# PMNS arithmetic for elliptic curve cryptography

Fangan-Yssouf Dosso, Sylvain Duquesne, Nadia El Mrabet, Emma Gautier

# PMNS arithmetic for elliptic curve cryptography

Fangan Yssouf Dosso[1][0000−0003−4158−9176], Sylvain Duquesne[2][0000−0002−3854−8253], Nadia El Mrabet[1][0000−0003−3840−584X], and Emma Gautier[2]

[1] SAS laboratory, École des Mines de Saint-Étienne, Gardanne, France
{fanganyssouf.dosso,nadia.el-mrabet}@emse.fr
[2] Univ Rennes, CNRS, IRMAR - UMR 6625, F-35000 Rennes, France
{sylvain.duquesne,emma.gautier}@univ-rennes.fr

**Abstract.** We show that using the polynomial modular number system (PMNS) can be relevant for real-world cryptographic applications even in terms of performance. More specifically, we consider elliptic curves for cryptography when pseudo-Mersenne primes cannot be used to define the base field (e.g. Brainpool standardized curves, JubJub curves in the zkSNARK context, pairing-friendly curves). All these primitives make massive use of the Montgomery reduction algorithm and well-known libraries such as GMP or OpenSSL for base field arithmetic. We show how this arithmetic can be replaced by PMNS, a number system with very high parallelisation capability, no carry propagation, which allows efficient arithmetic randomization. We provide good PMNS bases in the cryptographic context mentioned above, together with a C implementation that is competitive with GMP and OpenSSL for performing basic operations in the base fields considered. We also integrate this arithmetic into the Rust reference implementation of elliptic curve scalar multiplication for Zero-knowledge applications, and achieve better practical performances for such protocols. This shows that PMNS is an attractive alternative for the base field arithmetic layer in cryptographic primitives using elliptic curves or pairings.

**Keywords:** Polynomial Modular Number System · Cryptography · Elliptic curves · Pairings · Brainpool · JubJub.

## 1 Introduction

Elliptic curve cryptography is widely used nowadays: Internet communications make use of it through TLS [27], and several standards include it since the 2000s [48,51]. Additionally, the NIST (National Institute of Standard and Technology) has standardized several curves for cryptography. Let $\mathbb{F}_p$ be the finite field of characteristic $p$. An elliptic curve $E(\mathbb{F}_p)$ can be briefly defined by its Weierstrass equation as $E(\mathbb{F}_p) := \{(x, y) \in \mathbb{F}_p \times \mathbb{F}_p;\ y^2 = x^3 + ax + b,\ \text{for } a, b \in \mathbb{F}_p\}$. By construction, elliptic curve cryptography involves arithmetic over finite fields and the most popular ones (NIST curves, Curve25519) were chosen such that the modular arithmetic over $\mathbb{F}_p$ is very efficient.

More generally, modular arithmetic is one of the most important operations in asymmetric cryptography. Indeed, whether for RSA or ECC but also for more recent post-quantum primitives (Kyber, Dilithium, SqiSign for example), the efficiency of implementations relies on the efficiency of modular operations, and in particular modular multiplication. When $p$ can be chosen in a specific form, the modular multiplication can be performed as a classical multiplication followed by a modular reduction that is not expensive because of the sparsity of $p$. When possible, the shape of $p$ is then constructed as pseudo-Mersenne (or even Mersenne) primes [53]. This is for example the case of the elliptic curves standardised by the NIST P-224, P-256, P-384 and P-521. However, after the discovery of a trapdoor in the elliptic curve used for the DUAL ECB DRBG standard of the NIST [17,14], the scientific community has proposed other elliptic curves for the standards, generated in a more random way and ensuring the right level of security [11]. Some of these curves are defined on finite fields whose binary decomposition of the characteristic is not sparse [16,5,41]. Modular arithmetic is then more expensive and more generic algorithms like Montgomery multiplication [46] or Barrett reduction [7] must be used in order to ensure efficient implementation.

These methods are based on the classical binary number system; an element of $\mathbb{F}_p$ is an integer between 0 and $(p-1)$ or between $\frac{-(p-1)}{2}$ and $\frac{(p-1)}{2}$. To further improve modular arithmetic, alternative number systems have been proposed. A famous one is the Residue Number System (RNS) [29], which has been widely studied and implemented. It is an efficient number system which has a very high parallelisation capability. Another one is the Polynomial Modular Number System (PMNS) [2], proposed by Bajard et al. for modular arithmetic modulo a prime number $p$. In this system, elements are represented by polynomials with small coefficients. Like the RNS, it offers a very high parallelisation capability but also additional properties like the possibility of randomising the arithmetic. Many works have shown the efficiency of PMNS for both software [20,44,23] and hardware [50] implementations considering random prime numbers but without consideration for their cryptographic use. Our purpose in this paper is to show, and illustrate on concrete examples, that this way to represent numbers is interesting for cryptographic applications, even in terms of performance and for relatively small primes as the ones used in elliptic curve cryptography, which was not evident a priori.

Cryptographers have been working on developing open source libraries for finite fields and elliptic curves. There are two mainstream libraries for cryptography: GMP and OpenSSL. Since 1991, the GMP library manages large numbers (among others functionalities) for the C implementation language [31]. GMP is designed for maximum speed for large operands. Its efficiency comes from state-of-the-art arithmetic optimizations, fast algorithms, and highly optimized assembly code. GMP is continuously improved, with a new release every year. OpenSSL is another free to use software library for general-purpose cryptography and secure communication [40]. It includes implementations for standardised elliptic curves. These two libraries are considered to be a reliable resource and

are used by cryptographers to compare the effectiveness of new software implementations. That is the reason why we decided to compare our results with these libraries.

**Our contributions:** in this article we want to answer the question: can PMNS be advantageously used for real-world cryptography? To do this, we generate PMNS basis for the base fields of some popular elliptic curves that cannot be defined over pseudo-Mersenne primes. More precisely we study the Brainpool elliptic curves, pairing-friendly elliptic curves such as BN, BLS12 and KSS16, and the JubJub elliptic curve used in the cryptocurrency Zcash. We implement the PMNS arithmetic for each type of elliptic curve, and compare as fairly as we can the number of clock cycles for performing a modular multiplication between our implementation of PMNS without any assembly code and the two state-of-the-art versions of GMP [31] and OpenSSL [40]. As a result, our implementation is competitive with the reference ones at the finite field arithmetic level, sometimes better and sometimes slightly worse depending on the context (bit-size, use of assembly code for example).
Concerning the JubJub elliptic curve, we provide a comparison with the reference library of Zcash written in Rust both at the finite field level (modular multiplication) and at the elliptic curve level (scalar multiplication). In both cases, we show that using a PMNS arithmetic is significantly more efficient than using the reference arithmetic.
Note that the theoretical complexity in terms of word multiplications is better for the Montgomery arithmetic but there are many factors that can influence practical comparisons (carry propagation, impact of the cost of additions and other ancillary instructions, quality of the implementation and compilation, level of adaptation to specific field sizes,...). As a consequence, this paper does not prove that the PMNS always provides better efficiency for elliptic curve based cryptography but that, despite a higher theoretical complexity in terms of word multiplications, it is sufficiently competitive to be used in practice (especially if one could take advantage of its additional properties).

**Organization of the paper:** in Section 2 we recall elementary definitions of the PMNS, with the description of efficient algorithms. In particular, in Section 2.9, we recall the advantages and drawbacks of PMNS arithmetic. Our first comparison is made on the Brainpool elliptic curves in Section 3, we compare our implementation with the OpenSSL implementation. We then work on pairing-friendly elliptic curves in Section 4 and on the JubJub curve for Zcash in Section 5. We conclude our work and open new perspectives in Section 6.

## 2 PMNS

Let us first introduce and explain the Polynomial Modular Number System (PMNS) as well as its advantages and drawbacks in the cryptographic context. Let us start with some notations that we will be using throughout this paper.

### 2.1 Notations

For consistency, we assume that $p \geqslant 3$ and $n \geqslant 2$.

- $\mathbb{Z}_{n-1}[X]$ is the set of polynomials in $\mathbb{Z}[X]$ with degrees lower than or equal to $n-1$:

$$\mathbb{Z}_{n-1}[X] = \{C \in \mathbb{Z}[X] \ \mid \ \deg(C) \leqslant n-1\}.$$

- Let $A \in \mathbb{Z}_{n-1}[X]$ be a polynomial. We assume that:
  - $A(X) = a_0 + a_1 X + \cdots + a_{n-1}X^{n-1}$
  - $A$ can equivalently be represented as the vector $A = (a_0, \ldots, a_{n-1}) \in \mathbb{Z}^n$.
- Also, $\|A\|_\infty = \max\limits_{0 \leqslant i \leqslant n-1} |a_i|$.
- Let $\mathcal{W} \in \mathbb{Z}^{n \times n}$ be a matrix. $\|\mathcal{W}\|_1 = \max\limits_{0 \leqslant j \leqslant n-1} \sum_{i=0}^{n-1} |w_{ij}|$.

### 2.2 Definition and example

The PMNS has been introduced by Bajard et al. [2]. It is a non-positional number system for modular arithmetic modulo a prime integer $p$. A PMNS is defined by a tuple $(p, n, \gamma, \rho, E)$, where $p$, $n$, $\gamma$ and $\rho$ are positive non-zero integers and $E \in \mathbb{Z}_n[X]$ is a monic polynomial such that $E(\gamma) \equiv 0 \pmod{p}$. In this system, elements are represented by polynomials. More precisely, we have the following definition.

**Definition 1.** *A polynomial modular number system (PMNS) is defined by a tuple $\mathcal{B} = (p, n, \gamma, \rho, E)$, such that for each integer $y \in \mathbb{Z}/p\mathbb{Z}$ there exists a polynomial $V(X) = v_0 + v_1.X + \cdots + v_{n-1}.X^{n-1}$ such that:*

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

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

*Remark 1.* From the definition above, we say that a polynomial $A \in \mathbb{Z}[X]$ is in the PMNS $\mathcal{B} = (p, n, \gamma, \rho, E)$ if and only if:

$$\deg(A) < n \ \text{ and } \ \|A\|_\infty < \rho,$$

where $\|A\|_\infty = \max\limits_{0 \leqslant i \leqslant n-1} |a_i|$.

*Example 1.* Let us take $p = 19$. For this prime, we have the PMNS $\mathcal{B} = (9, 3, 7, 2, E)$, with $E(X) = X^3 - 1$. Table 1 gives an example of representation for each elements of $\mathbb{Z}/19\mathbb{Z}$. In this table, one can observe that the polynomial $X^2 - 1$ is a representation of 10, since we have $10 = 7^2 - 1 \pmod{19}$.

Table 1: Example of representations for the elements of $\mathbb{Z}/19\mathbb{Z}$ in $\mathcal{B} = (19, 3, 7, 2, X^3 - 1)$

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 0 | 1 | $-X^2 - X + 1$ | $X^2 - X - 1$ | $X^2 - X$ |

| 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|
| $X^2 - X + 1$ | $X - 1$ | $X$ | $X + 1$ | $-X^2 + 1$ |

| 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|
| $X^2 - 1$ | $X^2$ | $X^2 + 1$ | $-X + 1$ | $-X^2 + X - 1$ |

| 15 | 16 | 17 | 18 |
|---|---|---|---|
| $-X^2 + X$ | $-X^2 + X + 1$ | $X^2 + X - 1$ | $-1$ |

### 2.3 Arithmetic operations in the PMNS

Like most number systems, the main operations in the PMNS are the addition and the multiplication. Let $\mathcal{B} = (p, n, \gamma, \rho, E)$ be a PMNS, and $A, B \in \mathcal{B}$.

The addition is the simple polynomial addition, i.e. $C = A + B$. However, there is no guarantee that $C \in \mathcal{B}$, since we only have $\|C\|_\infty < 2\rho$. To guarantee the result in $\mathcal{B}$, we need to perform an operation called **internal reduction**. This operation is equivalent to the modular reduction in the classical representation. In Section 2.5 we describe it, and in Section 2.6 we explain why it is not interesting to perform this operation after a simple polynomial addition, and we present the parameter $\delta$ which has been introduced to deal with this "problem".

The multiplication is the polynomial multiplication, i.e. $C = A \times B$. Here also, there is no guarantee that $C \in \mathcal{B}$. In fact, we only have:

$$\deg(C) < 2n - 1 \quad \text{and} \quad \|C\|_\infty < n\rho^2.$$

Here, we first reduce the degree of $C$. That is, we compute a polynomial $R \in \mathbb{Z}_{n-1}[X]$ such that:

$$R(\gamma) \equiv C(\gamma) \pmod{p}.$$

This operation is called the **external reduction**. It is done using the polynomial $E$, which is thus called the external reduction polynomial.
The external reduction is the operation:

$$R = C \bmod E.$$

Remember that $E$ is a monic polynomial, with $\deg(E) = n$ and $E(\gamma) \equiv 0 \pmod{p}$. This thus guarantees that $\deg(R) < n$ and $R(\gamma) \equiv C(\gamma) \pmod{p}$.

In practice, we want this reduction to be very fast. So, the parameter $E$ is chosen very sparse, with very small coefficients. For example, $E(X) = X^n - \lambda$, where $|\lambda|$ is a very small non-zero integer.

Let us assume that $E(X) = X^n + e_{n-1}X^{n-1} + \cdots + e_1X + e_0$. To optimise the external reduction, the external reduction matrix $\mathcal{E}$ in introduced in [23]. It is the $(n-1) \times n$ matrix defined as follows:

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

Let us assume that $C = A \times B = c_0 + c_1X + \cdots + c_{2n-2}X^{2n-2}$. With the external reduction matrix $\mathcal{E}$, the authors show:

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

So for very efficient external reduction, we want the external reduction matrix $\mathcal{E}$ to be very sparse with very small coefficients. This is achieved with polynomials $E(X) = X^n - \lambda$ having $|\lambda|$ very small. Moreover, [23, Table 1] gives several additional polynomials $E$ that achieve this goal.

Let us now look at an example of modular multiplication in the PMNS.

*Example 2.* Let us consider the PMNS $\mathcal{B} = (9, 3, 7, 2, X^3 - 1)$ given in Example 1. Let $a = 10$ and $b = 15$. In Table 1, $A(X) = X^2 - 1$ is a representation of 10, while $B(X) = -X^2 + X$ is a representation of 15.
Let us multiply $A$ by $B$. We have:

$$C(X) = A(X) \times B(X) = -X^4 + X^3 + X^2 - X\,.$$

One can verify that: $C(7) \equiv 17 \pmod{19}$ and $10 \times 15 \equiv 17 \pmod{19}$. However, $C \notin \mathcal{B}$, since $\deg(C) > 2$. So, we perform the external reduction:

$$R(X) = C(X) \bmod E(X) = X^2 - 2X + 1\,.$$

We have $\deg(R) < 3$, however $R \notin \mathcal{B}$, because $\|R\|_\infty \not< \rho$.

Now let us consider the polynomial $T(X) = 3X - 2$, which is such that $T(\gamma) \equiv 0 \pmod{p}$. Let us compute the polynomial $S = R + T$.
We have $S(X) = X^2 + X - 1$. So,

$$S(\gamma) \equiv C(\gamma) \pmod{p} \quad \text{and} \quad S \in \mathcal{B}.$$

Thus, the polynomial $S$ is the result of $A \times B$ in the PMNS $\mathcal{B}$.
The question then is: given $R \in \mathbb{Z}_{n-1}[X]$, how does one compute such a polynomial $T$? This is the **internal reduction** we present in Section 2.5.
Before presenting the internal reduction, we discuss the Euclidean lattice that comes with every PMNS.

### 2.4 PMNS and euclidean lattices

A PMNS $\mathcal{B} = (p, n, \gamma, \rho, E)$ is associated with the $n$-dimensional full-rank Euclidean lattice $\mathcal{L}_{\mathcal{B}}$, defined as follows:

$$\mathcal{L}_{\mathcal{B}} = \{A \in \mathbb{Z}_{n-1}[X] \quad A(\gamma) \equiv 0 \pmod{p}\}. \tag{3}$$

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

$$\mathbf{B} = \begin{pmatrix} p & 0\,0 \ldots 0\,0 \\ t_1 & 1\,0 \ldots 0\,0 \\ t_2 & 0\,1 \ldots 0\,0 \\ \vdots & \ddots \quad \vdots \\ t_{n-2} & 0\,0 \ldots 1\,0 \\ t_{n-1} & 0\,0 \ldots 0\,1 \end{pmatrix} \begin{matrix} \leftarrow p \\ \leftarrow X + t_1 \\ \leftarrow X^2 + t_2 \\ \\ \leftarrow X^{n-2} + t_{n-2} \\ \leftarrow X^{n-1} + t_{n-1} \end{matrix}, \tag{4}$$

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

In [21], the following result gives a condition on the parameter $\rho$ for $\mathcal{B}$ to be a PMNS.

**Lemma 1.** *[21] Let $\mathcal{L}$ be a sub-lattice of $\mathcal{L}_{\mathcal{B}}$, having a matrix $\mathcal{G}$ as a basis. A tuple $\mathcal{B} = (p, n, \gamma, \rho, E)$ defines a PMNS if:*

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

### 2.5 The internal reduction

Let $\mathcal{B} = (p, n, \gamma, \rho, E)$ be a PMNS. Let $R \in \mathbb{Z}_{n-1}[X]$ be a polynomial, with possibly $\|R\|_\infty \geqslant \rho$. The internal reduction consists in computing a polynomial $S$ such that $\|S\|_\infty < \rho$ and $S(\gamma) \equiv R(\gamma) \pmod{p}$.
Many approaches have been proposed to perform this operation [2,1,47,44].
In [47], the authors propose a Montgomery-like method. This method has been proven efficient for both software [20,23] and hardware [50] implementations.
In [21], the authors show that this Montgomery-like method works on a particular sub-lattice $\mathcal{L}(\mathcal{M})$ of $\mathcal{L}_{\mathcal{B}}$ (see [21, Equation 9] which gives the basis $\mathcal{M}$ used). They then generalise the Montgomery-like method to any basis $\mathcal{G}$ of any sub-lattice $\mathcal{L}$ of $\mathcal{L}_{\mathcal{B}}$. In this paper, we focus on this extended Montgomery-like method, which they call **GMont-like**.

Let $\mathcal{L}$ be a sub-lattice of $\mathcal{L}_{\mathcal{B}}$. **GMont-like** requires three parameters: a basis $\mathcal{G}$ of $\mathcal{L}$, an integer $\phi \geqslant 2$ such that $\gcd(\phi, \det(\mathcal{G})) = 1$ , and the matrix $\mathcal{G}' = -\mathcal{G}^{-1} \pmod{\phi}$. In Section 2.8, we discuss the choice of $\mathcal{G}$ and $\phi$. Algorithm 1 describes **GMont-like**.

*Remark 2.* Note that **GMont-like** introduces a factor $\phi^{-1}$ on the output. So, similar to the classical Montgomery method, elements in the PMNS are placed

**Algorithm 1** Coefficients reduction for PMNS (**GMont-like**) [21]

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

in a Montgomery domain to ensure consistency of operations in the system. That is, an integer $a \in \mathbb{Z}/p\mathbb{Z}$ is represented in the PMNS $\mathcal{B}$ by a polynomial $A$ such that $A(\gamma) \equiv (a\phi) \pmod p$. This is done during the integer to PMNS conversion, as proposed in [20]. In Section 2.7 we recall the conversion processes.

With **GMont-like**, Algorithm 2 describes the modular multiplication in the PMNS.

**Algorithm 2** Modular multiplication for PMNS (**ModMult**)[21]

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

### 2.6 The parameter $\delta$ and bounds

Recall that the addition in the PMNS is the simple polynomial addition, which is quite simple. In comparison, the internal reduction (Algorithm 1) is very expensive. Thus, it is not interesting to perform this reduction after an addition (to guarantee the result in $\mathcal{B}$). To solve this "issue", the authors in [20] introduce a parameter $\delta$. It is the maximum number of consecutive additions of elements in $\mathcal{B}$ that we want to compute before doing a multiplication (**ModMult**: Algorithm 2). Its value is chosen according to the target application.

Let $A$ and $B$ each be the result of such successive additions. Then, we have: $\|A\|_\infty, \|B\|_\infty < (\delta + 1)\rho$. If $S =$ **ModMult**$(A, B)$, we want $S \in \mathcal{B}$.
This is achieved by taking the parameters $\rho$ and $\phi$ such that [22]:

$$\rho = \|\mathcal{G}\|_1 - 1 \quad \text{and} \quad \phi \geqslant 2w(\rho - 1)(\delta + 1)^2\,, \tag{5}$$

where:

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

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

*Remark 3.* In this paper we are not interested in redundancy control or equality test within the PMNS. If redundancy control in the PMNS and/or equality testing within the system is desired, the parameters $\rho$ and $\phi$ should instead be chosen as proposed in [21]:

$$\rho = \|\mathcal{G}\|_1 + 1 \quad \text{and} \quad \phi \geqslant 2\lceil w(\delta+1)^2 \|\mathcal{G}^{-1}\|_1 \|\mathcal{G}\|_1^2 \rceil.$$

### 2.7 Conversion operations

This section briefly presents the conversion operations to and from the PMNS. Two methods have been proposed for conversion to PMNS. One is slow (Algorithm 3) and is used for pre-computation. The other one (Algorithm 4), which is fast, requires pre-computed polynomials. It also performs the conversion to the Montgomery domain mentioned in Remark 2. The polynomials $P_i$ required by Algorithm 4 are computed using Algorithm 3.

**Algorithm 3** Exact Conversion to PMNS

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

**Algorithm 4** Fast Conversion to PMNS

**Require:** $a \in \mathbb{Z}/p\mathbb{Z}$, $\mathcal{B} = (p, n, \gamma, \rho, E)$ and the matrices $\mathcal{G}$, $\mathcal{G}'$,
$\beta$ the smallest power of 2 such that $\beta^n > p$, and $P_i \in \mathcal{B}$ such that $P_i(\gamma) \equiv \beta^i \phi^2 \pmod{p}$ for $i = 0 \ldots (n-1)$.
**Ensure:** $A \in \mathcal{B}$, such that $A \equiv (a\phi)_{\mathcal{B}}$
1: $t = (t_{n-1}, ..., t_0)_\beta$ # radix-$\beta$ decomposition of $a$
2: $U \leftarrow \sum_{i=0}^{n-1} t_i P_i(X)$
3: $A \leftarrow$ **GMont-like**$(U)$
4: return $A$

Conversion from PMNS is a simple polynomial evaluation modulo $p$. One must also not forget the conversion from the Montgomery domain. That is, let $A \in \mathcal{B}$, we compute: $a = A(\gamma)\phi^{-1} \pmod{p}$. An easy way to perform this

operation is to use the classical Horner's scheme [39], or it can be done more efficiently by pre-computing $\gamma^i \phi^{-1} \pmod p$, for $i = 0, \ldots, n-1$, for very fast polynomial evaluation modulo $p$.

### 2.8 Generating PMNS basis in practice

This section describes the steps to generate PMNS for a given modulus $p$. Before generating the PMNS, we need to chose the value of $\delta$ according to the target application (see Section 2.6). Additionally, for **GMont-like** to be efficient, the parameter $\phi$ must be taken as a power of two. Let us assume that the target hardware is a $h$-bit architecture. As suggested in [20], a good choice for efficient internal reduction is to take $\phi = 2^h$. Thus, implying the parameter $n$ to be such that:

$$n \geqslant \left\lfloor \frac{\lceil \log_2(p) \rceil}{h} \right\rfloor + 1 \,.$$

Depending on the value of $\delta$, the value of $n$ will be increased as much as necessary in order to find a suitable PMNS. As mentioned in Section 2.3, we want a polynomial $E$ which allows very efficient external reduction. Table 2 gives some examples of such polynomials $E$. Algorithm 5 describes PMNS generation process.

Table 2: Example of polynomials $E$ for efficient external reduction and small memory cost.

| $E(X)$ | $n$ | $w$ |
|---|---|---|
| $X^n - \lambda$, with $\lvert\lambda\rvert$ very small | – | $1 + (n-1)\lvert\lambda\rvert$ |
| $X^n \pm X^{\frac{n}{2}} + 1$ | 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 | $2n-1$ |
| $X^n \pm X \pm 1$ | – | $2n-1$ |
| $X^n + X^{n-1} + \cdots + X + 1$ | – | $2n-1$ |

**Algorithm 5** PMNS generation

**Require:** $p$, $\delta$, $\phi = 2^h$, and $\lambda_{\max}$ the maximum for $|\lambda|$ in Table 2.
**Ensure:** $\mathcal{B}$ is a PMNS
1: $n = \left\lfloor \frac{\lceil \log_2(p) \rceil}{h} \right\rfloor + 1$
2: found = False
3: **while** not found **do**
4: Randomly (or iteratively) choose a polynomial $E$ in Table 2 having a root $\gamma$ modulo $p$.
5: Compute the basis $\mathbf{B}$ of $\mathcal{L}_{\mathcal{B}}$ (see Equation 4)
6: $\mathcal{G} \leftarrow \mathrm{LLL}(\mathbf{B})$ # or use BKZ, HKZ, ...
7: $\rho \leftarrow \|\mathcal{G}\|_1 - 1$
8: **if** $\phi < 2w(\rho - 1)(\delta + 1)^2$ **then**
9: **if** all the polynomials $E$ have been checked **then**
10: $n \leftarrow n + 1$
11: **end if**
12: Choose another polynomial $E$ (i.e. go back to step 4)
13: **end if**
14: found = True
15: $\mathcal{B} \leftarrow (p,\ n,\ \gamma,\ \rho,\ E,\ \mathcal{G},\ \phi,\ \delta)$
16: **end while**
17: return $\mathcal{B}$

*Remark 4.* At line 6 of Algorithm 5, a lattice basis reduction is performed using the LLL algorithm. Here the goal is to have a basis $\mathcal{G}$ with the smallest possible 1-norm. Indeed, the parameter $\rho$, which is the upper bound on the infinity norm of PMNS elements, is computed with $\|\mathcal{G}\|_1$ (see line 7). Since we want to use as little memory as possible to represent elements in the PMNS, we want the $\mathcal{G}$ 1-norm to be as small as possible. In addition, the smaller $\rho$ is, the more likely it is that the **if** statement on line 8 will not be satisfied, thus giving more chance of finding PMNS with smaller of $n$, which is better for efficiency.
To sum up, if you have a better basis reduction algorithm/implementation than LLL (i.e. that leads to a basis $\mathcal{G}$ with a smaller 1-norm), you must use it instead.

### 2.9 Advantages and drawbacks

Elements in PMNS are polynomials. This has many advantages. Here, we highlight some of them:

- Since polynomial coefficients are independent, there is no carry propagation to deal with when performing arithmetic operations, unlike the classical representation where one has to deal with this propagation.
- Arithmetic and conversion operations have no conditional branching. This property is an advantage for efficiency and is also very helpful against side channel attacks.

- As the coefficients are independent, they can be computed simultaneously. As a consequence, PMNS is well fitted for a parallel implementation.
  Note that in this paper we only present results for sequential implementations. We do not take advantage of the high parallelisation capability of PMNS, which should lead to much better efficiency. Parallel hardware implementations of PMNS have been studied and compared to classical Montgomery modular multiplication in [50].
- PMNS is a redundant number system. This redundancy can easily be used to randomise arithmetic and conversion operations. This can be used to protect some cryptographic protocols against side-channel attacks, as shown in [19].
- Finally, thanks to the parameter $\delta$, it is possible to add PMNS elements without having to perform an internal reduction (which is in comparison very expensive). In the classical representation, this is not possible when the modulus size is $h \times n$ (e.g. when $\lceil \log_2(p) \rceil = 256 = 64 * 4$, with $h = 64$).

Compared to the classical representations, PMNS has two main disadvantages:

- The first one is the generation process. For applications where new moduli need to be generated many times, Algorithm 5 may be too expensive, mainly because of the lattice basis reduction at line 6. Note, however, that in practice the modulus size is in most cases small enough to make this reduction very efficient. This is the case, for example, with ECC moduli and moduli used in post-quantum cryptographic schemes. In fact, with our experiments using the SageMath library [54], we observe that for modulus sizes smaller than 2048 bits, this reduction is fast and efficient.
- The second one is the number of coefficients (i.e. the value of $n$). The PMNS generation process presented in Section 2.8 requires to take $n \geqslant \left\lfloor \frac{\lceil \log_2(p) \rceil}{h} \right\rfloor + 1$. In practice, this often results in PMNS having one more coefficient than the classical representation. However, thanks to the polynomial structure of its elements, PMNS may still remain very competitive in this case, as will be shown in the next application sections. Of course, when the modulus size leads to the same number of coefficients than the classical representation, PMNS will be more efficient. This is for example the case for the modulus of the parings `KSS16-330` and `BN-462` (see Section 4.3).

## 3 Application to Elliptic Curves defined over random prime fields

### 3.1 The Brainpool curves

For efficiency reasons, Mersenne or pseudo-Mersenne primes are very often used for base fields in classical elliptic curve cryptography. This is for example the case of the standard NIST curves such as the P256 curve [49] defined over the prime field of order $2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$ which has been extensively used in cybersecurity during the last decades. However it is not satisfying for several reasons :

- this parameter is not randomly generated which can be considered as a potential weakness. In particular, there has been some debate about whether the NSA introduced a backdoor in the P256 curve because the seed for generating it has not been published by the NIST,
- this may yield to use patented algorithm for efficient modular arithmetic specific to these primes (even if these patents have now expired),
- for hardware devices, specific and not flexible designs must be deployed to benefit of such an efficient modular arithmetic which may be very expensive [52],
- sparse prime field (as well as sparse order of the curve because of the Hasse-Weil bound) requires larger blinding factors for its randomization to protect against side-channel attacks [52].

To address these problems, the ECC Brainpool consortium suggested to use only pseudo-random numbers to generate ECC parameters and introduced rigidity in the process thanks to the notion of verifiable pseudo-random [16]. For example, they propose to use natural constants (as the digits of $e = exp(1)$) in place of random seeds. Even if the so-called Brainpool curves have not been defined following this procedure, the idea remains the same and is described in an RFC [45]. The Brainpool initiative was the first open work aiming to produce a fully transparent and verifiable pseudo-random elliptic curve generation process.

For example the `BrainpoolP256r1` curve is defined by the short Weierstrass equation

$$y^2 = x^3 + Ax + B$$

over the prime field $\mathbb{F}_p$ with

$$p = 76884956397045344220809746629001649093037950200943055203735601445031516197751$$

$$A = 56698187605326110043627228396178346077120614539475214109386828188763884139993$$

$$B = 17577232497321838841075697789794520262950426058923084567046852300633325438902$$

As a consequence of its randomizing, the base field $\mathbb{F}_p$ do not have any specificity so that only generic modular arithmetic can be used. It may then be interesting to generate a PMNS basis for this prime and compare the PMNS representation arithmetic in this field with more classical ones such as Montgomery [46] or Barrett [7].

Recently (November 2023), OpenSSL 3.2.0 integrated the Brainpool curves. This shows that, even if Curve25519 [12] is more and more popular for its efficiency, the need for randomly generated alternative is still relevant. As OpenSSL is clearly a recognised reference in cryptographic implementations, we decided to compared it with our PMNS arithmetic for the 256, 384 and 512-bits prime fields of Brainpool.

### 3.2 PMNS parameters

As mentioned above, we consider the moduli of Brainpool `P256r1`, `P384r1` and `P512r1` curves. This section gives the parameters of the PMNS generated for these moduli, using Algorithm 5 of Subsection 2.8.

Let us start with `brainpoolP256r1` modulus. We get the following parameters:

$p = 76884956397045344220809746629001649093037950200943055203735601445031516197751$

$n = 5$

$\gamma = 69419170576495936951229257827200835275135500751173168338114307761123131575941$

$\rho = 5798424837117361$

$E = X^5 - 5$

$\phi = 2^{64}$

$\delta = 7$

The parameters for `brainpoolP384r1` modulus are:

$p = 21659270770119316173069236842332604979796116387017648600081618503821089934025961822236561982844534088440708417973331$

$n = 7$

$\gamma = 11122730497867082426703774898586389214554972136239004234162613722782469803511232111483936084892743587563706779788925$

$\rho = 92143305512072319$

$E = X^7 - 2$

$\phi = 2^{64}$

$\delta = 1$

The parameters for `brainpoolP512r1` modulus are:

$p = 89489622076502325516566028151591534221626096440983545113445971872000570104135524399179343041919569427654465303864273459379638943099239285360705346078169 47$

$n = 9$

$\gamma = 20670140623632241292106902112791444395638356376389137152957073973914031952411388598133886529587536136669343079369256892137492508555078458075711180903933 37$

$\rho = 92143305512072319$

$E = X^9 - 2$

$\phi = 2^{64}$

$\delta = 0$

### 3.3 Measurement procedure

Before presenting our comparison results, we first describe our measurement procedure for `C` language implementations. Measurements were performed on a HP EliteBook laptop, with:

- Processor: Intel 11`th Gen Intel Core` $i5$-$1135G7@2.40GHz \times 8$
- Memory: 16 GiB of RAM
- OS: Ubuntu 20.04.6 LTS (64 bits)

The compiler is `gcc` version 9.4.0, the compiler options are as follows: `-Wall -O3 -lgmp -lcrypto`.
The test procedure is as follows:

- the *Turbo-Boost*® is deactivated during the tests;
- 1001 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 51 random data sets, and for each data set the minimum of the execution clock cycle numbers over a batch of 1001 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.

Our C implementations of PMNS presented in this paper can be found in this GitHub repository:

https://github.com/PMNS-APPLICATION/C-code-brainpool-pairing

### 3.4 PMNS Arithmetic vs `OpenSSL` library

In this section, we compare PMNS to `OpenSSL` for modular multiplication with the moduli of the Brainpool curves. We made this choice because Brainpool curves are now officially included in OpenSSL. Table 3 gives clock cycle numbers for PMNS, for the standard modular multiplication provided in `OpenSSL` (Std in the table) and for its implementation of Montgomery modular multiplication (Bloc-Mont in the table). Note that the latter is the most relevant element of comparison since PMNS uses a Montgomery-like reduction method, and both approaches require some pre-computation and conversion in a Montgomery domain. It can be seen that PMNS is faster for `brainpoolP256r1` and `brainpoolP384r1`, but slower for `brainpoolP512r1`. For the latter, we checked

the `OpenSSL` library and found that it uses the Karatsuba method of integer multiplication for this size of integer and above. Due to time constraints, we have not yet done this for PMNS. This will be done in a future work and we expect to outperform the `OpenSSL` implementation again. Also note that, as mentioned in Section 2.9, we are presenting a sequential implementation here, while PMNS is well suited for parallel implementation. So these PMNS should be expected to perform much better with a parallel implementation.

Table 3: Clock cycle number comparisons of Modular Multiplication for **brainpool** curve moduli.

| Modulus | `PMNS` | `OpenSSL` | |
|---|---|---|---|
| | | Bloc-Mont | Std |
| brainpoolP256r1 | 177 | 181 | 718 |
| brainpoolP384r1 | 267 | 294 | 1071 |
| brainpoolP512r1 | 405 | 347 | 1385 |

## 4 Application to pairing-friendly base fields

### 4.1 Pairing-friendly Base Fields

Pairings are interesting in cryptography because of the large number of protocols they can be used for; as identity-based cryptography, multiple signature schemes, attribute-based cryptography, or hierarchical encryption... [24,25]. Pairing-based cryptography is based on the arithmetic of finite fields $\mathbb{F}_p$, for $p$ a large prime number, extensions of finite fields $\mathbb{F}_{p^k}$, and the arithmetic of an elliptic curve defined over these two finite fields [25]. There are different constructions of pairing curves [28]. Some are called pairing-friendly elliptic curves because they allow efficient pairing calculations. Among the pairing-friendly elliptic curves, the BN [6], BLS [5], and KSS [41] curves present a good trade off between the size of $\mathbb{F}_p$ and $\mathbb{F}_{p^k}$. The BN and BLS curves are recommended by the Internet Engineering Task Force [51] at 128 and 194 bit security level.

Such pairing-friendly elliptic curves are determined by a polynomial expression for the number $p$ defining the finite field $\mathbb{F}_p$. This expression depends on a parameter often noted $u$, such that $p(u)$ is a prime of predetermined size in bits. The size of $p$ depends on the level of security. The most effective discrete logarithm attack for pairing-friendly elliptic fields and curves is Kim and Barbulescu's TNFS attack [42]. It has a different impact on families of pairing-friendly elliptic curves [3,4]. Construction of pairing-friendly elliptic curve relies on the Complex Multiplication [25]. The value of $p$ is randomly generated during the construction. Although the expression of $p$ obtained for a pairing-friendly elliptic curve is polynomial in $u$, the prime numbers allowing correct parameters

for pairing-friendly elliptic curve have no structure allowing efficient modular arithmetic [51]. We recall in Table 4 the parametric definition of the large prime $p$ for the BN, BLS, and KSS families. The BN-462 curve is standardised at the 128 bits security level [51]; the two others families are the ones that allow efficient pairing computation in terms of operation complexity, arithmetic on finite fields and elliptic curves, and execution time [51]. The BLS12 curve is implemented in the cryptocurrencies Zcash [57] and Ethereum [26]. The KSS curve does not seem to be used in practice, but it has good properties for efficient implementations [3].

Table 4: Expression of $p$ according to the pairing-friendly family of curve

| Family | Expression of $p$ |
|---|---|
| BN | $p(u) = 36u^4 + 36u^3 + 24u^2 + 6u + 1$ |
| BLS12 | $p(u) = \frac{1}{3}(u-1)^2(u^4 - u^2 + 1) + u$ |
| KSS16 | $p(u) = \frac{1}{980}\left(u^{10} + 2u^9 + 5u^8 + 48u^6 + 152u^5 + 240u^4 + 625u^2 + 2398u + 3125\right)$ |

### 4.2 PMNS parameters

Section 3.4 presents a comparison between PMNS and the `GMP` library for modular multiplication with the moduli used in pairing-based cryptography. More precisely, we consider the moduli of the base fields defining the pairing-friendly curves `BN-462`, `BLS12-381` and `KSS16-330`. This section gives the parameters of the PMNS generated for these moduli.

Let us start with `BLS12-381` modulus. Thanks to algorithm 5 given in Subsection 2.8, we get the following parameters:

$$
\begin{aligned}
p &= 4002409555221667393417789825735904156556882819939007885332058136124031650490837864442687629129015664037894272559787\\
n &= 7\\
\gamma &= 8439523423486319150638028379657090639981335620567352002001654235297606413082346297697287941199117717482560648925 58\\
\rho &= 68257199819371999\\
E &= X^7 + X + 1\\
\phi &= 2^{64}\\
\delta &= 2
\end{aligned}
$$

The parameters for `KSS16-330` modulus are:

$$
\begin{aligned}
p &= 212217013448868677911006528509820149111113937118179634024000718517587036701914288606384779931398810 9\\
n &= 6\\
\gamma &= 60808169966120777972963619891987217547500954332684752541949217311778145736788958698964721384788511 4\\
\rho &= 81242268367433855\\
E &= X^6 + X + 1\\
\phi &= 2^{64}\\
\delta &= 2
\end{aligned}
$$

The parameters for `BN-462` modulus are:

$$
\begin{aligned}
p &= 670181705631303708624894706631053844488208260530812457623040803884335754988635677985739336996701076480254100579671144035575350370105632360 3\\
n &= 8\\
\gamma &= 55098291808261871392910158198001852674413573883807121974779235181961160895734336464505544343973215151928599272173961181497037364444512082942\\
\rho &= 789727156710839423\\
E &= X^8 - X - 1\\
\phi &= 2^{64}\\
\delta &= 0
\end{aligned}
$$

### 4.3 PMNS Arithmetic vs `GMP` library

In this section, we compare PMNS to `GMP` for modular multiplication with the moduli presented above. We use exactly the same procedure described in Section 3.3 with the same hardware. Table 5 gives clock cycle numbers for PMNS, for the standard modular multiplication provided in `GMP` (Std in the table), for its implementation of Montgomery modular multiplication (Bloc-Mont in the table). `GMP` also allows the use of some low-level functions that are slightly faster than the standard ones. The corresponding function for modular multiplication is called Low-lvl in Table 5. Similar to the `OpenSSL` library, note that Bloc-Mont is the most relevant element of the comparison, since PMNS uses a Montgomery-like reduction method, and both approaches require some pre-computation and conversion in a Montgomery domain.

First, one can observe in Table 5 that our PMNS implementation is always faster than GMP standard and low-level implementations. Regarding GMP implementation of Montgomery modular multiplication (i.e. GMP Bloc-Mont in

Table 5: Clock cycle number comparisons of Modular Multiplication for pairing-friendly base fields

| Modulus | PMNS | GMP | | |
|---|---|---|---|---|
| | | Bloc-Mont | Low-lvl | Std |
| KSS16-330 | 225 | 248 | 494 | 541 |
| BN-462 | 349 | 368 | 709 | 762 |
| BLS12-381 | 275 | 249 | 496 | 547 |

the table), it can be seen that PMNS is also faster, except for `BLS12-381`. This is because, for `KSS16-330` and `BN-462`, PMNS and GMP use the same number of coefficients to represent their elements. As mentioned in section 2.9, PMNS is faster in this case. For `BLS12-381`, the corresponding PMNS uses one more coefficient than GMP ($n = 7$ instead of 6).

*Remark 5.* As mentioned earlier, we are comparing sequential implementations here, whereas PMNS is much more suited to parallel implementation. In addition, GMP uses assembly code to optimise its operations, whereas our implementation of PMNS is standard C code without any assembly code. With equivalent optimisation (perhaps in future work), we expect our PMNS implementation to always outperform the GMP implementation of Montgomery modular multiplication.

## 5 Application to Elliptic Curves for Zero-Knowledge protocols

Most of today's digital services use centralized systems, which implies a trusted authority. For some years now, there has been growing interest in decentralised systems, such as blockchains, which avoid the need for a central authority to certify data. However, most existing systems require parties to publish all their computations and recalculate those published by other parties to ensure that they are correct. This is expensive, especially in terms of memory storage. For example, the size of the bitcoin chain reached 600 GigaBytes in September 2024. To deal with this problem (and not only in the blockchain context), it is convenient to use proof systems [32], where a prover convinces a verifier that a given statement is correct. More specifically, zk-SNARK (zero-knowledge Succinct Non-interactive ARgument of Knowledge) provides a proof (that is short and easy to verify) of the correct execution of the computation [8] and is very useful in decentralised systems. Such systems are not trivial to construct, and it is not the aim of this paper to describe such construction but the interesting point for our purpose is that the most popular ones are using pairing-friendly elliptic curves [36,33,37,34,30,35]. Elliptic curves and base fields involved have specific constraints so that base field providing efficient reduction (such as pseudo-Mersenne prime) cannot be used in this context. As a consequence, using efficient alternative reduction algorithm as the PMNS must be considered. There

are many zk-SNARK construction depending on the use-case. For example if one wants to use proof of proofs, it is necessary to find chains of pairing-friendly elliptic curves such that the prime order of the first has to be chosen to define the base field of the second one [18]. Indeed, in pairing-based SNARK, the verification is usually made in the base field while the proof is made in the field defined by the prime order of the considered elliptic curve subgroup (usually called the scalar field). This illustrate the fact that at least one of the base field cannot be chosen in a specific form. This is for example used in the MINA protocol [15] which is a very light blockchain thanks to the use of zk-SNARK (fixed size of 22 KiloBytes). But there is no need to choose such advanced protocols to meet a base field that cannot be chosen in the blockchain context.

### 5.1 The JubJub elliptic curve

To illustrate the interest of using PMNS arithmetic in the zk-SNARK context, we will focus on Zcash which is a cryptocurrency based on bitcoin and where anonymity is ensured thanks to zk-SNARK. The zk-SNARK is made on the BLS12-381 pairing-friendly elliptic curve already introduced in Subsection 4 to prove an Elliptic Curve Diffie-Hellman (ECDH) key exchange following the CØCØ construction [43]. As the proof is made on the scalar field, the cardinality of the prime subgroup involved in the pairing must be used to define the base field of the elliptic curve where the ECDH key exchange is performed. As a consequence, the JubJub curve used in [56] is defined over $\mathbb{F}_p$ with

$$p = 52435875175126190479447740508185965837690552500527637822603658699938581184513$$

for which no specific reduction algorithm is known. Indeed, as it comes from the cardinality of the BLS12-381 curve, it has no specific form. The JubJub curve is then defined as a twisted Edwards curve in a deterministic way following the SafeCurve recommendations [13]

$$E_J : -u^2 + v^2 = 1 + du^2v^2 \tag{6}$$

with $d = -(10240/10241)$
$= 19257038036680949359750312669786877991949435402254120286184196891950884077233$

It has cofactor 8 and we can use $G = (8076246640662884909881801758704306714034609987455869804520522091855516602923, 13262374693698910701929044844600465831413122818447359594527400194675274060458)$ as a base point.

### 5.2 Efficient group law on the JubJub elliptic curve

The most efficient coordinate system for this curve is the extended projective one [38,9]. In this case, a point on the curve $E_J$ is given by 4 coordinates

$(U, V, Z, T)$ in $\mathbb{F}_p$ such that $u = \frac{U}{Z}, v = \frac{V}{Z}$ are satisfying the defining equation of $E_J$ 6 and $T = \frac{UV}{Z}$ .
For doubling a point in this form, the following formulas are used [38]

$$\begin{array}{ll} T_2 = V^2 + U^2 & T_2' = V^2 - U^2 \\ T_1 = (U+V)^2 - T_2 & J = 2Z^2 - T_2' \\ U = T_1 J & V = T_2' T_2 \\ Z = T_2' J & T = T_1 T_2 \end{array}$$

These formulas require 4 multiplications and 4 squarings in the base field $\mathbb{F}_p$. Note that many additions or subtractions are also involved in these formulas, so that the use of the $\delta$ parameter introduced in Subsection 2.6 is crucial in order to avoid internal reduction after each addition/subtraction operation. More precisely, there is 1 non-reduced addition in the expressions of $T_2, T_2'$ so that it is sufficient to have $\delta = 1$ in order to compute $V = T_2' T_2$ without intermediate internal reduction. However $T_1 = (U+V)^2 - T_2$ and $J = 2Z^2 - T_2'$ are made of respectively 2 and 3 additions assuming that $T_2$ and $T_2'$ have not been reduced. As a consequence we need to have a $\delta$ value equal to 3 in order to compute efficiently $U$ and $Z$.

We do not give the mixed addition formulas for this system of coordinates here (they can be find in [38] or on [10]) because they do not provide any additional information or constraint on the choice of $\delta$.

### 5.3 PMNS parameters for JubJub arithmetic

We then apply the method described in Subsection 2.8 to generate a PMNS basis for the modulus used to construct the `JubJub` elliptic curve. In this specific context, as explained in the previous section, we also have to choose a parameter $\delta$ at least equal to 3 to avoid internal reductions after addition/subtraction operations involved in the group law. The parameters of the PMNS generated for this modulus are given by:

$$\begin{aligned} p &= 52435875175126190479447740508185965837690552500527637822603658\\ &\quad 699938581184513\\ n &= 5\\ \gamma &= 17165118212817083565366397558045145856414930757760865323558718\\ &\quad 474655557272360\\ \rho &= 5216921637660971\\ E &= X^5 - 2\\ \phi &= 2^{64}\\ \delta &= 13 \end{aligned}$$

### 5.4 Comparison of efficiency

For safety reasons many cryptocurrency protocols are using Rust libraries. This is a nice opportunity to compare our PMNS arithmetic with classical ones in

another language than C. We chose the Zcash official library [55] to make this comparison. We first compare the cost of basic modular multiplication (which is classically done using Montgomery reduction in the Zcash library) as in the other sections of this paper.

All the measurments of this section were performed on a Legion by Lenovo laptop, with:

- Processor: Intel `7th Gen Intel Core` $i5\text{-}7300HQ \,@\, 2.50GHz \times 4$
- Memory: 8 GiB of RAM
- OS: Ubuntu 20.04.5 LTS (64 bits)

The compiler used is `cargo` version 1.84.1. The test procedure for the Modular Multiplication is as follows:

- 10000 runs are executed. For each iteration of run execution, the Modular Multiplication is made with two random elements of the JubJub base field (the same elements are used in both arithmetic systems for a fair comparison),
- the execution time and execution clock cycle numbers required by the modular multiplication operation are recorded for each iteration, the performance is the average of those iterations,
- the performance given in Table 6 is the average of 10 times the batch of 10000 iterations.

The execution time is obtained using the `std::time::{Duration, Instant}` library surrounding the multiplication. The clock cycle number is obtained using `std::arch::asm;` library to create the `rdtsc` instruction.

Our Rust implementations can be found in this GitHub repository:

https://github.com/PMNS-APPLICATION/Rust-code-jubjub

Table 6: Clock cycle number and execution time comparisons of Modular Multiplication for **JubJub** curve base field.

| | PMNS | | Classical | |
|---|---|---|---|---|
| | Clock cycle number | Execution time (ns) | Clock cycle number | Execution time (ns) |
| JubJub | 723 | 313.5 | 1157 | 484.1 |

Thanks to PMNS, we also get in this case an implementation of the base field arithmetic which is more efficient than the one by default in the targeted library.

To complete the comparison, we also decided to compare the full scalar multiplication computation at the elliptic curve level. For this, we use the Zcash

code [55] for the scalar multiplication (a constant-time double-and-add algorithm in extended projective coordinates [38,9]) on top of both their and our base field arithmetic. The group law formulas given in Subsection 5.2 are naturally used, but in practice, in the Zcash implementation, the last line of the doubling is not performed and the point is stored as $(U, V, Z, T_1, T_2)$. As the $\delta$ value of the PMNS basis is quite large in this case, we can easily avoid internal reduction after all the additions/subtraction operations involved in the group law formulas.

The comparison of the full scalar multiplication between the classical and PMNS arithmetic has been made in the same environment that the measurements of performance of the Modular Multiplication. The tests are made with a random multiple of the affine base point $G$ and a random element of the JubJub scalar field. Of course the same elements are used for both side of the comparison and new ones are drawn at each iteration. The performance result are given in Table 7.

Table 7: Clock cycle number and execution time comparisons of Full Scalar Multiplication on **JubJub** elliptic curve.

| | PMNS | | Classical | |
|---|---|---|---|---|
| | Clock cycle number | Execution time (ns) | Clock cycle number | Execution time (ns) |
| JubJub | 2 992 436 | 1.1655 | 3 386 125 | 1.3185 |

It is not very surprising that these results confirm the ones obtained on the base field. But it shows that PMNS arithmetic can be easily used to replace classical Montgomery arithmetic on the base field and that it remains an interesting alternative at the elliptic curve level. Moreover, these results do not take into account the inherent properties of the PMNS arithmetic compared to classical ones (easy parallelisation, possibility of randomisation).

## 6 Conclusion

In this article, we illustrate how PMNS can be used for concrete cryptographic applications. We have chosen to work on cryptography based on elliptic curves because they are now essential in asymmetric cryptography and because, in some contexts, they require modular arithmetic modulo primes $p$ that do not allow efficient specific arithmetic. PMNS arithmetic is an alternative to classical arithmetic for such primes. We consider the Brainpool curve (standardised and included in the widely deployed OpenSSL library), pairing-friendly elliptic curves for security level 128 (used in many advanced protocols), and the Jubjub elliptic curve (used in the Zcash cryptocurrency).

We compared our implementation on 64-bits architectures with the two mainstream libraries for cryptographers, GMP and OpenSSL. Both libraries rely on assembly code for optimisations. Our PMNS implementation was more naive, we implemented it in C in a sequential way without any assembly optimisations. We also compare a Rust version of our approach with the official Zcash library at both the finite field and elliptic curve levels. These implementations and test procedures can be checked out the following git:

https://github.com/PMNS-APPLICATION/

Our results are very promising, despite the lack of use of assembly code, and the extra machine word for the PMNS representation compared to the classical one. In many cases, our implementation outperformed OpenSSL, GMP and the Rust library for Zcash. This was not so obvious a priori because the addition of an extra machine word has an important impact when the total number of machine words is small, as it is the case for elliptic curve cryptography (n=4 to 8). This study then shows that the advantages of the PMNS representation (e.g. no carry propagation) compensate this drawback in practice. Note that this result must be put into perspective because, for our comparisons, we used general-purpose and widely deployed libraries which are not necessarily highly optimised for specific use cases (and our implementations either).

As a consequence, this work is the first step towards the introduction of PMNS arithmetic in cryptography. Further work on the implementation aspects, such as parallelisation or the introduction of some assembly routines, would be relevant to improve our PMNS implementation in the context of elliptic curve cryptography. It would certainly allow much better performances but also provide new opportunities inherent to PMNS properties, such as randomising the base field arithmetic against side channel attacks.

It would also be very interesting to complete this study in the context of post-quantum cryptography. In particular, the isogeny-based signature SQiSign makes intensive use of large prime field arithmetic and some important lattice-based cryptographic protocols use prime numbers (e.g. Dilithium or Fully Homomorphic Encryption).

**Acknowledgments.** This work was supported in part by French project ANR-11-LABX-0020-01 "Centre Henri Lebesgue". The authors thank Michael Scott for useful remarks and comments.

## References

1. Bajard, J., Imbert, L., Plantard, T.: Arithmetic operations in the polynomial modular number system, 206–213, 17th IEEE Symposium on Computer Arithmetic (ARITH-17 2005), 27-29.
2. Bajard, J.C., Imbert, L., Plantard, T.: Modular number systems: Beyond the mersenne family. In: Handschuh, H., Hasan, M.A. (eds.) Selected Areas in Cryptography. pp. 159–169. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)

3. Barbulescu, R., Duquesne, S.: Updating key size estimations for pairings. J. Cryptol. **32**(4), 1298–1336 (2019). https://doi.org/10.1007/S00145-018-9280-5, https://doi.org/10.1007/s00145-018-9280-5
4. Barbulescu, R., El Mrabet, N., Ghammam, L.: A taxonomy of pairings, their security, their complexity. IACR Cryptol. ePrint Arch. p. 485 (2019), https://eprint.iacr.org/2019/485
5. Barreto, P.S.L.M., Lynn, B., Scott, M.: On the selection of pairing-friendly groups. In: Matsui, M., Zuccherato, R.J. (eds.) Selected Areas in Cryptography, 10th Annual International Workshop, SAC 2003, Ottawa, Canada, August 14-15, 2003, Revised Papers. Lecture Notes in Computer Science, vol. 3006, pp. 17–25. Springer (2003). https://doi.org/10.1007/978-3-540-24654-1_2, https://doi.org/10.1007/978-3-540-24654-1_2
6. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Preneel, B., Tavares, S.E. (eds.) Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers. Lecture Notes in Computer Science, vol. 3897, pp. 319–331. Springer (2005). https://doi.org/10.1007/11693383_22, https://doi.org/10.1007/11693383_22
7. 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)
8. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von neumann architecture. In: Proceedings of the 23rd USENIX Conference on Security Symposium. p. 781–796. SEC'14, USENIX Association, USA (2014)
9. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. J Cryptogr Eng **2**, 77–89 (2012), https://doi.org/10.1007/s13389-012-0027-1
10. Bernstein, D.J., Lange, T.: Extended coordinates with a=-1 for twisted edwards curves, http://www.hyperelliptic.org/EFD/g1p/auto-twisted-extended-1.html
11. Bernstein, D.J., Lange, T.: Explicit-formulas database for elliptic curve cryptography. https://hyperelliptic. org/EFD
12. Bernstein, D.J.: Curve25519: New diffie-hellman speed records. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) Public Key Cryptography - PKC 2006. pp. 207–228. Springer Berlin Heidelberg (2006)
13. Bernstein, D.J., Lange, T.: Safecurves: choosing safe curves for elliptic-curve cryptography, https://safecurves.cr.yp.to
14. Bernstein, D.J., Lange, T., Niederhagen, R.: Dual EC: A standardized back door. In: Ryan, P.Y.A., Naccache, D., Quisquater, J. (eds.) The New Codebreakers - Essays Dedicated to David Kahn on the Occasion of His 85th Birthday. Lecture Notes in Computer Science, vol. 9100, pp. 256–281. Springer (2016). https://doi.org/10.1007/978-3-662-49301-4_17, https://doi.org/10.1007/978-3-662-49301-4_17
15. Bonneau, J., Meckler, I., Rao, V., Shapiro, E.: Coda: Decentralized cryptocurrency at scale. Cryptology ePrint Archive, Paper 2020/352 (2020), https://eprint.iacr.org/2020/352
16. Brainpool: Ecc brainpool standard curves and curve generation (2005), http://www.ecc-brainpool.org/download/Domain-parameters.pdf

17. Checkoway, S., Niederhagen, R., Everspaugh, A., Green, M., Lange, T., Ristenpart, T., Bernstein, D.J., Maskiewicz, J., Shacham, H., Fredrikson, M.: On the practical exploitability of dual EC in TLS implementations. In: Fu, K., Jung, J. (eds.) Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014. pp. 319–335. USENIX Association (2014), https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/checkoway
18. Costello, C., Fournet, C., Howell, J., Kohlweiss, M., Kreuter, B., Naehrig, M., Parno, B., Zahur, S.: Geppetto: Versatile verifiable computation. In: 2015 IEEE Symposium on Security and Privacy. pp. 253–270 (2015). https://doi.org/10.1109/SP.2015.23
19. Didier, L., Dosso, F., El Mrabet, N., Marrez, J., Véron, P.: Randomization of arithmetic over polynomial modular number system. https://doi.org/10.1109/ARITH.2019.00048, 26th IEEE Symposium on Computer Arithmetic, ARITH 2019, 199–206
20. 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)
21. 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
22. Dosso, F.Y., El Mrabet, N., Méloni, N., Palma, F., Véron, P.: Friendly primes for efficient modular arithmetic using the polynomial modular number system. Cryptology ePrint Archive, Paper 2025/090 (2025), https://eprint.iacr.org/2025/090
23. 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://hal.science/hal-03768546v1/file/TETC3187786.pdf
24. Dutta, R., Barua, R., Sarkar, P.: Pairing-based cryptographic protocols : A survey. IACR Cryptol. ePrint Arch. p. 64 (2004), http://eprint.iacr.org/2004/064
25. El Mrabet, N., Joye, M.: Guide to Pairing-Based Cryptography. Chapman & Hall/CRC (2016)
26. Ethereum: https://ethereum.org/en/
27. Force, I.E.T.: Transport layer security. https://datatracker.ietf.org/wg/tls/about/
28. Freeman, D., Scott, M., Teske, E.: A taxonomy of pairing-friendly elliptic curves. J. Cryptol. **23**(2), 224–280 (2010). https://doi.org/10.1007/S00145-009-9048-Z, https://doi.org/10.1007/s00145-009-9048-z
29. Garner, H.L.: The residue number system. IRE Transactions on Electronic Computers **EL 8**(6), 140–147 (1959)
30. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct nizks without pcps. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology – EUROCRYPT 2013. pp. 626–645. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
31. GMP: The GNU Multiple Precision Arithmetic library. https://gmplib.org
32. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on Computing **18**(1), 186–208 (1989). https://doi.org/10.1137/0218012, https://doi.org/10.1137/0218012
33. Groth, J.: Simulation-sound nizk proofs for a practical language and constant size group signatures. In: Lai, X., Chen, K. (eds.) Advances in Cryptology – ASIACRYPT 2006. pp. 444–459. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)

34. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Abe, M. (ed.) Advances in Cryptology - ASIACRYPT 2010. pp. 321–340. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
35. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016. pp. 305–326. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
36. Groth, J., Ostrovsky, R., Sahai, A.: Non-interactive zaps and new techniques for nizk. In: Dwork, C. (ed.) Advances in Cryptology - CRYPTO 2006. pp. 97–111. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
37. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N. (ed.) Advances in Cryptology – EUROCRYPT 2008. pp. 415–432. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
38. Hisil, H., Wong, K.K.H., Carter, G., Dawson, E.: Twisted edwards curves revisited. In: Pieprzyk, J. (ed.) Advances in Cryptology - ASIACRYPT 2008. pp. 326–343. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
39. Horner, W.G.: A new method of solving numerical equations of all orders, by continuous approximation. Philosophical Transactions of the Royal Society of London **109**, 308–335 (1819)
40. Software Foundation Inc: OpenSSL Library. https://openssl.org
41. Kachisa, E.J., Schaefer, E.F., Scott, M.: Constructing brezing-weng pairing-friendly elliptic curves using elements in the cyclotomic field. In: Galbraith, S.D., Paterson, K.G. (eds.) Pairing-Based Cryptography - Pairing 2008, Second International Conference, Egham, UK, September 1-3, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5209, pp. 126–135. Springer (2008). https://doi.org/10.1007/978-3-540-85538-5_9, https://doi.org/10.1007/978-3-540-85538-5_9
42. Kim, T., Barbulescu, R.: Extended tower number field sieve: A new complexity for the medium prime case. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9814, pp. 543–571. Springer (2016). https://doi.org/10.1007/978-3-662-53018-4_20, https://doi.org/10.1007/978-3-662-53018-4_20
43. Kosba, A.E., Zhao, Z., Miller, A.K., Qian, Y., Chan, T.H.H., Papamanthou, C., Shi, E.: Cøcø: A framework for building composable zero-knowledge proofs (2016), https://api.semanticscholar.org/CorpusID:17760053
44. Méloni, N.: An Alternative Approach to Polynomial Modular Number System Internal Reduction. IEEE Transactions on Emerging Topics in Computing (Jul 2022). https://doi.org/10.1109/TETC.2022.3190368
45. Merkle, J., Lochter, M.: Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation. RFC 5639 (2010). https://doi.org/10.17487/RFC5639, https://www.rfc-editor.org/info/rfc5639
46. Montgomery, P.: Modular multiplication without trial division. Mathematics of Computation **44 (170)**, 519–521 (1985). https://doi.org/10.1090/S0025-5718-1985-0777282-X
47. Negre, C., Plantard, T.: Efficient modular arithmetic in adapted modular number system using lagrange representation (2008), 463–477 (Springer Berlin Heidelberg, 2008)
48. NIST: The elliptic curve digital signature algorithm validation system. https://csrc.nist.gov/presentations/2004/the-elliptic-curve-digital-signature-algorithm-val
49. NIST: Digital signature standard (dss), nist fips 186-4 (2009), http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

50. 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)
51. Sakemi, Y., Kobayashi, T., Saito, T., Wahby, R.S.: Pairing-Friendly Curves. Internet-Draft draft-irtf-cfrg-pairing-friendly-curves-11, Internet Engineering Task Force (Nov 2022), https://datatracker.ietf.org/doc/draft-irtf-cfrg-pairing-friendly-curves/11/, work in Progress
52. Schütze, T.: Requirements for standard elliptic curves position paper of the ecc brainpool (2014), https://api.semanticscholar.org/CorpusID:16091050
53. Solinas, J.A.: Pseudo-Mersenne Prime, pp. 992–992. Springer US, Boston, MA (2011). https://doi.org/10.1007/978-1-4419-5906-5_42, https://doi.org/10.1007/978-1-4419-5906-5_42
54. Stein, W., al.: Sagemath (2005), http://www.sagemath.org/, last accessed 07 Jul 2022
55. Zcash: Rust implementation of the jubjub elliptic curve group, https://github.com/zkcrypto/jubjub/tree/main
56. Zcash: Zcash documentation, https://zcash.readthedocs.io/en/latest/
57. ZcashTeam: Zk snark in zcash. https://z.cash