# A software comparison of RNS and PMNS

Laurent-Stephane Didier, Jean-Marc Robert, Fangan Yssouf Dosso, Nadia El Mrabet

# A software comparison of RNS and PMNS

Laurent-Stéphane Didier, Jean-Marc Robert
*Laboratoire IMath*
*Université de Toulon*
La Garde, France
laurent-stephane.didier@univ-tln.fr
jean-marc.robert@univ-tln.fr

Fangan Yssouf Dosso, Nadia El Mrabet
*Département SAS*
*École des Mines de Saint-Étienne*
Gardanne, France
fanganyssouf.dosso@emse.fr
nadia.el-mrabet@emse.fr

***Abstract*—The Polynomial Modular Number System (PMNS) and the Residue Number System (RNS) are integer number systems which aim to speed up modular arithmetic. Their parallel properties make them suitable for the implementation of cryptographic applications on modern processors with SIMD instructions. In this work, we will show the implementation choices made for the modular multiplication in both systems and compare their implementation performances for several sizes of moduli. We target the Intel 64-bit sequential instruction set and the Intel AVX-512 vector instruction set. This instruction set allows significant speed-ups up to 1 621 bit size moduli, while the vectorized PMNS implementation is up to 2.5 times faster than the vectorized RNS, though the vectorized RNS becomes slightly better for 3 251 bits, due to the difficulty to find a PMNS with a suitable parameter $n$. The vectorized RNS implementations reach performance levels close the state-of-the-art GMP library, while the retired instruction counts are lower for sizes between 401 and 3 251 bits.**

***Index Terms*—Modular arithmetic, Residue Number System, Polynomial Number System, SIMD**

## I. Introduction

Most protocols in public key cryptography require modular arithmetic operations over large integers. The classical representation is the main choice of cryptographers and standards [1]. In this representation an element of a finite field $\mathbb{F}_p$, for $p$ a large integer, is seen as an integer modulo $p$. The GMP library [2] is the reference for efficiency in the classical representation.

Alternative candidates such as Residue Number System (RNS) and Polynomial Modular Number System (PMNS) have been independently studied in the literature [3]–[15]. Those representations of a finite field aim to speed up implementation of modular arithmetic. The RNS representation consists in choosing a base of $n$ small relatively primes $m_i$ such that $\prod_{i=1}^{n} m_i \geq p$. The computations are done in the smaller fields $\mathbb{F}_{m_i}$ and the reconstruction of the integers relies on the Chinese remainder theorem. The size of the $m_i$s is adapted to fit the machine words. The Polynomial Modular Number System (PMNS) is a non positional, modular integer number system in which elements are represented as polynomials with integer coefficients. The representation ensures that the coefficient size is small enough to make the computation efficient. In practice, this size corresponds to the size of machine words. These two representations support the paralellization of the computation since operations over each moduli or coefficients are independent. Furthermore, both arithmetics have side-channel attack resistance properties [9], [16], [17].

Several hardware implementations illustrate that those representations can be more efficient than the classical one [18]–[20]. However, no software comparison exists. The software library available in [15] implements the PMNS arithmetic in sequential mode. The parallelization is now possible using vectorization even in software implementation.

**Our contributions:** we propose the first software comparison of the RNS and PMNS, in both cases, i.e. parallel and sequential implementations of Montgomery modular multiplications. The coefficient number in PMNS representation influences the efficiency of computations. We propose an evaluation of the optimal number of coefficients for given sizes of large prime $p$. We compare the performance results of our parallel and sequential PMNS and RNS implementations to the GMP library [2].

**Organisation of the paper:** In Section II we will recall the definition and properties of RNS and PMNS representation. The complexity of both representations is analysed in Section III. Section IV presents our implementations and results.

## II. Background on PMNS and RNS

### A. Background on RNS

Residue Number Systems are non positional integer number systems based on the Chinese Remainder Theorem [21]–[23]. In this system an integer $x$ is represented by its remainders $x_i = x \bmod m_i$, where $m_i$ are relatively prime numbers. The set $\mathcal{B}_m = \{m_1, m_2, \ldots, m_n\}$ constitutes the RNS base composed of $n$ channels. Usually, the moduli $m_i$ are chosen of the same $w$-bit size. We denote $\boldsymbol{M}$ their product. The advantage of such a number system is that additions, subtractions and multiplications can be performed in parallel on each channel:

$$z_i = x_i \odot y_i \bmod m_i \text{ where } \odot \in \{+, -, \times\}$$

*a) Conversions:* The forward conversion from binary system is simply a modular operation on each base channel. The backward conversion can be done using different approaches. The Chinese Remainder Theorem provides a computation formula in the target number system [23].

2576-2265/22/$31.00 ©2022 IEEE
DOI 10.1109/ARITH54963.2022.00025

$$x = \left| \sum_{i=1}^{n} x_i \left( \frac{\boldsymbol{M}}{m_i} \right)^{-1}_{m_i} \boldsymbol{M}_i \right|_{\boldsymbol{M}} = \sum_{i=1}^{n} x_i \left( \frac{\boldsymbol{M}}{m_i} \right)^{-1}_{m_i} \boldsymbol{M}_i - k \cdot \boldsymbol{M} \quad (1)$$

where

$$\boldsymbol{M}_i \times \left( \frac{\boldsymbol{M}}{m_i} \right)^{-1}_{m_i} \equiv 1 \pmod{\boldsymbol{M}}$$

Unfortunately, the values used in this sum are large. The conversion to the Mixed Radix System requires modular computations on $w$-bit integers. In this positional system, an integer $x_{MRS}$ is expressed as follows:

$$x_{MRS} = x'_0 + x'_1 m_0 + x'_2 m_0 m_1 + \cdots + x'_{n-1} \prod_{i=0}^{n-2} m_i$$

The conversion into this system [24] requires $\mathcal{O}(n^2)$ operations on $w$-bit numbers and $\mathcal{O}(n^2)$ stored constants.

A trade-off between these two methods has been proposed by Kawamura et al. [6]. They propose to estimate $k$ in equation (1) with approximate values through $\mathcal{O}(n)$ operations on small values with $\mathcal{O}(n)$ constants.

*b) Base extension:* The base extension allows the conversion of an RNS number from one RNS base to another. This operation consists in a backward conversion and a forward conversion to the targeted RNS base. Both operations are usually interleaved in order to minimize intermediate-value storage.

The base extension proposed by Szabo and Tanaka is based on the mixed-radix conversion [24]. Shenoy and Kumaresan suggested to compute the value $k$ in equation (1) using an extra modulus $m_e$ [3]. The knowledge of $k$ allows to compute equation (1) in the target RNS base. Similarly, Kawamura *et al.* proposed another conversion method using an approximate evaluation of $k$ [6].

*c) Modular multiplication:* The RNS modular multiplication is derived from the Montgomery multiplication [25] and requires base extensions [4], [26]. In Algorithm 1 the base extensions can be performed with different strategies. In [5], the authors remark that if the dynamic range of base $\mathcal{B}_{m'}$ is large enough, then the first extension can be approximated. For the second, they use the Shenoy-Kumaresan method [3]. In the multiplication described in [6] both extensions are Kawamura's.

In our implementations of Algorithm 1, we chose $\mathcal{B}_m$ and $\mathcal{B}_{m'}$ in order to use the Bajard-Imbert [5] first extension at step 3. For the second extension in step 7, we use the Kawamura *et al.* method [6].

### B. Background on PMNS

Polynomial Modular Number System (PMNS) is a non-positional, modular integer number system in which elements are represented as polynomials with integer coefficients. A PMNS is defined by a tuple $(p, n, \gamma, \rho, E)$, where $p$, $n$, $\gamma$ and $\rho$ are nonzero positive integers, and $E \in \mathbb{Z}[X]$ is a monic polynomial, such that $\deg(E) = n$ and $E(\gamma) \equiv 0 \pmod{p}$.

**Algorithm 1** RNS Modular Multiplication

```
Require: $x$ in $\mathcal{B}_m$ and $\mathcal{B}_{m'}$; $y$ in $\mathcal{B}_m$ and $\mathcal{B}_{m'}$ such that $x < 2p$ and $y < 2p$.
Precomputation: $-p^{-1}$ in $\mathcal{B}_{m'}$; $p$ in $\mathcal{B}_m$; $\boldsymbol{M}^{-1}$ in $\mathcal{B}_m$
Ensure: $z = x \times y \times \boldsymbol{M}^{-1} \bmod p$ in $\mathcal{B}_m$ and $\mathcal{B}_{m'}$ such that $z < 2p$.
1: $s \leftarrow x \times y$ in $\mathcal{B}_{m'}$ and $\mathcal{B}_m$
2: $t \leftarrow s \times (-p^{-1})$ in $\mathcal{B}_{m'}$
3: Base extension of $t$ from $\mathcal{B}_{m'}$ to $\mathcal{B}_m$
4: $u \leftarrow t \times p$ in $\mathcal{B}_m$
5: $v \leftarrow s + u$ in $\mathcal{B}_m$
6: $w \leftarrow v \times \boldsymbol{M}^{-1}$ in $\mathcal{B}_m$
7: Base extension of $w$ from $\mathcal{B}_m$ to $\mathcal{B}_{m'}$
8: return $w$
```

This system has been pioneered by Bajard et al. [11]. Many works have then been done to improve it [12], [15], [27], [28] or to use it for higher level operations [9], [13], [14], [17], [29].

In the sequel, $\mathbb{Z}_n[X]$ denotes the set of polynomials in $\mathbb{Z}[X]$ which degrees are 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 can equivalently be represented as the vector $a = (a_0, \ldots, a_n)$. Let $D, E \in \mathbb{Z}[X]$ be two polynomials, then $D \bmod (E, \phi)$ denotes the polynomial reduction $D \bmod E$, where the coefficients of the result are computed modulo $\phi \in \mathbb{N} \setminus \{0, 1\}$.

**Definition 1.** Let $p \geqslant 3$ and $n, \rho \geqslant 2$ be three integers.
Let $E \in \mathbb{Z}_n[X]$ be a monic polynomial and $\gamma \in \mathbb{Z}/p\mathbb{Z} \setminus \{0\}$, such that $E(\gamma) \equiv 0 \pmod{p}$. A tuple $\mathcal{B} = (p, n, \gamma, \rho, E)$ is a PMNS if:

1) $\forall A \in \mathcal{B}$, $\deg(A) < n$ and $\|A\|_\infty < \rho$,
2) $\forall a \in \mathbb{Z}/p\mathbb{Z}$, $\exists A \in \mathcal{B}$ such that: $A(\gamma) \equiv a \pmod{p}$. Such a polynomial $A$ is called a **representation** of $a$ in $\mathcal{B}$ and we denote $A \equiv a_{\mathcal{B}}$.

Thus, $\mathcal{B}$ designates a PMNS $(p, n, \gamma, \rho, E)$.

The main operations in PMNS are polynomial addition and multiplication. However, additional operations must be done in order to ensure outputs in $\mathcal{B}$. Indeed, if $A, B \in \mathcal{B}$. Then $\|A + B\|_\infty$ and $\|A \times B\|_\infty$ might be greater than $\rho$. Also, $\deg(A \times B)$ is most-likely to be greater than $n$. For the first case, an internal reduction has to be performed and an external reduction for the latter.

*1) External reduction:* Let $C \in \mathbb{Z}[X]$. The *external reduction* consists in computing a polynomial $R \in \mathbb{Z}_{n-1}[X]$ such that $R(\gamma) \equiv C(\gamma) \pmod{p}$, using $E$ of $\mathcal{B}$, as follows:

$$R = C \bmod E\,.$$

Since $E \in \mathbb{Z}_n[X]$ and is monic, $R \in \mathbb{Z}_{n-1}[X]$. Moreover, $E(\gamma) \equiv 0 \pmod{p}$ ensures that $R(\gamma) \equiv C(\gamma) \pmod{p}$. The polynomial $E$ is called the *external reduction polynomial*.

*2) Internal reduction:* Let $R \in \mathbb{Z}_{n-1}[X]$. The *internal reduction* aims to compute a polynomial $S \in \mathcal{B}$ such that $S(\gamma) \equiv C(\gamma) \pmod{p}$. In this paper, we focus on the Montgomery-like approach presented in [12], see Algorithm 2.

It is one of the most efficient internal reduction methods and requires three additional parameters, two polynomials $M, M'$ and an integer $\phi \geqslant 2$ such that:

$$M(\gamma) \equiv 0 \pmod{p} \text{ and } M' = -M^{-1} \bmod (E, \phi)$$

**Algorithm 2** Coefficients reduction (RedCoeff) [12]

```
Require: $\mathcal{B} = (p, n, \gamma, \rho, E)$ a PMNS, $V \in \mathbb{Z}_{n-1}[X]$, $M \in \mathcal{B}$ 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 $S \in \mathbb{Z}_{n-1}[X]$
  $Q \leftarrow V \times M' \bmod (E, \phi)$
  $T \leftarrow Q \times M \bmod E$
  $S \leftarrow (V + T)/\phi$
  return $S$
```

This method is efficient if $\phi$ is a power of two. Also, it induces a multiplicative factor $\phi^{-1}$ on the output. In [28], Didier et al. explain how to deal with this factor. They also present a process to generate efficient PMNS, where $\phi$ can be any power of two.

Algorithm 3 presents the Montgomery-like modular multiplication. RedCoeff refers to Algorithm 2.

**Algorithm 3** Multiplication in PMNS

```
Require: $A \in \mathcal{B}$, $B \in \mathcal{B}$ and $\mathcal{B} = (p, n, \gamma, \rho, E)$
Ensure: $S \in \mathcal{B}$ with $S(\gamma) \equiv A(\gamma)\, B(\gamma)\, \phi^{-1} \pmod{p}$
  $C \leftarrow A.B \mod E$
  $S \leftarrow$ RedCoeff($C$)
  return $S$
```

### *C. Complexities*

In this section, we discuss RNS and PMNS complexities for modular multiplication with the Algorithms 1 and 3. We consider a $w$-bit processor architecture and basic arithmetic computations performed on $w$-bit words. Let $\mathcal{M}$ and $\mathcal{A}$ respectively denote the multiplication and the addition of two $w$-bit integers. Let $\mathcal{L}$ be the notation for logical operations, for instance the AND function, or the right and left shifts on $w$-bit integers.

*1) RNS complexity:* The RNS addition and multiplication complexities are linear, while the base extensions are quadratic. However, software RNS implementations suffer from the cost of the basic $w$-bit modular additions and multiplications which are not provided by the processor instruction set. In Table I, we provide the count of elementary arithmetic operations where $n$ is the number of RNS channels.

*2) PMNS complexity:*
The performances and the required memory storage of a PMNS mainly depend on the target architecture and the value of $n$. We assume the modular multiplication 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 $1 \leqslant t < h \leqslant w$. Since elements in $\mathcal{B}$ are polynomials, $n$ $w$-bit data words are required to represent each of them. Thus, addition is a simple polynomial addition, whose cost is $n\mathcal{A}$.

| Operation | Cost |
|---|---|
| Addition | $n\mathcal{M} + 2n\mathcal{A} + (2n-1) + 3n\mathcal{L}$ |
| Multiplication | $4n\mathcal{M} + 5n\mathcal{A} + 6n\mathcal{L}$ |
| $1^{st}$ Base ext. | $(4n^2 + 5n)\mathcal{M} + (5n^2 + 7n)\mathcal{A} + (6n^2 + 9n)\mathcal{L}$ |
| $2^{nd}$ Base ext. | $(7n^2 + 4n)\mathcal{M} + (9n^2 + 7n)\mathcal{A} + (9n^2 + 10n)\mathcal{L}$ |
| Modular Mult. | $(11n^2 + 30n)\mathcal{M} + (14n^2 + 41n)\mathcal{A}$<br>$+(18n^2 + 52n)\mathcal{L}$ |

TABLE I: RNS operation theoretical cost.

In this paper, we consider sparse external reduction polynomials $E$: $E(X) = X^n - 2$ or $E(X) = X^n - X - 1$. Table II gives the modular multiplication theoretical cost in both cases.

| $E(X)$ | Cost |
|---|---|
| $X^n - 2$ | $3n^2\mathcal{M} + (5n^2 - 3n)\mathcal{A} + (2n-1)\mathcal{L}$ |
| $X^n - X - 1$ | $3n^2\mathcal{M} + (5n^2 - n - 2)\mathcal{A} + n\mathcal{L}$ |

TABLE II: PMNS Modular multiplication theoretical cost, for for two external reduction polynomials, where $\phi = 2^h$.

## III. Some complexity analyses

In this section, we give an estimation of the optimal value of the parameter $n$ for RNS and PMNS. For RNS, we explain the construction of the system in the context of SIMD software implementations. This leads to the determination of the sizes of $p$ we will consider in the sequel.

We also compare PMNS and RNS complexities, given a modulus size.

### *A. Estimation of $n$ for a RNS*

The authors in [5] have provided a bound for the value of $\boldsymbol{M}$ depending of the value of $p$, thus its size, and also the number $n$ of moduli:

$$(n+2)^2 \cdot p < \boldsymbol{M}$$

The compliance of this condition ensures the correctness of the modular multiplication.

### *B. Estimation of $n$ for a PMNS*

From Table II, it appears that the smaller the number of coefficients $n$ is, the better the cost is. When the value of $\phi$ is given, parameter $n$ is computed during the PMNS generation process and cannot be estimated as easily as the number of channels in RNS. In this process, we have to set some PMNS parameters.

In [28], the authors introduce a parameter $\delta$ for the PMNS which corresponds to the desired maximum number of consecutive additions without an internal reduction before a modular multiplication. Our implementation strategy for PMNS requires to set values for $p$, $\delta$ and $\phi$.

It is shown [28] that Algorithm 3 outputs a polynomial $S \in \mathcal{B}$ with inputs $A, B \in \mathbb{Z}_{n-1}[X]$, $\|A\|_\infty, \|B\|_\infty < (\delta+1)\rho$, if:

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

where $E(X) = X^n - \lambda$, with $\lambda \in \mathbb{Z} \setminus \{0\}$.

If $E(X) = X^n - 2$ or $E(X) = X^n - X - 1$, we can bound the coefficients of $C$ computed step 1 in Algorithm 3 from [30, Chapter 2, Proposition 2.1]:

$$\|C\|_\infty \leqslant (2n-1)\|A\|_\infty\|B\|_\infty$$

Thus, the bounds on $\rho$ and $\phi$ presented in [28] become:

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

This leads to:

$$\phi \geqslant 4(\delta+1)^2(2n-1)^2\|M\|_\infty$$

As explained in [28], the parameter $M$ is computed as a short vector of a $n$-dimensional Euclidean lattice whose covolume is equal to $p$. From Minkowski's theorem [31], we can expect to have:

$$\|M\|_\infty \leqslant p^{1/n}$$

This leads to this approximation:

$$\phi \geqslant 4(\delta+1)^2(2n-1)^2p^{1/n} \tag{2}$$

Now, let $p$ be an $l$-bit modulus; that is $p < 2^l$. Thus, we look for the smallest integer $n$ such that:

$$\phi \geqslant 4(\delta+1)^2(2n-1)^2 2^{l/n} \tag{3}$$

So, once $p$, $\delta$ and $\phi$ are given, the approximate best value for $n$ is the smallest integer satisfying Equation 3. It can be noticed that the bigger $n$ is, the smaller the right side of Equation 3 is, since $p$ is supposed to be a large integer.

*Example* 1. Let's set $\phi = 2^{64}$, $\delta = 3$ and a prime $p$ of size 256 bits (i.e. $p < 2^{256}$). Then, Equation 3 becomes:

$$\begin{aligned} 2^{64} &\geqslant 4(3+1)^2(2n-1)^2 2^{256/n} \\ &= 64(4n^2 - 4n + 1)2^{256/n} \end{aligned}$$

Thus:

$$2^{58} \geqslant (4n^2 - 4n + 1)2^{256/n}$$

The minimum value satisfying this equation is $n = 5$.

*Remark* 1. Equation 3 allows to find an estimation of the best value for $n$, based on Minkowski's theorem. In practice, this is not always the smallest value of $n$. Indeed, the lattice reduction used to find $M$ (see [28, Section 5.4]) can compute a polynomial $M$ with an infinity norm small enough to allow better (i.e. smaller) value for $n$. For instance, with $\phi = 2^{64}$, $\delta = 5$ and a prime $p$ of size 401 bits, the minimum value of $n$ satisfying Equation 3 is 9. However, we were able to generate PMNS with $n = 8$. So, the better the lattice reduction is, the smaller the value of $n$ will be. This lattice reduction process depends on $p$, $\gamma$ and the algorithm used (LLL [32] in our case).

### C. *Complexity comparison for software implementation*

In the classic sequential implementation, the word size is 64 bits. In the SIMD case (`AVX512` instruction set) and for both systems, we make use of the fused multiplier-adder `VPMADD52`, which computes simultaneously 8 multiplications-additions of 52-bit operands for the multiplications, providing the results on 64-bit words.

For the RNS system, this leads to the consideration of the values of $n$ whose are multiples of 8. Table III provides the sizes of the modulus $p$ corresponding to the different RNS systems we consider in the SIMD implementation case, using 52-bit words multiplication operands.

| $n$ | 8 | 16 | 24 | 32 |
|---|---|---|---|---|
| Size of $p$ | 401 | 807 | 1 214 | 1 621 |
| $n$ | 40 | 48 | 56 | 64 |
| Size of $p$ | 2 029 | 2 436 | 2 844 | 3 251 |

TABLE III: Estimation of $p$ bit size for $n$ multiple of 8, for RNS SIMD implementations, with 52-bit moduli.

We choose to consider the sizes listed in Table III as reference sizes in our implementations. In the sequel, the value of $n$ for the RNS bases considered with 63-bit moduli is estimated in Table IV.

| Size of $p$ | 401 | 807 | 1 214 | 1 621 |
|---|---|---|---|---|
| $n$ | 7 | 13 | 20 | 26 |
| Size of $p$ | 2 029 | 2 436 | 2 844 | 3 251 |
| $n$ | 33 | 39 | 46 | 52 |

TABLE IV: Estimation of $n$ optimal values for RNS, with 64-bit moduli.

Let's now consider the PMNS case. For the integer sizes determined in Table III, we estimated the optimal value of $n$, for $\phi = 2^{52}$ and $\phi = 2^{64}$, when $\delta = 5$. Table V presents these estimations based on Equation 3.

| $\phi = 2^{64}$ | | | | |
|---|---|---|---|---|
| Size of $p$ | 401 | 807 | 1 214 | 1 621 |
| $n$ | 9 | 18 | 27 | 37 |
| Size of $p$ | 2 029 | 2 436 | 2 844 | 3 251 |
| $n$ | 47 | 57 | 67 | 77 |
| $\phi = 2^{52}$ | | | | |
| Size of $p$ | 401 | 807 | 1 214 | 1 621 |
| $n$ | 12 | 24 | 38 | 52 |
| Size of $p$ | 2 029 | 2 436 | 2 844 | 3 251 |
| $n$ | 66 | 81 | 96 | 112 |

TABLE V: Estimation of $n$ optimal values for PMNS, $\delta = 5$, for $\phi = 2^{64}$ and. $\phi = 2^{52}$

From Tables I and II, it is clear that for a given value of $n$, PMNS is more efficient than RNS. However, for a given modulus size, the number of coefficients for PMNS is greater than the number of RNS channels; see Tables III, IV, V. Thus, for larger $p$, the PMNS complexity will approach that of RNS. We now estimate the threshold above which the PMNS complexity will exceed the RNS one. For this purpose, we consider that all operations except multiplications are equivalent. The cost of one elementary multiplication is considered to be twice that of the other operations, i.e. $\mathcal{M} = 2\mathcal{A}$. This leads to the global cost of the RNS modular multiplication:

$$\mathcal{C}_{RNS} = (36n^2 + 101n)\mathcal{A}$$

where $n$ is the number of channels. Similarly, the cost of the PMNS modular multiplication is $\mathcal{C}_{PMNS} = (11c^2 - c - 2)\mathcal{A}$ where $c$ is the number of coefficients. So, RNS and PMNS performances will meet when:

$$\mathcal{C}_{RNS} \approx \mathcal{C}_{PMNS}, \text{ that is } c \approx 1.81n.$$

A numerical evaluation of $n$ and $c$ gives the following results:

- in case of 52-bit words, this threshold is reached for $p$ of approximately 5 281 bits;
- in case of 64-bit words, this threshold is reached for $p$ of approximately 45 000 bits.

One may notice that the sizes of $p$ in both cases are above the range of our implementation target. Furthermore, the schoolbook hypothesis for the polynomial multiplication complexity evaluation is undoubtedly greatly unfavorable to the PMNS.

## IV. Implementations

In this section, we describe the software implementations for both systems, RNS and PMNS. We target sequential native 64-bit and AVX512 SIMD implementations. The code is written in C and uses intrinsics for the vectorized version. The vector implementation uses `AVX512` instructions that simultaneously compute 8 operations, processed on 512-bit registers. This implementation also uses the FMA instructions which group a multiplication and an addition. Our implementations are available on GitHub[1].

### A. RNS implementation

The elementary operations in RNS are word-length modular operations. Several choices of moduli are possible. In our implementations we chose pseudo-Mersenne numbers whose values are $2^e - c$ with $c < 2^{e/2}$ [33]. The advantage of such moduli is that $c \equiv 2^e \bmod (2^e - c)$. Thus, the modular reduction is a simple multiplication by $c$ of the leftmost bits added to the rightmost bits [34]. The challenge is to obtain the carry of the addition and the rightmost word of a multiplication without conditional statement in order to have the most efficient use of the instruction pipeline.

In our sequential implementation, we target 64-bit operations. We consider 63-bit values in order to easily obtain the carry of an addition. In the target processor, the multiplication can be performed on 64-bit words and provides the result in one 128-bit word. Thus, the rightmost bits can be obtained with simple logical operations.

In our vectorized implementation we make use of the `VPMADD52` instructions which are vectorized, fused multiplier-adders. With these vector instructions, the 52-bit integers are packed in 64-bit elements. The higher and lower 52-bit parts of their 104-bit multiplication product is provided by separate instructions. These AVX512 instructions operate on 8-element vectors.

### B. PMNS implementation

Our sequential PMNS implementations also make use of the 64-bit native multiplier available in the Intel processors that computes a full 128-bit result. In this case, the value of $\phi$ is set to $2^{64}$.

The `VPMADD52` vectorized fused multiplier-adder instructions are also used in the AVX512 versions. As in the RNS case, we take advantage of the SIMD possibilities. In this case, the value of $\phi$ is set to $2^{52}$.

We generated and implemented modular multiplications for PMNS with various modulus $p$ sizes, and we set the parameter $\delta = 5$. This value is, for instance, large enough for elliptic curve scalar multiplication, using the Montgomery Ladder approach described in [35] (Algorithm 9); see [30, Chapter 3, Section 3.4.3] for details. We observed that the number $n$ of coefficients of the PMNS does not grow linearly with the width of $p$ as explained in section III-B, and for the AVX512 versions of sizes above 2 029 bits, due to the necessity of systems with positive coefficients for $M$, the value of $n$ is greater than the one for the corresponding sequential systems. The parameters of the generated PMNS are summarized in table VI. These PMNS were generated using the generator available on GitHub[2]. Our sequential implementations are obtained from the C codes generator available in the same repository. Additionally, we have implemented a C codes generator which takes advantage of AVX512 instruction set extension. It is also available on GitHub[3].

### C. Results

*1) Experimentation procedure:* The measurements were performed on a Dell Inspiron laptop with an 11th Gen Intel `Tiger Lake` processor `i7-1165G7  2.80GHz`. The code has been compiled with `gcc` 10.2.0 and the following options: `-O3 -g -lgmp -mavx512f -mavx512dq -mavx512vl -mavx512ifma`.

The test procedure was as follows:

- the *Turbo-Boost*® is deactivated during the tests;
- 1 000 runs are executed in order to "heat" the cache memory;
- 50 random data sets are generated, and for each data set the minimum of the execution clock cycle numbers over a batch of 1 000 runs is recorded;
- the performance is the average of all these minimums;

The same procedure has been used in order to record the number of instructions. The clock cycle counter is `rdtsc` and the instruction counter is `rdpmc` with the corresponding selection [36].

[1] https://github.com/rns-pmns-arith

[2] https://github.com/arithPMNS/low_memory_efficient_PMNS

[3] https://github.com/rns-pmns-arith/C-code-generators

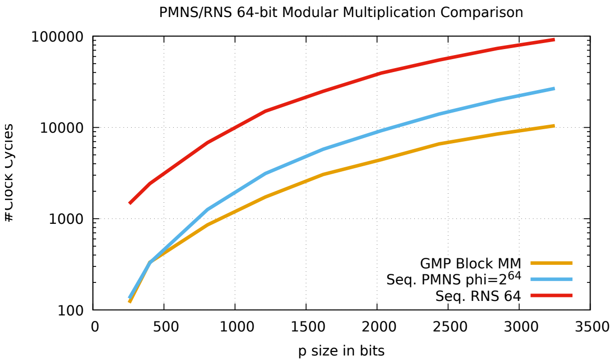

Fig. 1: Timing comparison, PMNS, RNS and GMP block Montgomery modular multiplications, 64-bit words

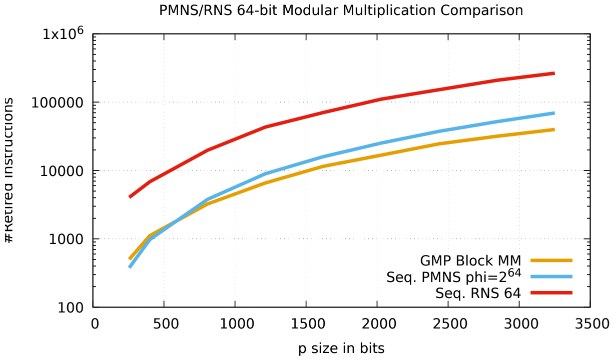

Fig. 2: Retired Instructions per cycle comparison, PMNS, RNS and GMP block Montgomery modular multiplications, 64-bit words

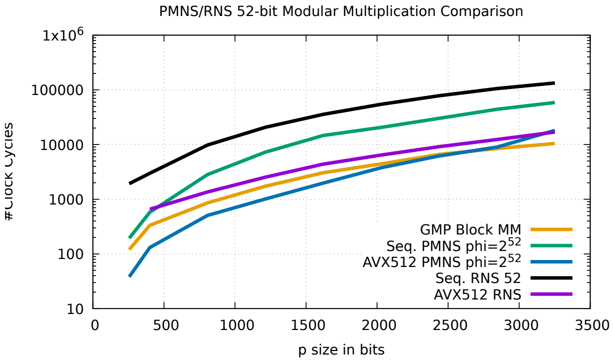

Fig. 3: Timing comparison, PMNS, RNS and GMP block Montgomery modular multiplications, 52-bit words

| Size of $p$ | Implem. | $n$ | $E(X)$ | $\phi$ | $\rho$ |
|---|---|---|---|---|---|
| 256 | Seq. | 5 | $X^5 - 2$ | $2^{64}$ | $2^{54}$ |
| | Seq. | 6 | $X^6 - X - 1$ | $2^{52}$ | $2^{46}$ |
| 401 | Seq. | 8 | $X^8 - X - 1$ | $2^{64}$ | $2^{53}$ |
| | Seq. | 11 | $X^{11} - X - 1$ | $2^{52}$ | $2^{41}$ |
| | AVX512 | 11 | $X^{11} - X - 1$ | $2^{52}$ | $2^{41}$ |
| 807 | Seq. | 17 | $X^{17} - 2$ | $2^{64}$ | $2^{52}$ |
| | Seq. | 25 | $X^{25} - X - 1$ | $2^{52}$ | $2^{40}$ |
| | AVX512 | 25 | $X^{25} - X - 1$ | $2^{52}$ | $2^{40}$ |
| 1 214 | Seq. | 26 | $X^{26} - X - 1$ | $2^{64}$ | $2^{51}$ |
| | Seq. | 40 | $X^{40} - X - 1$ | $2^{52}$ | $2^{40}$ |
| | AVX512 | 40 | $X^{40} - X - 1$ | $2^{52}$ | $2^{40}$ |
| 1 621 | Seq. | 35 | $X^{35} - X - 1$ | $2^{64}$ | $2^{51}$ |
| | Seq. | 56 | $X^{56} - X - 1$ | $2^{52}$ | $2^{39}$ |
| | AVX512 | 56 | $X^{56} - X - 1$ | $2^{52}$ | $2^{39}$ |
| 2 029 | Seq. | 44 | $X^{44} - X - 1$ | $2^{64}$ | $2^{51}$ |
| | Seq. | 65 | $X^{65} - 2$ | $2^{52}$ | $2^{38}$ |
| | AVX512 | 74 | $X^{74} - X - 1$ | $2^{52}$ | $2^{38}$ |
| 2 436 | Seq. | 54 | $X^{54} - X - 1$ | $2^{64}$ | $2^{51}$ |
| | Seq. | 78 | $X^{78} - X - 1$ | $2^{52}$ | $2^{38}$ |
| | AVX512 | 92 | $X^{92} - X - 1$ | $2^{52}$ | $2^{38}$ |
| 2 844 | Seq. | 64 | $X^{64} - X - 1$ | $2^{64}$ | $2^{50}$ |
| | Seq. | 94 | $X^{94} - 2$ | $2^{52}$ | $2^{38}$ |
| | AVX512 | 108 | $X^{108} - X - 1$ | $2^{52}$ | $2^{38}$ |
| 3 251 | Seq. | 74 | $X^{74} - X - 1$ | $2^{64}$ | $2^{50}$ |
| | Seq. | 107 | $X^{107} - X - 1$ | $2^{52}$ | $2^{38}$ |
| | AVX512 | 139 | $X^{139} - X - 1$ | $2^{52}$ | $2^{37}$ |

TABLE VI: Parameters of the implemented PMNS

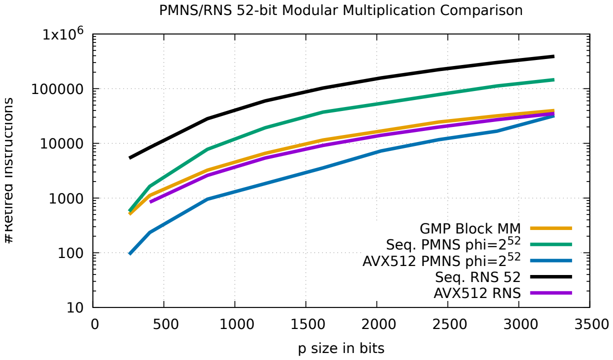

Fig. 4: Retired Instructions per cycle comparison, PMNS, RNS and GMP block Montgomery modular multiplications, 52-bit words

*2) Results:* The performance results are shown in Table VII, and figures 1, 2, 3 and 4. We compared the performance software implementations of RNS, PMNS and the `GMP` library [2] (6.2.0 version) for 256-bit up to 3 251-bit moduli $p$. The results for 256-bit moduli are irrelevant for the SIMD RNS implementation, since the number of channels should be at least 8 moduli. In our PMNS implementation, the polynomial multiplication is the schoolbook method. In a future work, we intend to study other strategies such as Karatsuba approach [37] and multiple word coefficients.

As a general fact, the vectorized implementations dramatically improve the performances of PMNS and RNS operations. The vectorized `AVX512` implementation of the PMNS version is the fastest and has the lowest instruction count. It is more than twice as fast as the state of the art GMP Block Montgomery multiplication and is always faster than both sequential and SIMD RNS implementations, except for the greatest size, that is $p$ of size 3 251 bits. Indeed, for such size, we could not find a PMNS with the parameter $n$ estimated section III-C and we had to increase $n$ dramatically in order to have positive coefficients for $M$ (see section IV-B and table VI). This explains why the AVX512 RNS is slightly better than the corresponding AVX512 PMNS for this size of $p$. Improving our generation process to obtain PMNS suited for AVX512 with smaller value for $n$ remains a challenge we are working on.

The GMP Block Montgomery multiplication is faster than the sequential C implementations of PMNS with $\phi = 2^{64}$, regardless of $n$. However, for the 401-bit primes, the performances are equivalent and the instruction count is smaller by 13 %. The GMP implementation does not seem to take a significant advantage of the SIMD architecture. As a matter of fact, the internal representation of integers is basically a high radix positional representation for which the carry chain has to be maintained. According to the GMP documentation, the library makes limited use of the SIMD instructions except on some specific cases for shift operations [2]. Furthermore, one may notice that the GMP Block Montgomery multiplication is not a constant time implementation, unlike our PMNS/RNS implementations, and might be weak to side channel attacks.

The sequential RNS implementation is the slowest approach, due to the complexity of the base extensions and the cost of the word-size modular arithmetic operations. However, this implementation appears to make the best use of the processor and retires up to 2.8 instructions per cycles.

The vectorized RNS is roughly 1.5 slower than the GMP Block Montgomery multiplication, though the retired instruction counts are 10 to 20 % lower. This is due to the lower Instructions Per Cycle characteristic of the instruction set at the processor micro-architectural level. The SIMD implementations present an acceleration of up to 8.5. This value over 8 is due to some SIMD instructions that have a throughput smaller than 1, and also the efficiency of the fused multiplier-adder VPMADD52.

## V. Conclusion

In this work, we have compared sequential and SIMD implementations of RNS and PMNS modular multiplication, and presented the performance results. The SIMD implementations largely take advantage of the parallel nature of both arithmetics and improves their performances, compared to the sequential cases. PMNS SIMD implementations are faster than GMP implementations, since the underlying arithmetic in GMP is not parallel. RNS implementations suffer from the lack of word-size, modular, arithmetic instructions in the target processor. The presence of such instructions might greatly improve the software performance of this arithmetic. We observed that the degree $n$ of PMNS built for large moduli $p$ grows faster than the number of RNS channels. Thus, for the moduli $p$ up to 1 621 bits, the SIMD PMNS implementation outperforms the GMP library by at least 35% (#clock cycles, $p$ of 1 621 bits). In a future work, we intend to study the impact of the use of multiple-word coefficients and Karatsuba polynomial multiplication, in order to further improve the PMNS implementations.

## References

[1] Nist and al., “NIST publications.” https://www.nist.gov/publications.

[2] T. Granlund and al., “GNU multiple precision arithmetic library 6.1.2.” https://gmplib.org/.

[3] A. Shenoy and R. Kumaresan, “Fast base extension using a redundant modulus in rns,” *IEEE Transactions on Computers*, vol. 38, no. 2, pp. 292–297, 1989.

[4] J.-C. Bajard, L.-S. Didier, and P. Kornerup, “An RNS montgomery modular multiplication algorithm,” *IEEE Transactions on Computers*, vol. 47, no. 7, pp. 766–776, 1998.

[5] J.-C. Bajard and L. Imbert, “A full RNS implementation of RSA,” *IEEE Transactions on Computers*, vol. 53, no. 6, pp. 769–774, 2004.

[6] S. Kawamura, M. Koike, F. Sano, and A. Shimbo, “Cox-rower architecture for fast parallel montgomery multiplication,” in *Advances in Cryptology — EUROCRYPT 2000* (B. Preneel, ed.), (Berlin, Heidelberg), pp. 523–538, Springer Berlin Heidelberg, 2000.

[7] S. Antão, J.-C. Bajard, and L. Sousa, “RNS based elliptic curve point multiplication for massive parallel architectures,” *The Computer Journal*, vol. 55, no. 5, pp. 629–647, 2012.

[8] J.-C. Bajard, J. Eynard, A. Hasan, and V. Zucca, “A full RNS variant of fv like somewhat homomorphic encryption schemes,” in *SAC 2016, Selected Areas in Cryptography, St. John’s, Newfoundland and Labrador, Canada*, 2016.

[9] L.-S. Didier, F.-Y. Dosso, N. El Mrabet, J. Marrez, and P. Véron, “Randomization of Arithmetic over Polynomial Modular Number System,” in *26th IEEE International Symposium on Computer Arithmetic*, vol. 1, (Kyoto, Japan), pp. 199–206, June 2019.

[10] T. Plantard, “Efficient word size modular arithmetic,” *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1506–1518, 2021.

[11] J.-C. Bajard, L. Imbert, and T. Plantard, “Modular number systems: Beyond the mersenne family,” in *Selected Areas in Cryptography, 11th International Workshop, SAC 2004, Waterloo, Canada*, pp. 159–169, 2004.

[12] C. Negre and T. Plantard, “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.

[13] N. El Mrabet and C. Nègre, “Finite field multiplication combining AMNS and DFT approach for pairing cryptography,” in *ACISP*, vol. 5594 of *Lecture Notes in Computer Science*, pp. 422–436, Springer, 2009.

[14] N. El Mrabet and N. Gama, “Efficient multiplication over extension fields,” in *WAIFI*, vol. 7369 of *Lecture Notes in Computer Science*, pp. 136–151, Springer, 2012.

[15] T. Coladon, P. Elbaz-Vincent, and C. Hugounenq, “MPHELL: A fast and robust library with unified and versatile arithmetics for elliptic curves cryptography (extended version),” in *ARITH 2021*, Transactions on Emerging Topics in Computing, (Torino, Italy), June 2021.

[16] J. Courtois, L. Abbas-Turki, and J.-C. Bajard, “Resilience of randomized rns arithmetic with respect to side-channel leaks of cryptographic computation,” *IEEE Transactions on Computers*, vol. 68, no. 12, pp. 1720–1730, 2019.

[17] C. Negre, “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.

[18] K. Bigou and A. Tisserand, “Single base modular multiplication for efficient hardware RNS implementations of ECC,” in *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16*, vol. 9293 of *Lecture Notes in Computer Science*, pp. 123–140, Springer, 2015.

|  |  | **This work** |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
| Modular Multiplication | `GMP` low level Block Montgomery | PMNS $\phi = 2^{64}$ | PMNS $\phi = 2^{52}$ | PMNS $\phi = 2^{52}$ | RNS (moduli size) $2^{63}$ | $2^{52}$ |  |
|  |  | seq. | seq. | `AVX512` | seq. | seq. | `AVX512` |
| **size of $p$ = 256 bits** |  |  |  |  |  |  |  |
|  |  | $n = 5$ | $n = 6$ | $n = 6$ | $n = 5$ | $n = 6$ |  |
| # clock cycles | 120 | 134 | 194 | 38 | 1 456 | 1 920 | - |
| # instructions | 502 | 378 | 571 | 93 | 4 045 | 5 366 | - |
| **size of $p$ = 401 bits** |  |  |  |  |  |  |  |
|  |  | $n = 8$ | $n = 11$ | $n = 11$ | $n = 7$ | $n = 8$ |  |
| # clock cycles | 333 | 329 | 579 | 131 | 2 423 | 2 983 | 655 |
| # instructions | 1 116 | 978 | 1 635 | 234 | 6 864 | 8 401 | 842 |
| **size of $p$ = 807 bits** |  |  |  |  |  |  |  |
|  |  | $n = 17$ | $n = 25$ | $n = 25$ | $n = 13$ | $n = 16$ |  |
| # clock cycles | 857 | 1 259 | 2 825 | 505 | 6 818 | 9 782 | 1 360 |
| # instructions | 3 236 | 3 784 | 7 784 | 953 | 19 758 | 28 255 | 2 598 |
| **size of $p$ = 1 214 bits** |  |  |  |  |  |  |  |
|  |  | $n = 26$ | $n = 40$ | $n = 40$ | $n = 20$ | $n = 24$ |  |
| # clock cycles | 1 728 | 3 126 | 7 267 | 1 009 | 15 050 | 20 663 | 2 526 |
| # instructions | 6 580 | 8 963 | 19 280 | 1 841 | 43 287 | 59 659 | 5 394 |
| **size of $p$ = 1 621 bits** |  |  |  |  |  |  |  |
|  |  | $n = 35$ | $n = 56$ | $n = 56$ | $n = 26$ | $n = 32$ |  |
| # clock cycles | 3 053 | 5 769 | 14 648 | 1 973 | 24 917 | 35 666 | 4 382 |
| # instructions | 11 503 | 15 909 | 37 230 | 3 558 | 70 561 | 102 580 | 9 187 |
| **size of $p$ = 2 029 bits** |  |  |  |  |  |  |  |
|  |  | $n = 44$ | $n = 65$ | $n = 74$ | $n = 33$ | $n = 40$ |  |
| # clock cycles | 4 431 | 9 223 | 20 604 | 3 770 | 39 391 | 54 608 | 6 431 |
| # instructions | 16 801 | 25 200 | 53 804 | 7 264 | 110 735 | 157 128 | 14 021 |
| **size of $p$ = 2 436 bits** |  |  |  |  |  |  |  |
|  |  | $n = 54$ | $n = 78$ | $n = 92$ | $n = 39$ | $n = 48$ |  |
| # clock cycles | 6 602 | 14 003 | 30 130 | 6 172 | 54 746 | 77 972 | 9 123 |
| # instructions | 24 559 | 37 415 | 77 945 | 11 691 | 152 353 | 223 247 | 19 828 |
| **size of $p$ = 2 844 bits** |  |  |  |  |  |  |  |
|  |  | $n = 64$ | $n = 94$ | $n = 108$ | $n = 46$ | $n = 56$ |  |
| # clock cycles | 8 472 | 19 224 | 44 177 | 9 006 | 73 390 | 105 663 | 12 378 |
| # instructions | 31 804 | 51 936 | 112 101 | 16 745 | 209 356 | 300 960 | 27 124 |
| **size of $p$ = 3 251 bits** |  |  |  |  |  |  |  |
|  |  | $n = 74$ | $n = 107$ | $n = 139$ | $n = 52$ | $n = 64$ |  |
| # clock cycles | 10 446 | 26 759 | 58 722 | 18 129 | 92 014 | 133 553 | 16 498 |
| # instructions | 39 847 | 69 387 | 145 766 | 32 106 | 265 233 | 390 242 | 35 170 |

TABLE VII: Performance comparison for modular multiplication, PMNS and RNS, sequential and `AVX512`

[19] K. Bigou and A. Tisserand, "Improving modular inversion in RNS using the plus-minus method," in *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23*, vol. 8086 of *Lecture Notes in Computer Science*, pp. 233–249, Springer, 2013.

[20] A. Chaouch, L. Didier, F. Dosso, N. E. Mrabet, B. Bouallegue, and B. Ouni, "Two hardware implementations for modular multiplication in the AMNS: sequential and semi-parallel," *J. Inf. Secur. Appl.*, vol. 58, p. 102770, 2021.

[21] H. L. Garner, "The residue number system," *IRE Transactions on Electronic Computers*, vol. EL 8, no. 6, p. 140–147, 1959.

[22] Taylor, "Residue arithmetic a tutorial with examples," *Computer*, vol. 17, no. 5, pp. 50–62, 1984.

[23] D. E. Knuth, *Art of computer programming, volume 2: Seminumerical algorithms*. Addison-Wesley Professional, 2014.

[24] N. S. Szabo and R. I. Tanaka, *Residue arithmetic and its applications to computer technology*. New York: McGraw-Hill, 1967.

[25] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, 1985.

[26] K. C. Posch and R. Posch, "Modulo reduction in residue number systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 5, pp. 449–454, 1995.

[27] J.-C. Bajard, L. Imbert, and T. Plantard, "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.
Extended (complete) version available at: https://hal-lirmm.ccsd.cnrs.fr/lirmm-00109201/document.

[28] L.-S. Didier, F. Y. Dosso, and P. Véron, "Efficient modular operations using the Adapted Modular Number System," *Journal of Cryptographic Engineering*, pp. 1–23, 2020.

[29] C. Bouvier and L. Imbert, "An alternative approach for sidh arithmetic," in *Public-Key Cryptography – PKC 2021* (J. A. Garay, ed.), (Cham), pp. 27–44, Springer International Publishing, 2021.

[30] F. Y. Dosso, *Contribution de l'arithmétique des ordinateurs aux implémentations résistantes aux attaques par canaux auxiliaires*. Theses, Université de Toulon, Apr. 2020.

[31] H. Minkowski, *Geometrie der Zahlen*. No. vol. 2 in Geometrie der Zahlen, B.G. Teubner, 1910.

[32] A. K. Lenstra, H. W. Lenstra, and L. Lovász, "Factoring polynomials with rational coefficients," *Mathematische Annalen*, vol. 261, pp. 515–534, Dec 1982.

[33] R. Crandall, "Method and apparatus for public key exchange in a cryptographic system," Oct. 1992. US Patent 5,159,632.

[34] T. Plantard, "Efficient word size modular arithmetic," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1506–1518, 2021.

[35] R. Goundar, M. Joye, A. Miyaji, M. Rivain, and A. Venelli, "Scalar multiplication on weierstraß elliptic curves from co-Z arithmetic," *J. Cryptographic Engineering*, vol. 1, no. 2, pp. 161–176, 2011.

[36] Intel, "Intel 64 and IA-32 architectures software developer's manual combined volumes 1, 2a, 2b, 2c, 2d, 3a, 3b, 3c, 3d, and 4," December 2021.

[37] A. A. Karatsuba and Y. P. Ofman, "Multiplication of many-digital numbers by automatic computers," in *Doklady Akademii Nauk*, vol. 145, pp. 293–294, Russian Academy of Sciences, 1962.