UNIVERSITÉ DE TOULON

ÉCOLE DOCTORALE 548 - MER ET SCIENCES
Laboratoire IMATH

THÈSE présentée par :

François PALMA

soutenue le : **12 décembre 2025**

pour obtenir le grade de Docteur en Informatique

Arithmétique modulaire pour la cryptographie à clé publique

THÈSE dirigée par :
M. VÉRON Pascal Maître de conférences HDR, Université de Toulon
THÈSE co-encadrée par :
M. MÉLONI Nicolas Maître de conférences, Université de Toulon

JURY :

Mme GUILLEVIC Aurore	Chargée de recherche Inria, Centre Inria Rennes, Rapporteure
M. PLANTARD Thomas	Chargé de recherche Nokia Bell Labs, Murray Hill, Rapporteur
M. BALLET Stéphane	Professeur des Universités, Aix-Marseille Université, Examinateur
M. IMBERT Laurent	Directeur de recherche CNRS, LIRMM Montpellier, Examinateur
M. LANGEVIN Philippe	Professeur des Universités, Université de Toulon, Examinateur
M. LERCIER Reynald	Ingénieur de l'État HDR, DGA & Université de Rennes, Examinateur

Remerciements

Je tiens tout d'abord à exprimer ma profonde gratitude envers Pascal Véron et Nicolas Méloni qui ont encadré cette thèse avec bienveillance et rigueur. Leur patience, leur disponibilité et la qualité de leurs conseils m'ont accompagné tout au long de ces années. C'est grâce à leur encouragement que j'ai envisagé, puis entrepris, cette aventure scientifique, et je les remercie sincèrement de m'avoir donné confiance en ce chemin.

Je remercie également Aurore Guillevic et Thomas Plantard d'avoir accepté d'évaluer ce travail en tant que rapporteurs et pour leurs commentaires et suggestions éclairés, ainsi que Stéphane Ballet, Laurent Imbert, Philippe Langevin et Reynald Lercier pour avoir accepté de faire partie du jury de soutenance.

Ma reconnaissance va également à Fabien Herbaut, Yves Aubry, Valérie Gillot, Jean-Pierre Zanotti, Jean-Marc Robert, Laurent-Stéphane Didier et tous les autres membres du laboratoire IMATH de l'université de Toulon pour leur accueil, leur soutien et la qualité des échanges. J'ai eu la chance d'évoluer dans un environnement à la fois stimulant et chaleureux, et je leur en suis profondément reconnaissant.

Merci aux doctorants du bâtiment M pour ces trois années partagées dans la bonne humeur. Un merci tout particulier à Camille et Yolhan pour leurs conseils précieux au début de ma thèse, qui m'ont aidé à trouver mes repères. Merci aussi à Ramzi, Léa, Léo, Alioune, Giacomo, Lorenzo, ainsi qu'à Juliette, Bilal, Cristiam et Martin pour leur présence, leur amitié et tous les moments partagés.

Enfin, je ne saurais trop remercier ma mère, véritable pilier durant les périodes difficiles, ainsi que ma sœur, dont la présence et le soutien ont compté plus que je ne saurais le dire.

Table des matières

Introduction

Internet, la frontière de l'infini. Jusqu'à 1994, la question de sécuriser les communications par Internet ne se posait guère. Conçu initialement pour faciliter la communication entre chercheurs, puis adopté par des hobbyistes de toutes envergures pour se retrouver sur des forums d'intérêts communs, l'enjeu de la protection des données sensibles ne s'était pas encore présenté dans l'esprit du grand public. Avec la bulle Internet des années 90 et la prolifération du commerce en ligne, cette problématique n'a pas tardé à être abordée. Ironiquement, de façon parallèle en 1994, Peter Shor établissait son algorithme de factorisation des nombres premiers nécessitant un ordinateur quantique, menaçant dès lors la robustesse des protocoles cryptographiques qui seraient proposés dans les années à venir.

Depuis 2018, la question ne se pose plus, la quasi-intégralité du trafic internet utilise le protocole HTTPS (HyperText Transfer Protocol Secure) dont la sécurité repose sur TLS (Transport Layer Security). Avant d'établir un canal de communication sécurisé, une "poignée de main" (*Handshake*) est échangée entre le client et le serveur. Une fois ce canal établi, les données transitent de manière confidentielle grâce à la cryptographie symétrique. Celle-ci, dite aussi à clé privée, repose sur l'existence d'un secret commun entre les parties. C'est dans ce contexte que l'intérêt de la phase de poignée de main du protocole s'explique. La cryptographie à clé publique, appelée aussi cryptographie asymétrique, permet de résoudre les problèmes fondamentaux de bootstrapping inhérents à la cryptographie symétrique : comment établir un secret commun de manière sécurisée sans déjà disposer d'un secret commun pour sécuriser cet échange ?

L'échange de clés de Diffie et Hellman est la réponse historique à cette question. Alice et Bob font usage de la clé publique l'un de l'autre afin de générer un secret partagé qui facilitera tout échange de secret dans le futur. À l'heure actuelle, le protocole initialement proposé a été supplanté par une version se basant sur la cryptographie sur les courbes elliptiques, un point commun de ces deux méthodes étant qu'elles reposent sur l'arithmétique modulaire.

L'arithmétique modulaire englobe toute forme d'arithmétique sur les anneaux finis, les corps finis, ou encore les extensions de corps finis. De fait, la majorité des protocoles cryptographiques utilisés historiquement sur Internet font usage de l'arithmétique modulaire, surtout du côté de la cryptographie asymétrique. Or, l'avènement potentiel de l'informatique quantique remet en cause cette ubiquité.

En contraste la cryptographie post-quantique, conçue pour résister aux attaques des ordinateurs quantiques, fait toujours usage de l'arithmétique modulaire mais dans une moindre mesure. Parmi les finalistes de la compétition du NIST (National Institute of Standards and Technology) pour sélectionner les protocoles

asymétriques post-quantiques à standardiser pour utilisation dans le futur, seuls Kyber [7] et Dilithium [21] ressortent comme protocoles reposant sur l'arithmétique modulaire, cette fois dans les réseaux euclidiens. Un renouveau d'intérêt envers la cryptographie basée sur les isogénies de courbes elliptiques a eu lieu pour les candidats supplémentaires en termes de protocoles de signature (tel que SQIsignHD [13]). Les protocoles basés sur les isogénies restent toutefois marqués par la fragilité de SIKE [33], initialement finaliste mais complètement cassé quelques mois après sélection, ce qui a jeté un doute durable sur la maturité de cette approche en termes de sécurité.

Toutefois l'intérêt autour de l'arithmétique modulaire reste fort dans le domaine et en particulier avec les protocoles hybrides, dont le principe consiste à combiner un protocole post-quantique avec la cryptographie basée sur les courbes elliptiques. Son utilité semble donc appelée à perdurer, même à l'ère quantique.

Le Polynomial Modular Number System est un système de représentation non positionnel proposé en 2004 par Jean-Claude Bajard, Laurent Imbert et Thomas Plantard afin d'accélérer les opérations d'arithmétique modulaire. Il était déjà établi que l'étape de réduction modulaire pouvait être une avenue d'amélioration. Outre le fait de choisir des modulos de formes particulières proches d'une puissance de 2, de nombreuses méthodes s'étaient dégagées pour l'éventualité où l'entier premier ne pouvait pas être de forme particulière. La possibilité d'utiliser un système de représentation particulier avait déjà été explorée avec le RNS (Residue Number System) proposé en 1959 et basé sur le théorème des restes chinois. Le PMNS quant à lui se base entièrement sur les opérations polynomiales modulaires.

Ce document est composé de trois parties.

La première partie traite de l'état de l'art concernant l'arithmétique modulaire. Le chapitre 1 résume les points clés sur l'arithmétique modulaire sur les entiers, les polynômes et les réseaux euclidiens avant de poser les bases des PMNS.

La deuxième partie donne un tour d'horizon sur des améliorations générales concernant les PMNS. Le chapitre 2 traite de l'utilisation de polynômes non unitaires à des fins de réduction modulaire, propose une optimisation sur le nombre d'additions nécessaires lors d'une opération de multiplication vecteur-matrice dans le cas où la matrice est de Toeplitz, présente une nouvelle borne plus fine sur le nombre de coefficients nécessaires pour représenter des entiers en PMNS et établit de nouveaux algorithmes de conversion et de test d'égalité.

La troisième partie a pour objet les améliorations sur les PMNS spécifiques à des cas d'utilisations précis. Dans le chapitre 3 est détaillé l'ensemble des méthodes considérées pour effectuer des opérations efficaces sur les grands entiers (de taille supérieure à 1024 bits), y compris l'utilisation de diverses formes de calcul parallèle. Enfin, le chapitre 4 propose une forme particulière d'entiers premiers permettant des opérations très rapides lorsqu'ils sont utilisés pour construire des PMNS.

Chapitre 1

État de l'art

Notations :
— $\mathbb{Z}_m[X]$ le $\mathbb{Z}$-module des polynômes à coefficients entiers de degré au plus m.
— $\nu_n : \mathbb{Z}_{n-1}[X] \to \mathbb{Z}^n$ l'isomorphisme tel que $\nu_n(a_0 + a_1X + \cdots + a_{n-1}X^{n-1}) = (a_0, a_1, \ldots, a_{n-1})$ pour un n donné.
— $\pi_n : \mathbb{Z}^n \to \mathbb{Z}_{n-1}[X]$ l'isomorphisme tel que $\pi_n((a_0, a_1, \ldots, a_{n-1})) = a_0 + a_1X + \cdots + a_{n-1}X^{n-1}$ pour un n donné.
— σ le nombre de valeurs possibles dans un registre mémoire (2^{64} pour la plupart des CPU modernes).
— $M_{i,j}$ pour M une matrice représente le coefficient situé sur la $i$ème ligne et la $j$ème colonne, en partant de 0. Une matrice M de dimension $n \times m$ aura donc des coefficients de $M_{0,0}$ à $M_{n-1,m-1}$.
— $\lfloor a \rceil$ pour $a \in \mathbb{R}$ représente la valeur arrondie de a, équivalente à $\lfloor a + \frac{1}{2} \rfloor$.
— Pour $v = (v_0, v_1, \ldots, v_{n-1}) \in \mathbb{R}^n$, on aura $\lfloor v \rceil = (\lfloor v_0 \rceil, \lfloor v_1 \rceil, \ldots, \lfloor v_{n-1} \rceil)$.
— Pour $M \in \mathcal{M}_n(\mathbb{R})$, on notera $\lfloor M \rceil$ la matrice telle que $\forall i, j \in [\![0, n-1]\!] \times [\![0, n-1]\!]$, $\lfloor M \rceil_{i,j} = \lfloor M_{i,j} \rceil$.

1.1 Introduction

Ce chapitre présente un état de l'art sur un certain nombre de domaines de l'arithmétique des ordinateurs, notamment l'arithmétique sur les entiers, l'arithmétique sur les polynômes, l'arithmétique sur les réseaux euclidiens et, pour conclure, un état de l'art sur le Polynomial Modular Number System (PMNS) qui est au cœur des travaux de cette thèse.

1.2 Arithmétique sur les entiers

La représentation des entiers dans l'arithmétique des ordinateurs est le plus souvent effectuée en rangeant la représentation binaire correspondante dans des registres mémoire pouvant contenir des multiples de 8 bits, selon l'architecture considérée. Pour simplifier, on notera σ le nombre de valeurs possibles dans un registre mémoire et pour nos exemples les registres utilisés seront souvent sur 64 bits au vu des capacités des processeurs actuels sur le marché.

On appelle entier en simple-précision un entier qui se représente sur un seul registre mémoire. Sa représentation binaire dans le registre est donc comme suit :

$$\overbrace{000\ldots0}^{\text{padding}}\underbrace{110\ldots010}_{\text{entier}}$$

avec le padding correspondant aux 0 ajoutés pour que tous les bits du registre aient une valeur. Cette représentation permet de stocker des entiers ayant des valeurs allant de 0 à $\sigma - 1$. Si l'on veut représenter des entiers négatifs, la solution employée sur un seul registre consiste à représenter σ moins la valeur absolue de cet entier. Toutes les opérations sur un seul registre sont naturellement effectuées modulo σ en pratique donc le résultat des opérations sera correct de ce point de vue. Cette méthode permet de représenter des entiers compris entre $-\frac{\sigma}{2}$ et $\frac{\sigma}{2}-1$. Si l'on emploie plutôt un bit de signe dans un registre auxiliaire, on peut représenter des valeurs comprises entre $-\sigma+1$ et $\sigma-1$ mais cela sous-entend la gestion de ce registre auxiliaire tout au long des opérations, sans compter la distinction entre -0 et $+0$ donc cette méthode n'est pas souvent utilisée.

On appelle entier multi-précision un entier dont la valeur est supérieure à σ et nécessite donc plusieurs registres mémoire pour le contenir. L'organisation en mémoire en représentation binaire est organisée comme suit

$$\underbrace{\overbrace{000\ldots0}^{\text{padding}}101\ldots101}_{\text{registre de poids fort}}\quad\underbrace{1101\ldots0100}_{\text{deuxième registre de poids fort}}\quad\ldots\quad\underbrace{0010\ldots1011}_{\text{registre de poids faible}}$$

avec les bits à 0 au début s'il n'existe pas de k tel que l'entier considéré soit dans l'intervalle $[\![\frac{\sigma^k}{2}, \sigma^k - 1]\!]$.

Soit a un entier multi-précision, on notera a_0 son registre de poids faible, a_1 le registre suivant et ainsi de suite.

Il est possible ici, de façon similaire aux entiers simple-précision, de considérer σ^k moins la valeur absolue de l'entier, pour un entier de valeur absolue strictement inférieure à σ^k, lorsque l'on veut représenter un entier négatif en multi-précision. Cependant, les processeurs ne gèrent pas nativement la multi-précision et donc les opérations devront être organisées au niveau logiciel. Pour cette raison, le bit de signe dans une variable auxiliaire est la solution la plus couramment employée dans ce cas.

Pour commencer, nous détaillons l'algorithme d'addition sur des entiers multi-précisions.

L'algorithme 1 peut facilement être adapté pour des entiers relatifs. On peut également le modifier pour effectuer une soustraction au lieu d'une addition. Par souci de simplicité, ces adaptations ne seront pas explorées ici. Comme on peut le voir, la complexité des opérations est linéaire en $n + m$.

Ensuite, nous détaillons l'algorithme de multiplication d'entiers multi-précisions dit "schoolbook" en raison de ses similitudes avec l'algorithme de multiplication élémentaire enseigné à l'école primaire.

L'algorithme 2 correspond à la multiplication multi-précision schoolbook. Il s'agit d'un algorithme de complexité en $O(nm)$ donc quadratique en la taille mémoire des opérandes. Cet algorithme peut être facilement adapté pour la gestion du signe avec des opérandes signées. En effet, si par exemple a est négatif, il suffit d'utiliser $-a$ comme opérande et de considérer le résultat comme négatif.

Algorithme 1 Addition multi-précision

Require: $a \in \mathbb{N}$ et $b \in \mathbb{N}$ les opérandes, $m \in \mathbb{N}$ la taille mémoire de a et $n \in \mathbb{N}$ celle de b avec $m \geqslant n$.
Ensure: c tel que $c = a + b$ et o la taille mémoire de c

$o \leftarrow m + 1$
$c \leftarrow (0, 0, \ldots, 0)$ # o registres à 0
for $i = 0..n - 1$ **do**
    if $c_i + a_i + b_i \geqslant \sigma$ **then** # Passage de retenue
        $c_{i+1} \leftarrow 1$
    end if
    $c_i \leftarrow c_i + a_i + b_i \bmod \sigma$
end for
for $i = n..m - 1$ **do**
    if $c_i + a_i \geqslant \sigma$ **then**
        $c_{i+1} \leftarrow 1$
    end if
    $c_i \leftarrow c_i + a_i \bmod \sigma$
end for
while $o \geqslant 1$ **and** $c_{o-1} = 0$ **do**
    $o \leftarrow o - 1$
end while
return c, o

Algorithme 2 Multiplication multi-précision schoolbook

Require: $a \in \mathbb{N}$ et $b \in \mathbb{N}$ les opérandes, $m \in \mathbb{N}$ la taille mémoire de a et $n \in \mathbb{N}$ celle de b.
Ensure: c tel que $c = a \times b$ et o la taille mémoire de c

$o \leftarrow m + n$
$c \leftarrow (0, 0, \ldots, 0)$ # o registres à 0
for $i = 0..m - 1$ **do**
    for $j = 0..n - 1$ **do**
        $d_1, d_0 \leftarrow a_i \times b_j$ # $d = a_i \times b_j$ sur 2 registres
        if $c_{i+j} + d_0 \geqslant \sigma$ **then**
            $d_1 \leftarrow d_1 + 1$
        end if
        $c_{i+j} \leftarrow c_{i+j} + d_0 \bmod \sigma$
        if $c_{i+j+1} + d_1 \geqslant \sigma$ **then** # Passage de retenue
            $k \leftarrow 2$
            while $c_{i+j+k} + 1 = \sigma$ **do**
                $c_{i+j+k} \leftarrow 0$
                $k \leftarrow k + 1$
            end while
            $c_{i+j+k} \leftarrow c_{i+j+k} + 1$
        end if
        $c_{i+j+1} \leftarrow c_{i+j+1} + d_1 \bmod \sigma$
    end for
end for
while $o \geqslant 1$ **and** $c_{o-1} = 0$ **do**
    $o \leftarrow o - 1$
end while
return c, o

Pour contraster avec l'algorithme schoolbook, nous présentons ensuite un algorithme de complexité sous-quadratique.

Algorithme 3 KaraMul : Multiplication multi-précision de Karatsuba récursive

Require: $a \in \mathbb{N}$ et $b \in \mathbb{N}$ les opérandes, $n \in \mathbb{N}$ la taille mémoire de a et de b.
Ensure: c tel que $c = a \times b$ et o la taille mémoire de c (si $n > 1$)

1: **if** $n = 1$ **then**
2: $c_1, c_0 \leftarrow a \times b$ # $c = a \times b$ sur 2 registres
3: **return** c
4: **end if**
5: $o \leftarrow 2n$
6: $c \leftarrow (0, 0, \ldots, 0)$ # o registres à 0
7: $m \leftarrow \lfloor \frac{n}{2} \rfloor$
8: $a_{lo} \leftarrow m$ premiers registres de a
9: $b_{lo} \leftarrow m$ premiers registres de b
10: $a_{hi} \leftarrow n - m$ registres de poids fort de a
11: $b_{hi} \leftarrow n - m$ registres de poids fort de b
12: $l \leftarrow$ **KaraMul**(a_{lo}, b_{lo}, m)
13: $d \leftarrow$ **KaraMul**$(a_{hi} - a_{lo}, b_{hi} - b_{lo}, n - m)$
14: $h \leftarrow$ **KaraMul**$(a_{hi}, b_{hi}, n - m)$
15: **for** $i = 0..2m - 1$ **do**
16: $c_i \leftarrow l_i$
17: **end for**
18: **for** $i = 0..2(n - m) - 1$ **do**
19: $c_{2m+i} \leftarrow h_i$
20: **end for**
21: **for** $i = 0..2(n - m) - 1$ **do**
22: **if** $c_{i+m} + d_i \geqslant \sigma$ **then** # Passage de retenue
23: $k \leftarrow 1$
24: **while** $c_{i+m+k} + 1 = \sigma$ **do**
25: $c_{i+m+k} \leftarrow 0$
26: $k \leftarrow k + 1$
27: **end while**
28: $c_{i+m+k} \leftarrow c_{i+m+k} + 1$
29: **end if**
30: $c_{i+m} \leftarrow c_{i+m} + d_i \bmod \sigma$
31: **end for**
32: **while** $o \geqslant 1$ **and** $c_{o-1} = 0$ **do**
33: $o \leftarrow o - 1$
34: **end while**
35: **return** c, o

L'algorithme 3 détaille la méthode de Karatsuba [34] pour effectuer une multiplication dont la complexité est sous-quadratique en la taille des opérandes. Pour simplification, on considère des opérandes de même taille mais cet algorithme peut être facilement adapté si les opérandes sont de tailles différentes. Le principe est de découper les opérandes a et b en $a = a_{hi} \times 2^k + a_{lo}$ et b en $b = b_{hi} \times 2^k + b_{lo}$ avec $k \in \mathbb{N}^*$. Le résultat escompté est

$$c = (a_{hi} \times b_{hi})2^{2k} + (a_{hi} \times b_{lo} + a_{lo} \times b_{hi})2^k + a_{lo} \times b_{lo}.$$

Ainsi, si l'on calcule plutôt $d = (a_{hi} - a_{lo}) \times (b_{hi} - b_{lo})$, on a

$$c = (a_{hi} \times b_{hi})2^{2k} + (a_{hi} \times b_{hi} + a_{lo} \times b_{lo} - d)2^k + a_{lo} \times b_{lo}$$

ce qui ne requiert que 3 multiplications au lieu de 4, en contrepartie d'additions et de soustractions supplémentaires. Ce processus peut être répété de façon récursive. Ainsi, pour n une puissance de 2, une multiplication multi-précision pourra être effectuée en seulement $n^{\log_2(3)}$ multiplications élémentaires au lieu des n^2 multiplications élémentaires de l'algorithme schoolbook. On note que pour des petites tailles le surcoût des additions/soustractions supplémentaires à effectuer prend le dessus sur le gain asymptotique et donc l'algorithme schoolbook s'avère plus rapide.

Comme exemple concret, la bibliothèque GNU Multiple Precision Arithmetic Library (GMP) [60], souvent utilisée dans des contextes cryptographiques et d'applications numériques haute performance, commence à privilégier l'algorithme de Karatsuba (appelé Toom-Cook 22 dans le code source) par rapport à l'algorithme schoolbook pour un seuil d'opérandes contenues sur 26 registres mémoire pour une architecture Skylake x86_64 [25]. Lorsque le nombre de registres augmente il existe d'autres méthodes plus efficaces pour effectuer une multiplication. Le lecteur intéressé peut consulter [9].

Ensuite nous présentons un algorithme de division adapté à la multi-précision.

Algorithme 4 Division du paysan russe

Require: $a \in \mathbb{N}$ et $b \in \mathbb{N}$ les opérandes.
Ensure: q, r tels que $a = q \times b + r$

```
1:  q ← 0
2:  r ← a
3:  m ← 0
4:  c ← b
5:  while c < a do
6:      c ← c × 2
7:      m ← m + 1
8:  end while
9:  while c ⩾ b do
10:     if c ⩽ r then
11:         r ← r − c
12:         q ← q + 2^m
13:     end if
14:     c ← c/2
15:     m ← m − 1
16: end while
17: return q, r
```

L'algorithme 4 reprend la méthode du paysan russe pour effectuer une division. Par souci de lisibilité, la gestion des registres mémoire n'est pas détaillée. L'idée principale est d'annuler les bits du dividende par soustractions consécutives par un multiple du diviseur. Ainsi, cet algorithme effectue au plus $\log_2(a) - \log_2(b)$ soustractions multi-précisions pour a et b les opérandes correspondantes. À cela se rajoute un nombre similaire de décalages binaires et d'additions de puissances de 2 pour déterminer le quotient. Pour n la taille mémoire de a, on a $\log_2(a) \leqslant \log_2(\sigma) \times n$ et donc il faut au plus $n \log_2(\sigma)$ soustractions multi-précisions, elles-mêmes

correspondant à au plus n soustractions élémentaires pour un nombre total de soustractions élémentaires (sur un seul registre) en $O(n^2)$ au total en considérant σ comme une constante. À cela s'ajoute au plus $2n\log_2(\sigma)$ décalages binaires multi-précisions, chacun coûtant au plus n décalages binaires élémentaires pour un nombre total en $O(n^2)$. Pour finir, le quotient requiert au plus $n\log_2(\sigma)$ additions élémentaires.

Remarque 1. Sur la plupart des processeurs modernes, un décalage binaire demande un temps d'exécution très similaire à celui d'une multiplication [26]. Ainsi, une division s'avère être plus coûteuse qu'une multiplication pour la même taille d'opérandes.

Une autre opération importante, qui est notamment au centre de protocoles cryptographiques tels que le protocole RSA [55], est l'exponentiation modulaire que nous présentons ensuite.

Algorithme 5 Right-to-left square and multiply

Require: $a \in \mathbb{N}$ la base, $b \in \mathbb{N}$ l'exposant et $m \in \mathbb{N}$ tel que $m > 1$ le modulo.
Ensure: c tel que $c = a^b \pmod m$

```
1:  c ← 1
2:  d ← a
3:  i ← 0
4:  while 2^i ⩽ b do
5:      if b & 2^i > 0 then
6:          c ← c × d (mod m)
7:      end if
8:      d ← d × d (mod m)
9:      i ← i + 1
10: end while
11: return c
```

L'algorithme 5 détaille la méthode d'exponentiation modulaire dénommée "Right-to-left square and multiply" qui est le miroir de la méthode de division du paysan russe présentée plus haut. Pour calculer a^b, au lieu de naïvement multiplier a par lui-même b fois, on visualise b comme une somme de puissances de 2 et on multiplie des puissances carrées de a au résultat au fur et à mesure correspondant aux puissances de 2 contenues dans b. C'est-à-dire, on pose $b = \sum_i b_i 2^i$ la décomposition binaire de b. On a donc $a^b = a^{\sum_i b_i 2^i} = \prod_i (a^{2^i})^{b_i}$. Ainsi, il est clair que l'on a besoin de multiplier les puissances carrées de a lorsque un b_i, représentant les bits de b, vaut 1.

À la ligne 4 de l'algorithme, le symbole & correspond à l'opérateur ET binaire (ET bit à bit). Cet algorithme effectue au plus $2\log_2(b)$ multiplications modulaires, ce qui est bien plus performant que la méthode naïve demandant b multiplications modulaires.

La multiplication modulaire est au cœur de nombreuses opérations critiques, surtout en cryptographie avec le protocole RSA évoqué plus haut mais aussi en ce qui concerne la cryptographie sur les courbes elliptiques [45, 36]. Il est donc important de porter une attention particulière à cette opération. Plusieurs méthodes d'optimisation existent, comme le fait de considérer des formes d'entiers particulières pour le modulo ou encore des algorithmes plus généraux comme celui de Peter Montgomery. Nous aborderons tout d'abord les entiers particuliers.

1.2.1 Entiers de formes particulières permettant une réduction modulaire rapide

Certains protocoles cryptographiques, comme ceux issus de la cryptographie sur les courbes elliptiques, permettent de choisir une forme d'entier appropriée car la sécurité de ces protocoles ne repose pas sur le tirage aléatoire d'un nombre premier. La spécificité de la forme repose souvent sur une relation particulière avec les puissances de 2. En effet, comme nous avons pu voir plus haut, les opérations les plus coûteuses sont souvent liées à la division. Les divisions par une puissance de 2 sur un ordinateur peuvent être effectuées avec un décalage binaire, ce qui correspond à une complexité linéaire en la taille des opérandes. Ainsi, choisir un nombre premier possédant un lien avec une puissance de 2 permet souvent une simplification de la division par application de décalages binaires.

Nombres de Mersenne

Les nombres de Mersenne doivent leur nom à Marin Mersenne qui les a étudiés au XVII$^{\text{e}}$ siècle. Un nombre est dit de Mersenne s'il peut s'écrire sous la forme $2^n - 1$. Une réduction modulaire d'un entier d'au plus $2n$ bits par un nombre de Mersenne revient à effectuer un décalage binaire de n bits suivi d'une addition. En effet, un tel entier est de la forme $q2^n + r$ avec $q < 2^n$ et $r < 2^n$. Or $2^n \equiv 1 \pmod{2^n - 1}$ donc $q2^n + r \equiv q + r \pmod{2^n - 1}$.

Un nombre de Mersenne est premier seulement si n est premier. Si on écrit $n = ab$, on a

$$2^n - 1 = 2^{ab} - 1 = (2^a)^b - 1 = (2^a - 1)((2^a)^{b-1} + (2^a)^{b-2} + \cdots + 2^a + 1).$$

Ainsi, pour n non premier, $2^n - 1$ peut se factoriser et ne peut donc pas être premier. Cette condition n'est cependant pas suffisante. En effet $2^{11} - 1 = 2047 = 23 \times 89$.

Certains premiers de Mersenne sont utilisés dans un contexte cryptographique tel que l'entier $p = 2^{521} - 1$ [1]. Malheureusement, les premiers de Mersenne sont relativement rares et à ce jour seulement une cinquantaine de nombres premiers de Mersenne sont connus [57]. Ainsi, certains intervalles d'entiers importants du point de vue cryptographique ne possèdent pas de premier de Mersenne correspondant. Notamment, l'écart entre $2^{127} - 1$ et $2^{521} - 1$ est problématique pour la cryptographie sur les courbes elliptiques qui nécessite des tailles d'entiers entre 255 et 511 bits du point de vue de la sécurité. Cela explique en partie pourquoi ces entiers ne sont pas utilisés plus souvent malgré leurs propriétés intéressantes.

Nombres pseudo-Mersenne

Les nombres pseudo-Mersenne sont une extension des nombres de Mersenne. Spécifiquement, un nombre est dit pseudo-Mersenne s'il est de la forme $2^n - c$ avec c un "petit" nombre. Le brevet déposé par Richard Crandall en 1991 [12] fixe précisément c comme étant impair et au plus sur 32 bits. Donc $1 \leqslant c < 2^{32}$.

Cette catégorie englobe évidemment celle des nombres de Mersenne en prenant $c = 1$. Une réduction modulaire par un nombre pseudo-Mersenne consiste à effectuer un décalage binaire, une multiplication par c et une addition. Cela est évidemment légèrement plus lent que pour les nombres de Mersenne, avec l'avantage

que les pseudo-Mersenne sont beaucoup plus nombreux. En effet, un nombre pseudo-Mersenne étant premier n'implique pas que n soit forcément premier, ce qui élargit le champ des possibilités. Un exemple trivial est $p = 11 = 2^4 - 5$.

Un exemple plus pertinent est $p = 2^{255} - 19$, un premier pseudo-Mersenne utilisé dans le protocole d'échanges de clés X25519 [6], notable du fait d'être le protocole basé sur la cryptographie sur les courbes elliptiques le plus utilisé à l'heure actuelle [41, page 8].

L'écart maximal entre deux nombres premiers est un problème ouvert mais à l'heure actuelle nous ne connaissons pas deux nombres premiers consécutifs p_1 et p_2 tels que $|p_1 - p_2| \geqslant \ln(p_1)^2$. Ce résultat est l'objet de la conjecture de Cramer–Shanks–Granville et le plus grand ratio $\frac{|p_1-p_2|}{\ln(p_1)^2}$ connu est de 0.9206 [58]. Ainsi on peut supposer que les nombres pseudo-Mersenne existent pour tout n tels que $\ln(2^n)^2 < 2^{32}$ (avec $2^n + 1$ premier dans le pire des cas). Si cette conjecture tient, cela impliquerait qu'il existe des entiers pseudo-Mersenne premiers pour des valeurs de n jusqu'au moins 94548.

Nombres de Mersenne généralisés

Cette catégorie est une autre extension des nombres de Mersenne proposée par Jerome Solinas en 1999 [59]. L'approche cette fois-ci consiste à considérer les nombres de la forme $2^{n_1} \pm 2^{n_2} \pm \cdots \pm 1$ avec un "petit" nombre de n_i.

En effet, le nombre de décalages et d'additions nécessaires pour une réduction modulaire dépend directement du nombre de termes. Ainsi, les performances sont comparables avec les nombres pseudo-Mersenne pour 3 ou 4 termes. Au-delà, les performances chutent rapidement car, sur les processeurs modernes, un décalage binaire prend à peu près le même nombre de cycles qu'une multiplication [26].

Un exemple où cette forme est utilisée en cryptographie est $p = 2^{448} - 2^{224} - 1$ dans la courbe elliptique Ed448-Goldilocks [30]. La plupart des courbes elliptiques officielles du NIST sont également construites sur des entiers premiers de cette forme, telle que la courbe secp256r1 [16], construite sur $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.

1.2.2 Coefficients réduits

Outre l'accélération de l'opération de réduction modulaire, le fait de fixer la forme d'entiers considérée permet d'employer d'autres optimisations sur les calculs. L'une d'entre elles est la méthode des coefficients réduits présentée notamment dans [6]. L'idée principale est de ranger les entiers dans les registres non pas comme vu plus haut, c'est-à-dire

$$\overbrace{\underbrace{000\ldots0}_{\text{padding}}101\ldots101}^{\text{registre de poids fort}} \quad \underbrace{1101\ldots0100}_{\text{deuxième registre de poids fort}} \quad \ldots \quad \underbrace{0010\ldots1011}_{\text{registre de poids faible}}$$

en représentation binaire mais plutôt de la façon suivante

$$\underbrace{\overbrace{000\ldots0}^{\text{padding}}101\ldots101}_{\text{registre de poids fort}}\quad\underbrace{\overbrace{000\ldots0}^{\text{padding}}1101\ldots111}_{\text{deuxième registre de poids fort}}\quad\ldots\underbrace{\overbrace{000\ldots0}^{\text{padding}}1001\ldots1011}_{\text{registre de poids faible}}$$

c'est-à-dire en ajoutant systématiquement des 0 de padding sur le poids fort de chaque registre. Le nombre de registres nécessaires pour contenir le même entier est donc augmenté mais en contrepartie les 0 ainsi rajoutés peuvent être utilisés comme stockage temporaire de retenue, permettant de repousser le passage de retenue jusqu'à la fin des opérations où il sera nécessaire d'effectuer une étape de remise en place des 0 de padding en passant la retenue.

Cette méthode est principalement réalisable dans le cas où la forme d'entier considérée a été fixée car le nombre de 0 de padding doit être choisi en conséquence. En effet, lorsque l'on connaît à l'avance les opérations qui vont être effectuées, on peut choisir la taille du padding pour garantir qu'il n'y aura pas de dépassement tout au long des opérations. Un tel dépassement serait néfaste car il faudrait mettre en place une gestion de la retenue au milieu des opérations, ce qui serait coûteux et irait à l'encontre du principe.

Par exemple, pour $\sigma = 2^{64}$, prenons $p = 2^{122} - 3$. Soit

$$a = \texttt{0x3e18d8b65f7ade931929a966f1ba496},$$

on choisit de ranger a comme suit en mémoire (en utilisant un padding de 3 bits) :

$$a = \underbrace{\texttt{0x1f0c6c5b2fbd6f49}}_{\text{registre de poids fort}}\underbrace{\texttt{0x11929a966f1ba496}}_{\text{registre de poids faible}}$$

avec le premier registre étant celui de poids fort et le deuxième étant celui de poids faible. On note a_0 le registre de poids faible et a_1 celui de poids fort. Ainsi $a = a_1 2^{61} + a_0$. Nous avons donc 3 bits à 0 comme marge de manœuvre. Si l'on veut maintenant effectuer une multiplication modulo p de a par

$$b = \texttt{0x3f65c8025599656328eabed151e28a1}$$

que l'on répartit de telle sorte à avoir

$$b = \underbrace{\texttt{0x1fb2e4012accb2b1}}_{\text{registre de poids fort}}\underbrace{\texttt{0x128eabed151e28a1}}_{\text{registre de poids faible}},$$

notant b_0 le registre de poids faible et b_1 celui de poids fort. Ainsi $b = b_1 2^{61} + b_0$. On a alors

$$a \times b = a_1 b_1 2^{122} + (a_1 b_0 + a_0 b_1) 2^{61} + a_0 b_0.$$

Étant donné qu'il existe q et r tels que

$$a_1 b_0 + a_0 b_1 = q 2^{61} + r,$$

on en déduit que

$$a \times b = (a_1 b_1 + q)2^{122} + r2^{61} + a_0 b_0$$

ce qui donne

$$a \times b \equiv (a_1 b_1 + q)3 + r2^{61} + a_0 b_0 \pmod{p}.$$

On peut donc procéder comme suit pour le calcul de $a \times b$:

1. $t_0 = a_0 \times b_0 =$ `0x14619fd86b0450a94d49493c9adf256`
2. $t_1 = a_0 \times b_1 + a_1 \times b_0 =$ `0x46d35aaeb9efda0d97e1b9975297c9f`
3. $a'_1 = 3 \times a_1 =$ `0x5d2545118f384ddb`
4. $t_2 = a'_1 \times b_1 =$ `0xb889a3cc4c8e73f981dae3824941a6b`
5. $r_0 = t_0 + t_2 =$ `0xcceb43a4b792c4a2cf242cbee420cc1`
6. $r_1 = (t_1 \bmod 2^{61}) \times 2^{61} + \lfloor \frac{t_1}{2^{61}} \rfloor \times 3 =$ `0x32fc3732ea52f944a3d080616e7c712`
7. $c = r_0 + r_1 =$ `0xffe77ad7a1e5bde772f4ad20529d3d3`

Pour finir on répartit sur deux registres :

$$c = \underbrace{\texttt{0x7ff3bd6bd0f2def3}}_{\text{premier registre}} \underbrace{\texttt{0x172f4ad20529d3d3}}_{\text{deuxième registre}}.$$

On remarque que le registre de poids fort dépasse les 61 bits de valeur attribués donc un dernier passage de retenue est nécessaire pour finir la réduction modulaire par $p = 2^{122} - 3$. On récupère les bits de poids fort dépassant notre taille fixée, on multiplie par 3 avant d'ajouter le résultat au registre de poids faible et on obtient

$$c = \underbrace{\texttt{0x1ff3bd6bd0f2def3}}_{\text{premier registre}} \underbrace{\texttt{0x172f4ad20529d3dc}}_{\text{deuxième registre}}.$$

On voit ainsi l'intérêt de cette méthode. Dans toutes les étapes de calcul on est assuré que le résultat tient toujours sur 1 registre (ligne 3) ou 2 registres (autres lignes).

Nous avons présenté ici un exemple jouet donc l'importance n'est pas forcément soulignée sur l'intérêt en termes de temps de calcul mais considérons maintenant le calcul de $a \times b$ modulo p' avec $p' = 2^{122} - 113$ avec la même répartition dans les registres. Les premières étapes sont identiques jusqu'à :

3. $a'_1 = 113 \times a_1 =$ `0xdb47bd440129e1f39`.

En effet, a'_1 dépasse $\sigma = 2^{64}$, ce qui impose un traitement de la retenue en cours de calcul. Cela va à l'encontre de l'intérêt de la méthode, qui visait justement à repousser cette gestion à la fin. Si a'_1 était inférieur à $\frac{\sigma}{113}$, ce genre de dépassement ne se produirait pas, donc cette retenue n'est pas systématique. Il s'agit ici d'un exemple sur des entiers pseudo-Mersenne et nous avons ici un exemple où cette méthode de coefficients réduits pose un problème. Un entier quelconque provoquerait évidemment encore plus de complications. Ceci montre bien que cette méthode n'est viable que lorsque la forme d'entier est fixée à l'avance.

Nous avons détaillé dans ces deux premières sous-sections les formes d'entiers particulières utilisées en arithmétique pour obtenir une réduction modulaire rapide ainsi que les méthodes disponibles dans ces situations. Dans la section suivante, nous détaillerons l'algorithme utilisé dans l'état de l'art pour des entiers quelconques lorsque le choix de la forme d'entier n'est pas possible, comme lors d'un tirage aléatoire.

1.2.3 Multiplication modulaire de Montgomery

La multiplication modulaire de Montgomery [47] reprend les principes sous-jacents aux opérations sur les entiers de Mersenne. Puisque les opérations de division par des puissances de 2 sont peu coûteuses, cette méthode consiste à remplacer la division par un entier quelconque par une division par une puissance de 2.

Algorithme 6 Multiplication Modulaire de Montgomery

Require: $m \in \mathbb{N}^*$ le module, $a \in \mathbb{Z}/m\mathbb{Z}$, $b \in \mathbb{Z}/m\mathbb{Z}$, $\phi = 2^h$ avec $h \in \mathbb{N}^*$ tel que $m < 2^h$.
Ensure: $c \in \mathbb{Z}/m\mathbb{Z}$ tel que $c \equiv ab\phi^{-1} \bmod m$
1: $c \leftarrow a \times b$
2: $q \leftarrow -c \times m^{-1} \bmod \phi$
3: $c \leftarrow (c + q \times m)/\phi$ # Division exacte
4: **return** c

L'algorithme 6 introduit un paramètre ϕ, une puissance de 2 strictement supérieure au module m. Au lieu de calculer $c = a \times b \pmod m$, on calcule plutôt

$$c = \frac{(a \times b) + (-a \times b \times m^{-1} \bmod \phi) \times m}{\phi}.$$

En effet,

$$(a \times b \times m^{-1} \bmod \phi) \times m \equiv a \times b \pmod \phi$$

donc

$$(a \times b) + (-a \times b \times m^{-1} \bmod \phi) \times m \equiv 0 \pmod \phi.$$

Ainsi, nous avons un multiple de ϕ et diviser par ϕ nous donne donc une quantité inférieure à m. De plus, puisque nous rajoutons un multiple de m, le résultat est bien tel que $c \equiv a \times b \times \phi^{-1} \pmod m$, le ϕ^{-1} étant conséquence de la division par ϕ. Cet algorithme échange donc une division euclidienne par m avec une division par une puissance de 2 et deux multiplications supplémentaires, une d'entre elles étant une multiplication modulaire avec une puissance de 2 en modulo. Ceci est donc généralement beaucoup moins coûteux pour m quelconque, ce qui explique l'utilisation fréquente de cet algorithme dans des implémentations réelles de protocoles cryptographiques.

Pour donner un exemple concret, on pose

$$m = \texttt{0x7fffffffffffffffffffffffffffffff},$$

$$a = \texttt{0x3eb356ba1dff10f6bed6d37ed5b3c47e}$$

et

$$b = \texttt{0xa50d31edfb65c30a0d3ca91457cffa5}.$$

On choisit $\phi = 2^{128}$. On effectue $c = a \times b$ et on obtient

$$c = \texttt{0x286cd23bf9fc987d4110206a83cdd700ab911dc71d08b4135e25604f09f2736}.$$

On calcule $q = -c \times m^{-1} \pmod{\phi}$ ce qui nous donne

$$q = \texttt{0xab911dc71d08b4135e25604f09f2736}.$$

Ainsi,

$$c + q \times m = \texttt{0x7e35611f8880f286f022d09208c710b000000000000000000000000000000000}.$$

Il apparaît ici clairement que ceci est bien un multiple de ϕ et on obtient pour finir

$$c = \texttt{0x7e35611f8880f286f022d09208c710b}$$

qui est bien tel que $c < m$ et $c \equiv ab\phi^{-1} \pmod{m}$.

Remarque 2. Puisque nous calculons $c \equiv ab\phi^{-1} \pmod{m}$ et non $c \equiv ab \pmod{m}$, l'accumulation du terme parasite ϕ^{-1} lors de calculs successifs peut s'avérer problématique. Ainsi, une méthode couramment utilisée consiste à rentrer dans ce qui est appelé le "domaine de Montgomery" où les entiers ont un facteur ϕ supplémentaire. Ainsi, les entrées de l'algorithme 6 seraient $a\phi$ et $b\phi$ pour avoir en sortie $c \equiv ab\phi \pmod{m}$. Ce facteur ϕ reste stable lors des opérations et permet de diminuer la correction à effectuer. Pour sortir du domaine, il suffira à la fin des opérations d'effectuer une dernière multiplication de Montgomery par 1 afin de retrouver le résultat escompté.

Variantes de la multiplication modulaire de Montgomery

L'algorithme de multiplication modulaire de Montgomery dispose de plusieurs variantes d'implémentations lorsque l'on considère les entiers multi-précision. Les plus pertinentes sont présentées dans [35] par Koç et al. Ces variantes consistent à ordonnancer les opérations de façon différente pour des raisons d'implémentations, que cela soit des considérations physiques pour le hardware, ou des considérations sur les instructions disponibles pour le software. Les deux variantes les plus utilisées en pratique sont la méthode CIOS (Coarsely Integrated Operand Scanning) et la méthode FIOS (Finely Integrated Operand Scanning). La variante FIOS se trouve particulièrement bien adaptée au contexte hardware. Puisque nous considérons principalement l'aspect logiciel, nous allons nous intéresser particulièrement à la méthode CIOS qui s'avère être meilleure dans ce contexte. En effet, même les bibliothèques ubiquitaires OpenSSL [50] et GMP proposent leurs implémentations de la multiplication modulaire Montgomery CIOS, ce qui souligne son intérêt. De plus, certaines implémentations logicielles de référence de cryptosystèmes comme celle de [10] font utilisation de cette variante (https://csidh.isogeny.org/software.html). À noter qu'en pratique, il y aurait sans doute peu de différence entre une implémentation logicielle des variantes FIOS et CIOS après application d'une optimisation du compilateur. On présente ici une paraphrase de l'algorithme de multiplication modulaire Montgomery CIOS présenté dans [35].

Le principe de l'algorithme 7 est de découper la multiplication de a par b en $\sum_{i=0}^{s-1} a \times (\lfloor \frac{b}{\sigma^i} \rfloor \bmod \sigma)\sigma^i$. À chaque tour de boucle, on peut annuler les $\log_2(\sigma)$ bits de poids faible du résultat en cours en multipliant par $-m^{-1} \bmod \sigma$ puis par m pour effectuer une réduction de Montgomery sur ces bits. Le résultat final est bien entendu le même que celui de l'algorithme de Montgomery classique mais la multiplication par $-m^{-1}$ est de complexité linéaire au lieu d'être quadratique grâce au fait de multiplier par $-m^{-1} \bmod \sigma$ s fois au

Algorithme 7 Multiplication modulaire Montgomery CIOS [35]

Require: $m \in \mathbb{N}$ le module, $a \in \mathbb{N}$ et $b \in \mathbb{N}$ les opérandes, w la taille de registre de l'ordinateur avec $2^w = \sigma$, ϕ la plus petite puissance de 2 telle que $\phi > m$, $s = \lceil \log_\sigma(\phi) \rceil$ et $m' \equiv -m^{-1} \pmod{\phi}$
Ensure: $c \in \mathbb{Z}/m\mathbb{Z}$ tel que $c \equiv a \times b \times \phi^{-1} \pmod{m}$

1: **for** $i = 0 \dots s-1$ **do**
2: $T \leftarrow 0$
3: **for** $j = 0 \dots s-1$ **do**
4: $T, S \leftarrow a_j \times b_i + T$
5: $c_j \leftarrow S$
6: **end for**
7: $(T, S) \leftarrow c_s + T$
8: $c_s \leftarrow S$
9: $c_{s+1} \leftarrow T$
10: $T \leftarrow 0$
11: $q \leftarrow c_0 \times m'_0 \pmod{\sigma}$
12: **for** $j = 0 \dots s-1$ **do**
13: $(T, S) \leftarrow c_j + q \times m_j + T$
14: $c_j \leftarrow S$
15: **end for**
16: $(T, S) \leftarrow c_s + T$
17: $c_s \leftarrow S$
18: $c_{s+1} \leftarrow c_{s+1} + T$
19: **for** $j = 0 \dots s$ **do**
20: $c_j \leftarrow c_{j+1}$
21: **end for**
22: **end for**
23: **return** c

lieu de multiplier par $-m^{-1} \bmod \sigma^s$ une seule fois.

Reprenons l'exemple de la section précédente :

$$m = \texttt{0x7fffffffffffffffffffffffffffffff},$$

$$a = \texttt{0x3eb356ba1dff10f6bed6d37ed5b3c47e}$$

et

$$b = \texttt{0xa50d31edfb65c30a0d3ca91457cffa5},$$

en prenant $\sigma = 2^{64}$. On a ici

$$m' = \texttt{0x8000000000000000000000000000001}.$$

Au lieu d'effectuer le calcul de a par b en première étape, on prend les 64 bits de poids faible de b, c'est-à-dire `0xa0d3ca91457cffa5` et on les multiplie par a (lignes 2 à 9 de l'algorithme).

$$\begin{aligned} c &= \texttt{0x3eb356ba1dff10f6bed6d37ed5b3c47e} \times \texttt{0xa0d3ca91457cffa5} \\ &= \texttt{0x2763f5a1e52e12c6acb76e8c51ea6ba135e25604f09f2736} \end{aligned}$$

On multiplie ensuite c par m' modulo σ (en prenant directement c_0 et m'_0 dans le calcul) afin d'obtenir q (ligne 11). On a ici

$$q = \texttt{0x35e25604f09f2736}.$$

Ainsi, à la fin des lignes 12 à 18, on a calculé

$$c + q \times m = \texttt{0x425520a45d7da661acb76e8c51ea6ba10000000000000000}.$$

On peut voir ici clairement que ceci est divisible par 2^{64} donc on peut effectuer la division lignes 19 à 21. L'étape suivante consiste à multiplier a par les 64 bits suivants de b, en l'occurrence `0xa50d31edfb65c30` et ajouter au total en cours.

$$\begin{aligned} c &= c + \texttt{0x3eb356ba1dff10f6bed6d37ed5b3c47e} \times \texttt{0xa50d31edfb65c30} \\ &= \texttt{0x425520a45d7da661acb76e8c51ea6ba1} + \texttt{0x286cd23bf9fc987acad0c64c30ecaa95e01a3501fe61fa0} \\ &= \texttt{0x286cd23bf9fc987ef022d09208c710b0ab911dc71d08b41} \end{aligned}$$

$$q = \texttt{0xab911dc71d08b41}$$

$$c + q \times m = \texttt{0x7e35611f8880f286f022d09208c710b0000000000000000}$$

Ainsi, après division par 2^{64}, on retrouve bien le même résultat que plus haut, c'est-à-dire

$$c = \texttt{0x7e35611f8880f286f022d09208c710b}$$

et l'on a bien $c < m$ comme voulu et $c \equiv ab\phi^{-1} \pmod{m}$.

Remarque 3. Du fait de l'agencement des opérations, la multiplication de a par b a une complexité strictement quadratique. Cela n'est pas problématique lorsque les tailles d'entiers considérées sont suffisamment petites (en dessous de 2048 bits sur les processeurs modernes) mais sur les plus grandes tailles d'entiers la multiplication de Montgomery classique (non-CIOS) devient plus rapide grâce à l'utilisation d'algorithmes sous-quadratiques pour les multiplications multi-précisions.

1.2.4 Réduction de Barrett

La réduction de Barrett [5] est une alternative à la multiplication modulaire de Montgomery. L'opération de multiplication a déjà été effectuée et il faut ici la réduire modulo m. L'avantage principal est que cette méthode ne nécessite pas de domaine particulier. Nous présentons ici une variante légèrement différente de l'algorithme classique utilisant des arrondis au lieu de tronquer.

Algorithme 8 Réduction modulaire de Barrett

Require: $m \in \mathbb{N}^*$ le module, $c \in \mathbb{N}$ tel que $c < m^2$ l'opérande à réduire, $h \in \mathbb{N}^*$ et $k \in \mathbb{N}^*$ des paramètres de réduction, m' pré-calculé tel que $m' = \left\lfloor \frac{2^{h+k}}{m} \right\rfloor$.

Ensure: $c' \in \mathbb{N}$ tel que $c' \equiv c \bmod m$ et $c' \leqslant 2^{h-1} + \frac{m^3}{2^{h+k+1}} + \frac{m}{2^{k+2}}\frac{m}{2}$

1: $q \leftarrow \left\lfloor \frac{1}{2^k} \left\lfloor \frac{c}{2^h} \right\rfloor \times m' \right\rfloor$
2: $c' \leftarrow c - q \times m$
3: **return** c'

L'algorithme 8 prend en paramètre un entier c tel que $c < m^2$ avec m le module considéré. On note $c_1 = \left\lfloor \frac{c}{2^h} \right\rceil$ et $c_0 = c - 2^h \times c_1$. On note que l'on a alors $c_1 \leqslant \left\lfloor \frac{m^2}{2^h} \right\rceil$ et $c_0 \leqslant \frac{2^h}{2}$.

Ensuite on calcule $q = \left\lfloor \frac{c_1}{2^k} \times \left\lfloor \frac{2^{h+k}}{m} \right\rceil \right\rceil$. Soit $\varepsilon \in \mathbb{R}$ tel que

$$\left\lfloor \frac{2^{h+k}}{m} \right\rceil = \frac{2^{h+k}}{m} - \varepsilon.$$

Remarquons que $\varepsilon \leqslant \frac{1}{2}$. Ainsi $q = \left\lfloor \frac{c_1}{2^k} \times (\frac{2^{h+k}}{m} - \varepsilon) \right\rceil = \left\lfloor \frac{c_1 2^h}{m} - \frac{c_1 \varepsilon}{2^k} \right\rceil$.
Posons $\varepsilon' \in \mathbb{R}$ tel que

$$q = \frac{c_1 2^h}{m} - \frac{c_1 \varepsilon}{2^k} - \varepsilon'.$$

On remarque que $\varepsilon' \leqslant \frac{1}{2}$.
On a donc

$$q \times m = (\frac{c_1 2^h}{m} - \frac{c_1 \varepsilon}{2^k} - \varepsilon') \times m = c_1 2^h - \frac{m c_1 \varepsilon}{2^k} - m\varepsilon'.$$

D'où

$$\begin{aligned} c - q \times m &= c - (c_1 2^h - \frac{m c_1 \varepsilon}{2^k} - m\varepsilon') \\ &= c - c_1 2^h + \frac{m c_1 \varepsilon}{2^k} + m\varepsilon' \\ &= c_0 + \frac{m c_1 \varepsilon}{2^k} + m\varepsilon' \end{aligned}$$

Nous avons noté plus haut que $c_1 \leqslant \left\lfloor \frac{m^2}{2^h} \right\rceil$. Donc $\exists \varepsilon'' \in \mathbb{R}$ tel que $c_1 = \frac{m^2}{2^h} - \varepsilon''$ avec $\varepsilon'' \leqslant \frac{1}{2}$. De plus, étant donné que $c_0 \leqslant \frac{2^h}{2}$, on en déduit

$$c - q \times m \leqslant 2^{h-1} + \frac{m^3}{2^{h+k+1}} + \frac{m}{2^{k+2}} + \frac{m}{2}.$$

Ainsi, avec un choix judicieux de h et k pour avoir $2^{h-1} + \frac{m^3}{2^{h+k+1}} + \frac{m}{2^{k+2}} < \frac{m}{2}$, on peut garantir $c' < m$ en sortie de l'algorithme.

Pour exemple, on reprend les valeurs utilisées pour l'algorithme de Montgomery plus haut, c'est-à-dire

$$m = \texttt{0x7fffffffffffffffffffffffffffffff},$$

$$a = \texttt{0x3eb356ba1dff10f6bed6d37ed5b3c47e}$$

et

$$b = \texttt{0xa50d31edfb65c30a0d3ca91457cffa5},$$

et donc $c = a \times b$ est tel que

$$c = \texttt{0x286cd23bf9fc987d4110206a83cdd700ab911dc71d08b4135e25604f09f2736}.$$

On choisit $h = 64$ et $k = 191$ pour vérifier la condition $2^{h-1} + \frac{m^3}{2^{h+k+1}} + \frac{m}{2^{k+2}} < \frac{m}{2}$ et avoir une division par

2^{64} qui sera peu coûteuse. On pré-calcule

$$m' = \left\lfloor \frac{2^{h+k}}{m} \right\rceil = \texttt{0x100000000000000000000000000000002}.$$

Ainsi,

$$q = \left\lfloor \frac{1}{2^k} \times \left\lfloor \frac{c}{2^h} \right\rceil \times m' \right\rceil = \texttt{0x50d9a477f3f930fa822040d5079bae0}$$

et

$$c' = c - q \times m = \texttt{0xfc6ac23f1101e50de045a124118e216}.$$

On a bien $c' \equiv a \times b \pmod{m}$ et $c < m$.

Cet algorithme coûte 2 multiplications multi-précisions et 2 divisions par des puissances de 2. Le coût est relativement similaire à celui de l'algorithme de Montgomery mais en pratique Montgomery semble être plus souvent utilisé, sans doute du fait que la multiplication par m^{-1} modulo ϕ (ligne 2 algorithme 6) est moins coûteuse qu'une multiplication multi-précision complète où il faut gérer la partie haute (ligne 1 algorithme 8). Cela dit, l'algorithme de Barrett présente un certain nombre d'avantages, notamment l'absence de domaine particulier.

1.2.5 Réduction de Plantard

Proposée dans [53, Algorithme 8], cette méthode se présente comme une alternative à la réduction de Montgomery qui permet davantage de pré-calculs lors de l'exécution d'une exponentiation modulaire, ou d'une évaluation polynomiale, etc. Nous présentons ici une version légèrement différente faisant utilisation d'arrondis plutôt que de tronquer les valeurs.

Algorithme 9 Réduction modulaire de Plantard

Require: $m \in \mathbb{N}$ le module, $c \in \mathbb{N}$ tel que $c < m^2$ l'opérande à réduire, $\phi = 2^h$ avec $h \in \mathbb{N}^*$ tel que $m < 2^{h-1}$.
Ensure: $c' \in \mathbb{Z}/m\mathbb{Z}$ tel que $c' \equiv c\phi^{-2} \bmod m$
1: $q \leftarrow -c \times m^{-1} \bmod \phi^2$
2: $s \leftarrow \left\lfloor \frac{q}{\phi} \right\rceil$
3: $c' \leftarrow \left\lfloor \frac{s}{\phi} \times m \right\rceil$
4: **return** c'

L'algorithme 9 fonctionne sur le même principe que celui de Montgomery détaillé plus haut. La différence consiste à effectuer la réduction en multipliant par l'inverse de m modulo ϕ^2 et non ϕ. Cela sous-entend une multiplication plus coûteuse car $m^{-1} \pmod{\phi^2}$ occupera sans doute deux fois plus de registres mémoire que $m^{-1} \pmod{\phi}$. En contrepartie, lors d'une multiplication où les termes se répètent, par exemple une exponentiation modulaire d'un terme $a \in \mathbb{Z}/m\mathbb{Z}$, on peut calculer une seule fois $a \times m^{-1} \pmod{\phi^2}$ pour pouvoir par la suite effectuer $a \times (a \times m^{-1} \pmod{\phi^2})$, avec a occupant probablement la moitié de la taille mémoire de $m^{-1} \pmod{\phi^2}$ mais $a \times m^{-1} \pmod{\phi^2}$ occupant la même taille mémoire que $m^{-1} \pmod{\phi^2}$.

La seule différence de méthode avec la réduction de Montgomery concerne l'addition à la ligne 2 de

l'algorithme 9. Cette addition correspond à un terme correcteur. À la sortie de l'algorithme on a

$$c' = \left\lfloor \frac{m}{\phi} \left(\left\lfloor \frac{q}{\phi} \right\rceil \right) \right\rceil.$$

Notons pour la suite $q_1 = \left\lfloor \frac{q}{\phi} \right\rceil$ et $q_0 = q - q_1\phi$, ce qui implique $q_0 \leqslant \frac{\phi}{2}$. On a alors :

$$\begin{aligned}
\left\lfloor \frac{m}{\phi} \left(\left\lfloor \frac{q}{\phi} \right\rceil \right) \right\rceil &= \left\lfloor \frac{m}{\phi}(q_1 + 1) \right\rceil \\
&= \left\lfloor \frac{(\phi q_1) \times m}{\phi^2} \right\rceil \\
&= \left\lfloor \frac{(q - q_0) \times m}{\phi^2} \right\rceil \\
&= \left\lfloor \frac{q \times m - q_0 \times m}{\phi^2} \right\rceil \\
&= \left\lfloor \frac{q \times m + c - q_0 \times m - c}{\phi^2} \right\rceil \\
&= \left\lfloor \frac{q \times m + c}{\phi^2} + \frac{-q_0 \times m - c}{\phi^2} \right\rceil
\end{aligned}$$

Ainsi, étant donné que $q_0 \leqslant \frac{\phi}{2}$, $m < \frac{\phi}{2}$ et $c < m^2$, on a

$$\frac{q_0 \times m + c}{\phi^2} < \frac{\frac{\phi}{2} \times \frac{\phi}{2} + \frac{\phi^2}{4}}{\phi^2} = \frac{\frac{\phi^2}{4} + \frac{\phi^2}{4}}{\phi^2} = \frac{1}{2}.$$

Ainsi

$$\left| \frac{-q_0 \times m - c}{\phi^2} \right| < \frac{1}{2}$$

et donc

$$c' = \frac{q \times m + c}{\phi^2}.$$

De plus, puisque $q \equiv -cm^{-1} \pmod{\phi^2}$, on a $qm + c \equiv 0 \pmod{\phi}^2$ et donc

$$\frac{q \times m + c}{\phi^2} \equiv c\phi^{-2} \pmod{m}.$$

Donnons maintenant un exemple concret en reprenant les valeurs utilisées plus haut, c'est-à-dire

$$m = \texttt{0x7fffffffffffffffffffffffffffffff},$$

$$a = \texttt{0x3eb356ba1dff10f6bed6d37ed5b3c47e}$$

et

$$b = \texttt{0xa50d31edfb65c30a0d3ca91457cffa5},$$

avec $c = a \times b$ tel que

$$c = \texttt{0x286cd23bf9fc987d4110206a83cdd700ab911dc71d08b4135e25604f09f2736}.$$

On choisit $\phi = 2^{128}$ qui est bien tel que $\phi > 2m$ pour vérifier les conditions de sortie de l'algorithme. On calcule ensuite $q = -c \times m^{-1} \bmod \phi^2$ et on obtient

$$q = \texttt{0x87e35611f8880f286f022d09208c710b0ab911dc71d08b4135e25604f09f2736}.$$

Ensuite,

$$s = \left\lfloor \frac{q}{\phi} \right\rceil = \texttt{0x87e35611f8880f286f022d09208c710b}.$$

Pour finir,

$$c' = \left\lfloor \frac{s}{\phi} \times m \right\rceil = \texttt{0x43f1ab08fc4407943781168490463885}$$

qui est bien tel que $c' < m$ et $c' \equiv ab\phi^{-2} \pmod{m}$.

1.2.6 Mesures

À des fins de comparaison nous avons mesuré le nombre de cycles moyen pour une opération en fonction de la taille des opérandes sur un processeur moderne. La table 1.1 recense nos résultats pour l'opération $C = A \times B$ avec des tailles variées de C, A et B. Plus explicitement, C_{128} correspond à une variable de type `__int128`, qui correspond à 128 bits logiques où le compilateur effectue des opérations sur deux registres physiques de façon transparente au niveau logiciel. En contraste, C_{64} correspond à un résultat sur 64 bits où la partie haute n'est pas considérée.
Les mesures sont effectués d'après les méthodes indiquées dans le livre blanc d'intel sur le sujet [51].

- Le *Turbo-Boost*® est désactivé pendant les tests ;
- 501 exécutions sont lancées pour "chauffer" la mémoire cache, c'est à dire que nous nous assurons que la mémoire cache (données et instructions) est dans un état suffisamment stable pour éviter les défauts de cache ;
- 1001 jeux de données aléatoires sont générés, et le nombre de cycles processeurs pour W appels (W étant fixé à l'avance) sur un lot de 501 exécutions est enregistré pour chaque jeu de données ;
- la mesure finale est la valeur médiane divisée par W.

Opération	Nombre de cycles	Ratio
$C_{128} = A_{64} \times B_{64}$	3.108	1
$C_{128} = A_{64} \times B_{128}$	4.358	1.402
$C_{128} = A_{128} \times B_{128}$	4.813	1.549
$C_{64} = A_{64} \times B_{64}$	1.796	0.5779

TABLE 1.1 – Tableau de comparaisons du coût d'une multiplication en fonction de la taille des opérandes avec gcc 12.3.0 sur un processeur intel i9-11900KF.

Comme on peut le voir, le fait de ne pas considérer la partie haute permet de presque doubler la vitesse des opérations.

1.3 Arithmétique sur les Polynômes

L'arithmétique des polynômes sur les ordinateurs est très similaire à celle sur les entiers. Pour cause, on peut voir la représentation classique d'un entier multi-précision comme un polynôme de puissances de σ.

En particulier, l'algorithme de Karatsuba présenté plus haut (algorithme 3) peut aussi être utilisé sur des polynômes.

Soit un polynôme $C \in \mathbb{Z}[X]$ de degré n, on note

$$C(X) = c_0 + c_1 X + c_2 X^2 + \cdots + c_n X^n.$$

On décompose en partie haute et partie basse :

$$C_l(X) = c_0 + c_1 X + \cdots + c_{n-\lfloor \frac{n}{2} \rfloor - 1} X^{n-\lfloor \frac{n}{2} \rfloor - 1}$$

et

$$C_h(X) = c_{n-\lfloor \frac{n}{2} \rfloor} + c_{n-\lfloor \frac{n}{2} \rfloor + 1} X + \cdots + c_n X^{n-\lceil \frac{n}{2} \rceil}.$$

Ainsi, pour n impair $\deg(C_l) = \deg(C_h) = \lfloor \frac{n}{2} \rfloor$ sinon $\deg(C_l) = \deg(C_h) + 1 = \frac{n}{2}$ et on a

$$C(X) = C_l(X) + C_h(X) \times X^{\lceil \frac{n}{2} \rceil}.$$

Donc, on peut effectuer les mêmes opérations que celles de l'algorithme de Karatsuba sur les entiers en découpant les polynômes de cette façon pour accomplir une multiplication en temps sous-quadratique.

Un enjeu particulier pour les polynômes est l'évaluation polynomiale modulaire. Nous présentons donc un algorithme efficace pour ce faire.

Algorithme 10 Évaluation polynomiale modulaire de Horner

Require: $m \in \mathbb{N}^*$ le module, $A \in \mathbb{Z}[X]$ de degré $n \in \mathbb{N}$ noté $A(X) = a_0 + a_1 X + \cdots + a_n X^n$, $v \in \mathbb{Z}/m\mathbb{Z}$ la valeur d'évaluation.
Ensure: $r \in \mathbb{Z}/m\mathbb{Z}$ tel que $r \equiv A(v) \bmod m$
1: $r \leftarrow a_n$
2: **for** $i = 1..n$ **do**
3: $\quad r \leftarrow r \times v + a_{n-i} \bmod m$
4: **end for**
5: **return** r

L'algorithme 10 est l'algorithme d'évaluation polynomiale de Horner [32] dans l'anneau $\mathbb{Z}/m\mathbb{Z}[X]$. Le principe est de considérer un polynôme $A(X) = a_0 + a_1 X + \cdots + a_n X^n$ comme une série de multiplications par X et d'additions, en remarquant que

$$A(X) = ((\ldots(a_n \times X + a_{n-1}) \times X + \ldots) \times X + a_1) \times X + a_0.$$

Ainsi, une évaluation polynomiale en une valeur v peut être effectuée en exactement n multiplications par v. Ceci contraste avec la méthode naïve de calculer les puissances de v pour l'évaluation. Cette méthode pren-

drait soit $\frac{n(n+1)}{2}$ multiplications par v pour une version très naïve ou demanderait de stocker les puissances de v successives en mémoire, ce qui est potentiellement coûteux. La méthode de Horner présente le meilleur des deux mondes entre ces deux méthodes, ne nécessitant pas plus de taille mémoire que la méthode naïve mais permettant le même temps d'exécution que l'autre méthode.

Nous aborderons des principes sur l'arithmétique des polynômes plus spécifiques dans les sections à suivre mais pour finir sur les principes généraux nous parlerons ici des polynômes réciproques.

Définition 1. Soit $P(X) = p_0 + p_1X + \cdots + p_nX^n$ un polynôme dans $\mathbb{C}[X]$. On appelle $P^*(X) = \overline{p_n} + \overline{p_{n-1}}X + \cdots + \overline{p_0}X^n$ le polynôme réciproque de P où $\overline{p_i}$ désigne le conjugué de p_i.

Une propriété particulière des polynômes réciproques est que si l'on note $r_0, r_1, \dots r_d$ les d racines non nulles de P, alors $\overline{r_0}^{-1}, \overline{r_1}^{-1}, \dots, \overline{r_d}^{-1}$ sont des racines de P^*. Cela s'explique par le fait que pour toute valeur non nulle v, $P^*(v) = v^n\overline{P(\overline{v}^{-1})}$. Ainsi $\forall i$, $P^*(\overline{r_i}^{-1}) = \overline{r_i}^{-n}\overline{P(r_i)} = 0$.

Remarque 4. Si l'on considère un polynôme $P(X) = p_0 + p_1X + \cdots + p_nX^n$ dans un corps fini, alors son polynôme réciproque est $P^*(X) = p_n + p_{n-1}X + \cdots + p_0X^n$. De plus, pour $r_0, r_1, \dots r_d$ les racines de P, $r_0^{-1}, r_1^{-1}, \dots r_d^{-1}$ sont des racines de P^*.

1.4 Arithmétique des réseaux euclidiens

Dans cette section nous allons aborder les aspects théoriques sur les réseaux euclidiens dont nous avons besoin pour la suite. Pour commencer définissons ce qu'est un réseau euclidien.

Définition 2. On appelle réseau euclidien un sous-groupe discret d'un espace vectoriel euclidien.

Par exemple, $\mathbb{Z}^n$ est un sous-groupe discret de $\mathbb{R}^n$. En effet, tout vecteur de $\mathbb{Z}^n$ est stable par addition d'un autre vecteur de $\mathbb{Z}^n$. De façon générale, un réseau peut être vu comme le groupe des combinaisons linéaires à coefficients entiers d'une famille de vecteurs d'un espace vectoriel euclidien. Puisque tous les espaces vectoriels euclidiens sont isomorphes à $\mathbb{R}^n$, on se permet de le prendre comme référence par la suite.

Définition 3. On appelle base d'un réseau $\Lambda \subset \mathbb{R}^n$ la famille libre $(v_0, v_1, \dots v_{d-1})$ telle que $\forall l \in \Lambda$, $\exists(\mu_0, \mu_1, \dots, \mu_{d-1}) \in \mathbb{Z}^d$ tels que $\sum_{i=0}^{d-1} \mu_i v_i = l$.

La base d'un réseau n'est pas forcément une base de $\mathbb{R}^n$. En effet, il n'y a pas forcément égalité entre d et n.

Définition 4. On appelle rang d'un réseau euclidien $\Lambda \subset \mathbb{R}^n$, noté rang($\Lambda$), la dimension du sous-espace vectoriel engendré par les vecteurs de la base de ce réseau. En particulier si rang(Λ) $= n$ on appelle Λ un réseau de rang plein.

Une autre notion importante concerne le volume du réseau. Celui-ci représente le volume du parallélépipède fondamental associé aux vecteurs de la base du réseau.

Définition 5. Soit $\Lambda \subset \mathbb{R}^n$ un réseau euclidien. Soit $\mathbf{B}$ la base de Λ. Le volume ou déterminant de Λ a l'expression suivante :

$$\text{vol}(\Lambda) = \det(\Lambda) = \sqrt{\det(\mathbf{B}\mathbf{B}^\intercal)}$$

En particulier si $\mathbf{B}$ est une base de $\mathbb{R}^n$, $\det(\Lambda) = |\det(\mathbf{B})|$. Une autre notion importante est la notion de sous-réseau.

Définition 6. Soit $\Lambda \subset \mathbb{R}^n$ un réseau euclidien. Soit $\Lambda' \subset \mathbb{R}^n$ un deuxième réseau euclidien. Si $\forall l \in \Lambda$, $l \in \Lambda'$ alors Λ est appelé sous-réseau de Λ'.

Pour finir, un réseau euclidien ne possède pas une base unique. En pratique, on préfère utiliser une base dont les coefficients sont les plus petits possibles pour réduire à la fois la taille mémoire et la complexité des opérations. Ainsi, on appellera une base courte une base dont les coefficients sont "suffisamment petits". Ce concept est souvent défini en considérant une approximation de la base la plus courte possible en termes de norme. Pour obtenir une base courte, plusieurs algorithmes sont couramment utilisés tels que LLL [40], BKZ [56] ou encore HKZ [37].

1.4.1 Normes

Les problèmes importants sur les réseaux reposent surtout sur les notions de bases courtes, vecteurs courts, etc. Ceux-ci sont définis en fonction de normes et donc nous aborderons ici brièvement quelques notions sur les normes.

Soit $v \in \mathbb{R}^n$ un vecteur. On note $v = (v_0, v_1, \ldots, v_{n-1})$. Les normes que nous considérerons sont les suivantes :

— La norme infinie de v dont l'expression est $\|v\|_\infty = \max_i(|v_i|)$.
— La norme euclidienne, aussi appelée norme 2, dont l'expression est $\|v\|_2 = \sqrt{(v_0)^2 + (v_1)^2 + \cdots + (v_{n-1})^2}$.
— La norme une de v dont l'expression est $\|v\|_1 = \sum_{i=0}^{n-1} |v_i|$.

Les relations entre les normes sont comme suit :

— $\|v\|_\infty \leqslant \|v\|_2 \leqslant \|v\|_1$
— $\sqrt{n}\|v\|_\infty \geqslant \|v\|_2$
— $\sqrt{n}\|v\|_2 \geqslant \|v\|_1$
— $n\|v\|_\infty \geqslant \|v\|_1$

Outre les normes sur les vecteurs, nous ferons aussi usage des normes sur les matrices qui sont définies différemment. Soit $M \in \mathcal{M}_n(\mathbb{R})$ une matrice, les normes qui nous intéresseront sont les suivantes :

— La norme infinie de M dont l'expression est $\|M\|_\infty = \max_i(\sum_j |M_{ij}|)$.
— La norme une de M dont l'expression est $\|M\|_1 = \max_j(\sum_i |M_{ij}|)$.

Les relations particulières entre les normes d'une matrice ne nous intéressent pas forcément. La seule chose que l'on puisse dire ici est que pour toute matrice M, $\|M\|_\infty \leqslant n\|M\|_1$ et $\|M\|_1 \leqslant n\|M\|_\infty$. On remarque aussi que $\|M^\intercal\|_1 = \|M\|_\infty$ et vice versa.

Une autre propriété que nous utiliserons par la suite concerne la majoration du résultat d'un produit matrice-vecteur en fonction des normes des entrées. En effet, $\forall v \in \mathbb{R}^n$ et $\forall M \in \mathcal{M}_n(\mathbb{R})$, $\|v \cdot M\|_\infty \leqslant \|v\|_\infty \times \|M\|_1$. Ceci découle du fait que

$$\|v \cdot M\|_\infty \leqslant \|\|v\|_\infty (1, 1, \ldots, 1) \cdot M\|_\infty \leqslant \|v\|_\infty \times \|M\|_1.$$

1.4.2 Rayons particuliers

On peut définir un certain nombre de rayons particuliers en relation avec les distances minimales dans un réseau euclidien. Les distances minimales d'un réseau telles que définies par Minkowski [46] correspondent aux vecteurs les plus courts d'un réseau. Pour commencer, nous considérons le rayon d'empilement.

Définition 7. Soit $\Lambda \subset \mathbb{R}^n$ un réseau euclidien. Le rayon d'empilement de Λ en norme ℓ (pour $\ell \in \{1, 2, \infty\}$) est le plus grand rayon r strictement positif tel que les boules ouvertes $\mathcal{B}(v, r) = \{x \in \mathbb{R}^n : \|x - v\|_\ell < r\}$ centrées en les points du réseau soient disjointes deux à deux, c'est-à-dire :

$$\forall v, w \in \Lambda \times \Lambda, v \neq w \implies \mathcal{B}(v, r) \cap \mathcal{B}(w, r) = \emptyset.$$

Ce rayon possède une relation particulière avec la distance minimale du réseau. Plus précisément :

Lemme 1.4.1. *Soit $\Lambda \subset \mathbb{R}^n$ un réseau euclidien. Soit $v_1 \in \Lambda \setminus \{0\}$ tel que $\forall v \in \Lambda \setminus \{0\}$, $\|v\|_\ell \geqslant \|v_1\|_\ell$. Alors, r le rayon d'empilement de Λ en norme ℓ est tel que $r = \frac{1}{2}\|v_1\|_\ell$.*

Démonstration. Il existe deux points du réseau x et y tels que $\|x - y\|_\ell = \|v_1\|_\ell$. Ainsi, si l'on suppose $r > \frac{1}{2}\|v_1\|_\ell$, on a $\|x - y\|_\ell < 2r$. Donc il existe deux boules ouvertes $\mathcal{B}(x, r)$ et $\mathcal{B}(y, r)$ ayant une intersection. Contradiction.

Ainsi, clairement, $r \leqslant \frac{1}{2}\|v_1\|_\ell$. Or, $\|v_1\|_\ell$ correspond à la distance minimale en termes de norme ℓ donc $\forall (x, y) \in \Lambda \times \Lambda$, $x \neq y \implies \|x - y\| \geqslant \|v_1\|_\ell$. Donc si l'on choisit $r = \frac{1}{2}\|v_1\|_\ell$, toutes les boules ouvertes de rayon r n'auront pas d'intersection. Ainsi, $r = \frac{1}{2}\|v_1\|_\ell$. □

Une autre notion importante est le rayon de recouvrement.

Définition 8. Soit $\Lambda \subset \mathbb{R}^n$ un réseau euclidien. Le rayon de recouvrement de Λ en norme ℓ est le plus petit rayon μ strictement positif tel que les boules fermées $\overline{\mathcal{B}}(v, \mu) = \{x \in \mathbb{R}^n : \|x - v\|_\ell \leqslant \mu\}$ centrées en les points du réseau recouvrent tout l'espace, c'est-à-dire :

$$\text{vect}(\Lambda) \subseteq \bigcup_{v \in \mathfrak{L}} \overline{\mathcal{B}}(v, \mu)$$

Nous établissons ensuite une borne sur le rayon de recouvrement dans un réseau venant de [27, Lemme 4.3].

Lemme 1.4.2. *Soit $\Lambda \subset \mathbb{R}^n$, un réseau de rang plein. Soient $v_1, v_2, \ldots, v_n$ les n vecteurs linéairement indépendants les plus courts du réseau en termes de norme ℓ. Alors μ_ℓ, le rayon de recouvrement du réseau en norme ℓ est tel que $\mu_\ell \geqslant \frac{1}{2}\|v_n\|_\ell$.*

Démonstration. On suppose $\mu_\ell < \frac{1}{2}\|v_n\|_\ell$. Montrons que l'on pourra alors construire n vecteurs linéairement indépendants que l'on notera $(w_1, w_2, w_3, \ldots, w_n)$ tels que $\max_i \|w_i\|_\ell < \|v_n\|_\ell$ ce qui contredit la supposition que v_n soit le $n$ième vecteur le plus court du réseau en termes de norme ℓ.

Pour ce faire, on construit d'abord chaque w_i pour i dans $[\![2, n]\!]$ avec un certain vecteur $l_i \in \text{vect}(\Lambda)$ orthogonal à $v_1, w_2, \ldots, w_{i-1}$ tel que $\|l_i\|_\ell > \mu_\ell$. Par définition du rayon de recouvrement, il existe un certain $w_i \in \Lambda$ tel que $\|l_i - w_i\|_\ell \leqslant \mu_\ell$. Par inégalité triangulaire, la distance de w_i à $\text{vect}(v_1, w_2, \ldots, w_{i-1})$ est d'au

moins $\|l_i\|_\ell - \mu_\ell > 0$ donc $w_i \notin \text{vect}(v_1, w_2, \ldots, w_{i-1})$. Ainsi, la famille $(v_1, w_2, \ldots, w_{i-1}, w_i)$ est forcément libre. De plus, $\|w_i\|_\ell \leqslant \|l_i\|_\ell + \mu_\ell$ donc $\|w_i\|_\ell < 2\mu_\ell$ donc $\|w_i\|_\ell < \|v_n\|_\ell$.

Pour finir, on construit w_1 de la même façon en prenant l_1 orthogonal à $w_2, w_3, \ldots, w_n$ avec $\|l_1\|_\ell > \mu_\ell$. On aura bien $w_1 \notin \text{vect}(w_2, w_3, \ldots, w_n)$ et $\|w_1\|_\ell < \|v_n\|_\ell$.

Ayant construit une famille de n vecteurs linéairement indépendants dont tous les vecteurs ont une norme strictement inférieure à celle de v_n, cela contredit la supposition que v_n soit le $n$ième plus petit vecteur du réseau. D'où $\mu_\ell \geqslant \frac{1}{2}\|v_n\|_\ell$. □

1.5 Polynomial Modular Number System

Le Polynomial Modular Number System (PMNS) est le point focal sur lequel ont porté les travaux de cette thèse. Présenté pour la première fois par Bajard, Imbert et Plantard [3], ce système permet des opérations d'arithmétique modulaire efficaces sur les entiers multi-précision.

Définition 9. Soient $p \geqslant 3$, $n \geqslant 2$, $\gamma \in [1, p-1]$ et $\rho \in [1, p-1]$. Soit $E \in \mathbb{Z}[X]$, un polynôme de degré n, tel que $E(\gamma) \equiv 0 \pmod p$. Un PMNS est un ensemble $\mathcal{B} \subset \mathbb{Z}[X]$ tel que :

1. $\forall A \in \mathcal{B}$, $\deg(A) < n$,
2. $\forall A(X) = \sum_{i=0}^{n-1} a_i X^i \in \mathcal{B}$, $-\rho < a_i < \rho$ pour tout i,
3. $\forall a \in \mathbb{Z}/p\mathbb{Z}$, $\exists A \in \mathcal{B}$ tel que $A(\gamma) \equiv a \pmod p$.

Pour la suite de ce document, on désignera un PMNS par un tuple (p, n, γ, ρ, E). Soit $a \in \mathbb{Z}/p\mathbb{Z}$, on appelle représentant de a tout polynôme $A \in \mathbb{Z}_n[X]$ tel que $A(\gamma) \equiv a \pmod p$. Nécessairement, par le fait que tous les éléments de $\mathbb{Z}/p\mathbb{Z}$ doivent avoir un représentant dans un PMNS construit sur p, tous les tuples (p, n, γ, ρ, E) ne seront pas valides. Il faut donc établir des bornes sur ces paramètres afin de garantir l'existence de ces représentants.

Soit un PMNS $\mathcal{B} = (p, n, \gamma, \rho, E)$. On pose le réseau $\mathfrak{L} = \left\{(x_0, \ldots, x_{n-1}) \in \mathbb{Z}^n : \sum_{i=0}^{n-1} x_i \gamma^i \equiv 0 \pmod p\right\}$ qui correspond au réseau de tous les vecteurs de dimension n pour lesquels le polynôme associé (obtenu par l'isomorphisme π_n), s'annule en γ modulo p. Une base canonique de $\mathfrak{L}$ est proposée dans [4] de la forme suivante :

$$\mathbf{B} = \begin{pmatrix} p & 0 & 0 & \ldots & 0 & 0 \\ -\gamma & 1 & 0 & \ldots & 0 & 0 \\ -\gamma^2 & 0 & 1 & \ldots & 0 & 0 \\ \vdots & & & \ddots & & \vdots \\ -\gamma^{n-2} & 0 & 0 & \ldots & 1 & 0 \\ -\gamma^{n-1} & 0 & 0 & \ldots & 0 & 1 \end{pmatrix} \tag{1.1}$$

La première ligne de $\mathbf{B}$ correspond à $\nu_n(p)$ et on a bien $p \equiv 0 \pmod p$. Les $n-1$ lignes restantes sont de la forme $\nu_n(X^i - \gamma^i)$ qui s'annulent bien en γ modulo p. De plus, on note que le déterminant de cette base

est de p.

Remarque 5. Puisque $\mathbf{B}$ de l'équation (1.1) est telle que $\det(\mathbf{B}) = p$, $\mathbf{B}$ étant une base de $\mathfrak{L}$, p est également le volume du réseau. De plus, toute base du réseau a forcément le même déterminant.

Tous les éléments du réseau $\mathfrak{L}$ correspondent à des représentants de 0 par isomorphisme. Ainsi $\forall a \in \mathbb{Z}/p\mathbb{Z}$, $\forall v \in \mathfrak{L}$, $\pi_n(v)(\gamma) + a \equiv a \pmod p$ donc le polynôme $\pi_n(v)(X) + a$ est un représentant de a. De plus, $\forall a \in \mathbb{Z}/p\mathbb{Z}$, $\forall A \in \mathbb{Z}_{n-1}[X]$ tel que $A(\gamma) \equiv a \pmod p$, $\nu_n(A(X) - a) \in \mathfrak{L}$. Il est donc naturel d'envisager une couverture de $\mathbb{Z}^n$ par des boules centrées sur des éléments de $\mathfrak{L}$ et d'étudier les conditions pour englober au moins un représentant pour chaque élément de $\mathbb{Z}/p\mathbb{Z}$. Pour commencer nous posons le lemme suivant.

Lemme 1.5.1. *Soit $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS. Soit*

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

Soit r_∞ le rayon d'empilement en norme infinie de $\mathfrak{L}$. Alors, si $\rho \leqslant r_\infty$, chaque élément de $\mathbb{Z}/p\mathbb{Z}$ possède au plus un représentant dans $\mathcal{B}$.

Démonstration. Soit $a \in \mathbb{Z}/p\mathbb{Z}$. Supposons qu'il existe A et A' dans le PMNS tels que $A \neq A'$ et $A(\gamma) \equiv A'(\gamma) \pmod p$. On a $\|A\|_\infty < \rho$ et $\|A'\|_\infty < \rho$ car A et A' sont dans le PMNS et ρ borne leurs coefficients. Donc A est dans la boule ouverte de centre $\nu_n(0)$ et de rayon ρ.

$(A - A')(\gamma) \equiv 0 \pmod p$ donc $\nu_n(A - A') \in \mathfrak{L}$. De plus, $\nu_n(A - A') \neq 0$ car nous avons supposé $A \neq A'$. Or, A est dans la boule ouverte de centre $\nu_n(A - A')$ et de rayon ρ. En effet,

$$\|\nu_n(A) - \nu_n(A - A')\|_\infty = \|\nu_n(A')\|_\infty < \rho.$$

Donc les boules ouvertes de centres $\nu_n(0)$ et $\nu_n(A - A')$ et de rayon ρ possèdent une intersection : le point $\nu_n(A)$. Cela est une contradiction car par définition du rayon d'empilement, toutes les boules ouvertes de rayon au plus r_∞ centrées en les éléments de $\mathfrak{L}$ sont disjointes deux à deux et nous avons supposé $\rho \leqslant r_\infty$. Donc $A = A'$ et tout élément de $\mathbb{Z}/p\mathbb{Z}$ possède au plus un représentant dans $\mathcal{B}$. □

Une conséquence du lemme 1.5.1 est que si $A(X)$ est un représentant d'un élément a de $\mathbb{Z}/p\mathbb{Z}$ et que $\nu_n(A) \in B(\nu_n(0), \rho)$ la boule ouverte de centre $\nu_n(0)$ et de rayon ρ, alors $A(X)$ est l'unique représentant dans le PMNS de a. Aucune autre boule ouverte $B(v, \rho)$ pour $v \in \mathfrak{L}$, $v \neq \nu_n(0)$, ne contient un autre représentant de a. Ainsi, pour tout $\rho' < \rho$, si $\nu_n(A) \notin B(\nu_n(0), \rho')$ alors a ne possède aucun représentant dans le PMNS de paramètre (p, n, γ, ρ', E). Cette remarque est cruciale pour établir une condition nécessaire sur ρ, la borne sur les coefficients polynomiaux, en relation avec le rayon d'empilement de $\mathfrak{L}$.

On note r_∞ le rayon d'empilement de $\mathfrak{L}$ en norme infinie. On distingue deux cas. Pour $r_\infty \in \mathbb{N}^*$, d'après le lemme 1.5.1, un PMNS $\mathcal{B} = (p, n, \gamma, \rho, E)$ avec $\rho = r_\infty$ est tel que chaque élément de $\mathbb{Z}/p\mathbb{Z}$ possède au plus un représentant. Soit $A \in \mathbb{Z}[X]$, un polynôme constant tel que $A(X) = r_\infty - 1$. A est alors l'unique représentant de $r_\infty - 1$ dans $\mathcal{B}$ car $\|A\|_\infty < \rho$. Ainsi, un PMNS $\mathcal{B}' = (p, n, \gamma, \rho', E)$ avec $\rho' = r_\infty - 1$ est tel que

$A \notin \mathcal{B}'$, car $\nu_n(A) \notin B(\nu_n(0), \rho')$, et donc $r_\infty - 1$ ne possède pas de représentant dans $\mathcal{B}'$. En effet, tout représentant potentiel de $r_\infty - 1$ dans $\mathcal{B}'$ serait également inclus dans $\mathcal{B}$ et A est l'unique représentant de $r_\infty - 1$ dans $\mathcal{B}$. Ainsi, il est nécessaire de choisir $\rho \geqslant r_\infty$ afin de représenter tous les éléments de $\mathbb{Z}/p\mathbb{Z}$ dans un PMNS.

De façon analogue pour $r_\infty \in \mathbb{R}^+ \setminus \mathbb{N}$, on considère le polynôme constant $A' \in \mathbb{Z}[X]$ tel que $A(X) = \lfloor r_\infty \rfloor$. A' est l'unique représentant de $\lfloor r_\infty \rfloor$ dans un PMNS (p, n, γ, ρ, E) avec $\rho = r_\infty$ car $\|A'\|_\infty < \rho$. Un PMNS (p, n, γ, ρ', E) avec $\rho' = \lfloor r_\infty \rfloor$ ne contient pas A' et ne contient donc pas de représentant de $\lfloor r_\infty \rfloor$. Il est donc nécessaire de choisir $\rho > \lfloor r_\infty \rfloor$ afin de représenter tous les éléments de $\mathbb{Z}/p\mathbb{Z}$ dans un PMNS. Si l'on impose ρ entier, cela se traduit par choisir $\rho \geqslant \lceil r_\infty \rceil$. Ainsi, dans les deux cas, la condition revient à choisir $\rho \geqslant \lceil r_\infty \rceil$ (si $r_\infty \in \mathbb{N}^*$, $\lceil r_\infty \rceil = r_\infty$).

Or, d'après le lemme 1.4.1, $r_\infty = \frac{1}{2}\|v_1\|_\infty$ pour v_1 le vecteur le plus court de $\mathfrak{L}$ en termes de norme infinie. Donc il est nécessaire de choisir $\rho \geqslant \lceil \frac{1}{2}\|v_1\|_\infty \rceil$. Il est difficile de façon générale d'obtenir v_1 pour un réseau quelconque. Cela dit on sait que pour toute base $\mathbf{B}$ de $\mathfrak{L}$, $\|\mathbf{B}\|_1 \geqslant \|v_1\|_\infty$. Ainsi, choisir $\rho \geqslant \lceil \frac{1}{2}\|\mathbf{B}\|_1 \rceil$ garantit d'avoir aussi $\rho \geqslant \lceil \frac{1}{2}\|v_1\|_\infty \rceil$. Puisque n'importe quelle base convient, choisir une base courte mènera à des bornes moins contraignantes sur les paramètres. Une telle base courte peut par exemple être obtenue en appliquant l'algorithme LLL [40] à la base canonique (équation (1.1)).

Remarque 6. Le nombre d'éléments dans un PMNS est de $(2\rho - 1)^n$, étant donné que chaque élément est un polynôme de degré au plus $n - 1$ dont les coefficients sont strictement bornés par ρ. Ainsi, une autre condition nécessaire est $(2\rho - 1)^n \geqslant p$, ou encore $\rho \geqslant \frac{p^{\frac{1}{n}}+1}{2}$, sans quoi le nombre d'éléments du PMNS sera inférieur au nombre d'éléments de $\mathbb{Z}/p\mathbb{Z}$. Tous les éléments de $\mathbb{Z}/p\mathbb{Z}$ ne pourront alors pas être représentés dans le PMNS. Il est donc nécessaire de choisir $\rho \geqslant \max(\lceil \frac{1}{2}\|v_1\|_\infty \rceil, \frac{p^{\frac{1}{n}}+1}{2})$.

Le fait de choisir $\rho \geqslant \max(\lceil \frac{1}{2}\|v_1\|_\infty \rceil, \frac{p^{\frac{1}{n}}+1}{2})$ comme borne sur les coefficients polynomiaux du PMNS est une condition nécessaire mais pas suffisante. Pour obtenir une condition suffisante, il faut considérer le rayon de recouvrement de $\mathfrak{L}$ en norme infinie μ_∞. Ce résultat vient de [4, Théorème 4.1] qui démontre que tout PMNS vérifiant $\rho > \mu_\infty$ possède au moins un représentant pour chaque entier de $\mathbb{Z}/p\mathbb{Z}$. Pour $a \in \mathbb{Z}/p\mathbb{Z}$, on pose $T_a(X) = a$. Par définition du rayon de recouvrement, il existe $V_a \in \mathfrak{L}$ tel que

$$\|\nu_n(T_a) - V_a\|_\infty \leqslant \mu_\infty.$$

Ainsi le polynôme $A(X) = T_a(X) + \pi_n(V_a)$ est tel que $\|A\|_\infty \leqslant \mu_\infty$ et

$$A(\gamma) \equiv T_a(\gamma) + \pi_n(V_a)(\gamma) \equiv a + 0 \equiv a \pmod{p}.$$

Donc si l'on choisit $\rho > \mu_\infty$, il existe un polynôme représentant tout entier de $\mathbb{Z}/p\mathbb{Z}$ dont la norme infinie est strictement bornée par ρ. D'après le lemme 1.4.2, $\mu_\infty \geqslant \frac{1}{2}\|v_n\|_\infty$ pour v_n le $n$ième vecteur le plus court de $\mathfrak{L}$. Il s'agit ici d'une minoration et de plus il est difficile d'obtenir v_n pour un réseau quelconque. Cela dit, [4, Theorem 4.2] démontre que choisir $\rho > \frac{1}{2}\|\mathbf{B}\|_1$ pour n'importe quelle base $\mathbf{B}$ de $\mathfrak{L}$ est une condition suffisante sur ρ, la borne sur les coefficients polynomiaux dans un PMNS, pour garantir l'existence d'un représentant pour tous les entiers de $\mathbb{Z}/p\mathbb{Z}$ dans le PMNS. On ne détaille pas la preuve ici car nous proposons plus tard une autre façon de démontrer cette borne. Cependant, la borne sur le rayon de recouvrement nous donne une intuition sur ce résultat en notant que toute base du réseau $\mathbf{B}$ est telle que $\|\mathbf{B}\|_1 \geqslant \|v_n\|_\infty$.

Remarque 7. Bajard et al. notent [4, Section 4.2] que le résultat de prendre $\rho > \frac{1}{2}\|\mathbf{B}\|_1$ pour toute base $\mathbf{B}$ de $\mathfrak{L}$ pour garantir l'existence d'un représentant de tout élément de $\mathbb{Z}/p\mathbb{Z}$ reste valide pour toute base $\mathbf{B}'$ d'un sous-réseau de $\mathfrak{L}$ également.

1.5.1 Opérations dans un PMNS

Soit $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS. Soient $a \in \mathbb{Z}/p\mathbb{Z}$ et $b \in \mathbb{Z}/p\mathbb{Z}$. Par définition, il existe $A \in \mathcal{B}$ et $B \in \mathcal{B}$ tels que $A(\gamma) \equiv a \pmod p$ et $B(\gamma) \equiv b \pmod p$. Pour calculer $c = a \odot b \pmod p$ dans le PMNS avec $\odot$ l'addition, la soustraction ou la multiplication, l'opération $C(X) = A(X) \odot B(X)$ est tout d'abord effectuée. Il s'agit d'une opération polynomiale classique correspondant à un nombre linéaire en n d'additions/soustractions pour l'addition/soustraction polynomiale et de complexité au mieux sous-quadratique en n en nombre de multiplications élémentaires pour la multiplication polynomiale.

Pour la multiplication, $\deg(C) = \deg(A) + \deg(B)$ et puisque $\deg(A) \leqslant n-1$ et $\deg(B) \leqslant n-1$ alors $\deg(C) \leqslant 2n-2$. Autrement dit, il se peut que $\deg(C) > n-1$. Dans ce cas-là $C \notin \mathcal{B}$. Pour cette raison, une opération de réduction de degré, appelée Réduction Externe, sera nécessaire. Celle-ci fera appel au paramètre E du PMNS choisi pour cette raison. En effet, puisque $E(\gamma) \equiv 0 \pmod p$, une réduction modulaire par E ne change pas l'évaluation en γ du résultat. Si l'on considère pour l'instant que E est unitaire, effectuer une réduction modulaire par E permet de garantir que le résultat soit de degré au plus $n-1$. On calcule alors $C' \in \mathbb{Z}[X]$ tel que $\deg(C') < n$ et $C'(X) \equiv C(X) \pmod{E(X)}$. On aura alors bien $C'(\gamma) \equiv C(\gamma) \equiv A(\gamma) \times B(\gamma) \equiv a \times b \pmod p$ comme voulu. La Réduction Externe est détaillée en section 1.5.3.

Suite à une opération $\odot$ quelconque, il est possible que $C'(X) \equiv A(X) \odot B(X) \pmod{E(X)}$ soit tel que $\|C'(X)\|_\infty \geqslant \rho$. Il faudra alors calculer $\tilde{C} \in \mathbb{Z}[X]$ tel que $\tilde{C}(\gamma) \equiv C'(\gamma) \pmod p$ avec $\|\tilde{C}(X)\|_\infty < \rho$. On appelle ce processus la Réduction Interne que l'on détaillera plus en section 1.5.4.

1.5.2 Le paramètre n

La complexité des opérations repose essentiellement sur le paramètre n puisqu'il représente le nombre de registres mémoire nécessaires pour stocker les éléments d'un PMNS donné. Ainsi, prendre n le plus petit possible est souvent au cœur des décisions concernant la construction d'un PMNS. Didier et al. [15] ont introduit le paramètre $n_{opt} = \left\lfloor \frac{\log_2(p)}{\log_2(\sigma)} \right\rfloor + 1$ comme le plus petit n possible pour un p donné sur une taille mémoire donnée.

Remarque 8. La notion de chercher le plus petit n tel que $p < 2^{nk}$ avec $k = \log_2(\rho)$ ou $k = \log_2(\rho) - 1$ avait été introduite dans [52, Algorithme 35], ce qui est proche de la notion de n_{opt} introduite dans [15].

1.5.3 Réduction externe

La réduction externe consiste à réduire le degré d'un polynôme afin d'obtenir un polynôme de degré borné par n. Pour ce faire, le paramètre E d'un PMNS est choisi comme un polynôme de degré n qui s'annule en γ modulo p. Ce paramètre est souvent choisi comme un polynôme unitaire.

En effet, soit $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS avec E unitaire et $A, B \in \mathcal{B} \times \mathcal{B}$, alors $\deg(A(X) \times B(X) \pmod{E(X)}) \leqslant n-1$. Soit $C(X) = A(X) \times B(X)$, on a

$$C(X) = c_0 + c_1 X + \cdots + c_{n-1}X^{n-1} + c_n X^n + c_{n+1}X^{n+1} \cdots + c_{2n-2}X^{2n-2}.$$

On peut écrire $E(X)$ comme $E(X) = X^n - P(X)$ avec $P(X) \in \mathbb{Z}_{n-1}[X]$. Alors $X^n \equiv P(X) \pmod{E(X)}$ et donc

$$C(X) \equiv c_0 + c_1 X + \cdots + c_{n-1}X^{n-1} + P(X) \times (c_n + c_{n+1}X + \ldots c_{2n-2}X^{n-2}) \pmod{E(X)}.$$

Si l'on pose

$$C'(X) = c_0 + c_1 X + \cdots + c_{n-1}X^{n-1} + P(X) \times (c_n + c_{n+1}X + \ldots c_{2n-2}X^{n-2}),$$

alors on a

$$\deg(C') \leqslant \max(n-1, \deg(P) + n - 2) \leqslant n - 1 + n - 2 = 2n - 3.$$

Il suffit d'appliquer cette transformation un maximum de $n-1$ fois pour obtenir un polynôme de degré au plus $n-1$. Puisque $E(\gamma) \equiv 0 \pmod{p}$ par construction, ces étapes ne modifient pas la valeur de l'évaluation en γ car nous avons alors que $\forall Q \in \mathbb{Z}[X]$, $A(\gamma)B(\gamma) - Q(\gamma)E(\gamma) \equiv A(\gamma)B(\gamma) \pmod{p}$.

Constatons que choisir E de la forme $E(X) = X^n - \lambda$ avec $\lambda \in \mathbb{Z}$ est particulièrement avantageux. En effet, on peut alors poser

$$C'(X) = c_0 + c_1 X + \cdots + c_{n-1}X^{n-1} + \lambda \times (c_n + c_{n+1}X + \ldots c_{2n-2}X^{n-2})$$

étant donné que $X^n \equiv \lambda \pmod{E(X)}$ et le résultat est de degré au plus $n-1$. Certains auteurs dans la littérature appellent un PMNS avec E de cette forme un Adapted Modular Number System (AMNS). Nous ne ferons pas cette distinction ici sauf lors de citations directes.

Polynôme E non unitaire

L'utilisation des polynômes non unitaires n'avait pas été étudiée dans le contexte des PMNS jusqu'à [54] qui propose de prendre $E = aX^n - b$. Ainsi, pour un polynôme

$$C(X) = c_0 + c_1 X + \cdots + c_{n-1}X^{n-1} + c_n X^n + c_{n+1}X^{n+1} \cdots + c_{2n-2}X^{2n-2},$$

on peut prendre

$$C'(X) = a \times (c_0 + c_1 X + \cdots + c_{n-1}X^{n-1}) + b \times (c_n + c_{n+1}X + \ldots c_{2n-2}X^{n-2}).$$

Cette opération sous-entend une multiplication par a, ce qui peut être pris en compte en utilisant un domaine particulier, de façon similaire à la multiplication de Montgomery présentée en section 1.2.3. Cette forme demande aussi un ajustement sur la réduction interne que nous détaillons dans la remarque 11 de la section 1.5.4.

Irréductibilité

La plupart des auteurs dans la littérature imposent l'utilisation d'un polynôme E irréductible. En théorie, rien n'empêche l'utilisation d'un E réductible mais en pratique cela entraine une perte d'optimalité sur le choix minimal possible de ρ. En effet, si E n'est pas irréductible, il existe $P, Q \in \mathbb{Z}[X] \times \mathbb{Z}[X]$ tels que $E(X) = P(X)Q(X)$ avec $\deg(P) \leqslant n-1$ et $\deg(Q) \leqslant n-1$. Ainsi, soit $P(\gamma) \equiv 0 \pmod p$ soit $Q(\gamma) \equiv 0 \pmod p$. Sans perte de généralité, considérons que cela soit P. On aura donc $\nu_n(P) \in \mathfrak{L}$ pour $\mathfrak{L} = \left\{(x_0, \ldots, x_{n-1}) \in \mathbb{Z}^n : \sum_{i=0}^{n-1} x_i \gamma^i \equiv 0 \pmod p\right\}$. Cela est un problème car E est généralement choisi comme creux et avec une faible norme. Pour un certain nombre de E (comme $E(X) = X^n \pm X \pm 1$ ou encore $E(X) = X^n \pm X^{\frac{n}{2}} + 1$ avec n pair), $\|E\|_2 \approx 1$, mais la forme la plus utilisée en pratique reste $E(X) = X^n - \lambda$. Donc $\|P\|_2 \approx \lambda\sqrt{n}$. $\nu_n(P)$ est donc très certainement un vecteur court de $\mathfrak{L}$, λ étant relativement petit (généralement $\lambda < 2^8$). Soit $\mathbf{B}$ la base courte, on note $v_1, v_2, \ldots, v_{n-1}$ ses $n-1$ dernières lignes avec P en première ligne et $c_0, c_1 \ldots, c_{n-1}$ ses colonnes. On a $\det(\mathbf{B}) = p$ (cf. remarque 5) et donc l'inégalité de Hadamard [28] nous donne

$$\lambda\sqrt{n} \times \prod_{i=1}^{n-1} \|v_i\|_2 \geqslant \|P\|_2 \times \prod_{i=1}^{n-1} \|v_i\|_2 \geqslant p$$

$$\implies \prod_{i=1}^{n-1} \|v_i\|_2 \geqslant \frac{p}{\lambda\sqrt{n}}$$

$$\implies \max(\|v_i\|_2)^{n-1} \geqslant \frac{p}{\lambda\sqrt{n}}$$

$$\implies \max(\|v_i\|_2) \geqslant \left(\frac{p}{\lambda\sqrt{n}}\right)^{\frac{1}{n-1}}$$

$$\implies \sqrt{n}\max_i(\|v_i\|_\infty) \geqslant \left(\frac{p}{\lambda\sqrt{n}}\right)^{\frac{1}{n-1}}$$

$$\implies \max_i(\|v_i\|_\infty) \geqslant \frac{1}{\sqrt{n}}\left(\frac{p}{\lambda\sqrt{n}}\right)^{\frac{1}{n-1}}$$

En notant que $\max_i(\|v_i\|_\infty) = \max_i(\|c_i\|_\infty)$, on en déduit

$$\implies \max_i(\|c_i\|_\infty) \geqslant \frac{1}{\sqrt{n}}\left(\frac{p}{\lambda\sqrt{n}}\right)^{\frac{1}{n-1}}$$

et donc

$$\|\mathbf{B}\|_1 \geqslant \frac{1}{\sqrt{n}}\left(\frac{p}{\lambda\sqrt{n}}\right)^{\frac{1}{n-1}}$$

La condition $\rho > \frac{1}{2}\|\mathbf{B}\|_1$ impose donc de choisir $\rho > \frac{1}{2\sqrt{n}}\left(\frac{p}{\lambda\sqrt{n}}\right)^{\frac{1}{n-1}}$. On note qu'en temps normal, on a plutôt $\rho > \frac{1}{2}p^{\frac{1}{n}}$ si aucun des vecteurs courts n'est particulièrement creux. L'écart est conséquent et donc pour garder un ρ le plus petit possible, choisir E irréductible est un critère important.

Il peut être intéressant de considérer certains cas particuliers, par exemple $E(X) = X^n - 1$, mais en général un E irréductible est plus sûr.

Le paramètre w

Un paramètre w a été introduit par Dosso et al. [20] pour borner la norme d'un polynôme après réduction externe. Soit $\mathcal{B} = (p, n, \gamma, \rho, E)$. Soient $A, B \in \mathcal{B} \times \mathcal{B}$. On veut borner la quantité $\|A(X) \times B(X) \bmod E(X)\|_\infty$ en fonction de ρ. Ainsi, $\forall A, B$ on aura $\|A(X) \times B(X) \bmod E(X)\|_\infty \leqslant w(\rho-1)^2$ avec w ce nouveau paramètre introduit.
Pour dégager une expression pour w, on reformule l'opération de multiplication polynomiale modulaire comme une opération vecteur-matrice. En effet, soient

$$A(X) = a_0 + a_1 X + \dots a_{n-1} X^{n-1}$$

et

$$B(X) = b_0 + b_1 X + \dots b_{n-1} X^{n-1}.$$

Pour $C(X) = A(X) \times B(X)$ que l'on note

$$C(X) = c_0 + c_1 X + \cdots + c_{n-1} X^{n-1} + c_n X^n + c_{n+1} X^{n+1} + \cdots + c_{2n-2} X^{2n-2},$$

on peut voir le calcul de $C'(X) \equiv A(X) \times B(X) \bmod E(X)$ comme l'opération suivante :

$$C'(X) = \pi_n((c_0, c_1, \dots, c_{n-1}) + (c_n, c_{n+1}, \dots, c_{2n-2}) \cdot \mathcal{E}) \tag{1.2}$$

avec

$$\mathcal{E} = \begin{pmatrix} -e_0 & -e_1 & \dots & -e_{n-1} \\ \dots & \dots & \dots & \dots \\ \vdots & \vdots & \vdots & \vdots \\ \dots & \dots & \dots & \dots \end{pmatrix} \begin{matrix} \leftarrow \nu_n(X^n \bmod E) \\ \leftarrow \nu_n(X^{n+1} \bmod E) \\ \\ \leftarrow \nu_n(X^{2n-2} \bmod E) \end{matrix} \tag{1.3}$$

Ainsi, [20, Proposition 2] établit que $\forall A, B$, le polynôme C' défini dans l'équation (1.2) est tel que $\|C'\|_\infty \leqslant w(\rho-1)^2$ pour $w = \|(1, 2, \dots, n) + (n-1, n-2, \dots, 1) \cdot \mathcal{E}'\|_\infty$ avec $\mathcal{E}'$ la matrice telle que $\forall i, j \in [\![0, n-2]\!] \times [\![0, n-1]\!]$, $\mathcal{E}'_{i,j} = |\mathcal{E}_{i,j}|$.

Certaines formes particulières de E sont proposées dans [20] et sont détaillées dans la table suivante.

Forme de E	w
$X^n - \lambda$	$\lvert\lambda\rvert(n-1)+1$
$X^n \pm X^{\frac{n}{2}} + 1$	$3\frac{n}{2}$ (n pair)
$X^n + \sum_{i=0}^{\frac{n}{2}-1} X^{2i}$	$2n-1$ (n pair)
$(-1)^n \sum_{i=0}^{n} (-1)^i X^i$	$2n-1$
$X^n \pm X \pm 1$	$2n-1$
$X^n + \sum_{i=0}^{n-1} X^i$	$2n-2$

TABLE 1.2 – Tableau de formes de E intéressantes pour leurs w présentées dans [20].

Remarque 9. Les formes du polynôme E les plus utilisées en pratique sont $E(X) = X^n - \lambda$, avec $X^n + 1$ en particulier étant très intéressant (lorsqu'il est irréductible) et $X^n - 2$ étant utilisé très fréquemment, et $E(X) = X^n \pm X \pm 1$.

1.5.4 Réduction interne

La réduction interne consiste à réduire les coefficients d'un polynôme afin qu'ils soient tous bornés par le paramètre ρ d'un PMNS. Une des façons de résoudre ce problème est de le considérer comme un problème de vecteur proche dans un réseau.

En effet, pour un PMNS $\mathcal{B} = (p, n, \gamma, \rho, E)$, on considère le réseau

$$\mathfrak{L} = \left\{ (x_0, \ldots, x_{n-1}) \in \mathbb{Z}^n : \sum_{i=0}^{n-1} x_i \gamma^i \equiv 0 \pmod{p} \right\}$$

introduit plus haut. On note que tout élément v de ce réseau sera tel que $\pi_n(v)(\gamma) \equiv 0 \pmod{p}$. Ainsi, pour un polynôme $C \in \mathbb{Z}_{n-1}[X]$ à réduire, si on considère $W \in \mathfrak{L}$ tel que $\forall V \in \mathfrak{L}$,

$$\|C(X) - \pi_n(W)\|_\infty \leqslant \|C(X) - \pi_n(V)\|_\infty$$

alors par définition du rayon de recouvrement on a

$$\|C(X) - \pi_n(W)(X)\|_\infty \leqslant \mu_\infty.$$

De plus, $C(\gamma) - \pi_n(W)(\gamma) \equiv C(\gamma)$. Comme vu en section 1.5, choisir le paramètre ρ tel que $\rho > \mu_\infty$ permet de s'assurer de pouvoir représenter tous les éléments de $\mathbb{Z}/p\mathbb{Z}$ dans un PMNS (d'après [4, Théorème 4.1]). Donc on aura $\|C(X) - \pi_n(W)(X)\|_\infty < \rho$ dans ce cas-là. On peut donc réduire C en remplaçant C par $C - W$. Malheureusement, le problème de trouver le vecteur proche en norme infinie dans un réseau a été prouvé NP-complet (réduction à partir du problème de la somme de sous-ensembles) [23, 38].

Or, en pratique trouver le vecteur le plus proche n'est pas stricto sensu nécessaire. En effet, il suffit de trouver un $V \in \mathfrak{L}$ tel que $\|C(X) - \pi_n(V)(X)\|_\infty < \rho$ (avec ρ qui n'est pas forcément choisi d'après la valeur de μ_∞ en pratique). Ainsi, un algorithme d'approximation du problème nous suffit. Parmi les différentes méthodes de réduction interne, celle retenue est la méthode Montgomery-like qui permet une complexité sous-quadratique grâce à Karatsuba. Cette méthode introduit le paramètre ϕ, souvent choisi comme une puissance de 2.

La réduction de Montgomery sur les entiers que nous avons introduite plus haut en section 1.2.3 est l'état de l'art pour l'arithmétique modulaire sur les entiers. L'algorithme a été adapté pour les PMNS dans [48] sous forme polynomiale. Pour rappel, le principe de la réduction de Montgomery sur les entiers pour réduire un certain entier $c \in \mathbb{N}$ modulo $m \in \mathbb{N}$ est de chercher un $q \in \mathbb{N}$ tel que $c + qm \equiv 0 \pmod{\phi}$. Ensuite on peut diviser $c + qm$ par ϕ afin d'obtenir une quantité réduite. La version polynomiale se base sur le même principe où, pour réduire un certain polynôme $C \in \mathbb{Z}[X]$, on trouvera $M \in \mathbb{Z}_{n-1}[X]$ tel que $M(\gamma) = kp$, $k \in \mathbb{Z}$. On cherche ensuite $Q(X) \in \mathbb{Z}_{n-1}[X]$ tel que $C(X) + Q(X)M(X) \equiv 0 \pmod{\phi}$. Ainsi une division

par ϕ peut être effectuée pour réduire les coefficients.

Algorithme 11 Réduction interne Montgomery-like Polynomiale

Require: $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS, $C \in \mathbb{Z}[X]$ le polynôme à réduire, $\phi \in \mathbb{N}^*$ et $M \in \mathbb{Z}[X]$ tel que $M(\gamma) \equiv 0 \pmod p$ inversible modulo E modulo ϕ.
Ensure: $C'(\gamma) \equiv C(\gamma)\phi^{-1} \bmod p$
1: $Q(X) \leftarrow -C(X) \times M^{-1}(X) \bmod E(X) \bmod \phi$
2: $C'(X) \leftarrow (C(X) + Q(X) \times M(X) \bmod E(X))/\phi$
3: **return** C'

Remarque 10. L'algorithme 11 nécessite l'utilisation du domaine de Montgomery, en effet, le résultat renvoyé vérifie $C'(\gamma) \equiv C(\gamma)\phi^{-1} \pmod p$. Pour cette raison, lorsque nous souhaitons utiliser l'algorithme après une multiplication il est possible de rajouter un facteur ϕ à chaque opérande. Soit $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS. Soient $a, b \in \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/p\mathbb{Z}$ et $A, B \in \mathcal{B} \times \mathcal{B}$ tels que $A(\gamma) \equiv a\phi \pmod p$ et $B(\gamma) \equiv b\phi \pmod p$, alors $C(X) \equiv A(X)B(X) \pmod{E(X)}$ sera tel que $C(\gamma) \equiv ab\phi^2 \pmod p$. Donc, avec C' la sortie de l'algorithme 11 appliquée à C, on aura $C'(\gamma) \equiv ab\phi \pmod p$, ce qui maintient le domaine de Montgomery. On peut appliquer ce domaine au moment de la conversion sans coût supplémentaire et en sortir pour la conversion dans l'autre sens peut être fait avec une dernière réduction interne.

Remarque 11. Pour la forme de polynôme E non unitaire présentée dans [54] avec $E(X) = aX^n - b$, Plantard propose de remplacer la multiplication par M^{-1} dans l'algorithme 11 par une multiplication par a^2M^{-1} en raison des besoins propres à l'utilisation des polynômes non unitaires soulevés en section 1.5.3, c'est-à-dire le besoin de multiplier par a pour chaque réduction externe par E.

Dosso et al. [20] proposent de choisir les paramètres de telle sorte que $\phi \geqslant 2w\rho$ (avec w le paramètre détaillé en section 1.5.3). En effet, ces bornes garantissent que la sortie de cet algorithme renvoie bien un polynôme C' tel que $\|C'\|_\infty \leqslant \rho$ après une seule application de la réduction interne, du moment que le polynôme C en entrée est le résultat d'une multiplication polynomiale de deux polynômes A et B tels que $\|A\|_\infty < \rho$ et $\|B\|_\infty < \rho$. De plus, Dosso et al. montrent que pour garantir la cohérence des paramètres, on peut prendre $\rho \geqslant 2\|\mathcal{M}\|_1$ avec $\mathcal{M}$ la matrice suivante :

$$\mathcal{M} = \begin{pmatrix} m_0 & m_1 & \dots & m_{n-1} \\ \dots & \dots & \dots & \dots \\ \vdots & \ddots & \ddots & \vdots \\ \dots & \dots & \dots & \dots \end{pmatrix} \begin{matrix} \leftarrow \nu_n(M) \\ \leftarrow \nu_n(X.M \bmod E) \\ \vdots \\ \leftarrow \nu_n(X^{n-1}.M \bmod E) \end{matrix} \tag{1.4}$$

que l'on appellera matrice de réduction interne.

Remarque 12. On remarque que $\mathcal{M}$ est une base d'un sous-réseau de $\mathfrak{L} = \left\{(x_0, \dots, x_{n-1}) \in \mathbb{Z}^n : \sum_{i=0}^{n-1} x_i\gamma^i \equiv 0 \pmod p\right\}$. En effet, on a bien $\nu_n(X^i.M \bmod E) \in \mathfrak{L}$ pour tout $i \in [\![0, n-1]\!]$ car $M(\gamma) \equiv 0 \pmod p$.

L'algorithme suivant est une version de l'algorithme 11 faisant usage des matrices de réduction interne.

Algorithme 12 Réduction interne Montgomery-like Polynomiale forme matricielle

Require: $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS, $C \in \mathbb{Z}[X]$ le polynôme à réduire, $\phi \in \mathbb{N}^*$ et $\mathcal{M}$ comme décrit dans l'équation (1.4).
Ensure: $C'(\gamma) \equiv C(\gamma)\phi^{-1} \bmod p$
1: $Q \leftarrow -\nu_n(C(X)) \cdot \mathcal{M}^{-1} \bmod \phi$
2: $C'(X) \leftarrow (C(X) + \pi_n(Q \cdot \mathcal{M}(X))/\phi$
3: **return** C'

Remarque 13. Outre la variante classique de la réduction de Montgomery, Noyez et al. dans [49] proposent une implémentation de la variante FIOS (cf. section 1.2.3) qui s'avère très adaptée dans un contexte matériel.

Le polynôme M

Le polynôme M est central à la réduction interne. En conséquence, la problématique de trouver un polynôme dans le réseau $\mathfrak{L}$ inversible modulo E modulo ϕ a déjà été longuement explorée. Le principe de base est de considérer une base courte de $\mathfrak{L}$. S'il existe l_i une ligne de la base courte qui est inversible modulo E modulo ϕ, alors on peut choisir $M(X) = \pi_n(l_i)$. Malheureusement, il n'existe pas de résultat général garantissant qu'au moins une des lignes de la matrice de la base courte est telle que son polynôme équivalent par isomorphisme π_n est inversible modulo E modulo ϕ pour E quelconque.

En revanche, il a été démontré dans [15, Propositions 8 et 11] que pour $E(X) = X^n - \lambda$ avec λ pair, au moins une ligne de $\mathbf{B}$ est inversible modulo E modulo ϕ avec ϕ une puissance de 2. Ce résultat reste valide si l'on considère une base courte obtenue en appliquant un algorithme de réduction de base tel que LLL sur $\mathbf{B}$. La question reste ouverte pour λ impair mais dans [15, Proposition 12] et [20, Proposition 7] les auteurs montrent qu'un polynôme M valide peut être trouvé en effectuant une combinaison linéaire binaire des lignes de la matrice $\mathbf{B}$ pour λ impair. Autrement dit il existe un tuple $(\mu_0, \mu_1, \ldots, \mu_{n-1}) \in \mathbb{F}_2^n$ tel que $\pi_n(\sum_{i=0}^{n-1} \mu_i l_i)$ est inversible avec l_i les lignes de $\mathbf{B}$. Encore une fois l'application de LLL ne change pas ce résultat.

De façon générale pour E quelconque M sera nécessairement une combinaison linéaire des lignes d'une base de $\mathfrak{L}$ mais une telle recherche pourrait être en théorie très coûteuse.

Le paramètre δ

Les algorithmes de réduction interne pour un PMNS $\mathcal{B} = (p, n, \gamma, \rho, E)$ quelconque ont une complexité au moins sous-quadratique en n du fait de la présence d'opérations de multiplication polynomiale de degré $n-1$ (donc sur n coefficients).

Soit $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS. Lorsque l'on exécute une série d'additions successives dans les PMNS, il se peut que l'on obtienne un résultat R ne respectant plus $\|R\|_\infty < \rho$. Si la prochaine opération est une multiplication par un élément du PMNS, il faudra donc en premier lieu appliquer une réduction interne à R, effectuer la multiplication et appliquer la réduction interne sur le produit obtenu. Dans [15], Didier et al. introduisent un paramètre δ qui indique que le PMNS construit permet de n'effectuer qu'une seule réduction interne après au plus δ additions successives suivies d'une multiplication. Dans ce cas, il est alors nécessaire de choisir $\phi \geqslant 2w\rho(1+\delta)^2$ [20, Proposition 4].

1.5.5 Opérations de conversion

Le Polynomial Modular Number System a pour but principal l'accélération des calculs dans l'arithmétique modulaire. Soit $\mathcal{B} = (p, n, \gamma, \rho, E)$, un PMNS. Il faut pouvoir convertir tous les éléments de $\mathbb{Z}/p\mathbb{Z}$ en un élément de $\mathcal{B}$. Ainsi, toutes les opérations nécessaires pourront être effectuées dans le PMNS. L'algorithme suivant a été proposé dans [17, Algorithme 22].

Algorithme 13 Conversion du binaire à l'AMNS [17]

Require: $a \in \mathbb{Z}/p\mathbb{Z}$, $\mathcal{B} = (p,\ n,\ \gamma,\ \rho, E)$ un PMNS, $P_0, P_1, \dots P_{n-1} \in \mathcal{B}^n$ pré-calculés tels que $\forall i \in [\![0, n-1]\!]$, $P_i(\gamma) \equiv (\rho^i \phi^2) \pmod p$, $\phi \in \mathbb{N}^*$.
Ensure: $A(X) \in \mathcal{B}$ tel que $A(\gamma) \equiv a\phi \pmod p$
1: $t \leftarrow (a_0, a_1, \dots, a_{n-1})$ # décomposition en base ρ de a
2: $U \leftarrow \sum_{i=0}^{n-1} a_i P_i(X)$
3: $A \leftarrow$ **RedCoeff**(U)
4: **return** A

Ici, **RedCoeff** fait référence à l'algorithme 11, ou autrement dit la réduction Montgomery-like polynomiale. Pour tout i, $\|P_i\|_\infty < \rho$ car $P_i \in \mathcal{B}$ par supposition. Ainsi, $\|U\|_\infty < n\rho^2$. En conséquence, [17, Corollaire 2] nous permet de conclure que $\|A\|_\infty < \rho$ comme voulu. De plus,

$$U(\gamma) \equiv \sum_{i=0}^{n-1} a_i P_i(\gamma) \equiv \sum_{i=0}^{n-1} a_i \rho^i \phi^2 \equiv a\phi^2 \pmod p$$

donc on a bien $A(\gamma) \equiv a\phi \pmod p$.

Une fois que tous les calculs sont finis, il faut pouvoir convertir un élément de $\mathcal{B}$ en l'unique élément de $\mathbb{Z}/p\mathbb{Z}$ qui lui est associé. Pour ce faire, l'algorithme suivant a été proposé dans [17, Algorithme 25].

Algorithme 14 Conversion de l'AMNS au binaire[17]

Require: $\mathcal{B} = (p,\ n,\ \gamma,\ \rho, E)$ un PMNS, $C \in \mathcal{B}$ avec $(c_0, c_1, \dots, c_{n-1})$ ses coefficients, $\phi \in \mathbb{N}^*$, $g_0, g_1, \dots, g_{n-1} \in (\mathbb{Z}/p\mathbb{Z})^n$ pré-calculés tels que $\forall i \in [\![0, n-1]\!]$, $g_i \equiv \gamma^i \pmod p$.
Ensure: $c \equiv C(\gamma)\phi^{-1} \pmod p$
1: $C \leftarrow$ **RedCoeff**(C) # On sort du domaine associé
2: $c \leftarrow c_0$
3: **for** $i = 1 \dots n-1$ **do**
4: $\quad c \leftarrow c + c_i g_i$
5: **end for**
6: $c \leftarrow c \bmod p$
7: **return** c

Dans l'algorithme 14, **RedCoeff** fait référence à la réduction Montgomery-like polynomiale (algorithme 11). Cet algorithme s'avère plus rapide qu'une évaluation polynomiale de Horner (algorithme 10). Le paramètre γ étant propre au système, on peut calculer ses puissances modulo p une seule fois, amortissant ainsi le coût pour toutes les opérations futures. Le réel avantage ici est qu'une seule opération de réduction modulaire doit être effectuée en dernière étape, avec $c \leqslant (n-1)(\rho-1)p + (\rho-1)$ en sortie de la boucle commencée en ligne 3.

1.5.6 Test d'égalité

Les tests d'égalité en PMNS sont non triviaux en raison de la redondance inhérente au système. Ainsi, une simple comparaison terme à terme ne suffit pas. Par souci de simplicité, on suppose $\phi = \sigma$ pour le reste de cette section.

Soit $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS et soient $A, B \in \mathcal{B}$, on peut simplement calculer $a \equiv A(\gamma) \pmod p$ et $b \equiv B(\gamma) \pmod p$ et tester si $a = b$. Une solution plus rapide serait de tester si $(A - B)(\gamma) \equiv 0 \pmod p$ afin de n'effectuer qu'une seule évaluation en γ. Effectuer $A - B$ a pour coût n soustractions élémentaires. Ensuite, nous pouvons utiliser l'algorithme 14 pour convertir le résultat. Nous n'avons pas besoin d'effectuer la réduction interne au début de l'algorithme car celle-ci a pour fonction de sortir du domaine de Montgomery. Ceci n'est pas utile pour un test d'égalité car si $A(\gamma) \equiv B(\gamma) \pmod p$, $(A - B)(\gamma) \equiv 0 \pmod p$ même avec le facteur ϕ supplémentaire. Nous pouvons donc commencer à partir de la ligne 2 de l'algorithme. Ainsi $n - 1$ multiplications par des nombres tenant sur au plus n registres mémoire sont effectuées donc au plus $n(n - 1)$ multiplications élémentaires suivies d'au plus $(n + 1)(n - 1)$ additions élémentaires pour sommer. Pour finir, une réduction modulaire par p qui coûtera au plus l'équivalent de $2n$ multiplications au vu du fait que le nombre à réduire tient sur au plus $n + 1$ registres mémoire. On peut résumer le total à $n^2 + n$ multiplications, n soustractions et $n^2 - 1$ additions.

Une autre méthode proposée dans [18, Algorithme 13] consiste à prendre $\phi > \lceil 2w(\|\mathcal{G}\|_1)^2\|\mathcal{G}^{-1}\|_1 \rceil$ au lieu de $\phi > 2w\rho$ avec $\rho \geqslant 2\|\mathcal{M}\|_1$ (avec $\mathcal{M}$ la matrice de l'équation (1.4) et $\mathcal{G}$ une base courte du réseau $\mathfrak{L}$, cf. équation (1.1)). Les bornes sont ainsi élargies. En échange, nous pouvons effectuer de façon efficace un test d'égalité lorsque les opérandes à tester sont chacune bornées par $\frac{1}{2}w(\rho - 1)^2$. Après une multiplication modulaire suivie d'une réduction externe, un polynôme sera borné par $w(\rho - 1)^2$ donc on peut supposer que cette méthode peut être utilisée dans certains cas.

Algorithme 15 Test d'égalité dans un PMNS[18]

Require: $A, B \in \mathbb{Z}_{n-1}[X] \times \mathbb{Z}_{n-1}[X]$ avec $\|A\|_\infty < l$ et $\|B\|_\infty < l$ pour $l = \frac{1}{2}w(\rho-1)^2$, $\mathcal{T} = (-u, -u, \ldots, -u)$ avec $u = \lceil w(\rho - 1)^2\|\mathcal{G}^{-1}\|_1 \rceil$.
Ensure: $S = 0$ si et seulement si $A(\gamma) \equiv B(\gamma) \pmod p$.
1: $C \leftarrow (A - B) + \pi_n(\mathcal{T})$
2: $S \leftarrow$ **RedCoeff**(C)
3: **return** S

L'idée sous-jacente est d'effectuer une translation sur $\nu_n(A - B)$ grâce au vecteur $\mathcal{T}$ choisi de telle sorte que le résultat soit dans un sous-domaine du réseau. À l'intérieur de ce domaine, les auteurs de [18] démontrent que chaque entier est représenté de façon unique par un polynôme du PMNS. Ainsi, l'entier 0 aura pour unique représentation le polynôme nul.

Les polynômes A et B ont des coefficients tenant a priori sur 2 registres, étant de l'ordre de ρ^2 et non ρ, donc l'opération $(A - B) + \pi_n(\mathcal{T})$ coûtera $4n$ additions/soustractions élémentaires. À cela se rajoute le coût de l'algorithme 12 utilisé ici pour **RedCoeff**, à savoir $2n^2$ multiplications et $2n(n - 1) + 2n$ additions (pour $\phi = \sigma$). Le coût total de l'algorithme 15 est donc de $2n^2$ multiplications et $2n^2 + 4n$ additions, que nous pouvons aussi voir comme 2 opérations de multiplication vecteur-matrice et $4n$ additions.

Remarque 14. L'algorithme présenté dans [18] utilise une base courte de $\mathfrak{L} = \left\{(x_0, \ldots, x_{n-1}) \in \mathbb{Z}^n : \sum_{i=0}^{n-1} x_i \gamma^i \equiv 0 \pmod{p}\right.$ et non $\mathcal{M}$. En effet, $\mathcal{M}$ est une base d'un sous-réseau de $\mathfrak{L}$ donc on le note comme ceci pour l'instant par souci de simplification. Cette distinction est détaillée dans le chapitre 2.

1.5.7 Génération

Le processus de génération de PMNS est décrit dans [52] avec l'algorithme suivant.

Algorithme 16 Création d'un système de représentation adapté

Ensure: $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS.
1: Choisir la taille des chiffres k.
2: Choisir le nombre de chiffres n pour la taille cryptographique nk telle que $p < 2^{nk}$
3: **while** $\mathcal{B}$ ne convient pas **do**
4: Choisir le polynôme unitaire E de degré n.
5: Choisir M tel que ses coefficients soient petits ou nuls.
6: Construire $\mathcal{M}$ (cf. équation (1.4)).
7: $p \leftarrow \det(2^k I_n - \mathcal{M})$
8: **if** p premier **then**
9: $\gamma \leftarrow$ racine PGCD$(E, 2^k - M)$
10: Choisir $\rho = 2^k$ ou $\rho = 2^{k+1}$
11: **end if**
12: **end while**

Les notions telles que la matrice de réduction interne de l'équation (1.4) sont un léger anachronisme car introduites plus tard mais sont équivalentes à ce qui était utilisé en pratique. Cet algorithme est présenté avec ces notations afin de garder des notations consistantes. Le choix de $\rho = 2^k$ ou $\rho = 2^{k+1}$ est laissé à l'appréciation de l'utilisateur selon si l'on préfère minimiser le coût mémoire ou limiter le nombre d'appels à la réduction interne, qui sera ici peu coûteuse si l'on choisit M avec des coefficients petits ou nuls comme indiqué. Cela sous-entend qu'un seul appel à la réduction interne n'est pas suffisant, mais avec M suffisamment creux la complexité de la réduction est linéaire et non quadratique. Le but de cet algorithme est de générer un PMNS en trouvant un nombre premier p de forme très particulière dont la taille est définie en paramètre et non de construire un PMNS sur un entier premier p fixé.

Un processus de génération de PMNS pour un p fixé en paramètre est proposé dans [15, Section 5.1] bien qu'aucun algorithme explicite ne soit fourni. Ce processus fixe également la forme de polynôme E considérée comme $X^n - \lambda$, $\lambda \in \mathbb{Z}^*$. Pour paraphraser :

1. On fixe p comme paramètre, supposé premier et δ comme autre paramètre (cf. section 1.5.4).
2. On choisit n tel que $n \geqslant n_{opt}$ (cf. section 1.5.2).
3. On choisit/trouve $\lambda \in \mathbb{Z}^*$ tel qu'il existe $\gamma \in \mathbb{Z}/p\mathbb{Z}$ tel que $\gamma^n \equiv \lambda \pmod{p}$.
4. On pose $E(X) = X^n - \lambda$ avec ce λ.
5. On génère M tel qu'il soit inversible modulo E modulo ϕ (cf. section 1.5.4).
6. Les valeurs de ρ et ϕ sont calculées comme $\rho = 2^{\lceil \log_2(2n\lvert\lambda\rvert\lVert M\rVert_\infty)\rceil}$ et $\phi = 2^{\lceil \log_2(2n\lvert\lambda\rvert\rho(\delta+1)^2)\rceil}$.

Nous avons ainsi un PMNS (p, n, γ, ρ, E). En particulier, il se peut que ρ et ϕ dépassent σ. Ainsi, pour des raisons d'efficacité, on peut imposer d'augmenter le paramètre n de façon incrémentale afin de réduire les

valeurs de ρ et ϕ. Ceci est dû au fait que l'on considère $\|M\|_\infty$ inversement proportionnel avec n. Cela s'explique par le fait que M est un vecteur court d'un réseau de volume p. Augmenter n augmente la dimension du réseau sans modifier le volume qui est fixé à p. En conséquence, augmenter la valeur de n réduit la valeur de $\|M\|_\infty$.

Remarque 15. On remarque qu'on peut effectuer le même processus sur un entier p de taille fixe généré aléatoirement en entrée.

Certaines optimisations sur les bornes ont été effectuées depuis par rapport au processus présenté ici, notamment dans [20] qui propose de prendre plutôt $\rho \geqslant 2\|\mathcal{M}\|_1$ et $\phi \geqslant 2w\rho(\delta+1)$. En considérant que ρ est pris comme une puissance de 2 par Dosso et al., cela revient à prendre $\rho = 2^{\log_2(\|\mathcal{M}\|_1)+1}$ tant que $2w\rho(\delta+1) \leqslant \sigma$, sinon on devra incrémenter n pour faire diminuer $\|\mathcal{M}\|_1$.

En ce qui concerne le choix du polynôme E, on peut remplacer les lignes 3 et 4 ci-dessus par une recherche de racine dans $\mathbb{Z}/p\mathbb{Z}$ pour E de la forme voulue (par exemple $E(X) = X^n \pm X \pm 1$).

Nous présentons des améliorations sur le processus de génération en section 2.5.

1.5.8 Exemples de PMNS

Pour clôturer ce chapitre, on présente ici un PMNS construit sur un petit entier premier pour exemple. Nous prendrons ici $\mathcal{B} = (p = 11, n = 3, \gamma = 7, \rho = 2, E = X^3 - 2)$. Nous pouvons énumérer tous les éléments de ce PMNS et les entiers qu'ils représentent. Nous le faisons dans le tableau 1.3.

0	0	X^2+X-1	$-X^2-X+1$
1	1	X^2+X	$-X^2+X-1$
2		X^2+X+1	$-X^2+X$
3	$-X-1$		$-X^2-X+1$
4	$-X$	X^2-1	
5	$-X+1$	X^2	$-X^2-1$
6	$X-1$	X^2+1	$-X^2$
7	X		$-X^2+1$
8	$X+1$	X^2-X-1	
9		X^2-X	$-X^2-X-1$
10	-1	X^2-X+1	$-X^2-X$

TABLE 1.3 – Tableau des éléments du PMNS $\mathcal{B} = (p = 11, n = 3, \gamma = 7, \rho = 2, E = X^3 - 2)$.

Comme on peut le voir, chaque élément de $\mathbb{Z}/11\mathbb{Z}$ possède au moins 2 représentants et certains en possèdent 3. Si on prend par exemple $a = 8$, on peut choisir comme représentant $A(X) = X + 1$ ou $A(X) = X^2 - X - 1$. Mettons que l'on choisisse le second, si l'on considère maintenant $b = 6$ que l'on représente par $B(X) = X - 1$, on peut effectuer une opération de multiplication comme suit :

$$C(X) = A(X)B(X) = (X^2 - X - 1)(X - 1) = X^3 - X^2 - X - X^2 + X + 1 = X^3 - 2X^2 + 1$$

$C(X)$ est de degré strictement supérieur à 2 donc on doit réduire son degré avec une réduction externe.

$$C'(X) = C(X) - E(X) = X^3 - 2X^2 + 1 - X^3 + 2 = -2X^2 + 3$$

On a bien ici $C'(X) \equiv C(X) \pmod{E(X)}$ et $C'(\gamma) \equiv C(\gamma) \pmod{p}$ car $E(\gamma) \equiv 0 \pmod{p}$. Ensuite, les coefficients étant trop gros, il faut effectuer une réduction interne. Sur un exemple aussi petit, il n'est pas nécessaire d'effectuer une réduction Montgomery-like. En effet, le polynôme le plus proche s'annulant en γ est $T(X) = -2X^2 + X + 3$ car

$$T(7) \equiv -2 \times 49 + 7 + 3 \equiv -88 \equiv 0 \pmod{11}.$$

Ainsi

$$C''(X) = C'(X) - T(X) = -2X^2 + 3 - (-2X^2 + X + 3) = -X$$

est bien un représentant de $a \times b$ car $8 \times 6 \equiv 48 \equiv 4 \pmod{11}$ et $-X$ est bien un représentant de 4.

On considère maintenant le PMNS $\mathcal{B}' = ($
$p = 726589512580075827165577815940205655126075658084463158895150129844038996753 09,$
$n = 5,$
$\gamma = 472336245400502276405725161213198813429259421923382283164897368044854918329 57,$
$\rho = 18014398509481984,$
$E(X) = X^5 - 2)$. On choisit ici

$$a = 553014175709862106646046373760907175781800949587855615929796726510483336976 56$$

et

$$b = 231404683068470016117638736449097732060102988381642652505899927137960325858 71$$

dont les représentants sont

$$A(X) = -16418323151436339X^4 + 276252764540687X^3 + 7437842018938432X^2 + 15470279641088753X - 2872875599240331$$

et

$$B(X) = 8415344909652960X^4 + 11618420586034114X^3 + 4878049157314445X^2 - 2466769851259518X + 13065840613549045.$$

Si l'on effectue la multiplication de a par b, on aura d'abord pour $C(X) \equiv A(X)B(X) \pmod{E(X)}$,

$$\begin{aligned} C(X) = &- 233545100329882424298044453147 99X^4 - 248983166611859065440149246043390X^3 \\ &- 331854432266834636863040666055161X^2 + 180643409684092552076393950487709X \\ &+ 479366495278812985529003803418995. \end{aligned}$$

Il est ainsi beaucoup plus dur de trouver le vecteur proche

$$\begin{aligned} T(X) = &- 233545100329882434526204409113 33X^4 - 248983166611859065729942493139204X^3 \\ &- 331854432266834636841312063328820X^2 + 180643409684092553155799694354599X \\ &+ 479366495278812986613056655149816 \end{aligned}$$

qui s'annule en γ modulo p. On comprend ainsi l'intérêt de la méthode de réduction Montgomery-like.

Chapitre 2

Nouvelles bornes sur les paramètres et implémentations efficaces

2.1 Notations

- $\mathbb{Z}_m[X]$ le $\mathbb{Z}$-module des polynômes à coefficients entiers de degré au plus m.
- $\nu_n : \mathbb{Z}_{n-1}[X] \to \mathbb{Z}^n$ l'isomorphisme tel que $\nu_n(a_0 + a_1X + \cdots + a_{n-1}X^{n-1}) = (a_0, a_1, \ldots, a_{n-1})$ pour un n donné.
- $\pi_n : \mathbb{Z}^n \to \mathbb{Z}_{n-1}[X]$ l'isomorphisme tel que $\pi_n((a_0, a_1, \ldots, a_{n-1})) = a_0 + a_1X + \cdots + a_{n-1}X^{n-1}$ pour un n donné.
- σ le nombre de valeurs possibles dans un registre mémoire (2^{64} pour la plupart des CPU modernes).
- $M_{i,j}$ pour M une matrice représente le coefficient situé sur la $i$ème ligne et la $j$ème colonne, en partant de 0. Une matrice M de dimension $n \times m$ aura donc des coefficients de $M_{0,0}$ à $M_{n-1,m-1}$.
- $\lfloor a \rceil$ pour $a \in \mathbb{R}$ représente la valeur arrondie de a, équivalente à $\lfloor a + \frac{1}{2} \rfloor$.
- Pour $v = (v_0, v_1, \ldots, v_{n-1}) \in \mathbb{R}^n$, on aura $\lfloor v \rceil = (\lfloor v_0 \rceil, \lfloor v_1 \rceil, \ldots, \lfloor v_{n-1} \rceil)$.
- Pour $M \in \mathcal{M}_n(\mathbb{R})$, on notera $\lfloor M \rceil$ la matrice telle que $\forall i, j \in [\![0, n-1]\!] \times [\![0, n-1]\!]$, $\lfloor M \rceil_{i,j} = \lfloor M_{i,j} \rceil$.
- Un PMNS $\mathcal{B}$ est un tuple (p, n, γ, ρ, E) représentant les entiers de $\mathbb{Z}/p\mathbb{Z}$ par des polynômes sur n coefficients (donc de degré au plus $n-1$). Un entier a est représenté par un polynôme $A(X)$ si $A(\gamma) \equiv a \pmod{p}$. Le paramètre ρ représente la borne sur la taille des coefficients polynomiaux des éléments du PMNS. Pour réduire le degré après multiplication, le polynôme E de degré n qui s'annule en γ modulo p est utilisé.

2.2 Introduction

Cette section présente diverses contributions relatives aux PMNS. Servant de base aux chapitres ultérieurs, ce chapitre établit des principes applicables à tous les PMNS, contrairement aux chapitres suivants qui se concentrent sur des aspects plus spécifiques.

Nous commencerons par examiner les éléments liés à la Réduction Externe, notamment l'utilisation de polynômes E non unitaires, l'évolution du paramètre w qui en découle, et la mise en œuvre d'algorithmes sous-quadratiques exploitant les matrices de Toeplitz pour certains polynômes E.

Nous traiterons ensuite de la Réduction Interne, en mettant l'accent sur l'évolution des bornes grâce à l'arithmétique signée en machine, une version légèrement modifiée de l'algorithme de Réduction Interne Barrett-like, et une nouvelle forme de Réduction Interne que nous nommerons Plantard-like.

Par la suite, nous approfondirons la question de la génération de paramètres, en particulier du point de vue du choix d'un polynôme M pour la Réduction Interne, dont tous les coefficients sont positifs, afin de l'utiliser avec le jeu d'instructions AVX512IFMA. Ce jeu d'instructions permet d'effectuer des opérations SIMD (Single Instruction Multiple Data), ce qui parallélise les calculs indépendants et mène à une accélération lorsqu'elles sont disponibles.

Enfin, nous réexaminerons les problèmes de conversion et de test d'égalité dans un PMNS à l'aide des outils présentés dans les autres sections.

2.3 Réduction externe

Dans cette section sont détaillées les contributions diverses sur la réduction externe réalisées au cours de cette thèse.

2.3.1 Polynôme E non unitaire

Dans cette section, nous explicitons les problèmes inhérents à l'utilisation d'un E non unitaire. Nous avons pu voir dans la section 1.5.3 que cette forme n'est généralement pas considérée. Nous allons d'abord détailler les problèmes liés à ce choix.

Pour commencer, utiliser un polynôme E non unitaire complique l'étape de réduction externe et mènera à des polynômes dont on ne peut pas réduire le degré, ce que nous exprimons explicitement dans la proposition suivante.

Proposition 2.3.1. *Soit $E \in \mathbb{Z}[X]$ tel que $E(X) = e_0 + e_1X + \dots e_nX^n$ avec $|e_n| > 1$, $\exists (A, B) \in \mathbb{Z}_{n-1}[X] \times \mathbb{Z}_{n-1}[X]$, tel que $\nexists C \in \mathbb{Z}_{n-1}[X]$ tel que $C(X) \equiv A(X) \times B(X) \pmod{E(X)}$.*

Démonstration. On pose $A(X) = X^{n-1}$ et $B(X) = X^{n-1}$. Alors $A(X) \times B(X) = X^{2n-2}$.
$e_n \neq 1$ donc $\nexists k \in \mathbb{Z}$ tel que

$$\deg(X^{2n-2} - kX^{n-2} \times E(X)) < 2n - 2.$$

Ainsi, $\forall C \in \mathbb{Z}[X]$ tel que $C(X) \equiv A(X) \times B(X) \pmod{E(X)}$, $\deg(C) \geqslant 2n - 2$.
En conséquence, $\nexists C \in \mathbb{Z}_{n-1}[X]$ tel que $C(X) \equiv A(X) \times B(X) \pmod{E(X)}$. □

Comme nous l'avons déjà précisé dans la section 1.5.3, une adaptation possible est détaillée dans [54] pour un polynôme E très creux. Nous étendons dans la proposition ci-dessous ce résultat à un polynôme E quelconque.

Proposition 2.3.2. *Soit $E \in \mathbb{Z}[X]$ tel que $E(X) = e_0 + e_1X + \dots e_nX^n$ avec $|e_n| > 1$, alors $\forall (A, B) \in \mathbb{Z}_{n-1}[X] \times \mathbb{Z}_{n-1}[X]$, $\exists C \in \mathbb{Z}_{n-1}[X]$ tel que $C(X) \equiv (e_n)^{n-1}A(X) \times B(X) \pmod{E(X)}$.*

Démonstration. On prend $(A, B) \in \mathbb{Z}_{n-1}[X] \times \mathbb{Z}_{n-1}[X]$ quelconques. On pose $R(X) = A(X) \times B(X)$. On a

$$R(X) = r_0 + r_1X + \dots + r_{n-1}X^{n-1} + r_nX^n + \dots + r_{2n-2}X^{2n-2}$$

avec $r_i = \sum_{j=0}^{i} a_j b_{i-j}$ pour $i \in [\![0, n-1]\!]$ et $r_i = \sum_{j=i-n+1}^{n-1} a_j b_{n-j}$ pour $i \in [\![n, 2n-2]\!]$.
Alors

$$R^{(1)}(X) = e_n R(X) - r_{2n-2} X^{n-2} E(X)$$

est tel que $\deg(R^{(1)}(X)) \leqslant 2n-3$ car le coefficient de degré $2n-2$ est annulé. De plus,

$$R^{(1)}(X) \equiv e_n A(X) \times B(X) \pmod{E(X)}.$$

Ensuite on pose

$$R^{(2)}(X) = e_n R^{(1)}(X) - r^{(1)}_{2n-3} X^{n-3} E(X).$$

Remarquons que

$$R^{(2)}(X) \equiv e_n R^{(1)}(X) \pmod{E(X)}$$

donc

$$R^{(2)}(X) \equiv (e_n)^2 A(X) \times B(X) \pmod{E(X)}.$$

De plus on a annulé le coefficient de degré $2n-3$ de $R^{(1)}(X)$ donc $\deg(R^{(2)}(X)) \leqslant 2n-4$. On définit ensuite $R^{(i)}(X)$ pour $i > 1$ comme

$$R^{(i)}(X) = e_n R^{(i-1)}(X) - r^{(i-1)}_{2n-2-(i-1)} X^{n-2-(i-1)} E(X).$$

En conséquence, $\deg(R^{(i)}(X)) \leqslant \max(2n-2-i, n-1)$ et

$$R^{(i)} \equiv (e_n)^i A(X) \times B(X) \pmod{E(X)}.$$

On pose $C(X) = R^{(n-1)}(X)$. Alors $\deg(C) \leqslant n-1$, donc $C \in \mathbb{Z}_{n-1}[X]$ et

$$C(X) \equiv (e_n)^{n-1} A(X) \times B(X) \pmod{E(X)}.$$

□

Remarque 16. Un des problèmes de cette méthode pour un E quelconque est l'ajout d'un facteur $(e_n)^{n-1}$ au résultat de l'évaluation en γ. Tout comme pour la solution proposée en [54], on peut ici considérer le domaine où les représentants sont multipliés par $(e_n)^{-(n-1)}$ au moment de la conversion.

Soit $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS avec $E(X) = e_n X^n + \cdots + e_1 X + e_0$. Soient $a, b \in \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/p\mathbb{Z}$ et $A, B \in \mathcal{B} \times \mathcal{B}$ tels que $A(\gamma) \equiv (e_n)^{-(n-1)} a \pmod{p}$ et $B(\gamma) \equiv (e_n)^{-(n-1)} b \pmod{p}$, alors

$$C(X) \equiv (e_n)^{n-1} A(X) B(X) \pmod{E(X)}$$

d'où

$$C(\gamma) \equiv (e_n)^{-(n-1)} ab \pmod{p},$$

restant ainsi dans le même domaine. On note que ceci est similaire au domaine de Montgomery déjà présenté en section 1.5.4 et peut être utilisé en conjugaison en prenant $A(\gamma) \equiv (e_n)^{-(n-1)} a\phi \pmod{p}$ et $B(\gamma) \equiv$

$(e_n)^{-(n-1)}b\phi \pmod p$ pour avoir

$$C(\gamma) \equiv (e_n)^{-(n-1)}ab\phi \pmod p$$

après réduction interne.

Pour des raisons de performances, soient $A \in \mathbb{Z}[X]$ et $B \in \mathbb{Z}[X]$, nous devons nous assurer que les coefficients de $C \in \mathbb{Z}[X]$, qui est tel que

$$C(X) \equiv (e_n)^{n-1}A(X)B(X) \pmod{E(X)},$$

tiennent sur au plus deux registres mémoire. Ainsi, plus $(e_n)^{n-1}$ est grand, plus les coefficients de A et B devront être réduits en conséquence. Cela se traduit généralement par l'augmentation du paramètre n pour obtenir un paramètre ρ plus petit. Si le polynôme E possède une forme particulière, une autre méthode permet de réduire grandement ce facteur supplémentaire.

Proposition 2.3.3. *Soit $E \in \mathbb{Z}[X]$ tel que $E(X) = e_0 + e_1X + \dots e_nX^n$ tel que $\forall i \in [\![2, n-1]\!]$, e_n divise e_i, alors $\forall (A, B) \in \mathbb{Z}_{n-1}[X] \times \mathbb{Z}_{n-1}[X]$, $\exists C \in \mathbb{Z}_{n-1}[X]$ tel que $C(X) \equiv e_nA(X) \times B(X) \pmod{E(X)}$.*

Démonstration. On prend $(A, B) \in \mathbb{Z}_{n-1}[X] \times \mathbb{Z}_{n-1}[X]$ quelconques. On pose $R(X) = A(X) \times B(X)$. On a

$$R(X) = r_0 + r_1X + \cdots + r_{n-1}X^{n-1} + r_nX^n + \cdots + r_{2n-2}X^{2n-2}$$

avec $r_i = \sum_{j=0}^{i} a_jb_{i-j}$ pour $i \in [\![0, n-1]\!]$ et $r_i = \sum_{j=i-n+1}^{n-1} a_jb_{n-j}$ pour $i \in [\![n, 2n-2]\!]$.
On pose $s_1 = r_{2n-2}$. Alors

$$R^{(1)}(X) = e_nR(X) - s_1X^{n-2}E(X)$$

est tel que $\deg(R^{(1)}(X)) \leqslant 2n-3$ car le coefficient de degré $2n-2$ est annulé. De plus,

$$R^{(1)}(X) \equiv e_nA(X) \times B(X) \pmod{E(X)}.$$

On a

$$R^{(1)}(X) = r_0^{(1)} + r_1^{(1)}X + \cdots + r_{n-1}^{(1)}X^{n-1} + r_n^{(1)}X^n + \cdots + r_{2n-3}^{(1)}X^{2n-3}$$

avec $\forall i \in [\![0, n-3]\!]$, $r_i^{(1)} = e_nr_i$ et $\forall i \in [\![n-2, 2n-3]\!]$, $r_i^{(1)} = e_nr_i - e_{i-n+2}s_1$ donc en particulier $r_{2n-3}^{(1)} = e_nr_{2n-3} - e_{n-1}s_1$. Puisque e_n divise e_{n-1}, on peut écrire $r_{2n-3}^{(1)} = e_n(r_{2n-3} - \frac{e_{n-1}}{e_n}s_1)$ avec $(r_{2n-3} - \frac{e_{n-1}}{e_n}s_1)$ un entier. On pose $s_2 = r_{2n-3} - \frac{e_{n-1}}{e_n}s_1$ et

$$R^{(2)}(X) = e_nR(X) - s_1X^{n-2}E(X) - s_2X^{n-3}E(X) = R^{(1)}(X) - s_2X^{(n-3)}E(X)$$

On a $\deg(R^{(2)}(X)) \leqslant 2n-4$ et

$$R^{(2)}(X) \equiv R^{(1)}(X) \equiv e_nA(X) \times B(X) \pmod{E(X)}.$$

De façon similaire à précédemment, on a $r_{2n-4}^{(2)} = e_nr_{2n-4} - e_{n-2}s_1 - e_{n-1}s_2$. Puisque e_n divise e_{n-2} et e_{n-1}, $s_3 = r_{2n-4} - \frac{e_{n-2}}{e_n}s_1 - \frac{e_{n-1}}{e_n}s_2$ est un entier.

On définit ensuite pour $i \geqslant 1$

$$R^{(i)}(X) = R^{(i-1)}(X) - s_i X^{n-2-(i-1)} E(X)$$

avec $s_i = r_{2n-1-i} - \sum_{j=1}^{i-1} \frac{e_{n-j}}{e_n} s_{i-j}$.
On a $\deg(R^{(i)}(X)) \leqslant \max(2n-2-i, n-1)$ et

$$R^{(i)}(X) \equiv e_n A(X) \times B(X) \pmod{E(X)}.$$

On pose $C(X) = R^{(n-1)}(X)$. Alors $\deg(C) \leqslant n-1$, donc $C \in \mathbb{Z}_{n-1}[X]$ et

$$C(X) \equiv e_n A(X) \times B(X) \pmod{E(X)}.$$

□

Remarque 17. On a $R^{(n-1)}(X) = R^{(n-2)}(X) - s_{n-1}E(X)$ avec

$$s_{n-1} = r_n - \sum_{j=1}^{n-2} \frac{e_{n-j}}{e_n} s_{n-1-j}$$

donc il faut que e_n divise tous les e_i jusqu'à e_2 compris. Il n'est donc pas nécessaire que e_n divise e_1.

Remarque 18. En particulier un polynôme E de la forme $E(X) = e_n X^n + e_0$ (mêmes conditions que dans [54]) vérifie bien les conditions de la proposition 2.3.3. On utilisera la notation $E(X) = \alpha X^n - \lambda$ pour la suite afin de garder la cohérence avec $E(X) = X^n - \lambda$.

Remarque 19. Cette forme non unitaire du polynôme E nécessite l'utilisation d'un domaine particulier, comme celui détaillé en remarque 16. En effet, soit $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS avec $E(X) = e_n X^n + \cdots + e_1 X + e_0$ avec $\forall i \in [\![2, n-1]\!]$, e_n divise e_i. Soient $a, b \in \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/p\mathbb{Z}$ et $A, B \in \mathcal{B} \times \mathcal{B}$ tels que $A(\gamma) \equiv (e_n)^{-1} a \pmod{p}$ et $B(\gamma) \equiv (e_n)^{-1} b \pmod{p}$, alors $C(X) \equiv e_n A(X) B(X) \pmod{E(X)}$ sera tel que $C(\gamma) \equiv (e_n)^{-1} ab \pmod{p}$, restant ainsi dans le même domaine.

De façon similaire au facteur $(e_n)^{n-1}$, nous pouvons conjuguer ce domaine au domaine de Montgomery en prenant $A(\gamma) \equiv (e_n)^{-1} a\phi \pmod{p}$ et $B(\gamma) \equiv (e_n)^{-1} b\phi \pmod{p}$ pour obtenir $C(\gamma) \equiv (e_n)^{-1} ab\phi \pmod{p}$ après réduction interne.

Les restrictions sur la divisibilité des coefficients du polynôme E par son coefficient de plus haut degré sont très contraignantes mais malheureusement nécessaires, comme on peut le voir avec la proposition suivante.

Proposition 2.3.4. *Soit $E \in \mathbb{Z}[X]$ avec $E(X) = e_0 + e_1 X + \ldots e_n X^n$ pour lequel $\exists i \in [\![2, n-1]\!]$ tel que e_n ne divise pas e_i. Alors $\exists (A, B) \in \mathbb{Z}_{n-1}[X] \times \mathbb{Z}_{n-1}[X]$, tel que $\nexists C \in \mathbb{Z}_{n-1}[X]$ vérifiant $C(X) \equiv e_n A(X) \times B(X) \pmod{E(X)}$.*

Démonstration. Soit $E \in \mathbb{Z}[X]$ avec $E(X) = e_0 + e_1 X + \ldots e_n X^n$. On pose i le plus haut degré tel que e_n ne divise pas e_i. Ainsi, $\forall j > i$, e_n divise e_j.
On pose $A(X) = e_i X^{n-1}$ et $B(X) = X^{n-1} + \frac{e_{n-1}}{e_n} X^{n-2} + \cdots + \frac{e_{i+1}}{e_n} X^i + X^{i-1}$. Alors

$$e_n A(X) \times B(X) = (e_i e_n) X^{2n-2} + (e_i e_{n-1}) X^{2n-3} + \cdots + (e_i e_{i+1}) X^{n+i-1} + (e_i e_n) X^{n+i-2}.$$

D'où

$$e_nA(X) \times B(X) - e_iX^{n-2}E(X) = (e_ne_i - (e_i)^2)X^{n+i-2} - e_iX^{n-2}(e_0 + e_1X + \cdots + e_{i-1}X^{i-1}).$$

e_n ne divise pas e_i donc $\nexists P(X) \in \mathbb{Z}[X]$ tel que

$$\deg((e_ne_i - (e_i)^2)X^{n+i-2} - e_iX^{n-2}(e_0 + e_1X + \cdots + e_{i-1}X^{i-1}) - P(X) \times E(X)) < n + i - 2.$$

Ainsi, $\forall C \in \mathbb{Z}[X]$ tels que

$$C(X) \equiv e_nA(X) \times B(X) \pmod{E(X)},$$

$\deg(C) \geqslant n + i - 2$. En particulier puisque l'on a supposé $i > 1$, $\deg(C) > n - 1$.
En conséquence, $\nexists C \in \mathbb{Z}_{n-1}[X]$ tel que

$$C(X) \equiv e_nA(X) \times B(X) \pmod{E(X)}.$$

□

2.3.2 Le paramètre w

Dans cette section, nous adaptons très légèrement le paramètre w présenté en section 1.5.3 pour prendre en compte les polynômes E non unitaires. On note $E(X) = e_0 + e_1X + \cdots + e_nX^n$. Pour alléger la notation par la suite, on pose $z \in \mathbb{Z}^*$ avec

$$z = \begin{cases} e_n & \text{si } \forall i \in [\![2, n-1]\!],\ e_n \text{ divise } e_i \\ (e_n)^{n-1} & \text{sinon.} \end{cases}$$

Soit $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS avec E non unitaire. Soient

$$A(X) = a_0 + a_1X + \ldots a_{n-1}X^{n-1}$$

et

$$B(X) = b_0 + b_1X + \ldots b_{n-1}X^{n-1}$$

et

$$C(X) = c_0 + c_1X + \ldots c_{n-1}X^{n-1} + c_nX^n + c_{n+1}X^{n+1} + \cdots + c_{2n-2}X^{2n-2}$$

avec $c_i = \sum_{j=0}^{i} a_jb_{i-j}$ pour $i \in [\![0, n-1]\!]$ et $c_i = \sum_{j=i-n+1}^{n-1} a_jb_{n-j}$ pour $i \in [\![n, 2n-2]\!]$. On a donc $C(X) = A(X)B(X)$. Si l'on reprend l'équation (1.2), page 35, dans le cas non unitaire, la réduction externe peut être vue comme l'opération :

$$C'(X) = \pi_n(z \times (c_0, c_1, \ldots, c_{n-1}) + (c_n, c_{n+1}, \ldots, c_{2n-2}) \cdot \mathcal{E}) \tag{2.1}$$

avec C' la version réduite de $z \times C$ en termes de degré. Ainsi,

$$w = \|\,|z| \times (1, 2, \ldots, n) + (n-1, n-2, \ldots, 1) \cdot \mathcal{E}'\|_\infty \tag{2.2}$$

avec $\mathcal{E}'$ la matrice telle que $\forall i,j \in [\![0,n-2]\!] \times [\![0,n-1]\!]$, $\mathcal{E}'_{i,j} = |\mathcal{E}_{i,j}|$ avec $\mathcal{E}$ la matrice suivante :

$$\mathcal{E} = \begin{pmatrix} -e_0 & -e_1 & \dots & -e_{n-1} \\ \dots & \dots & \dots & \dots \\ \vdots & \vdots & \vdots & \vdots \\ \dots & \dots & \dots & \dots \end{pmatrix} \begin{matrix} \leftarrow \nu_n(e_n X^n \bmod E) \\ \leftarrow \nu_n(e_n X^{n+1} \bmod E) \text{ ou } \nu_n((e_n)^2 X^{n+1} \bmod E) \\ \\ \leftarrow \nu_n(e_n X^{2n-2} \bmod E) \text{ ou } \nu_n((e_n)^{n-1} X^{2n-2} \bmod E) \end{matrix} \tag{2.3}$$

On considère $|z|$ et non z dans cette nouvelle expression pour la même raison que l'on doit considérer la matrice en valeur absolue. Le principe est de majorer $\|zA(X)B(X)\|_\infty$ par rapport à $\|A(X)\|_\infty$ et $\|B(X)\|_\infty$.

Ainsi, pour la forme non-unitaire présentée dans [54] avec $E(X) = \alpha X^n - \lambda$, on aura

$$\mathcal{E} = \begin{pmatrix} \lambda & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & \lambda & 0 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & \dots & \dots & 0 & \lambda & 0 \end{pmatrix} \begin{matrix} \leftarrow \nu_n(\alpha X^n \bmod E) \\ \leftarrow \nu_n(\alpha X^{n+1} \bmod E) \\ \\ \leftarrow \nu_n(\alpha X^{2n-2} \bmod E) \end{matrix} \tag{2.4}$$

et donc $w = \|\alpha \times (1,2,\dots,n) + (n-1,n-2,\dots,1) \cdot \mathcal{E}'\| = \max(\alpha n, |\lambda|(n-1)+\alpha)$ en supposant α toujours positif.

Remarque 20. Pour $e_n = 1$, la matrice $\mathcal{E}$ est identique à celle de l'équation (1.3), page 35.

Enfin une propriété clé concerne la valeur minimum de w.

Proposition 2.3.5. *Soit $n \in \mathbb{N}^*$. $\forall E \in \mathbb{Z}[X]$ avec $\deg(E) = n$, le paramètre w correspondant à E est tel que $w \geqslant n$.*

Démonstration. Soit $E \in \mathbb{Z}[X]$ de degré n quelconque. On note $E(X) = e_0 + e_1 X + \dots + e_n X^n$. L'équation (2.2) nous donne la valeur de w comme $w = \||z| \times (1,2,\dots,n) + (n-1,n-2,\dots,1) \cdot \mathcal{E}'\|_\infty$ avec $|z| \leqslant |(e_n)^{n-1}|$ et $\mathcal{E}'$ la matrice de l'équation (2.3) en valeur absolue.

La matrice étant prise en valeur absolue, les coefficients du résultat de $(n-1,n-2,\dots,1) \cdot \mathcal{E}'$ sont donc tous positifs. Ainsi,

$$\||z| \times (1,2,\dots,n) + (n-1,n-2,\dots,1) \cdot \mathcal{E}'\|_\infty \geqslant \||z| \times (1,2,\dots,n)\|_\infty = |z|n.$$

Ainsi pour tout E, $w \geqslant |z|n$. Étant donné que $|z| \geqslant 1$ on obtient donc $w \geqslant n$. □

En particulier, cette borne est atteinte pour des polynômes E de la forme $E(X) = X^n \pm 1$.

2.3.3 Matrices de Toeplitz

Les matrices de Toeplitz sont des formes de matrices particulières permettant des opérations vecteur-matrice rapides. Nous en parlons ici car, selon le polynôme E utilisé, les opérations polynomiales modulaires peuvent être vues comme des multiplications vecteur-matrice dont la matrice est de Toeplitz. Définissons tout d'abord ce qu'est une matrice de Toeplitz.

Définition 10. Soit K un corps. Soit $M \in \mathcal{M}_n(K)$. M est une matrice de Toeplitz s'il existe $(m_0, m_1, \ldots, m_{2n-2}) \in K^{2n-1}$ tels que l'on peut écrire M comme

$$M = \begin{pmatrix} m_0 & m_1 & \ldots & m_{n-2} & m_{n-1} \\ m_n & m_0 & \ddots & m_{n-3} & m_{n-2} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ m_{2n-3} & m_{2n-4} & \ddots & m_0 & m_1 \\ m_{2n-2} & m_{2n-3} & \ldots & m_n & m_0 \end{pmatrix}$$

On note que l'on voudra parfois écrire M comme

$$M = \begin{pmatrix} m_{n-1} & m_n & \ldots & m_{2n-3} & m_{2n-2} \\ m_{n-2} & m_{n-1} & \ddots & m_{2n-4} & m_{n-2} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ m_1 & m_2 & \ddots & m_{n-1} & m_n \\ m_0 & m_1 & \ldots & m_{n-2} & m_{n-1} \end{pmatrix} \tag{2.5}$$

ce qui est équivalent.

Les propriétés intéressantes des matrices de Toeplitz sont qu'il suffit de $2n-1$ coefficients différents pour les décrire et qu'il existe des algorithmes de multiplication matrice-vecteur et vecteur-matrice sous-quadratiques. En particulier, Hasan et Negre ont présenté un algorithme de multiplication matrice-vecteur récursif sur $\mathbb{F}_2$ dans [31].

La raison de l'intérêt de cette forme de matrice particulière est que lorsque l'on considère un PMNS $\mathcal{B} = (p, n, \gamma, \rho, E)$ avec $E = X^n - \lambda$, $\lambda \in \mathbb{Z}^*$, on peut alors exprimer la multiplication de deux polynômes modulo E comme une opération vecteur-matrice de Toeplitz. En effet, soient $A, B \in \mathcal{B} \times \mathcal{B}$ et $C \in \mathbb{Z}_{n-1}[X]$ tels que $C(X) \equiv A(X) \times B(X) \bmod E(X)$, alors on peut écrire $\nu_n(C)$ comme :

$$(c_0, c_1, \ldots, c_{n-1}) = (a_0, a_1, \ldots, a_{n-1}) \times \begin{pmatrix} b_0 & b_1 & \ldots & b_{n-2} & b_{n-1} \\ \lambda b_{n-1} & b_0 & \ldots & b_{n-3} & b_{n-2} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \lambda b_2 & \lambda b_3 & \ldots & b_0 & b_1 \\ \lambda b_1 & \lambda b_2 & \ldots & \lambda b_{n-1} & b_0 \end{pmatrix} \tag{2.6}$$

Soit M une matrice de Toeplitz de dimension n. Soit κ un facteur premier de n. On peut alors écrire M sous la forme

$$\begin{pmatrix} M_0 & M_1 & M_2 & \dots & M_{\kappa-3} & M_{\kappa-2} & M_{\kappa-1} \\ M_\kappa & M_0 & M_1 & \dots & M_{\kappa-4} & M_{\kappa-3} & M_{\kappa-2} \\ M_{\kappa+1} & M_\kappa & M_0 & \dots & M_{\kappa-5} & M_{\kappa-4} & M_{\kappa-3} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ M_{2\kappa-2} & M_{2\kappa-3} & M_{2\kappa-4} & \dots & M_{\kappa+1} & M_\kappa & M_0 \end{pmatrix} \tag{2.7}$$

avec chaque M_i une sous-matrice de dimension $\frac{n}{\kappa} \times \frac{n}{\kappa}$. On note que chaque sous-matrice est aussi de Toeplitz. Alternativement on peut les renuméroter pour obtenir

$$\begin{pmatrix} M_{\kappa-1} & M_\kappa & M_{\kappa+1} & \dots & M_{2\kappa-4} & M_{2\kappa-3} & M_{2\kappa-2} \\ M_{\kappa-2} & M_{\kappa-1} & M_\kappa & \dots & M_{2\kappa-5} & M_{2\kappa-4} & M_{2\kappa-3} \\ M_{\kappa-3} & M_{\kappa-2} & M_{\kappa-1} & \dots & M_{2\kappa-6} & M_{2\kappa-5} & M_{2\kappa-4} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ M_0 & M_1 & M_2 & \dots & M_{\kappa-3} & M_{\kappa-2} & M_{\kappa-1} \end{pmatrix} \tag{2.8}$$

Algorithme 17 TVMP : Algorithme de multiplication vecteur-matrice de Toeplitz récursif

Require: n la dimension, V un vecteur de dimension n avec $(v_0, v_1, \ldots, v_{n-1})$ ses coefficients et M une matrice de Toeplitz de dimension $n \times n$ avec $(m_0, m_1, \ldots m_{2n-2})$ ses coefficients (cf. équation (2.5)).
Ensure: $T = (t_0, t_1, \ldots, t_{n-1})$ tel que $T = V \cdot M$
  if $n = 1$ **then**
    return $(v_0 \times m_0)$ # vecteur de dimension 1
  end if
  $\kappa \leftarrow$ plus petit facteur premier de n
  $V_0, V_1, \ldots, V_{\kappa-1} \leftarrow$ décomposition de V en sous vecteurs de dimensions $\frac{n}{\kappa}$
  $M_0, M_1, \ldots, M_{2\kappa-2} \leftarrow$ décomposition de M en sous matrices de dimensions $\frac{n}{\kappa} \times \frac{n}{\kappa}$ (cf. équation (2.8))
  for $i = 0..\kappa - 1$ **do**
    $P_i \leftarrow$ **TVMP**$(\frac{n}{\kappa}, V_{\kappa-1-i}, \sum_{j=0}^{\kappa-1} M_{i+j})$ # Appel récursif
  end for
  $z \leftarrow 0$
  for $i = 0..\kappa - 2$ **do**
    for $j = 0..\kappa - 2 - i$ **do**
      $P_{\kappa+z} \leftarrow$ **TVMP**$(\frac{n}{\kappa}, V_j - V_{\kappa-1-i}, M_{\kappa-1+i-j})$ # Appel récursif
      $z \leftarrow z + 1$
    end for
  end for
  $z \leftarrow 0$
  for $i = 0..\kappa - 1$ **do**
    $z \leftarrow z + \kappa - i$
    for $j = 0..\frac{n}{\kappa} - 1$ **do**
      $t_{i\times\frac{n}{\kappa}+j} \leftarrow P_{i,j} + \sum_{y=z}^{z+\kappa-i-2} P_{y,j} - \sum_{x=0}^{i-1} P_{2\kappa-1-i+\kappa x-\frac{x(x+1)}{2},j}$
    end for
  end for
  return $(t_0, t_1, \ldots, t_{n-1})$

Proposition 2.3.6. *L'algorithme 17 calcule bien T tel que $T = V \cdot M$.*

Démonstration. Pour $n = 1$, on a bien $T = (v_0 \times m_0)$.

Pour $n \neq 1$, pour commencer on découpe le vecteur V en κ sous-vecteurs V_i de taille $\frac{n}{\kappa}$ avec $V_i = (v_{i\times\frac{n}{\kappa}}, v_{i\times\frac{n}{\kappa}+1}, \ldots, v_{i\times\frac{n}{\kappa}+\frac{n}{\kappa}-1})$ pour i allant de 0 à $\kappa - 1$. De même, la matrice M est découpée comme présenté dans l'équation (2.8) en $2\kappa - 1$ sous-matrices de dimension $\frac{n}{\kappa}$. Ensuite, pour i allant de 0 à $\kappa - 1$ on a :

$$\begin{cases} P_0 & = V_{\kappa-1} \cdot (M_0 + M_1 + \cdots + M_{\kappa-2} + M_{\kappa-1}) \\ P_1 & = V_{\kappa-2} \cdot (M_1 + M_2 + \cdots + M_{\kappa-1} + M_\kappa) \\ P_2 & = V_{\kappa-3} \cdot (M_2 + M_3 + \cdots + M_\kappa + M_{\kappa+1}) \\ \ldots & \\ P_{\kappa-1} & = V_0 \cdot (M_{\kappa-1} + M_\kappa + \cdots + M_{2\kappa-3} + M_{2\kappa-2}) \end{cases}$$

Ces κ premiers produits correspondent chacun à la multiplication d'un sous-vecteur par une colonne de la

matrice (cf. équation (2.8)). C'est-à-dire, par exemple, pour P_0 on a

$$P_0 = (V_{\kappa-1}, \dots, V_{\kappa-1}) \times \begin{pmatrix} M_{\kappa-1} \\ \vdots \\ M_0 \end{pmatrix}$$

Si $T = V \cdot M$ alors $T_0 = V_0 \cdot M_{\kappa-1} + V_1 \cdot M_{\kappa-2} + \cdots + V_{\kappa-2} \cdot M_1 + V_{\kappa-1} \cdot M_0$. Ainsi, ayant calculé $P_0 = V_{\kappa-1} \cdot M_{\kappa-1} + V_{\kappa-1} \cdot M_{\kappa-2} + \cdots + V_{\kappa-1} \cdot M_1 + V_{\kappa-1} \cdot M_0$, on a calculé T_0 de façon approximative avec des erreurs. On calcule donc les termes correctifs $P_\kappa, P_{\kappa+1}, \dots P_{2\kappa-3}, P_{2\kappa-2}$ qui sont tels que

$$\begin{cases} P_\kappa & = (V_0 - V_{\kappa-1}) \cdot M_{\kappa-1} \\ P_{\kappa+1} & = (V_1 - V_{\kappa-1}) \cdot M_{\kappa-2} \\ \dots & \\ P_{2\kappa-3} & = (V_{\kappa-3} - V_{\kappa-1}) \cdot M_2 \\ P_{2\kappa-2} & = (V_{\kappa-2} - V_{\kappa-1}) \cdot M_1 \end{cases}$$

On a alors

$$\begin{aligned} T_0 &= P_0 + P_\kappa + P_{\kappa+1} + \cdots + P_{2\kappa-3} + P_{2\kappa-2} \\ &= V_{\kappa-1} \cdot (M_0 + M_1 + \cdots + M_{\kappa-2} + M_{\kappa-1}) \\ &\quad + V_0 \cdot M_{\kappa-1} - V_{\kappa-1} \cdot M_{\kappa-1} \\ &\quad + V_1 \cdot M_{\kappa-2} - V_{\kappa-1} \cdot M_{\kappa-2} \\ &\quad + \dots \\ &\quad + V_{\kappa-3} \cdot M_2 - V_{\kappa-1} \cdot M_2 \\ &\quad + V_{\kappa-2} \cdot M_1 - V_{\kappa-1} \cdot M_1 \\ &= V_0 \cdot M_{\kappa-1} + V_1 \cdot M_{\kappa-2} + \cdots + V_{\kappa-2} \cdot M_1 + V_{\kappa-1} \cdot M_0 \end{aligned}$$

ce qui est correct. On procède de façon similaire pour T_1 où l'on pourra appliquer des termes correctifs à P_1. On veut calculer $V_0 \cdot M_\kappa + V_1 \cdot M_{\kappa-1} + \cdots + V_{\kappa-2} \cdot M_2 + V_{\kappa-1} \cdot M_1$ et l'on a $P_1 = V_{\kappa-2} \cdot M_\kappa + V_{\kappa-2} \cdot M_{\kappa-1} + \cdots + V_{\kappa-2} \cdot M_2 + V_{\kappa-2} \cdot M_1$. Donc les termes correctifs nécessaires sont

$$\begin{cases} P_{2\kappa-1} & = (V_0 - V_{\kappa-2}) \cdot M_\kappa \\ P_{2\kappa} & = (V_1 - V_{\kappa-2}) \cdot M_{\kappa-1} \\ \dots & \\ P_{3\kappa-5} & = (V_{\kappa-4} - V_{\kappa-2}) \cdot M_3 \\ P_{3\kappa-4} & = (V_{\kappa-3} - V_{\kappa-2}) \cdot M_2 \end{cases}$$

En particulier, au lieu de calculer $(V_{\kappa-1} - V_{\kappa-2}) \cdot M_1$, on remarque qu'on a déjà $P_{2\kappa-2} = (V_{\kappa-2} - V_{\kappa-1}) \cdot M_1$ et donc on peut le réutiliser.

On procède de façon similaire pour la suite. Pour chaque T_i, le nombre de termes correctifs à appliquer à P_i est de $\kappa - 1$ et l'on réutilise les termes déjà calculés au fur et à mesure. Il y a exactement i termes réutilisables pour chaque T_i et donc le nombre de termes correctifs à calculer est bien de $\frac{(\kappa-1)(\kappa)}{2}$. □

Proposition 2.3.7. *Soit $n = 1 \times \kappa_1 \times \kappa_2 \times \cdots \times \kappa_d$ avec chaque κ_i un facteur premier de n tel que $\forall i, j$, $j > i \implies \kappa_j \geqslant \kappa_i$, alors le nombre de multiplications effectuées dans l'algorithme 17 est de $1 \times \prod_{i=1}^{d} \left(\frac{\kappa_i(\kappa_i+1)}{2}\right)$.*

Démonstration. On pose $\mathrm{Mul}(n)$ la fonction qui pour un n donné renvoie le nombre de multiplications effectuées dans l'algorithme 17. On note que pour $n = 1$ on effectue $v_0 \times m_0$ donc une seule multiplication. On a donc $\mathrm{Mul}(1) = 1$.

Ensuite, on suppose

$$\mathrm{Mul}(n) = \prod_{i=1}^{f} \left(\frac{\kappa_i(\kappa_i+1)}{2}\right) \mathrm{Mul}\left(\frac{n}{\prod_{i=1}^{f} \kappa_i}\right)$$

pour un certain $f < d$. Avec les notations introduites, le plus petit facteur premier de $\frac{n}{\prod_{i=1}^{f} \kappa_i}$ est κ_{f+1}. La première boucle de l'algorithme 17 effectue κ_{f+1} appels récursifs pour calculer les produits vecteur-matrices approximatifs et la deuxième boucle effectue $\frac{\kappa_{f+1}(\kappa_{f+1}-1)}{2}$ appels récursifs pour calculer les termes correctifs. Ainsi, au total, l'algorithme 17 effectue $\frac{\kappa_{f+1}(\kappa_{f+1}+1)}{2}$ appels récursifs à lui-même sur des paramètres de dimension $\frac{n}{\prod_{i=1}^{f+1} \kappa_i}$. On a donc

$$\mathrm{Mul}\left(\frac{n}{\prod_{i=1}^{f} \kappa_i}\right) = \frac{\kappa_{f+1}(\kappa_{f+1}+1)}{2} \mathrm{Mul}\left(\frac{n}{\prod_{i=1}^{f+1} \kappa_i}\right).$$

D'où

$$\mathrm{Mul}(n) = \prod_{i=1}^{f+1} \left(\frac{\kappa_i(\kappa_i+1)}{2}\right) \mathrm{Mul}\left(\frac{n}{\prod_{i=1}^{f+1} \kappa_i}\right).$$

Par récurrence on obtient

$$\mathrm{Mul}(n) = \prod_{i=1}^{d} \left(\frac{\kappa_i(\kappa_i+1)}{2}\right) \times \mathrm{Mul}\left(\frac{n}{\prod_{i=1}^{d} \kappa_i}\right) = \prod_{i=1}^{d} \left(\frac{\kappa_i(\kappa_i+1)}{2}\right) \times \mathrm{Mul}(1).$$

D'où

$$\mathrm{Mul}(n) = 1 \times \prod_{i=1}^{d} \left(\frac{\kappa_i(\kappa_i+1)}{2}\right)$$

□

Remarque 21. Pour $n = 2^k$ pour $k \in \mathbb{N}^*$, on a $\mathrm{Mul}(n) = 3^k$. On a alors $\log_n(\mathrm{Mul}(n)) = \frac{\log(3^k)}{\log(2^k)} = \frac{\log(3)}{\log(2)} = \log_2(3)$. Le nombre de multiplications de l'algorithme 17 est alors égal à $n^{\log_2(3)} \approx n^{1.58}$. En comparaison, une multiplication vecteur-matrice classique pour vecteur et matrice quelconques nécessite n^2 multiplications élémentaires.

Remarque 22. On note que le nombre de multiplications dans la proposition 2.3.7 correspond au nombre de multiplications sur deux opérandes d'au plus $\log_2(\sigma)$ bits chacune où le résultat sera sur au plus $2\log_2(\sigma)$

bits. Si une telle opération n'est pas nativement possible sur le processeur utilisé, un découpage bas-bas, bas-haut, haut-bas, haut-haut devra être effectué à la place pour obtenir le résultat sur deux registres. Le nombre de multiplications se voit ainsi multiplié par 4.

Proposition 2.3.8. *Soit $n = 1 \times \kappa_1 \times \kappa_2 \times \cdots \times \kappa_d$ avec chaque κ_i un facteur premier de n tel que $\forall i, j$, $j > i \implies \kappa_j \geqslant \kappa_i$. Si V et M les entrées de l'algorithme 17 sont telles que $\|V\|_\infty < \sigma$ et $\|M\|_1 < \sigma$ alors l'algorithme 17 peut être effectué en*

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

additions avec $\kappa_0 = 1$.

Démonstration. On pose $\mathrm{Add}(n)$ la fonction qui pour un n donné renvoie le nombre d'additions effectuées dans l'algorithme 17. On note que $\mathrm{Add}(1) = 0$.

Pour $n \neq 1$, le plus petit facteur premier de n est κ_1. Pour i allant de 0 à $\kappa_1 - 1$ le calcul des P_i est effectué de la façon suivante :

$$\begin{cases} P_0 & = V_{\kappa_1-1} \cdot (M_0 + M_1 + \cdots + M_{\kappa_1-2} + M_{\kappa_1-1}) \\ P_1 & = V_{\kappa_1-2} \cdot (M_1 + M_2 + \cdots + M_{\kappa_1-1} + M_{\kappa_1}) \\ P_2 & = V_{\kappa_1-3} \cdot (M_2 + M_3 + \cdots + M_{\kappa_1} + M_{\kappa_1+1}) \\ \cdots & \\ P_{\kappa_1-1} & = V_0 \cdot (M_{\kappa_1-1} + M_{\kappa_1} + \cdots + M_{2\kappa_1-3} + M_{2\kappa_1-2}) \end{cases}$$

ce qui demande κ_1 additions de sous-matrices pour chaque P_i. Chaque sous-matrice possède $2\frac{n}{\kappa_1} - 1$ coefficients donc on pourrait naïvement effectuer $\kappa_1(\kappa_1 - 1) \times (2\frac{n}{\kappa_1} - 1)$ additions. Or, certaines de ces additions sont communes aux différentes sous-sommes. Nous pouvons donc les factoriser.

On calcule tout d'abord $M_s = \sum_{i=1}^{\kappa_1-1} M_i$. On peut alors poser $P_0 = V_{\kappa_1-1} \cdot (M_0 + M_s)$ et $P_1 = V_{\kappa_1-2} \cdot (M_s + M_{\kappa_1})$, ce qui nous permet de n'effectuer que κ_1 additions de sous-matrices pour les deux premiers sous-produits. Ensuite, on aura $P_2 = V_{\kappa_1-3} \cdot (M_s - M_1 + M_{\kappa_1} + M_{\kappa_1+1})$. Or, nous avons déjà calculé $(M_s + M_{\kappa_1})$ pour P_1, donc le résultat peut être réutilisé pour n'avoir à effectuer que 2 additions de sous-matrices supplémentaires pour P_2. On peut procéder de la même façon pour les $\kappa_1 - 3$ sous-produits restants. Cela nous ramène à $\kappa_1 + 2(\kappa_1 - 2)$ additions de sous-matrices de dimension $(2\frac{n}{\kappa_1} - 1)$ pour un total de $(2\frac{n}{\kappa_1} - 1)(3\kappa_1 - 4)$ additions, ce qui s'avère meilleur que l'approche naïve qui demande $\kappa_1(\kappa_1 - 1) \times (2\frac{n}{\kappa_1} - 1)$ additions.

Ensuite, pour i allant de κ_1 à $\frac{\kappa_1(\kappa_1+1)}{2}-1$, on a

$$\begin{cases} P_{\kappa_1} & = (V_0 - V_{\kappa_1-1}) \cdot M_{\kappa_1-1} \\ P_{\kappa_1+1} & = (V_1 - V_{\kappa_1-1}) \cdot M_{\kappa_1-2} \\ P_{\kappa_1+2} & = (V_2 - V_{\kappa_1-1}) \cdot M_{\kappa_1-3} \\ \cdots & \\ P_{\frac{\kappa_1(\kappa_1+1)}{2}-1} & = (V_0 - V_1) \cdot M_{2\kappa_1-3} \end{cases}$$

ce qui nécessite $\frac{n}{\kappa_1} \times \frac{\kappa_1(\kappa_1-1)}{2} = n \times \frac{(\kappa_1-1)}{2}$ additions.

Pour en finir avec κ_1, on groupe d'abord les t_j pour j allant de 0 à $n-1$ en $T_i = (t_{i\times\frac{n}{\kappa_1}}, t_{i\times\frac{n}{\kappa_1}+1}, \ldots, t_{i\times\frac{n}{\kappa_1}+\frac{n}{\kappa_1}-1})$ pour i allant de 0 à $\kappa_1 - 1$. Les T_i sont calculés comme suit :

$$\begin{cases} T_0 & = P_0 + P_{\kappa_1} + P_{\kappa_1+1} \cdots + P_{2\kappa_1-3} + P_{2\kappa_1-2} \\ T_1 & = P_1 + P_{2\kappa_1-1} + P_{2\kappa_1} \cdots + P_{3\kappa_1-5} + P_{3\kappa_1-4} - P_{2\kappa_1-2} \\ T_2 & = P_2 + P_{3\kappa_1-3} + P_{3\kappa_1-2} \cdots + P_{4\kappa_1-8} + P_{4\kappa_1-7} - P_{2\kappa_1-3} - P_{3\kappa_1-4} \\ \cdots & \\ T_{\kappa_1-1} & = P_{\kappa_1-1} - P_{\kappa_1} - P_{2\kappa_1-1} - \cdots - P_{\frac{\kappa_1(\kappa_1+1)}{2}-3} - P_{\frac{\kappa_1(\kappa_1+1)}{2}-1} \end{cases}$$

Il s'agit donc de κ_1 sommes de κ_1 sous-produits chacune. Puisque l'on a supposé $\|V\|_\infty < \sigma$ et $\|M\|_1 < \sigma$, on a $\|P_i\|_\infty < \sigma^2$. Ainsi, chaque sous-produit est un vecteur de $\frac{n}{\kappa_1}$ coefficients dont chaque coefficient tient sur 2 registres, donc on effectue $\frac{n}{\kappa_1}(2 \times \kappa_1(\kappa_1-1)) = n(2 \times (\kappa_1 - 1))$ additions. Pour résumer, avant tout appel récursif, on a un total de $(2\frac{n}{\kappa_1} - 1)(3\kappa_1 - 4) + n(\frac{\kappa_1-1}{2} + 2 \times (\kappa_1 - 1))$ additions. On effectue $\frac{\kappa_1(\kappa_1+1)}{2}$ appels récursifs sur une dimension de $\frac{n}{\kappa_1}$ donc on a

$$\mathrm{Add}(n) = (2\frac{n}{\kappa_1} - 1)(3\kappa_1 - 4) + n(\frac{\kappa_1 - 1}{2} + 2 \times (\kappa_1 - 1)) + \frac{\kappa_1(\kappa_1+1)}{2}\mathrm{Add}(\frac{n}{\kappa_1}).$$

On suppose

$$\begin{aligned} \mathrm{Add}(n) = \sum_{i=1}^{f} \left(\left(\prod_{j=0}^{i-1} \frac{\kappa_j(\kappa_j+1)}{2} \right) \times \left(\left(\frac{2n}{\prod_{j=1}^{i} \kappa_j} - 1 \right) (3\kappa_i - 4) + \frac{n}{\prod_{j=0}^{i-1} \kappa_j} \left(\frac{\kappa_i - 1}{2} + 2 \times (\kappa_i - 1) \right) \right) \right) \\ + \frac{\kappa_f(\kappa_f+1)}{2}\mathrm{Add}(\frac{n}{\prod_{i=1}^{f} \kappa_i}) \end{aligned}$$

pour un $f < d$ donné. Alors, de façon similaire à ce que nous venons de faire, pour κ_{f+1} le plus petit facteur premier de $\frac{n}{\prod_{i=1}^{f} \kappa_i}$ on a

$$\mathrm{Add}(\frac{n}{\prod_{i=1}^{f} \kappa_i}) = (2\frac{n}{\prod_{i=1}^{f+1} \kappa_i} - 1)(3\kappa_{f+1} - 4) + n(\frac{\kappa_{f+1} - 1}{2} + 2 \times (\kappa_{f+1} - 1)) + \frac{\kappa_{f+1}(\kappa_{f+1}+1)}{2}\mathrm{Add}(\frac{n}{\kappa_{f+1}})$$

et donc par récurrence le nombre total d'additions sera bien de

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

en notant que $\frac{\kappa_d(\kappa_d+1)}{2}\text{Add}(\frac{n}{\prod_{i=1}^{d}\kappa_i})=\frac{\kappa_d(\kappa_d+1)}{2}\times 0$ car $\text{Add}(1)=0$. □

Remarque 23. Soit $\mathcal{B}=(p,n,\gamma,\rho,E)$ un PMNS avec $E=X^n\pm X-1$, on peut décomposer la multiplication de deux polynômes modulo E en opérations vecteur-matrice circulante ou de Toeplitz. En effet, soient $A,B\in\mathcal{B}\times\mathcal{B}$ alors on peut voir la matrice de multiplication par B modulo E

$$\begin{pmatrix} b_0 & b_1 & \dots & b_{n-2} & b_{n-1}\\ b_{n-1} & b_0\pm b_{n-1} & \dots & b_{n-3} & b_{n-2}\\ \vdots & \ddots & \ddots & \ddots & \vdots\\ b_2 & b_3\pm b_2 & \dots & b_0\pm b_{n-1} & b_1\\ b_1 & b_2\pm b_1 & \dots & b_{n-1}\pm b_{n-2} & b_0\pm b_{n-1}\end{pmatrix}$$

comme

$$\begin{pmatrix} b_0 & b_1 & \dots & b_{n-2} & b_{n-1}\\ b_{n-1} & b_0 & \dots & b_{n-3} & b_{n-2}\\ \vdots & \ddots & \ddots & \ddots & \vdots\\ b_2 & b_3 & \dots & b_0 & b_1\\ b_1 & b_2 & \dots & b_{n-1} & b_0\end{pmatrix}+\begin{pmatrix}0 & 0 & \dots & 0 & 0\\ 0 & \pm b_{n-1} & \dots & 0 & 0\\ \vdots & \ddots & \ddots & \ddots & \vdots\\ 0 & \pm b_2 & \dots & \pm b_{n-1} & 0\\ 0 & \pm b_1 & \dots & \pm b_{n-2} & \pm b_{n-1}\end{pmatrix}$$

ou encore

$$\begin{pmatrix} b_0\pm b_{n-1} & b_1 & \dots & b_{n-2} & b_{n-1}\\ b_{n-1}\pm b_{n-2} & b_0\pm b_{n-1} & \dots & b_{n-3} & b_{n-2}\\ \vdots & \ddots & \ddots & \ddots & \vdots\\ b_2\pm b_1 & b_3\pm b_2 & \dots & b_0\pm b_{n-1} & b_1\\ b_1 & b_2\pm b_1 & \dots & b_{n-1}\pm b_{n-2} & b_0\pm b_{n-1}\end{pmatrix}+\begin{pmatrix}\pm b_{n-1} & 0 & \dots & 0 & 0\\ \pm b_{n-2} & 0 & \dots & 0 & 0\\ \vdots & \ddots & \ddots & \ddots & \vdots\\ \pm b_1 & 0 & \dots & 0 & 0\\ 0 & 0 & \dots & 0 & 0\end{pmatrix}$$

mais en pratique nous obtenons de meilleurs résultats en considérant simplement les polynômes E de la forme $X^n-\lambda$ et en effectuant l'algorithme 17.

2.4 Réduction interne

Toutes les méthodes de réduction présentées dans cette section utilisent une base courte de $\mathfrak{L}=\left\{(x_0,\dots,x_{n-1})\in\mathbb{Z}^n:\sum_{i=0}^{n-1}x_i\gamma^i\equiv 0\pmod p\right\}$ ou d'un sous-réseau de $\mathfrak{L}$.

Les premières méthodes de réduction interne dans la littérature faisaient usage d'un polynôme M qui s'annule en γ modulo p (cf. algorithme 11, page 37). Plus tard, la réduction interne a été vue comme une version matricielle des mêmes opérations (cf. algorithme 12, page 38). Ceci vient du fait que $\nu_n(M) \in \mathfrak{L}$ et donc on peut simplement construire la matrice composée des n vecteurs de la forme $\nu_n(M \times X^i \bmod E(X))$ pour i allant de 0 à $n-1$ pour M inversible modulo E (autrement dit la matrice de l'équation (1.4), page 37), ce qui est bien une base courte d'un sous-réseau de $\mathfrak{L}$. Ainsi, nous prenons dans cette section une vue matricielle des opérations et non polynomiale, mais les opérations restent équivalentes. Le fait de considérer les opérations sous forme matricielle est avantageux lorsque l'on considère les formes de matrices intéressantes, par exemple des matrices de Toeplitz pour $E(X) = \alpha X^n - \lambda$.

Chaque méthode de réduction interne présentée ici dépend d'un paramètre ϕ qui est utilisé différemment selon les méthodes de réduction.

2.4.1 Réduction interne Montgomery-like

Dans [44] avec Méloni et Véron nous avons généralisé la réduction interne Montgomery-like présentée en section 1.5.4, algorithme 11, page 37, pour faire usage de la base courte du réseau $\mathfrak{L}$ directement. En effet, la méthode pour générer un polynôme M pour la réduction Montgomery-like consistait à effectuer une combinaison linéaire binaire des lignes de la base courte pour obtenir un polynôme inversible modulo E et ϕ. Ainsi, la recherche d'un tel polynôme M demande 2^n itérations dans le pire des cas. Ceci peut s'avérer problématique pour n assez grand, d'où l'intérêt de considérer des méthodes alternatives.

Comme noté dans la remarque 12 (page 37), la matrice de réduction interne construite à partir d'un tel polynôme M est une base d'un sous-réseau de $\mathfrak{L}$. Ainsi, une question naturelle est de considérer une réduction utilisant une base d'un sous-réseau de $\mathfrak{L}$ quelconque, ce qui permettrait de s'affranchir de la génération du polynôme M.

En particulier la remarque 5 (page 30) indique que $\det(\mathfrak{L}) = p$. Ainsi, une base courte de $\mathfrak{L}$ sera toujours inversible modulo $\phi = 2^h$, $h \in \mathbb{N}^*$ pour $p > 2$. Ainsi, si l'on pose $\mathcal{G}$ une base courte de $\mathfrak{L}$, on pourrait alors substituer $\mathcal{M}$ par $\mathcal{G}$ et $\mathcal{M}^{-1} \bmod \phi$ par $\mathcal{G}^{-1} \bmod \phi$ dans l'algorithme 12, page 38.

Enfin, [18] a démontré que cette méthode est possible avec n'importe quelle base d'un sous-réseau de $\mathfrak{L}$.

Algorithme 18 GMont-like : Réduction interne Montgomery-like sur les réseaux [44]

Require: $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS, $C \in \mathbb{Z}[X]$, $\phi \in \mathbb{N}^*$ et $\mathcal{G}$ une base courte d'un sous-réseau de $\mathfrak{L}$ inversible modulo ϕ.
Ensure: $C'(\gamma) \equiv C(\gamma)\phi^{-1} \bmod p$
1: $q \leftarrow -\nu_n(C) \cdot \mathcal{G}^{-1} \bmod \phi$
2: $C' \leftarrow (C + \pi_n(q \cdot \mathcal{G}))/\phi$
3: **return** C'

Remarque 24. On note que cet algorithme, tout comme l'algorithme 11, 37, implique l'usage du domaine de Montgomery. Soit $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS. Soient $a, b \in \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/p\mathbb{Z}$ et $A, B \in \mathcal{B} \times \mathcal{B}$ tels que $A(\gamma) \equiv a\phi \pmod p$ et $B(\gamma) \equiv b\phi \pmod p$, alors $C(X) \equiv A(X)B(X) \pmod{E(X)}$ sera tel que $C(\gamma) \equiv ab\phi^2 \pmod p$. Donc, avec C' la sortie de l'algorithme 18 appliquée à C, on aura $C'(\gamma) \equiv ab\phi \pmod p$, ce qui maintient le domaine de Montgomery. Pour en sortir, il suffira d'exécuter une réduction interne supplémentaire. On note que ce domaine est compatible avec les domaines présentés dans les remarques 16 et 19.

Proposition 2.4.1. *L'algorithme 18 calcule bien $C'(X)$ tel que $C'(\gamma) \equiv C(\gamma)\phi^{-1} \pmod p$.*

Démonstration. On remarque que $\pi_n(q\cdot\mathcal{G})(X) \equiv C(X) \pmod\phi$. Ainsi, tous les coefficients de $(C - \pi_n(q\cdot\mathcal{G}))$ sont des multiples de ϕ. La division par ϕ ne perd donc aucune information. De plus $\pi_n(q\cdot\mathcal{G})(\gamma) \equiv 0 \pmod p$ et on a donc bien que $C'(\gamma) \equiv C(\gamma)\phi^{-1} \pmod p$. □

Proposition 2.4.2. *Si $\rho \geqslant \|\mathcal{G}\|_1 - 1$ et $\phi > 2w(\rho-1)$ et $\|C\|_\infty \leqslant w(\rho-1)^2$ alors l'algorithme 18 retourne un polynôme C' tel que $\|C'\|_\infty < \rho$.*

Démonstration. On calcule q comme $\nu_n(C)\cdot\mathcal{G}^{-1} \pmod\phi$. Or, on peut prendre chaque coefficient entre $-\frac{\phi}{2}$ et $\frac{\phi}{2} - 1$ pour effectuer la réduction modulo ϕ. En pratique, cela peut se traduire simplement en coercition de type pour $\phi = \sigma$ (on rappelle que σ dénote le nombre de valeurs possibles dans un registre mémoire, 2^{64} par exemple), ce qui est virtuellement gratuit. On peut donc calculer q de telle sorte à avoir $\|q\|_\infty \leqslant \frac{1}{2}\phi$. Donc, ayant supposé $\|C\|_\infty \leqslant w(\rho-1)^2$,

$$\|C'\|_\infty \leqslant \frac{1}{\phi}(w(\rho-1)^2 + \frac{\phi}{2}\|\mathcal{G}\|_1) = \frac{1}{\phi}w(\rho-1)^2 + \frac{1}{2}\|\mathcal{G}\|_1.$$

On a pour hypothèse $\phi > 2w(\rho-1)$ donc

$$\frac{1}{\phi}w(\rho-1)^2 + \frac{1}{2}\|\mathcal{G}\|_1 < \frac{1}{2w(\rho-1)}w(\rho-1)^2 + \frac{1}{2}\|\mathcal{G}\|_1$$

$$\implies \|C'\|_\infty < \frac{1}{2}(\rho-1) + \frac{1}{2}\|\mathcal{G}\|_1$$

Nous avons aussi supposé $\rho \geqslant \|\mathcal{G}\|_1 - 1$. Ainsi

$$\frac{1}{2}(\rho-1) + \frac{1}{2}\|\mathcal{G}\|_1 \leqslant \frac{1}{2}(\rho-1) + \frac{1}{2}(\rho+1)$$

$$\implies \|C'\|_\infty < \rho$$

Ainsi, pour $\rho \geqslant \|\mathcal{G}\|_1 - 1$ et $\phi > 2w(\rho-1)$, la sortie de l'algorithme 18 vérifie $\|C'\|_\infty < \rho$ si C est tel que $\|C\|_\infty \leqslant (\rho-1)^2$. □

Cette méthode de réduction utilisant directement la base courte de $\mathfrak{L}$ n'est pas utilisée en pratique. En effet, la base courte n'a pas de forme particulière, ce qui donne une complexité quadratique aux opérations de multiplication vecteur-matrice. Cela dit cet algorithme permet l'utilisation d'une base d'un sous-réseau de $\mathfrak{L}$ et donc nous pouvons plutôt utiliser la matrice $\mathcal{M}$ (équation (1.4), page 37), permettant l'utilisation de l'algorithme 17 qui est sous-quadratique. On note que les bornes plus fines détaillées dans la proposition 2.4.2 sont valables lorsque la matrice $\mathcal{M}$ est utilisée.

2.4.2 Réduction interne Barrett-like

Proposée en [2] (adaptée de [5] que nous avons présentée dans la section 1.2.4), on présente ici une version qui utilise la base courte du réseau directement. On note que la version originale prend des parties entières des coefficients et non des valeurs arrondies comme ici, mais un arrondi ne coûte qu'une addition de plus et l'on obtient de meilleures bornes en conséquence.

Algorithme 19 Réduction interne Barrett-like sur les réseaux

Require: $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS, $C \in \mathbb{Z}[X]$, $\phi \in \mathbb{N}^*$ et $\mathcal{G}$ une base courte d'un sous-réseau de $\mathfrak{L}$. $\mathcal{G}'$ pré-calculée telle que $\mathcal{G}' = \lfloor \rho\phi\mathcal{G}^{-1} \rceil$.

Ensure: $C'(\gamma) \equiv C(\gamma) \bmod p$

1: $Q \leftarrow \left\lfloor \frac{1}{\phi} \left\lfloor \frac{1}{\rho}\nu_n(C(X)) \right\rceil \cdot \mathcal{G}' \right\rceil$

2: $C' \leftarrow (C - \pi_n(Q \cdot \mathcal{G}))$

3: **return** C'

Remarque 25. On remarque que cette méthode est très similaire à celle présentée dans [42, Algorithmes 5 et 7].

Puisque l'on soustrait une combinaison linéaire des lignes de $\mathcal{G}$, qui est une base courte d'un sous-réseau de $\mathfrak{L}$, qui est le réseau des vecteurs qui correspondent par l'isomorphisme π_n aux polynômes qui s'annulent en γ modulo p, il est évident que $C(\gamma) \equiv C'(\gamma) \pmod p$. Démontrons maintenant que ses coefficients sont bien bornés par ρ.

Proposition 2.4.3. *Si $\phi > n \left\lfloor \frac{w(\rho-1)^2}{\rho} \right\rceil$ et $\rho \geqslant 2\|\mathcal{G}\|_1$ et $\|C\|_\infty \leqslant w(\rho-1)^2$, alors l'algorithme 19 retourne un polynôme C' tel que $\|C'\|_\infty < \rho$.*

Démonstration. On a supposé $\|C\|_\infty \leqslant w(\rho-1)^2$. Posons $C_1 = \left\lfloor \frac{1}{\rho}\nu_n(C(X)) \right\rceil$ et $C_0(X) = C(X) - \pi_n(\rho C_1)(X)$. Nous avons alors $\|C_1\|_\infty \leqslant \left\lfloor \frac{w(\rho-1)^2}{\rho} \right\rceil$. De plus, étant donné que $\exists \varepsilon \in \mathbb{R}^n$ tel que $C_1 = \frac{1}{\rho}\nu_n(C(X)) - \varepsilon$ avec $\|\varepsilon\|_\infty \leqslant \frac{1}{2}$, on en déduit que $C_0 = \rho\varepsilon$ et donc $\|C_0\|_\infty \leqslant \frac{1}{2}\rho$.

On a $Q = \left\lfloor \frac{1}{\phi} C_1 \cdot \mathcal{G}' \right\rceil$ et $\mathcal{G}' = \lfloor \rho\phi\mathcal{G}^{-1} \rceil$. Posons $\varepsilon \in \mathcal{M}_n(\mathbb{R})$ tel que $\mathcal{G}' = \rho\phi\mathcal{G}^{-1} - \varepsilon$ avec $\max_{i,j} |\varepsilon_{i,j}| \leqslant \frac{1}{2}$. On en déduit que $\|\varepsilon\|_1 \leqslant \frac{n}{2}$. On a alors

$$Q = \left\lfloor \frac{1}{\phi} C_1 \cdot (\rho\phi\mathcal{G}^{-1} - \varepsilon) \right\rceil = \left\lfloor \rho C_1 \cdot \mathcal{G}^{-1} - \frac{1}{\phi} C_1 \cdot \varepsilon \right\rceil.$$

Notons $\varepsilon' \in \mathbb{R}^n$ tel que

$$Q = \rho C_1 \cdot \mathcal{G}^{-1} - \frac{1}{\phi} C_1 \cdot \varepsilon - \varepsilon'$$

avec $\|\varepsilon'\|_\infty \leqslant \frac{1}{2}$. Alors

$$Q \cdot \mathcal{G} = (\rho C_1 \cdot \mathcal{G}^{-1} - \frac{1}{\phi} C_1 \cdot \varepsilon - \varepsilon') \cdot \mathcal{G} = \rho C_1 - (\frac{1}{\phi} C_1 \cdot \varepsilon) \cdot \mathcal{G} - \varepsilon' \cdot \mathcal{G}.$$

Remarquons que

$$\|(\frac{1}{\phi} C_1 \cdot \varepsilon) \cdot \mathcal{G}\|_\infty \leqslant \frac{1}{\phi}\|C_1\|_\infty \|\varepsilon\|_1 \|\mathcal{G}\|_1 \leqslant \frac{\left\lfloor \frac{w(\rho-1)^2}{\rho} \right\rceil \frac{n}{2}\|\mathcal{G}\|_1}{\phi} = \frac{n \left\lfloor \frac{w(\rho-1)^2}{\rho} \right\rceil \|\mathcal{G}\|_1}{2\phi}$$

et

$$\|\varepsilon' \cdot \mathcal{G}\|_\infty \leqslant \|\varepsilon'\|_\infty \|\mathcal{G}\|_1 \leqslant \frac{\|\mathcal{G}\|_1}{2}.$$

Ainsi

$$\begin{aligned} C - \pi_n(Q \cdot \mathcal{G}) &= C - \pi_n(\rho C_1 - \frac{1}{\phi} C_1 \cdot \varepsilon \cdot \mathcal{G} - \varepsilon' \cdot \mathcal{G}) \\ &= C - \pi_n(\rho C_1) + \pi_n(\frac{1}{\phi} C_1 \cdot \varepsilon \cdot \mathcal{G} + \varepsilon' \cdot \mathcal{G}) \\ &= C_0 + \pi_n(\frac{1}{\phi} C_1 \cdot \varepsilon \cdot \mathcal{G}) + \pi_n(\varepsilon' \cdot \mathcal{G}) \end{aligned}$$

On en déduit

$$\|C'\|_\infty \leqslant \|C_0\|_\infty + \|\pi_n(\frac{1}{\phi} C_1 \cdot \varepsilon \cdot \mathcal{G})\|_\infty + \|\pi_n(\varepsilon' \cdot \mathcal{G})\|_\infty \leqslant \frac{1}{2}\rho + \frac{n\left\lfloor \frac{w(\rho-1)^2}{\rho} \right\rceil \|\mathcal{G}\|_1}{2\phi} + \frac{\|\mathcal{G}\|_1}{2}.$$

Étant donné que l'on a supposé $\phi > n\left\lfloor \frac{w(\rho-1)^2}{\rho} \right\rceil$, on a $\frac{n\left\lfloor \frac{w(\rho-1)^2}{\rho} \right\rceil \|\mathcal{G}\|_1}{2\phi} < \frac{1}{2}\|\mathcal{G}\|_1$. On a donc $\|C'\|_\infty < \frac{1}{2}\rho + \|\mathcal{G}\|_1$. Donc, si $\rho \geqslant 2\|\mathcal{G}\|_1$, la sortie de l'algorithme 19 vérifie bien $\|C'\|_\infty < \rho$.

□

Cette méthode ne requiert pas de domaine spécifique contrairement à l'algorithme 18. Nous avons détaillé en section 1.5.4 le paramètre δ nécessaire lorsque des additions polynomiales sont effectuées avant une multiplication. La réduction Barrett-like permet une réduction rapide après addition sans modifier la valeur d'évaluation en γ de l'entrée contrairement à la réduction Montgomery-like, ce qui lui donne un intérêt particulier dans ce cas-là.

2.4.3 Réduction interne Plantard-like

Cette méthode adapte aux PMNS la méthode de réduction de Plantard sur les entiers présentée en section 1.2.5 qui est à la base une variante de la multiplication modulaire de Montgomery. Elle a la particularité de permettre un pré-calcul lors des opérations d'exponentiation modulaire. Cette adaptation effectue des arrondis de coefficients et non des parties entières pour obtenir de meilleures bornes (un arrondi ne coûte qu'une addition par coefficient supplémentaire par rapport à une partie entière).

Algorithme 20 Réduction interne Plantard-like

Require: $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS, $C \in \mathbb{Z}[X]$, $\phi \in \mathbb{N}^*$ et $\mathcal{G}$ une base courte d'un sous-réseau de $\mathfrak{L}$ inversible modulo ϕ^2.
Ensure: $C'(\gamma) \equiv C(\gamma)\phi^{-2} \bmod p$
1: $T \leftarrow \nu_n(C(X))$
2: $Q \leftarrow -T \cdot \mathcal{G}^{-1} \bmod \phi^2$
3: $S \leftarrow \left\lfloor \frac{1}{\phi} Q \right\rceil$
4: $C' \leftarrow \pi_n(\left\lfloor \frac{1}{\phi} S \cdot \mathcal{G} \right\rceil)$
5: **return** C'

Proposition 2.4.4. *Si* $\frac{\phi}{2}\|\mathcal{G}\|_1 + 2w(\rho-1)^2 < \frac{1}{2}\phi^2$ *et* $\|C\|_\infty \leqslant 2w(\rho-1)^2$, *alors l'algorithme 20 calcule correctement* $C'(X)$ *tel que* $C'(\gamma) \equiv C(\gamma)\phi^{-2} \pmod p$.

Démonstration. À l'étape 4 de l'algorithme, on a $C'(X) = \pi_n\left(\left\lfloor \frac{1}{\phi}\left(\left\lfloor \frac{1}{\phi}Q\right\rceil\right)\cdot\mathcal{G}\right\rceil\right)$.
Posons $Q_1 = \left\lfloor \frac{1}{\phi}Q\right\rceil$ et $Q_0 = Q - \phi Q_1$. On remarque que l'on a alors $\|Q_0\|_\infty \leqslant \frac{\phi}{2}$.

$$\begin{aligned}
\left\lfloor \frac{1}{\phi}\left(\left\lfloor \frac{1}{\phi}Q\right\rceil\right)\cdot\mathcal{G}\right\rceil &= \left\lfloor \frac{1}{\phi^2}\left(\phi Q_1\right)\cdot\mathcal{G}\right\rceil \\
&= \left\lfloor \frac{1}{\phi^2}\left(Q - Q_0\right)\cdot\mathcal{G}\right\rceil \\
&= \left\lfloor \frac{1}{\phi^2}(Q\cdot\mathcal{G} - Q_0\cdot\mathcal{G})\right\rceil \\
&= \left\lfloor \frac{1}{\phi^2}(Q\cdot\mathcal{G} + T - Q_0\cdot\mathcal{G} - T)\right\rceil \\
&= \left\lfloor \frac{1}{\phi^2}(Q\cdot\mathcal{G} + T) + \frac{1}{\phi^2}(-Q_0\cdot\mathcal{G} - T)\right\rceil
\end{aligned}$$

On note que $\|Q_0\cdot\mathcal{G}\|_\infty \leqslant \frac{\phi}{2}\|\mathcal{G}\|_1$. De plus, $\|T\|_\infty \leqslant 2w(\rho-1)^2$ du fait que $T = \nu_n(C(X))$ et on a supposé $\|C\|_\infty \leqslant 2w(\rho-1)^2$.

On en déduit que

$$\|\frac{1}{\phi^2}(Q_0\cdot\mathcal{G} + T)\|_\infty \leqslant \frac{1}{\phi^2}(\frac{\phi}{2}\|\mathcal{G}\|_1 + 2w(\rho-1)^2).$$

Or, par hypothèse,

$$\frac{\phi}{2}\|\mathcal{G}\|_1 + 2w(\rho-1)^2 < \frac{1}{2}\phi^2$$

donc

$$\|\frac{1}{\phi^2}(Q_0\cdot\mathcal{G} + T)\|_\infty < \frac{1}{2}.$$

De plus, puisque $Q \equiv -T\cdot\mathcal{G}^{-1} \bmod \phi^2$, on a $Q\cdot\mathcal{G} \equiv -T \pmod{\phi^2}$. Ou autrement dit $Q\cdot\mathcal{G}+T \equiv 0 \pmod{\phi^2}$.

D'où

$$\left\lfloor \frac{1}{\phi^2}(Q\cdot\mathcal{G} + T) + \frac{1}{\phi^2}(-Q_0\cdot\mathcal{G} - T)\right\rceil = \frac{1}{\phi^2}(Q\cdot\mathcal{G} + T) + \left\lfloor \frac{1}{\phi^2}(-Q_0\cdot\mathcal{G} - T)\right\rceil = \frac{1}{\phi^2}(Q\cdot\mathcal{G} + T).$$

Ainsi

$$\left\lfloor \frac{1}{\phi}\left(\left\lfloor \frac{1}{\phi}Q\right\rceil\right)\cdot\mathcal{G}\right\rceil = \frac{1}{\phi^2}(Q\cdot\mathcal{G} + T)$$

avec tous les coefficients de $Q\cdot\mathcal{G} + T$ des multiples de ϕ^2.

Or,

$$\pi_n(Q\cdot\mathcal{G} + T)(\gamma) \equiv \pi_n(Q\cdot\mathcal{G})(\gamma) + \pi_n(T)(\gamma) \equiv C(\gamma) \pmod{p}.$$

D'où

$$C'(\gamma) \equiv \pi_n(\frac{1}{\phi^2}(Q\cdot\mathcal{G} + T))(\gamma) \equiv C(\gamma)\phi^{-2} \pmod{p}.$$

□

Il reste maintenant à établir les conditions pour s'assurer que les coefficients de C' sont bien bornés par

ρ en sortie.

Proposition 2.4.5. *Si $\rho > \left\lfloor \frac{1}{2}\|\mathcal{G}\|_1 \right\rceil$ alors l'algorithme 20 retourne un polynôme C' tel que $\|C'\|_\infty < \rho$.*

Démonstration. À l'étape 2 de l'algorithme, on a $\|Q\|_\infty \leqslant \frac{\phi^2}{2}$ en effectuant le modulo en prenant des valeurs entre $-\frac{\phi^2}{2}$ et $\frac{\phi^2}{2} - 1$ au lieu de les prendre entre 0 et $\phi^2 - 1$. Il en découle que $\|S\|_\infty \leqslant \frac{\phi}{2} + 1$ à l'étape 3. Par conséquent, $\|S \cdot \mathcal{G}\|_\infty \leqslant (\frac{\phi}{2} + 1)\|\mathcal{G}\|_1$. On a donc

$$\|C'\|_\infty = \left\| \left\lfloor \frac{1}{\phi} S \cdot \mathcal{G} \right\rceil \right\|_\infty \leqslant \left\lfloor (\frac{1}{2} + \frac{1}{\phi})\|\mathcal{G}\|_1 \right\rceil = \left\lfloor \frac{1}{2}\|\mathcal{G}\|_1 \right\rceil .$$

□

Ainsi si $\rho > \left\lfloor \frac{1}{2}\|\mathcal{G}\|_1 \right\rceil$ et $\frac{\phi}{2}\|\mathcal{G}\|_1 + 2w(\rho-1)^2 < \frac{1}{2}\phi^2$, la méthode de réduction interne Plantard-like (algorithme 20) garantit que le résultat de la multiplication de deux éléments du PMNS est encore un élément du PMNS.

Remarque 26. On note que de façon similaire à l'algorithme 18 (voir remarque 24), on peut appliquer ici un domaine pour maintenir un facteur (ϕ^2 ici et non ϕ) après chaque opération.

Cette variante de la réduction interne, outre ses bornes intéressantes, permet des pré-calculs lorsque des opérations sont répétées. Soit $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS. Soient $a, b, c \in \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/p\mathbb{Z}$ avec $A, B, C \in \mathcal{B} \times \mathcal{B} \times \mathcal{B}$ tels que $A(\gamma) \equiv a\phi^2 \pmod p$, $B(\gamma) \equiv b\phi^2$ et $C(\gamma) \equiv c\phi^2 \pmod p$. Si l'on veut calculer $D_1, D_2 \in \mathcal{B} \times \mathcal{B}$ tels que $D_1(\gamma) \equiv ab\phi^2 \pmod p$ et $D_2(\gamma) \equiv cb\phi^2 \pmod p$, on peut pré-calculer $W(X) = \pi_n(\nu_n(B(X)) \cdot \mathcal{G}^{-1} \bmod \phi^2)$. On pourra ainsi prendre

$$D_1(X) = \pi_n\Big(\Big\lfloor \frac{1}{\phi}\Big(\Big\lfloor \frac{1}{\phi}\nu_n(-A(X)W(X) \bmod E(X) \bmod \phi^2)\Big\rceil\Big) \cdot \mathcal{G}\Big\rceil\Big)$$

et

$$D_2(X) = \pi_n\Big(\Big\lfloor \frac{1}{\phi}\Big(\Big\lfloor \frac{1}{\phi}\nu_n(-C(X)W(X) \bmod E(X) \bmod \phi^2)\Big\rceil\Big) \cdot \mathcal{G}\Big\rceil\Big),$$

économisant une multiplication vecteur-matrice au total. Cela est particulièrement utile pour une exponentiation, par exemple.

2.4.4 Le paramètre ϕ

Comme on peut le remarquer, chaque méthode de réduction interne mène à une borne différente sur ϕ. De façon générale, pour des raisons d'optimisation, on prend souvent $\phi = \sigma$ ($\sigma = 2^{64}$ sur la plupart des processeurs modernes) s'il existe nativement la possibilité de récupérer le résultat complet d'une multiplication de deux opérandes contenues sur un registre (le résultat étant donc borné par σ^2). En effet, on effectue des opérations soit de réduction modulaire soit de division par ϕ et ces opérations sont beaucoup plus rapides si $\phi = \sigma$. En particulier, la réduction modulaire ne coûte rien car le comportement de débordement naturel d'un registre mémoire mènera à un résultat modulo σ. La division par σ peut être remplacée par une simple instruction MOV, qui est une des opérations les plus rapides sur les processeurs modernes.

Dans le cas où prendre $\phi = \sigma$ s'avère être impossible, prendre $\phi = 2^h$, $h \in \mathbb{N}^*$ mène à des opérations plus rapides que pour ϕ quelconque. En effet, une division devient alors un simple décalage binaire et une réduction modulaire peut être effectuée par un simple masque.

Algorithme	Borne sur ϕ
Montgomery-like (algorithme 18)	$\phi > 2w(\rho-1)$
Barrett-like (algorithme 19)	$\phi > n \left\lfloor \frac{w(\rho-1)^2}{\rho} \right\rceil$
Plantard-like (algorithme 20)	$\phi(\phi - \|\mathcal{G}\|_1) > 4w(\rho-1)^2$

TABLE 2.1 – Tableau de comparaison des bornes sur ϕ pour chaque méthode de réduction interne présentée.

2.4.5 Le paramètre ρ

En raison de la condition exprimée en section 1.5 sur l'existence d'un représentant pour chaque élément de $\mathbb{Z}/p\mathbb{Z}$ dans un PMNS, on choisit toujours $\rho > \frac{1}{2}\|\mathcal{G}\|_1$ avec $\mathcal{G}$ une base courte de

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

Cependant, pour garantir la cohérence des bornes lors des opérations, il est parfois nécessaire de prendre ρ plus grand. En l'occurrence, le tableau 2.2 donne une borne sur ρ pour garantir que le résultat d'une multiplication polynomiale modulo E de deux polynômes dont les coefficients sont bornés par ρ aura bien ses coefficients bornés par ρ après réduction interne.

Dans la littérature, certains auteurs prennent ρ comme une puissance de 2 pour un processus de conversion plus rapide mais ceci n'est plus nécessaire (voir section 2.6). La seule considération restante est pour la réduction modulaire Barrett-like (algorithme 19) qui nécessite une division par ρ où prendre ρ comme une puissance de 2 permettrait d'effectuer un simple décalage binaire. Mais on peut tout aussi bien effectuer la division par $2^{\lceil \log_2(\rho) \rceil}$ (et prendre $\mathcal{G}' = \left\lfloor 2^{\lceil \log_2(\rho) \rceil} \phi \mathcal{G}^{-1} \right\rceil$)) en notant qu'il faudra alors que $\phi > n \left\lfloor \frac{w(\rho-1)^2}{2^{\lceil \log_2(\rho) \rceil}} \right\rceil$.

Algorithme	Borne sur ρ
Montgomery-like (algorithme 18)	$\rho \geqslant \|\mathcal{G}\|_1 - 1$
Barrett-like (algorithme 19)	$\rho \geqslant 2\|\mathcal{G}\|_1$
Plantard-like (algorithme 20)	$\rho \geqslant \left\lfloor \frac{1}{2}\|\mathcal{G}\|_1 \right\rceil + 1$

TABLE 2.2 – Tableau de comparaison des bornes sur ρ pour chaque méthode de réduction interne présentée.

2.4.6 Le paramètre δ

Soit $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS. Lorsque l'on applique la réduction interne Montgomery-like à un polynôme résultant de la multiplication de deux éléments du PMNS, si l'on choisit $\phi > 2w(\rho-1)(1+\delta)^2$ alors la sortie obtenue est un élément du PMNS si les 2 opérandes de la multiplication sont issus d'au plus

δ additions polynomiales. Autrement dit, la valeur de δ dans un PMNS donné peut être exprimée comme

$$\delta_{\max} = \sqrt{\frac{\phi}{2w(\rho-1)}} - 1 \tag{2.9}$$

De façon similaire, pour les réductions internes Barrett-like et Plantard-like on aura respectivement

$$\delta_{\max} = \sqrt{\frac{\phi}{n\left\lfloor \frac{w(\rho-1)^2}{\rho} \right\rceil}} - 1 \tag{2.10}$$

et

$$\delta_{\max} = \sqrt{\frac{\phi(\phi - \|\mathcal{G}\|_1)}{4w(\rho-1)^2}} - 1 \tag{2.11}$$

En particulier si l'on dépasse δ additions, la meilleure méthode de réduction interne pour revenir à un polynôme borné par ρ est la méthode Barrett-like car elle ne sort pas du domaine en cours s'il y en a un. Dans les autres cas, il faudra multiplier par un représentant de ϕ, respectivement ϕ^2, avant d'effectuer une réduction Montgomery-like, respectivement Plantard-like.

2.4.7 Analyse de complexité

Dans cette section nous comparons les 3 algorithmes de réduction présentés plus haut. Chacun effectue 2 opérations de multiplication vecteur-matrice, ce qui indique que les fonctions de complexité de ces 3 algorithmes sont très similaires.

Soit $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS avec $n = \kappa_1 \times \kappa_2 \times \cdots \times \kappa_d$ avec chaque κ_i un nombre premier. On suppose pour l'instant $\phi = \sigma$ pour simplifier. Si l'on utilise l'algorithme de multiplication vecteur-matrice de Toeplitz (algorithme 17), on aurait alors un nombre de multiplications de $\prod_{i=1}^{d}\left(\frac{\kappa_i(\kappa_i+1)}{2}\right)$ (cf. proposition 2.3.7) pour chaque opération vecteur-matrice. Pour alléger la notation, on notera $\mathrm{Toep}(n) = \prod_{i=1}^{d}\left(\frac{\kappa_i(\kappa_i+1)}{2}\right)$ pour le reste de cette section.

Cependant, l'algorithme 20 effectue des multiplications sur 2 registres mémoire. Si on considère que le coût d'une multiplication est celui d'une multiplication avec deux opérandes de $\log_2(\sigma)$ bits dont le résultat est sur $2\log_2(\sigma)$ bits, le tableau 1.1, page 24, indique qu'une multiplication de $\log_2(\sigma)$ bits par $2\log_2(\sigma)$ bits coûte environ 1,4 multiplications tandis qu'une multiplication de $2\log_2(\sigma)$ par $2\log_2(\sigma)$ bits en coûte environ 1,5. Une multiplication modulo ϕ pour $\phi = \sigma$ ne coûte que 0,6 multiplications. Cela donne le tableau suivant.

Algorithme \Étape	Multiplication par $\mathcal{G}^{-1}/\mathcal{G}'$	Multiplication par $\mathcal{G}$	Total
Montgomery-like 18	0,6Toep(n)	n + 1,4(Toep(n) − n)	2Toep(n) − 0,4n
Barrett-like 19	Toep(n)	Toep(n)	2Toep(n)
Plantard-like 20	1,5Toep(n)	n + 1,4(Toep(n) − n)	2,9Toep(n) − 0,4n

TABLE 2.3 – Nombre de multiplications effectuées entre opérandes de taille $\log_2(\sigma)$ pour chaque étape d'un algorithme de réduction interne dans le cas où $\phi = \sigma$.

La valeur n+1,4(Toep(n)−n) dans le tableau indique une opération de multiplication vecteur-matrice de

Toeplitz récursive lorsque le vecteur a été réduit modulo $\phi = \sigma$. En effet, exactement n opérations effectuées au total ne nécessiteront pas d'additions/soustractions de coefficients sur les vecteurs, donc ces opérations tiendront sur un seul coefficient. Le nombre restant de multiplications est donc de $(\text{Toep}(n) - n)$.

Si l'on considère que le coût principal en termes de complexité d'opérations est celui en multiplications, ce tableau semble indiquer que la réduction Montgomery-like est la plus rapide. En contraste, la réduction Plantard-like semble la plus lente, bien que permettant de meilleures bornes sur les paramètres.

Si l'on choisit plutôt $\phi < \sigma$, alors les réductions Montgomery-like et Plantard-like se voient plus impactées que la réduction Barrett-like. La réduction modulaire demandera un masque sur chaque coefficient si ϕ est une puissance de 2. La réduction Barrett-like ne demande pas de réduction modulaire donc ces coûts supplémentaires ne s'appliquent pas.

Pour les étapes de divisions, chacune coûtera une multiplication de plus par coefficient si $\phi < \sigma$ (ou un décalage binaire qui a un coût proche d'une multiplication). Le coût supplémentaire de choisir $\phi < \sigma$ lorsque ϕ est une puissance de 2 est linéaire en n mais non négligeable.

En ce qui concerne le nombre d'additions, le tableau suivant résume le coût pour chaque algorithme

Algorithme\Étape	Multiplication par $\mathcal{G}^{-1}/\mathcal{G}'$	Multiplication par $\mathcal{G}$	Total
Montgomery-like 18	$\text{Toep}_+(n) - \text{Ext}(n)$	$\text{Toep}_+(n) + \text{Rec}(n) + 2n$	$2\text{Toep}_+(n) + \text{Rec}(n) - \text{Ext}(n) + 2n$
Barrett-like 19	$\text{Toep}_+(n)$	$\text{Toep}_+(n) + 2n$	$2\text{Toep}_+(n) + 2n$
Plantard-like 20	$\text{Toep}_+(n) + \text{Mat}(n)$	$\text{Toep}_+(n) + \text{Rec}(n)$	$2\text{Toep}_+(n) + \text{Rec}(n) + \text{Mat}(n)$

TABLE 2.4 – Nombre d'additions/soustractions effectuées sur des opérandes de taille $\log_2(\sigma)$ pour chaque étape d'un algorithme de réduction interne pour $\phi = \sigma$.

avec $\text{Toep}_+(n)$ le nombre d'additions de la proposition 2.3.8,

$$\text{Ext}(n) = \sum_{i=1}^{d} \left(\left(\prod_{j=0}^{i-1} \frac{\kappa_j(\kappa_j+1)}{2} \right) \times \left(\frac{n}{\prod_{j=0}^{i-1} \kappa_j} (\kappa_i - 1) \right) \right),$$

$$\text{Rec}(n) = \sum_{i=1}^{d} \left(\left(\prod_{j=0}^{i-1} \frac{\kappa_j(\kappa_j+1)}{2} \right) \times \left(\left(\frac{2n}{\prod_{j=1}^{i} \kappa_j} - 1 \right) (3\kappa_i - 4) \right) \right) \text{ et}$$

$$\text{Mat}(n) = \sum_{i=1}^{d} \left(\left(\prod_{j=0}^{i-1} \frac{\kappa_j(\kappa_j+1)}{2} \right) \times \left(\frac{n}{\prod_{j=0}^{i-1} \kappa_j} \left(\frac{\kappa_i - 1}{2} \right) \right) \right).$$

La réduction Barrett-like n'a pas de considérations supplémentaires en termes d'additions. La réduction Montgomery-like effectue une réduction modulo ϕ à l'étape de multiplication par $\mathcal{G}^{-1}$ et donc les additions qui sont censées être sur 2 registres mémoire ne sont que sur un seul registre mémoire. $\text{Ext}(n)$ correspond à ce nombre d'opérations. En contraste, les éléments sortant de la réduction modulo ϕ provoqueront des dépassements mémoires lors des additions intermédiaires de l'algorithme 17 et nécessitent donc d'être rangés sur 2 registres mémoire chacun, ce qui entraîne un coût supplémentaire. $\text{Rec}(n)$ correspond à ce nombre

d'additions supplémentaires. Les considérations pour la réduction Plantard-like sont presque similaires avec la différence que la multiplication $\mathcal{G}^{-1}$ est effectuée intégralement donc le coût n'est pas réduit. En contrepartie les opérations sur les matrices sont plus coûteuses car elles ont des coefficients sur 2 registres. Ce surcoût correspond à $\mathrm{Mat}(n)$. L'étape de multiplication par $\mathcal{G}$ dans l'algorithme de réduction Plantard-like est identique à celle présente dans la réduction Montgomery-like donc les mêmes surcoûts sont entraînés.

2.4.8 Résultats

Dans cette section nous analysons les performances des 3 méthodes de réduction interne, tirant parti de la multiplication vecteur-matrice de toeplitz pour chaque cas sauf pour $n = 5$. Un code pour générer les tables de performance est disponible à l'adresse https://github.com/francoispalma/PMNS/tree/master/internal_reduction_comparison .

La procédure suivante a été mise en place pour effectuer les tests :

— Le *Turbo-Boost*® est désactivé pendant les tests ;

— 501 exécutions sont lancées pour "chauffer" la mémoire cache, c'est à dire que nous nous assurons que la mémoire cache (données et instructions) est dans un état suffisamment stable pour éviter les défauts de cache ;

— 1001 jeux de données aléatoires sont générés, et le nombre de cycles processeurs pour W exécutions sur un lot de 501 exécutions est enregistré pour chaque jeu de données ;

— la mesure finale est la valeur médiane divisée par W.

(À noter que la valeur de W dépend des tailles de paramètres).

$\log_2(p)$	256	512		1024			2048		
n	5	9	10	18	20	21	38	42	44
Plantard-like	169	507	618	2048	2720	3021	9314	9794	13125
Montgomery-like	108	310	369	-	1750	1737	-	6695	8931
Barrett-like	119	-	382	-	-	1600	-	-	7709

TABLE 2.5 – Nombre de cycles pour une multiplication modulaire sur des entiers de tailles de 256 à 2048 bits pour chaque méthode de réduction interne présentées dans cette section avec gcc 12.3.0 sur un processeur intel i9-11900KF.

$\log_2(p)$	4096			6144			8192		
n	80	84	100	128	132	152	168	184	232
Plantard-like	33065	33892	48584	75480	79650	110106	117485	157683	249113
Montgomery-like	-	22915	33896	-	51887	72088	-	105217	158601
Barrett-like	-	-	26230	-	-	69843	-	-	152731

TABLE 2.6 – Nombre de cycles pour une multiplication modulaire sur des entiers de tailles de 4096 à 8192 bits pour chaque méthode de réduction interne présentées dans cette section avec gcc 12.3.0 sur un processeur intel i9-11900KF.

Dans les tableaux 2.5 et 2.6 le symbole - signifie une taille où un PMNS n'existe pas pour cette méthode de réduction pour le n correspondant. Comme l'on peut l'observer, la méthode Plantard-like permet de choisir n plus petit grâce aux bornes plus généreuses. En contraste, la méthode Barrett-like semble la plus

rapide pour un n donné mais nécessite un n bien plus grand. Ainsi, la méthode Montgomery-like se dégage comme étant le meilleur compromis.

$\log_2(p)$	256	512	1024	2048	4096	6144	8192
Plantard-like	44	333	2703	24790	163666	583655	1152958
Montgomery-like	43	237	2665	20443	139179	472470	1332253
Barrett-like	47	299	2438	23509	160086	646854	1394221

TABLE 2.7 – Nombre de **milliers** de cycles pour un left to right square and multiply sur des entiers de tailles de 256 à 8192 bits pour chaque méthode de réduction interne présentées dans cette section avec gcc 12.3.0 sur un processeur intel i9-11900KF.

La table 2.7 recense les performances lors d'une exponentiation modulaire avec des PMNS. En théorie, la méthode Plantard-like devrait présenter des avantages dans cette situation tout comme dans la version sur les entiers. En effet, l'un des opérandes peut être calculé une seule fois, ce qui économise des opérations. Cet avantage semble se manifester sur des tailles d'entiers de 8192 bits et plus. À noter qu'il ne s'agit pas ici d'un vrai square and multiply étant donné que l'opération de mise au carré a été remplacée par une multiplication.

2.5 Génération

Dans cette section, nous abordons le processus de génération d'un PMNS générique. Il paraît naturel, une fois les pré-requis posés, de rechercher les meilleures performances possibles. Nous commençons donc par considérer une borne inférieure sur le paramètre n, dont dépend en grande partie la complexité des opérations au sein des PMNS.

2.5.1 Le paramètre n_{opt}

Comme précédemment expliqué dans la section 1.5.2, Didier et al. [15] ont introduit le paramètre $n_{opt} = \left\lfloor \frac{\log_2(p)}{\log_2(\sigma)} \right\rfloor + 1$. Cependant, il semblerait que cette borne ne soit pas la plus proche possible étant donné la progression du paramètre n lorsque la taille des entiers premiers considérés augmente.

Taille d'entier en bit	256	512	1024	2048	4096	6144	8192
n_{opt}	5	9	17	33	65	97	129
n_{actual}	5	9	18	36	72	108	144

TABLE 2.8 – Tableau de comparaison entre la valeur de n_{opt} présentée dans la littérature et le nombre de registres mémoire de 64-bit nécessaires pour un PMNS de la taille considérée (notée n_{actual}).

Comme on peut le voir dans le tableau, l'écart se creuse pour les plus grandes tailles. Cela s'explique par la proposition suivante présentée dans [44].

Proposition 2.5.1. *Soit* $n_{opt} = \left\lfloor \frac{\log_2(p)}{\log_2(\sigma)} \right\rfloor + 1$. *Pour tout* $\varepsilon > 0$, *pour tout* $\xi > 0$, *il existe un entier* p *tel que* $p^{\frac{1}{n_{opt}+\xi}} \geqslant \sigma - \varepsilon$.

Démonstration. On a

$$n_{opt} + \xi \leqslant \frac{\log_2(p)}{\log_2(\sigma)} + 1 + \xi$$

$$\iff n_{opt} + \xi \leqslant \frac{\log_2(p) + \log_2(\sigma) + \xi \log_2(\sigma)}{\log_2(\sigma)}$$

$$\iff \frac{\log_2(p)}{n_{opt} + \xi} \geqslant \frac{\log_2(p)\log_2(\sigma)}{\log_2(p) + \log_2(\sigma) + \xi \log_2(\sigma)}$$

$$\iff p^{\frac{1}{n_{opt}+\xi}} \geqslant \sigma^{\frac{\log_2(p)}{\log_2(p)+\log_2(\sigma)+\xi\log_2(\sigma)}}$$

Puisque

$$\lim_{p \to +\infty} \frac{\log_2(p)}{\log_2(p) + \log_2(\sigma) + \xi \log_2(\sigma)} = 1$$

alors il existe bien un p tel que

$$p^{\frac{1}{n_{opt}+\xi}} \geqslant \sigma - \varepsilon$$

□

Il est clair que l'ancienne définition de n_{opt} n'est pas réaliste. Dans cette section, nous présentons une nouvelle expression pour n_{opt}.

Pour commencer, rappelons le théorème de Minkowski :

Théorème de Minkowski. [46] *Soit $\mathfrak{L} \subseteq \mathbb{R}^n$, un réseau de rang plein. Soit $S \subseteq \mathbb{R}^n$ un espace convexe centralement symétrique. Si $vol(S) > 2^n \det(\mathfrak{L})$, alors S contient au moins un vecteur non nul du réseau.*

Une conséquence directe est le lemme suivant :

Lemme 2.5.1. *Soit $\mathfrak{L} \subseteq \mathbb{R}^n$, un réseau de rang plein.*
Soit $v \in \mathfrak{L} \setminus \{0\}$ tel que $\forall x \in \mathfrak{L} \setminus \{0\}, \|x\|_1 \geqslant \|v\|_1$. Alors $\|v\|_1 \leqslant (n! \det(\mathfrak{L}))^{\frac{1}{n}}$.

Démonstration. Soit $v \in \mathfrak{L} \setminus \{0\}$ tel que $\forall x \in \mathfrak{L} \setminus \{0\}, \|x\|_1 \geqslant \|v\|_1$. Autrement dit, v est le vecteur non nul le plus court de $\mathfrak{L}$ en termes de norme 1. Soit $S = \{x \in \mathbb{R}^n : \|x\|_1 < \|v\|_1\}$. S est à la fois convexe et centralement symétrique, nous plaçant dans les conditions du théorème de Minkowski. S est une n-boule en norme L^1 de rayon $\|v\|_1$ par construction, donc son volume est de $vol(S) = \frac{2^n}{n!}(\|v\|_1)^n$. Si l'on suppose que $\|v\|_1 > (n! \det(\mathfrak{L}))^{\frac{1}{n}}$, on en déduit que $vol(S) > 2^n \det(\mathfrak{L})$. D'après le théorème de Minkowski, S contient alors au moins un vecteur non nul du réseau $\mathfrak{L}$. Cela contredit la supposition que v est le vecteur non nul le plus court de $\mathfrak{L}$ en termes de norme infinie puisque tous les éléments de S ont une norme infinie plus courte que v. Il en découle que $\|v\|_1 \leqslant (n! \det(\mathfrak{L}))^{\frac{1}{n}}$. □

Ceci est un résultat classique sur les réseaux qui nous permet d'établir une borne réaliste sur la norme 1 d'une base courte d'un réseau. Plus précisément, cela nous permet de poser la proposition suivante.

Proposition 2.5.2. *Soit $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS avec $E(X) = X^n + 1$ et $n = 2^h$ avec $h \in \mathbb{N}^*$. Soit $\mathfrak{L} = \left\{(x_0, \ldots, x_{n-1}) \in \mathbb{Z}^n : \sum_{i=0}^{n-1} x_i \gamma^i \equiv 0 \pmod{p}\right\}$. Alors il existe $\mathcal{G}$ une base de $\mathfrak{L}$ telle que $\|\mathcal{G}\|_1 \leqslant (n!p)^{\frac{1}{n}}$.*

Démonstration. Soit $v \in \mathfrak{L} \setminus \{0\}$ tel que $\forall x \in \mathfrak{L} \setminus \{0\}, \|x\|_1 \geqslant \|v\|_1$. Autrement dit, v est le vecteur le plus court du réseau en termes de norme une. D'après le lemme 2.5.1, on a alors $\|v\|_1 \leqslant (n! \det(\mathfrak{L}))^{\frac{1}{n}}$. Étant donné que $\det(\mathfrak{L}) = p$ (cf. remarque 5 page 30) on a donc $\|v\|_1 \leqslant (n!p)^{\frac{1}{n}}$.

On note $v = (v_0, v_1, \ldots, v_{n-1})$ les coefficients de v. Alors, on peut construire la base suivante

$$\mathcal{G} = \begin{pmatrix} v_0 & v_1 & \ldots & v_{n-2} & v_{n-1} \\ -v_{n-1} & v_0 & \ldots & v_{n-3} & v_{n-2} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ -v_1 & -v_2 & \ldots & -v_{n-1} & v_0 \end{pmatrix} \begin{matrix} \leftarrow v \\ \leftarrow \nu_n(X \times \pi_n(v) \bmod E) \\ \vdots \\ \vdots \\ \leftarrow \nu_n(X^{n-1} \times \pi_n(v) \bmod E) \end{matrix} \tag{2.12}$$

Le polynôme $X^{2^h} + 1$ est irréductible, un résultat classique qui peut être obtenu avec le critère d'Eisenstein donc E est irréductible. Pour tout $P \in \mathbb{Z}_{n-1}[X] \setminus \{0\}$, le résultant de E et de P, noté $\mathrm{Res}(E, P)$ est non nul. D'après [20, Corollaire 2], on a $\det(\mathcal{G}) = \mathrm{Res}(E, \pi_n(v))$. D'où $\det(\mathcal{G}) \neq 0$. Les lignes sont donc linéairement indépendantes les unes des autres. De plus, $\forall i \in [\![0, n-1]\!]$, $\nu_n(X^i \times \pi_n(v)) \in \mathfrak{L}$. Donc $\mathcal{G}$ est bien une base d'un sous-réseau de $\mathfrak{L}$.

Comme on peut le remarquer, on a $\|\mathcal{G}\|_1 = \|v\|_1$. Donc $\|\mathcal{G}\|_1 \leqslant (n!p)^{\frac{1}{n}}$. □

Remarque 27. Il s'agit ici d'un cas particulier de [54, Lemme 4]. En effet, ce lemme affirme que pour tout PMNS $\mathcal{B} = (p, n, \gamma, \rho, E)$ avec $E(X) = X^n - \lambda$, il existe un polynôme M tel que la matrice de réduction interne $\mathcal{M}$ (cf. équation (1.4), page 37) qui lui est associé vérifie $\|\mathcal{M}\|_1 \leqslant |\lambda|(\frac{n!}{|\lambda|}p)^{\frac{1}{n}}$. Pour $E(X) = X^n + 1$, cela revient à dire que nous avons $\|\mathcal{M}\|_1 \leqslant (n!p)^{\frac{1}{n}}$.

Donc dans certains cas la norme une de la base courte d'un sous-réseau de $\mathfrak{L}$ peut être majorée par $(n!p)^{\frac{1}{n}}$. Comme noté en section 1.5, on choisira toujours ρ tel que $\rho > \frac{1}{2}\|\mathcal{G}\|_1$ avec $\mathcal{G}$ une base courte d'un sous-réseau de $\mathfrak{L}$ pour garantir l'existence d'un représentant dans le PMNS pour chaque entier de $\mathbb{Z}/p\mathbb{Z}$. Donc une borne que l'on peut choisir à des fins de génération peut être $\rho > \frac{1}{2}(n!p)^{\frac{1}{n}}$. Ce choix permet d'accélérer la recherche de PMNS par rapport au choix de $\rho > \frac{1}{2}(p)^{\frac{1}{n}}$. En effet, nous devrons alors par exemple choisir n tel que $\phi > 2w(\frac{1}{2}(n!p)^{\frac{1}{n}} - 1)$ pour les bornes de la méthode de réduction Montgomery-like (algorithme 18), ce qui imposera de choisir un n initial plus grand lorsque la taille de p augmente. Les valeurs de n en dessous de ce nouveau n initial n'auraient très probablement pas mené à des PMNS viables en pratique.

Par ailleurs, on doit toujours avoir $\rho \leqslant \sigma$ afin que les coefficients polynomiaux tiennent sur un seul registre. On peut imaginer un PMNS où l'on prendrait par exemple $\sigma^2 \geqslant \rho > \sigma$. On diviserait alors n par au plus 2 mais chaque coefficient tiendrait sur 2 registres mémoire. Cette possible représentation multi-précision des coefficients des polynômes manipulés au sein des PMNS est étudiée plus en profondeur dans le chapitre 3. Dans ce chapitre, nous nous contentons de considérer le cas $\rho \leqslant \sigma$ avec des coefficients simple-précision.

Puisque chaque coefficient peut être positif ou négatif, on prendra plutôt $\sigma \geqslant 2\rho$. Il en découle donc la borne

$$\sigma \geqslant (n!p)^{\frac{1}{n}} \tag{2.13}$$

Ainsi, le n optimal pour un p donné afin de construire un PMNS est le plus petit n qui satisfait cette borne. On a donc

$$2^{n\log_2(\sigma) - \log_2(n!)} \geqslant p$$

$$\implies n\log_2(\sigma) - \log_2(n!) \geqslant \log_2(p)$$

Notre nouvelle valeur de n_{opt} est donc le plus petit n vérifiant cette inégalité.
Cela nous donne la table suivante :

Taille d'entiers	256	512	1024	2048	4096	6144	8192
ex n_{opt}	5	9	17	33	65	97	129
nouveau n_{opt}	5	9	17	34	70	105	141
n_{actual}	5	9	18	36	72	108	144

TABLE 2.9 – Tableau de comparaison entre la valeur de n_{opt} présentée dans la littérature, la nouvelle expression de n_{opt} et le nombre de registres mémoire de 64-bit nécessaires pour un PMNS de la taille considérée (notée n_{actual}).

La nouvelle estimation s'avère plus précise. Une autre façon d'exprimer n_{opt} est

$$n_{opt} = \left\lfloor \frac{\log_2(p) + \log_2(n_{opt}!)}{\log_2(\sigma)} \right\rfloor + 1 \tag{2.14}$$

ce qui permet d'exprimer clairement le seuil à partir duquel cette nouvelle expression surpasse l'ancienne.

Proposition 2.5.3. *Soit p tel que $\log_2\left(\left(\left\lfloor \frac{\log_2(p)}{\log_2(\sigma)} \right\rfloor + 1\right)!\right) \geqslant \log_2(\sigma)$, alors*

$$n_{opt} > \left\lfloor \frac{\log_2(p)}{\log_2(\sigma)} \right\rfloor + 1$$

Démonstration. Tout d'abord, remarquons que $\left\lfloor \frac{\log_2(p)+\log_2(n_{opt}!)}{\log_2(\sigma)} \right\rfloor + 1 \geqslant \left\lfloor \frac{\log_2(p)}{\log_2(\sigma)} \right\rfloor + 1$. Ainsi, puisque $n_{opt} = \left\lfloor \frac{\log_2(p)+\log_2(n_{opt}!)}{\log_2(\sigma)} \right\rfloor + 1$, alors

$$n_{opt} \geqslant \left\lfloor \frac{\log_2(p)}{\log_2(\sigma)} \right\rfloor + 1$$

Donc dès que $\log_2\left(\left(\left\lfloor \frac{\log_2(p)}{\log_2(\sigma)} \right\rfloor + 1\right)!\right) \geqslant \log_2(\sigma)$, alors $\frac{\log_2(n_{opt}!)}{\log_2(\sigma)} \geqslant 1$.

En conséquence

$$n_{opt} = \left\lfloor \frac{\log_2(p) + \log_2(n_{opt}!)}{\log_2(\sigma)} \right\rfloor + 1 \geqslant \left\lfloor \frac{\log_2(p)}{\log_2(\sigma)} + 1 \right\rfloor + 1 > \left\lfloor \frac{\log_2(p)}{\log_2(\sigma)} \right\rfloor + 1$$

□

Par exemple, pour $\sigma = 2^{64}$, cette borne est atteinte pour $p \geqslant 2^{1280}$. Cela est dû au fait que $1280 = 64 \times 20$ et $21! > 2^{64}$ mais $20! < 2^{64}$. Ainsi, pour tout entier de taille supérieure à 1280 bits, notre nouvelle expression de n_{opt} est strictement supérieure à l'ancienne expression.

2.5.2 Algorithme de génération

Nous avons ainsi établi une borne inférieure sur le paramètre n. Ceci nous permet de définir à présent l'algorithme de génération suivant.

Algorithme 21 Génération de PMNS

Require: ℓ la taille d'entiers premiers considérés en bits, ϕ un paramètre utilisé dans la réduction interne qui nous permettra de borner le paramètre ρ lors de la génération, $\delta_{\min} \in \mathbb{N}$ une borne inférieure sur le paramètre δ escompté.

Ensure: un tuple $(p, n, \gamma, \rho, E, \delta)$ définissant un PMNS ayant au moins un représentant pour chaque élément de $\mathbb{Z}/p\mathbb{Z}$ avec $\delta \geqslant \delta_{\min}$.

1: $p \leftarrow$ random_prime$(2^{\ell-1}, 2^{\ell} - 1)$
2: $n \leftarrow n_{opt}$ (cf. équation (2.14))
3: **tant que** un PMNS n'est pas trouvé **faire**
4: **pour chaque** forme de E de degré n considérée **faire**
5: **si** E est irréductible dans $\mathbb{Z}$ mais a des racines dans $\mathbb{Z}/p\mathbb{Z}$ **alors**
6: $\gamma \leftarrow$ une racine de E
7: $\mathbf{B} \leftarrow$ la matrice de l'équation (1.1), page 29
8: $\mathcal{G} \leftarrow \mathbf{LLL}(B)$
9: $\rho \leftarrow \left\lfloor \frac{1}{2}\|\mathcal{G}\|_1 \right\rceil + 1$ ou $\|\mathcal{G}\|_1 - 1$ ou $2\|\mathcal{G}\|_1$ selon la méthode de réduction
10: $\delta \leftarrow \left\lfloor \sqrt{\frac{\phi(\phi - 3\|\mathcal{G}\|_1)}{4w(\rho-1)^2}} \right\rfloor - 1$ ou $\left\lfloor \sqrt{\frac{\phi}{2w\rho}} \right\rfloor - 1$ ou $\left\lfloor \sqrt{\frac{\phi}{n\left\lfloor \frac{w(\rho-1)^2}{\rho} \right\rceil}} \right\rfloor - 1$ selon la méthode de réduction
11: **si** $\delta \geqslant \delta_{\min}$ **alors**
12: **return** $(p, n, \gamma, \rho, E, \delta)$ # si $\delta \geqslant 0$, les bornes sur ϕ sont respectées
13: **fin si**
14: **fin si**
15: **fin pour**
16: $n \leftarrow n + 1$
17: **fin tant que**

Remarque 28. Nous n'avons pas besoin d'effectuer de comparaison sur ϕ dans l'algorithme 21 avant de valider un PMNS. En effet, $\delta \geqslant 0$ implique que les conditions entre ϕ et ρ sont bien vérifiées.

Remarque 29. Il est possible de substituer la ligne 1 de l'algorithme 21 par un p fixé pour les cas d'usage où l'entier premier est défini dans le standard et ne nécessite pas de génération aléatoire.

Remarque 30. On peut également substituer la ligne 8 de l'algorithme 21 par un autre algorithme de recherche de base courte, ou bien par la matrice de réduction interne définie par un polynôme M une fois un tel M trouvé comme vecteur court dans le réseau inversible modulo E modulo ϕ.

Remarque 31. Pour le choix de E, le tableau 1.2, page 35, donne les formes de polynômes unitaires les plus intéressantes.

2.5.3 Construction d'un polynôme M à coefficients positifs inversible modulo E et ϕ

Nous avons présenté en section 1.5.4 les travaux dans [15, 20] détaillant les conditions exactes pour garantir l'obtention d'un polynôme M inversible modulo E modulo ϕ. Dans cette section, nous étudions une heuristique pour trouver un tel polynôme M minimisant la norme 1 de la matrice de réduction interne dans le cas où $E(X) = X^n - \lambda$.

Une autre problématique est de trouver un polynôme M dont tous les coefficients sont positifs à des fins d'utilisation dans les instructions SIMD AVX512IFMA qui n'existent qu'en version non-signées. Nous

proposons dans cette section une méthode pour trouver systématiquement un tel polynôme dès lors que $E(X) = X^n - \lambda$ avec λ pair.

Après avoir construit une base courte en appliquant LLL (ou tout autre algorithme de recherche de base courte) à la base de l'équation (1.1) page 29, le polynôme M est souvent choisi comme la ligne de la base courte qui est telle que la matrice de réduction interne $\mathcal{M}$ (équation (1.4) page 37) possède la plus petite norme 1 possible, tout en étant inversible modulo ϕ.

Proposition 2.5.4. *Soit $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS avec $E(X) = X^n - \lambda$. Soit*

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

Soit $v = (v_0, v_1, \ldots, v_{n-1}) \in \mathfrak{L}$ et $\mathcal{V}$ la matrice de réduction interne associée à v. Soit $w = (v_1, \ldots, v_{n-1}, \frac{v_0}{\lambda})$ et $\mathcal{W}$ la matrice de réduction interne associée à w. Si $v_0 \equiv 0 \pmod{\lambda}$, alors $w \in \mathfrak{L}$ et $\|\mathcal{W}\|_1 \leqslant \|\mathcal{V}\|_1$.

Démonstration. Soit $v = (v_0, v_1, \ldots, v_{n-1}) \in \mathfrak{L}$. On note $V(X) = \pi_n(v)$. Soit $w = (v_1, \ldots, v_{n-1}, \frac{v_0}{\lambda})$. On note

$$W(X) = v_1 + v_2 X + \cdots + v_{n-1} X^{n-2} + \frac{v_0}{\lambda} X^{n-1}.$$

Alors, si $v_0 \equiv 0 \pmod{\lambda}$, $W(X) \in \mathbb{Z}[X]$. De plus, $W(X) \times X \equiv V(X) \pmod{E(X)}$ puisque $\frac{v_0}{\lambda} X^n \equiv v_0 \pmod{E(X)}$. Étant donné que $v \in \mathfrak{L}$, on a $V(\gamma) \equiv 0 \pmod{p}$ et donc $W(\gamma) \equiv \frac{V(\gamma)}{\gamma} \equiv 0 \pmod{p}$. Ainsi $w \in \mathfrak{L}$.

De plus, $\|\mathcal{V}\|_1 = |v_0| + |\lambda| \sum_{i=1}^{n-1} |v_i|$, en raison de la forme de la matrice, et $\|\mathcal{W}\|_1 = |v_1| + |\lambda| \sum_{i=2}^{n-1} |v_i| + |\lambda| |\frac{v_0}{\lambda}|$. Donc $\|\mathcal{W}\|_1 \leqslant \|\mathcal{V}\|_1$. □

Remarque 32. Il est logique de répéter cette transformation tant que le coefficient en première position est divisible par λ. En particulier, pour $\lambda = 2$, ce processus garantit d'obtenir un vecteur dont la première composante est impaire au final. Il est démontré dans [15, Proposition 8] que cette condition est suffisante pour garantir que le polynôme soit inversible modulo ϕ pour ϕ une puissance de 2.

Ainsi, nous présentons l'heuristique suivante qui permet de sélectionner M pour minimiser la norme 1 de la matrice de réduction interne :

1. On calcule $\mathbf{B}_{LLL}$, la forme LLL-réduite de $\mathbf{B}$.
2. On note $L_1(X) = \pi_n(l_1)$ avec l_1 la première ligne de $\mathbf{B}_{LLL}$.
3. On calcule L_1/X^i modulo $E(X)$ tant que L_1/X^i modulo $E(X)$ a des coefficients entiers.
4. On choisit $M_1 = L_1/X^j \pmod{E(X)}$ avec j la plus grande valeur telle que M_1 est inversible modulo $E(X)$ modulo ϕ.
5. On répète pour chaque ligne de $\mathbf{B}_{LLL}$.
6. On choisit M pour minimiser la norme 1 parmi les M_k.

Une ligne donnée n'est pas garantie de fournir un polynôme inversible modulo $E(X)$ modulo ϕ avec cette heuristique sauf si $\lambda = 2$.

Une autre considération est le signe. Comme noté plus haut, le jeu d'instructions SIMD AVX512IFMA nécessite l'utilisation d'un polynôme M dont tous les coefficients sont positifs.

Ainsi nous proposons ici une heuristique pour trouver systématiquement une combinaison linéaire des lignes de la base courte dont le résultat ne possède que des coefficients positifs et qui est garantie d'être inversible modulo E modulo ϕ pour λ pair.

1. On initialise $C(X) = \pi_n(\|\mathbf{B_{LLL}}\|_1 + 1, \|\mathbf{B_{LLL}}\|_1 + 1, \ldots, \|\mathbf{B_{LLL}}\|_1 + 1)$ et on pose $\rho' = \|\mathbf{B_{LLL}}\|_1$, un paramètre ρ temporaire à des fins de génération.
2. On calcule $c \in \mathbb{Z}/p\mathbb{Z}$ avec $C(\gamma) \equiv c \pmod p$.
3. En utilisant l'algorithme 5 de [19], on calcule $\overline{C}(X)$, un polynôme représentant c dans notre PMNS temporaire. Cela implique $\|\overline{C}\|_\infty \leqslant \frac{1}{2}\|\mathbf{B}_{LLL}\|_1 + 1$, comme démontré dans [19, Proposition 1].
4. Puisque $C(\gamma) \equiv \overline{C}(\gamma) \equiv c \pmod p$, on pose $M(X) = C(X) - \overline{C}(X)$ et nous avons bien $M(\gamma) \equiv 0 \pmod p$. De plus, par construction nous avons que $\forall i, \frac{1}{2}\rho' \leqslant m_i \leqslant \frac{3}{2}\rho' + 2$. Ainsi, tous les coefficients de M sont positifs.

Ce M n'est pas forcément inversible. Toutefois, dans le cas où λ est pair, il est démontré dans [15, Proposition 11], qu'au moins une des lignes de la base réduite $\mathbf{B}_{LLL}$ possède un premier coefficient impair. Par ailleurs, comme stipulé dans [15, Proposition 8], cette propriété est suffisante, lorsque λ est pair, pour garantir l'inversibilité modulo E, modulo ϕ du polynôme associé à cette ligne. Nous proposons donc la construction suivante.

Puisque au moins une des lignes de $\mathbf{B}_{LLL}$ possède un coefficient impair en première position, notons l_k cette ligne et $L_k(X) = \pi_n(l_k)$. Puisqu'aucun coefficient de $\mathbf{B}_{LLL}$ n'est plus grand que $\|\mathbf{B}_{LLL}\|_1$ (qui est notre ρ' temporaire) par définition de la norme 1 d'une matrice, alors $2 \times M(X) - L_k(X)$ a tous ses coefficients positifs et le premier coefficient impair. En effet, comme noté plus haut, $\forall i, \frac{1}{2}\rho' \leqslant m_i \leqslant \frac{3}{2}\rho' + 2$. Ainsi, $\forall i, \rho' \leqslant 2 \times m_i \leqslant 3\rho' + 4$. Donc, soustraire $L_k(X)$ à $2 \times M(X)$ mènera à des coefficients positifs. De plus, puisque le premier coefficient de $L_k(X)$ est impair, alors le premier coefficient du résultat de $2 \times M(X) - R_k(X)$ sera toujours impair. Ainsi, ce nouveau polynôme sera inversible positif. Puisque nous soustrayons un élément de $\mathfrak{L}$ à un autre élément de $\mathfrak{L}$, le résultat sera évidemment dans $\mathfrak{L}$.

Le problème de cette heuristique est l'agrandissement des bornes. En effet, $\|2 \times M(X) - L_k(X)\|_\infty \leqslant 4\rho' + 4$, ce qui est beaucoup plus grand en termes de norme ∞ qu'une ligne de $\mathbf{B}_{LLL}$. Toutefois, cette heuristique a pour but de trouver systématiquement un polynôme inversible positif dans le cas $E(X) = X^n - \lambda$ avec λ pair, ce qui est bel et bien achevé ici malgré l'augmentation en termes de norme.

2.6 Opérations de conversion

On adapte ici l'algorithme 14, page 39, pour tenir compte des autres domaines potentiels dans lesquels un élément d'un PMNS peut se trouver.

Algorithme 22 Conversion d'un élément du PMNS vers un entier en représentation binaire classique

Require: $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS, $C \in \mathcal{B}$ avec $(c_0, c_1, \ldots, c_{n-1})$ ses coefficients, $\phi \in \mathbb{N}^*$, $g_0, g_1, \ldots, g_{n-1} \in (\mathbb{Z}/p\mathbb{Z})^n$ pré-calculés tels que $\forall i \in [\![0, n-1]\!]$, $g_i \equiv \gamma^i \pmod p$, $RedMethod \in \{18, 19, 20\}$ correspondant au numéro de l'algorithme de réduction interne utilisé que l'on notera **RedInt**.

Ensure: $c \in \mathbb{Z}/p\mathbb{Z}$ tel que $c \equiv C(\gamma) \pmod p$ pour RedMethod $= 19$ ou $c \equiv C(\gamma)\phi^{-1} \pmod p$ pour RedMethod $= 18$ ou $c \equiv C(\gamma)\phi^{-2} \pmod p$ pour RedMethod $= 20$.

1: **if** $RedMethod \neq 19$ **then**
2: $\quad C \leftarrow$ **RedInt**(C) # On sort du domaine associé (voir remarques 24 et 26)
3: **end if**
4: **if** $E(X) = e_n X^n + \cdots + e_1 X + e_0$ avec $|e_n| > 1$ **then** # E non unitaire
5: $\quad$ **if** $\forall i \in [\![2, n-1]\!]$, e_n divise e_i **then**
6: $\quad\quad C \leftarrow e_n C$ # On sort du domaine associé (voir remarque 19)
7: $\quad$ **else**
8: $\quad\quad C \leftarrow (e_n)^{n-1} C$ # On sort du domaine associé (voir remarque 16)
9: $\quad$ **end if**
10: **end if**
11: $c \leftarrow c_0$
12: **for** $i = 1 \ldots n-1$ **do**
13: $\quad c \leftarrow c + c_i g_i$
14: **end for**
15: $c \leftarrow c \bmod p$
16: **return** c

Lemme 2.6.1. *La réduction modulaire à l'étape 15 de l'algorithme 22 peut être effectuée en au plus* $\lceil \log_2(nz\rho) \rceil \lceil \log_\sigma(p) \rceil$ *soustractions élémentaires, avec* $z = 1$ *pour* E *unitaire,* $z = e_n$ *pour* E *non unitaire et tel que* e_n *divise tous les* e_i *pour tout* $i \in [\![2, n-1]\!]$ *et* $z = (e_n)^{n-1}$ *sinon.*

Démonstration. Puisque $\|C\|_\infty < \rho$ en entrée, à l'étape 11 on a $\|C\|_\infty < z\rho$ avec $z = 1$ pour E unitaire, $z = e_n$ pour E non unitaire et tel que e_n divise tous les e_i pour tout $i \in [\![2, n-1]\!]$ et $z = (e_n)^{n-1}$ sinon. Par conséquent, on a aussi $c < z\rho$ à l'étape 11.

Ensuite, chaque tour de boucle ajoute un c_i borné par $z\rho$ multiplié par g_i dans $\mathbb{Z}/p\mathbb{Z}$ donc borné par p. On aura donc $c < (n-1)z\rho p + z\rho$ à la fin de la boucle. On note que $nz\rho p > (n-1)z\rho p + z\rho$ car $z\rho p > z\rho$. On peut donc effectuer la réduction modulaire à l'étape 15 de l'algorithme 22 en au plus $\lceil \log_2(nz\rho) \rceil$ soustractions par p, c'est-à-dire $\lceil \log_2(nz\rho) \rceil \lceil \log_\sigma(p) \rceil$ soustractions élémentaires. □

Remarque 33. La raison pour laquelle cette réduction peut être effectuée en $\lceil \log_2(nz\rho) \rceil \lceil \log_\sigma(p) \rceil$ soustractions élémentaires et non $nz\rho \lceil \log_\sigma(p) \rceil$ tient au fait que l'on peut pré-calculer les $2^i p$ pour i allant de 1 à $\lceil \log_2(nz\rho) \rceil$ afin de réduire le nombre de soustractions nécessaires. En effet, on pourra alors annuler les bits un par un de façon similaire à la méthode du paysan russe (algorithme 4, page 11).

Ensuite, on considère la conversion dans l'autre sens, d'un élément de $\mathbb{Z}/p\mathbb{Z}$ vers un élément du PMNS. En particulier, l'algorithme 13 page 39, fait usage d'une décomposition en base ρ, ce qui encourage à prendre ρ comme une puissance de 2. Mais on peut faire autrement, comme présenté dans [18]. Pour commencer, nous avons besoin de l'algorithme suivant, adapté légèrement de [19, Algorithme 5] (lui-même adapté de [18, Algorithme 8]). Cet algorithme effectue une conversion exacte dans le PMNS, c'est-à-dire que pour un entier a, il renvoie un polynôme $A(X)$ tel que $A(\gamma) \equiv a \pmod p$ et non pas tel que $A(\gamma) \equiv a\phi \mod p$.

Algorithme 23 Conversion exacte d'un entier en représentation classique vers un élément d'un PMNS [19]

Require: $a \in \mathbb{Z}/p\mathbb{Z}$, $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS, $\mathcal{G}$ une base courte d'un sous-réseau de $\mathfrak{L}$ avec $\rho > \frac{1}{2}\|\mathcal{G}\|_1 + 1$ et $\phi \in \mathbb{N}^*$.
Ensure: $A \in \mathcal{B}$ avec $A(\gamma) \equiv a \pmod p$ et $\|A\|_\infty \leqslant \frac{1}{2}\|\mathcal{G}\|_1$.
1: $A \leftarrow a \times \phi^{n+1} \pmod p$ # polynôme de degré 0
2: **for** $i = 0 \dots n$ **do**
3: $\quad A \leftarrow$ **GMont-like**(A)
4: **end for**
5: **return** A

GMont-like ici fait référence à l'algorithme 18 page 59. L'algorithme 23 est essentiellement utile en raison de la proposition suivante :

Proposition 2.6.1. *Le polynôme A en sortie de l'algorithme 23 est bien tel que $A(\gamma) \equiv a \pmod p$ et si $p \leqslant \frac{2(\phi^{n+2} - \phi^{n+1}) - (\phi^{n+1} - \phi)\|\mathcal{G}\|_1}{2(\phi - 1)}$, alors $\|A\|_\infty \leqslant \frac{1}{2}\|\mathcal{G}\|_1$.*

Démonstration. Pour commencer on a $A(\gamma) \equiv a \times \phi^{n+1}$ avant l'exécution de la boucle. La boucle consiste à appliquer **GMont-like** $n+1$ fois, l'algorithme retournant un polynôme dont l'évaluation en γ a été multipliée par ϕ^{-1}. En sortie de boucle on aura donc bien $A(\gamma) \equiv a\phi^{n+1}\phi^{-(n+1)} \equiv a \pmod p$.

Ensuite, on a $\|A\|_\infty < p$ en entrée de la boucle. Puis, à l'étape 2 de l'algorithme 18, on peut effectuer la réduction modulaire afin d'obtenir $\|q\|_\infty \leqslant \frac{1}{2}\phi$ comme noté dans la preuve de la proposition 2.4.2. Donc après un tour de boucle on a

$$\|A\|_\infty < \frac{1}{\phi}(p + \frac{\phi}{2}\|\mathcal{G}\|_1) = \frac{1}{\phi}p + \frac{1}{2}\|\mathcal{G}\|_1.$$

Après un deuxième tour de boucle on aura

$$\|A\|_\infty < \frac{1}{\phi}(\frac{1}{\phi}p + \frac{1}{2}\|\mathcal{G}\|_1 + \frac{\phi}{2}\|\mathcal{G}\|_1) = \frac{1}{\phi^2}p + \frac{1}{2}\|\mathcal{G}\|_1 + \frac{1}{2\phi}\|\mathcal{G}\|_1.$$

Au troisième tour on aura

$$\|A\|_\infty < \frac{1}{\phi}(\frac{1}{\phi^2}p + \frac{1}{2}\|\mathcal{G}\|_1 + \frac{1}{2\phi}\|\mathcal{G}\|_1 + \frac{\phi}{2}\|\mathcal{G}\|_1) = \frac{1}{\phi^3}p + \frac{1}{2}\|\mathcal{G}\|_1 + \frac{1}{2\phi}\|\mathcal{G}\|_1 + \frac{1}{2\phi^2}\|\mathcal{G}\|_1.$$

On aura donc, après les $n+1$ tours de boucle :

$$\|A\|_\infty < \frac{p}{\phi^{n+1}} + \sum_{i=0}^{n} \frac{1}{2\phi^i}\|\mathcal{G}\|_1$$

$$\iff \|A\|_\infty < \frac{p}{\phi^{n+1}} + \frac{1}{2}\|\mathcal{G}\|_1 \frac{1 - \frac{1}{\phi^{n+1}}}{1 - \frac{1}{\phi}}$$

$$\iff \|A\|_\infty < \frac{p}{\phi^{n+1}} + \frac{1}{2}\|\mathcal{G}\|_1 \frac{\phi^{n+2} - \phi}{\phi^{n+2} - \phi^{n+1}}$$

$$\iff \|A\|_\infty < \frac{p}{\phi^{n+1}} + \frac{1}{2}\|\mathcal{G}\|_1 \left(1 + \frac{\phi^{n+1} - \phi}{\phi^{n+2} - \phi^{n+1}}\right)$$

$$\iff \|A\|_\infty < \frac{1}{2}\|\mathcal{G}\|_1 + \frac{p}{\phi^{n+1}} + \frac{(\phi^{n+1}-\phi)\|\mathcal{G}\|_1}{2(\phi^{n+2}-\phi^{n+1})}$$

$$\iff \|A\|_\infty < \frac{1}{2}\|\mathcal{G}\|_1 + \frac{2p(\phi-1)+(\phi^{n+1}-\phi)\|\mathcal{G}\|_1}{2(\phi^{n+2}-\phi^{n+1})}$$

On a supposé $p < \frac{2(\phi^{n+2}-\phi^{n+1})-(\phi^{n+1}-\phi)\|\mathcal{G}\|_1}{2(\phi-1)}$ donc on a bien $\|A\|_\infty < \frac{1}{2}\|\mathcal{G}\|_1 + 1$ ou encore :

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

□

Remarque 34. Cet algorithme de conversion prend n'importe quel élément de $\mathbb{Z}/p\mathbb{Z}$ en entrée et retourne un représentant dans le PMNS avec sa norme infinie bornée par $\frac{1}{2}\|\mathcal{G}\|_1$, ce qui est la borne théorique. Cela confirme donc que tout élément peut bien être représenté en prenant ces bornes, ce qui est une autre façon de montrer que $\rho > \frac{1}{2}\|\mathcal{G}\|_1$ est bien une condition suffisante pour garantir un représentant dans le PMNS pour chaque élément de $\mathbb{Z}/p\mathbb{Z}$.

État donné qu'en pratique on a $p < \phi^n$, la condition requise sur p par la proposition précédente est toujours vérifiée. Cet algorithme est surtout utile pour les pré-calculs pour avoir les bornes les plus petites possibles. En particulier, cela est nécessaire pour les pré-calculs de l'algorithme suivant, adapté légèrement de [19, Algorithme 6]. Dans celui-ci, nous utilisons des valeurs P_i pré-calculées en utilisant l'algorithme 23 afin de garantir que $\|P_i\|_\infty \leqslant \frac{\|\mathcal{G}\|_1}{2}$.

Algorithme 24 Conversion rapide d'un entier en représentation classique vers un élément d'un PMNS

Require: $a \in \mathbb{Z}/p\mathbb{Z}$, $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS, avec $E(X) = e_n X^n + \cdots + e_1 X + e_0$, $z = (e_n)^{n-1}$ si $\forall i \in [\![0, n-1]\!]$, e_n divise e_i sinon $z = e_n$, Θ la plus petite puissance de 2 telle que $\Theta^n > p$ et $P_0, P_1, \ldots, P_{n-1} \in \mathcal{B} \times \mathcal{B} \times \cdots \times \mathcal{B}$ tels que $\forall i \in [\![0, n-1]\!]$, $P_i(\gamma) \equiv z^{-1}\Theta^i\phi^2 \pmod p$ et $\|P_i\|_\infty \leqslant \frac{\|\mathcal{G}\|_1}{2}$.
Ensure: $A \in \mathcal{B}$ tel que $A(\gamma) \equiv z^{-1}a\phi \pmod p$
1: $t \leftarrow (t_{n-1}, \ldots, t_0)$ # décomposition de a en base Θ
2: $U \leftarrow \sum_{i=0}^{n-1} t_i P_i(X)$
3: $A \leftarrow$ **GMont-like**(U)
4: **return** A

Ici aussi, **GMont-like** fait référence à l'algorithme 18. Cet algorithme permet une conversion rapide, ne demandant qu'une seule réduction interne, comparée à l'algorithme précédent qui en nécessitait plusieurs, en contrepartie de moins bonnes bornes en sortie. De plus, les anciennes opérations de conversion imposaient de prendre ρ comme une puissance de 2. Cette nouvelle méthode rend le paramètre ρ complètement indépendant vis-à-vis du processus de conversion en introduisant le paramètre Θ qui peut lui être pris comme une puissance de 2 sans introduire d'effets de bord.

Proposition 2.6.2. *L'algorithme 24 renvoie bien un polynôme A en sortie tel que $A(\gamma) \equiv z^{-1}a\phi \pmod p$ et de plus si $\Theta < \frac{2w(\|\mathcal{G}\|_1-2)(\|\mathcal{G}\|_1-4)}{n\|\mathcal{G}\|_1}$, alors $\|A\|_\infty < \rho$.*

Démonstration. Pour commencer, t est la décomposition de a en base Θ, ou autrement dit, $a = \sum_{i=0}^{n-1} t_i\Theta^i$.

On aura donc $U(\gamma) \equiv \sum_{i=0}^{n-1} t_i \Theta^i z^{-1}\phi^2 \equiv z^{-1}a\phi^2 \pmod p$. Puisque **GMont-like** effectue une division par ϕ, on aura bien $A(\gamma) \equiv z^{-1}a\phi \pmod p$.

Ensuite, par construction on a :
— $\forall i, \|P_i\|_\infty \leqslant \frac{\|\mathcal{G}\|_1}{2}$
— $\|t\|_\infty \leqslant \Theta - 1$
Donc, $\forall i \in [\![0, n-1]\!], \|t_i P_i\|_\infty \leqslant (\Theta-1)(\frac{\|\mathcal{G}\|_1}{2})$. Il en découle que $\|U\|_\infty \leqslant n(\Theta-1)(\frac{\|\mathcal{G}\|_1}{2})$.

Après application de **GMont-like** on obtient $\|A\|_\infty \leqslant \frac{n(\Theta-1)(\frac{1}{2}\|\mathcal{G}\|_1)+\frac{\phi}{2}\|\mathcal{G}\|_1}{\phi}$ ce qui nous donne :

$$\|A\|_\infty \leqslant \frac{n(\Theta-1)(\frac{1}{2}\|\mathcal{G}\|_1)}{\phi} + \frac{1}{2}\|\mathcal{G}\|_1. \tag{2.15}$$

Comme on peut le voir dans les tableaux 2.1 et 2.2, les bornes sur ϕ et ρ pour l'algorithme 18 sont $\phi > 2w(\rho-1)$ et $\rho \geqslant \|\mathcal{G}\|_1 - 1$. Donc, on en déduit :

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

et qu'une condition suffisante pour vérifier $\|A\|_\infty < \rho$ est d'imposer :

$$\frac{n(\Theta-1)(\frac{1}{2}\|\mathcal{G}\|_1)}{2w(\|\mathcal{G}\|_1-2)} + \frac{1}{2}\|\mathcal{G}\|_1 \leqslant \|\mathcal{G}\|_1 - 2 < \rho$$

Ce qui nous donne donc :

$$\begin{aligned}
&\frac{1}{2}\|\mathcal{G}\|_1 - 2 \geqslant \frac{n(\Theta-1)(\frac{1}{2}\|\mathcal{G}\|_1)}{2w(\|\mathcal{G}\|_1-2)} \\
\iff &\|\mathcal{G}\|_1 - 4 \geqslant \frac{n(\Theta-1)\|\mathcal{G}\|_1}{2w(\|\mathcal{G}\|_1-2)} \\
\iff &2w(\|\mathcal{G}\|_1-2)(\|\mathcal{G}\|_1-4) \geqslant n(\Theta-1)\|\mathcal{G}\|_1 \\
\iff &\frac{2w(\|\mathcal{G}\|_1-2)(\|\mathcal{G}\|_1-4)}{n\|\mathcal{G}\|_1} > \Theta
\end{aligned}$$

Donc pour résumer si

$$\Theta < \frac{2w(\|\mathcal{G}\|_1-2)(\|\mathcal{G}\|_1-4)}{n\|\mathcal{G}\|_1} \tag{2.16}$$

alors $\|A\|_\infty < \rho$ pour tout $\rho \geqslant \|\mathcal{G}\|_1 - 1$. □

Il est naturel de s'interroger sur les restrictions supplémentaires que l'on devra mettre en place afin de satisfaire les conditions de l'équation 2.16. Nous donnons donc une condition suffisante sur p et n pour que ce soit bien le cas.

Proposition 2.6.3. *Soit $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS et $\mathcal{G}$ une base de*

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

L'équation (2.16) est vérifiée si p est tel que $2^{\left\lceil \frac{\log_2(p)}{n} \right\rceil} > 12$.

Démonstration. Pour commencer, on réarrange un peu l'équation (2.16).

$$\Theta < \frac{2w(\|\mathcal{G}\|_1 - 2)(\|\mathcal{G}\|_1 - 4)}{n\|\mathcal{G}\|_1}$$

$$\iff \Theta < \frac{2w}{n}(\|\mathcal{G}\|_1 - 6) + \frac{16w}{n\|\mathcal{G}\|_1}$$

Puisque $\frac{16w}{n\|\mathcal{G}\|_1} > 0$, tant que $\Theta < \frac{2w}{n}(\|\mathcal{G}\|_1 - 6)$, l'équation sera bien vérifiée.

D'après la remarque 5, on a toujours $\det(\mathcal{G}) = p$. Si on note $c_0, c_1, \ldots, c_{n-1}$ les colonnes de $\mathcal{G}$, l'inégalité de Hadamard [28] nous donne :

$$\prod_{i=0}^{n-1} \|c_i\|_2 \geqslant \det(\mathcal{G})$$

Puisque pour tout vecteur v, $\|v\|_1 \geqslant \|v\|_2$, on en déduit :

$$\prod_{i=0}^{n-1} \|c_i\|_1 \geqslant p$$

Par conséquent :

$$\max_i(\|c_i\|_1)^n \geqslant p$$

ou encore

$$(\|\mathcal{G}\|_1)^n \geqslant p$$

d'où

$$\|\mathcal{G}\|_1 \geqslant p^{\frac{1}{n}}$$

Ensuite, puisque Θ est la plus petite puissance de 2 telle que $\Theta^n > p$, il en découle $\Theta = 2^{\left\lceil \frac{\log_2(p)}{n} \right\rceil}$. L'équation (2.16) sera donc vérifiée si :

$$\frac{2w}{n}(p^{\frac{1}{n}} - 6) > 2^{\left\lceil \frac{\log_2(p)}{n} \right\rceil}$$

$$\iff \frac{w}{n}(2p^{\frac{1}{n}} - 12) > 2^{\left\lceil \frac{\log_2(p)}{n} \right\rceil}$$

Comme l'on a pu le voir dans la proposition 2.3.5, $w \geqslant n$ pour tout E de degré n. Ainsi, si

$$2p^{\frac{1}{n}} - 12 > 2^{\left\lceil \frac{\log_2(p)}{n} \right\rceil}$$

alors l'inégalité précédente est vérifiée. Or

$$2^{\frac{\log_2(p)}{n}+1} - 12 > 2^{\left\lceil \frac{\log_2(p)}{n} \right\rceil}$$

$$\iff 2^{\frac{\log_2(p)}{n}+1} - 2^{\left\lceil \frac{\log_2(p)}{n} \right\rceil} > 12$$

$$\iff 2^{\frac{\frac{\log_2(p)}{n}+1}{\left\lceil\frac{\log_2(p)}{n}\right\rceil}} - 1 > \frac{12}{2^{\left\lceil\frac{\log_2(p)}{n}\right\rceil}}$$

On note qu'en particulier on a toujours $\frac{\log_2(p)}{n} + 1 > \left\lceil\frac{\log_2(p)}{n}\right\rceil$ donc $2^{\frac{\frac{\log_2(p)}{n}+1}{\left\lceil\frac{\log_2(p)}{n}\right\rceil}} > 2^1$. De plus, on a supposé p tel que $2^{\left\lceil\frac{\log_2(p)}{n}\right\rceil} > 12$ donc $\frac{12}{2^{\left\lceil\frac{\log_2(p)}{n}\right\rceil}} < 1$. Ce qui nous donne

$$2^{\frac{\frac{\log_2(p)}{n}+1}{\left\lceil\frac{\log_2(p)}{n}\right\rceil}} - 1 > 2^1 - 1 = 1 > \frac{12}{2^{\left\lceil\frac{\log_2(p)}{n}\right\rceil}}$$

L'équation (2.16) est donc bien vérifiée sous ces conditions. □

On note que cela se résume plus ou moins à vérifier que $p > 12^n$, ce qui est virtuellement toujours le cas.

2.7 Test d'égalité

L'algorithme 15 présenté en section 1.5.6 nécessite de choisir les paramètres de telle sorte que

$$\phi > \lceil 2w(\|\mathcal{G}\|_1)^2\|\mathcal{G}^{-1}\|_1\rceil$$

avec $\mathcal{G}$ une base courte d'un sous-réseau de $\mathfrak{L}$. En prenant en compte les bornes présentées en sections 2.4.4 et 2.4.5, nous pouvons en temps normal choisir

$$\phi > 2w(\|\mathcal{G}\|_1 - 1)$$

lorsque nous utilisons la méthode de réduction interne Montgomery-like. Ainsi, le fait d'utiliser l'algorithme 15 demande un élargissement de la borne sur ϕ d'un facteur environ $\lceil\|\mathcal{G}\|_1\|\mathcal{G}^{-1}\|_1\rceil$ ($2w$ et $\|\mathcal{G}\|_1$ étant des entiers).

La seule borne donnée dans [18] sur cette quantité est $\lceil\|\mathcal{G}\|_1\|\mathcal{G}^{-1}\|_1\rceil \geqslant 1$, il est donc difficile d'estimer le coût inhérent à cette méthode. Nous avons montré dans la proposition 2.5.2 que pour certains PMNS, $\|\mathcal{G}\|_1 \leqslant (n!p)^{\frac{1}{n}}$. Nous n'avons à l'heure actuelle pas de résultats sur une estimation de $\|\mathcal{G}\|_1$ pour un PMNS quelconque mais, en prenant en compte le lemme 2.5.1, il paraît raisonnable de s'attendre à ce que $\|\mathcal{G}\|_1 \approx (n!p)^{\frac{1}{n}}$. De façon similaire, on pourrait aussi s'attendre à ce que $\|\mathcal{G}^{-1}\|_1 \approx (n!\frac{1}{p})^{\frac{1}{n}}$ puisque $\det(\mathcal{G}^{-1}) = \frac{1}{p}$. Ainsi, nous pouvons poser comme heuristique que $\lceil\|\mathcal{G}\|_1\|\mathcal{G}^{-1}\|_1\rceil \approx (n!)^{\frac{2}{n}}$. Pour appuyer cette estimation, nous avons effectué des mesures sur un tirage aléatoire de PMNS dont les résultats se trouvent dans le tableau 2.10. Au cours de ce tirage, la base courte $\mathcal{G}$ est obtenue en appliquant l'algorithme LLL à la base de l'équation (1.1).

Taille de p en bits	256	512	768	1024
n utilisé	5	9	14	19
Moyenne de $\left\lfloor (n!p)^{\frac{1}{n}} \right\rceil$	6361862521831251	535958720685364648	193276485720075031	130789289731748217
Moyenne pour $\|\mathcal{G}\|_1$	6086952794385240	467803223897252584	165398937341399013	112765015876549604
Moyenne pour $\frac{\left\lfloor (n!\det(\mathcal{G}))^{\frac{1}{n}} \right\rceil}{\|\mathcal{G}\|_1}$	1.04516	1.14569	1.16855	1.15984
$(n!)^{\frac{2}{n}}$	6.7869	17.199	36.552	62.868
Moyenne pour $\lceil \|\mathcal{G}\|_1 \|\mathcal{G}^{-1}\|_1 \rceil$	7.0095	15.988	34.609	70.723
Min trouvé pour $\lceil \|\mathcal{G}\|_1 \|\mathcal{G}^{-1}\|_1 \rceil$	2	3	11	21
Max trouvé pour $\lceil \|\mathcal{G}\|_1 \|\mathcal{G}^{-1}\|_1 \rceil$	58	50	141	315

TABLE 2.10 – Tableau de comparaison de mesures de $\|\mathcal{G}\|_1$ et $\|\mathcal{G}^{-1}\|_1$ pour un tirage aléatoire uniformément distribué de un million de PMNS construits avec $E(X) = X^n - \lambda$ pour chaque taille considérée.

Ce facteur grandissant devient très problématique sur les grands entiers car cela oblige d'augmenter le paramètre n afin de satisfaire les conditions sur ϕ. Cela motive l'exploration d'autres méthodes de test d'égalité plus efficaces sans agrandir les bornes.

Pour rappel, la complexité de l'algorithme 15 est de 2 opérations de multiplication vecteur-matrice et $4n$ additions. En particulier puisqu'il faut utiliser la base $\mathcal{G}$ du réseau entier et non la base d'un sous-réseau, ces opérations vecteur-matrice ne seront pas à priori sous-quadratique en n. Nous allons ici considérer des alternatives qui ne changent pas les bornes.

Algorithme 25 Test d'égalité réduit

Require: $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS, $A, B \in \mathcal{B} \times \mathcal{B}$ donc $\|A\|_\infty < \rho$ et $\|B\|_\infty < \rho$, $\phi \in \mathbb{N}^*$ et $\mathcal{G}$ une base courte de $\mathfrak{L}$. $\mathcal{G}'$ pré-calculée telle que $\mathcal{G}' = \left\lfloor \phi \mathcal{G}^{-1} \right\rceil$.
Ensure: True si $A(\gamma) \equiv B(\gamma) \pmod p$ sinon False.

$C \leftarrow \nu_n(A(X) - B(X))$
$S \leftarrow \left\lfloor \frac{1}{\phi} C \cdot \mathcal{G}' \right\rceil \cdot \mathcal{G}$
$i \leftarrow 0$
while $i < n$ **and** $s_i = c_i$ **do**
    $i \leftarrow i + 1$
end while
if $i = n$ **then**
    return True
else
    return False
end if

Cet algorithme s'inspire de la réduction interne Barrett-like (algorithme 19) pour un cas où les coefficients du polynôme en entrée sont bornés par $2(\rho-1)$. Le coût de cet algorithme est de 2 opérations de multiplication vecteur-matrice (à priori quadratiques en n), n soustractions et au plus $2n + 1$ comparaisons. On note que l'on peut également tester les coefficients de S immédiatement au fur et à mesure qu'ils sont calculés afin d'améliorer la complexité dans le meilleur cas (la non-égalité).

Proposition 2.7.1. *Si $\phi > 2n(\rho-1)$, alors l'algorithme 25 retourne bien True lorsque $A(\gamma) \equiv B(\gamma) \pmod p$ et False sinon.*

Démonstration. Puisque $\mathcal{G}' = \left\lfloor \phi \mathcal{G}^{-1} \right\rceil$, il existe $\varepsilon \in \mathcal{M}_n(\mathbb{R})$ tel que $\forall i, j \in [\![0, n-1]\!] \times [\![0, n-1]\!]$, $|\varepsilon_{i,j}| \leqslant \frac{1}{2}$ et $\mathcal{G}' = \phi \mathcal{G}^{-1} + \varepsilon$. On note que l'on a alors $\|\varepsilon\|_1 \leqslant \frac{n}{2}$. Ainsi,

$$\begin{aligned}\left\lfloor \frac{1}{\phi} C \cdot \mathcal{G}' \right\rceil &= \left\lfloor \frac{1}{\phi} \left(C \cdot (\phi \mathcal{G}^{-1} + \varepsilon) \right) \right\rceil \\ &= \left\lfloor C \cdot \mathcal{G}^{-1} + \frac{1}{\phi} C \cdot \varepsilon \right\rceil.\end{aligned}$$

Remarquons que

$$\| \frac{1}{\phi} C \cdot \varepsilon \|_\infty \leqslant \frac{n(\rho - 1)}{\phi}$$

car $\|C\|_\infty \leqslant 2(\rho - 1)$. Or, nous avons supposé $\phi > 2n(\rho - 1)$. Donc, on en déduit que

$$\| \frac{1}{\phi} C \cdot \varepsilon \|_\infty < \frac{1}{2}.$$

Si $A(\gamma) \equiv B(\gamma) \pmod p$, alors $\pi_n(C)(\gamma) \equiv 0 \pmod p$ et $C \in \mathfrak{L}$. Ainsi, $\exists v \in \mathbb{Z}^n$ tel que $v \cdot \mathcal{G} = C$. En particulier $v = C \cdot \mathcal{G}^{-1}$. Donc

$$\left\lfloor C \cdot \mathcal{G}^{-1} + \frac{1}{\phi} C \cdot \varepsilon \right\rceil = C \cdot \mathcal{G}^{-1} = v.$$

La dernière étape consiste à comparer C avec $\left\lfloor C \cdot \mathcal{G}^{-1} + \frac{1}{\phi} C \cdot \varepsilon \right\rceil \cdot \mathcal{G}$ coefficient par coefficient et donc l'algorithme renvoie bien True lorsque $A(\gamma) \equiv B(\gamma) \pmod p$ et False sinon. □

Contrairement à l'algorithme 15, l'algorithme 25 requiert en entrée des polynômes dont la norme infinie est bornée par ρ. L'algorithme 15 accepte des paramètres bornés par $\frac{1}{2} w(\rho - 1)^2$, ce qui sous-entend qu'il est parfois possible d'effectuer un test d'égalité après une multiplication polynomiale suivie d'une réduction externe. Pour traiter ce cas particulier, on pourrait d'abord effectuer une réduction interne afin de se ramener à un cas avec des coefficients bornés par ρ mais cela rajoute évidemment le coût de la réduction interne au test d'égalité. Si cette réduction allait être effectuée quoi qu'il arrive, on peut la considérer "gratuite" pour le test d'égalité en lui-même. Dans un cas marginal où on voudrait vraiment effectuer un test d'égalité avant réduction interne, on présente l'algorithme suivant qui, encore une fois, ne nécessite aucune modification des bornes.

Algorithme 26 Test d'égalité étendu

Require: $A, B \in \mathbb{Z}_{n-1}[X] \times \mathbb{Z}_{n-1}[X]$ avec $\|A\|_\infty < w(\rho - 1)^2$ et $\|B\|_\infty < w(\rho - 1)^2$, $\phi \in \mathbb{N}^*$ et $\mathcal{G}$ une base courte de $\mathfrak{L}$ inversible modulo ϕ^2.
Ensure: $S = 0$ si et seulement si $A(\gamma) \equiv B(\gamma) \pmod p$.
1: $C \leftarrow A(X) - B(X)$
2: $S \leftarrow$ **Plantard-like**(C)
3: **return** S

Il s'agit ici d'une adaptation de l'algorithme 15 utilisant la réduction Plantard-like (algorithme 20). Le coût total est de n soustractions, puis les 2 opérations de multiplication vecteur-matrice et n additions de l'algorithme 20 pour un total de $2n$ additions et soustractions. En considérant que l'algorithme de réduction Plantard-like est plus lent que celui utilisé dans l'algorithme 15, qui est le Montgomery-like, cet algorithme est plus lent. En contrepartie, les bornes restent inchangées et les paramètres en entrée sont bornés par

$w(\rho-1)^2$ et non $\frac{1}{2}w(\rho-1)^2$ comme dans l'algorithme 15.

Proposition 2.7.2. *Si $2w(\rho-1)^2 < \phi^2$, alors l'algorithme 26 retourne bien 0 si et seulement si $A(\gamma) \equiv B(\gamma) \pmod{p}$.*

Démonstration. Si $A(\gamma) \equiv B(\gamma) \pmod{p}$ alors $\exists v \in \mathbb{Z}^n$ tel que $v \cdot \mathcal{G} = \nu_n(A(X) - B(X))$. On effectue donc tout d'abord la soustraction de A par B avant d'effectuer l'algorithme 20 avec $C(X) = A(X) - B(X)$ comme paramètre. Ainsi, lorsque l'on effectue le calcul de $Q = -\nu_n(C) \cdot \mathcal{G}^{-1}$ à l'étape 2 de l'algorithme 20, on s'assure que $Q = -v$. En particulier, $\|A(X) - B(X)\|_\infty \leqslant 2w(\rho-1)^2$ et donc par hypothèse $\|A(X) - B(X)\|_\infty < \phi^2$. Ainsi, les deux divisions successives par ϕ permettent de garantir que la sortie est bien le polynôme nul. Si $A(\gamma) \not\equiv B(\gamma) \pmod{p}$ alors la réduction interne renvoie forcément un polynôme non nul (cf. proposition 2.4.4). □

Chapitre 3

PMNS sur les très grands entiers

Notations :
- $\mathbb{Z}_m[X]$ le $\mathbb{Z}$-module des polynômes à coefficients entiers de degré au plus m.
- $\nu_n : \mathbb{Z}_{n-1}[X] \to \mathbb{Z}^n$ l'isomorphisme tel que $\nu_n(a_0 + a_1X + \cdots + a_{n-1}X^{n-1}) = (a_0, a_1, \ldots, a_{n-1})$ pour un n donné.
- $\pi_n : \mathbb{Z}^n \to \mathbb{Z}_{n-1}[X]$ l'isomorphisme tel que $\pi_n((a_0, a_1, \ldots, a_{n-1})) = a_0 + a_1X + \cdots + a_{n-1}X^{n-1}$ pour un n donné.
- σ le nombre de valeurs possibles dans un registre mémoire (2^{64} pour la plupart des CPU modernes).
- $M_{i,j}$ pour M une matrice représente le coefficient situé sur la $i$ème ligne et la $j$ème colonne, en partant de 0. Une matrice M de dimension $n \times m$ aura donc des coefficients de $M_{0,0}$ à $M_{n-1,m-1}$.
- $\lfloor a \rceil$ pour $a \in \mathbb{R}$ représente la valeur arrondie de a, équivalente à $\lfloor a + \frac{1}{2} \rfloor$.
- Pour $v = (v_0, v_1, \ldots, v_{n-1}) \in \mathbb{R}^n$, on aura $\lfloor v \rceil = (\lfloor v_0 \rceil, \lfloor v_1 \rceil, \ldots, \lfloor v_{n-1} \rceil)$.
- Pour $M \in \mathcal{M}_n(\mathbb{R})$, on notera $\lfloor M \rceil$ la matrice telle que $\forall i, j \in [\![0, n-1]\!] \times [\![0, n-1]\!]$, $\lfloor M \rceil_{i,j} = \lfloor M_{i,j} \rceil$.
- Un PMNS $\mathcal{B}$ est un tuple (p, n, γ, ρ, E) représentant les entiers de $\mathbb{Z}/p\mathbb{Z}$ par des polynômes sur n coefficients (donc de degré au plus $n-1$). Un entier a est représenté par un polynôme $A(X)$ si $A(\gamma) \equiv a \pmod{p}$. Le paramètre ρ représente la borne sur la taille des coefficients polynomiaux des éléments du PMNS. Pour réduire le degré après multiplication, le polynôme E de degré n qui s'annule en γ modulo p est utilisé.

3.1 Introduction

Les PMNS ont principalement démontré leur efficacité sur des entiers de tailles couramment utilisées en cryptographie sur les courbes elliptiques, c'est-à-dire dans l'intervalle de 256 à 512 bits. Dans une perspective d'extension de leur domaine d'application, il est naturel d'envisager leur adaptation à des entiers de taille supérieure, en particulier pour répondre aux exigences de sécurité de certains cryptosystèmes comme le RSA par exemple. La première étude explorant cette problématique est présentée dans [14], où Didier et al. comparent les performances des systèmes PMNS et RNS pour des tailles variant de 256 à 3251 bits. L'une des observations de ce travail concerne le phénomène d'augmentation du paramètre n en fonction de la taille du module p. On rappelle que la complexité des opérations dans un PMNS dépend principalement du paramètre n car il représente le nombre de registres mémoire nécessaires pour contenir un élément du

PMNS en mémoire. En particulier, l'opération la plus souvent étudiée est la multiplication modulaire de deux éléments qui a une complexité au mieux sous-quadratique en n.

À des fins de génération un certain paramètre n_{opt}, que nous avons introduit en section 1.5.2, avait été défini comme $n_{opt} = \left\lfloor \frac{\log_2(p)}{\log_2(\sigma)} \right\rfloor + 1$. Pour clarifier, aucun PMNS ne peut exister avec $n < n_{opt}$ sans que les coefficients polynomiaux ne débordent d'un registre mémoire. En particulier, on peut construire des PMNS pour des premiers p de 256 bits vérifiant $n = n_{opt}$. On pourrait donc considérer que la borne proposée est optimale. Cependant, Didier et al. constatent que le paramètre n s'éloigne de plus en plus de n_{opt} lorsque l'on génère des PMNS pour des entiers de plus grandes tailles (également remarqué auparavant dans [61, Section 2.4.3]). Par exemple, pour des entiers p de 3251 bits, alors que l'on a $n_{opt} = 51$, on ne trouve en pratique des PMNS qu'à partir de $n = 74$. Cette augmentation est expliquée en partie par les paramètres de construction. En particulier, la construction proposée dans [14] impose :

$$\phi \geqslant 144(2n-1)^2 p^{\frac{1}{n}}$$

avec $\phi = \sigma$ pour $E(X) = X^n - 2$ ou $X^n - X - 1$. Cela est dû à leur choix de paramètre δ, qui représente le nombre d'additions "gratuites" (cf. section 1.5.4). En effet, les PMNS générés dans [14] nécessitent $\delta = 5$ à des fins d'utilisation dans l'échelle de Montgomery pour les courbes elliptiques. Les auteurs concluent ainsi que les PMNS deviennent de moins en moins performants lorsque la taille des entiers considérés augmente, surtout en considérant la multiplication polynomiale "schoolbook" dont la complexité est quadratique en n.

Pour comparaison, si l'on prend $\delta = 0$, la borne présentée dans [14] est

$$\phi \geqslant 4(2n-1)^2 p^{\frac{1}{n}}$$

avec $\phi = \sigma$. Avec les nouvelles bornes établies dans les sections 2.5.1, 2.4.4 et 2.4.5, en considérant la réduction interne Montgomery-like, qui est celle utilisée dans [14], l'entier n doit satisfaire

$$\sigma \geqslant 2(2n-1)((n!p)^{\frac{1}{n}} - 2).$$

En effet, les paramètres doivent vérifier $\phi > 2w(\rho - 1)$, $\rho \geqslant \|\mathcal{G}\|_1 - 1$ avec une borne de $\|\mathcal{G}\|_1 \leqslant (n!p)^{\frac{1}{n}}$ établie pour certains PMNS par la proposition 2.5.2, page 70. De plus, empiriquement, les PMNS que nous trouvons vérifient $\|\mathcal{G}\|_1 \leqslant (n!p)^{\frac{1}{n}}$ en moyenne (cf. tableau 2.10 page 82). Puisque nous ne considérons ici, comme les auteurs de [14], que les polynômes E de la forme $E(X) = X^n - 2$ et $E(X) = X^n - X - 1$, on a alors $w = (2n-1)$ et donc

$$\phi > 2(2n-1)(\|\mathcal{G}\|_1 - 2).$$

Puisque qu'en moyenne les paramètres d'un PMNS construit avec $E(X) = X^n - \lambda$ vérifient

$$2(2n-1)(\|\mathcal{G}\|_1 - 2) \leqslant 2(2n-1)((n!p)^{\frac{1}{n}} - 2),$$

on peut de façon heuristique choisir ϕ tel que

$$\phi \geqslant 2(2n-1)((n!p)^{\frac{1}{n}} - 2)$$

à des fins de génération.

Taille de p	256	512	1024	2048	4096	6144	8192
$\left\lceil \frac{\log_2(p)}{\log_2(\sigma)} \right\rceil$	4	8	16	32	64	96	128
n_{opt} de l'équation (2.14), page 72	5	9	17	34	70	105	141
n tel que $\sigma \geqslant 2(2n-1)((n!p)^{\frac{1}{n}} - 2)$	5	9	19	39	81	125	169
borne sur n dans [14]	5	10	20	42	87	134	183

TABLE 3.1 – Tableau de comparaison entre $\left\lceil \frac{\log_2(p)}{\log_2(\sigma)} \right\rceil$, la nouvelle valeur de n_{opt}, le n satisfaisant $\sigma \geqslant 2(2n-1)((n!p)^{\frac{1}{n}} - 2)$ qui est le n pour la méthode de réduction Montgomery-like avec nos nouvelles bornes et la borne sur n présentée dans [14].

La première ligne du tableau 3.1 représente le nombre de registres mémoire nécessaires en arithmétique modulaire classique, par exemple celle de GMP, pour représenter des entiers de la taille considérée. On peut constater deux phénomènes dans ce tableau. Premièrement, la valeur de n_{opt} s'éloigne de plus en plus du nombre de registres en arithmétique modulaire classique. Deuxièmement, la borne sur n prise en pratique s'éloigne de plus en plus de n_{opt} également. On rappelle que la complexité des opérations dans un PMNS dépend directement du paramètre n. Il semblerait ainsi que les PMNS soient voués à devenir de moins en moins performants plus la taille d'entiers considérés augmente. Premièrement, de façon théorique avec l'augmentation de n_{opt} et deuxièmement de façon pratique avec les valeurs de n s'éloignant de n_{opt}.

Ce chapitre présente une gamme de solutions considérées pour pallier à ces problèmes et maintenir de bonnes performances avec les PMNS même en grande taille.

3.2 PMNS avec coefficients sur deux registres mémoire

La première idée que nous avons considéré avec Méloni et Véron dans [44] a été de relaxer les contraintes sur les bornes en augmentant la taille des coefficients. La plupart des processeurs modernes possèdent des registres mémoire sur 64 bits (donc $\sigma = 2^{64}$) mais certains compilateurs, en particulier GCC, proposent des opérations sur des variables sur 128 bits avec le type `__int128`. Ainsi, en prenant $\phi = \sigma^2$ au lieu de $\phi = \sigma$ comme traditionnellement, le paramètre n sera diminué en conséquence. Cependant, cela sous-entend d'organiser les coefficients polynomiaux sur 2 registres mémoire chacun.

Par exemple, soit le PMNS $\mathcal{B}_{64} = ($
$p = 1594325758232031126655209241620648907941546080653355270884289$,
$n = 4$,
$\gamma = 1295381300537454157680984276194919001807799661331067380711422$,
$\rho = 2497754239110494$,
$E(X) = X^4 - 2)$.
Pour $a = 1317903124572759072261564293267531387704041480804743856488383$,

$$A_{64}(X) = 569291221995662X^3 - 6836718980768X^2 - 521817887111740X + 593326083724649$$

est un représentant de a dans $\mathcal{B}_{64}$, étant donné que $A_{64}(\gamma) \equiv a \pmod{p}$. Pour $\sigma = 2^{64}$, chacun des coeffi-

cients de A_{64} tient sur un seul registre mémoire car $\|A_{64}\|_\infty < \sigma$.

En revanche, pour $\mathcal{B}_{128} = ($
$p = 159432575823203112665520924162064890794154608065335527088428 9$,
$n = 2$,
$\gamma = 147629585337220950400741801734175121240953447498794072480 6053$,
$\rho = 189192391353168009193781303705 6$,
$E(X) = X^2 - 2)$, un PMNS construit sur le même p que $\mathcal{B}_{64}$, un représentant de a serait par exemple

$$A_{128}(X) = -\texttt{0x5210c5d20ae2c60093f335cf8}X - \texttt{0x7328d4deded0ce5decc874ca0} \tag{3.1}$$

avec $\|A_{128}\|_\infty > \sigma$. Ainsi, pour ranger chaque coefficient de A_{128} en mémoire il faudra au moins deux registres. On a ainsi :

$$A_{128}(X) = -(\underbrace{\texttt{0x5210c5d20}}_{\text{premier registre}} \times 2^{64} + \underbrace{\texttt{0xae2c60093f335cf8}}_{\text{deuxième registre}})X - (\underbrace{\texttt{0x7328d4ded}}_{\text{premier registre}} \times 2^{64} + \underbrace{\texttt{0xed0ce5decc874ca0}}_{\text{deuxième registre}}) \tag{3.2}$$

Le type `__int128` permet de laisser au compilateur l'occasion d'organiser en mémoire les registres comme présenté ici de façon transparente.

Étant donné que la multiplication de 2 coefficients de 128 bits se traduit par 4 multiplications (bas-bas, bas-haut, haut-bas, haut-haut) entre des registres de 64 bits, les opérations de complexité en n^2 jusqu'à présent seront alors de complexité $4n^2$ avec un n plus petit. Tant que le paramètre n ainsi réduit du fait de prendre $\phi = \sigma^2$ est au moins deux fois plus petit que le paramètre n engendré par le choix classique $\phi = \sigma$, on peut s'attendre à un gain en performances avec cette nouvelle méthode.

Taille d'entiers	256	512	1024	2048	4096	6144	8192
$\left\lceil \frac{\log_2(p)}{\log_2(\sigma)} \right\rceil$	4	8	16	32	64	96	128
n tel que $2(2n-1)((n!p)^{\frac{1}{n}} - 2) \leqslant \sigma^2$	3	5	9	18	35	53	72
nombre de registres nécessaires	6	10	18	36	72	108	144
n_{opt} de l'équation (2.14), page 72	5	9	17	34	70	105	141
n tel que $\sigma \geqslant 2(2n-1)((n!p)^{\frac{1}{n}} - 2)$	5	9	19	39	81	124	169

TABLE 3.2 – Tableau de comparaison entre $\left\lceil \frac{\log_2(p)}{\log_2(\sigma)} \right\rceil$, n satisfaisant $2(2n-1)((n!p)^{\frac{1}{n}} - 2) \leqslant \sigma^2$ et le nombre de registres mémoire correspondants avec n_{opt} pour des registres mémoire de taille 64 bits et n satisfaisant $2(2n-1)((n!p)^{\frac{1}{n}} - 2) \leqslant \sigma$.

Comme on peut le remarquer dans le tableau 3.2, le nombre de registres mémoire nécessaires pour un élément d'un PMNS donné ainsi obtenu est très proche du n_{opt} théorique. Cela devrait, en théorie, s'accompagner d'une amélioration des performances en conséquence. Malheureusement, l'utilisation de coefficients multi-précisions sous-entend l'ajout d'une propagation de retenue intra-coefficient. Cela mène à un surcoût non négligeable. De plus, l'implémentation présentée en [44] est une implémentation naïve de la multi-précision. En effet, les choix faits mènent à de nombreux passages de retenues qui auraient pu être évités si les coefficients avaient été répartis plus équitablement sur chaque registre.

$\log_2(p)$	256	512	1024	2048	4096	6144	8192
GMP bas niveau	226	524	1739	5423	16864	34007	53241
GMP Montgomery	187	451	1564	4677	**14595**	**29227**	**44769**
GMP Montgomery CIOS	**105**	**340**	**1206**	**4327**	16647	34731	61474
PMNS avec $\phi = \sigma$	**108**	**310**	**1737**	**6695**	**22915**	**51887**	**105217**
PMNS avec $\phi = \sigma^2$	263	818	2944	9174	34435	81649	142523

TABLE 3.3 – Nombre de cycles pour une multiplication modulaire sur des entiers de tailles de 256 à 8192 bits pour des PMNS avec $\phi = \sigma$ et $\phi = \sigma^2$ utilisant la méthode de réduction interne Montgomery-like en comparison avec GMP avec gcc 12.3.0 sur un processeur intel i9-11900KF.

Les trois premières lignes du tableau 3.3 correspondent à l'utilisation de GMP sur de l'arithmétique classique pour des entiers de tailles correspondantes. Le bas niveau correspond à l'utilisation des fonctions `mpn_mul_n` et `mpn_tdiv_qr` de GMP. Montgomery et Montgomery CIOS correspondent à une implémentation des algorithmes en question proposée par GMP. Comme on peut le constater, prendre $\phi = \sigma^2$ n'apporte, en pratique, aucun gain par rapport à la version $\phi = \sigma$. Ceci est dû en grande partie au fait que l'algorithme de multiplication vecteur-matrice de Toeplitz (cf. section 2.3.3) devient beaucoup moins efficace lorsque les opérations intermédiaires provoquent des dépassements mémoire. Ces débordements, combinés aux coûts liés aux passages de retenue, dégradent les performances globales de l'algorithme bien qu'il y ait moins de coefficients à manipuler.

3.3 Interlude sur le Multi-threading

Puisque la méthode consistant à prendre $\phi = \sigma^2$ ne semblait pas porter fruit, une autre idée d'adaptation présentée en [44] a été de considérer les PMNS en multi-threading. La possibilité de paralléliser les calculs effectués au sein des PMNS avait déjà été démontrée dans [20] et [14] en utilisant le jeu d'instructions AVX512IFMA. Cependant, l'utilisation de cette vectorisation impose $\sigma = 2^{52}$ car ce jeu d'instructions ne permet de manipuler que des entiers de 52 bits. Une autre possibilité de paralléliser les opérations effectuées dans les PMNS en s'affranchissant de la contrainte sur la taille des registres mémoire est d'utiliser le multi-threading.

Ainsi, une implémentation des PMNS en parallèle a été proposée dans [44].

$\log_2(p)$	256	512	1024	2048	4096	6144	8192
GMP Montgomery	187	451	1564	4677	**14595**	**29227**	**44769**
GMP Montgomery CIOS	**105**	**340**	**1206**	**4327**	16647	34731	61474
PMNS 1 thread Toeplitz	**108**	**310**	**1737**	**6695**	22915	51887	105217
PMNS 2 threads schoolbook	6510	6764	7571	11548	24700	50987	95880
PMNS 4 threads schoolbook	9685	9726	10372	11879	**19523**	33367	56340
PMNS 6 threads schoolbook	12227	12834	12733	14314	19949	28839	44681
PMNS 8 threads schoolbook	14154	15300	15480	17534	22924	**28145**	**40596**

TABLE 3.4 – Nombre de cycles pour une multiplication modulaire sur des entiers de tailles de 256 à 8192 bits pour des PMNS en multi-threading utilisant la méthode de réduction interne Montgomery-like en comparaison avec GMP avec gcc 12.3.0 sur un processeur intel i9-11900KF.

Les performances sur plusieurs threads utilisent l'algorithme de multiplication vecteur-matrice school-

book. Comme on peut le voir dans le tableau 3.4, les performances en dessous de 2048 bits sont dégradées en multi-threading. Ceci est dû, par exemple, au surcoût nécessaire à la synchronisation des différents threads et à la mise en place de zones de mémoire partagée. Pour 4096 bits et plus, les performances sont meilleures qu'en single-thread. Pour les tailles 6144 et 8192, les performances obtenues sont meilleures que l'implémentation de la multiplication modulaire de Montgomery de GMP. Deux problèmes se posent alors.

Premièrement, il semblerait que dans l'intervalle 1024-6144 bits, les PMNS ne sont pas efficaces, malgré l'ajout du multi-threading. Cela s'explique en partie par le fait que l'algorithme de multiplication vecteur-matrice de Toeplitz (algorithme 17), utilisé en single-thread, est très rigide sur le nombre de découpages possibles des sous-produits. Le nombre de sous-produits est de la forme $\frac{\kappa(\kappa+1)}{2}$ avec κ un facteur premier de n. Cela nous donne 3 pour n pair, 6 pour n multiple de 3 et 15 pour n multiple de 5 par exemple. Il n'existe pas de découpage en 8 sous-produits possibles et donc cette méthode n'est pas optimale sur un processeur possédant 8 cœurs physiques, ce qui est le cas du processeur que nous avons choisi pour nos tests, et de la plupart des processeurs modernes.

Deuxièmement, nous nous comparons à GMP sur un seul thread, ce qui ne paraît pas pertinent. Nous avons donc parallélisé le code GMP afin d'évaluer s'il y a toujours un gain. À cela nous ajoutons la variante de PMNS parallélisée faisant utilisation du découpage Toeplitz en 6 nécessitant $n \equiv 0 \pmod 3$.

$\log_2(p)$	6144	8192
GMP bas niveau 2 threads	34362	49726
GMP Montgomery 2 threads	28256	40808
GMP bas niveau 4 threads	31274	45164
GMP Montgomery 4 threads	**25387**	**36411**
GMP bas niveau 8 threads	34361	50654
GMP Montgomery 8 threads	28842	41619
PMNS 4 threads schoolbook	33367	56340
PMNS 6 threads schoolbook	28839	44681
PMNS 8 threads schoolbook	28145	40596
PMNS 6 threads Toeplitz	29985	42982

TABLE 3.5 – Nombre de cycles pour une multiplication modulaire sur des entiers de tailles 6144 et 8192 bits pour des PMNS en multi-threading utilisant la méthode de réduction interne Montgomery-like en comparaison avec GMP en multithread avec gcc 12.3.0 sur un processeur intel i9-11900KF.

Le découpage Toeplitz n'est plus optimal en multi-threading pour des entiers de taille 8192 bits, contrairement aux travaux présentés dans [44]. En effet, le découpage Toeplitz en 6 threads nécessite $n = 189$ pour cette taille d'entiers alors que les nouvelles bornes (cf. sections 2.4.4, 2.4.5) permettent de générer des PMNS avec $n = 184$. En théorie, on pourrait effectuer un découpage de Toeplitz en 6 threads pour $n = 186$ mais cela ne mène pas à de meilleurs résultats en pratique car $186 = 2 \times 3 \times 31$ et le découpage est donc moins bon que $189 = 3 \times 3 \times 3 \times 7$.

Le passage de 1 à 8 threads améliore les performances des PMNS d'un facteur environ 2.59 sur des entiers de taille 8192 bits. En contraste, GMP ne s'améliore que d'un facteur d'environ 1.23 en 4 threads et d'un facteur d'environ 1.18 sur 8 threads. Cela confirme le fait que les PMNS sont plus enclins à être parallélisés que l'arithmétique classique sur les entiers. Malgré ce meilleur gain relatif, les performances de GMP s'avèrent

meilleures du fait que la version single-thread est déjà bien plus rapide que les PMNS en single-thread sur des entiers de taille 8192 bits. En conclusion, malgré les gains obtenus grâce à cette méthode, il semblerait que cela ne soit pas suffisant pour battre GMP.

3.4 PMNS multi-précisions sur coefficients réduits

Nous détaillons dans cette partie les diverses adaptations proposées dans [43] afin d'améliorer les performances des PMNS sur les entiers de grande taille.

Pour commencer, comme remarqué précédemment, l'implémentation proposée dans [44] était naïve et entraînait des passages de retenue très coûteux et une perte de performances avec la méthode de multiplication vecteur-matrice de Toeplitz (cf. section 2.3.3). Dans ce contexte, le recours à une multi-précision "plus adaptée" à la structure du problème apparaît comme une solution possible pour améliorer les performances.

Soit $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS avec $\rho > \sigma$. Soient $A, B \in \mathcal{B} \times \mathcal{B}$. On note $A(X) = a_0 + a_1 X + \cdots + a_{n-1}X^{n-1}$ et $B(X) = b_0 + b_1 X + \cdots + b_{n-1}X^{n-1}$. Pour stocker un certain coefficient a_i, la stratégie utilisée dans la section précédente consistait à stocker a_i modulo σ dans un registre et $\left\lfloor \frac{a_i}{\sigma} \right\rfloor$ dans un autre, comme vu dans l'équation (3.2). Malheureusement, lors de la multiplication de A par B, la multiplication de la partie basse d'un coefficient a_i donné avec la partie basse d'un coefficient b_j donné pouvait mener à un résultat sur exactement $2 \times \log_2(\sigma)$ bits, ce qui pouvait mener à des débordements lors de l'accumulation des produits partiels. De façon similaire, les opérations intermédiaires de Toeplitz pouvaient aussi mener à des débordements.

Cependant, si l'on adopte une autre stratégie de représentation, en stockant chaque a_i modulo $\lceil \sqrt{\rho} \rceil$ dans un registre et $\left\lfloor \frac{a_i}{\sqrt{\rho}} \right\rfloor$ dans un second, on s'assure qu'aucun des deux registres n'est complètement saturé. Pour reprendre l'exemple de l'équation (3.1), on aurait alors plutôt :

$$A_{128}(X) = -(\underbrace{\texttt{0x1790ebf360102}}_{\text{premier registre}} \times \lceil \sqrt{\rho} \rceil + \underbrace{\texttt{0x149543eb7ae08}}_{\text{deuxième registre}})X - (\underbrace{\texttt{0x10cb35b720480}}_{\text{premier registre}} \times \lceil \sqrt{\rho} \rceil + \underbrace{\texttt{0x2f11bdc9d272c}}_{\text{deuxième registre}}) \tag{3.3}$$

Comme on peut le voir, les registres sont plus équilibrés qu'auparavant. Ainsi, la multiplication de ces représentations par une partie quelconque de b_j, stockée de manière identique, n'entraînera pas de débordement, pourvu que ρ soit convenablement choisi. Cette méthode est très similaire à celle utilisée dans [6] où elle a la nomenclature de "coefficients réduits".

Afin de généraliser cette représentation, considérons à présent le cas où $\rho > \sigma^{s-1}$ pour un certain s. Chaque coefficient devra alors être contenu dans s registres mémoire. On note que l'on aura toujours $\phi > \rho$. Ainsi, puisque nous devons effectuer une division par ϕ, on peut diviser chaque coefficient de sorte à ce que seulement des valeurs entre $-\lceil \sqrt[s]{\phi} \rceil$ et $\lceil \sqrt[s]{\phi} \rceil - 1$ soient contenues dans chaque registre mémoire au début afin de pouvoir effectuer la division par ϕ en s étapes de façon efficace. Soit $C \in \mathcal{B}$, on a donc ainsi

$$C(X) = \sum_{k=0}^{n-1} c_k X^k$$

avec

$$c_k = \sum_{i=0}^{s-1} c_{k_i} \lceil \sqrt[s]{\phi} \rceil^i.$$

Donc,

$$C(X) = \sum_{i=0}^{s-1} (\sum_{k=0}^{n-1} c_{k_i} X^k) \lceil \sqrt[s]{\phi} \rceil^i = \sum_{i=0}^{s-1} C_i \lceil \sqrt[s]{\phi} \rceil^i$$

où

$$C_i = \sum_{k=0}^{n-1} c_{k_i} X^k \tag{3.4}$$

Par exemple si l'on prend le PMNS $\mathcal{B} = (p = 16777259, n = 2, \gamma = 171495, \rho = 5185)$ avec $\phi = 2^{16}$ et que l'on choisit $s = 2$, le polynôme $A \in \mathcal{B}$ tel que

$$A(X) = -\texttt{0xff0}X + \texttt{0xc53}$$

sera décomposé en

$$A_0 = \texttt{0x10}X + \texttt{0x53}$$

et

$$A_1 = -\texttt{0x10}X + \texttt{0xc},$$

ayant bien ici $A(X) = A_1 \times 2^8 + A_0$, ($-\texttt{0x1000} + \texttt{0x10} = -\texttt{0xff0}$ et $\texttt{0xc00} + \texttt{0x53} = \texttt{0xc53}$).

L'avantage de la méthode des coefficients réduits est que la propagation de retenue peut être repoussée jusqu'à la fin des opérations. Comme vu en section 2.3.3, l'utilisation des matrices de Toeplitz mène à de très bonnes performances. Afin d'exploiter ceci, on choisira donc $E(X)$ de la forme $X^n - \lambda$ lors de la génération du PMNS. En conséquence, il est crucial que les coefficients de la matrice de Toeplitz ne débordent pas des registres. Cela impose la contrainte suivante :

$$|\lambda| \lceil \sqrt[s]{\phi} \rceil < \sigma$$

De plus, il faut aussi prendre en compte les opérations intermédiaires de la méthode de Toeplitz. Le nombre maximal d'additions/soustractions effectuées pour un coefficient donné avant multiplication est de n, avec des coefficients de tailles bornés par $|\lambda| \lceil \sqrt[s]{\phi} \rceil$. On doit donc s'assurer que :

$$n|\lambda| \lceil \sqrt[s]{\phi} \rceil < \sigma \tag{3.5}$$

En effet, sans cela, la construction de la matrice de Toeplitz à partir de l'équation (2.6), page 51, deviendrait impossible en raison de dépassements de capacité.

À cela vient s'ajouter la nécessité de diviser par $\sqrt[s]{\phi}$ lors de la réduction interne (cf. section 2.4). Malheureusement, on ne pourra jamais avoir $\sqrt[s]{\phi} = \sigma$ avec les contraintes de l'équation (3.5). La meilleure approche consiste donc à s'assurer que $\sqrt[s]{\phi}$ soit une puissance de 2 afin d'effectuer la division avec des décalages binaires. On pose donc

$$\phi = 2^{s \left\lfloor \frac{\log_2(\sigma) - \log_2(n|\lambda|)}{s} \right\rfloor} \tag{3.6}$$

3.4.1 Génération

Dans cette section, nous détaillerons le processus de génération pour ces PMNS multi-précisions où chaque coefficient polynomial est stocké sur s registres mémoire. On considère ici des PMNS utilisant la méthode de réduction interne Montgomery-like (algorithme 18, page 59). Un code de génération est disponible à l'adresse https://github.com/francoispalma/PMNS/tree/master/multiprecision_pmns.

Algorithme 27 Génération de PMNS multi-précisions

Require: ℓ la taille d'entiers premiers considérés, un paramètre $n \in \mathbb{N}^*$
Ensure: un tuple $(p, n, \gamma, \rho, E, \delta, \phi, s)$ avec (p, n, γ, ρ, E) un PMNS tel que $E(X) = X^n - \lambda$ et $\rho < \sigma^s$

1: $p \leftarrow$ random_prime$(2^{\ell-1}, 2^\ell - 1)$
2: $\lambda \leftarrow$ une racine n-ième dans $\mathbb{Z}/p\mathbb{Z}$, de préférence telle que $X^n - \lambda$ soit irréductible dans $\mathbb{Z}$ (cf. section 1.5.3).
3: $E \leftarrow X^n - \lambda$
4: $\gamma \leftarrow \lambda^{\frac{1}{n}} \bmod p$
5: $\mathbf{B} \leftarrow$ la matrice de l'équation (1.1), page 1.1
6: $\mathcal{G} \leftarrow \mathrm{LLL}(B)$
7: $\rho \leftarrow \|\mathcal{G}\|_1 - 1$
8: $s \leftarrow \lceil \log_\sigma(2w\rho(\delta+1)^2) \rceil$
9: $\phi \leftarrow 2^{s\left\lfloor \frac{\log_2(\sigma) - \log_2(n|\lambda|)}{s} \right\rfloor}$
10: $\delta \leftarrow \left\lfloor \sqrt{\frac{\phi}{2w\rho}} \right\rfloor - 1$
11: **return** $(p, n, \gamma, \rho, E, \delta, \phi, s)$

Cet algorithme retourne forcément un PMNS valide pour tout n donné en entrée. Si l'on souhaite obtenir des PMNS performants en pratique pour une taille d'entiers donnée, il est préférable de considérer le n_{opt} pour la taille correspondante (cf. tableau 2.9, page 72) et de choisir comme paramètre n de l'algorithme un entier de la forme $\lceil \frac{n_{opt}}{z} \rceil$ pour z un petit entier. Si le PMNS obtenu en sortie est tel que $s = z$, le découpage sera alors plus efficace que dans le cas contraire, le PMNS étant optimal si $s \times n = n_{opt}$.

Remarque 35. On note que l'on peut substituer la ligne 1 de l'algorithme 27 par un choix de p arbitraire pour les cas d'usage où l'entier premier est défini dans le standard et ne nécessite pas de génération aléatoire.

3.4.2 Réduction interne Montgomery CIOS-like

La réduction interne présentée en sections 1.5.4 et 2.4.1 est basée sur une adaptation de l'algorithme classique de la réduction de Montgomery aux polynômes. Comme l'on a pu le voir dans la section 1.2.3, la variante CIOS est la plus utilisée en software dans l'état de l'art. Dans nos mesures de performance (par exemple tableau 3.3), Montgomery CIOS est plus rapide que la variante classique pour des coefficients de taille 2048 bits et moins. Donc, tant que $\sigma^s < 2^{2048}$, il semble pertinent de considérer la variante CIOS pour nos PMNS à coefficients polynomiaux multi-précisions. Nous présentons donc une variante de la réduction interne Montgomery-like (algorithme 18, page 59) qui intègre les étapes de multiplication et réduction externe afin de se conformer à la méthode CIOS.

Dans l'algorithme 28, chaque polynôme $P(X)$ est tel que $P(X) = \sum_{i=0}^{s-1} P_i(\sqrt[s]{\phi})^i$ avec $\forall i$, $\|P_i\|_\infty < \sqrt[s]{\phi}$ (cf. équation (3.4) pour l'expression de P_i). Pour guise d'exemple, prenons le PMNS $\mathcal{B} = ($
$p = 16777213$,

Algorithme 28 Multiplication Modulaire dans les PMNS avec réduction interne Montgomery CIOS-like

Require: $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS avec $E(X) = X^n - \lambda$, $A, B \in \mathcal{B} \times \mathcal{B}$, s le nombre de registres mémoire nécessaires pour contenir un seul coefficient polynomial, $\phi = 2^h$ avec $h \in \mathbb{N}^*$ un multiple de s, $M(X) \in \mathbb{Z}_{n-1}[X]$ tel que $M(\gamma) \equiv 0 \pmod{p}$ et $M' \equiv -M^{-1} \pmod{E(X), \phi}$.

Ensure: $C(\gamma) \equiv A(\gamma)B(\gamma)\phi^{-1} \pmod{p}$, avec $C \in \mathbb{Z}_{n-1}[X]$ tel que $\forall i$, $\|C_i\|_\infty < \sqrt[s]{\phi}$ et $\|C\|_\infty < \rho$

1: $R \leftarrow 0$
2: **for** $i = 0 \dots s-1$ **do**
3: **for** $j = 0 \dots s-1$ **do**
4: $R_{i+j} \leftarrow R_{i+j} + A_j \times B_i \pmod{E(X)}$
5: **end for**
6: $T \leftarrow R_i \times M'_0 \pmod{E(X), \lceil \sqrt[s]{\phi} \rceil}$
7: **for** $j = 0 \dots s-1$ **do**
8: $Q_j \leftarrow T \times M_j \pmod{E(X)}$
9: **end for**
10: **for** $j = 0 \dots s-1$ **do**
11: $R_{i+j} \leftarrow R_{i+j} + Q_j$
12: **end for**
13: $R_{i+1} \leftarrow R_i / \sqrt[s]{\phi}$ # Division exacte
14: **end for**
15: **for** $i = 0 \dots s-1$ **do**
16: $C_i \leftarrow R_{s+i} \pmod{\sqrt[s]{\phi}}$ # Reconstitution
17: $R_{s+i+1} \leftarrow \left\lfloor \frac{R_{s+i}}{\sqrt[s]{\phi}} \right\rfloor$ # Propagation de retenue repoussée comme mentionné page 92
18: **end for**
19: **return** C

$n = 2$,
$\gamma = 12384695$,
$\rho = 4194304$,
$E(X) = X^2 + 1$) avec $\phi = 2^{24}$ et $s = 2$.

On choisit $M(X) = (\texttt{0x101} \times 2^{12} + \texttt{0xf07})X + \texttt{0x44} \times 2^{12} + \texttt{0xf76}$, ce qui donne $M' = (\texttt{0xfc2} \times 2^{12} + \texttt{0xa2b})X + \texttt{0xd62} \times 2^{12} + \texttt{0x1e2}$. Considérons les polynômes

$$A(X) = (\texttt{0x2e2} \times 2^{12} + \texttt{0x3f8})X + (\texttt{0x1d0} \times 2^{12} + \texttt{0x1a0})$$

et

$$B(X) = (\texttt{0x2ff} \times 2^{12} + \texttt{0x238})X + (\texttt{0x396} \times 2^{12} + \texttt{0x3cb})$$

Supposons que l'on veuille calculer la multiplication modulaire de A par B. D'une façon similaire à la variante sur les entiers (cf. algorithme 7), au lieu de calculer directement $A(X) \times B(X) \pmod{E(X)}$, on calculera plutôt $A(X) \times B_0 \pmod{E(X)}$ que l'on réduira immédiatement après.

$$\begin{aligned} A(X) \times B_0 &\equiv ((\texttt{0x2e2} \times 2^{12} + \texttt{0x3f8})X + (\texttt{0x1d0} \times 2^{12} + \texttt{0x1a0})) \times (\texttt{0x238}X + \texttt{0x3cb}) \\ &\equiv (\texttt{0xef4b6} \times 2^{12} + \texttt{0x12a8a8})X + (\texttt{0x7a80} \times 2^{12} - \texttt{0x2a460}) \pmod{E(X)} \end{aligned}$$

Étant donné que la partie basse a déjà été complètement calculée, on peut alors calculer un polynôme Q pour annuler les bits de poids faible du résultat avant de diviser par $\sqrt[s]{\phi}$.

$$\begin{aligned} Q(X) &\equiv (\texttt{0x12a8a8}X - \texttt{0x2a460}) \times (\texttt{0xa2b}X + \texttt{0x1e2}) \\ &\equiv \texttt{0x30}X + \texttt{0xf08} \pmod{E(X), 2^{12}} \end{aligned}$$

Donc

$$\begin{aligned} A(X) \times B_0 + Q(X) \times M(X) &\equiv (\texttt{0xef4b6} \times 2^{12} + \texttt{0x3dc1f000})X + (\texttt{0x7a80} \times 2^{12} + \texttt{0xf333f000}) \\ &\equiv (\texttt{0x12d} \times 2^{24} + \texttt{0xd5} \times 2^{12})X + (\texttt{0xfa} \times 2^{24} + \texttt{0xdbf} \times 2^{12}) \pmod{E(X)} \end{aligned}$$

D'où

$$\frac{A(X) \times B_0 + Q(X) \times M(X)}{2^{12}} \equiv (\texttt{0x12d} \times 2^{12} + \texttt{0xd5})X + (\texttt{0xfa} \times 2^{12} + \texttt{0xdbf}) \pmod{E(X)}$$

On calcule ensuite $A(X) \times B_1 \pmod{E(X)}$ avant de l'ajouter au résultat en cours et ainsi de suite. Il s'agit simplement d'un réordonnancement de l'algorithme de réduction interne entremêlé avec l'étape de multiplication.

Pour finir, on effectue une propagation de retenue à la fin de l'algorithme. Comme noté plus haut, l'avantage de la méthode des coefficients réduits est que la propagation de retenue peut être repoussée jusqu'à la fin des opérations. Ainsi, nous avons repoussé la propagation de retenue jusqu'à la fin des opérations. Cela permet de n'avoir à effectuer qu'une seule étape de propagation de retenues au lieu de devoir systématiquement les effectuer au milieu des opérations. Ceci est seulement possible grâce à la marge supplémentaire permise par les coefficients réduits.

Chaque étape de multiplication dans l'algorithme fait usage de l'algorithme de multiplication vecteur-matrice de Toeplitz récursif (algorithme 17, page 53) afin de garantir une complexité sous-quadratique.

Nous allons maintenant analyser la complexité de cet algorithme en le comparant avec la réduction interne Montgomery-like (algorithme 18, page 59). Celui-ci avait déjà été analysé en section 2.4.7 pour référence. Chaque opération de multiplication dans cet algorithme fait usage de l'algorithme de multiplication vecteur-matrice récursif de Toeplitz (algorithme 17, page 53). Le pire cas pour son nombre de multiplications intervient lorsque n est premier, comme détaillé dans la proposition 2.3.7, page 55. Supposant n premier, le nombre de multiplications élémentaires pour un n donné sera de $n^{2-\log_n(2)}$. Ensuite, pour la multiplication de A par B, l'algorithme 28 effectue s^2 itérations. La multiplication par M'_0 est effectuée s fois. Pour finir, la multiplication par M est calculée s^2 fois.

Le tableau 3.6 résume les différences principales en termes de nombre de multiplications pour chaque étape, avec n_{sp} le paramètre n correspondant à un PMNS classique où $\rho < \sigma$ et n_{mp} l'équivalent pour la nouvelle variante multi-précision où chaque coefficient polynomial tient sur s registres mémoire. Cette table ne tient pas compte du fait que la variante simple-précision peut se permettre de prendre $\phi = \sigma$ pour effectuer les opérations de divisions avec une simple instruction MOV. De façon similaire, les réductions modulo ϕ seront forcément plus coûteuses en variante multi-précision puisqu'elles sont gratuites avec $\phi = \sigma$. Pour

Étape \ Méthode	Montgomery-like classique	Montgomery CIOS-like multi-précision
A fois B	$n_{sp}^{2-\log_{n_{sp}}(2)}$	$s^2 \times n_{mp}^{2-\log_{n_{mp}}(2)}$
Multiplication par $\mathcal{M}'$	$n_{sp}^{2-\log_{n_{sp}}(2)}$	$s \times n_{mp}^{2-\log_{n_{mp}}(2)}$
Multiplication par $\mathcal{M}$	$n_{sp}^{2-\log_{n_{sp}}(2)}$	$s^2 \times n_{mp}^{2-\log_{n_{mp}}(2)}$
Total	$3n_{sp}^{2-\log_{n_{sp}}(2)}$	$s \times (2s+1) \times n_{mp}^{2-\log_{n_{mp}}(2)}$

TABLE 3.6 – Nombre de multiplication pour chaque étape des algorithmes Montgomery-like et Montgomery CIOS-like pour n_{sp}/n_{mp} premiers.

compenser, on note qu'en pratique on a toujours $s \times n_{mp} \leqslant n_{sp}$, comme on peut le voir dans le tableau 3.7. Comme on peut le constater, l'algorithme 28 est sous-quadratique en n mais quadratique en s, ce qui justifiera de toujours prendre $n_{mp} \geqslant s$.

Taille d'entiers	1024	2048	4096	6144	8192
n_{opt} (équation (2.14), page 72)	17	34	69	105	141
n_{sp}	19	39	83	130	183
$n_{mp} \times s$ pour $s = 2$	18	36	80	120	160
$n_{mp} \times s$ pour $s = 3$	18	36	72	108	144
$n_{mp} \times s$ pour $s = 4$	20	36	72	108	144

TABLE 3.7 – Comparaisons des valeurs de n_{sp} et $s \times n_{mp}$ trouvées en pratique pour des PMNS construits sur des entiers de tailles allant de 1024 à 8192 bits.

Lorsque l'on augmente s dans les grandes tailles, $s \times n_{mp}$ s'approche de n_{opt}, ce qui justifie l'utilisation des coefficients polynomiaux multi-précision. Il semblerait que le bon compromis soit de choisir s le plus petit possible tel que $s \times n_{mp}$ soit optimal.

On présente ensuite les performances pour cet algorithme dans le tableau suivant.

Méthode \ Taille d'entiers	1024	2048	4096	6144	8192
GMP Montgomery	1564	4677	14595	29227	44769
GMP Montgomery CIOS	1206	4327	16647	34731	61474
PMNS avec $\phi = \sigma$	1737	6695	22915	51887	105217
PMNS avec $\phi = \sigma^2$	2944	9174	34435	81649	142523
PMNS multi-précision avec $s = 2$	**1173**	**3874**	16254	30795	54146
PMNS multi-précision avec $s = 3$	1195	3901	**13444**	**26497**	**43835**
PMNS multi-précision avec $s = 4$	1511	4218	14766	30973	45392

TABLE 3.8 – Nombre de cycles pour une multiplication modulaire sur les grands entiers avec GCC 12.3.0 sur un processeur intel i9-11900KF.

Comme l'illustre la table 3.8, cette nouvelle méthode est plus efficace et permet enfin de concurrencer GMP sur l'ensemble des tailles d'entiers considérées. Pour rappel, les PMNS étaient déjà plus performants que GMP pour les tailles comprises entre 256 et 512 bits. Cette nouvelle variante n'apporte donc pas de gain sur ces tailles, les PMNS classiques associés existants pour $n = n_{opt}$.

Il s'agit ici d'une implémentation purement séquentielle et compilée avec l'instruction de compilation **-fno-tree-vectorize** afin de s'assurer que les instructions SIMD ne sont pas automatiquement utilisées par GCC. Cela permet une comparaison équitable avec GMP qui n'en utilise pas non plus.

Une question naturelle est donc de considérer l'utilisation des instructions AVX512. Comme mentionné précédemment, les PMNS ont déjà fait leurs preuves dans de telles circonstances [20, 14]. La plupart des implémentations de référence en cryptographie ont tendance à proposer une version AVX2 qui permet des multiplications 32-bit complètes vectorisées mais cela impliquerait de prendre $\sigma = 2^{32}$. Ceci entraînerait une augmentation significative du paramètre n avec un impact non négligeable sur les performances globales. D'après nos expérimentations, cette approche ne s'est pas révélée avantageuse. En revanche, les instructions AVX512IFMA permettent des instructions fusionnées d'addition et de multiplications complètes sur 52 bits. Ceci permet de fixer $\sigma = 2^{52}$, ce qui diminue l'impact sur la taille du paramètre n par rapport au choix $\sigma = 2^{32}$. Les résultats expérimentaux obtenus montrent que le gain en performance justifie ce choix en pratique.

Méthode \ Taille d'entiers	1024	2048	4096	6144	8192
PMNS multi-précisions avec $s = 2$	747	1574	5227	12725	25220
PMNS multi-précisions avec $s = 3$	457	1577	5132	10891	15896
PMNS multi-précisions avec $s = 4$	-	-	5563	12844	19012

TABLE 3.9 – Nombre de cycles pour une multiplication modulaire sur les grands entiers utilisant le jeu d'instructions AVX512IFMA avec GCC 12.3.0 sur un processeur intel i9-11900KF.

Méthode \ Taille d'entiers	1024	2048	4096	6144	8192
PMNS multi-précisions avec $s = 2$	1.570	2.461	2.572	2.082	1.738
PMNS multi-précisions avec $s = 3$	2.566	2.457	2.619	2.433	2.758
PMNS multi-précisions avec $s = 4$	-	-	2.417	2.063	2.306

TABLE 3.10 – Ratio des valeurs dans la table 3.9 avec la meilleure valeur du tableau 3.8 pour chaque taille d'entiers.

Le jeu d'instructions AVX512IFMA ne fonctionne que pour des entiers non-signés. Ainsi, l'utilisation de ces instructions sous-entend d'utiliser un polynôme M dont tous les coefficients sont positifs. Cela est possible mais complique légèrement le processus de génération. À l'heure actuelle, la possibilité de trouver un polynôme M dans le réseau des polynômes qui s'annulent en γ modulo p inversible modulo E, modulo ϕ, dont tous les coefficients sont positifs pour tout PMNS reste une question ouverte.

Dans cette étude, nous avons limité nos comparaisons à des versions séquentielles, dans le but de mettre en évidence l'intérêt potentiel d'exploiter les instructions SIMD. Afin d'évaluer l'efficacité de la version SIMD des PMNS, il semble naturel de se comparer à une bibliothèque tirant aussi partie des instructions SIMD. À notre connaissance, la seule bibliothèque proposant une implémentation de la multiplication modulaire utilisant explicitement les instructions AVX512IFMA est OpenSSL.

Les résultats expérimentaux de la table 3.11 semblent montrer qu'OpenSSL s'avère être plus performant pour les entiers sur 1536 et 2048 bits. Ceci est à tempérer avec le fait que l'implémentation d'OpenSSL est

Méthode \ Taille d'entiers	1024	1536	2048
OpenSSL Montgomery CIOS	505	846	1177
PMNS multi-précision vectorisé	453	1065	1492

TABLE 3.11 – Comparaison du nombre de cycles pour une multiplication modulaire sur les grands entiers avec les seules tailles disponibles pour comparaison dans OpenSSL tirant partie des instructions AVX512IFMA avec GCC 12.3.0 sur processeur intel i9-11900KF.

en assembleur optimisé et notre code de comparaison est en C. Une implémentation des PMNS en assembleur optimisé, exploitant pleinement les instructions AVX512IFMA, pourrait éventuellement rivaliser avec OpenSSL. Ce point n'a toutefois pas été exploré dans ce travail, par manque de temps.

Pour finir, un autre point de comparaison pour les multiplications modulaires SIMD utilisant le jeu d'instructions AVX512IFMA est [14].

Méthode \ Tailles d'entiers	807	1214	1621	2029	2436	2844	3251
Résultats dans [14]	779	1004	1965	3764	6233	9093	19157
PMNS multi-précisions vectorisés	457	747	1573	2443	3244	5124	5125

TABLE 3.12 – Comparaison du nombre de cycles pour une multiplication modulaire sur les PMNS avec un $\delta = 5$ avec le jeu d'instructions AVX512IFMA avec GCC 12.3.0 sur processeur intel i9-11900KF.

Une particularité de ce tableau est que le paramètre δ, jusqu'à présent supposé égal à 0 pour des soucis d'optimisation, est pris comme $\delta = 5$ dans [14] pour utiliser la représentation PMNS au sein des formules d'addition de points sur les courbes elliptiques. Ainsi, pour que la comparaison soit équitable, nous utilisons ici des PMNS générés avec $\delta = 5$ pour chacune des tailles considérées. Les résultats dans le tableau attribués à [14] viennent d'une version recompilée de leur code source avec GCC 12.3.0 et mesurée sur la même machine afin d'avoir une comparaison pertinente.

Tous les tableaux présentés dans ce chapitre sont accompagnés de leur code source, disponible à l'adresse suivante : https://github.com/francoispalma/PMNS/tree/master/multiprecision_pmns.

3.5 Conclusion

Dans ce chapitre, nous avons exploré les différentes adaptations des PMNS pour le traitement des grands entiers. Il en ressort que l'utilisation de coefficients polynomiaux multi-précision soit, à ce jour, l'approche la plus efficace. Les PMNS s'avèrent être bien adaptés au multi-threading et revisiter le multi-threading en conjonction avec les coefficients polynomiaux multi-précision représente une possible piste d'exploration pour de futurs travaux. Les instructions SIMD, en particulier AVX512IFMA, confirment leur intérêt pour l'accélération des opérations arithmétiques. Néanmoins, il apparaît qu'OpenSSL conserve un avantage pour les entiers de tailles 1536 et 2048 bits, du fait certainement d'optimisations en assembleur. Une piste d'amélioration envisageable serait l'étude plus approfondie de la réduction interne Barrett-like en contexte multi-précision pour permettre aux PMNS de reprendre l'avantage sur les grandes tailles.

Chapitre 4

Construction de PMNS sur formes d'entier particulières

Notations :
- $\mathbb{Z}_m[X]$ le $\mathbb{Z}$-module des polynômes à coefficients entiers de degré au plus m.
- $\nu_n : \mathbb{Z}_{n-1}[X] \to \mathbb{Z}^n$ l'isomorphisme tel que $\nu_n(a_0 + a_1X + \cdots + a_{n-1}X^{n-1}) = (a_0, a_1, \ldots, a_{n-1})$ pour un n donné.
- $\pi_n : \mathbb{Z}^n \to \mathbb{Z}_{n-1}[X]$ l'isomorphisme tel que $\pi_n((a_0, a_1, \ldots, a_{n-1})) = a_0 + a_1X + \cdots + a_{n-1}X^{n-1}$ pour un n donné.
- σ le nombre de valeurs possibles dans un registre mémoire (2^{64} pour la plupart des CPU modernes).
- $M_{i,j}$ pour M une matrice représente le coefficient situé sur la $i$ème ligne et la $j$ème colonne, en partant de 0. Une matrice M de dimension $n \times m$ aura donc des coefficients de $M_{0,0}$ à $M_{n-1,m-1}$.
- $\lfloor a \rceil$ pour $a \in \mathbb{R}$ représente la valeur arrondie de a, équivalente à $\lfloor a + \frac{1}{2} \rfloor$.
- Pour $v = (v_0, v_1, \ldots, v_{n-1}) \in \mathbb{R}^n$, on aura $\lfloor v \rceil = (\lfloor v_0 \rceil, \lfloor v_1 \rceil, \ldots, \lfloor v_{n-1} \rceil)$.
- Pour $M \in \mathcal{M}_n(\mathbb{R})$, on notera $\lfloor M \rceil$ la matrice telle que $\forall i, j \in [\![0, n-1]\!] \times [\![0, n-1]\!]$, $\lfloor M \rceil_{i,j} = \lfloor M_{i,j} \rceil$.
- Un PMNS $\mathcal{B}$ est un tuple (p, n, γ, ρ, E) représentant les entiers de $\mathbb{Z}/p\mathbb{Z}$ par des polynômes sur n coefficients (donc de degré au plus $n-1$). Un entier a est représenté par un polynôme $A(X)$ si $A(\gamma) \equiv a \pmod{p}$. Le paramètre ρ représente la borne sur la taille des coefficients polynomiaux des éléments du PMNS. Pour réduire le degré après multiplication, le polynôme E de degré n qui s'annule en γ modulo p est utilisé.

4.1 Introduction

Ce chapitre détaille une nouvelle classe d'entiers premiers que nous avons proposé pour la première fois dans [19] avec Dosso, El Mrabet, Méloni et Véron. Cette classe d'entiers possède une forme particulière permettant une réduction interne linéaire dans un PMNS, obtenant ainsi des performances très proches de l'arithmétique des nombres pseudo-Mersenne tout en ayant une densité bien plus forte. Elle englobe notamment les deux classes d'entiers particuliers propices aux PMNS présentées dans [11] et [8].

4.2 PMNS à réduction interne linéaire

Cette approche tire parti de la réduction interne Montgomery-like sur les réseaux (cf. algorithme 18, page 59). Nous définissons donc tout d'abord une base d'un sous-réseau de

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

Proposition 4.2.1. *Soit* $\mathcal{B} = (p, n, \gamma, \rho, E)$ *avec*
- $p \in \mathbb{N}^*$
- $n \in \mathbb{N}$, $n > 2$
- $E \in \mathbb{Z}[X]$ *tel que* $\deg(E) = n$ *et* $E(\gamma) = kp$, $k \in \mathbb{Z}^*$

et soit $\mathfrak{L} = \left\{ (x_0, \dots, x_{n-1}) \in \mathbb{Z}^n : \sum_{i=0}^{n-1} x_i \gamma^i \equiv 0 \pmod{p} \right\}$, *alors*

$$\mathcal{G} = \begin{pmatrix} -\gamma & 1 & 0 & 0 & \dots & 0 \\ 0 & -\gamma & 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \ddots & -\gamma & 1 & 0 \\ 0 & 0 & \dots & 0 & -\gamma & 1 \\ -e_0 & -e_1 & \dots & \dots & -e_{n-2} & -e_{n-1} - e_n\gamma \end{pmatrix} \begin{matrix} \leftarrow \nu_n(X - \gamma) \\ \leftarrow \nu_n((X-\gamma)X) \\ \\ \\ \leftarrow \nu_n((X-\gamma)X^{n-3}) \\ \leftarrow \nu_n((X-\gamma)X^{n-2}) \\ \leftarrow \nu_n((X-\gamma)e_nX^{n-1} - E(X)) \end{matrix} \tag{4.1}$$

avec $e_0, e_1, \dots, e_{n-1}, e_n$ *les coefficients de* E, *est une base d'un sous-réseau de* $\mathfrak{L}$.

Démonstration. D'après la définition de $\mathfrak{L}$, pour $v \in \mathbb{Z}^n$, $\pi_n(v)(\gamma) \equiv 0 \pmod{p} \implies v \in \mathfrak{L}$.

On note $\mathcal{G}_i$ la $i$ème ligne de $\mathcal{G}$ pour $i \in [\![0, n-1]\!]$. $\forall i < n-1, \mathcal{G}_i \in \mathfrak{L}$ car $\pi_n(\mathcal{G}_i)(\gamma) = (\gamma - \gamma)\gamma^i$. $\mathcal{G}_{n-1} \in \mathfrak{L}$ également car $\pi_n(\mathcal{G}_{n-1})(\gamma) = -e_0 - e_1\gamma - \dots - e_{n-1}\gamma^{n-1} - e_n\gamma^n = -E(\gamma) = -kp$ et donc $\pi_n(\mathcal{G}_{n-1})(\gamma) \equiv 0 \pmod{p}$.
La sous-matrice rectangulaire composée des lignes $(\mathcal{G}_0, \mathcal{G}_1, \dots \mathcal{G}_{n-2})$ est sous forme échelonnée et donc ces lignes sont linéairement indépendantes les unes des autres. $\forall (\mu_0, \mu_1, \dots, \mu_{n-2}) \in \mathbb{R}^{n-1}$, $\pi_n(\mu_0\mathcal{G}_0 + \mu_1\mathcal{G}_1 + \dots + \mu_{n-2}\mathcal{G}_{n-2})(\gamma) = 0$ donc $\mathcal{G}_{n-1}$ ne peut pas être combinaison linéaire des autres lignes de $\mathcal{G}$ car $\pi_n(\mathcal{G}_{n-1})(\gamma) = -kp$.
Toutes les lignes sont linéairement indépendantes donc $\det(\mathcal{G}) \neq 0$. $\mathcal{G}$ est donc une base d'un sous-réseau de $\mathcal{L}$. □

Dans certains cas, la matrice $\mathcal{G}$ ci-dessus peut s'avérer être une base de $\mathfrak{L}$. On sait que $\det(\mathfrak{L}) = p$, donc il faudrait $|\det(\mathcal{G})| = p$ pour qu'il s'agisse d'une base de $\mathfrak{L}$ et pas seulement d'un sous-réseau. On exprime donc explicitement le déterminant de $\mathcal{G}$.

Proposition 4.2.2. *Soit* $\mathcal{B} = (p, n, \gamma, \rho, E)$ *avec*
- $p \in \mathbb{N}^*$
- $n \in \mathbb{N}$, $n > 2$
- $E \in \mathbb{Z}[X]$ *tel que* $\deg(E) = n$

et soit $\mathcal{G}$ *comme décrit dans l'équation (4.1), alors* $\det(\mathcal{G}) = (-1)^n E(\gamma)$.

Démonstration. Il suffit de remarquer que

$$\det(\mathcal{G}) = \begin{vmatrix} -\gamma & 1 & 0 & 0 & \dots & 0 \\ 0 & -\gamma & 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \ddots & -\gamma & 1 & 0 \\ 0 & 0 & \dots & 0 & -\gamma & 1 \\ -e_0 - e_1\gamma - \dots - e_n\gamma^n & 0 & \dots & \dots & 0 & 0 \end{vmatrix}.$$

En effet, si l'on note $\mathcal{G}_i$ la $i$ème ligne de $\mathcal{G}$ pour $i \in [\![0, n-1]\!]$, alors

$$\mathcal{G}_{n-1} = (-e_0, -e_1, \dots, -e_{n-3}, -e_{n-2}, -e_{n-1} - e_n\gamma)$$

donc

$$\mathcal{G}_{n-1} + (e_{n-1} + e_n\gamma)\mathcal{G}_{n-2} = (-e_0, -e_1, \dots, -e_{n-3}, -e_{n-2} - e_{n-1}\gamma - e_n\gamma^2, 0)$$

et

$$\mathcal{G}_{n-1} + (e_{n-1} + e_n\gamma)\mathcal{G}_{n-2} + (e_{n-2} + e_{n-1}\gamma + e_n\gamma^2)\mathcal{G}_{n-3} = (-e_0, -e_1, \dots, -e_{n-3} - e_{n-2}\gamma - e_{n-1}\gamma^2 - e_n\gamma^3, 0, 0)$$

l'idée étant d'annuler le terme de droite non-nul à chaque fois. D'où

$$\mathcal{G}_{n-1} + \sum_{i=0}^{n-2}\Big(\sum_{j=i+1}^{n} e_j\gamma^{j-i-1}\Big)\mathcal{G}_i = (-e_0 - e_1\gamma - \dots - e_n\gamma^n, 0, \dots, 0).$$

On peut donc substituer la dernière ligne de $\mathcal{G}$ par la combinaison linéaire des lignes précisée ci-dessus et obtenir une matrice dont le déterminant est bien $(-1)^n E(\gamma)$ par développement selon la dernière ligne. Ainsi $\det(\mathcal{G}) = (-1)^n E(\gamma)$. □

Nous nous intéressons ensuite à l'inverse de $\mathcal{G}$ car nous en avons besoin pour effectuer la réduction interne Montgomery-like (algorithme 18, page 59) que nous utilisons ici.

Proposition 4.2.3. *Soit $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS avec*
- *$p \in \mathbb{N}^*$*
- *$n \in \mathbb{N}$, $n > 2$*
- *$E \in \mathbb{Z}[X]$ tel que $\deg(E) = n$ et $E(\gamma) = kp$, $k \in \mathbb{Z}^*$*

et soit $\mathcal{G}$ comme décrit dans l'équation (4.1), pour

$$\Gamma =$$

$$\begin{pmatrix} \sum_{z=1}^{n} \gamma^{z-1} e_z & \sum_{z=2}^{n} \gamma^{z-2} e_z & \sum_{z=3}^{n} \gamma^{z-3} e_z & \dots & \sum_{z=n-2}^{n} \gamma^{z-n-2} e_z & \gamma e_n + e_{n-1} & 1 \\ -e_0 & \sum_{z=2}^{n} \gamma^{z-1} e_z & \sum_{z=3}^{n} \gamma^{z-2} e_z & \dots & \sum_{z=n-2}^{n} \gamma^{z-n-1} e_z & \gamma^2 e_n + \gamma e_{n-1} & \gamma \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -\gamma^{n-3} e_0 & -\gamma^{n-3} e_1 - \gamma^{n-4} e_0 & -\sum_{z=0}^{2} \gamma^{n-5+z} e_z & \dots & -\sum_{z=0}^{n-3} \gamma^{z} e_z & \gamma^{n-1} e_n + \gamma^{n-2} e_{n-1} & \gamma^{n-2} \\ -\gamma^{n-2} e_0 & -\gamma^{n-2} e_1 - \gamma^{n-3} e_0 & -\sum_{z=0}^{2} \gamma^{n-4+z} e_z & \dots & -\sum_{z=0}^{n-3} \gamma^{z+1} e_z & -\sum_{z=0}^{n-2} \gamma^{z} e_z & \gamma^{n-1} \end{pmatrix}$$

avec (en numérotant les lignes et colonnes de 0 à $n-1$) :

— $\forall (i,j) \in [\![0, n-2]\!]^2$, $i \leqslant j \implies \Gamma_{i,j} = \sum_{z=j+1}^{n} \gamma^{z+i-1-j} e_z$

— $\forall (i,j) \in [\![0, n-1]\!] \times [\![0, n-2]\!]$, $i > j \implies \Gamma_{i,j} = -\sum_{z=0}^{j} \gamma^{z+i-1-j} e_z$

— $\forall i \in [\![0, n-1]\!]$, $\Gamma_{i,n-1} = \gamma^i$

on a $\mathcal{G}^{-1} = \frac{\Gamma}{-E(\gamma)}$.

Démonstration. On considère $\Gamma \cdot \mathcal{G}$. On note c_0 la première colonne de $\mathcal{G}$. On a $c_0 = \begin{pmatrix} -\gamma \\ 0 \\ \vdots \\ 0 \\ -e_0 \end{pmatrix}$. Ainsi

$$\Gamma \cdot c_0 = \begin{pmatrix} \sum_{z=1}^{n} \gamma^{z-1} e_z \times -\gamma + 1 \times -e_0 \\ -e_0 \times -\gamma - e_0 \gamma \\ -\gamma e_0 \times -\gamma + \gamma^2 \times -e_0 \\ \dots \\ -\gamma^i e_0 \times -\gamma + \gamma^i \times -e_0 \\ \dots \\ -\gamma^{n-2} e_0 \times -\gamma + \gamma^{n-1} \times -e_0 \end{pmatrix} = \begin{pmatrix} -E(\gamma) \\ 0 \\ \dots \\ 0 \end{pmatrix}.$$

Pour $j \in [\![1, n-2]\!]$, on a $c_j = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ -\gamma \\ 0 \\ \vdots \\ 0 \\ -e_j \end{pmatrix}$ pour c_j la $j$ième colonne de $\mathcal{G}$. Pour clarifier, le $j$ème coefficient de

cette colonne est $-\gamma$. Donc pour Γ_i la $i$ème ligne de Γ pour $(i,j) \in [\![0, n-1]\!] \times [\![0, n-2]\!]$, on aura

$$\Gamma_i \cdot c_j = \begin{cases} \sum_{z=j}^{n} \gamma^{z+i-j} e_z \times 1 + \sum_{z=j+1}^{n} \gamma^{z+i-1-j} e_z \times -\gamma + \gamma^i \times -e_j & \text{pour } i < j \\ -\sum_{z=0}^{j-1} \gamma^{z+i-j} e_z \times 1 + \sum_{z=j+1}^{n} \gamma^{z+i-1-j} e_z \times -\gamma + \gamma^i \times -e_j & \text{pour } i = j \\ -\sum_{z=0}^{j-1} \gamma^{z+i-j} e_z \times 1 - \sum_{z=0}^{j} \gamma^{z+i-1-j} e_z \times -\gamma + \gamma^i \times -e_j & \text{pour } i > j \end{cases}$$

c'est-à-dire

$$\Gamma_i \cdot c_j = \begin{cases} \sum_{z=j}^{n} \gamma^{z+i-j} e_z - \sum_{z=j}^{n} \gamma^{z+i-j} e_z & \text{pour } i < j \\ -\sum_{z=0}^{n} \gamma^{z} e_z & \text{pour } i = j \\ -\sum_{z=0}^{j} \gamma^{z+i-1-j} e_z + \sum_{z=0}^{j} \gamma^{z+i-1-j} e_z & \text{pour } i > j \end{cases}$$

donc

$$\Gamma_i \cdot c_j = \begin{cases} 0 & \text{pour } i < j \\ -E(\gamma) & \text{pour } i = j \\ 0 & \text{pour } i > j \end{cases}$$

.

Finalement $c_{n-1} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ -e_{n-1} - e_n\gamma \end{pmatrix}$

D'où

$$\Gamma \cdot c_{n-1} = \begin{pmatrix} (\gamma e_n + \gamma e_{n-1}) \times 1 + 1 \times (-e_{n-1} - e_n\gamma) \\ (\gamma^2 e_n + \gamma e_{n-1}) \times 1 + \gamma \times (-e_{n-1} - e_n\gamma) \\ \cdots \\ (\gamma^{i+1} e_n + \gamma^i e_{n-1}) \times 1 + \gamma^i \times (-e_{n-1} - e_n\gamma) \\ \cdots \\ (\gamma^{n-1} e_n + \gamma^{n-2} e_{n-1}) \times 1 + \gamma^{n-2} \times (-e_{n-1} - e_n\gamma) \\ -\sum_{z=0}^{n-2} \gamma^z e_z \times 1 + \gamma^{n-1} \times (-e_{n-1} - e_n\gamma) \end{pmatrix} = \begin{pmatrix} 0 \\ \cdots \\ 0 \\ -E(\gamma) \end{pmatrix}.$$

On a donc $\Gamma \cdot \mathcal{G} = -E(\gamma) \times I_n$ donc $\mathcal{G}^{-1} = \frac{\Gamma}{-E(\gamma)}$. □

Cette forme de matrice très particulière permet une opération de multiplication vecteur-matrice en temps linéaire. Considérons tout d'abord le lemme suivant qui met en évidence une relation entre les colonnes :

Lemme 4.2.1. *Soit $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS avec*

- $p \in \mathbb{N}^*$
- $n \in \mathbb{N}$, $n > 2$
- $E \in \mathbb{Z}[X]$ *tel que* $\deg(E) = n$ *et* $E(\gamma) = kp$, $k \in \mathbb{Z}^*$

et soit $\mathcal{G}$ comme décrit dans l'équation (4.1), alors $(c_0, c_1, \dots c_{n-1})$ les colonnes de $\mathcal{G}^{-1}$ sont telles que

$$\forall j \in [\![0, n-2]\!], c_j = c_{n-1} \times \mathcal{G}_{0,j}^{-1} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ \gamma \\ \vdots \\ \gamma^{n-2-j} \end{pmatrix} \textit{ avec le coefficient à } 1 \textit{ en position } j+1.$$

Démonstration. Puisque $\mathcal{G} \cdot \mathcal{G}^{-1} = I_n$ et que $\forall i \in [\![0, n-2]\!], \mathcal{G}_i = \nu_n((X-\gamma)X^i)$ alors $\forall j \in [\![0, n-2]\!]$, pour $i \in [\![1, n-2]\!]$,

$$\begin{cases} i < j \implies \mathcal{G}_{i,j}^{-1} = \gamma \times \mathcal{G}_{i-1,j}^{-1} \\ i > j \implies \mathcal{G}_{i+1,j}^{-1} = \gamma \times \mathcal{G}_{i,j}^{-1}. \end{cases}$$

De plus $\mathcal{G}_{j+1,j}^{-1} = \gamma \times \mathcal{G}_{j,j}^{-1} + 1$ donc si $j \neq n-2$,

$$\mathcal{G}_{j+2,j}^{-1} = \gamma^2 \times \mathcal{G}_{j,j}^{-1} + \gamma.$$

On a donc $\forall j \in [\![0, n-2]\!]$, $\forall i \in [\![0, j]\!]$,

$$\mathcal{G}_{i,j}^{-1} = \mathcal{G}_{0,j}^{-1} \times \gamma^i$$

et $\forall i \in [\![j+1, n-1]\!]$,

$$\mathcal{G}_{i,j}^{-1} = \mathcal{G}_{0,j}^{-1} \times \gamma^i + \gamma^{i-j}.$$

Si l'on note $(c_0, c_1, \dots c_{n-1})$ les colonnes de $\mathcal{G}^{-1}$, $c_{n-1} = \begin{pmatrix} 1 \\ \gamma \\ \vdots \\ \gamma^{n-1} \end{pmatrix}$ donc on a bien que $\forall j \in [\![0, n-2]\!]$

$$c_j = c_{n-1} \times \mathcal{G}_{0,j}^{-1} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ \gamma \\ \vdots \\ \gamma^{n-2-j} \end{pmatrix}$$

□

Cette propriété sur les colonnes peut donc être exploitée pour donner une opération vecteur-matrice en temps linéaire. Nous explicitons cette méthode dans le lemme suivant.

Lemme 4.2.2. *Soit $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS avec*

- $p \in \mathbb{N}^*$
- $n \in \mathbb{N}$, $n > 2$
- $E \in \mathbb{Z}[X]$ *tel que* $\deg(E) = n$ *et* $E(\gamma) = kp$, $k \in \mathbb{Z}^*$

— $\phi = 2^h$ *avec* $h \in \mathbb{N}^*$ *tel que* $\phi \leqslant \sigma$

et soit $\mathcal{G}$ *comme décrit dans l'équation (4.1), si* $\mathcal{G}$ *est inversible modulo* 2*, et donc modulo* $\phi = 2^h$*, alors la multiplication vecteur-matrice par* $\mathcal{G}^{-1}$ *modulo* ϕ *ne nécessite que* $3n-3$ *multiplications et un masque binaire sur chaque coefficient.*

Démonstration. Soit $a \in \mathbb{Z}^n$. On note $a = (a_0, a_1, \ldots, a_{n-1})$. Si l'on note $(c_0, c_1, \ldots, c_{n-1})$ les colonnes de $\mathcal{G}^{-1}$, alors on a $a \cdot \mathcal{G}^{-1} = (a \cdot c_0, a \cdot c_1, \ldots, a \cdot c_{n-1})$. Or, d'après le lemme 4.2.1, $\forall j \in [\![0, n-2]\!], c_j = c_{n-1} \times \mathcal{G}_{0,j}^{-1} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ \gamma \\ \vdots \\ \gamma^{n-2-j} \end{pmatrix}$. Ainsi, on a

$$a \cdot \mathcal{G}^{-1} = (a \cdot (c_{n-1} \times \mathcal{G}_{0,0}^{-1} + \begin{pmatrix} 0 \\ 1 \\ \gamma \\ \vdots \\ \gamma^{n-2} \end{pmatrix}), a \cdot (c_{n-1} \times \mathcal{G}_{0,1}^{-1} + \begin{pmatrix} 0 \\ 0 \\ 1 \\ \gamma \\ \vdots \\ \gamma^{n-3} \end{pmatrix}), \ldots, a \cdot (c_{n-1} \times \mathcal{G}_{0,n-2}^{-1} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}), a \cdot c_{n-1})$$

On pose $r_{n-1} = a \cdot c_{n-1}$ et on développe

$$a \cdot \mathcal{G}^{-1} = (r_{n-1} \times \mathcal{G}_{0,0}^{-1} + \sum_{i=1}^{n-1} a_i \gamma^{i-1}, r_{n-1} \times \mathcal{G}_{0,1}^{-1} + \sum_{i=2}^{n-1} a_i \gamma^{i-2}, \ldots, r_{n-1} \times \mathcal{G}_{0,n-2}^{-1} + a_{n-1}, r_{n-1})$$

On pose $s_{n-2} = a_{n-1}$ et $\forall j \in [\![0, n-3]\!]$, $s_j = s_{j+1} \times \gamma + a_{j+1}$ pour avoir

$$a \cdot \mathcal{G}^{-1} = (r_{n-1} \times \mathcal{G}_{0,0}^{-1} + s_0, r_{n-1} \times \mathcal{G}_{0,1}^{-1} + s_1, \ldots, r_{n-1} \times \mathcal{G}_{0,n-2}^{-1} + s_{n-2}, r_{n-1})$$

Le calcul de r_{n-1} nécessite n multiplications. Il faut ensuite le multiplier par $\mathcal{G}_{0,j}^{-1}$ pour $j \in [\![0, n-2]\!]$ ce qui demande $n-1$ multiplications. Ensuite, le calcul de s_j pour $j \in [\![0, n-3]\!]$ nécessite en tout $n-2$ multiplications. Le total est donc bien de $3n-3$. Il suffit ensuite d'effectuer un masque binaire sur chaque coefficient pour obtenir $a \cdot \mathcal{G}^{-1} \pmod{\phi}$ si $\phi \neq \sigma$. □

L'algorithme 18 (page 59) effectue 2 opérations de multiplication vecteur-matrice successives. La multiplication par $\mathcal{G}^{-1}$ modulo ϕ vérifie les conditions du lemme 4.2.2. Ainsi, avec des conditions supplémentaires sur γ, nous pouvons obtenir une complexité linéaire au total.

Proposition 4.2.4. *Soit* $\mathcal{B} = (p, n, \gamma, \rho, E)$ *un PMNS avec*

— $p \in \mathbb{N}^*$

— $n \in \mathbb{N}$, $n > 2$

— $E \in \mathbb{Z}[X]$ *de degré* n*, écrit* $E(X) = e_0 + e_1 X + \cdots + e_n X^n$ *et tel que* $E(\gamma) = kp$, $k \in \mathbb{Z}^*$ *avec* $\|E\|_\infty < \sigma$

— $\gamma \in \mathbb{Z}/p\mathbb{Z}$, *tel que* $|-e_{n-1}-e_n|\gamma < \sigma$
— $\phi = 2^h$ *avec* $h \in \mathbb{N}^*$ *tel que* $\phi \leqslant \sigma$

et soit $\mathcal{G}$ *comme décrit dans l'équation (4.1), si* $\mathcal{G}$ *est inversible modulo* 2 *alors l'algorithme 18 peut être effectué en* $O(n)$ *multiplications.*

Démonstration. L'algorithme 18, page 59, commence par $q \leftarrow \nu_n(C) \cdot (\mathcal{G}^{-1}) \bmod \phi$. D'après le lemme 4.2.2, cette étape nécessite $3n-3$ multiplications.
Ensuite, sur une architecture permettant d'obtenir les $2\log_2(\sigma)$ bits du résultat d'une multiplication de deux opérandes sur $\log_2(\sigma)$ bits, $C' \leftarrow (C - \pi_n(q \cdot \mathcal{G}))/\phi$ nécessite $2 \times (n-1) + 1$ multiplications. Autrement, calculer $q \cdot \mathcal{G}$ peut être effectué avec un découpage haut-bas classique. C'est-à-dire, on pose q_{hi} et q_{lo} tels que $q = \sqrt{\sigma}q_{hi} + q_{lo}$, respectivement $\mathcal{G}_{hi}$ et $\mathcal{G}_{lo}$ tels que $\mathcal{G} = \sqrt{\sigma}\mathcal{G}_{hi} + \mathcal{G}_{lo}$. On effectue alors $q_{lo} \cdot \mathcal{G}_{lo}$, $q_{lo} \cdot \mathcal{G}_{hi}$, $q_{hi} \cdot \mathcal{G}_{lo}$ et $q_{hi} \cdot \mathcal{G}_{hi}$ pour obtenir 4 résultats intermédiaires. Si l'une des opérandes est une partie basse, il est possible que l'accumulation des produits mène à un passage de retenue, auquel cas elles pourront être transmises au résultat de $q_{hi} \cdot \mathcal{G}_{hi}$ en conséquence. Le coût total sera alors de $4 \times (2 \times (n-1) + 1)$ multiplications.
Il faut donc au total soit $5n-4$ multiplications soit $11n-7$ multiplications, ce qui est linéaire en n. □

Remarque 36. L'opération de division par ϕ consiste en un simple décalage binaire car on a supposé $\phi = 2^h$. Sur la plupart des processeurs modernes x86_64, les opérations de décalage binaire sont aussi coûteuses qu'une multiplication [26]. On peut donc potentiellement considérer l'ajout de n multiplications à la complexité pour être plus précis. À noter que pour le cas particulier $\phi = \sigma$, la division peut être remplacée par une instruction MOV sur chacun des coefficients. Le coût de cette instruction est proche d'une addition sur les processeurs modernes x86_64. En ce qui concerne la réduction modulaire par ϕ, comme noté plus haut, on peut effectuer un simple masque binaire sur chaque coefficient dont le coût est similaire à une addition. Encore une fois, dans le cas $\phi = \sigma$, le comportement intrinsèque d'overflow sur les registres se traduit en une réduction modulaire "gratuite".

4.3 PMNS associés aux premiers de la forme $\frac{\alpha\gamma^n-\lambda}{k}$ avec $\alpha \times \gamma < \sigma$

Dans cette section nous considérons tous les PMNS de la forme (p, n, γ, ρ, E) avec $p = \frac{\alpha\gamma^n-\lambda}{k}$ pour $\alpha, \lambda, k \in \mathbb{N}^* \times \mathbb{Z}^* \times \mathbb{Z}^*$. Dans la sous-section précédente, nous avons supposé E quelconque. Or, comme expliqué en section 2.3.1, un certain nombre de formes de E ne conviennent pas à un usage des PMNS en pratique. Pour le restant de ce chapitre, les polynômes E choisis seront de la forme $E(X) = \alpha X^n - \lambda$. Pour tout entier p considéré, puisque $p = \frac{\alpha\gamma^n-\lambda}{k}$, alors $E(\gamma) = kp$. Afin de garantir une réduction interne linéaire, les conditions $\alpha\gamma < \sigma$ et $|\lambda| < \sigma$ sont nécessaires pour vérifier les conditions de la proposition 4.2.4. Cette forme d'entier permet des performances proches de l'arithmétique des entiers pseudo-Mersenne tout en étant bien plus dense.

Remarque 37. Nous prendrons α positif à des fins de simplifications. Pour un PMNS (p, n, γ, ρ, E) avec $E(X) = \alpha X^n - \lambda$, le PMNS $(p, n, -\gamma, \rho, E')$ avec $E'(X) = -\alpha X^n + \lambda$ est un PMNS valide donc considérer tous les PMNS avec α positif englobe tous les PMNS avec α négatif, à transformation près.

Avec ce choix de E, on peut appliquer la proposition 4.2.1 et on a comme base d'un sous-réseau de $\mathfrak{L} = \left\{(x_0, \ldots, x_{n-1}) \in \mathbb{Z}^n : \sum_{i=0}^{n-1} x_i\gamma^i \equiv 0 \pmod{p}\right\}$ la matrice suivante :

$$\mathcal{G} = \begin{pmatrix} -\gamma & 1 & 0 & \dots & \dots & 0 \\ 0 & -\gamma & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & \dots & 0 & -\gamma & 1 \\ \lambda & 0 & \dots & \dots & 0 & -\alpha\gamma \end{pmatrix} \tag{4.2}$$

Remarque 38. Comme on peut le voir, $\|\mathcal{G}\|_1 = \max(\gamma + |\lambda|, \alpha\gamma + 1)$. Ainsi, pour la méthode de réduction de Montgomery (algorithme 18, page 59), il suffit de choisir $\rho = \max(\alpha\gamma, \gamma + |\lambda| - 1)$ pour maintenir la cohérence des opérations (cf. section 2.4.5).

4.3.1 Inversibilité de $\mathcal{G}$ modulo ϕ

Soit $\mathcal{B} = (p, n, \gamma, \rho, E)$ tel que $n > 2$ et $E(X) = \alpha X^n - \lambda$ et soit $\mathcal{G}$ comme décrit dans l'équation (4.2), alors, d'après la proposition 4.2.2, $\det(\mathcal{G}) = (-1)^n E(\gamma)$. Or, $E(\gamma) = kp$ par construction. La matrice $\mathcal{G}$ est inversible modulo ϕ, pour $\phi = 2^h$, si et seulement si $\det(\mathcal{G}) \equiv 1 \pmod 2$. Autrement dit il nous faut $kp \equiv 1 \pmod 2$, ce qui sous-entend k impair. D'après la proposition 4.2.3, on aura alors :

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

Cependant, une possibilité existe dans certains cas pour avoir k pair. Si l'on pose $k = z2^\psi$ pour $z \in \mathbb{Z}^*$ impair et $\psi \in \mathbb{N}$, dans le cas où $\alpha\gamma \equiv 0 \pmod{2^\psi}$ et $\lambda \equiv 0 \pmod{2^\psi}$, on peut considérer la matrice suivante :

$$\tilde{\mathcal{G}} = \begin{pmatrix} -\gamma & 1 & 0 & \dots & \dots & 0 \\ 0 & -\gamma & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & \dots & 0 & -\gamma & 1 \\ \frac{\lambda}{2^\psi} & 0 & \dots & \dots & 0 & \frac{-\alpha\gamma}{2^\psi} \end{pmatrix} \tag{4.3}$$

Cela est équivalent à considérer un E' tel que $E'(X) = \frac{\alpha X^n}{2^\psi} - \frac{\lambda}{2^\psi}$ dans $\mathbb{Q}[X]$ tel que $E'(\gamma) = zp$ pour construire la matrice. On aura bien que $\tilde{\mathcal{G}}$ est une base d'un sous-réseau de $\mathfrak{L}$ donc on peut l'utiliser dans notre réduction interne. Puisque z est impair, $\det(\tilde{\mathcal{G}}) \equiv 1 \pmod 2$ et donc l'inverse existe modulo ϕ.

Étant donné que $\tilde{\mathcal{G}}$ est obtenue à partir de la matrice $\mathcal{G}$ en divisant sa dernière ligne par 2^ψ, nous avons :

$$\mathcal{G}^{-1} \cdot \tilde{\mathcal{G}} = \begin{pmatrix} 1 & 0 & 0 & \dots & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & \dots & 0 & 1 & 0 \\ 0 & 0 & \dots & \dots & 0 & \frac{1}{2^\psi} \end{pmatrix}$$

Ainsi, la matrice inverse de $\tilde{\mathcal{G}}$ sera telle que :

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

ce qui correspond à la matrice $\mathcal{G}^{-1}$ dont la dernière colonne a été multipliée par 2^ψ.

4.3.2 Génération

Pour la génération nous devons d'abord poser les intervalles d'existence des paramètres. Soit ℓ la taille d'entiers que l'on souhaite obtenir. Générer des entiers premiers de la forme $p = \frac{\alpha\gamma^n - \lambda}{k}$ impose

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

comme condition sur ces paramètres. En considérant λ comme négligeable devant $\alpha\gamma^n$, on obtient :

$$k2^{\ell-1} \leqslant \alpha\gamma^n \leqslant k2^\ell$$

en considérant k comme un paramètre de génération. Une fois la valeur de α choisie, l'intervalle d'existence de γ devient alors :

$$\left\lceil (\frac{k}{\alpha} \times 2^{\ell-1})^{\frac{1}{n}} \right\rceil \leqslant \gamma \leqslant \left\lfloor (\frac{k}{\alpha} \times 2^\ell)^{\frac{1}{n}} \right\rfloor \tag{4.4}$$

Une fois γ choisi dans cet intervalle, il faut considérer l'intervalle de recherche pour trouver un nombre premier p sur lequel on pourra construire un PMNS. Étant donné que les valeurs de α et k sont des paramètres de la génération, nous pouvons considérer un intervalle de recherche de $[\![\lceil \frac{\alpha\gamma^n - \lambda_{\max}}{k} \rceil, \lfloor \frac{\alpha\gamma^n + \lambda_{\max}}{k} \rfloor]\!]$ avec $\lambda_{\max}$ une borne supérieure sur la valeur de λ garantissant que les bornes intervenant dans la réduction interne utilisée sont satisfaites.

Puisque nous utilisons la réduction interne Montgomery-like (algorithme 18, page 59), un PMNS est valide s'il vérifie $2w(\rho-1) < \phi$ (voir section 2.4.4). On fixe $\phi = \sigma$ pour cette construction. Ensuite, on a $w = \max(\alpha n, |\lambda|(n-1)+\alpha)$ pour $E(X) = \alpha X^n - \lambda$ comme vu en section 2.3.2. Quant à ρ, comme expliqué en section 2.4.5, on peut le choisir comme $\rho = \|\mathcal{G}\|_1 - 1$ (avec le $\mathcal{G}$ de l'équation (4.2)) en remarquant que $\mathcal{G}$ est bien la base d'un sous-réseau de $\mathfrak{L} = \left\{(x_0, \ldots, x_{n-1}) \in \mathbb{Z}^n : \sum_{i=0}^{n-1} x_i \gamma^i \equiv 0 \pmod{p}\right\}$. Comme noté dans la remarque 38, pour $E(X) = \alpha X^n - \lambda$, on aura $\|\mathcal{G}\|_1 = \max(\gamma + |\lambda|, \alpha\gamma + 1)$.

On en déduit que $\lambda_{\max}$ est la valeur dans $\mathbb{R}$ telle que

$$2(\max(\alpha n, \alpha + \lambda_{\max}(n-1)))(\max(\gamma + \lambda_{\max}, \alpha\gamma + 1) - 1) = \phi - 1 \tag{4.5}$$

Ainsi $\forall \lambda \in \mathbb{Z}$ tel que $|\lambda| \leqslant \lambda_{\max}$, les bornes seront respectées. Remarquons que

$$\max(\alpha n, \alpha + \lambda_{\max}(n-1)) = \begin{cases} \alpha n & \text{si } \alpha \geqslant \lambda_{\max} \\ \alpha + \lambda_{\max}(n-1) & \text{si } \alpha \leqslant \lambda_{\max}. \end{cases}$$

De plus, en considérant que $\gamma > \lambda_{\max}$,

$$\max(\gamma + \lambda_{\max}, \alpha\gamma + 1) = \begin{cases} \alpha\gamma + 1 & \text{si } \alpha > 1 \\ \gamma + \lambda_{\max} & \text{si } \alpha = 1. \end{cases}$$

Pour $2 \leqslant \alpha \leqslant \lambda_{\max}$, l'équation (4.5) devient

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

d'où

$$\lambda_{\max} = \frac{\phi - 2\alpha(\alpha\gamma - 1) - 1}{2(n-1)(\alpha\gamma - 1)} \tag{4.6}$$

Pour le cas $\alpha = 1$, l'équation (4.5) devient

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

un polynôme du second degré en $\lambda_{\max}$. D'où

$$\lambda_{\max} = \frac{(\gamma-2)}{2}\left(\sqrt{1 + \frac{2\phi - 2\gamma + 2}{(n-1)(\gamma-2)^2} + \frac{1}{((n-1)(\gamma-2))^2}} - 1\right) - \frac{1}{2n-2} \tag{4.7}$$

Remarque 39. Dans le cas $\alpha \geqslant \lambda_{\max}$, notre borne sur ϕ devient

$$2\alpha^2 n\gamma < \phi$$

ce qui n'impose aucune condition sur λ, outre le fait que $\alpha \geqslant \lambda$. Nous ne considérerons pas ce cas pour deux raisons. Premièrement, puisque $\lambda_{\max}$ correspond à une borne supérieure sur le paramètre λ, imposer $\lambda_{\max} \leqslant \alpha$ va à l'encontre du principe, surtout en considérant que le paramètre α est ici constant. Deuxièmement, w

grandit plus vite lorsque α augmente que lorsque λ augmente et nous voulons garder w le plus petit possible, donc supposer $\lambda_{\max} \geqslant \alpha$ est raisonnable.

Pour $\alpha > 1$, l'expression de $\lambda_{\max}$ indique que cette valeur est inversement proportionnelle à γ. L'expression pour $\alpha = 1$ est moins lisible. Si l'on suppose $\phi < \gamma^2$, un développement limité de l'équation (4.7) est :

$$\begin{aligned} DL_{\frac{2\phi-2\gamma+2}{(n-1)(\gamma-2)^2}\to 0}(\lambda_{\max}) &= \frac{\gamma-2}{2}\left(1+\frac{2\phi-2\gamma+2}{2(n-1)(\gamma-2)^2}-\frac{(2\phi-2\gamma+2)^2}{8(n-1)^2(\gamma-2)^4}+o(\frac{(2\phi-2\gamma+2)^2}{(n-1)^2(\gamma-2)^4})-1\right) \\ &= \frac{\phi-\gamma+1}{2(n-1)(\gamma-2)}-\frac{(\phi-\gamma+1)^2}{4(n-1)^2(\gamma-2)^3}+o(\frac{2(\phi-\gamma+1)^2}{(n-1)^2(\gamma-2)^3}) \end{aligned}$$

Si l'on suppose que $\gamma^3 > \phi^2$, il est clair que $\lambda_{\max}$ est également inversement proportionnel à γ pour $\alpha = 1$. Ainsi, pour déterminer si un PMNS existe pour un jeu de paramètres donné, nous devons calculer $\lambda_{\max}$ pour le plus petit γ de l'intervalle d'existence détaillé plus haut. Si cette valeur est strictement inférieure à 1, aucun PMNS ne peut exister avec ce jeu de paramètres.

Une autre considération est le paramètre δ. Il sera parfois nécessaire de générer un PMNS vérifiant $\delta \geqslant \delta_{\min}$ pour un certain $\delta_{\min} \in \mathbb{N}$ selon le cas d'utilisation. Ainsi, cela doit être pris en compte lors de la génération.

Pour finir, une fois un p trouvé tel que $p = \frac{\alpha\gamma^n-\lambda}{k}$ avec $|\lambda| < \lambda_{\max}$, si k est pair il faut appliquer la méthode détaillée plus haut qui consiste à prendre pour base la matrice de l'équation (4.3). Ceci n'est possible que si $\mathrm{PGCD}(\frac{k}{\mathrm{PGCD}(k,\lambda,\alpha\gamma)},\phi) = 1$ pour les raisons évoquées plus haut. Ainsi, une dernière vérification est effectuée pour vérifier que cela est bien le cas.

On obtient ainsi l'algorithme de génération suivant :

Algorithme 29 Génération de PMNS avec réduction interne linéaire

Require: ℓ la taille d'entiers premiers considérés, n le nombre de coefficients polynomiaux, $\phi = 2^h$, k, α et $\delta_{\min}$ les paramètres du système

Ensure: un PMNS $(p, n, \gamma, \rho, E = \alpha X^n - \lambda, \delta)$ avec réduction interne en temps linéaire tel que $k \times p = \alpha\gamma^n - \lambda$ si les bornes sont valides sinon 0

1: Calculer $\lambda_{\max}$ en prenant $\gamma = \left\lceil (\frac{k}{\alpha} \times 2^{\ell-1})^{\frac{1}{n}} \right\rceil$, cf. équations (4.6) et (4.7)
2: **if** $\lambda_{\max} < 1$ **then**
3: **return** 0
4: **end if**
5: **while** True **do**
6: Choisir γ de façon aléatoire tel que $\left\lceil (\frac{k}{\alpha} \times 2^{\ell-1})^{\frac{1}{n}} \right\rceil \leqslant \gamma \leqslant \left\lfloor (\frac{k}{\alpha} \times 2^{\ell})^{\frac{1}{n}} \right\rfloor$
7: Calculer $\lambda_{\max}$, cf. équations (4.6) et (4.7)
8: **if** $\lambda_{\max} \geqslant 1$ **then**
9: $p \leftarrow \text{next\_prime}(\lfloor \frac{\alpha\gamma^n - \lambda_{\max}}{k} \rfloor)$
10: $\lambda \leftarrow \alpha\gamma^n - kp$
11: Calculer δ, cf. équation (2.9), page 66
12: **if** $|\lambda| \leqslant \lambda_{\max}$ **and** $\text{PGCD}(\frac{k}{\text{PGCD}(k, \lambda, \alpha\gamma)}, \phi) = 1$ **and** $\delta \geqslant \delta_{\min}$ **then**
13: $\rho \leftarrow \max(\gamma + |\lambda|, \alpha\gamma + 1) - 1$
14: $E \leftarrow \alpha X^n - \lambda$
15: **return** $(p, n, \gamma, \rho, E, \delta)$
16: **end if**
17: **end if**
18: **end while**

Remarque 40. Une alternative à choisir γ de façon aléatoire consiste à parcourir l'intervalle de façon séquentielle jusqu'à trouver un PMNS valide mais le résultat ne sera alors pas aléatoire.

En ce qui concerne le nombre d'entiers premiers possédant cette forme particulière menant à une génération de PMNS fructueuse, nous devons quantifier le nombre d'entiers premiers dans l'intervalle $[\![\lceil \frac{\alpha\gamma^n - \lambda_{\max}}{k} \rceil, \lfloor \frac{\alpha\gamma^n + \lambda_{\max}}{k} \rfloor]\!]$. Il s'agit d'un intervalle d'au plus $\frac{2\lambda_{\max}}{k}$ valeurs possibles. Soient p_1 et p_2 deux entiers premiers consécutifs. Le Théorème des nombres premiers [29, 39] nous donne qu'en moyenne $p_2 - p_1 \approx \ln(p_1)$ lorsque p_1 est assez grand. Dans un contexte cryptographique c'est toujours le cas.

Si l'on considère des entiers premiers de ℓ bits, alors l'écart moyen entre deux nombres premiers est de $\ln(1.5 \times 2^{\ell-1})$. Ainsi, un intervalle de $\frac{2\lambda_{\max}}{k}$ valeurs possibles contiendra en moyenne $\frac{2\lambda_{\max}}{k \ln(1.5 \times 2^{\ell-1})}$ entiers premiers. En guise d'exemple, si l'on considère les entiers premiers de 256 bits avec $n = 5$, $k = 1$ et $\alpha = 1$, $\lambda_{\max}$ a pour valeur 952.75 en choisissant γ au milieu de l'intervalle des 3348928077626 γ possibles. Lorsque l'on évalue $\frac{k}{2} \ln(1.5 \times 2^{\ell-1})$, on obtient environ 88.57 donc on peut s'attendre à obtenir $952.75/88.57 \approx 10.75$ PMNS valides pour chaque γ en moyenne. Empiriquement, nous trouvons en moyenne environ 10.78 nombres premiers permettant de construire un PMNS valide pour chaque γ avec un tirage aléatoire suivant une distribution uniforme, ce qui confirme notre estimation.

4.3.3 Matrices inverses creuses

Avec des restrictions supplémentaires sur γ, on peut obtenir des matrices inverses très creuses. Par exemple, en posant $\gamma^2 \equiv 0 \pmod{\phi}$, on obtient :

$$\mathcal{G}^{-1} \equiv \begin{pmatrix} 0 & 0 & 0 & \dots & 0 & 0 & \frac{-\alpha\gamma}{kp} & \frac{-1}{kp} \\ 1 & 0 & 0 & \dots & 0 & 0 & 0 & \frac{-\gamma}{kp} \\ \gamma & 1 & 0 & \dots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & \gamma & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & \gamma & 1 \end{pmatrix} \pmod{\phi}$$

En effet, $\gamma^i \equiv 0 \pmod{\phi}$ pour $i \geqslant 2$ donc tous les coefficients de la forme γ^i pour $i \geqslant 2$ s'annulent lorsque la matrice est réduite modulo ϕ. De plus, $kp = \alpha\gamma^n - \lambda$ donc $kp \equiv -\lambda \pmod{\phi}$ pour $n \geqslant 2$ d'où $\frac{-\lambda}{kp} \equiv 1 \pmod{\phi}$. Ainsi tous les coefficients de la forme $\frac{-\lambda}{kp}$ sont égaux à 1 une fois la matrice réduite modulo ϕ. De façon générale, prendre $\gamma^\beta \equiv 0 \pmod{\phi}$ donnera des matrices inverses à β coefficients par ligne et par colonne modulo ϕ. Il est possible d'ajuster l'algorithme 29 en conséquence pour obtenir ces matrices inverses particulières en choisissant γ en conséquence.

4.4 Dérivation de nouveaux PMNS à partir de PMNS existants

Dans cette section, nous proposons des transformations à appliquer à des PMNS à réduction interne linéaire existants afin d'obtenir de nouveaux PMNS qui conservent la linéarité de la réduction interne, élargissant ainsi le nombre de PMNS de formes intéressantes sur la classe d'entiers $p = \frac{\alpha\gamma^n - \lambda}{k}$. Ces nouveaux PMNS dérivés de PMNS existants offrent de par ailleurs la possibilité de convertir un représentant d'un PMNS en un représentant d'un autre PMNS construit sur le même p de façon efficace, ce qui peut être intéressant du point de vue des contre-mesures contre les attaques par canaux cachés par exemple.

4.4.1 Transformation puissance

Cette transformation consiste à prendre γ^i au lieu de γ pour un certain $i > 1$.

Proposition 4.4.1. *Soit $\mathcal{B} = (p, n, \gamma, \rho, E(X) = \alpha X^n - \lambda)$ un PMNS avec $\alpha\gamma < \sigma$ et $|\lambda| < \sigma$, alors le PMNS $\mathcal{B}' = (p, n, \gamma^i, \rho' = \max(\gamma|\lambda|^{i-1} + |\lambda|, \alpha\gamma + \alpha^{i-1}) - 1, E'(X) = \alpha^i X^n - \lambda^i)$ pour un certain $i \in \mathbb{N}^*$ est valide tant que $2 \times (\max(\alpha^i n, \alpha^i + |\lambda^i|(n-1)))\rho' < \phi$. De plus, si $i \mid (n-1)$ et $|\lambda^{i-1}\gamma| < \sigma$, alors la réduction interne est linéaire.*

Démonstration. Par construction on a :

$$\alpha\gamma^n \equiv \lambda \pmod{p}$$
$$\implies \alpha^i(\gamma^i)^n - \lambda^i \equiv 0 \pmod{p}$$

Donc E' vaut bien 0 en γ^i modulo p.

Si $i \mid (n-1)$ alors $\alpha\gamma X^{\frac{n-1}{i}} - \lambda$ vaut 0 en γ^i mod p. On peut alors remplir les $n - \frac{n-1}{i}$ premières lignes de la matrice de base avec les vecteurs $(\lambda, 0, \dots, 0, -\alpha\gamma, 0, \dots, 0)$, $(0, \lambda, 0, \dots, 0, -\alpha\gamma, 0, \dots, 0)$, ... , $(0, \dots, 0, \lambda, 0, \dots, 0, -\alpha\gamma)$. Les $\frac{n-1}{i}$ lignes restantes seront alors remplies en appliquant l'isomorphisme ν_n aux multiples de $\alpha^{i-1}X^{n-\frac{n-1}{i}} - \lambda^{i-1}\gamma$ qui s'annulent aussi en γ^i modulo p ce qui nous donne les vecteurs

$(\lambda^{i-1}\gamma, 0, \ldots, 0, -\alpha^{i-1}, 0, \ldots, 0)$, $(0, \lambda^{i-1}\gamma, 0, \ldots, 0, -\alpha^{i-1}, 0, \ldots, 0)$, ... , $(0, \ldots, 0, \lambda^{i-1}\gamma, 0, \ldots, 0, -\alpha^{i-1})$ (cf. exemple ci-après).

La matrice de base est ainsi très creuse et sa norme 1 vaut $\max(\gamma|\lambda|^{i-1} + |\lambda|, \alpha\gamma + \alpha^{i-1})$ ce qui entraîne $\rho' = \max(\gamma|\lambda|^{i-1} + |\lambda|, \alpha\gamma + \alpha^{i-1}) - 1$. □

Remarque 41. Pour n impair, on peut donc toujours prendre $i = 2$ si les bornes le permettent car $2 \mid (n-1)$.

La conversion d'un élément de $\mathcal{B}$ vers un élément de $\mathcal{B}'$ et vice versa est très simple et rapide lorsque $\alpha = 1$. En effet soit $A(X) = a_0 + a_1X + \cdots + a_{n-1}X^{n-1}$ un élément de $\mathcal{B}$. On pose $a \in \mathbb{Z}/p\mathbb{Z}$ tel que $A(\gamma) \equiv a \pmod p$. On a donc

$$a_0 + a_1\gamma + \cdots + a_{n-1}\gamma^{n-1} \equiv a \pmod p.$$

On cherche $A' \in \mathcal{B}'$ tel que $A'(\gamma') \equiv a \pmod p$ avec $\gamma' = \gamma^i$ pour un certain i qui divise $n-1$. Autrement dit on cherche $(a'_0, a'_1, \ldots, a'_{n-1})$ tels que $a'_0 + a'_1\gamma^i + \ldots a'_{n-1}\gamma^{i(n-1)} \equiv a \pmod p$. On a donc

$$a_0 + a_1\gamma + \cdots + a_{n-1}\gamma^{n-1} \equiv a'_0 + a'_1\gamma^i + \ldots a'_{n-1}\gamma^{i(n-1)} \pmod p$$

On peut déjà poser $a'_0 = a_0$, $a'_1 = a_i$, $a'_2 = a_{2i}$, et ainsi de suite. On aura $a'_j = a_{j\times i}$ pour j allant de 0 à $\frac{n-1}{i}$ car i divise $(n-1)$. Ensuite, puisque $\gamma^n \equiv \lambda \pmod p$, alors $\forall j$, $(\gamma^i)^j \equiv \lambda^{\lfloor \frac{ij}{n} \rfloor}\gamma^{ij \bmod n}$. Il suffit donc de trouver les coefficients a'_j tels que $a'_j\lambda^{\lfloor \frac{ij}{n} \rfloor} = a_{ij \bmod n}$.

On peut donc donner la forme générale $a'_j = \frac{a_{ij \bmod n}}{\lambda^{\left\lfloor \frac{j\times i}{n} \right\rfloor}}$.

On note que les coefficients de A ne sont pas forcément divisibles par des puissances de λ. Cependant, on peut toujours ajouter un multiple de $X^j - \gamma X^{j-1}$ pour forcer qu'un coefficient voulu soit divisible. En effet, $X^j - \gamma X^{j-1}$ s'annule en γ donc on peut l'ajouter ou le soustraire autant de fois que voulu à A sans changer la valeur d'évaluation en γ du résultat. On rappelle que l'on a $\rho' = \max(\gamma|\lambda|^{i-1} + |\lambda|, \alpha\gamma + \alpha^{i-1}) - 1$ et on a supposé ici $\alpha = 1$ donc $\rho' \approx |\lambda^{i-1}|\rho$, en remarquant que $\rho = \gamma + |\lambda| - 1$ et que λ est très négligeable devant γ. Ainsi, puisque $\rho' \approx |\lambda|^{i-1}\rho$, rajouter au plus $\frac{|\lambda^{i-1}|}{2}$ fois γ à un coefficient en valeur absolue ne dépassera pas les bornes de $\mathcal{B}'$ et ceci est suffisant pour garantir la divisibilité.

À titre d'exemple, on considère le PMNS suivant à partir duquel on peut construire un nouveau PMNS avec la transformation puissance en prenant $\gamma' = \gamma^2$:
$p = 751257011608166639457727183574986928486644251508058756533606535646125741834 21$,
$\gamma = 2372234991632384 = 239 \times 2311 \times 2^{32}$, avec $\phi = 2^{64}$, on a $p = \gamma^5 - 3$. On note que l'on a alors $E(X) = X^5 - 3$.
En choisissant $\gamma' = 562749885552509698649699752345 6 = \gamma^2$, on a bien $\gamma' - \gamma^2 \equiv 0 \pmod p$ et $\gamma'^5 \equiv 9 \pmod p$ donc on peut prendre $E'(X) = X^5 - 9$. Les matrices de réduction interne du nouveau PMNS sont $\mathcal{G} =$

$$\begin{pmatrix} 3 & 0 & -2372234991632384 & 0 & 0 \\ 0 & 3 & 0 & -2372234991632384 & 0 \\ 0 & 0 & 3 & 0 & -2372234991632384 \\ 7116704974897152 & 0 & 0 & -1 & 0 \\ 0 & 7116704974897152 & 0 & 0 & -1 \end{pmatrix}$$

et $\mathcal{G}^{-1} \pmod{\phi} \equiv$

$$\begin{pmatrix} 12297829382473034411 & 0 & 14347731194550943744 & 0 & 0 \\ 0 & 12297829382473034411 & 0 & 12297038637475823616 & 0 \\ 0 & 0 & 12297829382473034411 & 0 & 12297038637475823616 \\ 2372234991632384 & 0 & 0 & 18446744073709551615 & 0 \\ 0 & 2372234991632384 & 0 & 0 & 18446744073709551615 \end{pmatrix}$$

et le tuple $(p, n, \gamma', \rho' = 7116704974897154, E')$ correspond bien à un PMNS.

Remarque 42. Il s'agit ici de la matrice de base ayant la forme détaillée dans la proposition 4.4.1 et non celle de l'équation (4.1).

4.4.2 Transformation racine

Cette transformation consiste à prendre $\gamma^{\frac{1}{i}}$ dans $\mathbb{Z}/p\mathbb{Z}$ au lieu de γ. Cela est possible seulement si γ n'a pas de racine $i$ème dans $\mathbb{Z}$. Cette transformation n'est évidemment possible que si $\gamma^{\frac{1}{i}}$ existe dans $\mathbb{Z}/p\mathbb{Z}$. On sait d'après Gauss [24] que γ est un résidu $i$ème modulo p si et seulement si $\gamma^{\frac{p-1}{\text{PGCD}(p-1,i)}} \equiv 1 \pmod{p}$. Il est donc facile de déterminer pour un γ donné si cette transformation est possible ou non.

Proposition 4.4.2. *Soit $\mathcal{B} = (p, n, \gamma, \rho, E(X) = \alpha X^n - \lambda)$ un PMNS avec $\alpha\gamma < \sigma$ et $|\lambda| < \sigma$ tel que $\alpha^{\frac{1}{i}} \in \mathbb{Z}$ et $\lambda^{\frac{1}{i}} \in \mathbb{Z}$ et $\gamma^{\frac{1}{i}} \notin \mathbb{Z}$, alors, le PMNS $\mathcal{B}' = (p, n, \gamma^{\frac{1}{i}} \pmod{p}, \rho' = \max(\sqrt[i]{\alpha}\gamma + 1, \gamma + \sqrt[i]{\lambda}) - 1, E'(X) = \sqrt[i]{\alpha}X^n - \sqrt[i]{\lambda})$ est valide. De plus, la réduction interne a une complexité linéaire.*

Démonstration. On a

$$\begin{aligned} & \alpha\gamma^n - \lambda \equiv 0 \pmod{p} \\ \iff & \alpha\gamma^n \equiv \lambda \pmod{p} \\ \iff & \sqrt[i]{\alpha}\gamma^{\frac{n}{i}} \equiv \sqrt[i]{\lambda} \pmod{p} \\ \iff & \sqrt[i]{\alpha}(\gamma^{\frac{1}{i}})^n - \sqrt[i]{\lambda} \equiv 0 \pmod{p} \end{aligned}$$

Donc E' s'annule bien en $\gamma^{\frac{1}{i}}$ modulo p.

On peut construire la matrice de base en remplissant les $n-i$ premières lignes avec des polynômes de la forme $(X^i - \gamma)X^j$ pour j allant de 0 à $n-1-i$ auxquels on applique l'isomorphisme ν_n, avant de compléter par des polynômes de la forme $(\sqrt[i]{\alpha}\gamma X^{n-i} - \sqrt[i]{\lambda})X^j$ pour j allant de 0 à $i-1$ auxquels on applique l'isomorphisme ν_n. Cette matrice aura pour norme 1 $\max(\sqrt[i]{\alpha}\gamma + 1, \gamma + \sqrt[i]{\lambda})$ donc on a bien $\rho' = \max(\sqrt[i]{\alpha}\gamma + 1, \gamma + \sqrt[i]{\lambda}) - 1$.

De plus, cette matrice ne possède que 2 coefficients par ligne et par colonne donc la multiplication vecteur-matrice est bien de complexité linéaire (cf. exemple ci-après). □

Remarque 43. Nous notons ici $\sqrt[i]{\alpha}$ et $\sqrt[i]{\lambda}$ mais $\gamma^{\frac{1}{i}}$ pour distinguer le fait que α et λ sont des puissances $i$èmes dans $\mathbb{Z}$ et ce sont donc des racines dans $\mathbb{Z}$ qui sont ici prises, en contraste avec $\gamma^{\frac{1}{i}}$ qui correspond à une valeur prise dans $\mathbb{Z}/p\mathbb{Z}$.

Remarque 44. Imposer à α et λ d'être des puissances $i$ème dans $\mathbb{Z}$ vient du fait que l'on veut $\alpha^{\frac{1}{i}}$ et $\lambda^{\frac{1}{i}}$ relativement petits et prendre une racine $i$ème dans $\mathbb{Z}/p\mathbb{Z}$ donne une valeur aux environs de p ce qui est beaucoup trop grand. En particulier pour $\alpha = 1$, cela ne nécessite que de trouver un λ qui est une puissance

naturelle dans $\mathbb{Z}$. De plus, on a alors $E'(X) = X^n - \sqrt[i]{\lambda}$ donc le nouveau w a pour valeur $\sqrt[i]{\lambda}(n-1)+1$, ce qui est plus intéressant que le w du PMNS à partir duquel nous avons effectué la construction qui a pour valeur $|\lambda|(n-1)+1$.

Un processus de conversion rapide pour passer d'un élément de $\mathcal{B}$ à un élément de $\mathcal{B}'$ lorsque $\alpha = 1$ et vice versa existe, mais pour en avoir un aussi simple que celui de la forme puissance d'au-dessus, il faut rajouter des conditions sur i. Plus précisément, il faut que $n \bmod i$ soit un générateur du groupe $(\mathbb{Z}/i\mathbb{Z}, +)$ (c'est-à-dire, il faut que n soit premier avec i). Cela vient du fait que

$$\gamma^{\frac{n+n \bmod i}{i}} \equiv \sqrt[i]{\lambda}\gamma^{\frac{n \bmod i}{i}} \pmod{p}.$$

Si $n \bmod i$ est générateur, on peut alors faire correspondre tous les coefficients d'un polynôme $A(X) = a_0 + a_1X + a_2X^2 + \cdots + a_{n-1}X^{n-1}$ dans $\mathcal{B}$ aux coefficients d'un polynôme A' dans $\mathcal{B}'$. En effet, on cherche les coefficients a'_j pour $j \in [\![0, n-1]\!]$ tels que

$$a'_0 + a'_1\gamma^{\frac{1}{i}} + a'_2\gamma^{\frac{2}{i}} + \cdots + a'_{n-1}\gamma^{\frac{n-1}{i}} \equiv a_0 + a_1\gamma + a_2\gamma^2 + \cdots + a_{n-1}\gamma^{n-1} \pmod{p}.$$

Les coefficients de la forme a'_0, a'_i, a'_{2i}, etc. correspondront évidemment aux coefficients de la forme a_0, a_1, a_2 etc. Pour les autres coefficients, nous avons noté plus haut que

$$\gamma^{\frac{n+n \bmod i}{i}} \equiv \sqrt[i]{\lambda}\gamma^{\frac{n \bmod i}{i}} \pmod{p}.$$

Ainsi, on peut poser

$$a'_{\frac{n \bmod i}{i}} = \sqrt[i]{\lambda}a_{\frac{n+n \bmod i}{i}}.$$

Il en découle que

$$a'_{2\frac{n \bmod i}{i}} = \sqrt[i]{\lambda}a_{\frac{n+2n \bmod i}{i}}$$

et ainsi de suite. On en déduit la forme générale pour la suite :

$$a'_j = (\sqrt[i]{\lambda})^{\frac{j}{n} \bmod i} a_{\lfloor \frac{n-1}{i} \rfloor + (\frac{j}{n} \bmod i)}$$

À titre d'exemple, on considère le PMNS suivant à partir duquel on peut construire un nouveau PMNS avec la transformation racine en prenant $\gamma' = \gamma^{\frac{1}{2}}$:
$p = 57914267522859713216204992113291927293174550089940125045933252694706095127951$,
$\gamma = 2251941547606016 = 7 \times 74903 \times 2^{32}$, avec $\phi = 2^{64}$, on a $p = \gamma^5 - 625$. On note que l'on a alors $E(X) = X^5 - 625$.
En choisissant $\gamma' = 92662828036575541145927987381271651724161853427116885402178757031539307969 56 = \gamma^{\frac{1}{2}}$ on a bien $\gamma'^2 - \gamma \equiv 0 \pmod{p}$ et $\gamma'^5 \equiv 25 \pmod{p}$ donc on peut prendre $E'(X) = X^5 - 25$. Les matrices de réduction interne du nouveau PMNS sont $\mathcal{G} =$

$$\begin{pmatrix} -2251941547606016 & 0 & 1 & 0 & 0 \\ 0 & -2251941547606016 & 0 & 1 & 0 \\ 0 & 0 & -2251941547606016 & 0 & 1 \\ 25 & 0 & 0 & -2251941547606016 & 0 \\ 0 & 25 & 0 & 0 & -2251941547606016 \end{pmatrix}$$

et $\mathcal{G}^{-1} \pmod{\phi} \equiv$

$$\begin{pmatrix} 0 & 17708964388423073792 & 0 & 10330176681277348905 & 0 \\ 0 & 0 & 17708964388423073792 & 0 & 10330176681277348905 \\ 1 & 0 & 0 & 17708964388423073792 & 0 \\ 0 & 1 & 0 & 0 & 17708964388423073792 \\ 2251941547606016 & 0 & 1 & 0 & 0 \end{pmatrix}$$

et le tuple $(p, n, \gamma', \rho' = 2251941547606040, E')$ correspond bien à un PMNS.

Remarque 45. Il s'agit ici de la matrice de base ayant la forme détaillée dans la proposition 4.4.2 et non celle de l'équation (4.1).

4.4.3 Transformation réciproque

Cette transformation consiste à considérer le polynôme réciproque de E.

Proposition 4.4.3. *Soit $\mathcal{B} = (p, n, \gamma, \rho, E(X) = \alpha X^n - \lambda)$ un PMNS avec $\alpha\gamma < \sigma$ et $\lambda < \sigma$, alors le PMNS $\mathcal{B}' = (p, n, \gamma' = \gamma^{-1} \pmod{p}, \rho, E'(X) = \lambda X^n - \alpha)$ est valide et possède une réduction interne à complexité linéaire.*

Démonstration. Si $\alpha\gamma^n - \lambda \equiv 0 \pmod{p}$ alors $\gamma^{-n} \times (\alpha\gamma^n - \lambda) \equiv \alpha - \lambda\gamma^{-n} \equiv 0 \pmod{p}$. D'où $E'(\gamma') \equiv \lambda\gamma'^n - \alpha \equiv 0 \pmod{p}$.
Une propriété des polynômes réciproques est que si γ est une racine d'un polynôme P alors γ^{-1} est une racine de son polynôme réciproque P'. Alors pour construire une matrice de base des polynômes qui s'annulent en γ^{-1} il suffit de considérer les polynômes réciproques de chacune des lignes de la base auxquels on pourra appliquer l'isomorphisme ν_n. Si l'on note $\mathcal{G}'$ cette base des polynômes de degré au plus n qui s'annulent en γ^{-1} modulo p, alors on aura bien $\|\mathcal{G}'\|_1 = \|\mathcal{G}\|_1$ et donc on peut prendre la même valeur de ρ pour ce nouveau PMNS. □

Remarque 46. La valeur de w devient $\max(|\lambda|n, |\lambda| + \alpha(n-1))$ et donc est modifiée si $\alpha \neq |\lambda|$. En conséquence, il est possible que le changement de la valeur de w entraîne un dépassement des bornes pour $\mathcal{B}'$ alors que les bornes de $\mathcal{B}$ étaient valides.

Pour la conversion d'un élément A de $\mathcal{B}$ vers un élément A' de $\mathcal{B}'$, on pose d'abord le lemme suivant :

Lemme 4.4.1. *Soit $\gamma \in \mathbb{Z}/p\mathbb{Z}$, et $\gamma' \in \mathbb{Z}/p\mathbb{Z}$ tels que $\gamma\gamma' \equiv 1 \pmod{p}$. Soit $A \in \mathbb{Z}[X]$ tel que $A(\gamma) \equiv a \pmod{p}$, alors A', le polynôme réciproque de A est tel que $A'(\gamma') \equiv \gamma'^{n-1}A(\gamma) \pmod{p}$.*

Démonstration. Soit $A(X) = a_0 + a_1X + \cdots + a_{n-2}X^{n-2} + a_{n-1}X^{n-1}$, on a :

$$A(\gamma) = a_0 + a_1\gamma + \cdots + a_{n-2}\gamma^{n-2} + a_{n-1}\gamma^{n-1} = a + z \times p, z \in \mathbb{Z}$$

$$\gamma'^{n-1}A(\gamma) = a_0\gamma'^{n-1} + a_1\gamma'^{n-1}\gamma + \cdots + a_{n-2}\gamma'^{n-1}\gamma^{n-2} + a_{n-1}\gamma'^{n-1}\gamma^{n-1} = \gamma'^{n-1}a + z' \times p, z' \in \mathbb{Z}$$

$$\gamma'^{n-1}A(\gamma) \equiv a_0\gamma'^{n-1} + a_1\gamma'^{n-2} + \cdots + a_{n-2}\gamma' + a_{n-1} \equiv \gamma'^{n-1}a \pmod{p}$$

Soit $A'(X) = a_0X^{n-1} + a_1X^{n-2} + \cdots + a_{n-2}X + a_{n-1}$, le polynôme réciproque de $A(X)$, on a que :

$$A'(\gamma') = a_0\gamma'^{n-1} + a_1\gamma'^{n-2} + \cdots + a_{n-2}\gamma' + a_{n-1} = \gamma'^{n-1}a + z' \times p, z' \in \mathbb{Z}$$

D'où $A'(\gamma') \equiv \gamma'^{n-1}A(\gamma) \pmod{p}$ □

Grâce à ce lemme, on sait que le polynôme réciproque d'un représentant de a dans un PMNS est un représentant de $\gamma'^{n-1}a$ dans le PMNS réciproque correspondant. Ainsi, si l'on peut facilement trouver un représentant de $\gamma^{n-1}a$ dans le PMNS d'origine, on aurait une façon simple d'obtenir un représentant de a dans le PMNS réciproque puisque $\gamma' \times \gamma \equiv 1 \pmod{p}$. Cela nous mène à la proposition suivante :

Proposition 4.4.4. *Soit $\mathcal{B} = (p, n, \gamma, \rho, E)$ un PMNS avec E unitaire, soit $A(X) \in \mathcal{B}$ tel que $A(\gamma) \equiv a \pmod{p}$, soit $V(X) \equiv X^{n-1}A(X) \pmod{E(X)}$, alors V' le polynôme réciproque de V est tel que $V'(\gamma') \equiv a \pmod{p}$, donc un représentant de a dans le PMNS réciproque.*

Démonstration. Soit $V(X) \in \mathbb{Z}[X]$ tel que

$$V(X) \equiv X^{n-1}A(X) \pmod{E(X)}.$$

Ainsi, $\exists P \in \mathbb{Z}[X]$ tel que

$$V(X) = X^{n-1}A(X) + P(X)E(X).$$

Alors,

$$V(\gamma) = \gamma^{n-1}A(\gamma) + P(\gamma)E(\gamma).$$

Or, d'après le lemme 4.4.1,

$$V'(\gamma') \equiv \gamma'^{n-1}V(\gamma) \pmod{p}.$$

Donc

$$V'(\gamma') \equiv \gamma'^{n-1}(\gamma^{n-1}A(\gamma) + P(\gamma)E(\gamma)) \pmod{p}.$$

Puisque $E(\gamma) \equiv 0 \pmod{p}$,

$$V'(\gamma') \equiv \gamma'^{n-1}\gamma^{n-1}A(\gamma) \pmod{p}.$$

De plus, $\gamma'^{n-1}\gamma^{n-1} \equiv 1 \pmod{p}$ d'où

$$V'(\gamma') \equiv A(\gamma) \equiv a \pmod{p}.$$

□

La multiplication par X^{n-1} modulo $E(X)$ peut s'avérer très peu coûteuse selon la forme de E et peut ne pas requérir de réduction interne. Par exemple, soit $A \in \mathbb{Z}_{n-1}[X]$ que l'on écrit $A(X) = a_0 + a_1X + \cdots + a_{n-1}X^{n-1}$. Si $E(X) = X^n - \lambda$, $\lambda \in \mathbb{Z}^*$, le calcul de $A(X)X^{n-1} \pmod{E(X)}$ peut être effectué en prenant le polynôme $\lambda a_1 + \lambda a_2 X + \cdots + \lambda a_{n-1}X^{n-2} + a_0X^{n-1}$ pour résultat. Ce calcul demande donc $n-1$ multiplications par λ et n instructions MOV. Pour le cas particulier $E(X) = X^n \pm 2$, les $n-1$ multiplications par λ peuvent être remplacées par $n-1$ additions de coefficients. Selon la valeur de δ, ce processus de conversion peut potentiellement être effectué sans nécessiter une réduction interne. Par exemple, pour $E(X) = X^n \pm 2$ et $\delta = 1$, si aucune addition supplémentaire n'est nécessaire après conversion, alors effectuer une multiplication par un élément du PMNS ne mènera pas à un dépassement des bornes et effectuer une réduction interne au préalable ne sera pas nécessaire. En revanche, pour $E(X) = X^n \pm 2$ avec $\delta = 0$, il faudrait appliquer la réduction interne sans quoi les coefficients dépasserait les bornes imposées et le résultat obtenu après une multiplication par un élément du PMNS pourrait dépasser ρ après réduction interne.

On note que ce procédé requiert E unitaire à cause de l'étape de réduction modulaire par E. Pour E non unitaire, il faut multiplier par le coefficient de plus haut degré de E pour garantir que le résultat soit de degré $n-1$ après réduction (cf. section 2.3.1). Pour E non unitaire, on peut procéder de la façon suivante :

Soit $E(X) = e_0 + e_1X + \cdots + e_nX^n$. On pose $z = e_n$ si $\forall i \in [\![2, n-1]\!]$, e_n divise e_i et $z = (e_n)^{n-1}$ sinon. Soit $\tau \in \mathbb{Z}[X]$, tel que $\tau(\gamma') \equiv \gamma^{n-1}\phi z^{-1} \pmod p$, on peut calculer $zA'(X)\tau(X) \pmod E$ et obtenir un représentant de a dans le PMNS réciproque après réduction interne (en supposant que l'on utilise la réduction interne Montgomery-like, sinon il faudrait prendre $\tau(\gamma') \equiv \gamma^{n-1}z^{-1} \pmod p$ pour la réduction Barrett-like et $\tau(\gamma') \equiv \gamma^{n-1}\phi^2 z^{-1} \pmod p$ pour la réduction Plantard-like).

Ce processus est bien entendu moins efficace. Pour avoir un processus de conversion efficace dans les deux sens, il faut que le coefficient de plus bas degré de E soit ± 1 afin d'obtenir un polynôme réciproque unitaire. Une option simple est de considérer $E(X) = X^n - X^{\frac{n}{2}} + 1$ car son polynôme réciproque est lui-même.

Un exemple de PMNS construit de façon réciproque est le suivant :
$p = 6163651274733867379356476526127538185054141306752907529441830157041008640001$,
$\gamma = 1985005035192320 = 5 \times 113 \times 409 \times 2^{33}$, avec $\phi = 2^{64}$, on a $p = 2\gamma^5 + 1$.
En choisissant $\gamma' = 6163651274733864274250360693684135259589516035959663258370069366925426688001$ qui est tel que $\gamma' \equiv \gamma^{-1} \pmod p$, on a bien $\gamma' \times \gamma \equiv 1 \pmod p$ et $\gamma'^5 + 2 \equiv 0 \pmod p$ donc on peut prendre $E'(X) = X^5 + 2$. On peut prendre pour matrice de réduction interne les matrices $\mathcal{G} =$

$$\begin{pmatrix} 0 & 0 & 0 & 1 & -1985005035192320 \\ 0 & 0 & 1 & -1985005035192320 & 0 \\ 0 & 1 & -1985005035192320 & 0 & 0 \\ 1 & -1985005035192320 & 0 & 0 & 0 \\ -3970010070384640 & 0 & 0 & 0 & -1 \end{pmatrix}$$

et $\mathcal{G}^{-1} \pmod \phi \equiv$

$$\begin{pmatrix} 18446744073709551615 & 18442774063639166976 & 0 & 0 & 0 \\ 18444759068674359296 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1985005035192320 \\ 0 & 0 & 1 & 1985005035192320 & 0 \\ 0 & 1 & 1985005035192320 & 0 & 0 \end{pmatrix}$$

et on a un PMNS $(p, n, \gamma', \rho = 3970010070384640, E')$.

4.5 Test d'égalité

Comme expliqué en section 2.7, les tests d'égalités dans les PMNS s'avèrent plus coûteux qu'en représentation classique. La forme particulière avec $\alpha\gamma < \sigma$ et $\lambda < \sigma$ permet l'utilisation d'une méthode plus simple que nous présentons ici.

L'algorithme 30 s'effectue dans le contexte d'un PMNS $\mathcal{B}$ à réduction interne linéaire. Comme noté en remarque 38, on peut choisir $\rho = \max(\alpha\gamma, \gamma + |\lambda| - 1)$. Soient $A, B \in \mathcal{B} \times \mathcal{B}$. Alors $C \in \mathbb{Z}[X]$ tel que $C(X) = A(X) - B(X)$ vérifie $C(\gamma) \equiv 0 \pmod p$ si $A(\gamma) \equiv B(\gamma) \pmod p$. Puisque $\|A\|_\infty < \max(\alpha\gamma, \gamma + |\lambda| - 1)$ et

$\|B\|_\infty < \max(\alpha\gamma, \gamma + |\lambda| - 1)$, alors $\|C\|_\infty < 2\max(\alpha\gamma, \gamma + |\lambda| - 1)$. Les coefficients polynomiaux de C sont donc proches de la représentation en base γ de $C(\gamma)$.

Ainsi, pour calculer $C(\gamma)$, il suffit de voir les coefficients de $C(X)$ comme un résultat intermédiaire d'un calcul en base γ auquel une propagation de retenues n'a pas encore été effectuée. Une fois la propagation de retenues effectuée, si $C(\gamma)$ est un multiple de p, alors clairement $C(\gamma) \equiv 0 \pmod{p}$. Il se présente ainsi deux cas distincts : $C(\gamma) = jp$ avec $j \in \mathbb{N}$ et $C(\gamma) = -jp$ avec $j \in \mathbb{N}$. Pour le premier cas, avec $p = \alpha\gamma^n - \lambda$, après propagation de retenues le coefficient de poids fort est égal à $j\alpha$ et le coefficient de poids faible sera égal à $-j\lambda$ et tous les autres coefficients seront nuls (pour λ négatif). Pour le deuxième cas, le coefficient de poids fort est égal à $\gamma - j\alpha - 1$ et le coefficient de poids faible est égal à $\gamma + j\lambda$ avec tous les autres coefficients égaux à $\gamma - 1$.

Sinon, pour $p = \frac{\alpha\gamma^n - \lambda}{k}$ avec $k \in \mathbb{Z}^*$, il peut être non trivial de déterminer si C est bien un multiple de p. Ainsi, on peut tout simplement prendre $C(X)$ comme $C(X) = k \times (A(X) - B(X))$. Si $A(\gamma) \equiv B(\gamma) \pmod{p}$, on aura $C(\gamma) \equiv k \times 0 \equiv 0 \pmod{p}$ et on se retrouve dans la même configuration que pour $k = 1$.

Remarque 47. Pour λ positif (et donc $-\lambda$ négatif) si $A(\gamma) \equiv B(\gamma)$ alors on obtient une configuration inverse où l'on a $\gamma - j\lambda$ en coefficient de poids faible et $j\alpha - 1$ en coefficient de poids fort avec les autres coefficients égaux à $\gamma - 1$ ou bien $j\lambda$ en coefficient de poids faible et $\gamma - j\alpha$ en coefficient de poids fort avec tous les autres coefficients égaux à 0 autrement.

Ainsi ce test d'égalité peut être réalisé de façon peu coûteuse. Toute autre configuration de coefficients implique que $A(\gamma) \not\equiv B(\gamma) \pmod{p}$.

Algorithme 30 Test d'égalité dans un PMNS à réduction interne linéaire

Require: $\mathcal{B} = (p, n, \gamma, \rho, E = \alpha\gamma^n - \lambda)$, un PMNS à réduction interne linéaire avec $\alpha\gamma < \sigma$ et $0 > \lambda > -\sigma$, $k \in \mathbb{Z}^*$ tel que $k = \frac{\alpha\gamma^n - \lambda}{p}$, $A, B \in \mathcal{B} \times \mathcal{B}$.
Ensure: True si $A(\gamma) \equiv B(\gamma) \pmod{p}$ sinon False
1: $C(X) \leftarrow k \times (A(X) - B(X))$
2: $\Xi \leftarrow (c_0, c_1, \ldots, c_{n-1}, 0)$
3: **for** $i = 0 \ldots n-1$ **do**
4: **while** $\Xi_i < 0$ **do**
5: $\Xi_i \leftarrow \Xi_i + \gamma$
6: $\Xi_{i+1} \leftarrow \Xi_{i+1} - 1$
7: **end while**
8: **while** $\Xi_i \geqslant \gamma$ **do**
9: $\Xi_i \leftarrow \Xi_i - \gamma$
10: $\Xi_{i+1} \leftarrow \Xi_{i+1} + 1$
11: **end while**
12: **end for**
13: **if** $\Xi_0 \equiv 0 \pmod{\lambda}$ **and** $\Xi_n \equiv 0 \pmod{\alpha}$ **then**
14: **for** $i = 1 \ldots n-1$ **do**
15: **if** $\Xi_i \neq 0$ **then**
16: **return** False
17: **end if**
18: **end for**
19: **return** True
20: **else if** $\Xi_0 - \gamma \equiv 0 \pmod{\lambda}$ **and** $\Xi_n + 1 \equiv 0 \pmod{\alpha}$ **then**
21: **for** $i = 1 \ldots n-1$ **do**
22: **if** $\Xi_i \neq \gamma - 1$ **then**
23: **return** False
24: **end if**
25: **end for**
26: **return** True
27: **else**
28: **return** False
29: **end if**

Remarque 48. Cet algorithme ne traite que le cas $\lambda < 0$, sinon voir remarque 47 pour les adaptations nécessaires pour $\lambda > 0$.

Remarque 49. L'algorithme ne considère que les n premiers coefficients de Ξ lors du passage de retenue. Ainsi, on ne se retrouvera jamais dans un cas où $\Xi_n = \gamma - j\alpha - 1$ mais plutôt dans un cas où $\Xi_n = -j\alpha - 1$. Ainsi, il suffit de tester si $\Xi_n + 1$ est bien un multiple de α (ligne 20), l'autre configuration possible étant $\Xi_n = j\alpha$ pour un cas d'égalité (ligne 13).

Pour exemple, prenons le PMNS $\mathcal{B} = ($
$p = 59035332321913162238945069486275575697749350848117855662210697292233631917991,$
$n = 5,$
$\gamma = 2715253637656946,$
$\rho = 5430507275313892,$
$E(X) = 2X^5 + 3)$ avec $p = \frac{2\gamma^5+3}{5}$.

On pose

$$A(X) = 4081997376032296X^4+2346784344690921X^3-1544613245205914X^2+4272082390606931X-1919351343487089$$

et

$$B(X) = 823693010843960X^4+4518987254816478X^3-2087663972737303X^2+3729031663075540X+2968105204295412$$

les deux polynômes sur lesquels nous voulons effectuer un test d'égalité. Afin de mettre en évidence pourquoi l'on calcule plutôt $C(X) = k \times (A(X) - B(X))$, on calcule tout d'abord $C'(X) = A(X) - B(X)$

$$C'(X) = 3258304365188336X^4-2172202910125557X^3+543050727531389X^2+543050727531391X-4887456547782501.$$

Lorsque $C'(X)$ est évalué en γ, on obtient

$$C'(\gamma) = 3258304365188336\gamma^4-2172202910125557\gamma^3+543050727531389\gamma^2+543050727531391\gamma-4887456547782501$$

ce qui est très proche de la représentation de $C(\gamma)$ en base γ. Nous effectuons le passage de retenues comme suit :

$$\begin{aligned}
C'(\gamma) &= 3258304365188336\gamma^4 - 2172202910125557\gamma^3 + 543050727531389\gamma^2 + 543050727531391\gamma - 4887456547782501 \\
&= 3258304365188336\gamma^4 - 2172202910125557\gamma^3 + 543050727531389\gamma^2 + 543050727531389\gamma + 543050727531391 \\
&= 3258304365188335\gamma^4 + 543050727531389\gamma^3 + 543050727531389\gamma^2 + 543050727531389\gamma + 543050727531391 \\
&= \gamma^5 + 543050727531389\gamma^4 + 543050727531389\gamma^3 + 543050727531389\gamma^2 + 543050727531389\gamma + 543050727531391
\end{aligned}$$

Il n'est pas clair ici que $C'(\gamma) \equiv 0 \pmod{p}$ mais c'est bien le cas. Ainsi, si l'on considère plutôt $C(X) = 5 \times (A(X) - B(X))$, on a

$$C(X) = 16291521825941680X^4-10861014550627785X^3+2715253637656945X^2+2715253637656955X-24437282738912505$$

ce qui est encore une fois une représentation proche de celle en base γ. On évalue :

$$\begin{aligned}
C(\gamma) &= 16291521825941680\gamma^4 - 10861014550627785\gamma^3 + 2715253637656945\gamma^2 + 2715253637656955\gamma - 244372827389125051 \\
&= 16291521825941680\gamma^4 - 10861014550627785\gamma^3 + 2715253637656945\gamma^2 + 2715253637656946\gamma + 9 \\
&= 16291521825941680\gamma^4 - 10861014550627785\gamma^3 + 2715253637656946\gamma^2 + 9 \\
&= 16291521825941680\gamma^4 - 10861014550627784\gamma^3 + 9 \\
&= 16291521825941676\gamma^4 + 9 \\
&= 6\gamma^5 + 9 \\
&= 3 \times 5p
\end{aligned}$$

Donc $A(\gamma) \equiv B(\gamma) \pmod{p}$ et l'algorithme retourne True.

Remarque 50. Puisque $\|C\|_\infty < 2k \max(\alpha\gamma, \gamma + |\lambda| - 1)$, le passage de retenues peut soit être effectué par une série d'additions/soustractions par γ, si $k \times \alpha$ n'est pas trop gros, soit par divisions par γ sur chaque

coefficient. Pour γ avec une forme particulière, cela peut être rapide, mais pour γ quelconque, il est possible de considérer une représentation en virgule fixe et effectuer la division par γ en récupérant plutôt la partie haute de la multiplication par $\left\lfloor \frac{\sigma}{\gamma} \right\rfloor$.

Nous venons de voir un exemple pour $C(\gamma) = jp$, considérons maintenant un exemple où $C(\gamma) = -jp$, dans le même PMNS. On pose

$$A(X) = 1427413962265778X^4+1958695243936225X^3+1290672570934783X^2+1129375635883819X+721060118513433$$

et

$$B(X) = 4685718327454115X^4-2928761303846279X^3+4548976936123118X^2+1672426363415209X-1451142791612122$$

donc $C(X) \equiv k \times A(X) \times B(X) \pmod{E(X)}$ est tel que

$$C(X) = -16291521825941685X^4+24437282738912520X^3-16291521825941675X^2-2715253637656950X+10861014550627775.$$

On évalue C en γ et on propage :

$$\begin{aligned}
C(\gamma) &= -16291521825941685\gamma^4 + 24437282738912520\gamma^3 - 16291521825941675\gamma^2 - 2715253637656950\gamma + 10861014550627775 \\
&= -16291521825941685\gamma^4 + 24437282738912520\gamma^3 - 16291521825941675\gamma^2 - 2715253637656947\gamma + 2715253637656937 \\
&= -16291521825941685\gamma^4 + 24437282738912520\gamma^3 - 16291521825941677\gamma^2 + 2715253637656945\gamma + 2715253637656937 \\
&= -16291521825941685\gamma^4 + 24437282738912513\gamma^3 + 2715253637656945\gamma^2 + 2715253637656945\gamma + 2715253637656937 \\
&= -16291521825941677\gamma^4 + 2715253637656945\gamma^3 + 2715253637656945\gamma^2 + 2715253637656945\gamma + 2715253637656937 \\
&= -7\gamma^5 + 2715253637656945\gamma^4 + 2715253637656945\gamma^3 + 2715253637656945\gamma^2 + 2715253637656945\gamma + 2715253637656937.
\end{aligned}$$

Comme on peut le voir ici, le coefficient de poids faible est égal à $\gamma + 3\lambda$ ($\lambda = -3$), le coefficient de poids fort est égal à $-3\alpha - 1$ et tous les autres coefficients sont égaux à $\gamma - 1$. On a donc $C(\gamma) = -3 \times 5p$ et donc $A(\gamma) \equiv B(\gamma) \pmod{p}$.

4.6 Résultats

De façon similaire aux autres sections, la procédure suivante a été mise en place pour effectuer les tests :

- Le *Turbo-Boost*® est désactivé pendant les tests ;
- 501 exécutions sont lancées pour "chauffer" la mémoire cache, c'est à dire que nous nous assurons que la mémoire cache (données et instructions) est dans un état suffisamment stable pour éviter les défauts de cache ;
- 1001 jeux de données aléatoires sont générés, et le nombre de cycles processeurs pour W exécutions sur un lot de 501 exécutions est enregistré pour chaque jeu de données ;
- la mesure finale est la valeur médiane divisée par W.

(À noter que la valeur de W dépend des tailles de paramètres). Le code de benchmarking est disponible sur github à l'adresse https://github.com/francoispalma/PMNS/tree/master/friendly_primes.

4.6.1 Entiers de petites tailles

L'efficacité des PMNS pour des tailles de 256 à 521 bits a déjà été démontrée (cf. [11]). Ces tailles d'entiers correspondent aux tailles couramment utilisées par exemple dans la cryptographie sur les courbes elliptiques. Nous avons donc sélectionné plusieurs courbes issues du site https://safecurves.cr.yp.to/ afin de les comparer avec notre nouvelle classe d'entiers premiers dite "PMNS-friendly". Les entiers utilisés dans les courbes elliptiques en pratique présentent souvent une forme particulière, choisie pour optimiser les calculs. Par exemple, la courbe E-521 est définie sur un nombre premier de Mersenne $(2^{521}-1)$ tandis que la courbe Ed448-Goldilocks repose sur un nombre premier Mersenne généralisé $(2^{448}-2^{224}-1)$. Toutes les autres courbes choisies pour comparaison se basent sur des entiers pseudo-Mersenne.

Méthode \ $\log_2(p)$	255	383	414	448	511	521
Courbe Correspondante	Curve25519	M-383	Curve41417	Ed448-Goldilocks	M-511	E-521
Multi-précision optimisée	61	104	103	122	151	136
PMNS avec $\gamma^2 \equiv 0 \pmod{\phi}$	64	108	109	125	153	153
PMNS à réduction linéaire	77	123	123	135	166	166
GMP bas niveau adapté	94	138	150	261	197	223

TABLE 4.1 – Nombre de cycles pour une multiplication modulaire sur des entiers de taille utilisée en cryptographie sur les courbes elliptiques avec gcc 12.3.0 sur un processeur intel i9-11900KF.

Méthode \ $\log_2(p)$	255	383	414	448	511	521
Courbe Correspondante	Curve25519	M-383	Curve41417	Ed448-Goldilocks	M-511	E-521
Multi-précision optimisée	66	130	113	145	235	194
PMNS avec $\gamma^2 \equiv 0 \pmod{\phi}$	66	118	122	151	162	163
PMNS à réduction linéaire	72	127	127	166	177	180
GMP bas niveau adapté	89	140	164	210	202	222

TABLE 4.2 – Nombre de cycles pour une multiplication modulaire sur des entiers de taille utilisée en cryptographie sur les courbes elliptiques avec clang 14.0.0 sur un processeur intel i9-11900KF.

La ligne "Multi-précision optimisée" dans les tables 4.1, 4.2 et 4.5 correspond à une adaptation d'une implémentation d'Adam Langley (https://code.google.com/archive/p/curve25519-donna/) en langage C d'un code originellement écrit en assembleur de Daniel J. Bernstein pour Curve25519 qui tire parti de l'arithmétique 128-bit offerte par gcc. Nous adaptons ce code aux autres courbes afin d'obtenir un point de comparaison utile.
La ligne "GMP bas niveau adapté" correspond à un code de comparaison n'utilisant que les instructions GMP bas niveau pour effectuer des multiplications modulaires avec une réduction adaptée aux (pseudo-) Mersenne (généralisés) au lieu de simplement utiliser l'instruction `mpn_tdiv_qr` pour effectuer la réduction modulaire, ce qui serait plus lent.

Les tables 4.1 et 4.2 correspondent au même code avec pour seule différence l'utilisation de clang dans la table 4.2. Comme on peut le voir, les résultats de nos entiers PMNS-friendly sont proches de ceux du code de multi-précision optimisé, et les battent même sur certaines tailles avec clang. On rappelle que cette classe d'entiers de la forme $\frac{\alpha\gamma^n-\lambda}{k}$ pour les PMNS est bien plus dense que celle des (pseudo-) Mersenne (généralisés).

n	9	10	11	12	13	14
PMNS avec $\gamma^2 \equiv 0 \pmod{\phi}$	156	185	273	242	360	314
PMNS à réduction linéaire	175	208	300	276	394	353

TABLE 4.3 – Nombre de cycles pour une multiplication modulaire d'entiers de 512 bits pour différentes valeurs de n avec gcc 12.3.0 sur un processeur intel i9-11900KF.

Dans la table 4.3, on montre l'évolution du nombre de cycles en fonction de n pour une même taille d'entiers p. On note qu'à cause de la nature récursive de l'algorithme de multiplication vecteur-matrice de Toeplitz, $n = 12$ est plus rapide que $n = 11$. En effet, 11 est premier donc une matrice 11×11 ne peut pas être subdivisée en sous-matrices par l'algorithme. Ainsi, un n plus petit ne produira pas forcément de meilleures performances en pratique. Plus n sera grand, plus il sera important de s'assurer qu'il possède des facteurs intéressants.

E	X^9-2	X^9-3	$2X^9-1$	$2X^9-3$	$3X^9-1$	$3X^9-2$
PMNS avec $\gamma^2 \equiv 0 \pmod{\phi}$	154	154	156	156	156	156
PMNS à réduction linéaire	167	167	173	172	172	172

TABLE 4.4 – Nombre de cycles pour une multiplication modulaire d'entiers de 512 bits pour différentes formes de E avec gcc 12.3.0 sur un processeur intel i9-11900KF.

Dans la table 4.4, on détaille les performances de plusieurs formes de E. On peut constater que les résultats sont extrêmement proches, ce qui justifie l'utilisation de $\alpha X^n - \lambda$ plutôt que seulement $X^n - \lambda$ afin d'élargir le champ des nombres premiers sur lesquels des PMNS à réduction linéaire peuvent être construits.

Méthode \ $\log_2(p)$	244	297	354	480
Pseudo-Mersenne correspondant	$2^{244}-189$	$2^{297}-123$	$2^{354}-153$	$2^{480}-47$
Multi-précision optimisée	63	82	103	156
PMNS avec $\gamma^2 \equiv 0 \pmod{\phi}$	47	66	85	123
PMNS à réduction linéaire	55	78	94	135
GMP bas niveau adapté	93	115	138	197

TABLE 4.5 – Nombre de cycles pour une multiplication modulaire sur des tailles d'entiers choisies à des tailles avantageuses pour les PMNS avec gcc 12.3.0 sur un processeur intel i9-11900KF.

Dans la table 4.5, les tailles d'entiers choisis correspondent à celles intéressantes du point de vue de la construction des PMNS. En effet, l'exposant des entiers (pseudo-) Mersenne (généralisés) est souvent choisi divisible par 32 ou 64 pour des raisons d'implémentations, mais les considérations pour une arithmétique efficace en PMNS ne sont pas les mêmes. Ainsi, on montre que pour ces tailles choisies, les PMNS sont plus efficaces qu'un code multi-précision optimisé. En l'occurrence, on se compare à des nombres pseudo-Mersenne de même taille, précisés dans la ligne "Pseudo-Mersenne correspondant" du tableau. Ces tailles particulières ont été choisies car elles correspondent à la plus grande taille pour laquelle des PMNS peuvent être construits possédant pour paramètre n la valeur en question. Il s'agit ainsi d'un scénario plus favorable pour ceux-ci. En contraste, pour ces tailles, l'arithmétique multi-précision optimisée nécessite de séparer les entiers sur plus de registres, ce qui mène bien entendu à de moins bonnes performances.

En guise de preuve de concept, nous avons également généré des courbes correspondant sur nos entiers PMNS-friendly à ces tailles qui vérifient les critères Brainpool [22]. Cette génération a été effectuée en adaptant un code disponible à l'adresse http://bada55.cr.yp.to/brainpool.html. Ces courbes sont disponibles dans le github https://github.com/francoispalma/PMNS/tree/master/friendly_primes/curves et les PMNS correspondants sont également disponibles dans le même dossier.

4.6.2 Entiers de plus grande taille

Méthode \ $\log_2(p)$	1023	2002	3979	7813
Pseudo-Mersenne correspondant	$2^{1023}-361$	$2^{2002}-297$	$2^{3979}-3819$	$2^{7813}-241$
PMNS avec $\gamma^2 \equiv 0 \pmod{\phi}$	522	1580	5037	15049
PMNS à réduction linéaire	573	1734	5351	15593
GMP bas niveau adapté	641	1933	6230	18409

TABLE 4.6 – Nombre de cycles pour une multiplication modulaire sur de grands entiers en utilisant gcc 12.3.0 sur un processeur intel i9-11900KF.

Dans la table 4.6, les tailles d'entiers considérées sont plus grandes pour montrer que l'efficacité de cette forme particulière d'entiers PMNS-friendly continue d'avoir du sens en plus grande taille. Il n'y a pas de points de comparaison en multi-précision optimisée par manque d'exemples auxquels se comparer. En effet, on ne pourrait pas adapter le code utilisé plus haut tel quel. Le code utilisant GMP utilise automatiquement des algorithmes de multiplication sous-quadratiques lorsque la taille augmente tandis que le code multi-précision optimisée utilisé en comparaison plus haut reste quadratique.

4.7 Conclusion

Dans ce chapitre, nous avons détaillé les formes d'entiers intéressantes du point de vue de l'arithmétique modulaire pour utilisation en cryptographie dans les PMNS. La forme d'entiers $p = \frac{\alpha\gamma^n-\lambda}{k}$ permet des performances comparables aux entiers pseudo-Mersenne, tout en étant bien plus dense. On estime environ 2^{51} entiers de cette forme sur 256 bits par exemple alors que pour les premiers pseudo-Mersenne bien souvent $2^{255}-19$ est préféré à $2^{256}-189$ pour des raisons d'efficacité. On peut donc estimer qu'il existe environ 2^{50} fois plus d'entiers de cette classe que de premiers pseudo-Mersenne tout en ayant des performances très similaires.

Conclusion

Dans ce manuscrit, nous avons exploré le Polynomial Modular Number System et son utilisation dans l'arithmétique modulaire sur des entiers de tailles utilisées dans la cryptographie à clé publique.

Nous avons montré l'intérêt de l'utilisation de polynômes de réduction externes non unitaires afin de diversifier les formes de PMNS intéressantes. Les adaptations nécessaires s'avèrent relativement peu coûteuses lorsque le coefficient de plus haut degré divise les autres coefficients sauf ceux de degré 1 et 0. Étant donné que les polynômes creux sont parmi les plus intéressants, cette condition de divisibilité s'avère relativement simple à vérifier en pratique. Nous avons ensuite étudié l'utilisation de matrices de Toeplitz à des fins d'accélération de calculs matriciels propres aux PMNS. En particulier, nous avons noté une optimisation permettant de réduire le nombre d'additions nécessaires en fonction de la dimension. Nous avons également proposé une nouvelle forme de réduction interne adaptée des travaux de Thomas Plantard sur les entiers qui présente des propriétés intéressantes du point de vue de l'exponentiation et du test d'égalité. Nous avons de plus proposé de nouvelles bornes afin d'élargir les intervalles de pertinence des PMNS par rapport à l'état de l'art en arithmétique modulaire optimisée. Nous avons aussi amélioré les opérations de conversion et proposé de nouvelles méthodes pour effectuer les tests d'égalité en conservant les bornes optimales sur les paramètres.

Des adaptations particulières nécessaires à l'utilisation des PMNS sur des entiers de taille supérieure à 1024 bits ont été présentées et nous avons proposé de nouvelles méthodes utilisant des coefficients polynomiaux multi-précisions afin de battre les bibliothèques multi-précisions GMP et OpenSSL en single-thread. La capacité des PMNS à être facilement parallélisé a également été mise en évidence avec l'utilisation du multi-threading et de la vectorisation (AVX512IFMA), avec de très bons gains relatifs malgré le fait que l'implémentation vectorisée d'OpenSSL donne de meilleurs résultats pour l'instant.

Enfin nous avons proposé de nouvelles formes d'entiers beaucoup plus fréquents que les entiers Pseudo-Mersenne qui permettent une arithmétique très rapide avec l'utilisation des PMNS. Ces entiers particuliers ont soulevé la question de modification de PMNS existants pour trouver de nouveaux PMNS construits sur le même entier premier et un certain nombre de transformations ont été proposées permettant des conversions de représentants rapides entre PMNS.

Cette gamme d'amélioration permet de positionner les PMNS comme choix raisonnable parmi les autres systèmes de représentations utilisés en arithmétique modulaire.

Un aspect peu exploré dans les PMNS concerne leurs propriétés en ce qui concerne la résistance aux

attaques par canaux cachés. Dans le futur, un travail portant sur ces propriétés pour l'instant largement hypothétiques devra certainement être effectué.

Bibliographie

[1] Diego F. Aranha, Paulo S. L. M. Barreto, Geovandro C. C. F. Pereira et Jefferson E. Ricardini. *A note on high-security general-purpose elliptic curves*. Cryptology ePrint Archive, Paper 2013/647. https://eprint.iacr.org/2013/647. 2013. URL : https://eprint.iacr.org/2013/647.

[2] J.-C. Bajard, L. Imbert et T. Plantard. « Arithmetic operations in the polynomial modular number system ». In : *17th IEEE Symposium on Computer Arithmetic (ARITH'05)*. 2005, p. 206-213. DOI : 10.1109/ARITH.2005.11.

[3] Jean-Claude Bajard, Laurent Imbert et Thomas Plantard. « Modular Number Systems : Beyond the Mersenne Family ». In : *Selected Areas in Cryptography*. Sous la dir. d'Helena Handschuh et M. Anwar Hasan. Berlin, Heidelberg : Springer Berlin Heidelberg, 2005, p. 159-169. ISBN : 978-3-540-30564-4.

[4] Jean-Claude Bajard, Jérémy Marrez, Thomas Plantard et Pascal Véron. « On Polynomial Modular Number Systems over $\mathbb{Z}/p\mathbb{Z}$ ». In : *Advances in Mathematics of Communications* 18.3 (juin 2024), p. 674-695. DOI : 10.3934/amc.2022018. URL : https://hal.science/hal-03611829.

[5] Paul Barrett. « Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor ». In : *Advances in Cryptology — CRYPTO' 86*. Sous la dir. d'Andrew M. Odlyzko. Berlin, Heidelberg : Springer Berlin Heidelberg, 1987, p. 311-323. ISBN : 978-3-540-47721-1.

[6] Daniel J. Bernstein. « Curve25519 : New Diffie-Hellman Speed Records ». In : *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography*. T. 3958. Lecture Notes in Computer Science. Springer, 2006, p. 207-228. DOI : 10.1007/11745853_14. URL : https://iacr.org/archive/pkc2006/39580209/39580209.pdf.

[7] Joppe Bos, Leo Ducas, Eike Kiltz, T Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler et Damien Stehle. « CRYSTALS - Kyber : A CCA-Secure Module-Lattice-Based KEM ». In : *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. 2018, p. 353-367. DOI : 10.1109/EuroSP.2018.00032.

[8] Cyril Bouvier et Laurent Imbert. « An Alternative Approach for SIDH Arithmetic ». In : *Public-Key Cryptography – PKC 2021*. Sous la dir. de Juan A. Garay. Cham : Springer International Publishing, 2021, p. 27-44. ISBN : 978-3-030-75245-3.

[9] Richard P. Brent et Paul Zimmermann. *Modern Computer Arithmetic*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2010. ISBN : 9780511921698. DOI : 10.1017/CBO9780511921698.

[10] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny et Joost Renes. *CSIDH : An Efficient Post-Quantum Commutative Group Action*. Cryptology ePrint Archive, Paper 2018/383. 2018. URL : https://eprint.iacr.org/2018/383.

[11] Titouan Coladon, Philippe Elbaz-Vincent et Cyril Hugounenq. « MPHELL : A fast and robust library with unified and versatile arithmetics for elliptic curves cryptography ». In : *28th IEEE Symposium on Computer Arithmetic, ARITH 2021, Lyngby, Denmark, June 14-16, 2021*. IEEE, 2021, p. 78-85. DOI : 10.1109/ARITH51176.2021.00026.

[12] Richard E. Crandall. *Method and apparatus for public key exchange in a cryptographic system*. US Patent 5,159,632. US Patent Application US07/761, filed 1991-09-17. 1992.

[13] Pierrick Dartois, Antonin Leroux, Damien Robert et Benjamin Wesolowski. « SQISignHD : New Dimensions in Cryptography ». working paper or preprint. Mars 2023. URL : https://hal.science/hal-04056062.

[14] Laurent-Stephane Didier, Jean-Marc Robert, Fangan Yssouf Dosso et Nadia El Mrabet. « A software comparison of RNS and PMNS ». In : *2022 IEEE 29th Symposium on Computer Arithmetic (ARITH)*. Los Alamitos, CA, USA : IEEE Computer Society, sept. 2022, p. 86-93. DOI : 10.1109/ARITH54963.2022.00025. URL : https://doi.ieeecomputersociety.org/10.1109/ARITH54963.2022.00025.

[15] Laurent-Stéphane Didier, Fangan Yssouf Dosso et Pascal Véron. « Efficient modular operations using the Adapted Modular Number System ». In : *Journal of Cryptographic Engineering* 10.2 (juin 2020), p. 111-133. ISSN : 2190-8516. DOI : 10.1007/s13389-019-00221-7. URL : https://doi.org/10.1007/s13389-019-00221-7.

[16] *Digital signature standard (DSS)*. URL : `http://csrc.nist.gov/publications/fips/`. Note : Federal Information Processing Standard 186-2. Washington : National Institute of Standards et Technology, 2000.

[17] Fangan Yssouf Dosso. « Contribution de l'arithmétique des ordinateurs aux implémentations résistantes aux attaques par canaux auxiliaires ». Theses. Université de Toulon, avr. 2020.

[18] Fangan Yssouf Dosso, Alexandre Berzati, Nadia El Mrabet et Julien Proy. *PMNS revisited for consistent redundancy and equality test*. Cryptology ePrint Archive, Paper 2023/1231. https://eprint.iacr.org/2023/1231. 2023. URL : https://eprint.iacr.org/2023/1231.

[19] Fangan Yssouf Dosso, Nadia El Mrabet, Nicolas Méloni, François Palma et Pascal Véron. « Friendly primes for efficient modular arithmetic using the Polynomial Modular Number System ». In : *Journal of Cryptographic Engineering* 15.3 (sept. 2025), p. 18. ISSN : 2190-8516. DOI : 10.1007/s13389-025-00382-8. URL : https://eprint.iacr.org/2025/090.

[20] Fangan Yssouf Dosso, Jean-Marc Robert et Pascal Véron. « PMNS for efficient arithmetic and small memory cost ». In : *IEEE Transactions on Emerging Topics in Computing* 10.3 (2022), p. 1263-1277. URL : https://hal.science/hal-03768546v1/file/TETC3187786.pdf.

[21] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler et Damien Stehlé. « CRYSTALS-Dilithium : A Lattice-Based Digital Signature Scheme ». In : *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018.1 (fév. 2018), p. 238-268. DOI : 10.13154/tches.v2018.i1.238-268.

[22] *ECC Brainpool standard curves and curve generation*. 2005. URL : https://web.archive.org/web/20070814070853/http://www.ecc-brainpool.org/download/Domain-parameters.pdf.

[23] Peter van Emde Boas. *Another NP-complete problem and the complexity of computing short vectors in a lattice*. Rapp. tech. University of Amsterdam, Department of Mathematics,Netherlands, 1981.

[24] Carl Friedrich Gauss et Arthur A. Clarke. *Disquisitiones Arithmeticae*. Yale University Press, 1965. ISBN : 9780300094732.

[25] GNU Project. *GNU Multiple Precision Arithmetic Library (GMP)*. https://gmplib.org. Version 6.2.1, file `mpn/x86_64/skylake/gmp-mparam.h`, macro `MUL_TOOM22_THRESHOLD`, released July 2020. 2020.

[26] Torbjörn Granlund. *Instruction latencies and throughput for AMD and Intel x86 processors*. 2019. URL : https://gmplib.org/~tege/x86-timing.pdf.

[27] Venkatesan Guruswami, Daniele Micciancio et Oded Regev. « The complexity of the covering radius problem ». In : *computational complexity* 14.2 (juin 2005), p. 90-121. ISSN : 1420-8954. DOI : 10.1007/s00037-005-0193-y. URL : https://doi.org/10.1007/s00037-005-0193-y.

[28] Jacques Hadamard. « Rèsolution d'une question relative aux dèterminants ». In : *Bulletin des Sciences Mathématiques* 2.17 (1893), p. 240-246.

[29] Jacques Hadamard. « Sur la distribution des zéros de la fonction $\zeta(s)$ et ses conséquences arithmétiques. » In : *Bulletin de la Société Mathématique de France* 24 (1896), p. 199-220. DOI : 10.24033/bsmf.545.

[30] Mike Hamburg. *Ed448-Goldilocks, a new elliptic curve*. Cryptology ePrint Archive, Paper 2015/625. 2015. URL : https://eprint.iacr.org/2015/625.

[31] M. Anwar Hasan et Christophe Nègre. « Multiway Splitting Method for Toeplitz Matrix Vector Product ». In : *IEEE Transactions on Computers* 62 (2013), p. 1467-1471. DOI : 10.1109/TC.2012.95.

[32] William George Horner. « A new method of solving numerical equations of all orders, by continuous approximation ». In : *Philosophical Transactions of the Royal Society of London* 109 (1819), p. 308-335. DOI : 10.1098/rstl.1819.0023.

[33] David Jao, Luca De Feo, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev et David Urbanik. *Supersingular Isogeny Key Encapsulation*. NIST Post-Quantum Cryptography Standardization Project, Round 1 Submission. NIST CSRC PQC Submission. 2017.

[34] Anatoly Karatsuba et Yuri Ofman. « Multiplication of Many-Digital Numbers by Automatic Computers ». In : *Proceedings of the USSR Academy of Sciences* 145 (1962), p. 293-294.

[35] C. Kaya Koc, T. Acar et B.S. Kaliski. « Analyzing and comparing Montgomery multiplication algorithms ». In : *IEEE Micro* 16.3 (1996), p. 26-33. DOI : 10.1109/40.502403.

[36] Neal Koblitz. « Elliptic curve cryptosystems ». In : *Mathematics of Computation* 48 (1987), p. 203-209. ISSN : 0025–5718.

[37] A Korkine et G Zolotareff. « Sur les formes quadratiques positives ». In : *Mathematische Annalen* 11.2 (1877), p. 242-292.

[38] Katharina Kreuzer. « Hardness of Lattice Problems ». In : *Archive of Formal Proofs* (fév. 2023). https://isa-afp.org/entries/CVP_Hardness.html, Formal proof development. ISSN : 2150-914x.

[39] Charles-Jean de La Vallée Poussin. « Recherches analytiques sur la théorie des nombres premiers. » In : *Annales de la Société scientifique de Bruxelles* 20B, 21B (1896), p. 183-256, 281-352, 363-397, 351-368.

[40] Arjen K Lenstra, Hendrik Willem Lenstra et László Lovász. « Factoring polynomials with rational coefficients ». In : *Mathematische annalen* 261 (1982), p. 515-534. DOI : 10.1007/BF01457454.

[41] Dimitri Mankowski, Thom Wiggers et Veelasha Moonsamy. « TLS → Post-Quantum TLS : Inspecting the TLS Landscape for PQC Adoption on Android ». In : *2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. Los Alamitos, CA, USA : IEEE Computer Society, juill. 2023, p. 526-538. DOI : 10.1109/EuroSPW59978.2023.00065. URL : https://doi.ieeecomputersociety.org/10.1109/EuroSPW59978.2023.00065.

[42] Nicolas Méloni. « An Alternative Approach to Polynomial Modular Number System Internal Reduction ». In : *IEEE Transactions on Emerging Topics in Computing* (juill. 2022). DOI : 10.1109/TETC.2022.3190368. URL : https://univ-tln.hal.science/hal-03635347.

[43] Nicolas Méloni, François Palma et Pascal Véron. « Multi-precision PMNS with CIOS reduction ». In : *Selected Areas in Cryptography*. Toronto (Ontario), Canada : Springer, 2025. URL : https://hal.science/hal-05144945.

[44] Nicolas Méloni, François Palma et Pascal Véron. « PMNS for Cryptography : A Guided Tour ». In : *Advances in Mathematics of Communications* (2023). DOI : 10.3934/amc.2023033. URL : https://hal.science/hal-04195613.

[45] Victor S. Miller. « Use of elliptic curves in cryptography ». In : *Advances in cryptology : CRYPTO '85*. Sous la dir. d'Hugh C. Williams. T. 218. Lecture Notes in Computer Science. Berlin : Springer, 1986, p. 417-426. ISBN : 3–540–16463–4. DOI : 10.1007/3-540-39799-X_31.

[46] Hermann Minkowski. *Zur Geometrie der Zahlen*. Teubner, Leipzig, 1905.

[47] Peter L. Montgomery. « Modular multiplication without trial division ». In : *Mathematics of Computation* 44.170 (1985), p. 519-521.

[48] Christophe Negre et Thomas Plantard. « Efficient Modular Arithmetic in Adapted Modular Number System Using Lagrange Representation ». In : *Information Security and Privacy, 13th Australasian Conference, ACISP 2008, Wollongong, Australia*. 2008, p. 463-477. DOI : 10.1007/978-3-540-70500-0_34.

[49] Louis Noyez, Nadia El Mrabet, Olivier Potin et Pascal Véron. « Modular multiplication in the AMNS representation : Hardware Implementation ». In : *Selected Areas in Cryptography*. Montréal (Québec), France, août 2024.

[50] *OpenSSL*. https://www.openssl.org/, last accessed 07 Jul 2022. 1998.

[51] Gabriele Paoloni. *White paper : How to Benchmark Code Execution Times on Intel® IA-32 and IA-64 Instruction Set Architectures*. Rapp. tech. Intel Corporation, 2010.

[52] Thomas Plantard. « Arithmétique modulaire pour la cryptographie ». Thèse de doct. Montpellier 2 University, France, 2005.

[53] Thomas Plantard. « Efficient Word Size Modular Arithmetic ». In : *IEEE Transactions on Emerging Topics in Computing* 9.3 (2021), p. 1506-1518. doi : 10.1109/TETC.2021.3073475.

[54] Thomas Plantard. *Polynomial Modular Number System on any Prime Fields using Binomials*. to be published. 2023.

[55] Ronald L Rivest, Adi Shamir et Leonard Adleman. « A method for obtaining digital signatures and public-key cryptosystems ». In : *Communications of the ACM* 21.2 (1978), p. 120-126.

[56] C.P. Schnorr. « A hierarchy of polynomial time lattice basis reduction algorithms ». In : *Theoretical Computer Science* 53.2 (1987), p. 201-224. issn : 0304-3975. doi : https://doi.org/10.1016/0304-3975(87)90064-8. url : https://www.sciencedirect.com/science/article/pii/0304397587900648.

[57] N. J. A. Sloane. *Mersenne exponents : primes p such that $2^p - 1$ is prime. Then $2^p - 1$ is called a Mersenne prime.* https://oeis.org/A000043.

[58] N. J. A. Sloane. *Prime p with prime gap q - p of n-th record Cramer-Shanks-Granville ratio.* https://oeis.org/A111943. 2005.

[59] Jerome Solinas. *Generalized mersenne numbers*. Research Report CORR-99-39, Center for Applied Cryptographic Research, University of Waterloo, Waterloo, ON, Canada. 1999.

[60] *The GNU Multiple Precision Arithmetic Library (GMP)*. https://gmplib.org/, last accessed 07 Jul 2022. 1991.

[61] Pascal Véron. « Contributions en arithmétique modulaire pour la cryptographie ». HDR. Université de Toulon, avr. 2022.

Annexe A

Lien github

Le code source pour les mesures présentées dans ce manuscrit est disponible à l'adresse https://github.com/francoispalma/pmns.

François PALMA
Laboratoire IMATH, Université de Toulon

Arithmétique modulaire pour la cryptographie à clé publique

Résumé en français

L'arithmétique modulaire, et en particulier l'arithmétique des corps finis, est la brique essentielle à partir de laquelle sont construits plusieurs protocoles, notamment RSA, ECC et également le protocole post-quantique SIKE. On trouve dans la littérature deux grandes approches concernant l'implantation de cette arithmétique. La première basée sur la représentation des nombres en multi-précision et la seconde consistant à définir un système de représentation non-conventionnel des entiers. Parmi les systèmes possibles, le système de représentation PMNS (Polynomial Modular Number System) proposé en 2005 n'a pas retenu l'attention de la communauté cryptographique malgré ses propriétés originales.

L'objectif de cette thèse est d'étudier l'apport d'un tel système pour la cryptographie à clé publique en général où les entiers manipulés sont de taille parfois supérieure à 1000 bits contrairement à ce qui se passe dans le cadre des courbes elliptiques. De nombreuses questions restent en suspens concernant ce système aussi bien d'un point de vue théorique (concernant l'existence de paramètres optimaux pour un protocole donné) que pratique (concernant l'efficacité et la sécurisation des implantations).

Mots clés : Arithmétique Modulaire, Polynomial Modular Number System, PMNS, Cryptographie à clé publique.

Modular arithmetic for public key cryptography

Résumé en anglais

Modular arithmetic, and in particular finite field arithmetic, is the essential building block from which several protocols are constructed, including RSA, ECC, and also the post-quantum SIKE protocol. There are two main approaches to implementing this arithmetic in the literature. The first is based on the representation of numbers in multi-precision, and the second consists of defining a non-conventional representation system for integers. Among the possible systems, the PMNS (Polynomial Modular Number System) representation system proposed in 2005 did not attract the attention of the cryptographic community despite its novel properties.

The objective of this thesis is to study the contribution of such a system to public-key cryptography in general, where the integers manipulated are sometimes larger than 1000 bits, unlike in the case of elliptic curves. Many questions remain unanswered regarding this system, both from a theoretical point of view (concerning the existence of optimal parameters for a given protocol) and from a practical point of view (concerning the feasibility of implementing such a system in practice).

Keywords : Modular Arithmetic, Polynomial Modular Number System, PMNS, Public Key Cryptography.