the result in  $\mathbb{Z}[X]/E(X)$ , we need to assume that  $e_{n-1} \equiv 0 \mod \alpha$ . By induction, to guarantee that each row of  $\mathcal{E}_{\alpha}$  belongs to  $\mathbb{Z}^n$ , we need to assume that

$$e_i \equiv 0 \mod \alpha, \text{ for } i = 2, \dots, n-1.$$
(7)

Notice that the matrix  $\mathcal{E}$  previously defined for monic E(X) is equal to  $\mathcal{E}_1$  and that the generalized external reduction process for any monic polynomials ( $\alpha = 1$ ) and any non-monic polynomials satisfying the conditions of Equation 7 can be computed as :

$$R = \alpha(c_0, \dots, c_{n-1}) + (c_n, \dots, c_{2n-2})\mathcal{E}_{\alpha}.$$
(8)

Doing so, one obtains a polynomial  $R \in \mathbb{Z}_{n-1}[X]$  such that:

$$R(\gamma) \equiv \alpha A(\gamma) B(\gamma) \pmod{p}$$

By convention,  $\alpha$  will be positive as we can always take -E instead of E to guarantee it. The case  $E(X) = \alpha X^n - \lambda$  is a specific instance of the aforementioned detailed in [27]. This recent work to appear proposes the following algorithm (Algorithm 4) for external reduction.

# Algorithm 4 External reduction using binomial (bMul) [27, Slide 22]

**Require:**  $A, B \in \mathbb{Z}_{n-1}[X]$  and  $E(X) = \alpha X^n - \lambda$  **Ensure:**  $R(\gamma) \equiv \alpha A(\gamma)B(\gamma) \pmod{p}$ , with  $R \in \mathbb{Z}_{n-1}[X]$ 1:  $C \leftarrow A \times B$ 2: Decompose  $C = L + UX^n$ , with  $\deg(L), \deg(U) < n$ 3:  $R \leftarrow \alpha L + \lambda U$ 4: return R

> Remark 1. As  $R(\gamma) \equiv \alpha A(\gamma)B(\gamma) \pmod{p}$ , the author of [27] recommends using a specific domain for the correctness of the computation. That is to say to compute  $a \times b$ , one should find P(X) and Q(X)such that  $P(\gamma) \equiv \alpha^{-1}a \pmod{p}$  and  $Q(\gamma) \equiv \alpha^{-1}b \pmod{p}$ . This way  $R(X) = \alpha P(X)Q(X) \pmod{E}$ satisfies  $R(\gamma) \equiv \alpha^{-1}a \pmod{p}$ .

From Equation 8, we deduce that

$$\forall i = 0, \dots n - 1, \qquad R_i = \alpha c_i + \sum_{k=0}^{n-2} c_{n+k} (\mathcal{E}_\alpha)_{ki}.$$

Now since C(X) = A(X)B(X), then

$$c_i = \sum_{k=0}^{i} a_k b_{i-k}$$
 for  $0 \le i \le n-1$  and  $c_{n-1+i} = \sum_{k=i}^{n-1} a_k b_{n-1+i-k}$  for  $1 \le i \le n-1$ .

Hence

 $c_i \leq (i+1) \|A\|_{\infty} \|B\|_{\infty}$  for  $0 \leq i \leq n-1$  and  $c_{n-1+i} \leq (n-i) \|A\|_{\infty} \|B\|_{\infty}$  for  $1 \leq i \leq n-1$ .

It follows that

$$\forall i = 0, \dots, n-1, \quad R_i \leq |\alpha|(i+1) + \sum_{k=0}^{n-2} (n-k-1)(\mathcal{E}'_{\alpha})_{ki},$$

 $\mathcal{E}'_{\alpha}$  being the  $(n-1) \times n$  matrix such that  $(\mathcal{E}'_{\alpha})_{ij} = |(\mathcal{E}_{\alpha})_{ij}|$ . This last result gives that

$$\|R\|_{\infty} \leqslant w_{\alpha} \|A\|_{\infty} \|B\|_{\infty} , \qquad (9)$$

with:

$$w_{\alpha} = \||\alpha|(1, 2, \dots, n) + (n - 1, n - 2, \dots, 1)\mathcal{E}'_{\alpha}\|_{\infty}.$$
 (10)

This generalizes the result given in [13, Proposition 2].

Remark 2. For  $E(X) = \alpha X^n - \lambda$  we have

$$w_{\alpha} = \max(\alpha n, \alpha + (n-1)|\lambda|).$$
(11)

# 3.2 Improvement on bounds

As mentioned in Section 2.5, it has been proposed in [12] to take  $\rho = \|\mathcal{G}\|_1 + 1$ . In this section, we provide a slight improvement of this value and give a bound on  $\phi$  that is somewhat more precise than that of [13, Proposition 3]. From now on, we consider the assumption in Equation 7. Let A and B be two elements of  $\mathcal{B}$  and  $V = A \times B \pmod{E}$ . From Equation 9, we have that:

$$||V||_{\infty} \leq w_{\alpha} ||A||_{\infty} ||B||_{\infty} \leq w_{\alpha} (\rho - 1)^{2}.$$

The output of Algorithm 3 will be in the PMNS if :

$$\frac{\|V+T\|_{\infty}}{\phi} \leqslant \rho - 1\,,$$

Now in Algorithm 3:

$$||T||_{\infty} < \phi ||\mathcal{G}||_1.$$

Therefore:

$$\frac{\|V+T\|_{\infty}}{\phi} < \frac{1}{\phi} (w_{\alpha}(\rho-1)^2 + \phi \|\mathcal{G}\|_1).$$

Traditionally, the mod  $\phi$  reduction is done by taking the value between 0 and  $\phi - 1$ , but we could just as easily take the value between  $-\frac{\phi}{2}$  and  $\frac{\phi}{2} - 1$ .

By making use of signed arithmetic, the output of Algorithm 3 will be in the PMNS if:

$$\frac{w_{\alpha}(\rho-1)^2}{\phi} + \frac{1}{2} \|\mathcal{G}\|_1 \leqslant \rho \,,$$

that is to say, if

$$w_{\alpha}(\rho-1)^2 \leqslant \phi(\rho-\frac{1}{2}\|\mathcal{G}\|_1)$$

This means we can then take  $\rho = \|\mathcal{G}\|_1 - 1$  for its minimal value. Indeed we have:

$$\frac{w_{\alpha}(\rho-1)^2}{\rho-\frac{\|\mathcal{G}\|_1}{2}} \leqslant \phi.$$

We take the derivative to find the minimum value:

$$\frac{w_{\alpha}(\rho-1)(\rho-\|\mathcal{G}\|_{1}+1)}{(\rho-\frac{\|\mathcal{G}\|_{1}}{2})^{2}}$$

This gives us a minimal value for  $\rho = \|\mathcal{G}\|_1 - 1$  which we can now set as our new optimal parameter  $\rho$ . The bound on  $\phi$  thus becomes:

$$2w_{\alpha}(\rho-1) \leqslant \phi. \tag{12}$$

instead of [13, Proposition 3]

 $2w_{\alpha}\rho \leqslant \phi$ .

Remark 3. If we take into account the parameter  $\delta$  used on page 5, our bound becomes  $2w_{\alpha}(\|\mathcal{G}\|_1 - 2)(\delta + 1)^2 \leq \phi$  instead.

Remark 4. Regarding the bound on  $\phi$  in Equation 5, page 4, it translates to  $\phi \ge 2\lceil w_{\alpha}(\rho-1) \|\mathcal{G}^{-1}\|_1 \|\mathcal{G}\|_1 (\delta+1)^2 \rceil$ , which is greater or equal than  $2w_{\alpha}(\rho-1)(\delta+1)^2$  since  $\|\mathcal{G}^{-1}\|_1 \|\mathcal{G}\|_1 \ge 1$ . Nevertheless, the bound on  $\phi$  in Equation 5 enables performing an equality test within the PMNS and provides precise control over redundancy (see next section).

# 3.3 Alternative conversion algorithms

In previous works on PMNS, authors generally take  $\rho$  as a power of 2 in order to use fast conversion algorithms, usually the smallest power of 2 above the optimal value of  $\rho$ . However, in [12], the authors propose a fast conversion algorithm that doesn't rely on the parameter  $\rho$  being any specific shape, allowing it to be taken as its optimal value.

In [12, Section 4.2], the authors introduced the translation vector, a well-chosen point of  $\mathcal{L}$  which allows to perform the internal reduction to space domains where the redundancy in the PMNS is precisely known and controlled. The use of this vector also makes it possible to carry out an equality test between the elements of a PMNS, which was previously an open problem. When fine control of redundancy or equality testing is not required, as in this paper, some of the algorithms presented in [12] can be slightly modified to enhance either efficiency or memory usage. This section presents alternative algorithms to conversion methods presented in [12, Algorithms 8, 9]. First, we modify the exact conversion algorithm from [12, Algorithm 8]. Adding an extra iteration to the for loop, we can obtain better bounds on the supremum norm of the output. This is needed for precomputed values of the fast conversion algorithm.

# Algorithm 5 Exact Conversion to PMNS

**Require:**  $a \in \mathbb{Z}/p\mathbb{Z}$ ,  $\mathcal{B} = (p, n, \gamma, \rho, E)$  and the matrices  $\mathcal{G}$  and  $\mathcal{G}'$ . **Ensure:**  $A \in \mathcal{B}$  with  $A(\gamma) \equiv a \pmod{p}$ . 1:  $\tau \leftarrow a \times \phi^n \pmod{p}$ 2:  $A \leftarrow (\tau, 0, \dots, 0) \ \#$  a vector of dimension n3: **for**  $i = 0 \dots n$  **do** 4:  $A \leftarrow \mathbf{GMont-like}(A)$ 5: **end for** 6: **return** A

**Proposition 1.** Algorithm 5 properly outputs a polynomial  $A \in \mathcal{B}$  such that  $||A||_{\infty} \leq \frac{||\mathcal{G}||_1}{2} + 1$  and  $A(\gamma) \equiv a \pmod{p}$ .

*Proof.* Let us denote  $A_i$  the value A will have at the iteration i of the loop in Algorithm 5. By construction we get that  $A_0(\gamma) \equiv a \times \phi^n \pmod{p}$ . Each iteration involves applying **GMont-like** and thus dividing by  $\phi$  so we naturally get that  $A_i(\gamma) \equiv a \times \phi^{n-i} \pmod{p}$ . Since we apply it n times we get  $A(\gamma) \equiv a \pmod{p}$ .

As for the bounds on A, we get that  $||A_0||_{\infty} < p$  since  $\tau < p$ . Furthermore, for a given iteration i, we will have that  $||A_{i+1}||_{\infty} \leq \frac{||A_i||_{\infty} + \frac{\phi}{2}||\mathcal{G}||_1}{\phi}$  from the application of **GMont-like**. If we simplify we get  $||A_{i+1}||_{\infty} \leq \frac{||A_i||_{\infty}}{\phi} + \frac{||\mathcal{G}||_1}{2}$ . If we go one iteration step further we get  $||A_{i+2}||_{\infty} \leq \frac{||A_{i+1}||_{\infty}}{\phi} + \frac{||\mathcal{G}||_1}{2} = \frac{||A_i||_{\infty}}{\phi} + \frac{||\mathcal{G}||_1}{2}$  which simplifies to  $||A_{i+2}||_{\infty} \leq \frac{||A_i||_{\infty}}{\phi^2} + \frac{||\mathcal{G}||_1}{2\phi} + \frac{||\mathcal{G}||_1}{2}$ . Because of our bounds from Section 3.2,  $\frac{||\mathcal{G}||_1}{\phi} < 1$ . Taken to its natural conclusion we can bound  $A_n$  in the following way:

$$||A_n||_{\infty} < \frac{||A_0||_{\infty}}{\phi^n} + \frac{||\mathcal{G}||_1}{2} + 1.$$

Hence

$$||A_n||_{\infty} < \frac{p}{\phi^n} + \frac{||\mathcal{G}||_1}{2} + 1.$$

By construction, we have that  $p < \phi^n$  therefore  $\frac{p}{\phi^n} < 1$  so:

$$\|A\|_{\infty} < \frac{\|\mathcal{G}\|_1}{2} + 2$$

Thus

$$\|A\|_{\infty} \leqslant \frac{\|\mathcal{G}\|_1}{2} + 1.$$

 $||A||_{\infty} < \rho.$ 

Since  $\rho = \|\mathcal{G}\|_1 - 1$  so we also have:

So  $A \in \mathcal{B}$ .

Remark 5. Algorithm 12 in [12, Section 5.3] ensures the output result in lattice fundamental domain  $\mathcal{H}' = \{t \in \mathbb{R}^n \mid t = \sum_{i=0}^{n-1} \mu_i \mathcal{G}_i \text{ and } -\frac{1}{2} \leq \mu_i < \frac{1}{2}\}$ , where the redundancy can be controlled and every element  $A \in \mathcal{H}'$  is such that  $||A||_{\infty} \leq \frac{1}{2} ||\mathcal{G}||_1$ . So, unlike Algorithm 5, the reached space domain is known with a slightly better bound. However, it is slower.

Algorithm 5 is slow and mostly useful for pre-computations used in Algorithm 6 although it does give an unconditional low-norm representative in a PMNS. In practice, for fast conversion, we will instead use the following adapted from [12, Algorithm 9]:

# Algorithm 6 Fast Conversion to PMNS

**Require:**  $a \in \mathbb{Z}/p\mathbb{Z}$ ,  $\mathcal{B} = (p, n, \gamma, \rho, E)$ , with E's leading coefficient being noted as  $\alpha$ , the matrices  $\mathcal{G}$  and  $\mathcal{G}', \Theta$  the smallest power of 2 such that  $\Theta^n > p$  and  $P_i \in \mathcal{B}$  such that  $P_i(\gamma) \equiv \alpha^{-1}\Theta^i\phi^2 \pmod{p}$  and  $\forall i, \|P_i\|_{\infty} \leq \frac{\|\mathcal{G}\|_1}{2} + 1$ . **Ensure:**  $A \in \mathcal{B}$  with  $A(\gamma) \equiv \alpha^{-1}a\phi \pmod{p}$ 1:  $t \leftarrow (t_{n-1}, \ldots, t_0) \ \# \Theta$ -radix decomposition of a2:  $U \leftarrow \sum_{i=0}^{n-1} t_i P_i(X)$ 3:  $A \leftarrow \mathbf{GMont-like}(U)$ 4: return A

**Proposition 2.** Algorithm 6 properly outputs a polynomial A such that  $A(\gamma) \equiv \alpha^{-1}a\phi \pmod{p}$  and furthermore, if  $\Theta < \frac{2w_{\alpha}}{n} (\|\mathcal{G}\|_1 - 7 + \frac{6}{(\|\mathcal{G}\|_1 + 2)})$ , then  $\|A\|_{\infty} < \rho$ .

*Proof.* Because t is the  $\Theta$ -radix decomposition of a, we can write a as  $a = \sum_{i=0}^{n-1} t_i \Theta^i$ . By construction,

we get that  $U(\gamma) \equiv \sum_{i=0}^{n-1} t_i \Theta^i \alpha^{-1} \phi^2 \equiv \alpha^{-1} a \phi^2 \pmod{p}$ . Since **GMont-like** applies a division by  $\phi$ ,  $A(\gamma) \equiv \alpha^{-1} a \phi \pmod{p}$ .

Now  $\forall i, \|P_i\|_{\infty} \leq \frac{\|\mathcal{G}\|_1}{2} + 1$  and  $\|t\|_{\infty} \leq \Theta - 1$  by construction therefore  $\forall i, \|t_i P_i\|_{\infty} \leq (\Theta - 1)(\frac{\|\mathcal{G}\|_1}{2} + 1)$ . It follows that  $\|U\|_{\infty} \leq n(\Theta - 1)(\frac{\|\mathcal{G}\|_1}{2} + 1)$ .

After applying **GMont-like** we get  $||A||_{\infty} \leq \frac{n(\Theta-1)(\frac{||\mathcal{G}||_1}{2}+1)+\frac{\phi||\mathcal{G}||_1}{2}}{\phi}$  which gives us:

$$||A||_{\infty} \leqslant \frac{n(\Theta - 1)(\frac{||\mathcal{G}||_1}{2} + 1)}{\phi} + \frac{||\mathcal{G}||_1}{2}.$$
(13)

Our bounds regarding  $\phi$  and  $\rho$  give us that  $\phi > 2w_{\alpha}\rho$  and the optimal value of  $\rho$  has been noted to be  $\|\mathcal{G}\|_1 - 1$  hence:

$$||A||_{\infty} \leqslant \frac{n(\Theta - 1)(\frac{||\mathcal{G}||_{1}}{2} + 1)}{2w_{\alpha}(||\mathcal{G}||_{1} - 1)} + \frac{||\mathcal{G}||_{1}}{2}.$$

Furthermore, we wish to obtain for all inputs  $||A||_{\infty} \leq \rho - 1$ , which is verified if:

$$\|\mathcal{G}\|_{1} - 2 \ge \frac{n(\Theta - 1)(\frac{\|\mathcal{G}\|_{1}}{2} + 1)}{2w_{\alpha}(\|\mathcal{G}\|_{1} - 1)} + \frac{\|\mathcal{G}\|_{1}}{2}.$$

Thus, the necessary condition becomes:

$$\frac{\|\mathcal{G}\|_{1}}{2} \ge \frac{n(\Theta - 1)(\frac{\|\mathcal{G}\|_{1}}{2} + 1)}{2w_{\alpha}(\|\mathcal{G}\|_{1} - 1)} + 2$$
  

$$\iff \|\mathcal{G}\|_{1} - 1 \ge \frac{n(\Theta - 1)(\|\mathcal{G}\|_{1} + 2)}{2w_{\alpha}(\|\mathcal{G}\|_{1} - 1)} + 3$$
  

$$\iff (\|\mathcal{G}\|_{1} - 1)(\|\mathcal{G}\|_{1} - 4) \ge \frac{n(\Theta - 1)(\|\mathcal{G}\|_{1} + 2)}{2w_{\alpha}}$$
  

$$\iff 2w_{\alpha}(\|\mathcal{G}\|_{1} - 4 - 3 + \frac{6}{(\|\mathcal{G}\|_{1} + 2)}) \ge n(\Theta - 1)$$

If we isolate  $\Theta$  we get:

$$\frac{2w_{\alpha}}{n}(\|\mathcal{G}\|_{1} - 7 + \frac{6}{(\|\mathcal{G}\|_{1} + 2)}) > \Theta.$$
(14)

**Proposition 3.** Let  $\mathcal{B} = (p, n, \gamma, \rho, E)$  a PMNS and let  $\ell$  such that  $2^{\ell-1} . Equation 14 is verified if <math>w_{\alpha} \geq \frac{3}{2}n$  and  $\ell > n \log_2(21) + 1$ .

*Proof.* From [12, Property 4.2], we have  $|\det(\mathcal{G})| \leq (||\mathcal{G}||_1)^n$ . Since  $p \leq |\det(\mathcal{G})|$ , we get that  $\sqrt[n]{p} \leq ||\mathcal{G}||_1$ . We have  $2^{\ell-1} and <math>\Theta$  being the smallest power of 2 such that  $\Theta^n > p$ , we get that  $2^{\frac{\ell}{n}} < \Theta < 2^{\frac{\ell+n-1}{n}}$  and equation 14 becomes:

$$3(2^{\frac{\ell-1}{n}} - 7 + \frac{6}{(\|\mathcal{G}\|_{1} + 2)}) > 2^{\frac{\ell+n-1}{n}}$$
  
$$\iff 3 \times 2^{\frac{\ell-1}{n}} - 2 \times 2^{\frac{\ell-1}{n}} - 21 + \frac{18}{(\|\mathcal{G}\|_{1} + 2)} > 0$$
  
$$\iff 2^{\frac{\ell-1}{n}} - 21 + \frac{18}{(\|\mathcal{G}\|_{1} + 2)} > 0$$

Since  $\frac{18}{(\|\mathcal{G}\|_{1}+2)} > 0$ , the inequality is satisfied as soon as:

$$2^{\frac{\ell-1}{n}} > 21$$

$$\iff \frac{\ell-1}{n} > \log_2(21)$$

$$\iff \ell > n \log_2(21) + 1$$

Remark 6. For the vast majority of PMNS, this inequality will be verified. However, for the rare exceptions, one can increase  $\rho$  in consequence so long as  $2w_{\alpha}(\rho-1)(\delta+1)^2 \leq \phi$ . If the bounds allow for  $\rho$  to be the smallest power of 2 bigger than  $\|\mathcal{G}\|_1 - 1$  (similar to what was done in previous works) then we can take  $\Theta = \rho$  and **Algorithm 6** will therefore correctly output an element whose coefficients are absolutely bounded by  $\rho$ . From equation 13:

$$\begin{split} \|A\|_{\infty} &\leqslant \frac{n(\rho-1)(\frac{\|\mathcal{G}\|_{1}}{2}+1)}{2w_{\alpha}(\rho-1)} + \frac{\|\mathcal{G}\|_{1}}{2} \\ &\iff \|A\|_{\infty} \leqslant \frac{n(\frac{\|\mathcal{G}\|_{1}}{2}+1)}{2w_{\alpha}} + \frac{\|\mathcal{G}\|_{1}}{2} \\ &\iff \|A\|_{\infty} \leqslant \frac{n}{w_{\alpha}}(\frac{\|\mathcal{G}\|_{1}}{4}+\frac{1}{2}) + \frac{\|\mathcal{G}\|_{1}}{2} \leqslant \frac{\|\mathcal{G}\|_{1}}{4} + \frac{1}{2} + \frac{\|\mathcal{G}\|_{1}}{2} \leqslant \frac{3}{4}\|\mathcal{G}\|_{1} + \frac{1}{2} \\ &\iff \|A\|_{\infty} \leqslant \|\mathcal{G}\|_{1} - 1 \leqslant \rho \end{split}$$

# 4 PMNS-friendly primes

The internal reduction is one of the most investigated aspects of PMNS because of how critical it is to the overall time complexity of operations. In the general case, an at best subquadratic amount of operations (in the degree of the polynomial to reduce) is expected but the authors of [9] and [7] have noted that specific prime shapes allow for linear-time internal reduction with sparse polynomials Mand M', using **Algorithm 1**. This was actually already observed for the NIST prime P521 in an earlier version of [11] (see Remark 11 and Appendix A of the arXiv preprint). In this section, we show that the prime class covered by those two methods are the same and we introduce the shape of an even broader class of primes that allow for linear-time reduction.

In [7], the authors consider the PMNS within the context of SIDH arithmetic. They explain a method for linear time internal reduction for any PMNS  $\mathcal{B} = (p, n, \gamma, \rho, E = X^n - \lambda)$  whenever  $M(X) = \frac{\gamma}{\lambda} X^{n-1} - 1$ . From [7, Theorem 11], if  $\gamma^2 \equiv 0 \pmod{\phi}$ , then M' will also be sparse. From [7, Theorem 13] the shape of M implies  $\gamma < \rho$ .

Meanwhile, in [9], the authors implement PMNS for ECC and, in [9, Section II.D], remark that for any Mersenne and some Generalized Mersenne numbers, one can achieve linear time internal reduction for a PMNS  $\mathcal{B} = (p, n, \gamma, \rho, E = X^n - \lambda)$  using  $M(X) = \gamma' X - 1$  and  $M'(X) = \gamma' X + 1$ , whenever  $\gamma'^2 \equiv 0 \pmod{\phi}$  and ([9, Equation 1])  $\gamma' < \rho$  with  $\gamma', \gamma$ 's modular inverse in  $\mathbb{Z}/p\mathbb{Z}$ .

We will now show that any prime that satisfies the clauses for the method of [7] also satisfies the clauses for [9] and a corresponding PMNS can be created with the same p and n but a different  $\gamma$  and E.

Let  $\mathcal{B} = (p, n, \gamma, \rho, X^n - \lambda)$  such that  $\gamma^2 \equiv 0 \pmod{\phi}$  and  $\gamma < \rho$  which are the conditions for [7]'s method. Let us denote  $\gamma'$  the modular inverse of  $\gamma$  in  $\mathbb{Z}/p\mathbb{Z}$ . We will have  $\gamma' \times \gamma = 1 + k \times p$  with  $k \in \mathbb{Z}$ . From the parameters we get that  $\gamma^n - \lambda = k' \times p$  with  $k' \in \mathbb{Z}$ .

$$\begin{aligned} k'p &= \gamma^n - \lambda \\ \iff \gamma'^n k'p &= \gamma'^n \gamma^n - \gamma'^n \lambda \\ \iff \gamma'^n k'p &= (\gamma'\gamma)^n - \gamma'^n \lambda \\ \iff \gamma'^n k'p &= (1+k \times p)^n - \gamma'^n \lambda \end{aligned}$$
  
e will write  $(1+k \times p)^n = 1 + k''p$  with  $k'' = \sum_{i=1}^n {n \choose i} k^i p^{i-1}$ 

$$\gamma^{\prime n} k' p = 1 + k'' p - \gamma^{\prime n} \lambda$$
$$\iff \lambda \gamma^{\prime n} - 1 = (k'' - \gamma^{\prime n} k') p$$

We can therefore construct the PMNS  $\mathcal{B}' = (p, n, \gamma', \rho', E' = \lambda X^n - 1)$ . Indeed, we clearly have  $E'(\gamma') \equiv 0 \pmod{p}$ . Furthermore,  $\gamma'$  is such that the square of its modular inverse is a multiple of  $\phi$  since its modular inverse is  $\gamma$ . This means the PMNS meets the conditions for [9]'s method. (Remark 7 explains why in fact  $\rho' = \rho$  and therefore we also meet the other requirement as well).

Conversely, let  $\mathcal{B} = (p, n, \gamma, \rho, E = X^n - \lambda)$  with  $\gamma', \gamma$ 's modular inverse such that  $\gamma'^2 \equiv 0 \pmod{\phi}$ and  $\gamma' < \rho$  which are the conditions for [9]'s method.

We can show in a similar way that we can construct the PMNS  $\mathcal{B}' = (p, n, \gamma', \rho', E' = \lambda X^n - 1)$ . Indeed, we clearly have  $E'(\gamma') \equiv 0 \pmod{p}$ . Furthermore,  $\gamma'$  is such that  $\gamma'^2 \equiv 0 \pmod{\phi}$  and  $\gamma' < \rho'$  which means the PMNS meets the conditions for [7]'s method. (Remark 7 explains why in fact  $\rho' = \rho$ ).

More generally, we can show that for any PMNS  $(p, n, \gamma, \rho, E)$  we can construct another PMNS  $(p, n, \gamma', \rho', E')$  with any E by taking E' the reciprocal polynomial of E and  $\gamma'$  the modular inverse of  $\gamma$ .

We therefore introduce the following definition.

For the sake of clarity we

**Definition 2.** Let  $\mathcal{B} = (p, n, \gamma, \rho, E)$  a PMNS with  $E(X) = e_n X^n + e_{n-1} X^{n-1} + \ldots + e_1 X + e_0$ . Let  $E'(X) = e_0 X^n + e_1 X^{n-1} + \ldots + e_{n-1} X + e_n$  the reciprocal polynomial of E. Let  $\gamma'$  such that  $\gamma \gamma' \equiv 1 \pmod{p}$ . The mirror PMNS of  $\mathcal{B}$  is  $\mathcal{B}' = (p, n, \gamma', \rho, E')$ .

Remark 7. Given a short basis  $\mathcal{G}$ , each row corresponds to a polynomial that vanishes in  $\gamma$ . If we take the reciprocal of each polynomial, they will vanish in  $\gamma'$  and thus we can construct a short basis of the mirror with the same 1-norm as  $\mathcal{G}$ . Since  $\rho$  depends on the 1-norm of the short basis, we will have the same  $\rho$  for  $\mathcal{B}$  and  $\mathcal{B}'$ .

Remark 8. If  $e_0$  is negative, we can take -E' instead so that the leading coefficient is positive.

PMNS have traditionally been considered with monic external reduction polynomials only, but [27], using  $E(X) = \alpha X^n - \lambda$ , paves the way for constructing a broader class of prime.

With consideration towards binomials, we can consider the class of prime numbers of the shape  $p = \frac{\alpha\gamma^n - \lambda}{k}$  having either  $\gamma < \rho$  or  $\gamma^{-1} < \rho$  and  $k \in \mathbb{Z} \setminus \{0\}$ . Note that this could be generalized to  $p = \frac{E(\gamma)}{k}$  for any E that allows for fast external reduction with the same conditions on k and  $\gamma$  (or  $\gamma^{-1}$ ). Moreover, in the following sections, we will show that even if  $\gamma^2 \not\equiv 0 \pmod{\phi}$  or  $\gamma^{-2} \not\equiv 0 \pmod{\phi}$  (which is required in [7,9]), using sparse M gives a polynomial inverse that, while not sparse, possesses structure that still allows for a linear multiplication process.

# 5 PMNS built with $M(X) = \frac{1}{\gamma}X - 1$

This Section is about PMNS with the internal reduction polynomial M such that M(X) = tX - 1, where  $t = \frac{1}{\gamma} \pmod{p}$ . We will see how such a polynomial M results in very sparse matrices  $\mathcal{G}$  and  $\mathcal{G}'$ . Thus, making **GMont-like** (Algorithm 3) very efficient. We also explain how to generate such a PMNS.

Several kinds of primes with interesting mathematical properties have been proposed [2]. However, most of them do not offer any particular advantage for efficient modular arithmetic. In this section, we show how to easily build very efficient PMNS for some of them. Thus, allowing us to get the best of both worlds: the interesting mathematical properties of these moduli, and very efficient modular arithmetic thanks to the efficient PMNS generated for them.

From now on, we focus on the polynomials E such that  $E(X) = \alpha X^n - \lambda$ . We first consider the case where only the matrix  $\mathcal{G}$  is guaranteed to be very sparse. We will see that this is sufficient to have a reduction algorithm with linear complexity in the number of small operations. Then, we explain how to choose the polynomial M so that the matrix  $\mathcal{G}'$  is also very sparse, which allows us to minimise the cost of the internal reduction. After presenting a generation process of these PMNS, we highlight a set of well-known primes for which very efficient PMNS can be easily built.

# 5.1 Coefficient reduction

Remember that the reduction polynomial  $M \in \mathcal{L}_{\mathcal{B}}$  (see Equation 1), i.e.  $M(\gamma) \equiv 0 \pmod{p}$ . The coefficient reduction is done using the **GMont-like** algorithm. So, we need to define the matrices  $\mathcal{G}$  and  $\mathcal{G}'$ . With M(X) = tX - 1, the reduction matrix  $\mathcal{G}$  is defined as follows:

$$\mathcal{G} = \begin{pmatrix} -1 \ t \ 0 \ 0 \ \dots \ 0 \\ 0 \ -1 \ t \ 0 \ \dots \ 0 \\ 0 \ 0 \ \ddots \ \ddots \ 0 \ 0 \\ 0 \ 0 \ \dots \ 0 \ -1 \ t \ 0 \\ 0 \ 0 \ \dots \ 0 \ -1 \ t \\ t\lambda \ 0 \ 0 \ \dots \ 0 \ -\alpha \end{pmatrix} \stackrel{\leftarrow}{\leftarrow} M \\ \leftarrow X.M \ \text{mod} \ E \\ \leftarrow X^{n-3}.M \ \text{mod} \ E \\ \leftarrow X^{n-2}.M \ \text{mod} \ E \\ \leftarrow \alpha X^{n-1}.M \ \text{mod} \ E \\ \leftarrow \alpha X^{n-1}.M \ \text{mod} \ E \end{cases}$$
(15)

Notice that:

$$\det(\mathcal{G}) = (-1)^{n-1} (\lambda t^n - \alpha) \tag{16}$$

In practice,  $|\lambda|$  and  $|\alpha|$  are chosen very small, and the polynomial M is a short vector of the lattice  $\mathcal{L}_{\mathcal{B}}$ , i.e.  $|t| \approx \sqrt[n]{p}$ . As a consequence, we have  $\det(\mathcal{G}) \neq 0$  and

$$\|\mathcal{G}\|_{1} = \max(|t\lambda| + 1, |t| + |\alpha|) \tag{17}$$

Moreover, each row of  $\mathcal{G}$  is an element of  $\mathcal{L}_{\mathcal{B}}$ . So,  $\mathcal{G}$  is the basis of a sub-lattice of  $\mathcal{L}_{\mathcal{B}}$ . Thus, det $(\mathcal{G}) = kp$ , with  $k \in \mathbb{Z} \setminus \{0\}$ .

**Proposition 4.** The inverse matrix of  $\mathcal{G}$  is:

$$\mathcal{G}^{-1} = \frac{(-1)^{n-1}}{\det(\mathcal{G})} \begin{pmatrix} \alpha & \alpha t & \alpha t^2 & \alpha t^3 & \dots & \alpha t^{n-2} & t^{n-1} \\ \lambda t^{n-1} & \alpha & \alpha t & \alpha t^2 & \dots & \alpha t^{n-3} & t^{n-2} \\ \lambda t^{n-2} & \lambda t^{n-1} & \alpha & \alpha t & \dots & \alpha t^{n-4} & t^{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \lambda t^3 & \lambda t^4 & \dots & \lambda t^{n-1} & \alpha & \alpha t & t^2 \\ \lambda t^2 & \lambda t^3 & \lambda t^4 & \dots & \lambda t^{n-1} & \alpha & t \\ \lambda t & \lambda t^2 & \lambda t^3 & \lambda t^4 & \dots & \lambda t^{n-1} & 1 \end{pmatrix}$$
(18)

*Proof.* We have:

$$\mathcal{G}.\mathcal{G}^{-1} = \frac{(-1)^{n-1}}{\det(\mathcal{G})} \times \mathcal{T} = \frac{1}{\lambda t^n - \alpha} \times \mathcal{T},$$

where:

$$\mathcal{T} = \begin{pmatrix} (-\alpha + \lambda t^{n}) & (-\alpha t + \alpha t) & (-\alpha t^{2} + \alpha t^{2}) & \dots & (-\alpha t^{n-2} + \alpha t^{n-2}) & (-t^{n-1} + t^{n-1}) \\ (-\lambda t^{n-1} + \lambda t^{n-1}) & (-\alpha + \lambda t^{n}) & (-\alpha t + \alpha t) & \dots & (-\alpha t^{n-3} + \alpha t^{n-3}) & (-t^{n-2} + t^{n-2}) \\ (-\lambda t^{n-2} + \lambda t^{n-2}) & (-\lambda t^{n-1} + \lambda t^{n-1}) & (-\alpha + \lambda t^{n}) & \dots & (-\alpha t^{n-4} + \alpha t^{n-4}) & (-t^{n-3} + t^{n-2}) \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ (-\lambda t^{2} + \lambda t^{2}) & (-\lambda t^{3} + \lambda t^{3}) & (-\lambda t^{4} + \lambda t^{4}) & \dots & (-\alpha + \lambda t^{n}) & (-t + t) \\ (\alpha \lambda t - \alpha \lambda t) & (\alpha \lambda t^{2} - \alpha \lambda t^{2}) & (\alpha \lambda t^{3} - \alpha \lambda t^{3}) & \dots & (\alpha \lambda t^{n-1} - \alpha \lambda t^{n-1}) & (-\alpha + \lambda t^{n}) \end{pmatrix}$$
Thus, 
$$\mathcal{G}.\mathcal{G}^{-1} = \frac{1}{\lambda t^{n} - \alpha} \begin{pmatrix} -\alpha + \lambda t^{n} & 0 & 0 & \dots & 0 & 0 \\ 0 & -\alpha + \lambda t^{n} & 0 & \dots & 0 & 0 \\ \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 & -\alpha + \lambda t^{n} \end{pmatrix}$$

For the parameter  $\mathcal{G}' = -\mathcal{G}^{-1} \pmod{\phi}$  to exist, we need  $\gcd(\phi, \lambda t^n - \alpha) = 1$ . As said earlier, the parameter  $\phi$  must be a power of two for efficiency. Thus,  $\lambda t^n - \alpha$  must be odd for  $\mathcal{G}'$  to exist.

Remark 9 (A special case for smaller memory cost).

Remember that  $E(X) = \alpha X^n - \lambda$ . Let us assume that  $\alpha$  divides t, i.e. there exists  $s \in \mathbb{Z} \setminus \{0\}$  such that  $t = s \times \alpha$ . Then, the matrix  $\mathcal{G}$  can be defined as follows:

$$\mathcal{G} = \begin{pmatrix} -1 & t & 0 & 0 & \dots & 0 \\ 0 & -1 & t & 0 & \dots & 0 \\ 0 & 0 & \ddots & \ddots & 0 & 0 \\ 0 & 0 & \ddots & -1 & t & 0 \\ 0 & 0 & \dots & 0 & -1 & t \\ s\lambda & 0 & 0 & \dots & 0 & -1 \end{pmatrix} \xleftarrow{\leftarrow} X^{n-3}.M \mod E$$

$$\leftarrow X^{n-2}.M \mod E$$

$$\leftarrow X^{n-2}.M \mod E$$

$$\leftarrow X^{n-1}.M \mod E$$
(19)

Its inverse matrix becomes:

$$\mathcal{G}^{-1} = \frac{(-1)^{n-1}}{\det(\mathcal{G})} \begin{pmatrix} 1 & t & t^2 & t^3 & \dots & t^{n-2} & t^{n-1} \\ s\lambda t^{n-2} & 1 & t & t^2 & \dots & t^{n-3} & t^{n-2} \\ s\lambda t^{n-3} & s\lambda t^{n-2} & 1 & t & \dots & t^{n-4} & t^{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ s\lambda t^2 & s\lambda t^3 & \dots & s\lambda t^{n-2} & 1 & t & t^2 \\ s\lambda t & s\lambda t^2 & s\lambda t^3 & \dots & s\lambda t^{n-2} & 1 & t \\ s\lambda & s\lambda t & s\lambda t^2 & s\lambda t^3 & \dots & s\lambda t^{n-2} & 1 \end{pmatrix},$$
(20)

where  $\det(\mathcal{G}) = (-1)^{n-1} (\lambda s t^{n-1} - 1)$ . Thus, we have  $\det(\mathcal{G}) \neq 0$  and:

$$|\mathcal{G}||_1 = \max(|s\lambda|, |t|) + 1 \tag{21}$$

Remember that  $\rho = \|\mathcal{G}\|_1 - 1$  and  $\|\mathcal{G}\|_1 = \max(|t\lambda| + 1, |t| + |\alpha|)$  in the general case. So, depending on the values of s and  $\lambda$ , this could lead to significant improvement on  $\rho$ . This will be highlighted in some examples later. Finally, taking  $\phi$  as a power of two,  $\lambda s t^{n-1} - 1$  must be odd for  $\mathcal{G}'$  to exist.

Remark 10 (Operation complexity).

- Remember that the parameter  $\rho$  is such that  $\rho > |\lambda t|$ , with  $|t| \approx \sqrt[n]{p}$ . As it can be observed, the matrix  $\mathcal{G}$  is very sparse. Additionally, in practice,  $\rho$  is chosen small enough to fit on a single memory register. So, any vector-matrix multiplication by  $\mathcal{G}$  can be done in linear time, with at most n + 1 "small" multiplications.
- Let us now consider the vector-matrix multiplication by  $\mathcal{G}'$ . Remember that the result is reduced modulo  $\phi$ . In practice, this parameter  $\phi$  is taken small enough so that any integer fits on a single memory register after modular reduction by it. Now, let us focus on the matrix  $\mathcal{G}^{-1}$  (Equation 18). Since M(X) = tX - 1 and  $E(X) = \alpha X^n - \lambda$ , we have  $\alpha = \lambda t^n \pmod{p}$  and  $\gamma = \frac{1}{t} \pmod{p}$ . So, reading the matrix  $\mathcal{G}^{-1}$  from left to right, it can first be observed that the second-last column is equal to the last one multiplied by  $\alpha t$ . Then, from the third column to the last, each column is the preceding one multiplied by  $\frac{1}{t}$ , with  $\frac{\alpha}{t} = \lambda t^{n-1}$ . So, a vector-matrix multiplication by  $\mathcal{G}^{-1}$  can be efficiently done by first multiplying the vector by the last column of  $\mathcal{G}^{-1}$  to get the last coefficient of the result, computed modulo  $\phi$ . Then, one multiplies that coefficient by  $\alpha t$  (modulo  $\phi$ ) to get the second-last coefficient of the result. Now from the second-last to the first, each coefficient is obtained by successively multiplying the current column by  $\gamma$  (modulo  $\phi$ ) to get the next coefficient. Thus, a vector-matrix multiplication by  $\mathcal{G}^{-1}$  (modulo  $\phi$ ) can be done in linear time complexity in the number of "small" multiplications.

To sum up, with these matrices, we obtain a linear time complexity for **GMont-like** (Algorithm 3). In the remaining, PMNS possessing linear-time internal reduction will be denoted LinearRed PMNS.

# 5.2 PMNS with the matrix $\mathcal{G}'$ also sparse

As seen above, taking the polynomial M such that M(X) = tX - 1, with  $t = \frac{1}{\gamma} \pmod{p}$ , makes the matrix  $\mathcal{G}$  very sparse and the matrix  $\mathcal{G}'$  friendly, leading to a linear time complexity for the internal reduction. In this section, we study the case when  $t^2 \equiv 0 \pmod{\phi}$ . We show that this leads to a matrix  $\mathcal{G}'$  also very sparse. Thus, achieving optimal cost for the internal reduction.

Remember that the parameter  $\phi$  is taken as a power of two for **GMont-like** to be efficient. From now on, we assume that:

$$\phi = 2^h, \text{ with } h \in \mathbb{N} \setminus \{0\}$$
(22)

To have  $t^2 \equiv 0 \pmod{\phi}$ , we need  $t \equiv 0 \pmod{\sqrt{\phi}}$ . That is,  $t = q \times 2^u$ , with  $u \in \mathbb{N}$  and  $u \ge \lceil h/2 \rceil$ . We assume the parameter  $n \ge 2$ . From Equation 18, the matrix  $\mathcal{G}' = -\mathcal{G}^{-1} \pmod{\phi}$  in this case becomes:

$$\mathcal{G}' = \begin{pmatrix} 1 & t & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & t & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & t & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 1 & t & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 & \alpha^{-1}t \\ \lambda \alpha^{-1}t & 0 & 0 & \dots & 0 & \alpha^{-1} \end{pmatrix}$$
(mod  $\phi$ ) (23)

As for  $\mathcal{G}$ , it can be observed that any vector-matrix multiplication by  $\mathcal{G}'$  can be done in linear time, with at most n + 1 "small" multiplications. In the remaining, PMNS with both  $\mathcal{G}$  and  $\mathcal{G}'$  very sparse will be denoted DoubleSparse PMNS.

Remark 11. Under the condition of Remark 9, i.e.  $\alpha$  is a divisor of t (with  $t = s \times \alpha$ ), the matrix  $\mathcal{G}'$  becomes (from Equation 20):

$$\mathcal{G}' = \begin{pmatrix} 1 & t & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & t & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & t & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 1 & t & 0 \\ s\lambda t & 0 & 0 & \dots & 0 & 1 & t \\ s\lambda s\lambda t & 0 & 0 & \dots & 0 & 1 \end{pmatrix}$$
(mod  $\phi$ ) (24)

It is interesting to note that if  $st \equiv 0 \pmod{\phi}$ , which is very likely to happen in practice when  $t^2 \equiv 0 \pmod{\phi}$  since  $\alpha$  is small, then  $s\lambda t \equiv 0 \pmod{\phi}$ . This will make the matrix above even more sparse.

# 5.3 Parameters generation

In previous sections, we saw that PMNS with very sparse matrix  $\mathcal{G}$  lead to linear time complexity for internal reduction. In this section, we explain how to generate such PMNS. We highlight the requirement on the modulus p. Then, we present the generation algorithm.

Remember that we take M(X) = tX - 1 and  $E(X) = \alpha X^n - \lambda$ , with both polynomials having  $\gamma$  as a root modulo p. This leads to the following equations:

$$\begin{cases} t\gamma \equiv 1 \pmod{p}, \\ \alpha\gamma^n \equiv \lambda \pmod{p}. \end{cases}$$
(25)

As a consequence, we have:

$$\alpha \frac{1}{t^n} \equiv \lambda \pmod{p} \tag{26}$$

Thus,

$$\Delta t^n \equiv \alpha \pmod{p} \tag{27}$$

That is,

$$k \times p = \lambda t^n - \alpha \,, \tag{28}$$

where  $k \in \mathbb{Z} \setminus \{0\}$ . This is our main equation to find efficient PMNS and primes.

Now, given the modulus size we want, the goal is to find a PMNS that satisfies Equation 28. We start by first choosing n,  $\lambda$  and  $\alpha$  (so, consequently the polynomial E). Then, we search for a suitable modulus p, using an iterative process on the possible values for t. To do so, we first need to find the minimum suitable value for t, which we designate as  $t_{\min}$ . Also, for the process to end, we need a maximum value for t, which we designate as  $t_{\max}$ . This will be computed thanks to a bound on the parameter  $\phi$ .

Let  $\ell$  be the (minimum) bit size of the modulus p we want. So,  $p > 2^{\ell-1}$ . Let us consider Equation 28. To find  $t_{\min}$ , we take the minimum positive value for k (i.e. k = 1) and the maximum non-zero positive value allowed for  $\lambda$  and  $\alpha$ , we respectively note  $\lambda_{\max}$  and  $\alpha_{\max}$ . Since  $p > 2^{\ell-1}$ , this leads to the following equation:

$$2^{\ell-1} < \lambda_{\max} t_{\min}^n + \alpha_{\max}$$

Thus, we take:

$$t_{\min} = \left\lceil \sqrt[n]{\frac{2^{\ell-1} - \alpha_{\max}}{\lambda_{\max}}} \right\rceil$$
(29)

Let us now compute  $t_{\text{max}}$ . From Equation 12, the parameter  $\phi$  is taken such that:

$$\phi > 2w_{\alpha}(\|\mathcal{G}\|_{1} - 2)(\delta + 1)^{2} \tag{30}$$

To find  $t_{\max}$ , we take the minimum positive value, which is 1, for  $\lambda$  and  $\alpha$ . With Equation 17, this implies that  $\|\mathcal{G}\|_1 = |t| + 1$ . Knowing that  $w_{\alpha} \ge n$ , we obtain that:

$$\phi > 2n(t_{\max} - 1)(\delta + 1)^2$$

Thus, we take:

$$t_{\max} = \left\lfloor \frac{\phi}{2n(\delta+1)^2} \right\rfloor + 1 \tag{31}$$

As mentioned earlier, the generation process is based on Equation 28. This process requires knowing whether the integer  $y = \lambda t^n - \alpha$  has a prime factor p of bit size equal to (or greater than)  $\ell$ . To avoid the factorization, which can be quite hard, we need a bound on y and define an efficient "cleaning process" which returns such a prime factor p if it exists. Given  $t_{\max}$ , one can compute  $y_{\max}$  as follows:

$$y_{\max} = \lambda_{\max} t_{\max}^n + \alpha_{\max}$$

Let  $l_{max}$  be the bit size of  $y_{max}$ :

$$_{\max} = \lceil \log_2(y_{\max}) \rceil \tag{32}$$

Given  $\ell$  and  $l_{\text{max}}$ , Algorithm 7 describes a basic example of how this "cleaning process" can be done. Notice that very efficient implementations of this cleaning process can be found in popular libraries such as SageMath [31]. Algorithm 8 describes the process to find LinearRed and DoubleSparse PMNS,

# Algorithm 7 CleanInt, a basic example of "cleaning process" to find a desired prime factor (if any)

l

**Require:**  $y, \ell$  and  $l_{\max}$ . **Ensure:** p is a prime factor of y such that  $\lceil \log_2(p) \rceil \ge \ell$ , or returns 0 1: p = y2:  $a = 2^{l_{\max} - \ell + 1}$ 3: b = 24: while (p not prime) and  $(\lceil \log_2(p) \rceil > \ell)$  and  $(b \leq a)$  do  $p \leftarrow clean \ factor(p, b) \ \#$  remove the power of b in p factorisation 5:6:  $b \leftarrow b.next \ prime()$ # the next prime greater than b7: end while 8: if  $(\lceil \log_2(p) \rceil < \ell)$  or (p not prime) then return 0 9: 10: end if 11: return p

with the desired modulus size.

Remark 12. It is not necessary to choose  $l_{max}$  according to Equation 32 for the cleaning process. It needs to be such that  $l \leq l_{max} \leq \lceil \log_2(y_{max}) \rceil$ . Taking  $l_{max}$  smaller than  $\lceil \log_2(y_{max}) \rceil$  might only lead to missing some "LinearRed" or "DoubleSparse" PMNS in the generation process.

Remember that we take  $E(X) = \alpha X^n - \lambda$ . So, from Equations 11, 17 and 30, we want the parameter  $\phi$  to be such that:

$$\phi > 2(\delta+1)^2 \times \max(\alpha n, \alpha + (n-1)|\lambda|) \times (\max(|t\lambda|+1, |t|+|\alpha|) - 2)$$
(33)

Remark 13. As mentioned earlier, the parameter  $\phi$  is taken as a power of two for efficiency. Let us assume that the target hardware is a *h*-bit architecture. As suggested in [11], a good choice for efficient modular arithmetic is to take  $\phi = 2^h$ . Thus, implying the parameter *n* to be such that:

$$n \geqslant n_{min} = \lfloor \frac{l}{h} \rfloor + 1$$

So, Algorithm 8 must be executed with  $n \ge n_{min}$ . Also, depending on the value of  $\delta$ , the value of n must be increased as much as necessary in order to find a suitable PMNS.

*Remark 14.* Algorithm 8 stops when it finds a PMNS (see line 20). However, this is not necessary. One can continue the search if more than one PMNS is desired.

Algorithm 8 LinearRed and DoubleSparse PMNS generation

**Require:** prime (minimum) bit size  $\ell$ , the parameters  $n, \phi = 2^h$  and  $\delta$ , and  $\lambda_{\max}$  and  $\alpha_{\max}$ . **Ensure:**  $\mathcal{B}$  is a LinearRed or DoubleSparse PMNS, or returns 0 1: Compute  $t_{\min}$ , see Equation 29 2: Compute  $t_{\text{max}}$ , see Equation 31 3: Compute  $l_{\text{max}}$ , see Equation 32 and Remark 12 4: for  $t \in \{-t_{max}, ..., -t_{min}\} \cup \{t_{min}, ..., t_{max}\}$  do #IMPORTANT: take  $t \equiv 0 \pmod{2^{\lceil h/2 \rceil}}$  if a DoubleSparse PMNS is desired (see Section 5.2) 5: Randomly (or iteratively) choose a non-zero  $\alpha \in \{-\alpha_{\max}, \ldots, \alpha_{\max}\}$ 6: Randomly (or iteratively) choose a non-zero  $\lambda \in \{-\lambda_{\max}, \ldots, \lambda_{\max}\}$ 7:8: if Equation 33 is not satisfied then 9: Jump to the next iteration for t with |t| smaller, or change  $(\alpha, \lambda)$ 10:end if  $y \leftarrow \lambda t^n - \alpha$ 11: 12: $p \leftarrow \mathbf{CleanInt}(y, \ell, l_{\max}) \quad \# \text{ see Algorithm 7 for example}$ 13:if p = 0 then Make another choice for  $(\alpha, \lambda)$ , or jump to the next iteration for t 14:end if 15: $\gamma \leftarrow \frac{1}{t} \pmod{p}$ 16: $\rho \leftarrow \max(|t\lambda| + 1, |t| + |\alpha|) - 1$ 17: $E \leftarrow \alpha X^n - \lambda \quad \# \text{ take } -E \text{ if } \alpha < 0$ 18:19: $M \leftarrow tX - 1$  $\mathcal{B} \leftarrow (p, n, \gamma, \rho, E, M, \phi, \delta)$ 20: return  $\mathcal{B}$ 21:22: end for 23: return 0

Here are some examples of PMNS found with Algorithm 8.

Example 1. For a 256-bit modulus.

- p = 60440003927590133985782451365630693872755589432225658125679387443792520937473
- n = 5
- $\gamma = 35113295015827659083538442227693603887830925641299948652724224$
- $\rho = 6885141813133313.$
- $E(X) = X^5 4$
- M(X) = -1721285453283328.X 1
- $\phi = 2^{64}$
- $\delta = 3$

Example 2. For a 384-bit modulus.

- p = 20563376626493461018222137199352894753424815984240003749142821491197470524712223452918959096229392120215140627906559
- *n* = 7
- $\gamma = 870258118347971125321513664674627813201777625547094021760712296297683007321$  307612563373840421355520
- $\rho = 118145273183600641.$
- $E(X) = X^7 5$
- M(X) = 23629054636720128.X 1
- $\phi = 2^{64}$
- $\delta = 0$

#### $\mathbf{5.4}$ Efficient PMNS from special primes

The method presented above does not allow to choose the modulus but only its bit size. Additionally, the generated modulus p does not have any special shape, except that it is a divisor of  $\lambda t^n - \alpha$  (see Equation 28). Thus, one may wonder if it is possible to generate LinearRed or DoubleSparse PMNS for any given prime. It seems impossible to do this for all primes. However, some particular shapes of primes allow to generate such PMNS very easily. In this section, we highlight some examples of such primes and explain how to easily build efficient PMNS for them.

On the Wikipedia prime number page [2], a list of primes with special shapes is given. Although these primes have interesting mathematical properties, they do not offer any particular advantage for efficient modular arithmetic, except for a few, such as Fermat, Mersenne, and Pseudo-Mersenne primes, which serve as bases for many cryptographic protocol parameters, because of the performance they offer for modular arithmetic in classical representation. We will see how to generate LinearRed or DoubleSparse PMNS for primes of some special shapes, including many of those given in the wiki link above. Thus, allowing us to get the best of both worlds: the interesting mathematical properties of these moduli, and very efficient modular arithmetic thanks to the LinearRed or DoubleSparse PMNS generated for them. Let us start with primes p such that:

$$p = \frac{u2^l - c}{r},\tag{34}$$

where u > 0 and  $c \neq 0$  are small integers, and  $r \in \mathbb{N} \setminus \{0\}$ .

This includes several types of primes. Some examples are given in Table 1, with the values of u, c and r that lead to the corresponding type of prime.

Prime type	u	c	r	Prime shape
Cullen prime	l	-1	1	$p = l \times 2^l + 1$
Woodall prime	l	1	1	$p = l \times 2^l - 1$
Thabit prime	3	1	1	$p = 3 \times 2^l - 1$
Thabit prime of the second kind	3	-1	1	$p = 3 \times 2^l + 1$
Fermat prime	1	-1	1	$p = 2^l + 1$
Mersenne prime	1	1	1	$p = 2^l - 1$
Pseudo-Mersenne prime	1	c > 0	1	$p = 2^l - c$
	1	c > 0	1	$p = 2^l + c$
Proth prime		-1	1	$p=u\times 2^l+1$
		1		$p = u \times 2^l - 1$
Wagstaff prime	1	-1		$p = \frac{2^l + 1}{3}$
	1	1	3	$p = \frac{2^l - 1}{3}$

Table 1: Example of friendly prime shapes for optimal PMNS

Equation 34 implies that  $r \times p = u2^{l} - c$ . Let l = wn + s be the Euclidean division of l by n. Then  $r \times p = 2^{s} u (2^{w})^{n} - c$ . So, a DoubleSparse PMNS (see Section 5.2) can be built with:

• 
$$E(X) = cX^n - 2^s u$$

• 
$$M(X) = 2^w X = 1$$

• 
$$\gamma = \frac{1}{2^w} \pmod{p}$$

Alternatively, if s and w are taken such that l = wn - s. Then,  $r \times p = 2^{-s} u(2^w)^n - c$ . So, a DoubleSparse PMNS can be built with:

•  $E(X) = 2^s c X^n - u$ 

•  $M(X) = 2^w X - 1$ •  $\gamma = \frac{1}{2^w} \pmod{p}$ 

In both cases, note that if c < 0, you can multiply E by -1 to make its leading coefficient positive.

### Remark 15 (The lowest/best cost).

In both cases, the generated DoubleSparse PMNS are such that M(X) = tX - 1, where t is a power of two. Thus, depending on the values of  $\alpha$  and  $\lambda$ , a vector-matrix multiplication by  $\mathcal{G}$  (see Equation 15) costs at most 2 "small" multiplications, while a multiplication by  $\mathcal{G}'$  (see Equation 24) costs at most 3 "small" multiplications. Indeed, in this case, a multiplication by t is a simple left shift, which is very cheap. As a consequence, an internal reduction with **GMont-like** costs at most 5 "small" multiplications, regardless of the value of n. This is the best/lowest possible cost that can be expected for this algorithm. This should make these PMNS more efficient than Pseudo-Mersenne primes (in classical representation) and nearly as efficient as Mersenne primes (also in classical representation). Moreover, based on Remarks 9 and 11, if  $\alpha$  (which is either c or  $2^{s}c$ ) divides  $t = 2^{w}$ , then we obtain PMNS with smaller value of  $\rho$  and **GMont-like** costs at most 4 "small" multiplications.

Example 3. For this example and for the next ones in this section, we assume that  $\phi = 2^{64}$ . Let us consider the NIST prime  $p = 2^{521} - 1$ . With n = 9, knowing that  $521 = 58 \times 9 - 1$ , we have a DoubleSparse PMNS with the following parameters:

• 
$$E(X) - 2X^9 - 1$$

- $E(X) = 2X^9 1$   $M(X) = 2^{58}X 1$   $\gamma = \frac{1}{2^{58}} \pmod{p}$   $\rho = 2^{58} + 1$  (see Remarks 9 and 11)

Example 4. Let us now consider the Proth prime  $p = 7 \times 2^{320} + 1$ . With n = 6, knowing that 320 = $54 \times 6 - 4$ , we have a DoubleSparse PMNS with the following parameters:

• 
$$E(X) = 2^4 X^6 + 7$$

- $M(X) = 2^{54}X 1$   $\gamma = \frac{1}{2^{54}} \pmod{p}$   $\rho = 2^{54} + 1$  (see Remarks 9 and 11)

*Example 5.* As a last example, let us consider the Wagstaff prime  $p = \frac{2^{347}+1}{3}$ . With n = 6, knowing that  $347 = 58 \times 6 - 1$ , we have a DoubleSparse PMNS with the following parameters:

- $E(X) = 2X^6 + 1$

- $M(X) = 2^{58} X 1$   $M(X) = 2^{58} X 1$   $\gamma = \frac{1}{2^{58}} \pmod{p}$   $\rho = 2^{58} + 1$  (see Remarks 9 and 11)

# Remark 16 (Choice of n).

To build these PMNS, the choice of n must be done according to the actual bit size of p (i.e.  $\lceil \log_2(p) \rceil$ ), instead of l in Equation 34. That is,  $n \ge \lfloor \frac{\lceil \log_2(p) \rceil}{h} \rfloor + 1$ . Also, as mentioned in Remark 13, depending on the value chosen for  $\delta$ , the parameter n may need to be increased.

Equation 34 can be generalized to encompass even more types of primes. It can be extended to the following:

$$p = \frac{u a^l - c}{r} \,, \tag{35}$$

where u > 0,  $a \ge 2$  and  $c \ne 0$  are small integers, and  $r \in \mathbb{N} \setminus \{0\}$ . For small bases a, this includes the generalized repunit primes [14]:

$$p = \frac{a^l - 1}{a - 1} \,,$$

obtained for u = c = 1 and r = a - 1.

The same goes for the primes of the form [15]:

$$p = \frac{a^l + 1}{a+1} \,,$$

obtained for u = 1, c = -1 and r = a + 1. Note that this generalizes the Wagstaff primes above mentioned.

Similar to Equation 34, let us first assume that l = wn + s. Then, Equation 35 implies that  $r \times p =$  $a^{s}u(a^{w})^{n} - c$ . In this case, a LinearRed PMNS (see Section 5.1) can be built with:

- $E(X) = cX^n a^s u$   $M(X) = a^w X 1$

•  $\gamma = \frac{1}{a^w} \pmod{p}$ Likewise, if l = wn - s, then  $r \times p = a^{-s}u(a^w)^n - c$ . So, a LinearRed PMNS can be built with: •  $E(X) = a^s c X^n - u$ 

- $M(X) = a^w X 1$
- $\gamma = \frac{1}{a^w} \pmod{p}$

Again, note that if c < 0, you can multiply E by -1 to make its leading coefficient positive. Also, depending on the value of a, the generated PMNS may be DoubleSparse.

Example 6. For  $\phi = 2^{64}$ , let us consider the generalized repunit prime  $p = \frac{3^{103}-1}{2}$ . With n = 3, knowing that  $103 = 34 \times 3 + 1$ , we have a LinearRed PMNS with the following parameters: •  $E(X) = X^3 - 3$ 

- $M(X) = 3^{34}X 1$
- $\gamma = \frac{1}{3^{34}} \pmod{p}$   $\rho = 3^{35} + 1$

*Example 7.* Finally, let us consider the prime  $p = \frac{3^{281}+1}{4}$ . With n = 8, knowing that  $281 = 35 \times 8 + 1$ , we have a LinearRed PMNS with the following parameters:

- $E(X) = X^8 + 3$
- $M(X) = 3^{35}X 1$
- $\gamma = \frac{1}{3^{35}} \pmod{p}$
- $\rho = 3^{36} + 1$

#### PMNS built with $M(X) = \frac{\gamma}{\lambda} X^{n-1} - 1$ 6

In this section, we generalize the PMNS shape used in [7] as introduced in Section 4 that uses M(X) = $\frac{\gamma}{2}X^{n-1}-1$ . In the process, we also show that this leads to linear time internal reduction even for  $\gamma^2 \neq 0$  $(\mod \phi)$ . In the sequel of this section, we assume E to be of the shape  $\alpha X^n - \lambda$ .

The authors from [7] choose their parameters to get  $p = \gamma^n / \lambda - 1$  (with  $\gamma$  a multiple of  $\lambda$ ) mainly for efficiency reasons and also require  $\gamma < \rho$  [7, Theorem 13]. In this section, we generalize this without sacrificing too much in terms of speed. Namely we consider primes of the shape  $p = \frac{\alpha \gamma^n - \lambda}{k}, k \in \mathbb{Z}^*$ . This is a generalization because if  $\alpha = 1$  and  $k = \lambda$  we return to the shape from [7].

We will have  $\gamma \approx \sqrt[n]{\frac{k}{\alpha}}p$  which means that for small enough k we get the required property  $\gamma < \rho$  (since as noted earlier  $\rho \approx \sqrt[n]{p}$  which in practice means it can fit in a word-size register (since we normally choose  $\rho$  so that our coefficients fit on a single register). This Section will show that this property of having a small enough  $\gamma$  to fit on a memory register leads to linear time coefficient reduction which helps our modular multiplication become competitive with optimized Pseudo-Mersenne modular multiplication algorithms with a broad class of primes.

#### Existence 6.1

Let  $\ell$  be the size of primes considered. We have  $2^{\ell-1} . This leads to <math>k \times 2^{\ell-1} < k \times p < k \times 2^{\ell}$ . We therefore choose  $\gamma$  as follows:

$$k \times 2^{\ell - 1} \leqslant \alpha \gamma^n \leqslant k \times 2^{\ell}$$

Which gives us:

$$\left\lceil \left(\frac{k}{\alpha} \times 2^{\ell-1}\right)^{\frac{1}{n}} \right\rceil \leqslant \gamma \leqslant \left\lfloor \left(\frac{k}{\alpha} \times 2^{\ell}\right)^{\frac{1}{n}} \right\rfloor$$
(36)

For a given  $\gamma$ , we can bound  $\lambda$  by considering two consecutive primes  $p_1$  and  $p_2$  such that  $p_1 < p_2$  and also such that  $p_1 \leq \left\lfloor \frac{\alpha \gamma^n}{k} \right\rceil \leq p_2$ . We can always bound an integer between consecutive prime numbers. This leads to  $kp_1 < \alpha \gamma^n < kp_2$  which gives us

$$|\lambda| \leqslant \frac{kp_2 - kp_1}{2}.\tag{37}$$

The problem of upper bounds on the gap between consecutive primes is closely related to the Riemann Hypothesis so we won't give an exact upper bound for all sizes of primes. Shanks's conjecture from 1964 which still holds to this day gives the maximum gap between two consecutive primes  $p_1$  and  $p_2$  to be  $\sim ln(p_1)^2$ . The largest known ratio  $\frac{p_2-p_1}{ln(p_1)^2}$  to date is approximately 0.9206 [29].

However, a classical result on the average gap comes from the Prime Number Theorem [18,32] that states that the gap between two consecutive primes is asymptotically  $\sim ln(p_1)$ . The following table shows our empirical result for various integer sizes of the ratio  $\frac{p_2-p_1}{ln(p_1)}$  and shows for our purpose we can indeed consider the gap to be approximately  $ln(p_1)$  on average for the sizes we are considering.

prime size in bits	256	512	1024	2048
ratio	1.0058	1.0031	1.0005	1.0007

Average ratio between the prime gap at given integer sizes and their natural logarithm for a random sample of  $2^{21}$  primes of given size

We have  $|\lambda| \leq \frac{k(p_2-p_1)}{2}$  (equation 37) which we can now take as  $|\lambda| \leq \frac{k(ln(p_1))}{2}$ . We want primes between  $2^{\ell-1}$  and  $2^{\ell}$  so on average  $ln(p_1) = ln(1.5 \times 2^{\ell-1})$ . If our bounds allow taking  $\lambda$  such that  $\lambda \geq \lfloor \frac{k}{2}ln(1.5 \times 2^{\ell-1}) \rfloor$  we can therefore expect to find at least one prime p on average within the bounds.

The number of possible  $\gamma$  for a given parameter set of  $(\ell, n, k, \alpha)$  is simply  $\left\lfloor \left(\frac{k}{\alpha} \times 2^{\ell}\right)^{\frac{1}{n}} \right\rfloor - \left\lceil \left(\frac{k}{\alpha} \times 2^{\ell-1}\right)^{\frac{1}{n}} \right\rceil$ 

(from equation 36) which is  $\sim \sqrt[n]{\frac{k}{\alpha}} ((\sqrt[n]{2}-1)\sqrt[n]{2^{\ell-1}})$ . If we note  $\lambda_{\max}$  the maximum allowed value for  $\lambda$  by our bounds, each  $\gamma$  will find on average  $\frac{\lambda_{\max}}{\left\lceil \frac{k}{2} ln(1.5 \times 2^{\ell-1}) \right\rceil}$  valid primes so the total number of primes for a given range can be estimated to be  $(\sqrt[n]{\frac{k}{\alpha}}((\sqrt[n]{2}-1)\sqrt[n]{2^{\ell-1}})) \times \frac{\lambda_{\max}}{\frac{k}{2}ln(1.5 \times 2^{\ell-1})}$ .

Remark 17. Traditionally in AMNS, authors choose small values for  $\lambda$  to allow for use of faster instructions such as *LEA* or binary shifts, however, integer multiplication is 3-4 cycles on most modern CPUs [17] so the value of our  $\lambda$  needs not to be small while preserving optimal computation speeds. Furthermore, even on architectures where integer multiplication is much more costly, the tradeoff of getting a linear-time reduction in exchange is virtually always worth the cost. With that said the density of this class of primes is big enough that in practice we can still expect small values for  $\lambda$  to exist depending on our choice of parameters.

# 6.2 Coefficient Reduction

The external reduction process with non-monic E(X) involves an additional multiplication by  $\alpha$ . In [27], the internal reduction is performed using polynomial multiplication which leads to applying further reductions by E and multiplying by  $\alpha$  for each step which means additional adaptation to get a correct result once evaluated in  $\gamma$ . This requirement comes from needing to use polynomial multiplication in the internal reduction process. However, in this paper, we use **Algorithm 3** for our internal reduction instead, which avoids any additional external reduction steps to reduce our coefficients due to not relying on polynomial operations. This means we need to construct a short basis for a sub-lattice of  $\mathfrak{L} = \left\{ (x_0, \ldots, x_{n-1}) \in \mathbb{Z}^n : \sum_{i=0}^{n-1} x_i \gamma^i \equiv 0 \mod p \right\}$ . Normally, we would do this by constructing the companion matrix of  $M(X) = \frac{\gamma}{\lambda} X^{n-1} - 1$  but for our purposes we can simply use the following proposition instead.

**Proposition 5.** Let  $p = \frac{\alpha \gamma^n - \lambda}{k}$ . The matrix

$$\mathcal{G} = \begin{pmatrix} -\gamma & 1 & 0 & \dots & \dots & 0 \\ 0 & -\gamma & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & \dots & 0 & -\gamma & 1 \\ \lambda & 0 & \dots & \dots & 0 & -\alpha\gamma \end{pmatrix}$$

is a basis of a sub-lattice of  $\mathfrak{L} = \left\{ (x_0, \dots, x_{n-1}) \in \mathbb{Z}^n : \sum_{i=0}^{n-1} x_i \gamma^i \equiv 0 \mod p \right\}.$ 

*Proof.* The first n-1 rows correspond to the polynomials  $X^i - \gamma X^{i-1}$ . When evaluated in  $\gamma$  they become  $\gamma^i - \gamma \times \gamma^{i-1}$  which becomes 0. The last row corresponds to  $-\alpha\gamma X^{n-1} + \lambda$  which becomes  $-\alpha\gamma^n + \lambda$  when evaluated in  $\gamma$ . We know

from construction that this is equal to  $-k \times p$  and is thus congruent to 0 modulo p.

We get that each row of the matrix vanishes in  $\gamma \mod p$ . They are therefore all elements of  $\mathfrak{L}$  = ſ n-1

$$\left\{ (x_0, \dots, x_{n-1}) \in \mathbb{Z}^n : \sum_{i=0}^n x_i \gamma^i \equiv 0 \mod p \right\}.$$

The value of the determinant is equal to  $-\gamma \times (-\alpha \gamma)^{n-1}$  plus or minus  $\lambda$  depending on parity of n. Therefore  $|\det(\mathcal{G})| = k \times p$ .

Since  $det(\mathcal{G}) \neq 0$  each row is linearly independent and we have a basis.

Remark 18. For  $k = \pm 1$  we will have a basis of the full lattice.

Remark 19. Note that except for the last row, this is equivalent to building the companion matrix of  $X - \gamma$ .

Because of the shape of the matrix, any vector-matrix multiplication can be done in linear time. Indeed we only have two non-0 coefficients per row and per column and  $\gamma < \rho$  which makes the coefficients fit on a single memory register. In the general case for random primes,  $\gamma$  can be of the same size as p and therefore would not fit on a single memory register. This would cause a vector-matrix multiplication to be quadratic instead. In our specific case, having a small enough  $\gamma$  will mean  $\mathcal{G}$  as defined in proposition 5 will be a short basis.

Indeed, notice that  $\|\mathcal{G}\|_1 = \max(\gamma + |\lambda|, |\alpha\gamma| + 1)$ . Since  $\alpha$  and  $\lambda$  are chosen to be small and  $\gamma \approx \sqrt[n]{\frac{k}{\alpha}p}$ , we get  $\|\mathcal{G}\|_1 \approx \sqrt[n]{\alpha^{n-1}kp}$ . Minkowski's theorem from [23] gives us that for any lattice  $\mathcal{L}$  of dimension n, its short basis will have an upper bound on its 1-norm of  $\sqrt[n]{n! \det(\mathcal{L})}$ . In our case, this becomes  $\sqrt[n]{n!p}$ . In other words,  $\mathcal{G}$  is a short basis as long as  $\alpha^{n-1}k \leq n!$  which in practice will always be the case.

This gives us sufficient conditions to construct a basis that is both short and sparse which gives us linear time complexity vector-matrix multiplication. Note that in practice even if  $\mathcal{G}$  is not the shortest possible basis, as long as all its coefficients fit on a memory register the time complexity will still be linear. However, we still want  $\rho < \frac{\phi}{2w}$  and  $\rho$  is set depending on the 1-norm of  $\mathcal{G}$  so we want it as small as possible.

Remark 20. Since  $|\det(\mathcal{G})| = k \times p$  if we want it to be invertible modulo  $\phi = 2^h$ , k can be chosen odd to guarantee it. Otherwise, for any  $k = z2^{\psi}$ ,  $(z, \psi) \in \mathbb{Z}^* \times \mathbb{Z}$  with z odd, if  $\alpha \gamma$  and  $\lambda$  are divisible by  $2^{\psi}$ , one can use the following basis instead:

$$\tilde{\mathcal{G}} = \begin{pmatrix} -\gamma \ 1 \ 0 \ \dots \ 0 \\ 0 \ -\gamma \ 1 \ 0 \ \dots \ 0 \\ \vdots \ \vdots \ \vdots \ \vdots \ \vdots \ \vdots \\ 0 \ \dots \ 0 \ -\gamma \ 1 \\ \frac{\lambda}{2^{\psi}} \ 0 \ \dots \ 0 \ \frac{-\alpha\gamma}{2^{\psi}} \end{pmatrix}$$

We get  $|\det(\tilde{\mathcal{G}})| = z \times p$  instead. Since z is odd, this is invertible mod  $\phi$ . Requiring  $\alpha \gamma$  and  $\lambda$  divisible by  $2^{\psi}$  implies fewer potential candidates but it is still reasonable for small  $\psi$ .

**Proposition 6.** The inverse matrix of G is :

$$\mathcal{G}^{-1} = \begin{pmatrix} \frac{-\alpha\gamma^{n-1}}{kp} & \frac{-\alpha\gamma^{n-2}}{kp} & \cdots & \frac{-\alpha\gamma^2}{kp} & \frac{-\alpha\gamma}{kp} & \frac{-1}{kp} \\ \frac{-\lambda}{kp} & \frac{-\alpha\gamma^{n-1}}{kp} & \cdots & \frac{-\alpha\gamma^3}{kp} & \frac{-\alpha\gamma^2}{kp} & \frac{-\gamma}{kp} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{-\lambda\gamma^{n-3}}{kp} & \frac{-\lambda\gamma^{n-4}}{kp} & \cdots & \frac{-\lambda}{kp} & \frac{-\alpha\gamma^{n-1}}{kp} & \frac{-\gamma^{n-2}}{kp} \\ \frac{-\lambda\gamma^{n-2}}{kp} & \frac{-\lambda\gamma^{n-3}}{kp} & \cdots & \frac{-\lambda\gamma}{kp} & \frac{-\lambda}{kp} & \frac{-\gamma^{n-1}}{kp} \end{pmatrix}$$

Proof.

$$\mathcal{G}.\mathcal{G}^{-1} = \begin{pmatrix} \frac{\alpha\gamma^{n-\lambda}}{kp} & \frac{-\alpha\gamma^{n-1} + \alpha\gamma^{n-1}}{kp} & \dots & \frac{-\alpha\gamma^{3} + -\alpha\gamma^{3}}{kp} & \frac{-\alpha\gamma^{2} + \alpha\gamma^{2}}{kp} & \frac{-\gamma^{+\gamma}}{kp} \\ \frac{\lambda\gamma - \lambda\gamma}{kp} & \frac{\alpha\gamma^{n-\lambda}}{kp} & \dots & \frac{-\alpha\gamma^{4} + -\alpha\gamma^{4}}{kp} & \frac{-\alpha\gamma^{3} + -\alpha\gamma^{3}}{kp} & \frac{-\gamma^{2} + \gamma^{2}}{kp} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\lambda\gamma^{n-2} - \lambda\gamma^{n-2}}{kp} & \frac{\lambda\gamma^{n-3} - \lambda\gamma^{n-3}}{kp} & \dots & \frac{\lambda\gamma - \lambda\gamma}{kp} & \frac{\alpha\gamma^{n-\lambda}}{kp} & \frac{-\gamma^{n-1} + \gamma^{n-1}}{kp} \\ \frac{\alpha\lambda\gamma^{n-1} - \alpha\lambda\gamma^{n-1}}{kp} & \frac{\alpha\lambda\gamma^{n-2} - \alpha\lambda\gamma^{n-2}}{kp} & \dots & \frac{\alpha\lambda\gamma^{2} - \alpha\lambda\gamma^{2}}{kp} & \frac{\alpha\lambda\gamma - \alpha\lambda\gamma}{kp} & \frac{\alpha\gamma^{n} - \lambda}{kp} \end{pmatrix}$$

The second to last column of  $\mathcal{G}^{-1}$  is equal to the last column times  $\alpha\gamma$  (for the last coefficient, note that  $-\alpha\gamma^n \equiv \lambda \pmod{p}$ ). Similarly, each column before that is just the next column times  $\gamma$ . As such, when computing a vector-matrix multiplication with  $\mathcal{G}^{-1}$ , it can be done by first computing the vector times the last column to get the last coefficient of the result and then multiplying that product successively for each previous column to get each other coefficient. In other words, a vector-matrix multiplication with  $\mathcal{G}^{-1}$  can be done in linear time complexity.

Remark 21. Since we reduce the result of the vector-matrix product by  $\mathcal{G}^{-1}$  modulo  $\phi$  in practice we use  $\mathcal{G}^{-1}$  (mod  $\phi$ ) which guarantees the coefficients will all fit in a memory register each.

As noted earlier, since  $\mathcal{G}$  is sparse, the product can similarly be done in linear time which gives us an overall linear time complexity for Algorithm 3.

*Remark 22.* Contrary to [7] and [9], this linear-time internal reduction process doesn't require  $\gamma^2 \equiv 0 \pmod{\phi}$ . In fact,  $\gamma$  could be coprime with  $\phi$  without affecting the time complexity.

As noted in the previous section (section 5.1, on page 14), for the sake of expediency, PMNS possessing linear-time internal reduction shall be denoted LinearRed PMNS.

# 6.3 Generation

This Section details the generation algorithm. We are searching for primes of the shape  $p = \frac{\alpha \gamma^n - \lambda}{k}$ . For the generation process, we will try to find primes near  $\frac{\alpha \gamma^n}{k}$  which means we first must bound the maximum value for  $\lambda$ .

As noted in Section 3.2, the optimal value for  $\rho$  is  $\rho = \|\mathcal{G}\|_1 - 1$  and our bound on  $\phi$  is:

$$2w(\|\mathcal{G}\|_1 - 2) < \phi$$

with  $w = \max(\alpha n, \alpha + (n-1)|\lambda|)$  (from equation 11) which means

$$w = \begin{cases} \alpha + |\lambda|(n-1) & \text{if } \alpha \leq |\lambda|, \\ \alpha n & \text{otherwise} \end{cases}$$

and furthermore we have

$$\|\mathcal{G}\|_1 = \begin{cases} \gamma + |\lambda| & \text{if } \alpha = 1, \\ \alpha \gamma + 1 & \text{otherwise.} \end{cases}$$

For  $\alpha = 1$  we have:

$$2((n-1)|\lambda| + 1)(\gamma + |\lambda| - 2) < \phi$$

If we note  $\lambda_{\max}$  a bound on the absolute value of  $\lambda$ , we get that  $\lambda_{\max}$  will verify:

$$2((n-1)\lambda_{\max}+1)(\gamma+\lambda_{\max}-2) = \phi - 1$$
  

$$\iff 2(n-1)\lambda_{\max}^{2} + 2((n-1)(\gamma-2)+1)\lambda_{\max} + 2\gamma - \phi - 3 = 0$$
  

$$\iff \lambda_{\max} = \frac{\sqrt{(2((n-1)(\gamma-2)+1)^{2} - 4(2(n-1))(2\gamma-\phi-3)} - 2((n-1)(\gamma-2)+1)}{2(2(n-1))}$$
  

$$\iff \lambda_{\max} = \frac{(\gamma-2)\left(\sqrt{1 + \frac{2\phi-2\gamma+2}{(n-1)(\gamma-2)^{2}} + \frac{1}{((n-1)(\gamma-2))^{2}}} - 1\right) - \frac{1}{n-1}}{2}$$

Since this is not very human-readable, we give this simplified form through series expansion to give a clearer intuition of the value:

$$\lambda_{\max} \sim \frac{\phi - \gamma + 1}{2(n-1)(\gamma - 2)}.$$

Meanwhile for all other  $\alpha$  with  $|\lambda| \ge \alpha$  we have:

$$2(\alpha + |\lambda|(n-1))(\alpha\gamma - 1) < \phi.$$

Which means:

$$\lambda_{\max} = \frac{\phi - 2\alpha(\alpha\gamma - 1) - 1}{2(n-1)(\alpha\gamma - 1)}.$$

 $\begin{array}{l} Remark \ 23. \ \text{As we can see, } \lambda_{\max} \ \text{is inversely proportional to } \gamma \ \text{in all cases. For } \alpha = 1 \ \text{we get } \lambda_{\max} \sim \frac{\phi - \gamma + 1}{2(n-1)(\gamma-2)} = \frac{\phi - 1}{2(n-1)(\gamma-2)} - \frac{1}{2(n-1)} \ \text{while otherwise we get } \frac{\phi - 2\alpha(\alpha\gamma - 1) - 1}{2(n-1)(\alpha\gamma - 1)} = \frac{\phi - 1}{2(n-1)(\alpha\gamma - 1)} - \frac{2\alpha}{2(n-1)}. \end{array}$ 

Our bound on  $\lambda$  can therefore be expressed as:

$$\lambda_{\max} = \begin{cases} \frac{(\gamma - 2)\left(\sqrt{1 + \frac{2\phi - 2\gamma + 2}{(n-1)(\gamma - 2)^2} + \frac{1}{((n-1)(\gamma - 2))^2}} - 1\right) - \frac{1}{n-1}}{2} & \text{if } \alpha = 1, \\ \min\left(\alpha, \frac{\phi - 2\alpha(\alpha\gamma - 1) - 1}{2(n-1)(\alpha\gamma - 1)}\right) & \text{otherwise.} \end{cases}$$
(38)

Remark 24. For  $\delta > 0$  our bound is  $2w(\|\mathcal{G}\|_1 - 2)(\delta + 1)^2 < \phi$  which gives us:

$$\delta_{\max} = \left\lfloor \sqrt{\frac{\phi}{2w(\|\mathcal{G}\|_1 - 2)}} \right\rfloor - 1 \tag{39}$$

# Algorithm 9 LinearRed PMNS generation

**Require:** prime size  $\ell$ , number of coefficients n, word size  $\phi$ , parameters k,  $\alpha$  and  $\delta_{min}$ **Ensure:** A PMNS  $(p, n, \gamma, \rho, E := \alpha X^n - \lambda, \delta)$  with linear time coefficient reduction such that  $k \times p = \alpha \gamma^n - \lambda$ if within bounds else 0 1: Compute  $\lambda_{\max}$  using  $\gamma = \left[ \left( \frac{k}{\alpha} \times 2^{\ell-1} \right)^{\frac{1}{n}} \right]$ , see Equation 38 2: if  $\lambda_{\max} < 1$  then 3: return 0 4: **end if** 5: while True do Choose  $\gamma$  randomly such that  $\left[\left(\frac{k}{\alpha} \times 2^{\ell-1}\right)^{\frac{1}{n}}\right] \leqslant \gamma \leqslant \left\lfloor \left(\frac{k}{\alpha} \times 2^{\ell}\right)^{\frac{1}{n}} \right\rfloor$ 6: Compute  $\lambda_{\max}$ , see Equation 38 7: 8: if  $\lambda_{\max} \ge 1$  then  $p \leftarrow \text{next\_prime}(\lfloor \frac{\alpha \gamma^n - \lambda_{\max}}{k} \rfloor)$ 9:  $\lambda \leftarrow \alpha \gamma^n - kp$ 10: Compute  $\delta$ , see Equation 39 11: if  $|\lambda| \leq \lambda_{\max}$  and  $GCD(\frac{k}{GCD(k,\lambda,\alpha\gamma)},\phi) = 1$  and  $\delta \geq \delta_{min}$  then 12: $\rho \leftarrow \max(\gamma + |\lambda|, \alpha\gamma + 1) - 1$ 13: $E \leftarrow \alpha X^n - \lambda$ 14:return  $(p, n, \gamma, \rho, E, \delta)$ 15:16:end if 17:end if 18: end while

On the first line of **Algorithm 9** we compute  $\lambda_{\max}$  with the smallest value of  $\gamma$  for the input parameters because, as seen in remark 23,  $\lambda_{\max}$  is inversely proportional to  $\gamma$  and as such if  $\lambda_{\max}$  is lesser than 1 for such  $\gamma$ , it will also be the case for all other  $\gamma$  and we won't be able to construct a valid PMNS. Note that for lower values of  $\lambda_{\max}$ , **Algorithm 9** may not find a valid PMNS and run forever. To guarantee finite time execution, one may choose to iterate over all possible values of  $\gamma$  instead, although the result would not be random like it is in the current algorithm (or alternatively find all PMNS for that range first and then randomly choose one among them although doing so may take an inconveniently long time). For additional steps for  $GCD(k, \phi) \neq 1$ , see *Remark 20*.

Next, as to the potential outputs, as an example for  $(\ell, n, k, \alpha) = (256, 5, 1, 1)$  there are 334838928077626 possible  $\gamma$  which is approximately  $2^{48}$ . Here,  $\lambda_{\max}$  evaluates to 952.75 for the middle of the interval, and  $\frac{k}{2}ln(1.5 \times 2^{\ell-1})$  evaluates to approximately 88.57 so we can expect to find on average 952.75/88.57  $\approx 10.75$  valid PMNS per  $\gamma$ . Experimentally we find on average about 10.78 valid primes per  $\gamma$  with a uniform sampling in that interval which fits our estimation. Furthermore, with an exhaustive search for values  $\gamma$  such that  $\gamma \equiv 0 \pmod{2^{32}}$ , we find at least one corresponding prime for each  $\gamma$ .

As k increases,  $\gamma$  will naturally increase (see equation 36). Seeing as  $\lambda_{\max}$  is inversely proportional to  $\gamma$  (see remark 23),  $\lambda_{\max}$  is therefore inversely proportional to k. As seen earlier since the average number of primes per  $\gamma$  can be expressed as  $\frac{\lambda_{\max}}{\left\lceil \frac{k}{2}ln(1.5\times 2^{\ell-1})\right\rceil}$ , it is, therefore, proportional to  $\lambda_{\max}$  and consequently the average number of primes for a given interval will decrease as k increases. As an empirical example, while we estimate that there are approximately  $2^{51}$  valid prime numbers for the parameter set  $(\ell, n, k, \alpha) = (256, 5, 1, 1)$ , for  $(\ell, n, k, \alpha) = (256, 5, 11, 1)$  we estimate that there are only  $2^{48}$  primes that fall within our bounds.

# 6.4 Specific sparse inverse matrices

As a consequence of the shape of  $\mathcal{G}^{-1}$ , we get that if  $\gamma^2 \equiv 0 \pmod{\phi}$ , the inverse matrix will have exactly 2 non-0 coefficients per row and 2 non-0 coefficients per column if reduced modulo  $\phi$ . This is because each column and each row has all the powers of  $\gamma$  from 0 to n-1 included, with none of them repeating. Hence the only coefficients that won't become 0 after modular reduction are the ones with  $\gamma^0$  and  $\gamma^1$ . Furthermore,  $kp = \alpha \gamma^n - \lambda$  so  $kp \equiv -\lambda \pmod{\phi}$  which means  $\frac{-\lambda}{kp} \equiv 1 \pmod{\phi}$ . We thus get the following matrix:

	$\begin{pmatrix} 0\\ 1 \end{pmatrix}$	$\begin{array}{c} 0 \\ 0 \end{array}$	$\begin{array}{c} 0 \\ 0 \end{array}$		 	$\begin{array}{c} 0 \\ 0 \end{array}$	$-\alpha\gamma(kp)^{-1}$ 0	$\frac{-(kp)^{-1}}{-\gamma(kp)^{-1}}$	)
	$\gamma$	1	0	0		0	0	$\binom{np}{0}$	
	0	$\gamma$	1	0		0	0	0	
$\mathcal{G}^{-1}\equiv$		÷	·	۰.	÷	÷	:	:	$\pmod{\phi}$
		÷	÷	۰.	۰.	÷	:	•	
	:	÷	÷	÷	۰.	۰.	:	•	
	0	0	0	0		$\gamma$	1	0 /	/

Remark 25. Note that k and p are chosen such that they are invertible mod  $\phi$ .

As noted in the previous section (section 5.2, on page 14) we refer to this specific combination of having both  $\mathcal{G}$  and its inverse as sparse matrices as DoubleSparse instead of just LinearRed.

Remark 26. As noted in [7], this can be expanded to have exactly  $\kappa$  non-0 coefficients per row and per column for any  $2 \leq \kappa < n$  by choosing  $\gamma^{\kappa} \equiv 0 \pmod{\phi}$ .

# 7 Cost analysis and implementations

This Section details the cost analysis and performances of the algorithms with this new PMNS shape. Note that the shapes detailed in Sections 5 and 6 in practice have both the same theoretical operations cost and in practice our measures support that fact. Hence we will treat both cases as being the same for the sequel of this Section.

# 7.1 Operation costs

A full modular multiplication in PMNS is done through first a polynomial multiplication and external reduction in one go, and then the internal reduction step is applied.

# POLYNOMIAL MULTIPLICATION AND EXTERNAL REDUCTION

First, as to the combined polynomial multiplication and external reduction step, we adapt [27, Algorithm 1] (see **Algorithm 4** on page 6), since we use  $E(X) = \alpha X^n - \lambda$ , by transforming it into a normal vector-matrix multiplication.

The goal is to compute  $C(X) = A(X) \times B(X) \pmod{E(X)}$ . We consider the vectors  $(a_0, a_1, \dots, a_{n-1})$ and  $(b_0, b_1, \dots, b_{n-1})$  representing the coefficients of A and B respectively. Each coefficient of C can be written as  $c_i = \alpha \sum_{i=1}^{i} a_j b_{i-j} + \lambda \sum_{i=0}^{n-i-1} a_{i+j} b_{n-j}$ . This means we can rewrite the operation as follows:

$$(c_0, c_1, \dots, c_{n-2}, c_{n-1}) = (a_0, a_1, \dots, a_{n-2}, a_{n-1}) \times \begin{pmatrix} \alpha b_0 & \alpha b_2 \dots & \alpha b_{n-2} & \alpha b_{n-1} \\ \lambda b_{n-1} & \alpha b_0 \dots & \alpha b_{n-3} & \alpha b_{n-2} \\ \vdots & \ddots & \ddots & \vdots \\ \lambda b_2 & \lambda b_3 \dots & \alpha b_0 & \alpha b_1 \\ \lambda b_1 & \lambda b_2 \dots & \lambda b_{n-1} & \alpha b_0 \end{pmatrix}$$

This is notable because the matrix is a Toeplitz matrix and thus allows us to use the Toeplitz Recursive Splitting vector-matrix multiplication algorithm [16]. Previous works [22] have shown that this is better than using recursive Karatsuba multiplication.

Constructing the matrix itself costs n multiplications by  $\alpha$  and n-1 multiplications by  $\lambda$ . After this, the vector-matrix multiplication itself is computed recursively. As shown in [19, Equation 22], the number of integer multiplications of a vector of size  $\kappa$  with a  $\kappa \times \kappa$  size matrix with the Toeplitz algorithm is  $\kappa^{\log_{\kappa}\left(\frac{\kappa(\kappa+1)}{2}\right)}$  or in other words  $\frac{\kappa(\kappa+1)}{2}$ . If we take  $n = \kappa^{z}$  then we can split the  $n \times n$  multiplication into  $n^{\log_{\kappa}\left(\frac{\kappa(\kappa+1)}{2}\right)}$  integer multiplications instead which is equal to  $\kappa^{z \times \log_{\kappa}\left(\frac{\kappa(\kappa+1)}{2}\right)}$  which becomes  $\left(\frac{\kappa(\kappa+1)}{2}\right)^{z}$ . If we write  $n = \kappa_{1}^{z_{1}} \times \kappa_{2}^{z_{2}} \times \cdots \times \kappa_{g}^{z_{g}}$  with each  $\kappa_{i}$  a prime factor of n, then the overall number of integer multiplications of a full recursive Toeplitz split of n would be  $\prod_{i=1}^{g} \left(\frac{\kappa_{i}(\kappa_{i}+1)}{2}\right)^{z_{i}}$ . Added together with the multiplications needed to construct the matrix in the first place gets us a total of

$$2n - 1 + \prod_{i=1}^{g} \left(\frac{\kappa_i(\kappa_i + 1)}{2}\right)^z$$

multiplications.

*Remark 27.* The number of multiplications is sub-quadratic in *n*. For each prime factor  $\kappa_i$ , the complexity is essentially  $(\frac{\kappa_i+1}{\kappa_i})\kappa_i^{2-\log_{\kappa_i}2}$ , which means at worst for prime *n* the complexity is  $O(n^{2-\log_n 2})$ .

Meanwhile, the number of additions is also given by [19, Equation 22]. However, that paper was aimed at hardware implementations so some shortcuts are available in software that wouldn't be realistic otherwise due to surface constraints.

If this time we write the decomposition of n as  $n = \kappa_0 \times \kappa_1 \times \kappa_2 \times \cdots \times \kappa_f$  where each  $\kappa_i$  for  $i \ge 1$  is a prime factor of n such that  $\forall i < j, \kappa_i \le \kappa_j$  and  $\kappa_0 = 1$  (notice that this isn't the same exact decomposition as earlier) the number of additions ends up being:

$$\sum_{i=1}^{f} \left( \left( \prod_{j=0}^{i-1} \frac{\kappa_j(\kappa_j+1)}{2} \right) \times \left( \left( \frac{2n}{\prod_{j=1}^{i} \kappa_j} - 1 \right) (3\kappa_i - 4)A_s + \frac{n}{\prod_{j=0}^{i-1} \kappa_j} \left( \frac{\kappa_i - 1}{2}A_s + (\kappa_i - 1)A_d \right) \right) \right)$$

where  $A_s$  represents additions on a simple register and  $A_d$  additions made on 2 registers.

The proof is detailed in Appendix A.

# **INTERNAL REDUCTION**

The internal reduction step is performed with **Algorithm 3**. The time complexity for computing T is a vector-matrix multiplication by  $\mathcal{G}$ . Because of the shape of  $\mathcal{G}$ , the total cost is (n + 1) multiplications and n additions.

For LinearRed, i.e. the general case, computing Q involves first computing the last coefficient which is (n-1) multiplications followed by (n-1) additions. Each of the other coefficients of Q is an additional multiplication and addition from that which means a total of (2n-2) multiplications and (2n-2) additions to compute Q fully. Note that because of the modular reduction, those are additions on a single register.

Meanwhile, for DoubleSparse,  $\mathcal{G}^{-1}$  being sparse means the corresponding costs become (n+1) multiplications and n additions just like for sparse  $\mathcal{G}$ .

In both cases, computing S is simply n additions, the division by  $\phi$  being considered "free" since it is at worst a single MOV per coefficient (since  $\phi$  is chosen to be the size of a register).

Shape $\setminus$ Step	Computing Q	Computing T	Computing S	Total
LinearRed	$(2n-2)M + (2n-2)A_s$	$(n+1)M + nA_d$	$nA_d$	$(3n-1)M + 2nA_d + (2n-2)A_s$
DoubleSparse	$(n+1)M + nA_s$	$(n+1)M + nA_d$	$nA_d$	$(2n+2)M + 2nA_d + (n-1)A_s$

# 7.2 Implementations

The following procedure has been applied for every measure done for this paper :

- the  $Turbo-Boost(\mathbf{\hat{R}})$  is deactivated during the tests;
- 501 runs are executed in order to "heat" the cache memory, that is to say we ensure that the cache memory (data and instruction) is in a stable enough state in order to avoid untimely cache misses;
- one generates 1001 random data sets, and the number of clock cycle for W executions over a batch of 501 runs is recorded for each data set;
- the performance is the median value divided by W.

(Note that W varies depending on the parameter sizes). All code is available at https://github.com/linearRedPMNS. More specifically, the code to generate the following tables is available at https://github.com/linearRedPMNS/pmns\_with\_gamma\_small.

# **ECC-SIZE INTEGERS**

PMNS are known to be effective for prime sizes used for elliptic curves (see [9]) so we chose primes from https://safecurves.cr.yp.to/ for comparison with our new class of PMNS-friendly primes. Most of the primes are Pseudo-Mersenne, with the exception of the E-521 Mersenne prime.

$\begin{tabular}{lllllllllllllllllllllllllllllllllll$	255	383	414	511	521
Corresponding Curve	Curve25519	M-383	Curve41417	M-511	E-521
Optimized Multiprecision	61	104	103	151	136
DoubleSparse PMNS	64	108	109	153	153
LinearRed PMNS	77	123	123	166	166
Adapted GMP low level	94	138	164	197	223

Table 2: Cycle count for modular multiplication on ECC-size integers using gcc 12.3.0 on intel processor i9-11900KF

The algorithm used for our benchmarks of "Optimized Multiprecision" in tables 2 and 5 bases itself on code from Adam Langley (https://code.google.com/archive/p/curve25519-donna/) which adapts Daniel J. Bernstein's code from Curve25519 to seamless 128-bit arithmetic offered by modern CPUs. We then adapt the code to the other primes, including the Mersenne prime. The "Adapted GMP low level" benchmarks refer to the use of low-level GMP primitives to perform a (Pseudo) Mersenne modular multiplication (as opposed to using the general-use mpn\_tdiv\_qr to perform the reduction).

As can be seen in table 2 our benchmarks are close to optimized Pseudo-Mersenne performances

Method $\setminus$ Prime size	255	383	414	511	521
Corresponding Curve	Curve25519	M-383	Curve41417	M-511	E-521
Optimized Multiprecision	66	130	113	235	194
DoubleSparse PMNS	66	118	122	162	163
LinearRed PMNS	72	127	127	177	180
Adapted GMP low level	89	140	164	202	222

Cycle count for modular multiplication on ECC-size integers using clang 14.0.0 on intel processor i9-11900KF

with a much greater density of valid primes. For example at the 255-bit prime size the PMNS we use to compare ourselves to Curve25519 arithmetic is the same prime, that is to say  $p = 2^{255}-19$ , and yet we are only a few cycles apart from the very optimized code widely used in cryptography.

In Table 3, we show how an increase in n affects performance. Of note is that due to the recursive Toeplitz splitting approach we use for multiplication sometimes bigger values of n produce faster running code such as n = 12 being faster than n = 11 due to 11 being a prime number. We thus cannot split it further into smaller sub-matrices unlike for n = 12.

n					13	
DoubleSparse PMNS						
LinearRed PMNS	175	208	300	276	394	353

Table 3: Cycle count for modular multiplication for 512-bit integers for various values of n using gcc 12.3.0 on intel processor i9-11900KF

In Table 4, we show that specific shapes of external reduction polynomial, such as  $\alpha X^n - 1$  and  $X^n - \lambda$ , result in almost identical performance compared to the more general  $\alpha X^n - \lambda$ .

E	$X^9 - 2$	$X^{9} - 3$	$2X^9 - 1$	$2X^9 - 3$	$3X^9 - 1$	$3X^9 - 2$
DoubleSparse PMNS	154	154	156	156	156	156
LinearRed PMNS	167	167	173	172	172	172

Table 4: Cycle count for modular multiplication for 512-bit integers for various shapes of E using gcc 12.3.0 on intel processor i9-11900KF

Prime fields in Elliptic Curve Cryptography are usually chosen for the underlying prime number's properties surrounding fast modular arithmetic, as can be seen by the fact that the primes chosen are usually Mersenne, Pseudo-Mersenne or Generalized Mersenne. Furthermore, the prime sizes are usually chosen for speed considerations as long as they satisfy specific security thresholds. We have shown in table 2 that this work generates a class of primes with which curves can be constructed with similar performance as existing elliptic curves based on Mersenne (or Mersenne derivative) primes for a given size with no impact on the resulting security level. However, the prime sizes were chosen to be optimal for a positional number system and not necessarily optimal for PMNS. In table 5 we propose various prime sizes for which our work gives better performances than corresponding Pseudo-Mersenne primes of the same size in a positional system, showing a proof of concept for potential applications. To be clearer, the Pseudo-Mersenne primes used for comparison in table 5 are only used because they are the most advantageous at this size for multi-precision arithmetic in terms of speed and not out of any consideration for constructing viable Elliptic Curves with them.

With that said, as part of our proof of concept, we have constructed good elliptic curves with our PMNS-friendly primes using the Brainpool [8] Standard, adapting a SageMath curve generation code from http://bada55.cr.yp.to/brainpool.html. Examples of such curves generated with our primes are available at https://github.com/linearRedPMNS/pmns\_with\_gamma\_small/tree/main/curves and the corresponding PMNS are also available on the same GitHub repository.

Remark 28. Note that for some parameters we will always have  $p \equiv 1 \pmod{4}$  such as choosing  $p = \gamma^4 \pm 1$  for our 244 bit primes or  $p = \gamma^8 \pm 1$  for 480 bit. This technically goes against the requirement of the Brainpool standard of needing  $p \equiv 3 \pmod{4}$ . However, this requirement, according to the Brainpool standard, "allows efficient point compression" [8, Section 3.1] and is not linked to any security requirement. Note that we can construct elliptic curves verifying every other Brainpool criteria with primes generated from this work's generation algorithms and in the general case, the primes we generate show no strong bias towards being congruent to either 3 or 1 mod 4.

$\begin{tabular}{ c c c c c } \hline Method \ \ \ Prime size \end{tabular}$	244	297	354	480
Corresponding Pseudo-Mersenne	$2^{244} - 189$	$2^{297} - 123$	$2^{354} - 153$	$2^{480} - 47$
Optimized Multiprecision	63	82	103	156
DoubleSparse PMNS	47	66	85	123
LinearRed PMNS	55	78	94	135
Adapted GMP low level	93	115	138	197

Table 5: Cycle count for modular multiplication on proof of concept sizes of integers chosen at advantageous breakpoints for PMNS using gcc 12.3.0 on intel processor i9-11900KF

In Table 6, we give proof of concept benchmarks for large primes for when we can choose the exact prime arbitrarily (such as for Diffie-Hellmann). We compare ourselves to "Adapted GMP low level" (see above) which automatically uses subquadratic algorithms for large sizes. Optimized algorithms in the vein of those used for smaller integers could be faster but no example code exists for the arbitrary prime sizes which were chosen for optimal construction of PMNS with specific n for best Toeplitz-Matrix splitting performance. In [22], GMP's implementation of the Montgomery modular multiplication was shown to be faster than PMNS for sizes above the 2048-bit range unless multi-threading was used. Table 6 shows PMNS can beat GMP using Pseudo-Mersenne specific arithmetic, which is faster than the Montgomery modular multiplication, while single-threaded.

# 8 Conclusion

In this work we have made the link between two PMNS-friendly classes of primes from [9] and [7] to construct a broader class that includes them while giving performances comparable to Pseudo and Generalized Mersenne primes. We have also improved on the parameter bounds of previous works on PMNS while adapting conversion algorithms in consequence. We utilize

Method $\setminus$ Prime size	1023	2002	3979	7813
Corresponding Pseudo-Mersenne	$2^{1023} - 361$	$2^{2002} - 297$	$2^{3979} - 3819$	$2^{7813} - 241$
DoubleSparse PMNS	522	1580	5037	15049
LinearRed PMNS	573	1734	5351	15593
Adapted GMP low level	641	1933	6230	18409

Table 6: Cycle count for modular multiplication on large integers using gcc 12.3.0 on intel processor i9-11900KF

the works from [22] and [12] and show we can construct sub-lattices with interesting properties for internal reduction instead of relying on LLL-reduced bases. Additionally, we constructed a database of Elliptic Curves using the Brainpool standard using our friendly primes which ensures fast low-level arithmetic with PMNS. Finally, we also introduced the concept of Mirror PMNS which is applicable to the general case and may lead to interesting applications in the future.

# References

- 1. Sec 2. standards for efficient cryptography group: Recommended elliptic curve domain parameters. Certicom Research (2010)
- List of prime numbers wikipedia (2024), https://en.wikipedia.org/wiki/List\_of\_prime\_ numbers#Lists\_of\_primes\_by\_type, last accessed 23 Sep 2024
- Aranha, D.F., Barreto, P.S.L.M., Pereira, G.C.C.F., Ricardini, J.E.: A note on high-security general-purpose elliptic curves. Cryptology ePrint Archive, Paper 2013/647 (2013), https:// eprint.iacr.org/2013/647
- Bajard, J.C., Imbert, L., Plantard, T.: Modular number systems: Beyond the mersenne family. In: Selected Areas in Cryptography, 11th International Workshop, SAC 2004, Waterloo, Canada. pp. 159–169 (2004). https://doi.org/10.1007/978-3-540-30564-4\_11
- Bajard, J.C., Marrez, J., Plantard, T., Véron, P.: On Polynomial Modular Number Systems over Z/pZ. Advances in Mathematics of Communications 18(3), 674–695 (Jun 2024). https://doi.org/10.3934/amc.2022018
- Bernstein, D.J.: Curve25519: New diffie-hellman speed records. In: Public Key Cryptography PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography. Lecture Notes in Computer Science, vol. 3958, pp. 207–228. Springer (2006). https://doi.org/10.1007/11745853 14
- Bouvier, C., Imbert, L.: An alternative approach for sidh arithmetic. In: Garay, J.A. (ed.) Public-Key Cryptography – PKC 2021. pp. 27–44. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-75245-3 2
- 8. Brainpool, E.C.C.: Ecc brainpool standard curves and curve generation (2005), https://web.archive.org/web/20070814070853/http://www.ecc-brainpool.org/download/ Domain-parameters.pdf
- 9. Coladon, T., Elbaz-Vincent, P., Hugounenq, C.: MPHELL: A fast and robust library with unified and versatile arithmetics for elliptic curves cryptography. In: ARITH 2021. Transactions on Emerging Topics in Computing, Torino, Italy (Jun 2021). https://doi.org/10.1109/ARITH51176.2021.00026
- Crandall, R.E.: Method and apparatus for public key exchange in a cryptographic system. US Patent 5,159,632. (1992), US Patent Application US07/761, filed 1991-09-17.
- Didier, L.S., Dosso, F.Y., Véron, P.: Efficient modular operations using the Adapted Modular Number System. Journal of Cryptographic Engineering pp. 1-23 (2020). https://doi.org/10.1007/s13389-019-00221-7, https://arxiv.org/pdf/1901.11485
- Dosso, F.Y., Berzati, A., El Mrabet, N., Proy, J.: PMNS revisited for consistent redundancy and equality test. Cryptology ePrint Archive, Paper 2023/1231 (2023), https://eprint.iacr.org/ 2023/1231

- Dosso, F.Y., Robert, J.M., Véron, P.: PMNS for efficient arithmetic and small memory cost. IEEE Transactions on Emerging Topics in Computing 10(3), 1263–1277 (2022). https://doi.org/10.1109/tetc.2022.3187786
- Dubner, H.: Generalized Repunit Primes. Mathematics of Computation 61(204), 927–930 (1993), https://doi.org/10.2307/2153263
- Dubner, H., Granlund, T.: Primes of the Form (b<sup>n</sup>+1)/(b+1). Journal of Integer Sequences 3, 0-7 (09 2000), http://www.kurims.kyoto-u.ac.jp/EMIS/journals/JIS/VOL3/DUBNER/dubner.pdf
- Fan, H., Hasan, M.: Alternative to the karatsuba algorithm for software implementation of GF(2<sup>n</sup>) multiplication. IET Information Security 3, 60–65(5) (June 2009). https://doi.org/10.1049/ietifs.2007.0132
- 17. Granlund, T.: Instruction latencies and throughput for amd and intel x86 processors (2019), https://gmplib.org/~tege/x86-timing.pdf
- 18. Hadamard, J.: Sur la distribution des zéros de la fonction  $\zeta(s)$  et ses conséquences arithmétiques. Bulletin de la Société Mathématique de France **24**, 199–220 (1896). https://doi.org/10.24033/bsmf.545
- Hasan, M.A., Nègre, C.: Multiway splitting method for toeplitz matrix vector product. IEEE Transactions on Computers 62, 1467–1471 (2013). https://doi.org/10.1109/TC.2012.95
- Korkine, A., Zolotareff, G.: Sur les formes quadratiques positives. Mathematische Annalen 11(2), 242–292 (1877)
- Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische annalen 261, 515–534 (1982)
- Méloni, N., Palma, F., Véron, P.: PMNS for Cryptography : A Guided Tour. Advances in Mathematics of Communications (2023). https://doi.org/10.3934/amc.2023033
- 23. Minkowski, H.: Zur Geometrie der Zahlen. B. G. Teubner (1905)
- Negre, C., Plantard, T.: Efficient modular arithmetic in adapted modular number system using lagrange representation. In: Information Security and Privacy, 13th Australasian Conference, ACISP 2008, Wollongong, Australia. pp. 463–477 (2008). https://doi.org/10.1007/978-3-540-70500-0\_34
- Nguyen, P.Q., Stehlé, D.: LLL on the Average. In: Proceedings of the 7th Algorithmic Number Theory Symposium (2006). https://doi.org/10.1007/11792086\_18
- Noyez, L., El Mrabet, N., Potin, O., Véron, P.: Modular multiplication in the AMNS representation : Hardware Implementation. In: Selected Areas in Cryptography. Montréal (Québec), France (Aug 2024)
- Plantard, T.: Polynomial modular number system on any prime fields using binomials. WRACH 2023 : Workshop on Randomness and Arithmetics for Cryptographic Hardware, https:// wrach2023.sciencesconf.org/file/955367
- Schnorr, C.P., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. In: International Symposium on Fundamentals of Computation Theory. pp. 68–85. Springer (1991). https://doi.org/10.1007/3-540-54458-5 51
- 29. Sloane, N.J.A.: Prime p with prime gap q p of *n*-th record cramer-shanks-granville ratio (2005), https://oeis.org/A111943
- Solinas, J.: Generalized mersenne numbers. Research Report CORR-99-39, Center for Applied Cryptographic Research, University of Waterloo, Waterloo, ON, Canada (1999)
- 31. Stein, W., al.: Sagemath (2005), http://www.sagemath.org/, last accessed 07 Jul 2022
- 32. de La Vallée Poussin, C.J.: Recherches analytiques sur la théorie des nombres premiers. Annales de la Société scientifique de Bruxelles 20B, 21B, 183–256, 281–352, 363–397, 351–368 (1896)

# Appendix A Addition count in Recursive Toeplitz Splitting algorithm

Since we have a Toeplitz matrix of size n we can split it as follows:

$$\begin{pmatrix} M_0 & M_1 & M_2 & \dots & M_{\kappa-3} & M_{\kappa-2} & M_{\kappa-1} \\ M_{\kappa} & M_0 & M_1 & \dots & M_{\kappa-4} & M_{\kappa-3} & M_{\kappa-2} \\ M_{\kappa+1} & M_{\kappa} & M_0 & \dots & M_{\kappa-5} & M_{\kappa-4} & M_{\kappa-3} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ M_{2\kappa-2} & M_{2\kappa-3} & M_{2\kappa-4} & \dots & M_{\kappa+1} & M_{\kappa} & M_0 \end{pmatrix}$$

with each  $M_i$  a submatrix of dimension  $\frac{n}{\kappa}$  (with  $\kappa$  a divisor of n). Similarly, we take our vector V of size n and split it into  $\kappa$  sub vectors of size  $\frac{n}{\kappa}$ . The number of additions can be broken down as follows: for each resulting sub-vector a total of  $\frac{\kappa(\kappa+1)}{2}$  sub-products must be performed. The first  $\kappa$  subproducts require  $\kappa - 2$  matrix additions each, which gives us  $\kappa(\kappa - 2)$  additions. They are as follows:

$$\begin{cases} P_0 = V_0(M_0 + M_1 + \dots + M_{\kappa-2} + M_{\kappa-1}) \\ P_1 = V_1(M_{\kappa} + M_0 + \dots + M_{\kappa-3} + M_{\kappa-2}) \\ P_2 = V_2(M_{\kappa-1} + M_{\kappa} + \dots + M_{\kappa-4} + M_{\kappa-3}) \\ \dots \\ P_{\kappa-1} = V_{\kappa-1}(M_{2\kappa-2} + M_{2\kappa-3} + \dots + M_{\kappa} + M_0) \end{cases}$$

with each  $P_i$  a corresponding subproduct.

However, some of these sums have common terms so they can be reduced to  $3\kappa - 4$  additions in total through partial sums. We first compute  $P_s = \sum_{i=0}^{\kappa-2} M_i$  which comes down to  $\kappa - 2$  matrix additions. We then have  $P_0 = V_0(P_s + M_{\kappa-1})$  and  $P_1 = V_1(P_s + M_{\kappa})$ . This allows us to compute the first two subproducts in just  $\kappa$  matrix additions instead of  $2(\kappa - 1)$  matrix additions total like in [19]. For the remaining  $\kappa - 2$  subproducts, they can be computed through one matrix subtraction and one matrix addition to the previous partial sum. For example for  $P_2$ , having already computed  $M_{\kappa} + \sum_{i=0}^{\kappa-2} M_i$  for  $P_1$ , we keep the partial sum in memory and add  $M_{\kappa-1}$  while subtracting  $M_{\kappa-2}$ . So we add  $2(\kappa - 2)$  additions for the remaining subproducts which give us  $3\kappa - 4$  matrix additions in total.

As mentioned earlier, each submatrix is a Toeplitz  $\frac{n}{\kappa} \times \frac{n}{\kappa}$  matrix with only  $2\frac{n}{\kappa} - 1$  distinct coefficients. This means so far the number of integer additions comes down to  $(2\frac{n}{\kappa} - 1)(3\kappa - 4)$ . After this, the remaining  $\frac{\kappa(\kappa-1)}{2}$  subproducts require a vector subtraction each which gives us an additional  $\frac{n}{\kappa} \times \frac{\kappa(\kappa-1)}{2}$  integer additions. Once all the sub-products are performed the final results have to be computed. This is done through an additional  $\kappa(\kappa-1)$  vector additions/subtractions which comes down to  $\frac{n}{\kappa} \times \kappa(\kappa - 1)$  more integer additions. Note that those additions are performed on 2 registers in practice so they are more costly. For brevity, we note  $A_s$  additions on a simple register and  $A_d$  additions made on 2 registers. This gives us  $(2\frac{n}{\kappa} - 1)(3\kappa - 4)A_s + n(\frac{\kappa-1}{2}A_s + (\kappa - 1)A_d)$  additions in summary.

Let us now break down the overall number of additions. As noted on page 27 in section 7.1, we write the decomposition of n as  $n = \kappa_0 \times \kappa_1 \times \kappa_2 \times \cdots \times \kappa_f$  where each  $\kappa_i$  for  $i \ge 1$  is a prime factor of n such that  $\forall i < j, \kappa_i \le \kappa_j$  and  $\kappa_0 = 1$ . For the top level of the recursion, we get  $(2\frac{n}{\kappa_1} - 1)(3\kappa_1 - 4)A_s + n(\frac{\kappa_1-1}{2}A_s + (\kappa_1 - 1)A_d)$ . We then perform  $\frac{\kappa_1(\kappa_1+1)}{2}$  sub products. For each of them,  $(2\frac{n}{\kappa_1\kappa_2} - 1)(3\kappa_2 - 4)A_s + \frac{n}{\kappa_1}(\frac{\kappa_2-1}{2}A_s + (\kappa_2 - 1)A_d)$  additions will be performed assuming n is not prime. We then proceed recursively and get

$$\sum_{i=1}^{f} \left( \left( \prod_{j=0}^{i-1} \frac{\kappa_j(\kappa_j+1)}{2} \right) \times \left( \left( \frac{2n}{\prod_{j=1}^{i} \kappa_j} - 1 \right) (3\kappa_i - 4)A_s + \frac{n}{\prod_{j=0}^{i-1} \kappa_j} \left( \frac{\kappa_i - 1}{2}A_s + (\kappa_i - 1)A_d \right) \right) \right)$$

*Remark 29.* In practice, we perform the bottom level of the recursion with the schoolbook algorithm instead. At that scope using the schoolbook algorithm minimises the memory usage and movements required. Meanwhile, using Toeplitz on recursion levels above means minimizing the number of calls to the bottom level.