PMNS for Efficient Arithmetic and Small Memory Cost

Fangan Yssouf Dosso, Jean-Marc Robert, Pascal Veron

PMNS for efficient arithmetic and small memory cost

Fangan Yssouf Dosso, Jean-Marc Robert, and Pascal Véron

Abstract—The Polynomial Modular Number System (PMNS) is an integer number system which aims to speed up arithmetic operations modulo a prime p. Such a system is defined by a tuple (p, n, γ, ρ, E), where p, n, γ and ρ are positive integers, $E \in \mathbb{Z}[X]$, with $E(\gamma) \equiv 0 \pmod p$. In [15] conditions required to build efficient AMNS (PMNS with $E(X) = X^n - \lambda$, where $\lambda \in \mathbb{Z} \setminus \{0\}$) are provided. In this paper, we generalise their approach for any monic polynomial $E \in \mathbb{Z}[X]$ of degree n. We present new bounds and highlight a set of polynomials E for very efficient operations in the PMNS and low memory requirement. We also provide AMNS and PMNS modular multiplication implementations, for a prime of size 256 bits, in classic C. We also provide, for the same prime, the first implementation taking advantage of the SIMD `AVX512` instruction set. The `AVX512` PMNS is 72 % faster than its AMNS counterpart (classical C version). This version presents a more than 60 % speed-up in comparison with the state-of-the-art Montgomery-CIOS modular multiplication of the `GMP` library.

Index Terms—Modular arithmetic Polynomial modular number system External reduction

1 Introduction

Modular arithmetic is an essential building block of many topics in computer algebra, like cryptography [31], the polynomial factorization [21] and Gröbner basis algorithms [5]. Every progress in the area of modular arithmetic has a direct impact on the efficiency of the algorithms used in any of these topics. Many approaches have been proposed to perform arithmetic operations efficiently. When the use case allows it, it is possible to use special modulo belonging to the family of Mersenne numbers [12], which allows very efficient modular reduction. For the general case, when choosing such modulo is not possible, several methods exist. Montgomery and Barrett modular reduction methods [26], [10] are among the best ones. As alternatives approaches to improve modular arithmetic in general case, new representation systems have been proposed, among them one may mention the RNS (Residue Number System) [18] and the PMNS (Polynomial Modular Number System) [8].

PMNS is a representation system for elements in $\mathbb{Z}/p\mathbb{Z}$, introduced by J.C. Bajard, L. Imbert and T. Plantard. In order to compute $a \star b \bmod p$, where $\star = +$ or $\times$, a and b are mapped to two polynomials $A(X)$ and $B(X)$ whose degrees are strictly lower than some integer n, and whose coefficients are all bounded by some fixed value ρ. The result is obtained by computing $A(X) \star B(X)$ modulo a polynomial $X^n + \alpha X + \beta$, where $\alpha,\ \beta \in \mathbb{Z}$. In [14], the definition of a PMNS has been enlarged to take into account any monic polynomial E of degree n.

- *J.-M. Robert and P. Véron are with Université de Toulon, Laboratoire IMath, Toulon, France.*
  E-mail: jean-marc.robert,pascal.veron@univ-tln.fr
- *F. Y. Dosso is with École des Mines de Saint-Étienne, Département SAS, Gardanne, France .*
  E-mail: fanganyssouf.dosso@emse.fr

Definition 1. Let $p \geqslant 3, n \geqslant 2, \gamma \in [1, p-1]$ 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) $\forall A(X) = \sum_{i=0}^{n-1} a_i X^i \in \mathcal{B}, -\rho < a_i < \rho$ for all i,
3) $\forall a \in \mathbb{Z}/p\mathbb{Z}, \exists A \in \mathcal{B}$ such that $A(\gamma) \equiv a \pmod p$.
   So A is a **representation** of a in $\mathcal{B}$ and we denote $A \equiv a_{\mathcal{B}}$.

The tuple (p, n, γ, ρ, E) characterizes the PMNS $\mathcal{B}$.
When $E(X) = X^n - \lambda$, with $\lambda \in \mathbb{Z} \setminus \{0\}$, $\mathcal{B}$ is also called AMNS (Adapted Modular Number System)[7].

Modular arithmetic is at the core of numerous public key cryptosystems. In this context, the efficiency of the AMNS has been proved in recent works :

- in [15], when the modulus is a prime p whose size fits the standard sizes used in elliptic curve cryptography, it has been shown that a software implementation of the AMNS performs well compared to the standard libraries like GnuMP [4] or OpenSSL [29],
- in [13], the efficiency has been confirmed in the MPHELL library for other platforms (Armv8, STMF32F4),
- in [11], the authors generalized the representation system to the field $\mathbb{F}_{p^k}$ and improved this way the performances of the SIKE protocol [3], one of the alternate KEM candidate of the NIST post-quantum standardization process [2].

The main operations in the PMNS are addition and multiplication, and these operations must be consistent, i.e. they must return elements in the system, such that their degree and infinity norm are strictly lower than respectively n and ρ. Let $A, B \in \mathcal{B}$, $R = A \times B$ and $S = A + B$. While S has a suitable degree ($\deg(S) < n$), the degree of R is not bounded by n ($\deg(R) < 2n - 1$). To get an element

of appropriate degree, we compute $C = R \bmod E$. Since $\deg(E) = n$ and $E(\gamma) \equiv 0 \pmod p$, we have $C(\gamma) \equiv R(\gamma) \pmod p$, thus C represents the same element as R in $\mathbb{Z}/p\mathbb{Z}$. The operation $R \bmod E$ is called the **external reduction**; more details are given in Section 3.2.

It is now necessary to ensure that the results have their infinity norm lower than ρ, which is generally not the case for S and C. This is done by an operation called the **internal reduction**. Many methods have been proposed to perform this operation; some give total freedom on the choice of p [8], [28] and some does not [7]. In this paper, we focus on the Montgomery-like method proposed in [28], which is currently the best choice among the ones giving total freedom of the modulus p. We describe this method in Section 3.3.

2 Contributions

Since the seminal work on PMNS [7], there have been several papers on this topic. The optimisation of the internal reduction is discussed in [28]. Practical use of the PMNS in the area of cryptography is described in [17], [16], [11], [13]. The redundancy of this representation system is used to protect modular operations in cryptographic protocols [14], [27]. A precise study on this system concerning its existence, how to generate the set of parameters, and its efficiency compare to well-known multi precision libraries is given in [15].

All research on this topic has been focused on the use of polynomials $E(X) = X^n - \lambda$ (with λ "small"), in order to guarantee the efficiency of the external reduction operation as well as to minimize the value of ρ. The purpose is to minimize the theoretical number of bits required to represent an integer mod p, which is $t = n(\lfloor \log_2(2\rho-1) \rfloor + 1)$. In practice, in case of software implementation on a $k-$bit architecture, such integers will be stored in $\lceil \frac{t}{k} \rceil$ machine words. One may notice however that for hardware implementations, the space used could be exactly equal to t bits. Among the polynomials $X^n - \lambda$, the most interesting is $E(X) = X^n \pm 2$, which leads to efficient external reduction using only shifts and additions. Moreover, it provides the smallest value of ρ, see section 7.4.

In this paper, we highlight some other polynomials suited to design PMNS presenting similar performances as the ones of AMNS defined with $E(X) = X^n \pm 2$. In order to use these polynomials, we first generalise the theoretical results and practical tools described in [15] to any monic polynomial $E \in \mathbb{Z}[X]$ of degree n. Next, we identify a set of polynomials E which provide both efficiency and small value for ρ. Finally, using `AVX512` instruction set, we present efficient implementations of a PMNS.

This paper is organised as follows. In Section 3, we remind the background on PMNS. In Section 4, new parameters are introduced in order to make PMNS efficient, given any polynomial $E \in \mathbb{Z}_n[X]$. Then, in Section 5, the existence of PMNS is studied and improvements on existing bounds for consistency are given. In Section 6, essential algorithms to use the PMNS are presented. In Section 7, we explain how to generate efficient PMNS with small memory cost. Next, in Section 8, we discuss implementations and results. Finally, we conclude in Section 9.

3 Background on the PMNS

This section presents all the background on the PMNS. For consistency, we assume in this paper that $p \geqslant 3$ and $n, \gamma, \rho \geqslant 1$.
In the sequel, $\mathbb{Z}_n[X]$ denotes the set of polynomials in $\mathbb{Z}[X]$ whose degree is lower than or equal to n.
If $A \in \mathbb{Z}_n[X]$, we assume $A(X) = a_0 + a_1 X + \cdots + a_n X^n$ and $A(X)$ can equivalently be represented as the vector $a = (a_0, \ldots, a_n)$.
Unless specified, the $\mathcal{B}$ symbol will always designate a PMNS (p, n, γ, ρ, E), where $E \in \mathbb{Z}_n[X]$ is a monic polynomial.
Let $C \in \mathbb{Z}[X]$, then $C \bmod (E, \phi)$ denotes the polynomial reduction $C \bmod E$, where the coefficients of the result are computed modulo ϕ.

3.1 Conversion and arithmetic operations in the PMNS

Except the external reduction process, all the algorithms for conversion and arithmetic operation in a PMNS remain the same regardless the polynomial E. These algorithms are given in Section 6.

3.2 External reduction

The *external reduction* is a polynomial modular reduction, whose purpose is to keep degree of the PMNS elements bounded by n. Let $E \in \mathbb{Z}_n[X]$ be a *monic* polynomial, such that $E(\gamma) \equiv 0 \pmod p$. Let $C \in \mathbb{Z}[X]$ be a polynomial. This operation consists in computing a polynomial R such that:

$$R \in \mathbb{Z}_{n-1}[X] \text{ and } R(\gamma) \equiv C(\gamma) \pmod p$$

The Euclidean division of C by E computes H and R so that $C = H \times E + R$, with $R \in \mathbb{Z}_{n-1}[X]$ and $H \in \mathbb{Z}[X]$. Since $E(\gamma) \equiv 0 \pmod p$, $C(\gamma) = H(\gamma) \times E(\gamma) + R(\gamma) \equiv R(\gamma) \pmod p$. We call *external reduction* the computation of $R = C \bmod E$. The polynomial E is called the *external reduction polynomial*.

In this work, we present efficient approaches to compute this external reduction, for any monic polynomial $E \in \mathbb{Z}_n[X]$.

3.3 Internal reduction

The *internal reduction* aims to ensure that the coefficients of the polynomials are bounded by ρ. Let $V \in \mathbb{Z}_{n-1}[X]$ be a polynomial, with $\|V\|_\infty \geqslant \rho$. The main idea to reduce the coefficients of V is to find a polynomial $D \in \mathbb{Z}_{n-1}[X]$, such that $D(\gamma) \equiv 0 \pmod p$ and $|v_i - d_i| < \rho$, in order to get the polynomial $R = V - D$ such that:

$$R \in \mathcal{B} \text{ and } R(\gamma) \equiv V(\gamma) \pmod p .$$

One of the most efficient methods to perform this operation is a Montgomery-like approach presented in [28] (see Algorithm 1). This algorithm relies upon a polynomial $M \in \mathbb{Z}_{n-1}[X]$ such that $M(\gamma) \equiv 0 \bmod p$. The algorithm computes a polynomial $T \in \mathbb{Z}_{n-1}[X]$ such that all the coefficients of $(V + T)$ are multiples of a fixed parameter ϕ. The polynomial T must be a multiple of M so that $(V + T)(\gamma) \equiv V(\gamma) \bmod p$. Hence, one needs to compute a polynomial Q such that $V + QM = 0$ in $(\mathbb{Z}/\phi\mathbb{Z})[X]/E(X)$.

So, $Q = -M^{-1}V \mod (E, \phi)$, which explains line 1 of Algorithm 1.

A sufficient condition to guarantee that the result S will be in the PMNS is that $\|T\|_\infty$ and $\|V\|_\infty$ are strictly less than $\frac{\rho\phi}{2}$. In Section 5.2, we present tight bounds on ρ, ϕ and the input V to guarantee that these conditions are fulfilled for any monic polynomial E.

Algorithm 1 Coefficients reduction [28]

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

4 New parameters

Let $\mathcal{B}$ be a PMNS such that $E(X) = X^n + e_{n-1}X^{n-1} + \cdots + e_1X + e_0$.

4.1 External reduction matrix

Let us consider the following $(n-1) \times n$ matrix :

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

where each row contains the coefficients of the polynomial $X^{n+i} \pmod E$, for $i = 0, \dots, n-2$. From now on, $\mathcal{E}$ will be called the **external reduction matrix**.

Proposition 1. Let $A, B \in \mathbb{Z}_{n-1}[X]$ be two polynomials. Let C, R be two polynomials, such that:

$$C = AB = c_0 + c_1X + \cdots + c_{2n-2}X^{2n-2},$$
$$R = C \bmod E.$$

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

Proof. We have:

$$R = c_0 + c_1X + \cdots + c_{n-1}X^{n-1} + X^n(c_n + c_{n+1}X + \cdots + c_{2n-2}X^{n-2}) \bmod E$$

We also have $c_{n+i}X^{n+i} \bmod E = c_{n+i} \times \mathcal{E}_i$ (the i-th row of $\mathcal{E}$), for $i = 0, \dots, n-2$. As a consequence,

$$(c_nX^n + c_{n+1}X^{n+1} + \cdots + c_{2n-2}X^{2n-2}) \bmod E = (c_n, \dots, c_{2n-2})\mathcal{E}$$

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

4.2 A matricial view of the modular multiplication

Like the external reduction, the efficiency of the internal reduction could be affected by the polynomial E. Indeed, one can see that operations at lines 1 and 2 of Algorithm 1 depend on E.

Since parameters M, M' and E remain constant once the PMNS generation is done, the polynomial multiplications at lines 1 and 2 of Algorithm 1 can be optimised for efficient and constant time internal reduction operation. Let $\mathcal{M}$ and $\mathcal{M}'$ be the two following matrices :

$$\mathcal{M} = \begin{pmatrix} m_0 & m_1 & \dots & m_{n-1} \\ \dots & \dots & \dots & \dots \\ \vdots & \vdots & & \vdots \\ \dots & \dots & \dots & \dots \end{pmatrix} \begin{matrix} \leftarrow M \\ \leftarrow X.M \bmod E \\ \\ \leftarrow X^{n-1}.M \bmod E \end{matrix} \quad (2)$$

$$\mathcal{M}' = \begin{pmatrix} m'_0 & m'_1 & \dots & m'_{n-1} \\ \dots & \dots & \dots & \dots \\ \vdots & \vdots & & \vdots \\ \dots & \dots & \dots & \dots \end{pmatrix} \begin{matrix} \leftarrow M' \\ \leftarrow X.M' \bmod (E, \phi) \\ \\ \leftarrow X^{n-1}.M' \bmod (E, \phi) \end{matrix} \quad (3)$$

In the sequel, $\mathcal{M}$ and $\mathcal{M}'$ are called the **internal reduction matrices**.

Using these two matrices, lines 1 and 2 of Algorithm 1 become:

$$Q = (v_0, \dots, v_{n-1})\mathcal{M}' \pmod \phi \quad (4)$$
$$T = (q_0, \dots, q_{n-1})\mathcal{M} \quad (5)$$

From Proposition 1, the internal reduction and modular multiplication can be efficiently performed, for any monic polynomial $E \in \mathbb{Z}_n[X]$, using Algorithms 2 and 3. From now on, we will refer to Algorithm 2 as RedCoeff.

Algorithm 2 Coefficients reduction for PMNS (RedCoeff)

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

Algorithm 3 Modular multiplication for PMNS

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

Remark 1. Similarly to the classical Montgomery method, Algorithm 3 induces a ϕ^{-1} factor on the output. In order to ensure the consistency of modular multiplication, unlike what is specified in the third item of Definition 1, an element $a \in \mathbb{Z}/p\mathbb{Z}$ is now represented by a polynomial $A \in \mathcal{B}$ such that: $A(\gamma) \equiv a\phi \pmod p$ (see Section 6.1).

Remark 2. There is no reason for the polynomials M and M' to be sparse. Thus, the corresponding matrices $\mathcal{M}$ and

$\mathcal{M}'$ have no particular shape, regardless the shape of the polynomial E (see Equations 2, 3). Thus, the efficiency of the internal reduction does not depend on E. However, the shape of E has an impact on the external reduction efficiency, (see line 2, Algorithm 3). The effect of E is quantified by a new parameter w we introduce in the next section.

4.3 A new parameter w

In this section, we introduce a parameter w which represents the memory overhead introduced by the polynomial E when an external reduction is done. By overhead, we mean that the coefficients of $AB \bmod E$ are bounded by $w\|A\|_\infty\|B\|_\infty$, for some w depending on E. In the next proposition, we give for any polynomial E a tight bound on w, in order to define optimal bounds on parameters ρ and ϕ of the PMNS.

Let us define the $(n-1)\times n$ matrix $\mathcal{E}'$ such that $\mathcal{E'}_{ij} = |\mathcal{E}_{ij}|$.

Proposition 2. Let $A, B \in \mathbb{Z}_{n-1}[X]$ be two polynomials. Let R be a polynomial, such that $R = AB \bmod E$, then

$$\|R\|_\infty \leqslant w\|A\|_\infty\|B\|_\infty\,,$$

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

Proof. Let $k = \|A\|_\infty\|B\|_\infty$ and C be a polynomial such that $C = AB$. From Proposition 1, we know that $R = (c_0,\ldots,c_{n-1}) + (c_n,\ldots,c_{2n-2})\mathcal{E}$. Thus, for $i = 0,\ldots,n-1$

$$r_i = c_i + \sum_{j=0}^{n-2}\mathcal{E}_{ji}c_{n+j}.$$

Consequently,

$$\begin{aligned}|r_i| &= |c_i + \sum_{j=0}^{n-2}\mathcal{E}_{ji}c_{n+j}|\\ &\leqslant |c_i| + \sum_{j=0}^{n-2}|\mathcal{E}_{ji}c_{n+j}| \leqslant |c_i| + \sum_{j=0}^{n-2}|\mathcal{E}_{ji}||c_{n+j}|\,.\end{aligned}$$

Since

- $|c_i| \leqslant (i+1)k$, for $i = 0,\ldots,n-1$,
- $|c_{n+j}| \leqslant (n-(j+1))k$, for $j = 0,\ldots,n-2$,

one finally has:

$$\begin{aligned}|r_i| &\leqslant k[(i+1) + \sum_{j=0}^{n-2}(|\mathcal{E}_{ji}|(n-j-1)]\\ &= k[(i+1) + \sum_{j=0}^{n-2}(\mathcal{E}'_{ji}(n-j-1)]\\ &= k[(i+1) + (n-1,n-2,\ldots,1)\begin{pmatrix}\mathcal{E}'_{0i}\\ \mathcal{E}'_{1i}\\ \vdots\\ \mathcal{E}'_{(n-2)i}\end{pmatrix}]\end{aligned}$$

Thus the result.

Remark 3. If $E(X) = X^n - \lambda$,

$$\mathcal{E} = \begin{pmatrix}\lambda & 0 & \ldots & \ldots & 0\\ 0 & \lambda & & & \vdots\\ \vdots & & \ddots & & \vdots\\ 0 & \ldots & \ldots & \lambda & 0\end{pmatrix}.$$

It leads to $w = 1 + (n-1)|\lambda|$ which is a slight improvement of the bound proposed in [7] which was equal to $n|\lambda|$.

5 Existence of PMNS and bounds for consistency

In this section, we first give a sufficient condition for a tuple (p, n, γ, ρ, E) to be a PMNS. This condition will then allow us to define optimal bounds on some PMNS parameters for the consistency of operations.

5.1 Existence of PMNS

Given parameters p, n, E and γ, we need to set the value of ρ to ensure convenient PMNS buildings. In [8] (Theorem 2), Bajard et al. give an answer when E is an *irreducible polynomial* such that $E(X) = X^n + \beta X + \lambda$, with $\beta, \lambda \in \mathbb{Z}$. In this section, we consider the general case where $E \in \mathbb{Z}_n[X]$ is any monic polynomial.

For any vector $S \in \mathbb{R}^n$, the classical Babaï round-off approach [6] can be used to find a vector T in a lattice $\mathcal{L}$ such that $\|S - T\|_\infty \leqslant \frac{1}{2}\|B\|_1$, where B is a basis of $\mathcal{L}$. A proof of this result is given in [9, Theorem 4.2] by considering the lattice $\mathcal{L}$ of the polynomials of degree at most $n-1$, for which γ is a root modulo p. This proof remains correct for any lattice $\mathcal{L}$ and any base B of the latter. From this remark and [9, Propostion 4.1], the following result comes:

Proposition 3. Let $p \geqslant 3$ and $n \geqslant 1$ be two integers. Let $E \in \mathbb{Z}_n[X]$ be a monic polynomial and $\gamma \in (\mathbb{Z}/p\mathbb{Z})\backslash\{\bar{0}\}$, such that $E(\gamma) \equiv 0 \pmod p$. If there exists a polynomial $M \in \mathbb{Z}_{n-1}[X]$ such that $\gcd(E, M) = 1$ in $\mathbb{Q}[X]$ and $M(\gamma) \equiv 0 \pmod p$, then (p, n, γ, ρ, E) defines a PMNS as soon as:

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

where $\mathcal{M}$ is the internal reduction matrix (Equation 2).

Proof. Let us first recall that :

$$\|\mathcal{M}\|_1 = \max_{j=1\ldots n}\sum_{i=1}^{n}|\mathcal{M}_{ij}|\,.$$

From [9, Propostion 4.1], the matrix $\mathcal{M}$ is a base of $\mathcal{L} = \{ZM \bmod E, \text{ with } Z \in \mathbb{Z}_{n-1}[X]\}$. Let $a \in \mathbb{Z}/p\mathbb{Z}$ and $v = (a, 0, \ldots, 0)$. Using Babaï round-off approach, one can find a vector t in $\mathcal{L}$ such that $\|v - t\|_\infty \leqslant \frac{1}{2}\|\mathcal{M}\|_1$. Let $Y = V - T$, where V, T, Y are the polynomials corresponding to vectors v, t, y. We have $Y(\gamma) = V(\gamma) - T(\gamma)$. As $t \in \mathcal{L}$, $T(\gamma) \equiv 0 \pmod p$, because $E(\gamma) \equiv M(\gamma) \equiv 0 \pmod p$. So, $Y(\gamma) \equiv V(\gamma) \pmod p = a$. To sum up, for any $a \in \mathbb{Z}/p\mathbb{Z}$, there exists a polynomial $Y \in \mathbb{Z}_{n-1}[X]$ such that $a = Y(\gamma) \pmod p$ and $\|Y\|_\infty \leqslant \frac{1}{2}\|\mathcal{M}\|_1$. Hence, as soon as $\rho > \frac{1}{2}\|\mathcal{M}\|_1$, the tuple $(p,\ n,\ \gamma,\ \rho,\ E)$ defines a PMNS.

5.2 Bounds on parameters

As already mentioned, an internal reduction might be required after a polynomial addition to ensure that the result is still in the PMNS. However, this is highly inefficient, since the internal reduction is a lot more expensive than the simple polynomial addition. Therefore, our goal is, as much as possible, to avoid internal reduction after additions. Let δ be the maximum number of consecutive additions of elements in $\mathcal{B}$ to compute before a modular multiplication. Taking into account δ, bounds on ρ and ϕ are given in [15], to ensure operation consistency in the PMNS, when $E(X) = X^n - \lambda$. In this section, based on Proposition 2, we improve these bounds, for any monic polynomial $E \in \mathbb{Z}_n[X]$.

Proposition 4. Let $V \in \mathbb{Z}_{n-1}[X]$ be a polynomial. If ρ, ϕ and V are such that:

$$\begin{gathered}\|V\|_\infty < w(\delta+1)^2\rho^2,\\ \rho \geqslant 2\|\mathcal{M}\|_1,\\ \phi \geqslant 2w\rho(\delta+1)^2,\end{gathered}$$

then, the output S of Algorithm 2 (with V as input) is such that $\|S\|_\infty < \rho$ (i.e., $S \in \mathcal{B}$).

Proof. Algorithm 2 outputs S is such that: $\|S\|_\infty \leqslant \frac{1}{\phi}(\|V\|_\infty + \|T\|_\infty)$. Since $\|V\|_\infty < w(\delta+1)^2\rho^2$ and $\|Q\|_\infty < \phi$,

$$\begin{aligned}\|S\|_\infty &< \frac{1}{\phi}(w(\delta+1)^2\rho^2 + \phi\|\mathcal{M}\|_1)\\ &= \frac{1}{\phi}w(\delta+1)^2\rho^2 + \|\mathcal{M}\|_1\\ &\leqslant \rho/2 + \|\mathcal{M}\|_1, \text{ since } \phi \geqslant 2w\rho(\delta+1)^2\\ &\leqslant \rho, \text{ since } \rho \geqslant 2\|\mathcal{M}\|_1\,.\end{aligned}$$

Corollary 1. Let δ be the maximum number of consecutive additions of elements in $\mathcal{B}$ to compute before a modular multiplication. Let U and W be the results of such consecutive additions. If ρ and ϕ are such that:

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

then, with U and W as inputs, Algorithm 3 outputs a polynomial S such that $\|S\|_\infty < \rho$ (i.e., $S \in \mathcal{B}$).

Proof. With U and W as inputs, the polynomial V (line 2, Algorithm 3) is such that:

$$\begin{aligned}\|V\|_\infty &\leqslant w(\delta+1)^2(\rho-1)^2 \text{ (see Proposition 2)}\\ &< w(\delta+1)^2\rho^2\,.\end{aligned}$$

So, Proposition 4 concludes.

Remark 4. When $E(X) = X^n - \lambda$, the bounds on ρ and ϕ given in [15] are:

$$\rho \geqslant 2n|\lambda|\|M\|_\infty \text{ and } \phi \geqslant 2n|\lambda|\rho(\delta+1)^2\,.$$

Since $w = 1 + (n-1)|\lambda| < n|\lambda|$ and $\|\mathcal{M}\|_1 < n|\lambda|\|M\|_\infty$, Corollary 1 improves these bounds, except for $\lambda = \pm 1$ where $w = n$. In Section 7.4.2, we explain why $\lambda = \pm 1$ is not a good choice to design a PMNS.

In [14], these bounds have been generalised to any monic polynomial $E \in \mathbb{Z}_n[X]$ (when $\delta = 0$), and are :

$$\rho \geqslant 2n\|\mathbf{S}\|_1\|M\|_\infty \text{ and } \phi \geqslant 2n\|\mathbf{S}\|_1\rho\,,$$

where S is the $(2n-1) \times n$ matrix whose rows contain the coefficients of $X^i \bmod E$, for $0 \leq i \leq 2\,n-2$.

From the definition of $\mathcal{E}$ (Equation 1), it is obvious that $\|\mathbf{S}\|_1 = 1 + \|\mathcal{E}\|_1 = 1 + \|\mathcal{E}'\|_1$. As a consequence :

$$\begin{aligned}w &= \|(1,2,\ldots,n) + (n-1,n-2,\ldots,1)\mathcal{E}'\|_\infty\\ &\leqslant n + \|(n-1,n-2,\ldots,1)\mathcal{E}'\|_\infty < n + n\|\mathcal{E}'\|_1\,.\end{aligned}$$

Hence, $w < n\|\mathbf{S}\|_1$. Also, $\|\mathcal{M}\|_1 \leqslant w\|M\|_\infty$ (see Proposition 8), as a consequence $\|\mathcal{M}\|_1 < n\|\mathbf{S}\|_1\|M\|_\infty$.

Thus, Proposition 4 and Corollary 1 improve the bounds proposed in [14].

Finally, Coladon et al. [13] improve the bound on ρ, when $E(X) = X^n - \lambda$ and $\delta = 0$. They propose to take $\rho \geqslant 2\alpha - 1$, where $\alpha = |m_0| + |\lambda|(|m_1| + |m_2| + \cdots + |m_{n-1}|)$. One can see that $\alpha = \|\mathcal{M}\|_1$.

6 Algorithms for other operations in the PMNS

First, let us remind that addition and subtraction in the PMNS are classical polynomial addition and subtraction respectively.

6.1 Conversion algorithms

As explained in Remark 1, an element $a \in \mathbb{Z}/p\mathbb{Z}$ must be represented by a polynomial $A \in \mathcal{B}$ such that $A(\gamma) \equiv a\phi \pmod p$. For efficiency reason, we aim to design a conversion algorithm which only needs one internal reduction. Hence, we need to convert $a\phi$ in a polynomial U such that $\|U\|_\infty < w(\delta+1)\rho^2$ (from Proposition 4). Let us consider the polynomials $P_i \in \mathcal{B}$ such that $P_i(\gamma) \equiv (\rho^i\phi^2) \pmod p$ and let $t = (t_{n-1}, ..., t_0)_\rho$ be the radix-ρ decomposition of a. The polynomial $U(X) = \sum_{i=0}^{n-1} t_i P_i(X)$ satisfies $\|U\|_\infty < n\rho^2$. Since $n \leqslant w$, $\|U\|_\infty \leqslant w(\delta+1)^2\rho^2$. Algorithm 4 performs the conversion into PMNS. The polynomials P_i are computed using Algorithm 5.

Algorithm 4 Conversion from classical representation to PMNS

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

Algorithm 5 Exact conversion from binary to PMNS

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

The radix-ρ decomposition (in Algorithm 4) is always possible and efficient. Indeed, Proposition 3 shows that

as soon as $\rho > \frac{1}{2}\|\mathcal{M}\|_1$, one can build a PMNS. This implies that $(2\|\mathcal{M}\|_1 - 1)^n > p$, given that the coefficients of the polynomials can be negative. From Proposition 4, the parameter ρ is taken such that $\rho \geqslant 2\|\mathcal{M}\|_1$. Thus, $\rho^n > p$. Hence, this radix-ρ decomposition with n symbols is always possible. In Section 7.1, we propose to take $\rho = 2^{\lceil \log_2(2\|\mathcal{M}\|_1)\rceil}$, i.e. a power of two, which ensures an efficient decomposition.

The conversion of A from PMNS to binary representation is the operation: $a = A(\gamma)\phi^{-1} \pmod p$. This can efficiently be done using Algorithm 6. The elements g^i are precomputed.

Algorithm 6 Conversion from PMNS to classical representation

Require: $A \in \mathbb{Z}_n[X]$, $\mathcal{B} = (p, n, \gamma, \rho, E)$ and $g_i = \phi^{-1}\gamma^i \pmod p$, for $i = 1, \ldots, n-1$
Ensure: $a = A(\gamma)\phi^{-1} \pmod p$
1: $a \leftarrow 0$
2: **for** $i = 0 \ldots n-1$ **do**
3: $\quad a \leftarrow a + a_i g_i$
4: **end for**
5: $a \leftarrow a \pmod p$
6: return a

6.2 Exact coefficient reduction

Let us consider a k-bit architecture. For software implementation, the internal reduction is optimised when $\phi = 2^k$. From Corollary 1, we have $\phi \geqslant 4w(\delta+1)^2\|\mathcal{M}\|_1$. This leads to an upper bound Δ for δ. In the context where we have to execute more than Δ additions, an internal reduction must be computed after this sequence of additions. Since RedCoeff induces a factor ϕ^{-1} on the output, we cannot use it as such. We need an "exact" reduction which computes a polynomial $S \in \mathbb{Z}_{n-1}[X]$ from a polynomial $V \in \mathbb{Z}_{n-1}[X]$ such that $S(\gamma) \equiv V(\gamma) \pmod p$. Algorithm 7 performs this exact coefficient reduction. It requires the polynomial P_0 already computed for conversion (see Algorithm 4). Notice, that this reduction is more than twice as slow as RedCoeff.

Algorithm 7 ExactRedCoeff

Require: $V \in \mathbb{Z}_{n-1}[X]$, $P_0 \equiv (\phi^2)_{\mathcal{B}}$, $\mathcal{B} = (p, n, \gamma, \rho, E)$, and the matrix $\mathcal{E}$.
Ensure: $S \in \mathbb{Z}_{n-1}[X]$ such that $S(\gamma) \equiv V(\gamma) \pmod p$
1: $T \leftarrow \text{RedCoeff}(V)$
2: $C \leftarrow T \times P_0$
3: $U = (c_0, \ldots, c_{n-1}) + (c_n, \ldots, c_{2n-2})\mathcal{E}$
4: $S \leftarrow \text{RedCoeff}(U)$
5: return S

7 Parameters generation

The parameters that define a PMNS for internal reduction using the Montgomery-like method are:

- p: a prime integer, $p \geqslant 3$.
- n: the number of coefficients of the elements in PMNS, $n \geqslant 2$.
- E: the external reduction polynomial.
- γ: a root (modulo p) of E.
- M: the internal reduction polynomial.
- ρ: the upper-bound on the infinity norm of the elements of $\mathcal{B}$.
- δ: the desired maximum number of consecutive additions before a modular multiplication.
- ϕ: the integer used in RedCoeff, Algorithm 2.
- M': a polynomial such that $M' = -M^{-1} \bmod (E, \phi)$.
- $\mathcal{E}$: the external reduction matrix.
- $\mathcal{M}$ and $\mathcal{M}'$: the internal reduction matrices.

Some of these parameters have to be chosen while the others are computed. In the following section, we provide the parameter generation process.

7.1 Parameter generation process

The parameter δ and the prime p are chosen with regard to the target application. The next step is to choose the parameter n with regard to the target architecture. Let us consider that we have a k-bit processor architecture. Then n must be chosen such that $nk \geqslant \lceil log_2(p) \rceil$ in order to ensure that each coefficient of the PMNS element can fit in one machine word.

After p and n are chosen, one chooses a monic polynomial $E \in \mathbb{Z}_n[X]$, with $\|E\|_\infty$ small, having a root $\gamma \in \mathbb{Z}/p\mathbb{Z}$ (modulo p). The parameters ϕ and ρ depend on w which in turn depends on E (and this explains the constraint "$\|E\|_\infty$ small"). Therefore, the latter must be chosen wisely. In Section 7.4, we discuss about interesting choices of E. With E, one can compute the external reduction matrix $\mathcal{E}$.

For efficiency sake, ϕ must be a power of two. The choice must be done while ensuring the existence of the polynomial $M' = -M^{-1} \bmod (E, \phi)$. In Section 7.2, we give a necessary and sufficient condition for such M' to exist. In Section 7.3, we explain how to generate the polynomial M which fulfills this requirement; see Algorithm 8 for the generation of parameters M and $\mathcal{M}$.

Using Corollary 1, ρ and ϕ are computed as follows:

$$\rho = 2^{\lceil \log_2(2\|\mathcal{M}\|_1)\rceil} \text{ and } \phi = 2^{\lceil \log_2(2w\rho(\delta+1)^2)\rceil}.$$

The polynomial M' is computed as $M' = -M^{-1} \bmod (E, \phi)$. Then, one computes the internal reduction matrix $\mathcal{M}'$. For forward and backward conversions to PMNS, we precompute the representatives $P_i(X)$ of $(\rho^i\phi^2)$ in $\mathcal{B}$ and the elements $g_i = \phi^{-1}\gamma^i \pmod p$, as explained in Section 6.1. We remind that P_0 is also required for ExactRedCoeff (Algorithm 7).

The generation strategy for efficient software implementation remains the same as described in [15, Section 5.1].

7.2 Existence of the polynomial M'

In [15, Proposition 7], the authors have established a sufficient condition of existence for M', given any polynomials $E, M \in \mathbb{Z}[X]$ and an integer $\phi \geqslant 2$. In [13], Coladon et al. show that this condition is also necessary, when $E(X) = X^n - \lambda$ and ϕ a power of two. In this section, we extend the result of [13] to any monic polynomial E and

integer $\phi \geqslant 2$. Moreover, we give a necessary and sufficient condition of existence for M', when $\phi = 2^k$ with $k \geqslant 1$, given any monic polynomial $E \in \mathbb{Z}_n[X]$.

First, let us remind some elements about the resultant $\text{Res}(A, B)$ of two polynomials A and B.

Definition 2 (Resultant)**.** [25, Def. 7.2.2, p. 227] Let $\mathcal{A}$ be a commutative ring with identity. Let A and B be two polynomials in $\mathcal{A}[X]$. The resultant $\text{Res}(A, B)$ of A and B is the determinant of their Sylvester matrix.

If $A(X) = a_0 + a_1X + \cdots + a_nX^n$ and $B(X) = b_0 + b_1X + \cdots + b_mX^m$, then their Sylvester matrix is the $(n + m) \times (n + m)$ matrix defined as follows:

$$\mathcal{S}_{A,B} = \underbrace{\begin{bmatrix} a_m & \cdots & a_0 & \\ \ddots & & \ddots & \\ & a_m & \cdots & a_0 \\ b_n & \cdots & b_0 & \\ \ddots & & \ddots & \\ & b_n & \cdots & b_0 \end{bmatrix}}_{m+n} \begin{matrix} \left.\vphantom{\begin{matrix} a \\ a \\ a \end{matrix}}\right\} n \\ \left.\vphantom{\begin{matrix} a \\ a \\ a \end{matrix}}\right\} m \end{matrix}$$

In [20, Theorem 1.18] or [30], Theorem 1 below is given. Our results will be based on this theorem.

Theorem 1. Using notations above for polynomials A and B, we have:

$$\text{Res}(A,B) = a_n^m \det(\mathcal{D}),$$

where $\mathcal{D} \in \mathbb{Z}^{n \times n}$ is such that:

$$\mathcal{D} = \begin{pmatrix} \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \vdots & \vdots & & \vdots \\ \cdots & \cdots & \cdots & \cdots \end{pmatrix} \begin{matrix} \leftarrow B \bmod A \\ \leftarrow X.B \bmod A \\ \\ \leftarrow X^{n-1}.B \bmod A \end{matrix}$$

Corollary 2. Let $E \in \mathbb{Z}_n[X]$ be a monic polynomial and $M \in \mathbb{Z}_{n-1}[X]$, then:

$$\text{Res}(E,M) = \det(\mathcal{M}),$$

where $\mathcal{M}$ is the internal reduction matrix computed with E and M (see Equation 2).

Proof. Applying Theorem 1, the polynomials E, M and matrix $\mathcal{M}$ correspond to A, B and $\mathcal{D}$ respectively. Since E is a monic polynomial, one can conclude.

Before stating the existence criterion of polynomial M', we establish the useful next property.

Property 1. Let $M \in \mathbb{Z}_{n-1}[X]$ be a polynomial and $\phi \geqslant 2$ be an integer. Let us assume that the polynomial $J = M^{-1} \bmod (E, \phi)$ exists. Then, the matrix $\mathcal{J}$ defined below is such that:

$$\mathcal{M} \times \mathcal{J} \pmod{\phi} = \mathcal{I}_n \,,$$

where $\mathcal{I}_n$ is the $(n \times n)$ identity matrix and $\mathcal{M}$ the internal reduction matrix. That is, $\mathcal{J}$ is the inverse of $\mathcal{M}$ in the set of $n \times n$ matrices defined over $\mathbb{Z}/\phi\mathbb{Z}$.

$$\mathcal{J} = \begin{pmatrix} j_0 & j_1 & \cdots & j_{n-1} \\ \cdots & \cdots & \cdots & \cdots \\ \vdots & \vdots & & \vdots \\ \cdots & \cdots & \cdots & \cdots \end{pmatrix} \begin{matrix} \leftarrow J \\ \leftarrow X.J \bmod (E, \phi) \\ \\ \leftarrow X^{n-1}.J \bmod (E, \phi) \end{matrix} \quad (6)$$

Proof. For any vector V, the product $V\mathcal{J}$ gives a vector whose coordinates are the coefficients of $V(X)J(X) \bmod (E, \phi)$. Hence, the result of $\mathcal{M} \times \mathcal{J}$ is a set of n rows, where the i^{th} row corresponds to the coefficients of $X^{i-1}M.J = X^{i-1} \bmod (E, \phi)$. Hence, for $1 \leqslant i \leqslant n$, the i^{th} row is equal to $(0, \ldots, 0, 1, 0, \ldots, 0)$ where the value 1 is at position i.

Now, we give a general existence criterion for the polynomial M', given any integer $\phi \geqslant 2$.

Proposition 5. Let $E \in \mathbb{Z}_n[X]$ be a monic polynomial and $M \in \mathbb{Z}_{n-1}[X]$. Let $\phi \geqslant 2$ be an integer. The polynomial $M' = -M^{-1} \bmod (E, \phi)$ exists if and only if $\gcd(\text{Res}(E, M), \phi) = 1$.

Proof. From Proposition 7 in [15], if $\gcd(\text{Res}(E, M), \phi) = 1$ then the polynomial $J = M^{-1} \bmod (E, \phi)$ (and so M') exists. Hence, it remains to show that if J exists then $\gcd(\text{Res}(E, M), \phi) = 1$.
Let us assume that $J = M^{-1} \bmod (E, \phi)$ exists. Since the matrix $\mathcal{J}$ (Equation 6) is defined by J, the inverse (in the set of $(n \times n)$ matrices defined over $\mathbb{Z}/\phi\mathbb{Z}$) of $\mathcal{M}$ exists, i.e. $\det(\mathcal{M}) \in (\mathbb{Z}/\phi\mathbb{Z})^*$, according to Property 1. With Corollary 2, we deduce that $\gcd(\text{Res}(E, M), \phi) = 1$.

As a consequence of the previous proposition, we now state our main existence criterion for the polynomial M', when ϕ is a power of two.

Corollary 3 (**Existence criterion**). Let $E \in \mathbb{Z}_n[X]$ be a monic polynomial and $M \in \mathbb{Z}_{n-1}[X]$. Let $\phi = 2^k$, for $k \geqslant 1$ an integer. The polynomial $M' = -M^{-1} \bmod (E, \phi)$ exists if and only if $\det(\widetilde{\mathcal{M}}) = 1$, where $\widetilde{\mathcal{M}}$ is the $(n \times n)$ binary matrix such that $\widetilde{\mathcal{M}}_{ij} = \mathcal{M}_{ij} \pmod 2$.

Proof. Since ϕ is a power of two, $\gcd(\text{Res}(E, M), \phi) = 1$ iff $\text{Res}(E,M)$ is odd, which means that $\det(\mathcal{M})$ is odd. This is equivalent to have $\det(\widetilde{\mathcal{M}}) = 1$.

Remark 5. Since $\widetilde{\mathcal{M}}$ is a binary matrix, its determinant can be computed very efficiently using only logical operators.

7.3 Generation of the polynomial M

Let $\mathcal{B} = (p, n, \gamma, \rho, E)$ be a PMNS. Let us consider the set of polynomials $\mathcal{L}_\mathcal{B}$ defined as follows:

$$\mathcal{L}_\mathcal{B} = \{A \in \mathbb{Z}_{n-1}[X] \mid A(\gamma) \equiv 0 \pmod p\}. \quad (7)$$

In [8] (Lemma 1), the authors have shown that $\mathcal{L}_\mathcal{B}$ is a full-rank euclidean lattice of dimension n. A basis of $\mathcal{L}_\mathcal{B}$ is:

$$\mathbf{B} = \begin{pmatrix} p & 0 & 0 & \ldots & 0 & 0 \\ t_1 & 1 & 0 & \ldots & 0 & 0 \\ t_2 & 0 & 1 & \ldots & 0 & 0 \\ \vdots & & & \ddots & & \vdots \\ t_{n-2} & 0 & 0 & \ldots & 1 & 0 \\ t_{n-1} & 0 & 0 & \ldots & 0 & 1 \end{pmatrix} \quad (8)$$

where $t_i = (-\gamma)^i \bmod p$.
As a consequence, any basis $\mathbf{C}$ of $\mathcal{L}_\mathcal{B}$ is such that $\mathbf{C} = \mathbf{U} \times \mathbf{B}$ where $\mathbf{U}$ is a $n \times n$ unimodular matrix.

Corollary 3 gives a sufficient (and necessary) condition on the existence of the the polynomial M' when ϕ is a power of two. In this section, we present a generation process which ensures to always find at least one suitable polynomial M, given any basis of the lattice $\mathcal{L}_\mathcal{B}$.

Definition 3. Let $A \in \mathbb{Z}_{n-1}[X]$ be a polynomial, such that: $A(X) = a_0 + a_1X + \cdots + a_{n-1}X^{n-1}$. We denote by $\overline{A}$ the polynomial such that: $\overline{A}(X) = a'_0 + a'_1X + \cdots + a'_{n-1}X^{n-1}$, where $a'_i = a_i \pmod 2$.

The next proposition highlights a polynomial M for which the polynomial M' always exists, given any monic polynomial $E \in \mathbb{Z}_n[X]$ and ϕ a power of two.

Proposition 6. Let $E \in \mathbb{Z}_n[X]$ be a monic polynomial and $M \in \mathbb{Z}_{n-1}[X]$. If the polynomial M is such that:

$$m_0 \equiv 1 \pmod 2 \text{ and } m_i \equiv 0 \pmod 2 \text{ for } 1 \leqslant i < n,$$

i.e. $\overline{M} = 1$, then the corresponding internal reduction matrix $\mathcal{M}$ is such that: $\widetilde{\mathcal{M}} = \mathcal{I}_n$. Therefore, M' exists.

Proof. Let us assume $E(X) = X^n - S(X)$, where $S \in \mathbb{Z}_{n-1}[X]$. Let $V \in \mathbb{Z}_{n-1}[X]$, such that $v_{n-1} \equiv 0 \pmod 2$. Then,

$$\overline{(X.V(X)) \bmod E(X)} = v'_0X + v'_1X^2 + \cdots + v'_{n-2}X^{n-1}\,.$$

Indeed, $X.V(X) \bmod E(X) = v_0X + v_1X^2 + \cdots + v_{n-2}X^{n-1} + v_{n-1}S(X)$. Since $v_{n-1} \equiv 0 \pmod 2$, $\overline{v_{n-1}S(X)} = 0$.
Now, let us consider the polynomial M. To build the matrix $\mathcal{M}$, one computes $X^iM(X) \bmod E(X)$, for $0 \leqslant i < n$. Since $\overline{M} = 1$, then

$$\overline{X^iM(X) \bmod E(X)} = \overline{X^i}\,,$$

which corresponds in $\widetilde{M}$ to the row $(0, \ldots, 0, 1, 0, \ldots, 0)$, the non-zero element being on the diagonal. Hence, $\widetilde{\mathcal{M}} = \mathcal{I}_n$. Since $\det(\mathcal{I}_n) = 1$, one can conclude with Corollary 3 that $M' = -M^{-1} \bmod (E, \phi)$ exists.

According to Property 6, if $\overline{M} = 1$, then M' exists. We now evaluate the complexity of finding such a polynomial M. The next proposition answers this question.

Proposition 7. Let $\mathcal{G} = \{\mathcal{G}_0, \ldots, \mathcal{G}_{n-1}\}$ be a basis of $\mathcal{L}_\mathcal{B}$. Then, there exists a vector $\beta = (\beta_0, \ldots, \beta_{n-1}) \in \mathbb{F}_2^n$, such that the vector $M = \sum_{i=0}^{n-1} \beta_i\mathcal{G}_i$ satisfies: $m_0 \equiv 1 \pmod 2$ and $m_i \equiv 0 \pmod 2$ for $1 \leqslant i < n$; i.e. $\overline{M} = 1$.

Proof. Let us consider $V = (p, 0, \ldots, 0)$, the first row of the basis $\mathbf{B}$ (Equation 8). Since $p \geqslant 3$ and prime, we have: $\overline{V} = (1, 0, \ldots, 0)$.
Also $V \in \mathcal{L}_\mathcal{B}$, so there exists a vector $\alpha = (\alpha_0, \ldots, \alpha_{n-1}) \in \mathbb{Z}^n$ such that: $V = \sum_{i=0}^{n-1} \alpha_i\mathcal{G}_i$.
Now, let us consider the polynomial $M = \sum_{i=0}^{n-1} \beta_i\mathcal{G}_i$, where $\beta_i = \alpha_i \pmod 2$, then $\overline{V} = \overline{M}$.

Given any basis $\mathcal{G}$ of $\mathcal{L}_\mathcal{B}$, according to Proposition 7, at least one binary linear combination of $\mathcal{G}$ rows computes a polynomial M such that $\overline{M} = 1$. Thus, one needs to check at most 2^n linear combinations of $\mathcal{G}$ rows to find a suitable polynomial M. In practice (i.e. for cryptographic sizes), n is small enough to make possible the test of all these combinations. For example, with p of size 256 bits, one has $n = 5$.

Let $\theta = \max_{0 \leqslant i < n} \|\mathcal{G}_i\|_\infty$. For each binary linear combination, the computed polynomial M verifies $\|M\|_\infty \leqslant n\theta$. Thus, if $\|\mathcal{G}_i\|_\infty$ are small, then $\|M\|_\infty$ will also be small, since n is small and negligible compared to θ.

Remark 6. The purpose of Proposition 7 is to provide the guarantee of existence of at least one suitable M among the binary linear combinations of $\mathcal{G}$ rows. In practice, the input $\mathcal{G}$ should be a reduced based of $\mathcal{L}_\mathcal{B}$, such that $\|\mathcal{G}\|_1 \approx p^{\frac{1}{n}}$, in order to obtain a polynomial M such that $\|M\|_\infty \approx p^{\frac{1}{n}}$. Such a basis $\mathcal{G}$ can be computed by applying a lattice reduction algorithm (like LLL [24], BKZ[32] or HKZ[23]) on the basis $\mathbf{B}$ (Equation 8).

Proposition 7 guarantees that Algorithm 8 will always output a suitable result. On the other hand, since n is small enough in practice, in order to find the best candidate(s), it remains possible to check all the 2^n binary combinations. The best candidates are defined here as the ones that minimise the 1-norm of the corresponding internal reduction matrix $\mathcal{M}$, since this will lead to the smallest value for ρ.

Algorithm 8 Generation of parameters M and $\mathcal{M}$

Require: n, $E \in \mathbb{Z}_n[X]$ a monic polynomial having γ as a root modulo p and $\mathcal{G}$ a reduced basis computed from $\mathbf{B}$ (Equation 8).
Ensure: $M \in \mathbb{Z}_{n-1}[X]$ such that: $M(\gamma) \equiv 0 \pmod p$ and $M' = -M^{-1} \bmod (E, 2^k)$ exists, for any integer $k \geqslant 1$.

```
for i = 1 ... 2^n − 1 do
    t ← (t_0, ..., t_{n−1}) # binary decomposition of i
    M ← (t_0, ..., t_{n−1})G
    Compute M # see Equation 2, page 3
    if det(M̃) = 1 then
        return (M, M)
    end if
end for
```

Remark 7. If $\gcd(\text{Res}(E, M), \phi) = 1$, then $\text{Res}(E, M) \neq 0$. So, $\gcd(E, M) = 1$ in $\mathbb{Q}[X]$. Since the parameter ρ is such that: $\rho = 2^{\lceil \log_2(2\|\mathcal{M}\|_1) \rceil} > \frac{1}{2}\|\mathcal{M}\|_1$ (see Section 7.1), Proposition 3 guarantees that the tuple (p, n, γ, ρ, E) defines a PMNS.

7.4 Choice of the polynomial E

As mentioned in Remark 2, the parameter w computed from E has an effect on ρ. In this section, we present a new set of polynomials E for small memory cost. Let us first show the following result.

Proposition 8. Let $E \in \mathbb{Z}_n[X]$ be a monic polynomial and $M \in \mathbb{Z}_{n-1}[X]$. Then,

$$\|\mathcal{M}\|_1 \leqslant w\|M\|_\infty\,,$$

where w is the parameter given in Proposition 2.

Proof. We have

$$\begin{aligned} X^iM \bmod E =& (0, \ldots, 0, m_0, \ldots, m_{n-1-i}) \\ &+ (m_{n-i}, \ldots, m_{n-1}, 0, \ldots, 0)\mathcal{E}\,. \end{aligned}$$

Since each m_i is bounded by $k = \|M\|_\infty$, then

$$\|\mathcal{M}\|_1 \leqslant k\| \begin{pmatrix} 1 & 1 & \ldots & 1 \\ 0 & 1 & \ldots & 1 \\ & & & \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & \ldots & 0 \\ 1 & 0 & \ldots & 0 \\ \vdots & \ddots & & \vdots \\ 1 & \ldots & 1 & 0 \end{pmatrix} \mathcal{E}'\|_1$$

Hence the result, since

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

Under the hypothesis that $\|M\|_\infty \simeq p^{1/n}$ (from Minkowski's theorem), Proposition 8 shows that a way to lower $\|\mathcal{M}\|_1$ (and thus ρ) is to minimize the value of w. For an AMNS, $w = 1 + (n-1)|\lambda|$ (see remark 3), thus taking $|\lambda| = 2$ will give the smallest upper bound on $\|\mathcal{M}\|_1$ (next section explains why $|\lambda| > 1$). Moreover it leads to a very fast external reduction process which uses only additions and shift operations. Hence, among the AMNS polynomials, $X^n \pm 2$ are the best choices and correspond to a value $w = 2n-1$.

In the previous sections, we have seen how to generate PMNS parameters given any monic polynomial $E \in \mathbb{Z}_n[X]$, which has a root γ modulo p and also how to perform a constant time internal reduction regardless the polynomial E (see Section 4.2, Algorithm 2). The impact of the polynomial E on the efficiency of the external reduction can be deduced from Algorithm 3 line 2. Indeed, one can see that the corresponding computation uses the external reduction matrix $\mathcal{E}$ which depends on E. The more the matrix $\mathcal{E}$ is sparse, the faster the external reduction is. Moreover, to be competitive with the AMNS context, the matrix should contain only coefficients equal to ± 1 or $\pm 2^i$ so that the external reduction can be done only with additions and shift operations.

Taking into account the optimal AMNS context, the following question arises: are there other polynomials E which lead to such sparse matrices $\mathcal{E}$ and a value $w \leqslant 2n-1$?

In Section 7.4.1, we present, through examples, a set of polynomials $E \in \mathbb{Z}_n[X]$ which provide such a value w and a sparse matrix $\mathcal{E}$. Enlarge the set of polynomials has already been considered in [9], but the authors did not consider the parameter w.

7.4.1 Examples of polynomials E for fast external reduction and small memory cost

All the matrices $\mathcal{E}$ presented in this section are such that: $\mathcal{E}_{ij} \in \{-1, 0, 1\}$, each column has at most two non-zero elements and at least two columns have only one non-zero element. Thus, the operation $V \times \mathcal{E}$, where $V \in \mathbb{Z}^{n-1}$, costs at most $n-2$ additions of V elements.

Let us start by the most interesting ones, which are obtained when n is even. We provide a proof for one case only. The other cases can be proven in the same way.

Example 1. Let us assume that $n \equiv 0 \pmod 2$. If $E(X) = X^n + jX^{n/2} + 1$, with $j \in \{-1, 1\}$, then:

$$\mathcal{E} = \begin{pmatrix} -\mathcal{I}_{\frac{n}{2}} & -j\mathcal{I}_{\frac{n}{2}} \\ j\mathcal{I}_{\frac{n}{2}-1} & 0_{\frac{n}{2}-1,\frac{n}{2}+1} \end{pmatrix}$$

So, $w = (3n)/2$ (see Proposition 11 for proof, when $j = 1$).

Now, here are some other polynomials which gives a value w that is lower than the value w computed for $X^n - \lambda$.

Example 2. If $E(X) = X^n + X^{n-1} + \cdots + X + 1$ ($E(X)$ is the all-one polynomial), then:

$$\mathcal{E} = \begin{pmatrix} -1 & -1 & \dots & -1 \\ & \mathcal{I}_{n-2} & & 0_{n-2,2} \end{pmatrix}$$

So, $w = 2n-1$. Indeed, each column contains exactly two ones except the last two. Hence, $(n-1, n-2, \dots, 2, 1)\mathcal{E}' = (2n-3, 2n-4, \dots, n-1, n-1)$, so $(1, 2, \dots, n) + (n-1, n-2, \dots, 2, 1)\mathcal{E}' = (2n-2, 2n-2, 2n-2, \dots, 2n-2, 2n-1)$.

Example 3. If $E(X) = X^n + iX + j$, with $i, j \in \{-1, 1\}$, then:

$$\mathcal{E} = \begin{pmatrix} -j & -i & 0 & \dots & 0 \\ 0 & -j & -i & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & -j & -i \end{pmatrix}$$

So, $w = 2n-1$. Indeed, $(1, 2, \dots, n) + (n-1, n-2, \dots, 2, 1)\mathcal{E}' = (n, 2n-1, 2n-2, 2n-3, \dots, n+1)$.

Example 4. Let us assume that $n \equiv 0 \pmod 2$. If $E(X) = X^n + X^{n-2} + X^{n-4} + \cdots + X^2 + 1$, then:

$$\mathcal{E} = \begin{pmatrix} -1 & 0 & -1 & 0 & \dots & -1 & 0 \\ 0 & -1 & 0 & -1 & \dots & 0 & -1 \\ & \mathcal{I}_{n-3} & & & & 0_{n-3,3} & \end{pmatrix}$$

So, $w = 2n-2$. Indeed, let $U = (1, 2, \dots, n) + (n-1, n-2, \dots, 2, 1)\mathcal{E}'$. For $0 \leqslant i \leqslant n-4$, we have: $u_i = 2n-3-(i \bmod 2)$; and for $n-3 \leqslant i < n$, we have: $u_i = n+i-(i \bmod 2)$. Thus, the maximum, which is $2n-2$, is obtained for $i = n-2$ and $i = n-1$, since n is even.

The first two rows of $\mathcal{E}$ come from

$$X^n \equiv -(X^{n-2} + X^{n-4} + \cdots + X^2 + 1) \bmod E\,.$$

For the third row, we have $X^{n+2} \equiv -(X^n + X^{n-2} + \cdots + X^4 + X^2) \bmod E \equiv 1 \bmod E$. Hence, the rows 3 to $n-2$ of $\mathcal{E}$ contains only one element equal to 1 which is shifted from one row to the next row.

For example 2, we use the fact that a part of w is computed from a vector-matrix product, with the matrix $\mathcal{E}'$. Thus, a way to minimize w is to consider a matrix where there are few "small" non-zero elements in each column. As an example, when $E(X) = X^n - \lambda$, the matrix $\mathcal{E}$ has only one non-zero element per column (see Remark 3). In order to minimize w, we propose a heuristic allowing to obtain some matrices $\mathcal{E}'$ which contain at most 2 ones in each column. Let $E(X) = X^n - S(X)$, each row of $\mathcal{E}$ is a right shift of the preceding row plus the element coming out of this right shift multiplied by the coefficients of S. A first way to reach our goal is to try to find a polynomial $S(X) = s_{n-1}X^{n-1} + \cdots + s_1X + s_0$, with $s_0 = \pm 1$, such that $XS(X) = X^n - S(X) + s_0$. Indeed, in this case, $XS(X) \equiv s_0 \pmod{E(X)}$; hence, from lines 2 to $n-1$, the rows of $\mathcal{E}$ will contain only one non-zero element.

We have:

$$XS(X) = X^n - S(X) + s_0 \iff S(X) = \frac{X^n + s_0}{X+1}\,.$$

Let us consider $s_0 = 1$ and n odd, then $X+1$ is a divisor of $X^n + 1$. Hence, we can choose $S(X) = X^{n-1} - X^{n-2} + X^{n-3} - \cdots - X + 1$. Thus $E(X) = X^n - X^{n-1} + X^{n-2} - X^{n-3} + \cdots + X - 1$, with n odd and $w = 2n-1$.

Let us now consider $s_0 = -1$ and n even, then $X+1$ is a divisor of $X^n - 1$. Hence, we can choose:

$$S(X) = \frac{X^n - 1}{X+1} = X^{n-1} - X^{n-2} + X^{n-3} - \cdots - X^2 + X - 1\,.$$

Finally,

$$E(X) = X^n - X^{n-1} + X^{n-2} - X^{n-3} + \cdots + X^2 - X + 1\,,$$

with n even and $w = 2n - 1$.

Another strategy is to find a polynomial S such that: $XS(X) = -X^n + S(X) - s_0$, in order to have: $XS(X) \equiv -s_0 \pmod{E(X)}$. This gives:

$$S(X) = -\frac{X^n + s_0}{X - 1}.$$

Let us consider $s_0 = -1$. Since $X - 1$ divides $X^n - 1$, for any n, we can choose:

$$S(X) = -\frac{X^n - 1}{X - 1} = -X^{n-1} - X^{n-2} - \cdots - X - 1.$$

It corresponds to Example 2.
Let $s_0 = 1$, then $X - 1$ does not divide $X^n + 1$, since 1 is not a root of $X^n + 1$.

Example 3 is obtained by considering that the reduction modulo E adds some new elements in the matrix, except if each shift does not bring in the last column a non zero element. Hence, a natural way to prevent this is to start with the first line having only two non zero elements in the first two positions.
This naturally leads us to consider the more general set of trinomials $X^n \pm X^j \pm 1$ (see Example 1).

Proposition 9. Let $j < \frac{n}{2}$ and $E(X) = X^n + X^j + 1$, then $w \geqslant 2n - 1$.

Proof. Each row of $\mathcal{E}'$ contains the absolute value of the coefficients of $X^i.(-X^j - 1) \pmod{E(X)}$ for $i = 0 \ldots n-2$, from constant term to leading term. The first line contains the absolute value of the coefficients of $-1 - X^j$. Hence, there is a one in column $j + 1$. Next the $(j + 1)^{\text{th}}$ row contains the absolute value of $X^j.(-X^j - 1) = -X^j - X^{2j}$, no reduction is done , since $j < \frac{n}{2}$. Thus, the $(j + 1)^{\text{th}}$ column contain at least two ones (one at row 1, and another one at row $j + 1$). From the definition of w, we deduce that $w \geqslant j + 1 + n - 1 + n - (j + 1) = 2n - 1$.

Proposition 10. Let $j > \frac{n}{2}$ and $E(X) = X^n + X^j + 1$, then $w > \frac{3n}{2}$.

Proof. The first line of $\mathcal{E}'$ contains the absolute value of the coefficients of $-1 - X^j$. Suppose that $j < n - 1$, since $j > \frac{n}{2}$, then $n - j < n - 1$ and $2n - 2j < n - 1$. Hence, the rows $n - j + 1$ and $2n - 2j + 1$ are well defined and respectively contains the absolute values of the coefficients of $X^{n-j}(-1 - X^j) \pmod{E(X)}$ and $X^{2n-2j}(-1 - X^j) \pmod{E(X)}$. These two lines are computed from the polynomials $1 - X^{n-j} + X^j$ and $-1 + X^{n-j} - X^{2n-2j} - X^j$. Thus, the first column of $\mathcal{E}'$ contains at least three ones (row 1, row $n-j+1$ and row $2n-2j+1$). From the definition of w, one has $w \geqslant 1+n-1+n-1-(n-j)+n-1-2(n-j)$, thus, $w \geqslant 3j - 2$. Since $j = \frac{n}{2} + k$, with $k \geqslant 1$, we conclude that $w > \frac{3n}{2}$.
We do not detail the particular case $j = n - 1$. In this case, each multiplication by X provides the monomial $\pm X^n$, which in turn is reduced (mod E) in the constant term ± 1. Hence, the first column of $\mathcal{E}'$ is the all-1 column which implies that $w \geqslant 1 + \frac{n(n-1)}{2}$.

Proposition 11. Let's assume that $n \equiv 0 \pmod 2$. Let $j = \frac{n}{2}$ and $E(X) = X^n + X^j + 1$, then $w = \frac{3n}{2}$.

Proof. We have:

- for $i \in [0, \frac{n}{2} - 1]$, $X^{n+i} = -X^{\frac{n}{2}+i} - X^i \bmod E(X)$,
- for $i = \frac{n}{2}$, $X^{n+\frac{n}{2}} = -X^n - X^{\frac{n}{2}} = 1 \bmod E(X)$,
- for $i \in [\frac{n}{2} + 1, n - 2]$, $X^{n+i} = X^{i-\frac{n}{2}}$.

Hence, the matrix $\mathcal{E}'$ is as follows:

$$\begin{pmatrix} I_{\frac{n}{2}} & I_{\frac{n}{2}} \\ I_{\frac{n}{2}-1} & 0_{\frac{n}{2}-1,\frac{n}{2}+1} \end{pmatrix},$$

where I_t is the $t \times t$ identity matrix, and $0_{\frac{n}{2}-1,\frac{n}{2}+1}$ is the $(\frac{n}{2} - 1) \times (\frac{n}{2} + 1)$ zero matrix.
Let $\tilde{w}$ be the vector $(1, 2, \ldots, n) + (n - 1, n - 2, \ldots, 2, 1)\mathcal{E}'$. Then $\tilde{w}_i = i + \sum_{j=1}^{n-1}(n - j)\mathcal{E}'_{ij}$ for $i \in [1, n - 1]$. Now:

- $\tilde{w}_i = i + (n - i) + (n - (i + \frac{n}{2})) = \frac{3n}{2} - i$ for $i \in [1, \frac{n}{2}]$, hence $w_i < \frac{3n}{2}$,
- $\tilde{w}_{\frac{n}{2}} = \frac{n}{2} + (n - \frac{n}{2}) = n$,
- $\tilde{w}_i = i + n - (i - \frac{n}{2}) = \frac{3n}{2}$ for $i \in [\frac{n}{2}, n - 1]$.

From the preceding propositions, it appears that one of the trinomials computing the best value of w is $X^n + X^{\frac{n}{2}} + 1$, with n even.

Table 1 summarizes the polynomials presented in this section, with the requirement on the parity of n and the corresponding value of w. The symbol $-$ in this table means that the parity of n doesn't matter.

$E(X)$	n	w
$X^n \pm X^{\frac{n}{2}} + 1$	even	$3n/2$
$X^n + X^{n-2} + X^{n-4} + \cdots + X^2 + 1$	even	$2n - 2$
$X^n - X^{n-1} + X^{n-2} - X^{n-3} + \cdots - X + 1$	even	$2n - 1$
$X^n - X^{n-1} + X^{n-2} - X^{n-3} + \cdots + X - 1$	odd	$2n - 1$
$X^n \pm X \pm 1$	$-$	$2n - 1$
$X^n + X^{n-1} + \cdots + X + 1$	$-$	$2n - 1$

TABLE 1: Example of polynomials E, for efficient external reduction and small memory cost.

7.4.2 The case $E(X)$ non-irreducible

Let us remind that the memory required to represent PMNS elements depends on n and ρ. Given n, our goal is to minimize ρ. The generation process presented above does not require the polynomial E to be irreducible. In practice, however, with non-irreducible polynomials E, it leads to PMNS having a too large parameter ρ. We now provide an explanation of this.

As seen with parameters w and $\mathcal{E}$, $\|E\|_\infty$ has to be very small, for instance less than or equal to 8. Consequently $\|E\|_2$ is also very small, bounded by $8\sqrt{n}$, if $\|E\|_\infty \leqslant 8$. This leads to an efficient external reduction with small memory overhead. If E is non-irreducible in $\mathbb{Z}_n[X]$, then there exist two monic polynomials $G, H \in \mathbb{Z}[X]$ such that $E(X) = G(X)H(X)$, with $\deg(G), \deg(H) < n$ and $\|G\|_2$, $\|H\|_2$ both very small. In the generation process, the lattice reduction can be done with algorithms such as LLL, BKZ or HKZ, in order to get the reduced basis. When E is non-irreducible, the first row b_0 of this reduced basis $(b_0, b_1, \ldots, b_{n-1})$, computed from $\mathbf{B}$ (Equation 8), is either G or H. The volume of this lattice is p, since $\det(\mathbf{B}) = p$.

Hence, $\det(b_0, b_1, \ldots, b_{n-1}) = p$. From Hadamard inequality, we know that:

$$\prod_{i=0}^{n-1} \|b_i\|_\infty \geqslant \frac{p}{n^{n/2}} .$$

Since $\deg(G), \deg(H) < n$ and $\|b_0\|_\infty \approx \|E\|_2$ which is very small, the vectors b_i for $1 \leqslant i < n$ are such that: $\|b_i\|_\infty \approx p^{\frac{1}{k}}$ where $k < n$, because the parameter n is negligible compared to p. Thus, since the polynomial M is a binary linear combination of the vectors b_i, $\|M\|_\infty$ has the same order of magnitude as $p^{\frac{1}{k}}$.

In practice, in order to represent PMNS elements with optimal memory cost, we set $\rho \approx p^{\frac{1}{n}}$. Thus, from Equation 2, M is such that $\|M\|_\infty \approx p^{\frac{1}{n}}$, since $\|E\|_\infty$ is very small. With p meant to be a large integer and n a small one, one has $p^{\frac{1}{k}} >> p^{\frac{1}{n}}$. Consequently, on the one hand, searching M among binary linear combinations of a reduced basis will not allow to generate a suitable PMNS, $\|M\|_\infty$ (and therefore ρ) being too large. On the other hand, enlarging the set of possible linear combinations will drastically increase the complexity of the generation process.

Remark 8. The polynomial $E(X) = X^n - 1$ falls in this case, since $X^n - 1 = (X-1)(X^{n-1} + \cdots + 1)$.
When n is even ($n = 2k$), then -1 must be a $(2k)^{\text{th}}$ residue modulo p, for $E(X) = X^n + 1$ to have a root modulo p. It implies that $p \equiv 1 \pmod 4$ [19]. Hence, if $p \not\equiv 1 \pmod 4$, no PMNS can be built with $E(X) = X^n + 1$ and n even.
If n is not a power of 2 (which includes n odd), i.e. $n = 2^s j$ with $j > 1$ odd, then $X^n + 1 = (X^{2^s} + 1)S(X)$, where:

$$S(X) = (X^{2^s})^{j-1} - (X^{2^s})^{j-2} + (X^{2^s})^{j-3} - \cdots - X + 1 .$$

Thus, the polynomial $E(X) = X^n + 1$ can be used to build a suitable PMNS only when $p \equiv 1 \pmod 4$ and n is a power of 2, which is unlikely to happen most of the time in practice.

8 Implementations and analyses

In [15, Section 6], a deep study of AMNS memory requirement and efficiency (both theoretical and practical) is done, along with some other aspects. Since, only the form of the external polynomial E differs from AMNS to PMNS, this study remains applicable to PMNS. In this section, we discuss the differences induced by the choice we suggest for E. We focus on polynomials E presented in Section 7.4.

8.1 Theoretical performances

The performances and the required memory storage of a PMNS depend mainly on the target architecture and the value of n. Let us consider a k-bit processor architecture, then the basic arithmetic computations are performed on k-bit words.

We assume algorithms inputs belong to a PMNS $\mathcal{B} = (p, n, \gamma, \rho, E)$, such that: $\rho = 2^t$ and $\phi = 2^h$, where $t, h \in \mathbb{N}$ and $1 \leqslant t < h \leqslant k$.
Since elements in $\mathcal{B}$ are polynomials, n k-bit data words are required to represent each of them. As a consequence, an element in $\mathcal{B}$ requires nk bits to be represented.

Let $\mathcal{M}$ and $\mathcal{A}$ respectively denote the multiplication and the addition of two k-bit integers. We also respectively denote $\mathcal{S}_l^i$ and $\mathcal{S}_r^i$ a left shift and a right shift of i bits. Let $x = x_1 x_2$ and $y = y_1 y_2$, where x_i and y_i are k-bit data words; the computation of $x + y$ costs $2\mathcal{A}$.
With the internal reduction matrices $\mathcal{M}$ and $\mathcal{M}'$, RedCoeff cost becomes independent of E and is:

$$2n^2\mathcal{M} + (3n^2 - n)\mathcal{A} + n\mathcal{S}_r^k .$$

Let $A, B \in \mathcal{B}$. The operation $C = A \times B$, which is also independent of E, costs:

$$n^2\mathcal{M} + (2n^2 - 4n + 2)\mathcal{A} .$$

Now, let us focus on the external reduction; i.e. the operation $R = C \bmod E$. As shown in Proposition 1, $R = (c_0, \ldots, c_{n-1}) + (c_n, \ldots, c_{2n-2})\mathcal{E}$, where $\mathcal{E}$ is the external reduction matrix corresponding to E. It involves n additions and the cost of $(c_n, \ldots, c_{2n-2})\mathcal{E}$.
All the polynomials presented in Section 7.4 compute matrices $\mathcal{E}$ such that: $\mathcal{E}_{ij} \in \{-1, 0, 1\}$, each column has at most two non-zero elements and at least two columns have only one non-zero element.
As a consequence, since each c_i is a $2k$-bit integer, the operation $(c_n, \ldots, c_{2n-2})\mathcal{E}$ costs at most $2(n-2)\mathcal{A}$.

If $E(X) = X^n - \lambda$, with $|\lambda| \neq 1$, this operation is less expensive only when $\lambda = \pm 2^i$. Indeed, in this case, it costs $(n-1)\mathcal{S}_l^i$, where shift operations are done on $2k$-bit integers. For such polynomials we have $w = 1 + |\lambda|(n-1)$. Hence, from a storage point of view, it is advised to take $|\lambda| = 2$ (i.e. $w = 2n - 1$) to minimize the impact of w on ρ.

To sum up, with the polynomials E presented in Section 7.4, we have $w \leqslant 2n - 1$, and the modular multiplication costs at most:

$$3n^2\mathcal{M} + (5n^2 - n - 2)\mathcal{A} + n\mathcal{S}_r^k .$$

On the other hand, with the polynomials $E(X) = X^n \pm 2$, we have $w = 2n - 1$ and the modular multiplication costs:

$$3n^2\mathcal{M} + (5n^2 - 3n)\mathcal{A} + (n-1)\mathcal{S}_l^2 + n\mathcal{S}_r^k .$$

In the next section, we compare our results (for modular multiplication) to GMP and OpenSSL libraries, which implement Montgomery-CIOS method [22], an improvement of the initial Montgomery modular multiplication method. Let us assume that the modulus p is of l-bit size. These libraries store an element a of $\mathbb{Z}/p\mathbb{Z}$ into an array of $j = \lceil l/k \rceil$ elements of size k, with additional data for some flags which have non significant impact on operations efficiency. Each element of this array is a coefficient of the radix 2^k decomposition of a.
The Montgomery-CIOS method combines the multiplication and reduction steps to improve both the efficiency and the memory requirement during the operation and its cost is [22, Section 5]:

$$(2j^2 + j)\mathcal{M} + (4j^2 + 4j + 2)\mathcal{A} .$$

This complexity is lower than the one of the PMNS. However, in the Montgomery-CIOS method one has to deal with the carry propagation between the blocks representing an integer. This is not the case for the PMNS since each block is a coefficient of a polynomial. In practice (see next section) the performances of the PMNS is almost the same than OpenSSL or GMP when using low level functions and

Montgomery-CIOS method.

8.2 Implementations and results

We implemented a PMNS generator based on the process presented in Section 7.1. This generator focuses on polynomials E proposed in Section 7.4 and the ones of AMNS. We also implemented a C code generator. Given a PMNS (obtained from the preceding generator), it generates C codes for all the necessary operations (including forward and backward conversion to PMNS, addition, multiplication) to easily integrate this PMNS into any higher level application that uses the corresponding modulo p for modular arithmetic. These generators were implemented using SageMath library [33] and are available on GitHub [1].

As an example, we present below the performances of software implementations of some AMNS and PMNS systems. These tests aim to provide an overview of the AMNS and PMNS approaches presented in this paper and to give a first comparison with the state of the art `GMP` library [4] (6.2.0 version) and the OpenSSL library [29] (1.1.1 version).

8.2.1 Tested implementation configurations

For the same prime of size 256 bits, we generated the AMNS/PMNS systems with the new generation process. In order to compare with previous works, we provide in Table 2 the sizes of ρ for the targeted prime p. These results show that we are able to reduce the value of ρ and consequently the number of bits required to store a field element. For a modulus p of size 256 bits, we spare 10 to 15 bits out of roughly 300 bits.

The software implementations are based upon four systems:

- an AMNS with $n = 5$, $E(X) = X^n - 2$, $\phi = 2^{64}$ and $\rho = 2^{55}$;
- a PMNS with $n = 5$, $E(X) = X^n - X - 1$, $\phi = 2^{64}$ and $\rho = 2^{55}$;
- an AMNS with $n = 6$, $E(X) = X^n - 2$, $\phi = 2^{52}$ and $\rho = 2^{47}$.
- a PMNS with $n = 6$, $E(X) = X^n - X - 1$, $\phi = 2^{52}$ and $\rho = 2^{47}$.

While both first systems are implemented using classic C language, the next systems take advantage of the `AVX512` SIMD[2] instruction set. In order to use this instruction set, the value of ϕ has to be chosen according to the integer multiplication instructions available, among which there are:

- `VPMULLD`, computing 16 products of 32 bit operands and returning the 32 low bits;
- `VPMULLQ`, computing 8 products of 64 bit operands and returning the 64 low bits;
- `VPMADD52LUQ` and `VPMADD52HUQ`, computing 8 products of 52 bit integers stored in contiguous 64 bit words, providing respectively the 52 low bits or the 52 high bits of the results.

The first two instructions are well suited for systems with $\phi = 2^{32}$. However, there is no instruction computing full multiplications of 64 bit operands, making $\phi = 2^{64}$ a worse choice. The `VPMADD52` is a vectorized fused multiplier-adder instruction. In our implementation, we take advantage of this instruction setting $\phi = 2^{52}$, allowing to avoid the additions in the computations. While this value leads to $n = 6$ higher than the one for the other systems, the use of SIMD instructions decreases drastically the instruction number, roughly divided by more than 6 in comparison with the classic C implementations.

One may notice that with $n = 6$, the polynomial multiplications require 36 elementary multiplications. In our implementation, for each polynomial multiplication $\bmod E(X)$, we use 6 `VPMADD52` instructions per 52 bit words of the required results, leading to compute $6 \times 8 = 48$ elementary multiplications, using only 36 of them. Thus, we can expect that one can choose values of n slightly greater without significant penalty. This also means that for prime modules of greater sizes (with $n \leqslant 8$ and $\phi \leqslant 2^{52}$), the cost of the PMNS modular multiplication will not increase as significantly as expected when implemented with this instruction set.

8.2.2 Experimentation procedure

We hereby present the performance test procedure. Measurements were performed on a Dell Inspiron laptop with an Intel `Tiger Lake` processor.

```
vendor_id : GenuineIntel
model name : 11th Gen Intel(R) Core(TM)
             i7-1165G7 @ 2.80GHz
```

The compiler is `gcc` version 10.2.0, the compiler options are as follows:
`-O3 -g -lgmp -mavx512f -mavx512dq -mavx512vl -mavx512ifma`.
The test procedure is as follows:

- the *Turbo-Boost*® is deactivated during the tests;
- 1000 runs are executed in order to "heat" the cache memory, i.e. we ensure that the cache memory (data and instruction) is in an enough stabilized state in order to avoid untimely cache faults;
- one generates 50 random data sets, and for each data set the minimum of the execution clock cycle numbers over a batch of 1000 runs is recorded;
- the performance is the average of all these minimums;
- this procedure is run on console mode, to avoid system perturbations, and obtain the most accurate cycle counts.

The clock cycle number is obtained using the `rdtsc` instruction which loads the current value of the processor's time-stamp counter into a 64-bit register. The processor monotonically increments the time-stamp counter every clock cycle. Hence calling this instruction before and just after a sequence of instructions allows to obtain the number of elapsed cycles. To count the number of instructions, we make use of the `rdpmc` instruction which allows to read the performance-monitoring counters. These counters can be configured to count events such as the number of interrupts received or the number of instructions decoded (see [1]).

8.2.3 Performance results and comparison

The performance results are shown in Table 3. Some comments about the results :

1. https://github.com/arithPMNS/low_memory_efficient_PMNS
2. Single Instruction Multiple Data

$p = 10334922082758664738683805719218010591837432945968628478824689491763472846218 3$				
	AMNS $n=5$	AMNS $n=6$	PMNS $n=5$	PMNS $n=6$
size of ρ in bits				
old parameters [15]	56	48	56	48
new parameters **(This work)**	53	46	54	46
theoretical number of bits required to store a field element in the system				
old parameters [15]	285	294	285	294
new parameters **(This work)**	270	282	275	282

TABLE 2: Comparison of the theoretical sizes of ρ and the required size in order to store a field element, prime p size of 256 bits

$p = 10334922082758664738683805719218010591837432945968628478824689491763472846218 3$								
			This work					
Modular Multiplication	Openssl Montg.-CIOS	`GMP` low level Montg.-CIOS	AMNS $n=5$ Seq. C	AMNS $n=6$ Seq. C	AMNS $n=6$ `AVX512`	PMNS $n=5$ Seq. C	PMNS $n=6$ Seq. C	PMNS $n=6$ `AVX512`
# clock cycles	133	121	134	188	43	135	208	38
# instructions	522	502	378	563	83	381	571	92

TABLE 3: Performance comparison for 256 bits modular multiplication, OpenSSL and GMP, both Montgomery-CIOS multiplications, AMNS $n=5$ and $n=6$, PMNS $n=5$ and $n=6$, `AVX512` $n=6$

- The vectorized `AVX512` implementation of the PMNS $n=6$ version is the fastest, and has the lowest instruction count. It is three times as fast as the SOA `GMP` Montgomery-CIOS multiplication, and 3.5 times as fast as the corresponding AMNS for $n=5$. Notice that because we must choose $\phi = 2^{52}$ for the `AVX512` implementation, one cannot build a PMNS for a 256-bit prime p with $n=5$.
- Though the comparison is not so fair, the vectorized `AVX512` implementation of the PMNS $n=6$ version is more than 5 times faster than its counterpart in sequential C. This is due to the dramatic reduction of the retired instruction number thanks to the `VPMADD` instruction which lowers both multiplication and addition numbers. This holds also for the AMNS equivalent versions.
- The vectorized version of PMNS $n=6$ is slightly faster than the vectorized AMNS $n=6$, while the instruction count is higher. This may be due to the better pipelining of the PMNS version, which does not make use of vectorized shifts. For equivalent sequential versions, the AMNS version is slightly better for both items, i.e. timings and retired instruction counts.
- The SOA `GMP` Montgomery-CIOS multiplication is slightly faster than the classic C implementations of PMNS and AMNS sytems. However, the performance levels of both versions (AMNS and PMNS) are very close, in spite of the different reductions $\bmod E(X)$, and equivalent to the ones of the Openssl Montgomery multiplication, which has a larger number of retired instruction count.
- One may notice that the `GMP` and OpenSSL Montgomery-CIOS multiplications are not constant time implementations, unlike our AMNS/PMNS implementations.

All the implementations of this work are available on github[3].

3. https://github.com/arithcrypto/AMNS-PMNS-PolyMult

9 Conclusion

In this paper, we generalised the results in [15] to any monic polynomial $E \in \mathbb{Z}_n[X]$. We introduced new parameters that improve both the efficiency and the memory requirement of the PMNS. We highlighted some very interesting polynomials E for the external reduction and PMNS parameters, that provide suitable and efficient alternatives to the AMNS. Finally, we presented some results for classic C implementations and provide the first implementation using `AVX512` instruction set. On one hand, these results showed that the polynomials E we highlighted are as efficient as the best polynomials for AMNS, which are $E(X) = X^n \pm 2$. On the other hand, it appeared that PMNS, with the Montgomery-like approach [28] we focused on for internal reduction, is very well suited for vectorisation, since it leads to a speedup of more than 60% compared to GMP or OpenSSL, and more than 71% compared to a classical C implementation of the best AMNS.

Funding This work has been partially funded by TPM Metropol (AAP2021-IMPECQ project).

References

[1] IA-32 Intel(R) architecture software developer's manual, volume 3
[2] Nist post-quantum cryptography standardization, https://csrc.nist.gov/Projects/post-quantum-cryptography
[3] Sike – Supersingular Isogeny Key Encapsulation, https://sike.org/
[4] The GNU Multiple Precision Arithmetic Library (GMP), https://gmplib.org/
[5] Ajwa, I.A., Liu, Z., Wang, P.S.: Gröbner bases algorithm. Technical Reports of the Institute for Computational Mathematics, ICM-199502-00 (versió del 2003) Kent State University (1995)
[6] Babai, L.: On Lovász' lattice reduction and the nearest lattice point problem. Combinatorica **6**(1), 1–13 (1986)
[7] 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)
[8] Bajard, J.C., Imbert, L., Plantard, T.: Arithmetic operations in the polynomial modular number system. In: 17th IEEE Symposium on Computer Arithmetic (ARITH-17) 2005, Cape Cod, MA, USA. pp. 206–213 (2005)

[9] Bajard, J.C., Marrez, J., Plantard, T., Véron, P.: On Polynomial Modular Number Systems over Z/pZ (Feb 2022), https://arxiv.org/abs/2001.03741, to appear in Advances in Mathematics of Communications
[10] Barrett, P.: Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In: Odlyzko, A.M. (ed.) Advances in Cryptology — CRYPTO' 86. pp. 311–323. Springer, Berlin, Heidelberg (1987)
[11] 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)
[12] Chung, J., Hasan, A.: More generalized mersenne numbers. In: International Workshop on Selected Areas in Cryptography. pp. 335–347. Springer (2003)
[13] 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)
[14] Didier, L.S., Dosso, F.Y., El Mrabet, N., Marrez, J., Véron, P.: Randomization of Arithmetic over Polynomial Modular Number System. In: 26th IEEE International Symposium on Computer Arithmetic. vol. 1, pp. 199–206. Kyoto, Japan (Jun 2019)
[15] 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)
[16] El Mrabet, N., Gama, N.: Efficient multiplication over extension fields. In: WAIFI. Lecture Notes in Computer Science, vol. 7369, pp. 136–151. Springer (2012)
[17] El Mrabet, N., Nègre, C.: Finite field multiplication combining AMNS and DFT approach for pairing cryptography. In: ACISP. Lecture Notes in Computer Science, vol. 5594, pp. 422–436. Springer (2009)
[18] Garner, H.L.: The residue number system. IRE Transactions on Electronic Computers **EL 8**(6), 140–147 (1959)
[19] Gauss, C.F., Clarke, A.A.: Disquisitiones Arithmeticae. Yale University Press (1965), http://www.jstor.org/stable/j.ctt1cc2mnd
[20] Janson, S.: Resultant and discriminant of polynomials, http://www2.math.uu.se/~svante/papers/sjN5.pdf
[21] Kedlaya, K.S., Umans, C.: Fast polynomial factorization and modular composition. SIAM Journal on Computing **40**(6), 1767–1802 (2011)
[22] Koc, C.K., Acar, T., Kaliski, B.S.: Analyzing and comparing montgomery multiplication algorithms. IEEE micro **16**(3), 26–33 (1996)
[23] Korkine, A., Zolotareff, G.: Sur les formes quadratiques positives. Mathematische Annalen **11**(2), 242–292 (1877)
[24] Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische annalen **261**, 515–534 (1982)
[25] Mishra, B.: Algorithmic Algebra. Springer-Verlag, Berlin, Heidelberg (1993)
[26] Montgomery, P.L.: Modular multiplication without trial division. Mathematics of Computation **44**(170), 519–521 (1985)
[27] Negre, C.: Side channel counter-measures based on randomized AMNS modular multiplication. In: Proceedings of the 18th International Conference on Security and Cryptography. SCITEPRESS - Science and Technology Publications (2021)
[28] 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)
[29] Project, T.O.: Openssl, https://www.openssl.org/
[30] Proskuryakov, I.: Resultant. Encyclopedia of mathematics, https://encyclopediaofmath.org/wiki/Resultant
[31] Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM **21**(2), 120–126 (1978)
[32] 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)
[33] Stein, W., al.: Sagemath, http://www.sagemath.org/index.html

10 Biography

Fangan Yssouf Dosso obtained his PhD from University de Toulon, Toulon, France in 2020. He is now holding a post-doctoral position at École des Mines de Saint-Étienne. His research interests are computer arithmetic and efficient implementation of cryptographic protocols.

Jean-Marc Robert is an associate professor at University de Toulon, Toulon, France. His research interests are computer arithmetic and efficient implementation of cryptographic protocols.

Pascal Véron is an associate professor at University de Toulon, Toulon, France. His research interests are code based cryptography, computer arithmetic and efficient implementation of cryptographic protocols.