Efficient modular operations using the adapted modular number system

Laurent-Stéphane Didier, Fangan-Yssouf Dosso, Pascal Véron

Noname manuscript No.
(will be inserted by the editor)

Efficient modular operations using the Adapted Modular Number System

Laurent-Stéphane Didier · Fangan-Yssouf Dosso · Pascal Véron

This is a pre-print of an article published in "Journal of Cryptographic Engineering". The final authenticated version is available online at:
https://doi.org/10.1007/s13389-019-00221-7

the date of receipt and acceptance should be inserted later

Abstract The Adapted Modular Number System (AMNS) is an integer number system which aims to speed up arithmetic operations modulo a prime p. Such a system is defined by a tuple (p, n, γ, ρ, E), where p, n, γ and ρ are integers and $E \in \mathbb{Z}[X]$. In [12] conditions required to build AMNS with $E(X) = X^n + 1$ are provided. In this paper, we generalise their approach and provide a method to generate multiple AMNS for a given prime p with $E(X) = X^n - \lambda$ and $\lambda \in \mathbb{Z} \setminus \{0\}$. Moreover, we propose a complete set of algorithms without conditional branching to perform arithmetic and conversion operations in the AMNS, using a Montgomery-like method described in [26]. We show that our implementation outperforms GNU MP and OpenSSL libraries. Finally, we highlight some properties of the AMNS which state that it could lead to a helpful countermeasure against some side channel attacks.

Keywords Modular number system · Modular arithmetic · Side-channel countermeasure

1 Introduction

Efficient implementations of most modern public-key cryptography algorithms rely on the efficiency of the modular arithmetic implementation. Such cryptosystems usually need fast and regular arithmetic modulo integers of size from 160 bits up to several thousand bits. Adapting or building unusual arithmetic for cryptographic purpose may offer significant improvements.

L.-S. Didier, F.-Y. Dosso, P. Véron
Institut de Mathématiques de Toulon
Université de Toulon, France
E-mail: didier@univ-tln.fr
E-mail: dosso@univ-tln.fr
E-mail: veron@univ-tln.fr

For instance, Residue Number Systems [14] is a non positional arithmetic which have parallel properties. This makes them suitable for SIMD architectures [2] and well fitted to many cryptosystems [5,4,2]. This system offers also countermeasures against some side channel attacks. [6,3].

In usual positional number system, a positive integer k is represented in radix γ as follows:

$$k = \sum_{i=0}^{n-1} k_i \gamma^i,$$

where $0 \leqslant k_i < \gamma$. If $k_{n-1} \neq 0$, k is called a n-digit radix-γ number. The radix is often taken as a power of two.

In modular arithmetic, the computations are performed modulo p. This modulus is generally used several times and the radix γ is chosen according to the target architecture. In this context, elements modulo p are represented as polynomials of degree lower than the digits number of p in base γ, with $\gamma \approx p^{1/n}$.

As an example, let us consider $p = 521$ and the radix $\gamma = 2$. Ten digits are required to represent p. Each integer $0 \leqslant a < p$ is represented by a polynomial $A(x)$ such that $\deg(A(x)) < 10$, $\|A\|_\infty < \gamma$ and $A(\gamma) \equiv a \pmod{p}$.

1.1 The Modular Number System

In [7], Bajard et al. introduced the Modular Number System (MNS) as an extension of the positional number system in order to represent integers modulo p. In a MNS, every integer $0 \leqslant x < p$ is represented as a polynomial in γ. The main idea of the MNS consists to relax the condition $\gamma \approx p^{1/n}$ by allowing it to be freely chosen in $\mathbb{Z}/p\mathbb{Z}$.

Definition 1 [7] A modular number system (MNS) $\mathcal{B}$ is defined by a tuple (p, n, γ, ρ), such that for every integer $0 \leqslant x < p$, there exists a vector $V = (v_0, \ldots, v_{n-1})$ such that:

$$x \equiv \sum_{i=0}^{n-1} v_i \gamma^i \pmod{p} ,$$

with $|v_i| < \rho$, $\rho \approx p^{1/n}$ and $0 < \gamma < p$.
In this case, we say that V (or equivalently the polynomial $V(X) = v_0 + v_1 X + \cdots + v_{n-1} X^{n-1}$) is a representative of x in $\mathcal{B}$ and we notate $V \equiv x_{\mathcal{B}}$.

Example 1 Let $p = 19$ and $\mathcal{B} = (19, 3, 7, 2)$ be a MNS. In Table 1, we give a representative in the MNS $\mathcal{B}$ of each element of $\mathbb{Z}/19\mathbb{Z}$.

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

4	5	6	7
$X^2 - X$	$X^2 - X + 1$	$X - 1$	X

8	9	10	11
$X + 1$	$-X^2 + 1$	$X^2 - 1$	X^2

12	13	14	15
$X^2 + 1$	$-X + 1$	$-X^2 + X - 1$	$-X^2 + X$

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

Table 1: The elements of $\mathbb{Z}/19\mathbb{Z}$ in $\mathcal{B} = (19, 3, 7, 2)$

It can be checked in Table 1 that any representative A of an element $a \in \mathbb{Z}/19\mathbb{Z}$ is such that: $\deg(A) < 3$, $\|A\|_\infty < 2$ and $A(\gamma) \equiv a \pmod{p}$. For instance, $\gamma^2 - \gamma + 1 = 49 - 7 + 1 = 43 \equiv 5 \bmod 19$ shows that $X^2 - X + 1$ is a representative of 5 in $\mathcal{B}$.

In MNS, the arithmetic computations are performed on polynomials. Let $V \equiv x_{\mathcal{B}}$ and $W \equiv y_{\mathcal{B}}$ be two MNS numbers. The polynomial $T = VW$ satisfies $T(\gamma) \equiv xy \pmod{p}$. However, T might not be a valid representative of xy in $\mathcal{B}$ because its degree could be greater than or equal to n. To keep the degree bounded by n, the product VW has to be computed modulo a polynomial E such that: $E(\gamma) \equiv 0 \pmod{p}$ and $\deg(E) = n$. This operation is called the *external reduction*. In fact, if $E(\gamma) \equiv 0 \pmod{p}$ and $T = VW \pmod{E}$, then $T(\gamma) \equiv xy \pmod{p}$ and $\deg(T) < n$.

Even if $\deg(T) < n$, T might not be a representative of $xy \pmod{p}$ in $\mathcal{B}$, because its coefficients could be greater than or equal to ρ. In order to retrieve the result in $\mathcal{B}$, a specific primitive called the *internal reduction* has to be applied.

The polynomial $S = V + W$ satisfies $S(\gamma) \equiv (x + y) \pmod{p}$ and $\deg(S) < n$. Again, S might not be a valid representative in $\mathcal{B}$, since its coefficients could be greater than or equal to ρ. So, an internal reduction might be required to retrieve the result in $\mathcal{B}$.

In order to perform the external reduction and the internal reduction efficiently, two subsets of the MNS have been introduced: the Adapted Modular Number system (AMNS) and the Polynomial Modular Number system (PMNS).

1.2 The Polynomial Modular Number System

In [8], Bajard et al. introduce the Polynomial Modular Number System (PMNS). A PMNS is defined by a tuple (p, n, γ, ρ, E) such that: (p, n, γ, ρ) is a MNS and $E(X) = X^n - \alpha X - \lambda$ is an irreducible polynomial in $\mathbb{Z}[X]$, where α and λ are very small integers and γ is a root modulo p of E. The shape of the polynomial E allows to perform the external reduction very efficiently. Bajard et al. show that once the parameters p, n, E and γ are chosen and if ρ is such that $\rho \geqslant (|\alpha| + |\lambda|)p^{1/n}$, then the tuple (p, n, γ, ρ, E) defines a PMNS. Unfortunately, their proof has some issues. It appears that a factor n is missing and that the correct bound should be : $\rho \geqslant n(|\alpha| + |\lambda|)p^{1/n}$.
They also provide a method to build a PMNS for a given prime integer p.

In the same paper, the authors describe how to perform the internal reduction using two methods: one using lookup tables and the other using the Barrett multiplication algorithm [10]. The lookup tables method requires a lot of read operations and the memory storage grows very fast with the value of n and/or p. The Barrett-like method is less memory consuming but requires more computations. For both methods, the authors provide examples, the complexity of their proposals and make some comparisons. But their results do not allow to conclude whether or not PMNS can be an interesting alternative to the usual number system.

1.3 The Adapted Modular Number system

In [7], Bajard et al. introduced the Adapted Modular Number System (AMNS) as a subset of the MNS. In this system, $\gamma^n \equiv \lambda \pmod{p}$ where λ is a very small nonzero integer (for instance, $\lambda = \pm 1, \pm 2$ or ± 3). Here,

the operations are done modulo the polynomial $E(X) = X^n - \lambda$. This choice of E is the best in order to get an efficient external reduction. Notice that $E(\gamma) \equiv 0 \pmod{p}$ since $\gamma^n \equiv \lambda \pmod{p}$. Also, the polynomial E is not required to be irreducible, unlike the PMNS.

In the same paper, the authors propose a method to build AMNS which allows a very efficient internal reduction. However, in this approach, the modulus p cannot be chosen. Its value is computed during the process and one can only choose the modulus size. It makes this class of AMNS irrelevant for some cryptographic standards where the value of p is already known (see for example RFC5903 for IPSEC [30]).

In [26], Negre and Plantard propose a method based on the Montgomery multiplication [25] for the internal reduction in AMNS with a given p. It is more efficient than those proposed in [8]. It is the best known method to perform the internal reduction in AMNS (even in PMNS). Their algorithm (see Algorithm 3) requires two polynomials M and M' such that $M \in \mathcal{B}$, $M(\gamma) \equiv 0 \pmod{p}$ and $M' = -M^{-1} \bmod (E, \phi)$, with $\phi \in \mathbb{N} \setminus \{0\}$. Many reductions modulo ϕ and exact divisions by ϕ are done in this algorithm. So, a significant speed-up can be achieved when ϕ is a power of two.

Negre and Plantard also give a necessary but not sufficient condition to ensure the existence of M'. Moreover, the construction they propose does not guarantee the existence of M' nor that ϕ can be chosen as a power of two. In other words, once p given, there is no proof that one can always build an AMNS which allows to use their Montgomery-like method.

Some applications of AMNS have been published. In [13], El Mrabet and Negre give a new approach for multiplication in $\mathbb{F}_{p^k}$ using the AMNS. It allows to decrease the number of multiplications in $\mathbb{F}_p$ by slightly increasing the number of additions in $\mathbb{F}_p$. With well chosen AMNS, their approach decreases the number of multiplications in $\mathbb{F}_p$ by 50%. For modular operations in $\mathbb{F}_p$ (using the AMNS), they suggest to use the method proposed in [26]. However, they do not address the issue of the existence of the polynomial M'.

In [12], El Mrabet and Gama propose to use AMNS in order to speed-up multiplication over extension fields. Their proposal uses the Montgomery-like method described in [26] for arithmetic operations in the base field associated to the extension field. They consider the case where ϕ is a power of two and $E(X) = X^n + 1$. For the first time, they suggest a new construction of the AMNS parameters that guarantees the existence of the polynomial M'. The construction they propose allows to build systems that make multiplications over extension fields more efficient than some existing solutions. They provide software implementation results and show that their approach is faster than the NTL library [29] implementation of multiplication over extension fields in many cases. In their construction, they distinguish two cases: when the modulus p can be freely chosen and when the modulus p is already known. In the latter case, the proof of their construction has unfortunately some issues.

2 Contributions

Some issues remain in the AMNS construction when the modulus p is known. The first issue is about the internal reduction process using the Montgomery-like method which is the most efficient. So far, there is no correct proof of the existence of the polynomial M' when ϕ is a power of two.

In this article, we show that it is always possible to build many AMNS for any prime p. We present a construction process which ensures the existence of the polynomial M' used in the Montgomery-like method [26], when ϕ is a power of two. Here, we consider the polynomial $E(X) = X^n - \lambda$ with $\lambda \in \mathbb{Z} \setminus \{0\}$. Thus, we generalise the work done in [12]. We give a complete set of algorithms without conditional branching to perform arithmetic and conversion operations in AMNS using the Montgomery-like internal reduction method.

As already mentioned, after an addition, an internal reduction might be required to bring back the result in the MNS. However, all the internal reduction methods available in the literature [7,8,26] are too costly compared to the polynomials addition. For instance, in [26] the proposed reduction costs more than two polynomials multiplications modulo E (see Algorithm 3). In this article, we address this issue by introducing a parameter δ, such that up to δ consecutive additions followed by one multiplication only requires one internal reduction to bring back the final result in the AMNS. The value of δ depends on the target application and has to be chosen during the generation process of the AMNS.

Finally, we provide software implementations and some results which show that AMNS can be a faster alternative to the usual number system. We also discuss some properties of the AMNS that can be very helpful against some side channel attacks.

This paper is organised as follows. We remind the principal definitions and properties of the AMNS in Section 3. In Section 4, we present a complete set of algorithms to perform usual arithmetic, forward and backward conversion operations. We also introduce the parameter δ mentioned above. In Section 5, we show

that an AMNS can always be built for any prime and we present a new generation process including the construction of the polynomials M, M' and all the parameters. In Section 6, we provide the complexity and the memory requirement of the software implementation of the AMNS. We also discuss the number of AMNS that can be built given a prime, along with our software implementation strategy. Additionally, we provide software implementation results for cryptographic sizes integers. We compare our implementations to popular big integer libraries. We also give some analyses about side channel attacks.

3 Adapted Modular Number System

In the sequel of this article, an element A in the AMNS is considered either as a polynomial $A(X)$ of degree $n-1$ such that $A(X) = a_0 + a_1X + \cdots + a_{n-1}X^{n-1}$ or equivalently as a vector $(a_0, a_1, \ldots, a_{n-1})$ depending on the context.

Definition 2 [7] An Adapted Modular Number System (AMNS) is defined by a tuple $\mathcal{B} = (p, n, \gamma, \rho, E)$ such that (p, n, γ, ρ) is a MNS, γ is a root of the polynomial $E(X) = X^n - \lambda$, with λ a very small nonzero integer (for instance, $\lambda = \pm 1, \pm 2$ or ± 3).

Example 2 Let's take the MNS $(19, 3, 7, 2)$ given in Example 1. We have $\gamma^n = 7^3 \equiv 1 \pmod{19}$, which is very small. So, with $E(X) = X^3 - 1$, the tuple $(19, 3, 7, 2, E)$ defines an AMNS.

In the sequel of the article, the tuple $\mathcal{B} = (p, n, \gamma, \rho, E)$ defines the parameters of an AMNS. We denote either the set of parameters or the corresponding AMNS by $\mathcal{B}$. We present the generation process of the AMNS in Section 5.

Proposition 1 gives a necessary condition on ρ for a tuple $\mathcal{B} = (p, n, \gamma, \rho, E)$ to be an AMNS, when p and n are given.

Proposition 1 *If $\mathcal{B}$ is an AMNS, then:*

$$\lceil (\sqrt[n]{p} - 1)/2 \rceil \leqslant \rho\,.$$

Proof. The number of elements in $\mathcal{B}$ is $(2\rho - 1)^n$, as elements may have negative coefficients and their absolute values are lower than ρ (see Definition 1). We want to represent all elements in $\mathbb{Z}/p\mathbb{Z}$, so ρ has to be bounded such that $p \leqslant (2\rho - 1)^n$. □

3.1 Some notations and conventions

We denote by $\mathbb{Z}_n[X]$ the set of polynomials in $\mathbb{Z}[X]$ which degrees are strictly lower than n:

$$\mathbb{Z}_n[X] = \{A \in \mathbb{Z}[X], \text{ such that: } \deg(A) < n\}$$

Let $A \in \mathbb{Z}[X]$, we denote by $A \bmod (E, \phi)$ the polynomial reduction $A \bmod E$ where the coefficients of the result are computed modulo ϕ.

Finally, in the sequel of this paper, we assume that the parameter n of the AMNS is such that $n \geqslant 2$. This should always be the case, otherwise using AMNS does not have any interest.

From Section 1, we know that the arithmetic operations in the MNS require an external reduction and an internal reduction. We remind how these two operations work in the AMNS.

3.2 External reduction

The *external reduction* is a polynomial modular reduction. The purpose of this operation is to keep degree of the AMNS representatives bounded by n. Let $C \in \mathbb{Z}[X]$ be a polynomial. This operation consists in computing a polynomial R such that:

$$\deg(R) < n \text{ and } R(\gamma) \equiv C(\gamma) \pmod{p}$$

The Euclidean division of C by E computes Q and R so that:

$$C = Q \times E + R$$

with $\deg(R) < n$ and $Q \in \mathbb{Z}[X]$. Now, $C(\gamma) = Q(\gamma) \times E(\gamma) + R(\gamma)$. Because $E(\gamma) \equiv 0 \pmod{p}$, then $R(\gamma) \equiv C(\gamma) \pmod{p}$. The external reduction process computes $R = C \bmod E$. The polynomial E is called the *external reduction polynomial*.

Let $A \in \mathbb{Z}_n[X]$ and $B \in \mathbb{Z}_n[X]$. Let $C = AB$ be a polynomial. Then, $\deg(C) < 2n - 1$. When $E(X) = X^n - \lambda$, with λ very small, the external reduction can be done very efficiently. Algorithm 1, proposed by Plantard in [27] (see Algorithm 28, Section 3.2.1), can be used to perform this operation.

3.3 Internal reduction

The aim of the *internal reduction* is to ensure that the coefficients of polynomials are bounded by ρ. Let $C \in \mathbb{Z}_n[X]$ be a polynomial, with $\|C\|_\infty \geq \rho$. This operation

Algorithm 1 External Reduction in AMNS, [27]

Require: $C \in \mathbb{Z}[X]$ with $\deg(C) < 2n-1$ and $E(X) = X^n - \lambda$

Ensure: $R \in \mathbb{Z}[X]$ such that $R = C \bmod E$

1: **for** $i = 0 \dots n-2$ **do**
2: $\quad r_i \leftarrow c_i + \lambda c_{n+i}$
3: **end for**
4: $r_{n-1} \leftarrow c_{n-1}$
5: return R $\quad \# \ R = (r_0, \ \dots, \ r_{n-1})$

consists in computing a polynomial R such that $R \in \mathcal{B}$ and $R(\gamma) \equiv C(\gamma) \pmod p$.

Several implementations of this operation have been provided. For the specific class of AMNS defined in [7], the internal reduction is essentially a vector-matrix multiplication plus a polynomials addition that are repeated as many times as necessary to get the result in the AMNS. The generation process of these AMNS allows to choose a very sparse matrix whose non zero elements are powers of two. This makes the vector-matrix multiplication very fast. However, this generation process does not give control on the value of the modulus but a partial control on its size in bits.

When the value of the modulus is required, the most efficient algorithm to perform the internal reduction in AMNS and PMNS is a Montgomery-like reduction method, which has been published by Negre and Plantard [26]. In their approach, they combine the multiplication operation with their Montgomery-like internal reduction (Algorithm 2). However, they do not deal with the addition which might require an internal reduction in order to keep the result in the AMNS. In this paper, we split the multiplication process from the internal reduction (see Algorithms 4 and 3) because the internal reduction will also be used to convert an integer in the AMNS (and vice versa).

Algorithm 2 Modular Multiplication in AMNS, [26]

Require: $A \in \mathcal{B}$, $B \in \mathcal{B}$ and $\mathcal{B} = (p, n, \gamma, \rho, E)$

Ensure: $S \in \mathcal{B}$ with $S(\gamma) \equiv A(\gamma)B(\gamma)\phi^{-1} \pmod p$

1: $V \leftarrow A \times B \bmod E$
2: $Q \leftarrow V \times M' \bmod (E, \phi)$
3: $T \leftarrow Q \times M \bmod E$
4: $S \leftarrow (V + T)/\phi$
5: return S

4 Arithmetic and conversion operations in AMNS

In this section, we present a set of algorithms to perform addition, multiplication and conversion operations in the AMNS, using the Montgomery-like internal reduction method [26]. All these operations make use of the internal reduction operation.

4.1 The internal reduction with Montgomery

In [26], Negre and Plantard combine the polynomials multiplication modulo E with the internal reduction in one algorithm. Here, we put apart the internal reduction from the multiplication because we use it in other operations. Algorithm 3 corresponds to the internal reduction method embedded in the modular multiplication given in [26].

Algorithm 3 RedCoeff (Coefficient reduction), [26]

Require: $\mathcal{B} = (p, n, \gamma, \rho, E)$, $V \in \mathbb{Z}_n[X]$, $M \in \mathcal{B}$ such that $M(\gamma) \equiv 0 \pmod p$, $\phi \in \mathbb{N} \setminus \{0\}$ and $M' = -M^{-1} \bmod (E, \phi)$.

Ensure: $S(\gamma) = V(\gamma)\phi^{-1} \pmod p$

1: $Q \leftarrow V \times M' \bmod (E, \phi)$
2: $T \leftarrow Q \times M \bmod E$
3: $S \leftarrow (V + T)/\phi$
4: return S

Similarly to the Montgomery modular reduction method, Algorithm 3 outputs S such that $S(\gamma) \equiv V(\gamma)/\phi \pmod p$, because $M(\gamma) \equiv 0 \pmod p$ and $E(\gamma) \equiv 0 \pmod p$ imply $T(\gamma) \equiv 0 \pmod p$. We show in Section 4.4 how to deal with this multiplicative constant.

Algorithm 3 requires two polynomials multiplications (one modulo E and another modulo (E, ϕ)), one polynomials addition and n integer divisions by ϕ (at line 3). These divisions are proven to be exact in [26]. Usually, ϕ is a power of two in order to simplify the reductions modulo ϕ (line 1) and divisions by ϕ (line 3). The polynomials M and M' have to be carefully built. In Section 5.4, we present a generation process of M which ensures the existence of M', when ϕ is a power of two.

Theorem 1 (Theorem 1 in [26]) gives sufficient conditions on ρ and ϕ so that the output S of Algorithm 2 remains in $\mathcal{B}$.

Theorem 1 *[26] Let A and $B \in \mathcal{B}$. If ρ and ϕ are such that:*

$$\rho \geqslant 2n|\lambda|\|M\|_\infty \quad \textit{and} \quad \phi \geqslant 2n|\lambda|\rho$$

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

As a consequence, we have:

Corollary 1 *Let $V \in \mathbb{Z}_n[X]$ be a polynomial. If ρ, ϕ and V are such that:*

$$\|V\|_\infty < n|\lambda|\rho^2, \ \rho \geqslant 2n|\lambda|\|M\|_\infty \quad \textit{and} \quad \phi \geqslant 2n|\lambda|\rho$$

then, the output S of the Algorithm 3 is such that $\|S\|_\infty < \rho$ (i.e., $S \in \mathcal{B}$).

Proof. With the polynomials A and B of Theorem 1 as inputs, the polynomial V (at line 1 of Algorithm 2) satisfies $\|V\|_\infty < n|\lambda|\rho^2$ as shown in Equation 5, Section 3 of [7]. Hence, Theorem 1 can be applied to conclude. □

4.2 Multiplication

The multiplication in AMNS is a polynomials multiplication followed by an external reduction and then by an internal reduction. Algorithm 4 implements the multiplication that is identical to the one proposed in [26], except that the inputs are considered to be in $\mathbb{Z}_n[X]$. This extension on the input domain will be helpful in the sequel to show how to postpone the internal reduction process after a succession of additions.

Algorithm 4 Multiplication in AMNS

Require: $A, B \in \mathbb{Z}_n[X]$ and $\mathcal{B} = (p,\, n,\, \gamma,\, \rho, E)$

Ensure: $S \in \mathbb{Z}_n[X]$ such that $S(\gamma) \equiv A(\gamma)B(\gamma)\phi^{-1} \pmod p$

1: $R \leftarrow A \times B \bmod E$

2: $S \leftarrow \text{RedCoeff}(R)$

3: return S

In Algorithm 4, we have $R(\gamma) \equiv A(\gamma)B(\gamma) \pmod p$, because $E(\gamma) \equiv 0 \,(\text{mod}\, p)$. Thus, this algorithm outputs a polynomial S such that $S(\gamma) \equiv A(\gamma)B(\gamma)\phi^{-1} \pmod p$. As shown in [26], the result S is an AMNS element (i.e., $\|S\|_\infty < \rho$), if the requirements of Theorem 1 are met.

4.3 Addition

Elements being polynomials in AMNS, an addition in $\mathcal{B}$ is a simple polynomials addition (see Section 5.1 of [8]).

Algorithm 5 Addition in AMNS

Require: $A \in \mathcal{B}$, $B \in \mathcal{B}$ and $\mathcal{B} = (p,\, n,\, \gamma,\, \rho, E)$

Ensure: $S = A + B$

1: $S \leftarrow A + B$

2: return S

The output S of Algorithm 5 is such that $\|S\|_\infty < 2\rho$ and might not be in $\mathcal{B}$. In [27] (see Algorithm 29, Section 3.2.2), Plantard proposed a polynomials addition followed by an internal reduction. However, it is not efficient to perform an internal reduction after each addition, because all the internal reduction methods available in the literature [7,8,26] are too costly compared to the polynomials addition. We propose here a solution that allows to perform as much additions as we want without an internal reduction before a modular multiplication.

We denote δ the maximum number of consecutive additions of elements in $\mathcal{B}$ that have to be done before a modular multiplication. This value relies on the target application and is usually known. For instance, in elliptic curve cryptography, the formulas for point addition and doubling are known; so, δ can easily be estimated.

The following proposition gives new bounds on ρ and ϕ that ensure that a multiplication can be performed with Algorithm 4, with inputs being the results of δ consecutive additions of elements in $\mathcal{B}$.

Proposition 2 *Let δ be the maximum number of consecutive additions of elements in $\mathcal{B}$ that have to be done before a modular multiplication. Let U and W be the results of such consecutive additions, using Algorithm 5. If ρ and ϕ are such that:*

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

then, with U and W as inputs, Algorithm 4 outputs a polynomial S such that $\|S\|_\infty < \rho$ (i.e., $S \in \mathcal{B}$).

Proof. We have: $\|U\|_\infty < (\delta+1)\rho$ and $\|W\|_\infty < (\delta+1)\rho$. We also have $\rho \geqslant 2n|\lambda|\|M\|_\infty$. So, as proven in [26] (see proof of Theorem 1), the output S of Algorithm 4 is such that: $\|S\|_\infty < \frac{n|\lambda|(\delta+1)^2\rho^2}{\phi} + \frac{\rho}{2}$.
In order to have $\|S\|_\infty < \rho$, it suffices to take ϕ such that $\phi \geqslant 2n|\lambda|\rho(\delta+1)^2$. □

One may observe that, for $\delta = 0$, Proposition 2 and Theorem 1 in [26] are the same. Our approach does not affect the bound on ρ. So, there is no additional memory cost.

Corollary 2 is a generalization of Corollary 1 which gives conditions on ρ and ϕ to ensure that RedCoeff outputs a result in $\mathcal{B}$.

Corollary 2 *Let $V \in \mathbb{Z}_n[X]$ be a polynomial. If ρ, ϕ and V are such that:*

$$\|V\|_\infty < n|\lambda|(\delta+1)^2\rho^2,$$
$$\rho \geqslant 2n|\lambda|\|M\|_\infty,$$
$$\phi \geqslant 2n|\lambda|\rho(\delta+1)^2,$$

then, the output S of the Algorithm 3 (with V as input) is such that $\|S\|_\infty < \rho$ (i.e., $S \in \mathcal{B}$).

Proof. With the polynomials U and W of Proposition 2 as inputs, the polynomial R (at line 1 of Algorithm 4) satisfies $\|R\|_\infty < n|\lambda|(\delta+1)^2\rho^2$ as shown in Section 3 of [7]. Hence, Proposition 2 can be applied to conclude. □

From Corollary 2, we deduce that the result of up to $n|\lambda|(\delta+1)^2\rho - 1$ consecutive additions of elements in $\mathcal{B}$ is brought back into the AMNS with one call to RedCoeff. It means that if no modular multiplication is done, many polynomials additions can be done before performing an internal reduction.

4.4 Conversion operations

The modular multiplication method (Algorithm 4) presented in Section 4.2 uses the internal reduction method RedCoeff (Algorithm 3) which introduces a multiplicative coefficient ϕ^{-1} in the result. As a consequence, the computation of the product $\alpha_1\alpha_2\cdots\alpha_k$, with $\alpha_i \in \mathbb{Z}/p\mathbb{Z}$, using their representatives in $\mathcal{B}$, outputs a polynomial S such that $S(\gamma) \equiv \phi^{-k}\prod_{i=1}^{k}\alpha_i \pmod p$.

A common solution to deal with this multiplicative coefficient is to keep the values α_i in the Montgomery domain. In this domain, any element $a \in \mathbb{Z}/p\mathbb{Z}$ is replaced by $a\phi \pmod p$ and has a representative $A \in \mathcal{B}$ such that $A(\gamma) \equiv a\phi \pmod p$. Thus, if A, $B \in \mathcal{B}$ are the representatives in the Montgomery domain of a, $b \in \mathbb{Z}/p\mathbb{Z}$, then the internal reduction procedure RedCoeff, applied to $V = A \times B \bmod E$, outputs a polynomial $S \in \mathcal{B}$, such that $S(\gamma) \equiv ab\phi \pmod p$, which is also in the Montgomery domain.
So, in order to ensure that the modular multiplication in the AMNS (Algorithm 4) is consistent, we will keep the values in the Montgomery domain.

In the sequel of this section, we assume that:
$\rho \geqslant 2n|\lambda|\|M\|_\infty$ and $\phi \geqslant 2n|\lambda|\rho(\delta+1)^2$ and $n \geqslant 2$.

4.4.1 Conversion from binary representation to AMNS.

The idea is to use, in the target AMNS, the radix-ρ decomposition of the integer to be converted. As already explained, we need to keep the intermediate values in the Montgomery domain. For this method (Algorithm 6), we must precompute the exact representatives $P_i(X)$ of $(\rho^i\phi^2)$ in $\mathcal{B}$ (and not the representatives in the Montgomery domain). We will later explain how to do it. This algorithm is identical to Algorithm 30 in [27] (Section 3.2.3) except that the output and the polynomials $P_i(X)$ are different here, because it is required to work in the Montgomery domain.

Algorithm 6 Conversion from classical representation to AMNS

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

At line 2 of Algorithm 6, U is a representative of $a\phi^2$ and $\|U\|_\infty < n\rho^2$. Hence, from Proposition 2, $\|A\|_\infty < \rho$ and $A \equiv (a\phi)_\mathcal{B}$.

The complexity of the radix-ρ decomposition of a at line 1 becomes very low if ρ is a power of two. It is always possible to make this choice. For instance, one can take $\rho = 2^{\lceil \log_2(2n|\lambda|\|M\|_\infty)\rceil}$, as one only needs $\rho \geqslant 2n|\lambda|\|M\|_\infty$. The operation at line 2 requires n^2 multiplications of integers lower than ρ and $n(n-1)$ additions of integers lower than ρ^2. This cost is less than the cost of the multiplication $A \times B \bmod E$, with $A, B \in \mathcal{B}$, see Table 2. Thus, Algorithm 6 is faster than a modular multiplication in AMNS (Algorithm 4).

Computation of the representatives P_i

Algorithm 6 requires the representatives $P_i(X)$ of $(\rho^i\phi^2)$ in $\mathcal{B}$. In order to compute them, we need Proposition 3 which allows to quantify the effect of the internal reduction algorithm (cf. Algorithm 3) on the coefficients of a polynomial.

Proposition 3 *Let $V \in \mathbb{Z}_n[X]$ be a polynomial. If ρ, ϕ are such that:*

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

then, one call to RedCoeff (Algorithm 3) divides the coefficients of V by at least 2ρ.

Proof. We remind the assumption $n \geqslant 2$ (see Section 3.1). Let $S = \text{RedCoeff}(V)$, i.e., S is the output of Algorithm 3 with V as input.
We have $\|S\|_\infty \leqslant (\|V\|_\infty + \|T\|_\infty)/\phi$.
Since $T = QM \bmod E$, then from [7] (see Section 3), $\|T\|_\infty < n|\lambda|\|Q\|_\infty\|M\|_\infty < n|\lambda|\phi\|M\|_\infty$, because $\|Q\|_\infty < \phi$. As a consequence,

$$\|S\|_\infty < (\|V\|_\infty + n\phi|\lambda|\|M\|_\infty)/\phi\,.$$

We have $\phi \geqslant 2n|\lambda|\rho(\delta+1)^2 \geqslant 4\rho$ (because $n \geqslant 2$) and $\rho \geqslant 2n|\lambda|\|M\|_\infty$. It follows that:

$$\|S\|_\infty < \frac{\|V\|_\infty}{4\rho} + \frac{\rho}{2}\,.$$

So, if $\|V\|_\infty < 2\rho^2$, then $\|S\|_\infty < \rho$ (i.e., $S \in \mathcal{B}$).
But, if $\|V\|_\infty \geqslant 2\rho^2$, then $\|S\|_\infty < \frac{\|V\|_\infty}{2\rho}$. This means that one call to RedCoeff divides the coefficients of V by at least 2ρ, if $\|V\|_\infty \geqslant 2\rho^2$. □

From Proposition 3, we know that one call to RedCoeff method, with an input V, outputs a polynomial S such that:

- $\|S\|_\infty < \frac{\|V\|_\infty}{2\rho}$, if $\|V\|_\infty \geqslant 2\rho^2$
- $\|S\|_\infty < \rho$ (i.e: $S \in \mathcal{B}$), if $\|V\|_\infty < 2\rho^2$

Also, we have $S(\gamma) \equiv V(\gamma)\phi^{-1} \pmod p$.

Let $\tau = \phi^n \pmod p$. Algorithm 7, based on Proposition 3, describes an iterative way to compute the representatives of $\rho^i\phi^2$. This idea of iterative conversion was already mentioned in [7] (Section 6.1) and also in [27] (Section 3.2.3).

Algorithm 7 Exact conversion from binary to AMNS

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

At line 2 (in Algorithm 7), A is a polynomial of degree 0 which constant coefficient is strictly less than p. From Proposition 1, we know that $p \leqslant (2\rho-1)^n$. Thus, $p < (2\rho)^n$. As a consequence, calling n times RedCoeff on A ensures that the algorithm outputs A in $\mathcal{B}$, according to Proposition 3. Hence, if we apply this algorithm to $a = \rho^i\phi^2$, the output is its representative $P_i(X)$.

Algorithm 7 uses one modular multiplication in $\mathbb{Z}/p\mathbb{Z}$ plus n calls to RedCoeff. So, this algorithm is not suitable to convert the elements of $\mathbb{Z}/p\mathbb{Z}$ in $\mathcal{B}$. Also, it is a lot more costly than Algorithm 6. We only use it to precompute the representatives $P_i(X)$.

There is a faster way to compute the polynomials $P_i(X)$, using Algorithm 7. First, compute the polynomials $\Phi \equiv (\rho\phi)_{\mathcal{B}}$ and $P_0 \equiv (\phi^2)_{\mathcal{B}}$, using Algorithm 7. Then, for $1 \leqslant i < n$, P_i is computed by multiplying P_{i-1} by Φ, using Algorithm 4.
This second approach requires $3n-1$ calls to RedCoeff and two modular multiplications in $\mathbb{Z}/p\mathbb{Z}$ while the first one needs n^2 calls to RedCoeff and n modular multiplications in $\mathbb{Z}/p\mathbb{Z}$.

4.4.2 Conversion from AMNS to binary representation.

Most of the algorithms in this paper make use of RedCoeff procedure. We already explained that one has to keep the intermediate values in the Montgomery domain. An element $a \in \mathbb{Z}/p\mathbb{Z}$ is represented in $\mathcal{B}$ by a polynomial A such that $A(\gamma) \equiv (a\phi) \pmod p$. For the sake of consistency, we consider that the values we want to convert are in this domain.

We present here a slight modification of two algorithms given in [27] that compute the integer value corresponding to an element in $\mathbb{Z}_n[X]$.

Method 1

An easy way to perform this operation is to use the classical Horner's scheme [18]. Algorithm 8 is a slight modification of Algorithm 31 in [27] (Section 3.2.3). Compared to that algorithm, we add line 1 to convert the input from the Montgomery domain. So, it has an additional cost of one call to RedCoeff method.

Method 2

Algorithm 8 is straightforward and does not require any precomputed data. However, it requires $n-1$ modular multiplications and one call to RedCoeff. Algorithm 9 improves Algorithm 8, at the cost of some precomputations and data storage. The values $g_i = \gamma^i \pmod p$, for $i = 1, \ldots, n-1$ have to be precomputed before

Algorithm 8 Conversion from AMNS to classical representation

Require: $A \in \mathbb{Z}_n[X]$ and $\mathcal{B} = (p,\, n,\, \gamma,\, \rho, E)$

Ensure: $a = A(\gamma)\phi^{-1} \pmod p$

1: $A \leftarrow \text{RedCoeff}(A)$
2: $a \leftarrow a_{n-1}$
3: **for** $i = n-2 \ldots 0$ **do**
4: $\quad a \leftarrow (a\gamma + a_i) \pmod p$
5: **end for**
6: return a

using it. This algorithm is a slight modification of Algorithm 32 in [27] (Section 3.2.3), to take account of the Montgomery domain. Compared to that algorithm, we add line 1 to convert the input from the Montgomery domain. So, it has an additional cost of one call to RedCoeff method.

Algorithm 9 Conversion from AMNS to classical representation

Require: $A \in \mathbb{Z}_n[X]$, $\mathcal{B} = (p,\, n,\, \gamma,\, \rho, E)$ and $g_i \equiv \gamma^i \pmod p$, for $i = 1, \ldots, n-1$

Ensure: $a = A(\gamma)\phi^{-1} \pmod p$

1: $A \leftarrow \text{RedCoeff}(A)$
2: $a \leftarrow a_0$
3: **for** $i = 1 \ldots n-1$ **do**
4: $\quad a \leftarrow a + a_i g_i$
5: **end for**
6: $a \leftarrow a \pmod p$
7: return a

In this algorithm, $n-1$ multiplications and one modular reduction are done (plus one call to RedCoeff). If $A \in \mathcal{B}$, then $|a_i| < \rho$. Thus, the multiplications $a_i g_i$ are not full multiplications in $\mathbb{Z}/p\mathbb{Z}$. At the end of the loop (before line 6), we have $|a| < n\rho p$. As $\rho \approx p^{1/n}$, the modular reduction at line 6 is less expensive than a full modular reduction in $\mathbb{Z}/p\mathbb{Z}$.

4.5 Exact coefficients reduction

In Section 4.4, we explained that to keep elements in the Montgomery domain, a representative $A \in \mathcal{B}$ of an element $a \in \mathbb{Z}/p\mathbb{Z}$ is such that $A(\gamma) \equiv a\phi \pmod p$.

Once in the Montgomery domain, another issue may occur when one needs to compute $\alpha_1+\alpha_2+\cdots+\alpha_k$, with $\alpha_i \in \mathbb{Z}/p\mathbb{Z}$, using their representatives in $\mathcal{B}$. The corresponding result is a polynomial R such that $R(\gamma) \equiv \phi \sum_{i=1}^{k} \alpha_i \pmod p$.
If $k \leqslant (\delta+1)$, then, as shown with Proposition 2, there is no need for an internal reduction.
However, if $k > (\delta+1)$ and one needs (or wants) to bring back the result R in the AMNS (for storage requirement, for instance), then an internal reduction has to be done. RedCoeff method (Algorithm 3) introduces a multiplicative constant ϕ^{-1}. So, applying it on R will output a polynomial S such that $S(\gamma) \equiv R(\gamma)\phi^{-1} \equiv \sum_{i=1}^{k} \alpha_i \pmod p$. Hence, S is no more in the Montgomery domain. To solve this issue, we propose Algorithm 10, which takes as input a polynomial V and outputs a polynomial S such that $S(\gamma) \equiv V(\gamma) \pmod p$. So, if V is in the Montgomery domain, then S will also be in this domain.

Algorithm 10 ExactRedCoeff

Require: $V \in \mathbb{Z}_n[X]$, $P_0 \equiv (\phi^2)_{\mathcal{B}}$ and $\mathcal{B} = (p,\, n,\, \gamma,\, \rho, E)$

Ensure: $S(\gamma) \equiv V(\gamma) \pmod p$

1: $T \leftarrow \text{RedCoeff}(V)$
2: $U \leftarrow T \times P_0 \bmod E$
3: $S \leftarrow \text{RedCoeff}(U)$
4: return S

At line 1 of Algorithm 10, we have $T(\gamma) \equiv V(\gamma)\phi^{-1} \pmod p$. Since $P_0(\gamma) \equiv \phi^2 \pmod p$, we have $U(\gamma) \equiv V(\gamma)\phi \pmod p$. Thus, $S(\gamma) \equiv V(\gamma) \pmod p$.

ExactRedCoeff (Algorithm 10) is more expensive than the modular multiplication (Algorithm 4); an additional call to RedCoeff method is done. Also, it uses P_0, an exact representative of ϕ^2 in the AMNS, i.e., $P_0(\gamma) \equiv \phi^2 \pmod p$. In Section 4.4.1, we have explained how to compute P_0 because it is also needed for (fast) conversion in the AMNS.

Proposition 4 gives the requirements on ρ, ϕ and the input V for the output S of Algorithm 10 to be in $\mathcal{B}$.

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

$$\|V\|_\infty < n|\lambda|(\delta+1)^2\rho^2,$$
$$\rho \geqslant 2n|\lambda|\|M\|_\infty,$$
$$\phi \geqslant 2n|\lambda|\rho(\delta+1)^2,$$

then, the output S of the Algorithm 10 (with V as input) is such that $\|S\|_\infty < \rho$ (i.e., $S \in \mathcal{B}$).

Proof. From Corollary 2, we have: $\|T\|_\infty < \rho$ (at line 1 of Algorithm 10). Then, line 2 and line 3 combined are equivalent to a modular multiplication, with inputs in $\mathcal{B}$. So, Proposition 2 suffices to conclude that $S \in \mathcal{B}$. □

Let $\omega = n|\lambda|(\delta+1)^2\rho - 1$. From Proposition 4, one can deduce that if a polynomial V is the result of up to ω consecutive additions of elements in $\mathcal{B}$, then one call to ExactRedCoeff is enough to bring back V in $\mathcal{B}$, with the result still in the Montgomery domain.

To sum up, let $\mu \in]\delta, \omega]$, ExactRedCoeff is meant to be used when μ consecutive additions of elements in $\mathcal{B}$ has been done and the result has to be brought back into the AMNS.

Until now, we have defined the AMNS and given its essential properties in Section 3. In Section 4, we gave a complete set of algorithms to perform conversion and main arithmetic operations in the AMNS, along with some additional properties. This allows us to present all the parameters required for these algorithms and also to give the constraints these parameters must respect. In the following section, we show how to generate all the required parameters.

5 AMNS parameter generation

We remind here the complete set of parameters of the AMNS $\mathcal{B}$ that are used in all the algorithms described in Section 4:

- p: a prime integer, $p \geqslant 3$.
- n: the number of coefficients of the elements in AMNS, $n \geqslant 2$.
- λ: a small nonzero integer.
- γ: a nth-root (modulo p) of λ.
- E: the external reduction polynomial, defined as: $E(X) = X^n - \lambda$.
- M: the internal reduction polynomial.
- ρ: the upper-bound on the infinity norm of the elements of $\mathcal{B}$, $\rho \geqslant 2n|\lambda|\|M\|_\infty$.
- δ: the maximum number of consecutive additions that will be done before a modular multiplication, as defined in Section 4.3.
- ϕ: the integer used in RedCoeff, $\phi \geqslant 2n|\lambda|\rho(\delta+1)^2$.
- M': a polynomial such that $M' = -M^{-1} \bmod (E, \phi)$.

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

5.1 Parameter generation process

The parameters δ and the prime p are chosen with regard to the target application. The parameter δ is the maximum number of consecutive additions that need to be done before a modular multiplication. The next step is to choose the parameter n with regard to the target architecture. Let's consider that we have a k-bit processor architecture. Then n must be chosen such that $nk \geqslant \lceil \log_2(p) \rceil$ in order to ensure that each coefficient of an AMNS element can fit in one data word.

After the choice of n, we choose a small integer $\lambda \in \mathbb{Z} \setminus \{0\}$ such that a nth-root γ modulo p of λ exists. In Section 5.2, we show that it is always possible to find such λ and γ, given p and n. Then, we set the external reduction polynomial $E(X) = X^n - \lambda$.

The internal reduction polynomial M has to be generated while ensuring the existence of the polynomial $M' = -M^{-1} \bmod (E, \phi)$, for any ϕ being a power of two. In Section 5.3, we give the requirements on the polynomial M such that M' exists. In Section 5.4, we explain how to generate the polynomial M while fulfilling these requirements.

After the computation of M, we compute ρ and ϕ as:

$$\rho = 2^{\lceil \log_2(2n|\lambda|\|M\|_\infty) \rceil},$$
$$\phi = 2^{\lceil \log_2(2n|\lambda|\rho(\delta+1)^2) \rceil},$$

to satisfy the requirements of Section 4.

Next, M' is computed as follows:

$$M' = -M^{-1} \bmod (E, \phi).$$

Additionally, for the (fast) conversion method (Algorithm 6), we need to precompute representatives $P_i(X)$ of $(\rho^i \phi^2)$ in $\mathcal{B}$ (as explained in Section 4.4.1). We remind that P_0 is also required for ExactRedCoeff (Algorithm 10). If fast conversion from AMNS to classical representation is wanted, elements $g_i = \gamma^i \pmod p$, for $i = 1, \ldots, n-1$, have to be precomputed in order to use Algorithm 9.

For software implementation, another strategy regarding the choice of ϕ can lead to a better efficiency. Assuming that we have a k-bit processor architecture, a good idea is to take $\phi = 2^k$. With this choice, division and modular reduction by ϕ can be done with very simple mask and shift operations. This choice also makes the multiplication in $\mathbb{Z}/\phi\mathbb{Z}$ (line 1, Algorithm 3) very efficient. Additionally, since $\rho < \phi$, this choice implies that $n \geqslant \lfloor \frac{\log_2 p}{k} \rfloor + 1$. It also ensures that each coefficient of an element in $\mathcal{B}$ fits in one data word. Which is also good for efficiency.

Table 2 (Section 6.1) shows that the value of n has a large impact on the efficiency of the arithmetic operations in the AMNS. Thus, for $\phi = 2^k$, the optimal value for n is $\lfloor \frac{\log_2 p}{k} \rfloor + 1$. The parameter ρ is computed as follows: $\rho = 2^{\lceil \log_2(2n|\lambda|\|M\|_\infty) \rceil}$. So, in order to guarantee that we can take $\phi = 2^k$ with $\phi \geqslant 2n|\lambda|\rho(\delta+1)^2$, we need to compute the polynomial M with $\|M\|_\infty$ small enough to allow that. In section 5.4, we explain how this can be accomplished, thanks to lattice reduction.

5.2 Existence of γ

The first constraint in the generation of an AMNS is to ensure the existence and the computation of γ, a nth-root modulo p of λ. Here, we give some results that show that, for a given p and n, it is always possible to find a small $\lambda \in \mathbb{Z} \setminus \{0\}$ such that γ exists.

Proposition 5 *Let $E(X) = X^n - \lambda$, for $\lambda \in \mathbb{Z} \setminus \{0\}$. Let g be a generator of $(\mathbb{Z}/p\mathbb{Z}) \setminus \{0\}$ and y such that $g^y \equiv \lambda \bmod p$. If $\gcd(n, p-1) \mid y$, then there exists $\gcd(n, p-1)$ roots γ of $E(X)$ in $\mathbb{Z}/p\mathbb{Z}$.*

Proof. If $\gcd(n, p-1) \mid y$, the equation $nx \equiv y \pmod{p-1}$ admits k solutions, where $k = \gcd(n, p-1)$. Let x_0 be one of these solutions, and let consider $\gamma \equiv g^{x_0} \pmod p$. Then $\gamma^n \equiv \lambda \pmod p$. □

Proposition 5 gives a requirement on the existence of the nth-roots modulo p of λ and their number. This proposition relies on the discrete logarithm of λ in $\mathbb{Z}/p\mathbb{Z}$. It requires y such that $g^y \equiv \lambda \bmod p$. The computation of such y can be very hard if p is large enough.

We give below sufficient (but not necessary) conditions that are easy to verify and that guarantee the existence of a nth-root modulo p of λ, taking eventually $\lambda = 1$.

Corollary 3 *If $\gcd(n, p-1) = 1$ then there exists a unique nth-root γ of λ in $\mathbb{Z}/p\mathbb{Z}$, for any $\lambda \in \mathbb{Z} \setminus \{0\}$.*

Proof. If $\gcd(n, p-1) = 1$, this nth-root can be easily computed. In fact, using the extended euclidean algorithm, we compute Bezout coefficients for $(n, p-1)$. That is, u and v in $\mathbb{Z}$ such that $nu + (p-1)v = 1$. So, $\lambda = \lambda^{nu+(p-1)v} = (\lambda^u)^n(\lambda^{p-1})^v$. As, $\lambda^{p-1} \equiv 1 \pmod p$, it is obvious that $\lambda \equiv (\lambda^u)^n \pmod p$. Which means that $\lambda^u \pmod p$ is a nth-root modulo p of λ. □

If $\gcd(n, p-1) = 1$ and $\lambda = 1$, then, using Corollary 3, the unique nth-root γ of λ is 1. With $\gamma = 1$, an AMNS can not be built. In this case, the largest value that can be represented in this AMNS is lower than $n\rho$. Indeed, the elements in the AMNS are polynomials of degree $n-1$ evaluated in $\gamma = 1$ and their coefficients are bounded by ρ. So, it is not possible to generate all elements in $\mathbb{Z}/p\mathbb{Z}$ with such a system, because $\rho \approx p^{1/n}$. The following corollary deals with the case $\lambda = 1$ and $\gcd(n, p-1) > 1$.

Corollary 4 *If $\gcd(n, p-1) > 1$, then there exists at least one non-trivial nth-root γ of 1. Therefore, we can take $\lambda = 1$.*

Proof. Let $\lambda = 1$, we are looking for γ such that $\gamma^n \equiv 1 \pmod p$. It is well known that there are $\gcd(n, p-1)$ nth-roots of unity modulo p. Let g be a generator of $(\mathbb{Z}/p\mathbb{Z}) \setminus \{0\}$, $d = \gcd(n, p-1)$ with $d > 1$, and let $h = g^{(p-1)/d} \pmod p$. Then, $h^n \equiv 1 \pmod p$ and $h \neq 1$. So, h is a non-trivial nth-root of λ.
The other nth-roots are $h^i \pmod p$, for $2 \leqslant i \leqslant d$. □

Remark 1 Except the algorithms for conversion from AMNS to classical representation, none of the algorithms we presented uses the parameter γ. Hence, this parameter has no impact on the efficiency of the arithmetic operations and the conversion operations from classical representation to AMNS.

From Corollary 3 and 4, given p and n, we deduce that if $\gcd(n, p-1) = 1$, then we can choose λ being any small integer different from 0 and 1.This states that a (unique) nth-root γ exists and its computation is easy. Likewise, if $\gcd(n, p-1) > 1$, it suffices to take $\lambda = 1$, because, in this case, non-trivial nth-roots of λ exist and can be computed easily.

Remark 2 If $\gcd(n, p-1) > 1$, Corollary 4 does not imply that it is necessary to take $\lambda = 1$. If a given $\lambda \neq 1$ has nth-roots modulo p, then there exists an algorithm [19] (used in SageMath library [31]) that computes these nth-roots. However, this algorithm computes a discrete logarithm in a small field. Its practical efficiency relies on the size of this "small" field.
Another possibility is to compute the irreducible factors of $E(X) = X^n - \lambda$ in $\mathbb{Z}/p\mathbb{Z}$ and to see if the degree of some of them is one. Their root will give a nth-root of λ. A probabilistic polynomial time algorithm to achieve this goal is given in [15].

5.3 Existence of the polynomial M'

In [26], the authors state that M must be chosen such that $\gcd(E,M) = 1$, but this does not guarantee the existence of M'. Indeed, if $\gcd(E,M) = 1$, then it exists $M' \in \mathbb{Q}[X]$ such that $MM' \equiv 1 \pmod E$, but nothing guarantees that the coefficients of M' are invertible modulo $\phi \geqslant 2$. Hence, there is no evidence that $\gcd(E,M)=1$ implies that $M'M \equiv 1 \bmod (E, \phi)$.

A first attempt, to prove it, has been published in [12]. In Section 3.3 of that article, when the value of p is already known and ϕ is a power of 2, the authors show how to build a lattice whose reduced basis always contains a polynomial M invertible modulo (E, ϕ). Unfortunately, their proof uses the fact that a polynomial M is invertible modulo (E, ϕ) if the evaluation of M over all integers is odd. This is a necessary but not a sufficient condition. For instance, let $E(X) = X^6 + 1$ and $M(X) = X^4 + X^2 + 1$, the evaluation of M is odd over all integers. However, it is not invertible modulo (E, ϕ) whenever ϕ is an even number, because the resultant of E and M is 16. This leads us to first remind some essential elements about the resultant of two polynomials.

Definition 3 (Resultant) [24, Def. 7.2.2, p. 227] Let $\mathcal{A}$ be a commutative ring with identity. Let A and B be two polynomials in $\mathcal{A}[X]$. The resultant $Res(A, B)$ of A and B is the determinant of their Sylvester matrix. Therefore, it is an element of $\mathcal{A}$.

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

$$\mathcal{S}_{A,B} = \begin{pmatrix} a_n & 0 & \dots & 0 & b_m & 0 & \dots & 0 \\ a_{n-1} & a_n & \ddots & \vdots & \vdots & b_m & \ddots & \vdots \\ \vdots & a_{n-1} & \ddots & 0 & \vdots & & \ddots & 0 \\ \vdots & \vdots & \ddots & a_n & b_1 & & & b_m \\ a_0 & & & a_{n-1} & b_0 & \ddots & \vdots & \vdots \\ 0 & \ddots & & \vdots & 0 & \ddots & b_1 & \vdots \\ \vdots & \ddots & a_0 & \vdots & \vdots & \ddots & b_0 & b_1 \\ 0 & \dots & 0 & a_0 & 0 & \dots & 0 & b_0 \end{pmatrix}$$

Notice that the matrix $\mathcal{S}_{A,B}$ above corresponds to the transpose of the representation given by Definition 7.2.1 in [24, p. 227]. Since matrix transposition does not modify matrix determinant, the two conventions apply to Definition 3.

In $\mathbb{Z}[X]$, there is no Bézout's identity, but the following essential property will help us to set an existence criteria for the polynomial M'.

Proposition 6 *[24, Lemma 7.2.1, p. 228] Let $\mathcal{A}$ be a commutative ring with identity and let A and B two non-zero polynomials in $\mathcal{A}[X]$ such that* $\deg(A)+\deg(B) \geqslant 1$. *There exist U and V in $\mathcal{A}[X]$ such that $A(X)U(X) + B(X)V(X) = Res(A, B)$,* $\deg(U) < \deg(B)$, *and* $\deg(V) < \deg(A)$.

We can now state our existence criteria for the polynomial M'.

Proposition 7 (Existence criteria) *Let $M \in \mathbb{Z}[X]$, $E \in \mathbb{Z}[X]$ and $\phi \geqslant 2$ an integer.*
If $\gcd(\mathrm{Res}(E, M), \phi) = 1$, *then there exists $J \in \mathbb{Z}[X]$ such that $JM \equiv 1$ mod (E, ϕ). That is, $J = M^{-1}$* mod (E, ϕ) *exists.*

Proof. Let $r = \mathrm{Res}(E,M)$ and let ψ be the inverse of r modulo ϕ. From Proposition 6, there exist U and V in $\mathbb{Z}[X]$ such that $U(X)M(X) + V(X)E(X) = r$. Hence, $\psi U(X)M(X) + \psi V(X)E(X) = \psi r$, which implies $\psi U(X)M(X) \equiv 1 \bmod (E, \phi)$. □

With Proposition 7, we established the requirements on E and M for the existence of the polynomial $J = M^{-1} \bmod (E, \phi)$. Since $M' = -M^{-1} \bmod (E, \phi)$, M' exists if and only if J exists.

The RedCoeff method (Algorithm 3) is efficiently implemented if ϕ is a power of two. This leads us to Corollary 5.

Corollary 5 *Let $M \in \mathbb{Z}[X]$, $E \in \mathbb{Z}[X]$ and $\phi = 2^j$, with $j \geqslant 1$ an integer.*
If $\mathrm{Res}(E, M)$ *is odd, then there exists $J \in \mathbb{Z}[X]$ such that $JM \equiv 1$ mod (E, ϕ).*

Proof. As ϕ is a power of two, $\gcd(\mathrm{Res}(E, M), \phi) = 1$ is equivalent to $\mathrm{Res}(E, M)$ is odd. Then, we can conclude with Proposition 7. □

Corollary 5 states that the Montgomery-like method can be used, with ϕ being a power of two, when $\mathrm{Res}(E,M)$ is odd.

In the sequel of this section, we give criteria on E and M so that $\mathrm{Res}(E,M)$ is odd. Let $E(X) = X^n - \lambda$ and $M(X) = m_0 + m_1X + \cdots + m_{n-1}X^{n-1}$. The Sylvester matrix of E and M is the $(2n-1) \times (2n-1)$ matrix defined as follows:

$$\mathcal{S}_{E,M} = \begin{pmatrix} 1 & 0 & \dots & 0 & m_{n-1} & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & m_{n-2} & m_{n-1} & \dots & 0 & 0 \\ \vdots & & \ddots & & \vdots & & & & \vdots \\ 0 & 0 & \dots & 1 & m_1 & m_2 & \dots & m_{n-1} & 0 \\ 0 & 0 & \dots & 0 & m_0 & m_1 & \dots & m_{n-2} & m_{n-1} \\ -\lambda & 0 & \dots & 0 & 0 & m_0 & \dots & m_{n-3} & m_{n-2} \\ \vdots & & & & \vdots & & \ddots & & \vdots \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & m_0 & m_1 \\ 0 & 0 & \dots & -\lambda & 0 & 0 & \dots & 0 & m_0 \end{pmatrix}$$

When ϕ is a power of two, the polynomial M' exists when $\mathrm{Res}(E,M)$ is odd. Since $\mathrm{Res}(E,M) = \det(\mathcal{S}_{E,M})$, it is equivalent to ensure that $\det(\mathcal{S}_{E,M})$ is odd. In order to establish conditions on E and M that guarantee $\det(\mathcal{S}_{E,M})$ is odd, we distinguish two cases according to the parity of the parameter λ of the AMNS. Before that, we need to introduce the following property.

Property 1 (Matrix determinant parity)
Let $A, B \in \mathcal{M}_{n\times n}(\mathbb{Z})$ be two matrices, such that: $A = (a_{ij})_{0\leqslant i,j<n}$ and $B = (b_{ij})_{0\leqslant i,j<n}$, with $b_{ij} = a_{ij} \pmod 2$. Then, the determinants of A and B have the same parity.

Proof. The application φ from $\mathbb{Z}$ to $\mathbb{Z}/2\mathbb{Z}$ such that $\varphi(n) = n \bmod 2$ is a ring morphism. □

5.3.1 Existence of the polynomial M' when λ is even.

Proposition 8 *Let $E(X) = X^n - \lambda$ be a polynomial such that λ is even. Let $M = m_0+m_1X+\cdots+m_{n-1}X^{n-1}$ be a polynomial. Then, det($\mathcal{S}_{E,M}$) is odd if and only if m_0 is odd.*

Proof. Since λ is even, using Property 1, it is obvious that the determinant of $\mathcal{S}_{E,M}$ has the same parity than the following matrix (where $\overline{m_i} = m_i \pmod 2$ and λ is replaced by 0) :

$$\begin{pmatrix}
1 & 0 & \dots & 0 & \overline{m_{n-1}} & 0 & \dots & 0 & 0\\
0 & 1 & \dots & 0 & \overline{m_{n-2}} & \overline{m_{n-1}} & \dots & 0 & 0\\
\vdots & & \ddots & & \vdots & & & & \vdots\\
0 & 0 & \dots & 1 & \overline{m_1} & \overline{m_2} & \dots & \overline{m_{n-1}} & 0\\
0 & 0 & \dots & 0 & \overline{m_0} & \overline{m_1} & \dots & \overline{m_{n-2}} & \overline{m_{n-1}}\\
0 & 0 & \dots & 0 & 0 & \overline{m_0} & \dots & \overline{m_{n-3}} & \overline{m_{n-2}}\\
\vdots & & & & \vdots & & \ddots & & \vdots\\
0 & 0 & \dots & 0 & 0 & 0 & \dots & \overline{m_0} & \overline{m_1}\\
0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & \overline{m_0}
\end{pmatrix}$$

It is an upper triangular matrix with only the value 1 or $\overline{m_0}$ on the diagonal. So, its determinant is 1 if and only if m_0 is odd. Therefore, det($\mathcal{S}_{E,M}$) is odd if and only if m_0 is odd. □

From Proposition 8, when ϕ is a power of two and λ even, the polynomial M' exists if m_0 is odd.

5.3.2 Existence of the polynomial M' when λ is odd.

For the case λ odd, we need some additional results.
Let $\mathcal{H}_1$ be the matrix obtained from $\mathcal{S}_{E,M}$ by replacing m_i by $\overline{m_i} = m_i \pmod 2$. We also replace λ by -1.

$$\mathcal{H}_1 = \begin{pmatrix}
1 & 0 & \dots & 0 & \overline{m_{n-1}} & 0 & \dots & 0 & 0\\
0 & 1 & \dots & 0 & \overline{m_{n-2}} & \overline{m_{n-1}} & \dots & 0 & 0\\
\vdots & & \ddots & & \vdots & & & & \vdots\\
0 & 0 & \dots & 1 & \overline{m_1} & \overline{m_2} & \dots & \overline{m_{n-1}} & 0\\
0 & 0 & \dots & 0 & \overline{m_0} & \overline{m_1} & \dots & \overline{m_{n-2}} & \overline{m_{n-1}}\\
-1 & 0 & \dots & 0 & 0 & \overline{m_0} & \dots & \overline{m_{n-3}} & \overline{m_{n-2}}\\
\vdots & & & & \vdots & & \ddots & & \vdots\\
0 & 0 & \dots & 0 & 0 & 0 & \dots & \overline{m_0} & \overline{m_1}\\
0 & 0 & \dots & -1 & 0 & 0 & \dots & 0 & \overline{m_0}
\end{pmatrix}$$

Since λ is odd, Property 1 assures us that det($\mathcal{S}_{E,M}$) and det($\mathcal{H}_1$) have the same parity.

The addition of one row to another row does not change the value of the determinant. Therefore, det($\mathcal{H}_2$) = det($\mathcal{H}_1$), with $\mathcal{H}_2$ defined as follows:

$$\mathcal{H}_2 = \begin{pmatrix}
1 & 0 & \dots & 0 & \overline{m_{n-1}} & 0 & \dots & 0 & 0\\
0 & 1 & \dots & 0 & \overline{m_{n-2}} & \overline{m_{n-1}} & \dots & 0 & 0\\
\vdots & & \ddots & & \vdots & & & & \vdots\\
0 & 0 & \dots & 1 & \overline{m_1} & \overline{m_2} & \dots & \overline{m_{n-1}} & 0\\
0 & 0 & \dots & 0 & \overline{m_0} & \overline{m_1} & \dots & \overline{m_{n-2}} & \overline{m_{n-1}}\\
0 & 0 & \dots & 0 & \overline{m_{n-1}} & \overline{m_0} & \dots & \overline{m_{n-3}} & \overline{m_{n-2}}\\
\vdots & & & & \vdots & & \ddots & & \vdots\\
0 & 0 & \dots & 0 & \overline{m_2} & \overline{m_3} & \dots & \overline{m_0} & \overline{m_1}\\
0 & 0 & \dots & 0 & \overline{m_1} & \overline{m_2} & \dots & \overline{m_{n-1}} & \overline{m_0}
\end{pmatrix}$$

Then, we have det($\mathcal{H}_3$) = det($\mathcal{H}_2$), where $\mathcal{H}_3$ is the circulant matrix defined as follows:

$$\mathcal{H}_3 = \begin{pmatrix}
\overline{m_0} & \overline{m_1} & \dots & \overline{m_{n-2}} & \overline{m_{n-1}}\\
\overline{m_{n-1}} & \overline{m_0} & \dots & \overline{m_{n-3}} & \overline{m_{n-2}}\\
\vdots & & \ddots & & \vdots\\
\overline{m_2} & \overline{m_3} & \dots & \overline{m_0} & \overline{m_1}\\
\overline{m_1} & \overline{m_2} & \dots & \overline{m_{n-1}} & \overline{m_0}
\end{pmatrix}$$

Therefore, det($\mathcal{S}_{E,M}$) and det($\mathcal{H}_3$) have the same parity. Since $\mathcal{H}_3 \in \mathcal{M}_{n\times n}(\mathbb{Z}/2\mathbb{Z})$, det($\mathcal{H}_3$) is either 0 or 1. In the sequel of this section, we will focus on the circulant matrix $\mathcal{H}_3$ to establish the requirement on M so that M' exists.

With Definition 4, we introduce a notation useful to establish our requirement.

Definition 4 Let $P \in \mathbb{Z}[X]$ be a polynomial such that $P(X) = p_0 + p_1X + \cdots + p_{n-1}X^{n-1}$. We denote by $\overline{P}$ the polynomial such that: $\overline{P}(X) = p'_0 + p'_1X + \cdots + p'_{n-1}X^{n-1}$ where $p'_i = p_i \bmod (2)$.

To establish the requirement on M using $\mathcal{H}_3$, we need the application Υ defined in Proposition 9.

Proposition 9 *[9, Section 3.4] Let $R_n = \mathbb{F}_2[X]/(X^n - 1)$ be the algebra of all polynomials modulo $(X^n - 1)$ over $\mathbb{F}_2$. Let C_n be the ring of $n \times n$ binary circulant matrices.*
Let Υ be the application defined as follows:

$$\begin{array}{rccc}
\Upsilon : & R_n & \to & C_n\\
& a_0 + \cdots + a_{n-1}X^{n-1} & \mapsto & \begin{pmatrix}
a_0 & a_1 & \dots & a_{n-2} & a_{n-1}\\
a_{n-1} & a_0 & \dots & a_{n-3} & a_{n-2}\\
\vdots & & \ddots & & \vdots\\
a_2 & a_3 & \dots & a_0 & a_1\\
a_1 & a_2 & \dots & a_{n-1} & a_0
\end{pmatrix}
\end{array}$$

Then, R_n is isomorphic to C_n and $\Upsilon(R_n) = C_n$.

From Proposition 9, we deduce Corollary 6 which is essential to establish the existence of the polynomial M'.

Corollary 6 *Let $A \in R_n$ be a polynomial, det($\Upsilon(A)$) $= 1$ if and only if* $\gcd(A, X^n - 1) = 1$.

Proof. A is invertible if and only if $\gcd(A, X^n - 1) = 1$. An element in C_n is invertible if and only if its determinant is 1. Since R_n is isomorphic to C_n through Υ, this concludes the proof. □

Using Definition 4 and Corollary 6, we can now state the existence criteria for the polynomial M'.

Proposition 10 *Let $E(X) = X^n - \lambda$ such that λ is odd. Let $M = m_0 + m_1 X + \cdots + m_{n-1} X^{n-1}$ be a polynomial. Then, det($\mathcal{S}_{E,M}$) is odd if and only if* $\gcd(\overline{M}, X^n - 1) = 1$.

Proof. The circulant matrix $\mathcal{H}_3$, defined above, is such that $\mathcal{H}_3 = \Upsilon(\overline{M})$. As $\overline{M} \in R_n$, $\det(\mathcal{H}_3) = 1$ if and only if $\gcd(\overline{M}, X^n - 1) = 1$, using Corollary 6. Since $\det(\mathcal{S}_{E,M})$ and $\det(\mathcal{H}_3)$ have the same parity, we deduce that $\det(\mathcal{S}_{E,M})$ is odd iff $\gcd(\overline{M}, X^n - 1) = 1$. □

From Proposition 10, when ϕ is a power of two and λ is odd, the polynomial M' exists if $\gcd(\overline{M}, X^n - 1) = 1$.

To sum up, when ϕ is a power of two and $E(X) = X^n - \lambda$ with $\lambda \in \mathbb{Z} \setminus \{0\}$, the polynomial M' exists:

- if λ is even and m_0 is odd,
- if λ is odd and $\gcd(\overline{M}, X^n - 1) = 1$.

5.4 Generation of the polynomial M

Let ϕ be a power of two, we show in this section how to build the internal reduction polynomial $M(X) = m_0 + m_1 X + \cdots + m_{n-1} X^{n-1}$ so that the requirements for the existence of M' are met. Before that, we need to consider some facts about the polynomial M.

From Proposition 1, we have $p \leqslant (2\rho - 1)^n$. It is wise to choose ρ as small as possible. This way, it lowers the memory amount required to represent elements in $\mathbb{Z}/p\mathbb{Z}$. The parameter ρ is such that $\rho \geqslant 2n|\lambda|\|M\|_\infty$. As a consequence, $\|M\|_\infty$ must be small to ensure that ρ be small. Choosing $\|M\|_\infty$ small is also important for the generation strategy we presented in Section 5.1 for software implementation.

In order to find such a polynomial, we use the lattice $\mathcal{L}$ of all polynomials having γ as root modulo p and which degree is lower than n:

$$\mathcal{L} = \{a(X) \in \mathbb{Z}[X], \text{ such that: } deg(a) < n \text{ and } a(\gamma) \equiv 0 \bmod (p)\}$$

This lattice is well defined in [8]. The idea for finding M is to compute a reduced basis of $\mathcal{L}$ using a lattice reduction algorithm (like LLL algorithm [23]) and then to take M as an element (or a special combination of elements) of this reduced basis. However, a naive approach does not guarantee that this reduced basis always provides a polynomial M which meets the requirements of the existence of the polynomial M'. In Section 5.4.1 and Section 5.4.2, we address this issue according to the parity of λ.

5.4.1 Generation of M when λ is even.

Here, we expect to built a polynomial M such that m_0 is odd. This ensures that M' exists.
A simple basis of the lattice $\mathcal{L}$ is:

$$\mathcal{M}_1 = \begin{pmatrix} p & 0\,0 \ldots & 0\,0 \\ t_1 & 1\,0 \ldots & 0\,0 \\ t_2 & 0\,1 \ldots & 0\,0 \\ \vdots & & \ddots & \vdots \\ t_{n-2} & 0\,0 \ldots & 1\,0 \\ t_{n-1} & 0\,0 \ldots & 0\,1 \end{pmatrix}$$

where $t_i = (-\gamma)^i \bmod p$.
In the basis $\mathcal{M}_1$, each row corresponds to a polynomial having γ as a root modulo p. For instance, row two corresponds to the polynomial $X + t_1$.

Proposition 11 *Let $\mathcal{G} = \{\mathcal{G}_0, \ldots, \mathcal{G}_{n-1}\}$ be a reduced basis of the lattice $\mathcal{L}$ obtained from the basis $\mathcal{M}_1$. Then, at least one (row) vector $\mathcal{G}_i$ of $\mathcal{G}$ is such that $\mathcal{G}_{i,0}$ is odd.*

Proof. $\mathcal{G}$ is a basis of $\mathcal{L}$, so the vector $V = (p, 0, \ldots, 0)$ (i.e., the first line of $\mathcal{M}_1$) is a linear combination (over $\mathbb{Z}$) of elements of $\mathcal{G}$. Now, let's assume that the first component of all elements of $\mathcal{G}$ is even. This means that every linear combination of elements from $\mathcal{G}$ will output a vector whose first component is even. As p is odd, this is in contradiction with the fact that $V \in \mathcal{L}$ and $\mathcal{G}$ is a basis of $\mathcal{L}$. Thus, at least one element $\mathcal{G}_i$ of $\mathcal{G}$ is such that $\mathcal{G}_{i,0}$ is odd. □

From Proposition 11, we deduce that any lattice reduction algorithm applied to the basis $\mathcal{M}_1$ outputs a reduced basis which contains at least one polynomial M such that m_0 is odd.

5.4.2 Generation of M when λ is odd.

Here, we expect to built a polynomial M such that $\gcd(\overline{M}, X^n - 1) = 1$. This ensures that M' exists.
We consider another basis of the lattice $\mathcal{L}$:

$$\mathcal{M}_2 = \begin{pmatrix} p & 0\,0 \ldots 0\,0 \\ s_1 & 1\,0 \ldots 0\,0 \\ s_2 & 0\,1 \ldots 0\,0 \\ \vdots & \ddots & \vdots \\ s_{n-2} & 0\,0 \ldots 1\,0 \\ s_{n-1} & 0\,0 \ldots 0\,1 \end{pmatrix}$$

where: $s_i = t_i + pk_i$ with $t_i = (-\gamma)^i \bmod p$ and $k_i = t_i \bmod 2$. Notice that all s_i are even, because p is odd. Similarly to the previous case, each row corresponds to a polynomial having γ as a root modulo p.

Proposition 12 *Let $\mathcal{G} = \{\mathcal{G}_0, \ldots, \mathcal{G}_{n-1}\}$ be a reduced basis of the lattice $\mathcal{L}$ obtained from the basis $\mathcal{M}_2$. Then, there exists a linear combination $(\beta_0, \ldots, \beta_{n-1})$, with $\beta_i \in \{0, 1\}$, such that $M = \sum_{i=0}^{n-1} \beta_i \mathcal{G}_i$ satisfies:*

$$\gcd(\overline{M}, X^n - 1) = 1\,.$$

Proof. Since all s_i are even in $\mathcal{M}_2$, we have that:

$$\overline{\mathcal{M}_2} = \begin{pmatrix} 1\,0\,0 \ldots 0\,0 \\ 0\,1\,0 \ldots 0\,0 \\ 0\,0\,1 \ldots 0\,0 \\ \vdots & \ddots & \vdots \\ 0\,0\,0 \ldots 1\,0 \\ 0\,0\,0 \ldots 0\,1 \end{pmatrix}$$

where $\overline{\mathcal{M}_2}_{ij} = \mathcal{M}_{2ij} \bmod 2$.
Let $R_n = \mathbb{F}_2[X]/(X^n - 1)$ be the algebra of all polynomials modulo $(X^n - 1)$ over $\mathbb{F}_2$ (see Proposition 9). Each line i of $\overline{\mathcal{M}_2}$ corresponds to the polynomial $X^i \in R_n$, for $0 \leqslant i < n$. This means that $\overline{\mathcal{M}_2}$ is a basis of R_n. Let $U \in R_n$ be a polynomial such that $gcd(U, X^n - 1) = 1$. Since $\overline{\mathcal{M}_2}$ is a basis of R_n, there exists $T = (t_0, \ldots, t_{n-1}) \in \mathbb{F}_2^n$ such that $U = T\overline{\mathcal{M}_2}$. As $T\overline{\mathcal{M}_2} = \overline{T\mathcal{M}_2}$, we obtain that $U = \overline{T\mathcal{M}_2}$.
We have $T\mathcal{M}_2 \in \mathcal{L}$, so there exists $V = (v_0, \ldots, v_{n-1}) \in \mathbb{Z}^n$ such that $V\mathcal{G} = T\mathcal{M}_2$, since $\mathcal{G}$ is a basis of $\mathcal{L}$. Thus, $U = \overline{V\mathcal{G}}$.
Let $\beta = (\beta_0, \ldots, \beta_{n-1}) \in \mathbb{F}_2^n$ such that $\beta_i = v_i \bmod 2$, then one has $U = \overline{\beta\mathcal{G}}$.
Let $M \in \mathcal{L}$ be a polynomial such that $M = \sum_{i=0}^{n-1} \beta_i \mathcal{G}_i$, then $\overline{M} = U$, hence $\gcd(\overline{M}, X^n - 1) = 1$. □

From Proposition 12, any lattice reduction algorithm applied to the basis $\mathcal{M}_2$ outputs a reduced basis $\mathcal{G}$ such that at least one binary linear combination of its row gives a polynomial M such that $\gcd(\overline{M}, X^n - 1) = 1$. Thus, one needs to check at most 2^n linear combinations (of the rows of $\mathcal{G}$) to find a suitable polynomial M. For cryptographic sizes, n is small enough to allow the test of all these combinations (see examples in Appendix B, for some possible values of n).

Let $\theta = \max_{0 \leqslant i < n} \|\mathcal{G}_i\|_\infty$. For each binary linear combination, the corresponding polynomial M verifies $\|M\|_\infty \leqslant n\theta$. As a consequence, if elements of $\mathcal{G}$ are small, then $\|M\|_\infty$ is also small, because n is small and negligible compared to θ.

6 Implementation and analyses

In this section, we study the complexity of the main operations and the memory requirement of AMNS. We also discuss the number of AMNS that can be built for a given prime. Additionally, we provide some implementation strategies, along with some comparison with multiprecision libraries implementations. We end this section with a discussion on some features of the AMNS regarding side channel attacks.

6.1 Theoretical performances and memory requirement

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

We assume that the inputs of our algorithms belong to the AMNS $\mathcal{B} = (p, n, \gamma, \rho, E)$, with $\rho = 2^t$, $E(X) = X^n - \lambda$ and $\lambda = \pm 2^i + \varepsilon 2^j$, with $t, i, j \in \mathbb{N}$ and $\varepsilon \in \{-1, 0, 1\}$. This kind of λ ensures the fastest external reduction; especially for $\varepsilon = 0$. From Section 5.2, we know that it is always possible to choose such λ. We give here the theoretical performances and memory requirement according to the software implementation strategy we explained in Section 5.1. The memory usage is expressed as a function of the number of k-bit data words. The operations performances are expressed as a function of the number of k-bit integer multiplications, additions and shifts.

Since elements are polynomials in $\mathcal{B}$, n k-bit data words are required to represent each of them. As a consequence, an element in $\mathcal{B}$ requires nk bits to be represented.

Let $\mathcal{M}$ and $\mathcal{A}$ respectively denote the multiplication and the addition of two k-bit integers. We also respectively denote $\mathcal{S}_l^i$ and $\mathcal{S}_r^i$ a left shift and a right shift of i bits. In Table 2, we give the costs of the polynomials addition, multiplication, the internal reduction and the external reduction. We remind that, with the parameter δ, it is always possible to postpone as late as

wanted the internal reduction after an addition. The polynomials M and M' are constants that are known once the AMNS is generated. The multiplications by these polynomials (modulo E) are optimised by directly combining them with the reduction modulo E. In this table, *Mod. Mult.* is a complete modular multiplication of two elements in $\mathcal{B}$. It is the sequence of a polynomials multiplication (*Poly. Mult.*), an external reduction (*Ext. reduct.*) and an internal reduction (*Int. reduct*). Also, *Poly. Add.* denotes the addition of two elements in $\mathcal{B}$.

Let $x = x_1x_2$ and $y = y_1y_2$, where x_i and y_i are k-bit data words, i.e., x and y are products of two k-bit integers. In Table 2, we consider that the computation $x + y$ costs $2\mathcal{A}$.

Table 2: Theoretical cost of operations, where $E(X) = X^n - \lambda$, with $\lambda = \pm 2^i + \varepsilon 2^j$, $\varepsilon \in \{-1, 0, 1\}$ and $\phi = 2^k$.

Poly. Add.	$n\mathcal{A}$
Poly. Mult.	$n^2\mathcal{M} + (2n^2 - 4n + 2)\mathcal{A}$
Ext. reduct.	$2(n-1)\mathcal{A} + (n-1)\mathcal{S}_l^i$ $+ \lvert\varepsilon\rvert\big(2(n-1)\mathcal{A} + (n-1)\mathcal{S}_l^j\big)$
Int. reduct.	$2n^2\mathcal{M} + (3n^2 - n)\mathcal{A} + n\mathcal{S}_r^k$
Mod. Mult.	$3n^2\mathcal{M} + (5n^2 - 3n)\mathcal{A} + (n-1)\mathcal{S}_l^i + n\mathcal{S}_r^k$ $+ \lvert\varepsilon\rvert\big(2(n-1)\mathcal{A} + (n-1)\mathcal{S}_l^j\big)$

6.2 Some advantages of the AMNS

In Section 4, we have presented a set of algorithms that are the basis of all the other operations that might be done in the AMNS. Here, we highlight some advantages of these algorithms.

First, none of them has a conditional branching. This property is an advantage for efficiency and is also very helpful against side channel attacks.

Secondly, because the AMNS elements are polynomials, their coefficients are independent and there is no carry propagation to deal with, when performing arithmetic operations.

Lastly, except the conversion procedures from AMNS to binary representation, each line of the algorithms presented in Section 4 computes a polynomial. As the coefficients are independent, they can be computed simultaneously. As a consequence, AMNS is well fitted for a parallel implementation.

6.3 Numbers of AMNS for a given prime

An interesting but complex question is how many AMNS can be generated given a prime number and a target architecture. This question is difficult to answer because of the large range of parameters that define an AMNS and also because it is linked to the existence of a nth-root γ of a given λ in $\mathbb{Z}/p\mathbb{Z}$.

Here, we give an answer while focusing on the efficiency of arithmetic operations. This leads us to add some constraints on n and λ which of course reduce the number of AMNS.

Let us first assume that we have a k-bit architecture. Once p is known, we have to choose the parameters n, ϕ and λ. For any AMNS, we have: $\phi \geqslant 2|\lambda|n\rho$. Also, from Proposition 1, we have $p \leqslant (2\rho - 1)^n$, i.e,: $p < (2\rho)^n$. So, $\log_2|\lambda| \leqslant \log_2\phi - \log_2 n - (\log_2 p)/n$. Here, we consider the generation strategy for software implementation presented in Section 5.1. This means that $\phi = 2^k$ and $n > \frac{\log_2 p}{k}$. Here, n is the number of k-bit words used to represent elements in $\mathcal{B}$. We want n as small as possible to minimize the computations. Let's assume that we choose n such that:

$$\frac{\log_2 p}{k} + 1 + c \geqslant n > \frac{\log_2 p}{k}$$

This means that we allow at most c more coefficients than the optimal value $\lfloor\frac{\log_2 p}{k}\rfloor + 1$. According to Table 2 (Section 6.1), the smaller is c, the better are the performances and memory requirement. Therefore, $\log_2 n > \log_2\log_2 p - \log_2 k$ and $\frac{\log_2 p}{n} \geqslant \frac{k\log_2 p}{\log_2 p + kc + k}$. So, one has:

$$\log_2|\lambda| < k + \log_2 k - \log_2\log_2 p - \tfrac{k\log_2 p}{\log_2 p + kc + k}.$$

Let $\zeta = \lfloor k + \log_2 k - \log_2\log_2 p - \frac{k\log_2 p}{\log_2 p + kc + k}\rfloor$. As said in Section 6.1, we choose λ such that $\lambda = \pm 2^i + \varepsilon 2^j$, with $i, j \in \mathbb{N}$, $\varepsilon \in \{-1, 0, 1\}$ and $|\lambda| \leqslant 2^\zeta$, in order to speed up the reduction modulo $X^n - \lambda$ in the external reduction process.
Let $\Omega = \{\lambda \neq 0$, such that: $\lambda = \pm 2^i + \varepsilon 2^j$, with $i, j \in \mathbb{N}, \varepsilon \in \{-1, 0, 1\}$ and $|\lambda| \leqslant 2^\zeta\}$, hence we can choose $\mathrm{card}(\Omega)$ distinct values for λ. The main difficulty is to determine the number of nth-roots modulo p which can be computed with this set of values for λ. Some of them could have many nth-roots while others not even one. To give an answer, we distinguish two cases according to $\gcd(n, p-1)$.

6.3.1 Case 1: gcd$(n, p-1) = 1$.

Using Corollary 3, we obtain that any value $\lambda \in \mathbb{Z} \setminus \{0, 1\}$ gives a suitable nth-root modulo p. So, in this case, one can generate at least $\mathrm{card}(\Omega) - 1$ AMNS.

6.3.2 Case 2: $gcd(n, p-1) > 1$.

This case is more difficult because it required to compute the factorisation of $X^n - \lambda$ (see Remark 2). However, from Corollary 4, when $\lambda = 1$, at least $\gcd(n, p-1) - 1$ AMNS can be generated.

Remark 3 In both cases, we provided the minimum numbers of AMNS that can be generated. Indeed, once λ and $\gamma = \lambda^{1/n} \pmod p$ are determined, we compute the polynomial M using the lattice reduction. From Section 5.4, we know that any reduced basis (obtained from the appropriate base) provides at least one polynomial which satisfies the required constraints on M. Our numerical experiments show that there are more than one suitable candidate for M. Moreover, some linear combinations of elements of the reduced basis give suitable candidates for M. For a tuple (p, n, λ, γ), distinct polynomials M lead to distinct AMNS. Thus, one can generate much more AMNS than the minimum numbers we gave, using some linear combinations of the polynomials of the reduced basis.

Example 3 We generated a set of AMNS for some primes of 192, 224, 256, 384 and 521 bits. These sizes correspond to the NIST recommended key sizes for elliptic curve cryptography. For this experiment, we took $k = 64$ and $c = 2$. Doing so, we allow at most 2 more coefficients than the optimal value $\lfloor \frac{\log_2 p}{64} \rfloor + 1$, which is quite restrictive but good for performances. So, $\zeta = \lfloor 70 - \log_2 \log_2 p - \frac{64 \log_2 p}{\log_2 p + 192} \rfloor$. We chose $\lambda \in \Omega$, as defined above.

For our test, we used SageMath library which implements the algorithm proposed in [19] to compute the nth-root modulo p of λ. As it can be time consuming, we put in our code a timeout of thirty minutes. Thus, some values of λ that have nth-roots might have been discarded. Finally, in order to extend the number of AMNS, we checked all the 2^n binary combinations of the vectors of the reduced basis computed during the generation of the polynomial M. A larger set of combinations should lead to more AMNS.

With these parameters and constraints, the Table 3 gives the number of AMNS we found for each prime. We call these primes p192, p224, p256, p384 and p521 according to their bit size (see Appendix A for their values).

Prime number	p192	p224	p256
Number of AMNS	10418	5118	11877

Prime number	p384	p521
Number of AMNS	14787	19871

Table 3: A lower bound on the number of distinct AMNS for some prime integers, using a timer of 30 minutes for nth-root computation.

6.4 Implementation results

We wrote with the SageMath library [31] a code that generates a C code for any AMNS defined with its complete set of parameters. Our implementations of AMNS generation, the C code generator and the AMNS we used for our numerical experiments below are available on GitHub:

`https://github.com/arithPMNS/generalisation_amns`

For our tests we used a Dell Precision Tower 3620 on Ubuntu gnome 16.04-64 bits with an Intel Core i7-6700 processor and 32GB RAM. We compiled our tests with gcc 5.4 using -O3 compiling option and we compared our results to GNU MP 6.1.1 and OpenSSL 1.0.2g implementations. These libraries have also been compiled with gcc and -O3 option.

6.4.1 Performances.

For each NIST recommended key size for elliptic curve cryptography (i.e., 192, 224, 256, 384 and 521 bits), we generated a set of AMNS for many primes of the specified size. See Appendix B for some examples of AMNS.

Like in Example 3, we took $k = 64$ and $\phi = 2^k$. Therefore, the optimal value of n for software implementation is $n_{opt} = \lfloor \frac{\log_2 p}{k} \rfloor + 1$. We also took $c = 2$, which means that for each prime, we generated AMNS with n equals to n_{opt}, $n_{opt} + 1$ and $n_{opt} + 2$. Additionally, λ was chosen such that $\lambda = \pm 2^i + \varepsilon 2^j$, with $i, j \in \mathbb{N}$ and $\varepsilon \in \{-1, 0, 1\}$.

For each AMNS, we computed 2^{25} modular multiplications using the AMNS representatives of some random elements of $\mathbb{Z}/p\mathbb{Z}$. As a comparison, we also computed 2^{25} modular multiplications with the well known libraries GNU MP [17] and OpenSSL [28] with the same inputs. For OpenSSL, we used both the default modular multiplication procedure and the Montgomery modular multiplication. For GNU MP, we compared the AMNS to both the low level and the high level functions for modular multiplication.

In Table 4, we give the mean ratio between the performances obtained for the AMNS, GNU MP and OpenSSL. We compute these ratios for n equals to n_{opt}, $n_{opt} + 1$ and $n_{opt} + 2$.

In Table 5, we give the best ratios obtained for $n = n_{opt}$ for AMNS. They are obtained for λ in $\{\pm 1, \pm 2, \pm 4\}$.

p size	192			224		
n	**4**	**5**	**6**	**4**	**5**	**6**
ratio 1	**0.86**	1.41	2.04	**0.57**	**0.98**	1.41
ratio 2	0.10	0.17	0.24	0.08	0.14	0.19
ratio 3	0.21	0.34	0.49	0.16	0.27	0.39
ratio 4	0.36	0.58	0.86	0.23	0.39	0.58

p size	256			384		
n	**5**	**6**	**7**	**7**	**8**	**9**
ratio 1	**0.98**	1.42	1.84	**0.98**	1.34	1.67
ratio 2	0.14	0.20	0.26	0.19	0.25	0.31
ratio 3	0.30	0.43	0.55	0.43	0.58	0.73
ratio 4	0.45	0.67	0.87	0.61	0.80	1.04

p size	521		
n	**10**	**11**	**12**
ratio 1	**0.95**	1.18	1.36
ratio 2	0.25	0.29	0.34
ratio 3	0.56	0.69	0.80
ratio 4	0.69	0.83	0.96

ratio 1: AMNS/OpenSSL Montgomery modular mult.
ratio 2: AMNS/OpenSSL default modular mult.
ratio 3: AMNS/GNU MP mult. + modular reduction.
ratio 4: AMNS/GNU MP mult. + modular reduction, using low level functions.

Table 4: Relative performances of AMNS vs GNU MP and OpenSSL modular multiplications, with n equals to n_{opt}, $n_{opt} + 1$ and $n_{opt} + 2$ for the AMNS.

(p size, n)	(192, 4)	(224, 4)	(256, 5)
ratio 1	**0.77**	**0.56**	**0.91**
ratio 2	0.09	0.08	0.13
ratio 3	0.19	0.16	0.28
ratio 4	0.33	0.23	0.41

(p size, n)	(384, 7)	(521, 10)
ratio 1	**0.92**	**0.91**
ratio 2	0.18	0.24
ratio 3	0.40	0.54
ratio 4	0.57	0.66

ratio 1: AMNS/OpenSSL Montgomery modular mult.
ratio 2: AMNS/OpenSSL default modular mult.
ratio 3: AMNS/GNU MP mult. + modular reduction.
ratio 4: AMNS/GNU MP mult. + modular reduction, using low level functions.

Table 5: Relative performances of AMNS vs GNU MP and OpenSSL modular multiplications, with n equals to n_{opt} for the AMNS (best ratios).

Remark 4 When p has 521 bits, the optimal value for n is 9. With our constraints on k, ϕ, λ and the timer we used to compute a nth-root modulo p of λ, we did not find an AMNS with $n = 9$. So, for this size, the ratios were computed with n equals to $n_{opt} + 1$, $n_{opt} + 2$ and $n_{opt} + 3$.

In both Tables 4 and 5, it can be observed that the AMNS performs modular multiplication more efficiently than the library GNU MP and the default method in OpenSSL for all values of n. We also observe that AMNS modular multiplication is slightly faster than the Montgomery method in OpenSSL when the value of n is optimal. Moreover, it can be observed that when n_{opt} is equal to the number of k-bit blocks used to represent an integer of $\mathbb{Z}/p\mathbb{Z}$ in GNU MP and OpenSSL, AMNS largely outperforms both libraries, even the Montgomery method in OpenSSL. It is the case for $p = 224$ and $p = 521$. As explained in Remark 4, we did not find AMNS for $(p, n) = (521, 9)$. Yet, with $(p, n) = (521, 10)$, AMNS is more efficient than both libraries.
In the other cases, AMNS uses at least one more block than GNU MP and OpenSSL but remains competitive.

To sum up, AMNS outperforms both GNU MP and OpenSSl because:

- there is no carry to manage nor conditional branching because of the polynomial structure of AMNS elements,
- the shape of the polynomial $E(X) = X^n - \lambda$, makes the external reduction very fast with Algorithm 1 (Section 3.2). Moreover, the parameter λ has been chosen equal to $\pm 2^i + \varepsilon 2^j$, which also speeds up the external reduction (best ratios are obtained for $\lambda \in \{\pm 1, \pm 2, \pm 4\}$),
- ϕ has been chosen according to the software implementation strategy described in Section 5.1.

These results do not take advantage of the parallel aspect of the AMNS. As explained in Section 6.2, a parallel implementation should speed up arithmetic operations, because the AMNS elements are polynomials. This approach could divide by n the execution time of line 1 in Algorithm 4. Likewise, the same thing can be done with all the lines of the RedCoeff method (Algorithm 3), which is called in Algorithm 4.

Remark 5 Although, AMNS is a lot faster than GNU MP and OpenSSL default method for modular multiplication, the most relevant ratio in Table 4 and Table 5 is *ratio 1*, because AMNS requires roughly the same amount of precomputed data than the Montgomery modular multiplication.

An AMNS addition should always be faster because it is a simple polynomials addition without carries to

manage, unlike the classic binary representation. Moreover, it is possible to simulteanously compute all the n coefficients of the result. This should speed up addition.

6.4.2 Memory requirement.

Table 6 gives the amount of memory required for storing an integer modulo p in AMNS, GNU MP and OpenSSL. We give the number of 64-bit integers used to represent the elements of $\mathbb{Z}/p\mathbb{Z}$. Here, we consider AMNS built with $n = n_{opt}$. We give in Appendix D the source code corresponding to the data structure used for storing an integer with GNU MP and OpenSSL. For GNU MP low level functions, an element is an array of 64-bit integers.

In Table 6, it can be observed that the memory usage of AMNS and GNU MP *mpz_t* type are almost the same while OpenSSL requires more memory. However, AMNS uses generally one more 64-bit integer than for GNU MP low level functions.

Size in bits of p	192	224	256	384	521
AMNS	4	4	5	7	10
GNU MP (low level)	3	4	4	6	9
GNU MP (mpz_t)	4	5	5	7	10
OpenSSL (BIGNUM)	5	6	6	8	11

Table 6: Number of 64-bit words used to store elements of $\mathbb{Z}/p\mathbb{Z}$ in GNU MP, OpenSSL and the AMNS (when n equals to n_{opt}).

Remark 6 For AMNS and OpenSSL Montgomery, some data must be precomputed. However, the memory requirements of these data are negligible compared to the overall usage of memory when performing multiple arithmetic operations.

6.5 About side channel attacks

AMNS have very interesting properties regarding side channel attacks.

6.5.1 Regular algorithms.

All the described algorithms (reductions, conversions, addition and multiplication) contain no conditional branching which are one of the basic weaknesses used in simple power analysis (SPA) attacks [21].

6.5.2 Randomisation inside the AMNS.

A classical countermeasure to protect an algorithm against differential power analysis (DPA) attacks [22,1] is to randomise the input and all intermediate values. In a recent work [11], Didier et al. show how to use the redundancy inside the AMNS to reach this goal. Additionally, they show that their proposal has some specific advantages regarding special attacks like Goubin's attack [16].

6.5.3 Randomisation using several AMNS.

In Section 6.3, we show that several AMNS can be built for the same modulus p. This property could be used to randomise the representation of the input data of a cryptographic protocol.

Given a tuple (p, n, λ), if γ a nth-root (modulo p) of λ exists then, from Proposition 5, the number of such roots is $\gcd(n, p-1)$. Each of them allows to build at least one AMNS for the same prime p. Moreover, Corollaries 3 and 4 show that for any pair (p, n), it is always possible to choose $\lambda \in \mathbb{Z} \setminus \{0\}$ such that $\gamma^n \pmod p \equiv \lambda$, with $\gamma \in \mathbb{Z}/p\mathbb{Z}$ easily computable. This means that it is always possible to generate many AMNS, given a prime p.

For instance, with our implementation, we show that it is possible to generate thousands of AMNS for many moduli using restrictive conditions (see Table 3 in Example 3).

One could take advantage of the existence of many distinct AMNS for a given modulus p in order to have many representatives for each element of $\mathbb{Z}/p\mathbb{Z}$ and randomise cryptographic protocols. Here, the randomisation is achieved by first precomputing a set Γ of AMNS for the modulus p. Then, each time a protocol using that modulus p is executed, one randomly selects an AMNS in Γ and performs the arithmetic operations using that AMNS. Doing so, each execution of that protocol, with the same input, is expected to lead to a random representative of that input. In Appendix C, we give examples of representatives for some elements from one AMNS to another. Now, it remains to study if there exist correlations, between the representatives of $\mathbb{Z}/p\mathbb{Z}$ elements from one AMNS to another, which can lead to a potential cryptanalysis. We plan to make this study in a future work.

6.5.4 A simple DPA countermeasure for ECC.

In [20], the authors show how to randomise the base point P to thwart DPA attacks [22]. The main idea is to change the (x, y) coordinates of P by $(u^{-2}x, u^{-3}y)$ for

a random $u \in (\mathbb{Z}/p\mathbb{Z}) \setminus \{0\}$. Such a countermeasure can easily be implemented in the AMNS conversion procedure. The forward and backward conversion algorithms are modified to take an extra argument used to change the representative of x and y (Algorithms 11 and 12). Hence, before the computation of kP, the procedures DPA_Conv_2_AMNS(x,u^{-2}) and DPA_Conv_2_AMNS(y,u^{-3}) are called. Once the computation is done, a call to DPA_Conv_2_BIN(x,u^2) and DPA_Conv_2_BIN(y,u^3) allows to find back the coordinates of kP.

Algorithm 11 DPA_Conv_2_AMNS(a,β)

Require: $a \in \mathbb{Z}/p\mathbb{Z}$, $\mathcal{B} = (p, n, \gamma, \rho, E)$ and $\beta \in (\mathbb{Z}/p\mathbb{Z}) \setminus \{0\}$

Ensure: $A \equiv (a\beta\phi)_{\mathcal{B}}$

1: $b = a\beta \pmod p$

2: $t = (b_{n-1}, ..., b_0)_\rho$ # radix-ρ decomposition of b

3: $U \leftarrow \sum_{i=0}^{n-1} t_i P_i(X)$

4: $A \leftarrow \text{RedCoeff}(U)$

5: return A

Algorithm 12 DPA_Conv_2_BIN(A,β)

Require: $A \in \mathcal{B}$, $\mathcal{B} = (p, n, \gamma, \rho, E)$ and $\beta \in (\mathbb{Z}/p\mathbb{Z}) \setminus \{0\}$

Ensure: $a = A(\gamma)\beta\phi^{-1} \pmod p$

1: $A \leftarrow \text{RedCoeff}(A)$

2: $a \leftarrow a_{n-1}$

3: **for** $i = n-2 \ldots 0$ **do**

4: $\quad a \leftarrow (a\gamma + a_i) \pmod p$

5: **end for**

6: $a \leftarrow a\beta \pmod p$

7: return a

7 Conclusion

In this paper, we generalised some results in [12] to a larger set of polynomials E. We presented a complete set of algorithms for arithmetic and conversion operations in the AMNS and showed how to generate all parameters needed for these algorithms. Our implementations have shown that AMNS allows to perform modular operations more efficiently than well known libraries like GNU MP and OpenSSL. Moreover, compared to OpenSSL which is faster than GNU MP when using the Montgomery multiplication algorithm, AMNS needs less memory storage. Additionally, AMNS can be much more efficient if its high parallelisation capability is used. Finally, we brought some arguments and elements to point out that AMNS should be considered as a potential countermeasure in the context of side channel attacks.

Acknowledgment

The authors would like to thank the referees for their constructive comments which helped improving the quality of the paper.

References

1. Abarzúa, R., Valencia, C., López, J.: Survey for performance and security problems of passive side-channel attacks countermeasures in ECC. Cryptology ePrint Archive, Report 2019/010 (2019)
2. Antão, S., Bajard, J.C., Sousa, L.: RNS based elliptic curve point multiplication for massive parallel architectures. The Computer Journal **55**(5), 629–647 (2012)
3. Bajard, J.C., Duquesne, S., Ercegovac, M.: Combining leak-resistant arithmetic for elliptic curves defined over f_p. Publications Mathématiques de Besançon. Algrèbre et Théorie des Nombres pp. 67–87 (2013), iSSN: 1958-7236
4. Bajard, J.C., Eynard, J., Hasan, A., Zucca, V.: 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
5. Bajard, J.C., Imbert, L.: A full RNS implementation of RSA. IEEE Transactions on Computers **53**(6), 769–774 (2004)
6. Bajard, J.C., Imbert, L., Liardet, P.Y., Teglia, Y.: Leak resistant arithmetic. In: Workshop on Cryptographic Hardware and Embedded Systems CHES 2004, Cambridge (Boston), USA. pp. 62–75. Lecture Notes in Computer Science, Springer (2004)
7. Bajard, J.C., Imbert, L., Plantard, T.: Modular number systems: Beyond the mersenne family. In: Selected Areas in Cryptography, 11th International Workshop, SAC 2004, Waterloo, Canada. pp. 159–169 (2004)
8. Bajard, J.C., Imbert, L., Plantard, T.: Arithmetic operations in the polynomial modular number system. In: 17th IEEE Symposium on Computer Arithmetic (ARITH-17) 2005, Cape Cod, MA, USA. pp. 206–213 (2005), Extended (complete) version available at: `https://hal-lirmm.ccsd.cnrs.fr/lirmm-00109201/document`
9. Baldi, M.: QC-LDPC Code-Based Cryptography. SpringerBriefs in Electrical and Computer Engineering, Springer International Publishing (2014)
10. Barrett, P.: Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In: Odlyzko, A.M. (ed.) Advances in Cryptology — CRYPTO' 86. pp. 311–323. Springer, Berlin, Heidelberg (1987)
11. Didier, L.S., Dosso, F.Y., El Mrabet, N., Marrez, J., Véron, P.: Randomization of Arithmetic over Polynomial Modular Number System. In: 26th

IEEE International Symposium on Computer Arithmetic. vol. 1, pp. 199–206. Kyoto, Japan (Jun 2019). https://doi.org/10.1109/ARITH.2019.00048
12. El Mrabet, N., Gama, N.: Efficient multiplication over extension fields. In: WAIFI. Lecture Notes in Computer Science, vol. 7369, pp. 136–151. Springer (2012)
13. El Mrabet, N., Nègre, C.: Finite field multiplication combining AMNS and DFT approach for pairing cryptography. In: ACISP. Lecture Notes in Computer Science, vol. 5594, pp. 422–436. Springer (2009)
14. Garner, H.L.: The residue number system. IRE Transactions on Electronic Computers **EL 8**(6), 140–147 (1959)
15. Gathen, J.V.Z., Hartlieb, S.: Factoring modular polynomials. Journal of Symbolic Computation **26**(5), 583 – 606 (1998)
16. Goubin, L.: A refined power-analysis attack on elliptic curve cryptosystems. In: International Workshop on Public Key Cryptography. pp. 199–211. Springer (2003)
17. Granlund, T., al.: GNU multiple precision arithmetic library 6.1.2, `https://gmplib.org/`
18. 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)
19. Johnston, A.M.: A generalized qth root algorithm. In: Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 929–930. SODA '99, Society for Industrial and Applied Mathematics (1999)
20. Joye, M., Tymen, C.: Protections against differential analysis for elliptic curve cryptography — an algebraic approach —. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems — CHES 2001. pp. 377–390. Springer, Berlin, Heidelberg (2001)
21. Kocher, P.C.: Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: Koblitz, N. (ed.) Advances in Cryptology — CRYPTO '96. pp. 104–113. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)
22. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: CRYPTO. Lecture Notes in Computer Science, vol. 1666, pp. 388–397. Springer (1999)
23. Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische Annalen **261**(4), 515–534 (Dec 1982)
24. Mishra, B.: Algorithmic Algebra. Springer-Verlag, Berlin, Heidelberg (1993)
25. Montgomery, P.L.: Modular multiplication without trial division. Mathematics of Computation **44**(170), 519–521 (1985)
26. Negre, C., Plantard, T.: Efficient modular arithmetic in adapted modular number system using lagrange representation. In: Information Security and Privacy, 13th Australasian Conference, ACISP 2008, Wollongong, Australia. pp. 463–477 (2008)
27. Plantard, T.: Arithmétique modulaire pour la cryptographie. Ph.D. thesis, Montpellier 2 University, France (2005)
28. Project, T.O.: Openssl, `https://www.openssl.org/`
29. Shoup, V., al.: Ntl: A library for doing number theory, `https://www.shoup.net/ntl/`
30. Solinas, J., Fu, D.E.: Elliptic Curve Groups modulo a Prime (ECP Groups) for IKE and IKEv2. RFC 5903 (Jun 2010). https://doi.org/10.17487/RFC5903, `https://rfc-editor.org/rfc/rfc5903.txt`
31. Stein, W., al.: Sagemath, `http://www.sagemath.org/index.html`

Appendix A List of prime numbers used for Table 3

- $p192 = 0xE06F20509A52674228D4F0701A08EB3B08C1714F0A93F719$
- $p224 = 0xE886C555B533B33B037F4F356CB97E00B560DD1B5A9C252CCEAF301B$
- $p256 = 0x8FFB5E3E4BD153C220C28FDBA587F9C23D454DBE31C17D0B44462E26684B46E5$
- $p384 = 0xF3D1CD992E8EA43D29612F131C05A03215F247E92951AB3D741FEA820526FD185CDBEC7AEFC31F75BEA2D2F4F43D1547$
- $p521 = 0x15683E5BD61DA4E3A10A95DE122E3B015FAC3F355F6360F33FA19D036CA02897BAF3D615ADAF6508A1E5B325B0345F39505A7B84ED01A8F913CA0D6395A9E135BE3$

Appendix B Examples of AMNS for different primes

In this section, we give some examples of the AMNS we used in Section 6.4.1 for our numerical experiments. All these AMNS have the common parameter $\phi = 2^{64}$.

B.1 AMNS 1: 192-bit prime number.

- $p = 0xE06F20509A52674228D4F0701A08EB3B08C1714F0A93F719$
- $n = 4$
- $\lambda = -1$
- $\rho = 2^{51}$
- $\gamma = 0x7AB09A124AA5065B2E20034E0D0FE3D0A5F2A276C33E2515$
- $E(X) = X^4 + 1$
- $M(X) = 0x4B3D12868945.X^3 - 0x924097D431D8.X^2 + 0x39B561D62725.X + 0xC580DC0A05E3$
- $M'(Y) = 0x6E2B6D9BAF275F4F.Y^3 + 0x8F59D05762288B18.Y^2 + 0x69A1F846105E39CF.Y + 0xBEDE53CF67CF2747$

B.2 AMNS 2: 224-bit prime number.

- $p = 0xE886C555B533B33B037F4F356CB97E00B560DD1B5A9C252CCEAF301B$
- $n = 4$
- $\lambda = -2$
- $\rho = 2^{60}$
- $\gamma = 0x64892FE7A2B9E28E496952B025FE138C2238260106F31C90E9354AFEF$
- $E(X) = X^4 + 2$
- $M(X) = -0x6A2300C9FAC40E.X^3 - 0xE12EC6DCB579A6.X^2 - 0x272839DE2E827E.X - 0x43419ADAFCFB61$
- $M'(Y) = 0x7D4F705603D9CE42.Y^3 + 0xE0922181D0445FA6.Y^2 + 0x5A4FA29325678B32.Y + 0xDDDE890AB0458D59$

B.3 AMNS 3: 256-bit prime number.

- $p = 0x8FFB5E3E4BD153C220C28FDBA587F9C23D454DBE31C17D0B44462E26684B46E5$
- $n = 5$
- $\lambda = 2$
- $\rho = 2^{55}$
- $\gamma = 0x42559355ED8CAAA92688CE0A9322458EE43724D997327755F385B1901F25E507$
- $E(X) = X^5 - 2$
- $M(X) = -0x7F360937497B.X^4 - 0x45FB30302B149.X^3 - 0x1910C5989E6B8.X^2 - 0x28750BDCB9CA3.X + 0x3935AF11550E5$
- $M'(Y) = 0x6AC1B8BE18685FC6.Y^4 + 0x1E8123E1FA66C4B2.Y^3 + 0x5C7430F9C82014D1.Y^2 + 0x33A24848D6BF6427.Y + 0xCC7C0CE54B67A803$

B.4 AMNS 4: 384-bit prime number.

- $p = 0xF3D1CD992E8EA43D29612F131C05A03215F247E92951AB3D741FEA820526FD185CDBEC7AEFC31F75BEA2D2F4F43D1547$
- $n = 7$
- $\lambda = 2$
- $\rho = 2^{59}$
- $\gamma = 0xA5C4FB2BBF7D447D0E58D14E3F440AD5C7A0BB773BCFA856914ED875B1A8B3DD5C6327E24B34890BDA7782DE3050EEC4$
- $E(X) = X^7 - 2$
- $M(X) = 0x2B70420C25B6F9.X^6 + 0x27597E8FAEFBA6.X^5 + 0x2A259AA4E719E1.X^4 + 0x12391F5D00D4A7.X^3 - 0x26AC55039EACFD.X^2 + 0x2747CE657C0F2D.X - 0x426A85C33ACE17$
- $M'(Y) = 0x36E06AB70DC02E0C.Y^6 + 0x91EC3470F30AB1DD.Y^5 + 0x521BCB522168C88C.Y^4 + 0x51579EF6AC4A01C8.Y^3 + 0x7145B435BA15791A.Y^2 + 0xCCD28607261C6227.Y + 0x4E6A294F1FBE2093$

B.5 AMNS 5: 521-bit prime number.

- $p = 0x15683E5BD61DA4E3A10A95DE122E3B015FAC3F355F6360F33FA19D036CA02897BAF3D615ADAF6508A1E5B325B0345F39505A7B84ED01A8F913CA0D6395A9E135BE3$
- $n = 10$
- $\lambda = -2$
- $\rho = 2^{57}$
- $\gamma = 0x3BEB85F1AC84420C044C472B8845A1896C68ACD6C78773C9392B6CE871027BD5C333EF238A11733384E0A7318139218D99ADDCBB39694C1207938B6CA6789BC3B1$
- $E(X) = X^{10} + 2$
- $M(X) = -0x3D52F259CF52C.X^9 - 0x2F155A2F83CC6.X^8 + 0x3C5398A0AA3D2.X^7 - 0x6161944D2155C.X^6 + 0x92266960FE012.X^5 - 0x68DFAA2817992.X^4 - 0x996D8B98C7860.X^3 - 0x31E83951B9F38.X^2 + 0x3E716C4C0B2A4.X + 0x3304421CB90FD$
- $M'(Y) = 0xBA9CFB5216CEA3CC.Y^9 + 0x4DD219C801C0DD06.Y^8 + 0x10DEC022F71CC8F2.Y^7 + 0x199161BB290DEE2C.Y^6 + 0x924D10687452E482.Y^5 + 0x7F6A883FEED1B396.Y^4 + 0x6923B242682C1CA0.Y^3 + 0x76FA75CEF1B36AC8.Y^2 + 0xBD1EDFD16FA95474.Y + 0xC7E79022CD8CD813$

Appendix C Examples of AMNS for the same prime

In Section 6.5.3, we said that the existence of many AMNS for a given prime could be used to randomise data. Here, we give three examples of AMNS for the prime $p = 2^{255} + 95$. We also give representatives of three random elements of $\mathbb{Z}/p\mathbb{Z}$ in these AMNS.

C.1 The AMNS

Common parameters:

- $p = 2^{255} + 95$
- $\phi = 2^{64}$
- $n = 5$

C.1.1 AMNS 1.

- $\lambda = 2$
- $\rho = 2^{55}$
- $\gamma = 0x4A11EC963214E75587B184AF9B09E8871D0DF5991483661DE2FF6BB1E251199C$
- $E(X) = X^5 - 2$
- $M(X) = 0x28AE865829ED0.X^4 + 0x3B47735E8CB55.X^3 - 0x1337D2969BC11.X^2 + 0x46647D3BC6C24.X - 0x2B2A32D7CA88B$
- $M'(Y) = 0x705370302B557A79.Y^4 + 0xF4EF33F4C4A73DDD.Y^3 + 0x35A8B6E9AE5BB345.Y^2 + 0xB1EAB7F74DA8C6B4.Y + 0xF83A6F9196747A23$

C.1.2 AMNS 2.

- $\lambda = 4$
- $\rho = 2^{56}$
- $\gamma = 0x4FB25BB223F254D0EC52A2EE155F444C45582C268782AEE4D4E9FCA973434A6C$
- $E(X) = X^5 - 4$
- $M(X) = -0x38B51AD5722AE.X^4 + 0x53FB8DAF6F024.X^3 + 0x35A85724CB9CE.X^2 - 0x3D243A4DF4584.X - 0x117F860FE1135$
- $M'(Y) = 0x403B2C2CE09E21F6.Y^4 + 0xB893AB63E6BC1344.Y^3 + 0x35C181058EB18F0E.Y^2 + 0xA4AED1FFC25D5C5C.Y + 0x6FC13791D5CE795D$

C.1.3 AMNS 3.

- $\lambda = -3$
- $\rho = 2^{56}$
- $\gamma = 0x1EBF5A56EC92F9F46C7F0870E5E3702D3E8383DEAF56E4B4C3D368BD0BF3BD40$
- $E(X) = X^5 + 3$
- $M(X) = -0x1C961F979254D.X^4 + 0x1D9EAFCB6057C.X^3 - 0x3CE080AECD314.X^2 - 0x539D41F2093E8.X + 0x709FEB927094$
- $M'(Y) = 0x4C53B117C5A624FC.Y^4 + 0x6F5067DF289E2148.Y^3 + 0x4D82701329D99964.Y^2 + 0x1194DEB36C42D649.Y + 0x823B9BE066BDC6EC$

C.2 The representatives

Here, we give representatives for three elements of $\mathbb{Z}/p\mathbb{Z}$ in the preceding AMNS. Let w_1, w_2 and w_3 be three elements of $\mathbb{Z}/p\mathbb{Z}$, such that:

$w_1 = 0x413F124E07F832A9615B0F4DF8839FB84654F83$
$EFBE271109B37B5FF3C45F86B$

$w_2 = 0x1D52208BBA6F67BDAB73B52C108E35297$
$D77D319A2B960774879AD379A9EFA2C$

$w_3 = 0x51D89683548C1AB20A5DD1B6ED40D27539$
$9CACB8099775C365EAB9E643D0188B$

C.2.1 Representatives of w_1

Some representatives of w_1 in the AMNS above are:

- In AMNS 1: $0x1D919BA97C9EE.X^4 + 0x8D2E4EA9D2522.X^3 + 0x387177645E956.X^2 + 0x8B7F74DE7D127.X + 0x7DDB08BBFB2D4$
- In AMNS 2: $0xC2BDA39B9CAA.X^4 - 0x5C79DF0B874A7.X^3 + 0x38E436F97A141.X^2 + 0x83CD4D9F668F0.X + 0x889ED1F53B42$
- In AMNS 3: $-0xE6E254EE2A56.X^4 + 0x1B69F43ED2D64.X^3 - 0x910A16595CB6C.X^2 + 0x3C2140E66E677.X + 0xDCAABE99C9D26$

C.2.2 Representatives of w_2

Some representatives of w_2 in the AMNS above are:

- In AMNS 1: $0x5D790C3E1A61C.X^4 + 0x70948EA695150.X^3 + 0xA169F530662D0.X^2 + 0xAF3DC447BD060.X + 0xDB6398B75B911$
- In AMNS 2: $0x2C77ABDC5D961.X^4 - 0x829EAF67CA083.X^3 + 0xDFB7B8EBFB188.X^2 + 0xB64039D6B0FC.X - 0x8087A5968F930$
- In AMNS 3: $-0x299A203AE211F.X^4 + 0xE944E22B7F0C.X^3 - 0x6E0B58CF0B1C4.X^2 + 0x54687E54FF785.X + 0x13E4F275D429CF$

C.2.3 Representatives of w_3

Some representatives of w_3 in the AMNS above are:

- In AMNS 1: $0x26CE79EBC2B79.X^4 + 0x58D977D7CBF80.X^3 + 0x8344B92319D5F.X^2 + 0x469C2152EEA87.X + 0xB6CE8E4FA85CF$
- In AMNS 2: $0x37D32EF24504E.X^4 - 0x3D646DBF95505.X^3 + 0xA0F25DE041BA3.X^2 - 0x32F2DBD31EB84.X - 0xA65A7B1EB4B8A$
- In AMNS 3: $-0x5E2A89914C89F.X^4 - 0x7EDD111585E3.X^3 - 0x3BAB02C97D067.X^2 - 0x20F6AFC3EC4DA.X + 0xFB824ADD2ECF8$

Appendix D Integers structures in GNU MP and OpenSSL

In this section, we give the integer structures in GNU MP and OpenSSL. These are the structures we used to compute memory consumptions in Table 6.

GNU MP mpz_t structure:

```
typedef struct
{
  int _mp_alloc;   /* Number of *limbs* allocated and
                   pointed to by the '_mp_d' field. */
  int _mp_size;    /* abs(_mp_size) is the number of
                   limbs the last field points to.
                   If _mp_size is negative this
                   is a negative number. */
  mp_limb_t *_mp_d;   /* Pointer to the limbs. */
} __mpz_struct;
```

In GNU MP mpz_t structure, there are 2 integers of type *int* and an array of type *mp_limb_t*. On the computer we used for our tests (see features at Section 6.4), *int* is 32 bits wide and *mp_limb_t* is 64 bits wide. In the computation of memory consumption, we considered the 2 integers of type *int* as one integer of 64 bits. For more details, see: `https://gmplib.org/manual/Integer-Internals.html`.

OpenSSL bignum_st structure:

```
struct bignum_st
{
   BN_ULONG *d; /* Pointer to an array of 'BN_BITS2'
                 bit chunks. */
   int top;     /* Index of last used d +1. */
/* The next are internal book keeping for bn_expand. */
   int dmax;    /* Size of the d array. */
   int neg;     /* one if the number is negative */
   int flags;
};
```

In OpenSSL bignum_st structure, there are 4 integers of type *int* and an array of type *BN_ULONG*, which is 64 bits wide (on our computer). In memory consumption computation, we considered the 4 integers of type *int* as two 64-bit integers. For more details, see: `https://linux.die.net/man/3/bn_internal`.