Habilitation à diriger les recherches
Université de Toulon

❧

Discipline : Informatique

Contributions en arithmétique modulaire pour la cryptographie

Par : **Pascal Véron**

Membres du jury:

Rapporteur : Daniel Augot, Directeur de Recherche, Inria.
Rapporteur : Sylvain Duquesne, Professeur des Universités, Rennes.
Rapporteur : Marc Girault, Expert émérite Orange, Caen.
Examinateur : Jean-Claude Bajard, Professeur des Universités, Sorbonne Université.
Examinateur : Sylvie Boldo, Directrice de Recherche, Inria.
Garant : Philippe Langevin, Professeur des Universités, Toulon.

Date de soutenance : 06/04/2022

Table des matières

Chapitre 1

Chaînes d'additions euclidiennes

Ce chapitre se veut volontairement long, résultant d'une frustration du fait du diktat d'une majorité de revues et conférences sur le nombre limité de pages pour un article. L'objectif de ce chapitre est de faire une synthèse d'un certain nombre de résultats parfois peu détaillés ou sous-entendus (mais non démontrés) concernant les chaînes d'additions euclidiennes (CAE). En particulier, je donnerai les algorithmes et preuves permettant d'établir clairement le lien entre les CAE, l'algorithme d'Euclide (soustractif et multiplicatif), et les fractions continues. Enfin, je terminerai ce chapitre en démontrant l'efficacité de ces chaînes dans le domaine de la cryptographie basée sur les courbes elliptiques qui est le fruit de plusieurs travaux réalisés en collaboration avec Yssouf Dosso, Fabien Herbaut et Nicolas Méloni. Enfin ce chapitre est une occasion de mettre à jour certains des résultats numériques obtenus dans [Her+10 ; Dos+18 ; HMV21], de compléter certaines preuves (cf. propositions 1.14,1.15, 1.20, 1.22), ainsi que de corriger une légère erreur dans l'algorithme 5 de [DV17].

1.1 Chaînes d'additions et exponentiation

L'exponentiation modulaire est une opération majeure en cryptographie. Il s'agit de calculer de façon "efficace" la quantité $x^j \pmod n$ où x, j et n sont des entiers. On la retrouve au cœur du cryptosystème à clé publique RSA ainsi que dans le non moins célèbre protocole d'échange de clés de Diffie-Hellman-Merkle. Par "efficace", il faut comprendre qu'en pratique cette opération doit être réalisée avec des entiers d'au moins 512 bits [Res99 ; Kit18]. On ne peut donc se contenter d'une méthode de calcul naïve qui consisterait à multiplier successivement x par lui même afin d'obtenir x^j.

Le "square-and-multiply" est un algorithme classique qui réalise cette opération en au plus $2(\lfloor \log_2 j \rfloor + 1)$ multiplications modulaires en utilisant les règles suivantes :

1. $x^0 = 1$,
2. $x^{2k} = (x^k)^2$,
3. $x^{2k+1} = x^{2k}.x$

Remarquons que cet algorithme est valable pour n'importe quel groupe G muni d'une loi de composition $\circ$ et qu'il permet de calculer $x \circ x \circ \cdots \circ x$ en au plus $2(\lfloor \log_2 j \rfloor + 1)$ opérations de composition dans le groupe G. Ainsi, dans le cas où le groupe G est le groupe des points d'une courbe elliptique définie sur un corps $\mathbb{K}$, muni de la loi d'addition sur les points, on obtient alors l'algorithme "double-and-add". En toute rigueur, cette méthode devrait s'intituler "self-compose-and-compose". Afin de calculer x^{15}, cet algorithme calcu-

Algorithm 1 self-compose-and-compose

Require: $n \geq 0$ and $x \neq 0$
Ensure: $y = x \circ x \circ \cdots \circ x$
1: $y \leftarrow e$ (élément neutre de G)
2: **while** $n \neq 0$ **do**
3: **if** n is odd **then**
4: $y \leftarrow y \circ x$ (compose)
5: **end if**
6: $x \leftarrow x \circ x$ (self-compose)
7: $n \leftarrow n/2$
8: **end while**
9: **return** y

lera successivement les valeurs suivantes lors de la mise à jour de y (en gras) et de x : $\mathbf{x}$, x^2, $\mathbf{x^3}$, x^6, $\mathbf{x^7}$, x^{14}, $\mathbf{x^{15}}$. Le résultat est donc obtenu en 6 multiplications. Cette séquence de calcul fait apparaître la suite 1, 2, 3, 6, 7, 14, 15 que l'on appelle chaîne d'additions.

Définition 1.1. *Une chaîne d'additions calculant un entier k est une séquence croissante $u_0, u_2, \ldots, u_s$ d'entiers positifs telle que :*

1. $u_0 = 1$, $u_s = k$,
2. $\forall i \in [1, s]$, $u_i = u_j + u_t$ *pour* $0 \leqslant j, t < i$.

On appelle taille de la chaîne *la valeur de l'entier s.*

La séquence 1, 2, 3, 6, 9, 15 est une chaîne d'additions qui permet aussi de calculer x^{15} en seulement 5 multiplications $x \to x^2 \to x.x^2 \to x^6 \to x^3.x^6 \to x^9.x^6$. **Ce nombre de multiplications correspond exactement à la taille de la chaîne**. Le calcul "efficace" de x^j (et plus généralement de $x \circ \cdots \circ x$) est donc relié au problème de la recherche de la plus petite chaîne d'additions permettant d'obtenir l'entier j. En toute rigueur, ceci n'est pas entièrement exact. En effet selon l'architecture sur laquelle le calcul sera réalisé, une élévation au carré (opération *self-compose*) peut être moins coûteuse qu'une multiplication (opération *compose*). On pourra donc, dans certains cas, préférer utiliser une chaîne un peu plus longue si celle-ci permet d'effectuer de nombreuses élévations au carré.

Le terme *chaîne d'additions* semble avoir été introduit en 1937 dans un papier de Sholz [Sch37]. Cependant, cette notion de suite d'entiers permettant d'indiquer comment effectuer le calcul d'une exponentiation apparaît en 1894 dans le tome 1 de "L'intermédiaire des Mathématiciens", page 163, sous la dénomination de *série* ou *échelle génératrice*.

Une étude détaillée sur cette thématique se trouve dans [Knu97]. Il n'existe pas d'algorithme permettant de calculer une chaîne de taille minimale pour un entier n, tout comme il n'existe pas d'algorithme permettant de calculer cette taille. Cependant, plusieurs méthodes de construction de "chaînes courtes" sont connues (cf. par exemple [BBB94]).

La valeur de la taille minimale $\ell(n)$ d'une chaîne d'additions calculant un entier n a fait l'objet de nombreuses recherches. Le lecteur intéressé pourra par exemple consulter les travaux de Brauer [Bra39] , Yao [Yao76], et un travail de synthèse de Subbarao [Sub89].

A ce stade, il convient de distinguer deux problèmes :

1. Trouver une borne inférieure pour $\ell(n)$. Un résultat classique dans ce domaine de recherche donne l'encadrement suivant :

$$\log_2 n + \log_2 \omega(n) - 2.13 \leqslant \ell(n) \leqslant \lfloor \log_2 n \rfloor + \omega(n) - 1\,,$$

   où $\omega(n)$ est le poids de Hamming de n. Remarquons que la borne supérieure provient de l'algorithme self-compose-and-compose. La borne inférieure provient de [Sch75]. Dans [WJ68, page 10], une borne supérieure indépendante du poids de l'entier n est donnée :

$$\ell(n) < \frac{9 \log_2 n}{\log_2 71} \simeq 1.4635 \log_2 n\,.$$

   Une conjecture importante due à Scholz-Brauer stipule que $\forall n \in \mathbb{N}^*, \ell(2^n - 1) \leqslant n - 1 + \ell(n)$. Brauer démontre dans son papier que $\ell(n) \sim \log_2 n$ [Bra39].

2. Trouver un algorithme "efficace" permettant de générer des chaînes d'additions de taille minimale.

Ces deux problèmes sont considérés comme difficiles [Bah06]. Pour le second problème,

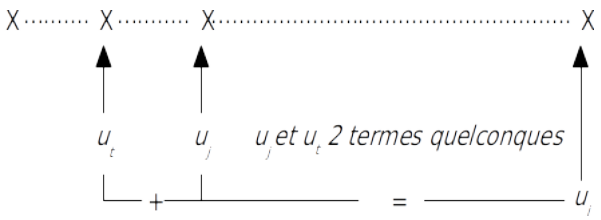


Figure 1.1 – Chaîne d'additions quelconque

bien qu'il n'existe pas à l'heure actuelle de résultat sur sa classe de complexité, un problème plus général est lui prouvé être NP-complet [DLS81] :

Séquence d'additions

Données : Une suite $n_1, \ldots, n_r$ d'entiers positifs et L un entier.
Question : Existe-t-il une chaîne d'additions de longueur au plus $\leqslant L$ contenant tous les entiers n_i ?

Ce problème est lié au calcul simultané de g^{n_1}, g^{n_2},...,g^{n_r} pour un même entier g. Bien entendu ceci ne permet pas de conclure que le problème de la recherche de la taille minimale d'une chaîne d'additions calculant un entier n (cas $r = 1$) reste NP-complet contrairement à ce que l'on peut lire dans certains papiers.

Remarque. *Si l'on disposait d'un algorithme $\mathcal{A}$ renvoyant une chaîne de taille minimale calculant un entier n, cela ne signifie pas que l'utilisation successive de ce dernier pour obtenir des chaînes pour les entiers n_1, n_2, ..., n_r permettrait de minimiser le nombre d'opérations pour calculer g^{n_1}, g^{n_2},...,g^{n_r}. A titre d'exemple, supposons que l'on souhaite calculer g^{13} et g^{47}. Pour des petits exposants, il existe des bases de données de chaînes d'additions de taille minimale. On pourra par exemple consulter :*
https://bo.blackowl.org/random/ln
L'entier 13 peut être obtenu à partir d'une chaîne de taille 5 et l'entier 47 par une chaîne de taille 8. Ainsi, les deux exécutions de l'algorithme $\mathcal{A}$ permettraient de calculer g^{13} et g^{47} en 13 opérations (multiplications et/ou élévations au carré). Or la chaîne $1, 2, 3, 5, 8, 13, 21, 34, 47$ permet de réaliser le même calcul en 8 opérations.

L'utilisation d'une chaîne d'additions pour le calcul de x^j peut nécessiter le stockage de plusieurs valeurs intermédiaires. Par exemple, le calcul de x^{20} à partir de la chaîne 1, 2, 4, 5, 7, 9, 13, 20, impose lors de l'étape de calcul de x^9 de disposer d'une variable contenant le résultat courant (x^9) mais aussi de variables auxiliaires contenant les valeurs x^2, x^4 et x^7 respectivement nécessaires pour le calcul de x^9, x^{13} et x^{20}.

En 1939, Brauer a introduit une classe de chaînes permettant l'utilisation d'un accumulateur : les *star chains* ou *chaînes de Brauer* [Bra39].

Définition 1.2. *Une chaîne de Brauer (ou* star chain*) est une chaîne d'additions telle que $u_0 = 1$ et $\forall i \in [1, s]$, $u_i = u_{i-1} + u_j$ pour $1 \leqslant j < i$.*

Exemple : $1, 2, 3, 5, 8, 13, 26, 39$ est une chaîne de Brauer de longueur 7 permettant de calculer x^{39}.

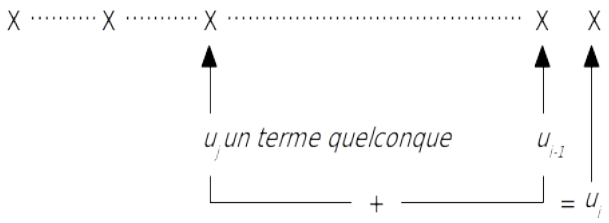


Figure 1.2 – Chaîne de Brauer

La recherche d'une chaîne d'additions de Brauer réduit le nombre de combinaisons possibles puisqu'il est imposé que le terme courant soit toujours obtenu à partir du terme précédent et d'un autre. Une chaîne de Brauer permet donc à partir de certaines valeurs précalculées d'utiliser un accumulateur pour combiner ces valeurs entre elles afin d'obtenir le résultat voulu. Pour l'exemple ci-dessus, deux variables intermédiaires (`Aux`, `Aux2`) sont nécessaires en plus de l'accumulateur (`Acc`) :

. `Acc` $\leftarrow x$,
. `Aux` $\leftarrow$ `Acc` *(Aux=x)*,
. `Acc` $\leftarrow$ `Acc x Aux` *(Acc=x^2)*,
. `Aux2` $\leftarrow$ `Acc` *(Aux2=x^2)*,
. `Acc` $\leftarrow$ `Acc x Aux` *(Acc=x^3)*,
. `Aux` $\leftarrow$ `Acc` *(Aux=x^3)*,
. `Acc` $\leftarrow$ `Acc x Aux2` *(Acc=x^5)*,
. `Aux2` $\leftarrow$ `Acc` *(Aux2=x^5)*,
. `Acc` $\leftarrow$ `Acc x Aux` *(Acc=x^8)*,
. `Acc` $\leftarrow$ `Acc x Aux2` *(Acc=x^{13})*,
. `Aux` $\leftarrow$ `Acc` *(Aux=x^{13})*,
. `Acc` $\leftarrow$ `Acc x Acc` *(Acc=x^{26})*,
. `Acc` $\leftarrow$ `Acc x Aux` *(Acc=x^{39})*

1.2 Chaînes d'additions euclidiennes, une première approche

En 2007, Nicolas Méloni, dans son mémoire de thèse [Mél07a, page 40], constate que pour une courbe elliptique E définie sur $\mathbb{F}_p$, les formules d'additions de deux points P et Q représentés en coordonnées jacobiennes, et partageant la même coordonnée Z, permettent d'obtenir aussi les coordonnées d'un point $\tilde{P}$ dans la même classe d'équivalence que le point P, ayant la même coordonnée Z que $P+Q$. On rappelle qu'en coordonnées jacobiennes, l'équivalence est donnée par $(X:Y:Z) \sim (\lambda^2 X:\lambda^3 Y:\lambda Z)$. Cette opération d'addition nécessite en tout 5 multiplications et deux élévations au carré dans le corps $\mathbb{F}_p$.

C'est la formule nécessitant le moins d'opérations pour calculer $P+Q$, lorsque ces 2 points partagent la même coordonnée Z :
https://www.hyperelliptic.org/EFD/g1p/auto-shortw-jacobian-0.html
Cette formule d'addition (nommée *NewAdd* par Nicolas Méloni [Mél07b]) transforme donc le couple (P,Q) en un couple $(\tilde{P}, P+Q)$ où $\tilde{P}$ est un point équivalent à P. Par convention on omettra pour la suite la classe d'équivalence, et on ne considèrera que NewAdd(P,Q) renvoie le couple $(P, P+Q)$. Plus tard, cette opération sera reprise dans de nombreux papiers et rebaptisée *ZADDU* [GJM10 ; Gou+11 ; Bal+12 ; GJ16]. De même, l'addition de points partageant la même coordonnée Z sera baptisée *Arithmétique Co-Z*.

Ainsi partant du couple (P,P), on peut calculer $(P,2P)$. A ce stade, deux alternatives sont possibles, soit calculer le point $(P,3P)$, soit utiliser la procédure *ZADDU* avec pour entrées le couple $(2P,P)$, afin d'obtenir $(2P,3P)$. De façon plus générale, on engendre donc une suite de couple de points de la forme (a_iP, b_iP) avec :

. $a_0 = 1$, $b_0 = 1$,
. $a_{i+1} = b_i$ ou $a_{i+1} = a_i$,
. $b_{i+1} = a_i + b_i$.

La suite $(b_i)_{i\in\mathbb{N}}$ est une chaîne de Brauer puisque le terme b_{i+1} est obtenu en fonction de b_i et d'un autre terme b_j $(j < i)$ de la suite. Plus précisément, posons $j = 0$, alors les 3 règles précédentes se traduisent de la façon suivante pour la suite $(b_i)_{i\in\mathbb{N}}$:

. $b_0 = 1$, $b_1 = 2$,
. $\forall i \geqslant 1$, $(b_{i+1}, j) \leftarrow (b_i + b_j, j)$ ou $(b_{i+1}, j) \leftarrow (b_i + b_{i-1}, i-1)$.

Ainsi pour le calcul de b_{i+1}, on ne peut pas choisir n'importe quel élément précédent. Le choix ne peut se faire qu'entre les éléments b_{i-1} ou b_j, la valeur j étant remise à jour dans le cas où $b_{i+1} = b_i + b_{i-1}$. C'est ainsi que Nicolas Méloni introduit dans sa thèse la notion de *chaîne d'additions euclidienne* (nous reviendrons sur le terme "euclidien" dans le paragraphe suivant).

Définition 1.3. *Une chaîne d'additions euclidienne (CAE) calculant un entier k est une suite strictement croissante d'entiers vérifiant $u_0 = 1, u_1 = 2, u_2 = u_1+u_0$ et $\forall\ 3 \leq i \leq s-1$, si $u_i = u_{i-1}+u_j$ pour un certain $j < i-1$, alors $u_{i+1} = u_i+u_{i-1}$ (grand pas) ou $u_{i+1} = u_i + u_j$ (petit pas).*

A chaque étape, on a donc le choix entre u_j et u_{i-1} pour réaliser une addition avec u_i. Le terme *petit pas* provient du fait que l'on choisit le plus petit des 2 (u_j) et le terme *grand pas* est utilisé quand le choix se porte sur u_{i-1}.

Exemple : $(1,2,3,4,5,9,13,22,35,57,79)$ est une chaîne d'additions euclidienne de longueur 10 calculant l'entier 79.

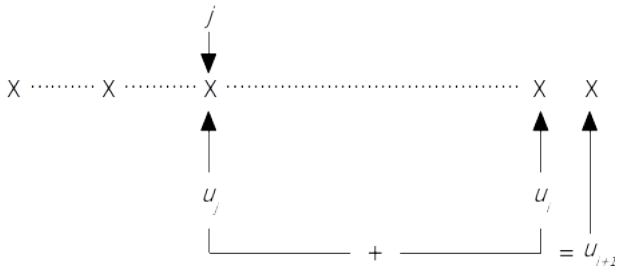


Figure 1.3 – Chaîne d'additions euclidienne (petit pas)

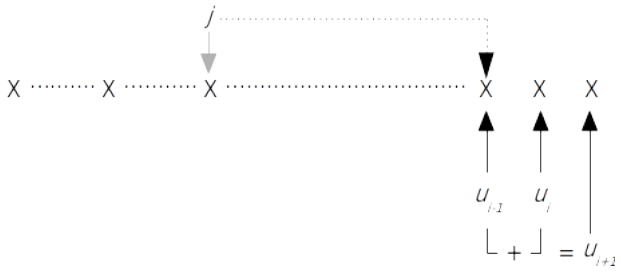


Figure 1.4 – Chaîne d'additions euclidienne (grand pas)

1.3 Chaînes d'additions euclidiennes, division euclidienne et fractions continues : définitions, algorithmes et preuves

Considérons à nouveau la séquence $(a_i, b_i)_{i\in\mathbb{N}}$ du paragraphe précédent. Pour la suite on notera (v, u) un terme quelconque de cette séquence. Notons alors que le prochain terme de la séquence est soit $(v, u + v)$ soit $(u, u + v)$.

Proposition 1.1. *Le pgcd de v est u est égal à 1 et hormis le cas $v = u = 1$, on a toujours $v < u$.*

Démonstration. Initialement $u = v = 1$ donc pgcd(v,u)=1. Si pgcd$(v, u) = 1$ alors, étant donné que pgcd$(v, u+v)$=pgcd$(u, u+v)$=pgcd(v, u), la conclusion est immédiate. De même u et v étant strictement positifs, on a toujours $u + v > v$ et $u + v > u$. □

Le passage d'un couple (v, u) au couple suivant revient à appliquer la transformation :

$$(v, u) \leftarrow (cv + (1 - c)u, u + v)$$

où $c \in \{0, 1\}$. La valeur $c = 1$ correspond à un petit pas et $c = 0$ à un grand pas. Cette remarque permet de définir la notion de chaîne d'additions euclidienne à partir d'une

séquence binaire.

Définition 1.4. *Une chaîne d'additions euclidienne de longueur s est représentée par une suite $(c_i)_{i=1...s}$ où $c_i \in \{0,1\}$. La chaîne d'additions associée à cette suite est la suite $(u_i)_{i=0..s}$ calculée à partir de la séquence $(v_i, u_i)_{i=0..s}$ telle que :*

. $v_0 = 1$, $u_0 = 1$,

. $\forall i \geqslant 1$, $(v_i, u_i) = (c_i v_{i-1} + (1 - c_i)u_i, v_{i-1} + u_{i-1})$.

L'entier k calculé par la chaîne d'additions euclidienne représentée par la suite $(c_i)_{i=1...s}$ correspond à l'élément u_s.

Exemple : La suite (1110100011) représente une chaîne d'additions euclidienne de longueur 10 permettant de calculer l'entier 79 en appliquant les transformations suivantes : $(1,1) \xrightarrow{1} (1,2) \xrightarrow{1} (1,3) \xrightarrow{1} (1,4) \xrightarrow{0} (4,5) \xrightarrow{1} (4,9) \xrightarrow{0} (9,13) \xrightarrow{0} (13,22) \xrightarrow{0} (22,35) \xrightarrow{1} (22,57) \xrightarrow{1} (22,79)$.
La chaîne correspondante (projection de la deuxième coordonnée) est la séquence : 1, 2, 3, 4, 5, 9, 13, 22, 35, 57, 79.
A ce stade il convient d'introduire les matrices S_0 et S_1 qui correspondent aux transformations $(v,u) \leftarrow (u, u+v)$ et $(v,u) \leftarrow (v, u+v)$:

$$S_0 = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}, S_1 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

On a alors :

$$(v_s, k) = (1,1) \prod_{i=1..s} S_{c_i}$$

On en déduit que :

$$(1,1) = (v_s,k)(\prod_{i=1..s} S_{c_i})^{-1} = (v_s,k) \prod_{i=1..s} S^{-1}_{c_{s-i+1}}, \tag{1.1}$$

où

$$S_0^{-1} = \begin{pmatrix} -1 & 1 \\ 1 & 0 \end{pmatrix}, \quad S_1^{-1} = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}.$$

1.3.1 Chaînes d'additions euclidiennes et algorithme d'Euclide soustractif

Les deux matrices précédentes sont respectivement liées aux transformations :

$$(v,u) \leftarrow (u-v, u) \text{ et } (v,u) \leftarrow (v, u-v).$$

Ce sont les deux règles de transformation qui sont à la base de l'algorithme d'Euclide soustractif pour le calcul du pgcd :

. $\text{pgcd}(b, a) \leftarrow \text{pgcd}(b, a - b)$, si $a \geqslant 2b$,
. $\text{pgcd}(b, a) \leftarrow \text{pgcd}(a - b, b)$, si $a < 2b$,
. $\text{pgcd}(0, a) = a$.

Si les transformations "petit pas" et "grand pas" satisfont les deux conditions énoncées ci-dessus, alors l'équation 1.1 indique qu'à partir du couple (v_s, u_s), on peut retrouver l'intégralité de la séquence (v_i, u_i) en appliquant l'algorithme d'Euclide soustractif. D'après la proposition 1.1, on a toujours $v < u$. Ainsi, après un "petit pas", le couple $(v, u + v)$ satisfait $u + v \geqslant 2v$, on peut donc appliquer la règle numéro 1 de l'algorithme d'Euclide soustractif. Après un "grand pas", le couple $(u, u+v)$ vérifie $u+v < 2u$, ce qui correspond à la condition de la règle numéro 2. On obtient ainsi une méthode pour générer facilement une CAE pour un entier k. Il suffit de choisir un entier $g < k$ quelconque et premier avec k et appliquer l'algorithme d'Euclide soustractif au couple $(v_s, u_s) = (g, k)$ jusqu'à obtenir le couple $(1, 1)$.

1.3.1.1 Version "compressée" de la représentation

Avant de détailler l'algorithme de génération d'une CAE, remarquons qu'étant donné que $k = u_s = v_{s-1} + u_{s-1}$, le dernier pas c_s est inutile. La connaissance du couple (v_{s-1}, u_{s-1}) est suffisante pour calculer l'entier k. De même, quel que soit la valeur du premier pas c_1, on aboutit sur le couple $(1, 2)$.

Représentation d'une CAE

On peut donc représenter une CAE de longueur s par une suite $(c_i)_{i=1..s-2}$ de $s - 2$ éléments, et calculer cette dernière à partir de la séquence $(v_i, u_i)_{i=1..s-1}$ telle que :

. $v_1 = 1$, $u_1 = 2$,
. $\forall i \in [2, s - 1]$, $(v_i, u_i) = (c_{i-1}v_{i-1} + (1 - c_{i-1})u_i, v_{i-1} + u_{i-1})$.

Exemple : La suite (11010001) est la version "compressée" de l'exemple de la définition 1.4. Elle permet de calculer l'entier 79 en appliquant les transformations suivantes (cf. Algorithme 2) :
$(1, 2) \xrightarrow{1} (1, 3) \xrightarrow{1} (1, 4) \xrightarrow{0} (4, 5) \xrightarrow{1} (4, 9) \xrightarrow{0} (9, 13) \xrightarrow{0} (13, 22) \xrightarrow{0} (22, 35) \xrightarrow{1} (22, 57)$.
L'entier calculé vaut 79=22+57. Avec les notations précédentes, on a donc :

$$k = (1, 2) \prod_{i=1..s-2} S_{c_i} \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

et

$$(v_{s-1}, u_{s-1}) = (1, 2) \prod_{i=1..s-2} S_{c_i}.$$

Algorithm 2 CAE2Int(c)

$v \leftarrow 1$
$u \leftarrow 2$
for $i = 1..\text{len(c)}$ **do**
    $v \leftarrow c_i v + (1 - c_i)u$
    $u \leftarrow u + v$
end for
return $u + v$

Ainsi

$$(1, 2) = (v_{s-1}, u_{s-1}) \prod_{i=1..s-2} S_{c_{s-i-1}}^{-1} .$$

Pour générer une CAE pour un entier k, on peut donc simplement choisir un entier g premier avec k et appliquer l'algorithme d'Euclide au couple $(v_{s-1}, u_{s-1}) = (k - g, g)$ jusqu'à obtenir le couple $(1, 2)$ (cf. Algorithme 3). Notons que la condition $k - g < g$ implique que g doit être choisi strictement supérieur à $k/2$.

Algorithm 3 CalculCAE(k)

Require: $k \geqslant 4$.
Choisir aléatoirement un entier $g > k/2$, premier avec k.
$(v, u) \leftarrow (k - g, g)$
while $u > 2$ **do**
    if $u \geqslant 2v$ **then**
        $(v, u) \leftarrow (v, u - v)$
        Output 1
    else
        $(v, u) \leftarrow (u - v, v)$
        Output 0
    end if
end while

Remarque. *On pourrait très bien choisir $g \leqslant k/2$. Dans ce cas, il suffit d'initialiser le couple (v, u) avec $(g, k - g)$ pour respecter la condition $v < u$. Posons $g' = k - g$, ceci revient donc à initialiser le couple (v, u) avec $(k - g', g')$ avec $g' \geqslant k/2$. Ainsi g et g' donneront la même représentation binaire d'une CAE.*

L'algorithme 3 renvoie en fait l'image miroir $(c_{s-2}, c_{s-1}, \ldots, c_1)$ de la séquence binaire représentant une CAE. Nous verrons un peu plus loin qu'une séquence binaire et son image miroir représentent deux CAE calculant le même entier k (cf. prop. 1.3).

Exemple : Reprenons l'entier 79 de l'exemple illustrant la définition 1.4. Choisissons

$g = 49$. Le déroulement de l'algorithme 3 est le suivant : $(30, 49) \xrightarrow{0} (19, 30) \xrightarrow{0} (11, 19) \xrightarrow{0} (8, 11) \xrightarrow{0} (3, 8) \xrightarrow{1} (3, 5) \xrightarrow{0} (2, 3) \xrightarrow{0} (1, 2)$. La séquence binaire (0010000) représente la CAE $1, 2, 3, 5, 8, 11, 19, 30, 49, 79$ et l'image miroir (0000100) renvoyée par l'algorithme représente la CAE $1, 2, 3, 5, 8, 13, 21, 29, 50, 79$.

1.3.2 Chaînes d'additions euclidiennes, algorithme d'Euclide classique et fractions continues

La représentation binaire d'une CAE peut aussi être obtenue directement à partir de l'algorithme de calcul du pgcd classique. En effet considérons un couple (v, u) avec $v < u$ et v **premier avec** u. Il existe un unique couple d'entiers (q, r) tel que $u = qv + r$ et $0 < r < v$. L'algorithme classique d'Euclide consiste alors à remplacer le couple (v, u) par le couple $(u \bmod v, v)$, c'est à dire le couple (r, v). Etant donné que $u > qv$, ceci revient à appliquer la règle $\text{pgcd}(v, u) \leftarrow \text{pgcd}(v, u - v)$ exactement $q - 1$ fois afin d'obtenir le couple $(v, u - (q - 1)v) = (v, r + v)$, ($q - 1$ "petits pas"). Puis, comme $r + v < 2v$, on applique ensuite la règle $\text{pgcd}(v, u) \leftarrow \text{pgcd}(u - v, v)$ afin d'obtenir le couple (r, v), ce qui correspond à un "grand pas".
Si jamais $r = 1$, on a donc un nouveau couple (v, u) qui est de la forme $(1, u)$ qui ne peut être obtenu qu'avec $u - 2$ "petits pas" à partir du couple $(1, 2)$ (cf. Algorithme 4). En d'autres termes, le dernier quotient partiel q obtenu correspond à une suite de $q - 2$ "petits pas".
Exemple : Soit $k = 52$ et $g = 29$. Appliquons l'algorithme d'Euclide classique au couple $(23, 29)$:

- $29 = 1 \times 23 + 6 \rightarrow 0$,
- $23 = 3 \times 6 + 5 \rightarrow 110$,
- $6 = 1 \times 5 + 1 \rightarrow 0$,
- $5 = 5 \times 1 + 0 \rightarrow 111$.

En lisant la sortie obtenue de la droite vers la gauche, on obtient donc que la suite (11100110) représente une CAE permettant de calculer l'entier 52. Cette version de l'algorithme de calcul de la représentation binaire d'une CAE permet naturellement d'établir le lien avec les fractions continues. Il est bien connu que si on note $q_1, \ldots, q_s$ la suite des quotients partiels obtenus lors du calcul du pgcd des entiers u et v avec l'algorithme d'Euclide classique, alors le développement en fractions continues de u/v est $[q_1, \ldots, q_s]$,

Algorithm 4 CalculCAE(k)

Require: $k \geqslant 4$.

1: Choisir aléatoirement un entier $g > k/2$, premier avec k.
2: $(v, u) \leftarrow (k - g, g)$
3: **while** $v \neq 1$ **do**
4: $\quad (q, r) \leftarrow (\lfloor u/v \rfloor, u \bmod v)$
5: $\quad$ Output $\underbrace{11\ldots1}_{q-1}0$
6: $\quad (v, u) \leftarrow (r, v)$
7: **end while**
8: Output $\underbrace{11\ldots1}_{u-2}$

i.e :

$$\frac{u}{v} = q_1 + \cfrac{1}{q_2 + \cfrac{1}{q_3 + \cfrac{1}{\ddots \cfrac{1}{q_{s-1} + \cfrac{1}{q_s}}}}}$$

Ainsi pour l'exemple précédent, on a :

$$\frac{29}{23} = 1 + \cfrac{1}{3 + \cfrac{1}{1 + \cfrac{1}{5}}}$$

L'algorithme 4 permet naturellement de définir un algorithme qui à partir du développement en fractions continues $[q_1, \ldots, q_s]$ de u/v renvoie une séquence binaire représentant une CAE calculant l'entier $u + v$ (Alg. 5).

Algorithm 5 Cont2CAE($q_1, \ldots, q_s$)

1: Output $\underbrace{11\ldots1}_{q_s-2}$
2: $i \leftarrow s - 1$
3: **while** $i \geqslant 1$ **do**
4: $\quad$ Output $0\underbrace{11\ldots1}_{q_i-1}$
5: $\quad i \leftarrow i - 1$
6: **end while**

Inversement, à partir d'une séquence binaire représentant une CAE calculant un entier k,

on peut facilement retrouver le développement en fractions continues du quotient u/v tel que $u+v=k$. En effet, les t premiers petits pas (éventuellement $t=0$) indiquent que la valeur du dernier quotient partiel vaut $t+2$, ensuite tout motif de la forme $0\underbrace{1\dots1}_{t\geqslant 0}$ correspond à un quotient partiel qui a pour valeur $t+1$. On en déduit l'algorithme 6 permettant d'obtenir le développement en fractions continues $[q_1,\dots,q_s]$ de u/v à partir de la représentation binaire d'une CAE calculant l'entier $u+v$ (l'algorithme renvoyant la décomposition dans l'ordre $q_s,\dots,q_1$).

Algorithm 6 CAE2Cont$(c_1,\dots,c_n)$

```
q ← 0
i ← 1
while (c_i == 1) and (i ⩽ n) do
    q ← q + 1
    i ← i + 1
end while
Output q + 2
while i ⩽ n do
    q ← 0
    while (c_i == 1) and (i ⩽ n) do
        q ← q + 1
        i ← i + 1
    end while
    Output q + 1
    i ← i + 1 /* on saute le grand pas */
end while
```

Proposition 1.2

Soit $k \geqslant 4$ un entier. Soit $[q_1,\dots,q_s]$ le développement en fractions continues du quotient u/v tel que $u+v=k$, $u>v$ et pgcd(u,v)=1. La séquence binaire représentant une CAE calculant l'entier k, obtenue à partir de l'algorithme 4 est de la forme :

$$\underbrace{1\dots1}_{q_s-2}0\underbrace{1\dots1}_{q_{s-1}-1}\dots0\underbrace{1\dots1}_{q_i-1}\dots0\underbrace{1\dots1}_{q_1-1}.$$

Réciproquement, étant donné que toute séquence binaire peut s'écrire sous la forme :

$$\underbrace{1\dots1}_{q_1}0\underbrace{1\dots1}_{q_2}\dots0\underbrace{1\dots1}_{q_i}\dots0\underbrace{1\dots1}_{q_s},$$

avec $q_i \geqslant 0$, une telle séquence calcule à partir de l'algorithme 2 un couple (v, u) tel que le développement en fractions continues de u/v soit :

$$[q_s + 1, q_{s-1} + 1, \ldots, q_1 + 2]\,.$$

Démonstration. C'est une conséquence directe des algorithmes 5 et 6. □

1.3.2.1 Représentation et représentation "miroir" d'une CAE

Nous pouvons à présent prouver pourquoi une séquence binaire et sa séquence miroir calculent bien le même entier k lorsque l'on applique l'algorithme 2. Dans les papiers où ce résultat est mentionné, la démonstration renvoie vers l'ouvrage de D. Knuth mais aucune preuve explicite ne s'y trouve.

Rappelons tout d'abord la définition de polynôme continuant.

Définition 1.5. *Soit $\{x_i\}_{i\geqslant 1}$ une famille d'indéterminées. Pour $n \geqslant 1$, on définit le $n^{\text{ème}}$ polynôme continuant par :*

- $K(x_1) = x_1$,
- $K(x_1, \ldots, x_n) = x_n K(x_1, \ldots, x_{n-1}) + K(x_1, \ldots, x_{n-2})$, *pour tout $n \geqslant 2$.*

Proposition 1.3

Soit $(c_i)_{i=1..t}$ la représentation binaire d'une CAE calculant un entier k à partir de l'algorithme 2. Soit $(\tilde{c}_i)_{i=1..t}$ la séquence définie par $\tilde{c}_i = c_{t-i}$ pour $i \in [1, t]$. L'algorithme 2 appliqué à la séquence $(\tilde{c}_i)_{i=1..t}$ renvoie pour valeur de sortie l'entier k.

Démonstration. L'algorithme 6 permet d'obtenir à partir de la séquence $(c_i)_{i=1..t}$ le développement en fractions continues $[q_1, \ldots, q_s]$ du quotient u/v tel que $u + v = k$ (u et v sont uniquement déterminés par la séquence $(c_i)_{i=1..t}$ et sont premiers entre eux). D'après la proposition 1.2, ceci nous indique que la séquence $(c_i)_{i=1..t}$ est de la forme :

$$\underbrace{1\ldots1}_{q_s-2}0\underbrace{1\ldots1}_{q_{s-1}-1}\ldots0\underbrace{1\ldots1}_{q_i-1}\ldots0\underbrace{1\ldots1}_{q_1-1}\,.$$

Un résultat classique sur les polynômes continuants indique que si $[q_1, \ldots, q_s]$ est le développement en fractions continues de u/v, alors ([GKP94, p. 304]) :

$$u = K(q_1, \ldots, q_s)\text{pgcd}(u, v) \quad \text{et} \quad v = K(q_2, \ldots, q_s)\text{pgcd}(u, v)\,.$$

Etant donné que u et v sont premiers entre eux, on en déduit que l'entier calculé par la chaîne d'additions issue de la séquence $(c_i)_{i=1\dots t}$ vaut

$$k = K(q_1, \dots, q_s) + K(q_2, \dots, q_s)\,.$$

Soit $(\tilde{c}_i)_{i=1\dots t}$ la séquence miroir correspondante. Cette dernière est donc de la forme :

$$\underbrace{1\dots1}_{q_1-1}0\underbrace{1\dots1}_{q_2-1}\dots0\underbrace{1\dots1}_{q_i-1}\dots0\underbrace{1\dots1}_{q_s-2}\,.$$

D'après la proposition 1.2, en appliquant l'algorithme 2 à cette séquence, on obtient un entier $\tilde{k}$ de la forme $\tilde{v} + \tilde{u}$ où le développement en fractions continues de $\tilde{u}/\tilde{v}$ vaut $[q_s - 1, q_{s-1}, \dots, q_i, \dots, q_2, q_1 + 1]$. On en déduit donc que :

$$\tilde{u} = K(q_s - 1, q_{s-1}, \dots, q_2, q_1 + 1) \quad \text{et} \quad \tilde{v} = K(q_{s-1}, \dots, q_2, q_1 + 1)\,.$$

On a donc :

$$\tilde{k} = K(q_s - 1, q_{s-1}, \dots, q_2, q_1 + 1) + K(q_{s-1}, \dots, q_2, q_1 + 1)\,.$$

Afin de conclure, nous allons utiliser deux propriétés des polynômes continuants :

- $K(x_1, \dots, x_n) = K(x_n, \dots, x_1)$ ([GKP94, p. 304]),
- $K(x_1, \dots, x_n + y) = K(x_1, \dots, x_{n-1}, x_n) + yK(x_1, \dots, x_{n-1})$ ([GKP94, p. 303]).

On a alors :

$$\begin{aligned} K(q_s - 1, q_{s-1}, \dots, q_2, q_1 + 1) &= K(q_s - 1, q_{s-1}, \dots, q_1) + K(q_s - 1, q_{s-1}, \dots, q_2) \\ &= K(q_1, \dots, q_{s-1}, q_s - 1) + K(q_2, \dots, q_{s-1}, q_s - 1) \\ &= K(q_1, \dots, q_{s-1}, q_s) - K(q_1, \dots, q_{s-1}) + K(q_2, \dots, q_{s-1}, q_s) \\ &\quad - K(q_2, \dots, q_{s-1})\,. \end{aligned}$$

De même,

$$\begin{aligned} K(q_{s-1}, \dots, q_2, q_1 + 1) &= K(q_{s-1}, \dots, q_1) + K(q_{s-1}, \dots, q_2) \\ &= K(q_1, \dots, q_{s-1}) + K(q_2, \dots, q_{s-1})\,. \end{aligned}$$

Ainsi

$$\tilde{k} = K(q_1, \ldots, q_{s-1}, q_s) + K(q_2, \ldots, q_{s-1}, q_s) = k\,.$$

□

La proposition suivante permet d'établir le lien entre la représentation d'une CAE et sa représentation "miroir".

Proposition 1.4

Soit $(c_i)_{i=1..t}$ la représentation binaire d'une CAE et (v, u) le couple obtenu à partir de l'algorithme 2. De même, soit $(\tilde{c}_i)_{i=1..t}$ la séquence définie par $\tilde{c}_i = c_{t-i}$ pour $i \in [1, t]$ et $(\tilde{v}, \tilde{u})$ le couple obtenu à partir de l'algorithme 2. On a :

$$u\tilde{u} \equiv \pm 1 \pmod{u+v} \text{ et } v\tilde{v} \equiv \pm 1 \pmod{u+v}\,.$$

Démonstration. Soit $[q_1, \ldots, q_s]$ le développement en fractions continues de u/v, d'après la démonstration de la proposition précédente, on a :

$$u = K(q_1, \ldots, q_s),\ v = K(q_2, \ldots, q_s)\,,$$

et

$$\tilde{u} = K(q_1, \ldots, q_s) - K(q_1, \ldots, q_{s-1}) + K(q_2, \ldots, q_s) - K(q_2, \ldots, q_{s-1})\,,$$

$$\tilde{v} = K(q_1, \ldots, q_{s-1}) + K(q_2, \ldots, q_{s-1})\,.$$

Les polynômes continuants satisfont la propriété suivante [Knu97, p. 357, eq. 8] :

$$K(x_1, \ldots, x_n)K(x_2, \ldots, x_{n+1}) - K(x_1, \ldots, x_{n+1})K(x_2, \ldots, x_n) = (-1)^n\,. \tag{1.2}$$

On a :

$$\tilde{u} \equiv -K(q_1, \ldots, q_{s-1}) - K(q_2, \ldots, q_{s-1}) \pmod{u+v}\,,$$

d'où

$$\tilde{u}u \equiv -K(q_1, \ldots, q_{s-1})K(q_1, \ldots, q_s) - K(q_1, \ldots, q_s)K(q_2, \ldots, q_{s-1}) \pmod{u+v}\,.$$

Or

$$K(q_1, \ldots, q_s) \equiv -K(q_2, \ldots, q_s) \pmod{u+v}\,,$$

donc

$$\tilde{u}u \equiv K(q_1, \ldots, q_{s-1})K(q_2, \ldots, q_s) - K(q_1, \ldots, q_s)K(q_2, \ldots, q_{s-1}) \pmod{u+v}\,.$$

Ainsi d'après la relation 1.2, $\tilde{u}u \equiv (-1)^{s-1} \pmod{u+v}$. De même, on a :

$$\tilde{v}v = K(q_1, \ldots, q_{s-1})K(q_2, \ldots, q_s) + K(q_2, \ldots, q_{s-1})K(q_2, \ldots, q_s)\,.$$

Or

$$K(q_2, \ldots, q_s) \equiv -K(q_1, \ldots, q_s) \pmod{u+v}\,,$$

donc

$$\tilde{v}v = K(q_1, \ldots, q_{s-1})K(q_2, \ldots, q_s) - K(q_1, \ldots, q_s)K(q_2, \ldots, q_{s-1}) \pmod{u+v}\,.$$

La relation 1.2 permet de conclure que $\tilde{v}v \equiv (-1)^{s-1} \pmod{u+v}$. □

Les algorithmes 3 et 4 permettent, d'après la remarque de la page 10, de trouver les $\varphi(n)/2$ CAE calculant un entier k où $\varphi(n)$ est la fonction d'Euler. Cependant, ils ne permettent pas de connaître à priori, en fonction de l'entier g choisi, la longueur de la chaîne que l'on va obtenir (hormis le cas trivial $g = k-1$ qui produit une chaîne de longueur $k-1$). Ainsi, dans l'exemple de la page 10, en choisissant $g = 49$, on a obtenu une chaîne de longueur 9 pour calculer l'entier 79. Pour l'exemple de la définition 1.4, en choisissant $g = 57$, on obtient une chaîne de longueur 10 pour calculer ce même entier.

1.3.3 Chaînes de Lucas, taille des CAE

L'algorithme 3 nous permet de voir que la taille d'une CAE calculant un entier k est égale à $t+2$ où t est le nombre d'étapes de l'algorithme d'Euclide soustractif. L'algorithme 4 permet quant à lui de voir que cette taille est de la forme $q_1+q_2+\cdots+q_s$ si $[q_1, q_2, \ldots, q_s]$ est le développement en fractions continues de $g/(k-g)$.

En 1975, Yao et Knuth montrent qu'asymptotiquement, le nombre moyen d'étapes de l'algorithme d'Euclide soustractif est de l'ordre de [YK75] :

$$6\pi^{-2}(\ln k)^2 + \mathcal{O}(\log k(\log\log k)^2).$$

Ce résultat semble donc indiquer qu'asymptotiquement la taille moyenne d'une CAE est de l'ordre de $\mathcal{O}((\log k)^{2+\varepsilon})$. En pratique, cette estimation est très pessimiste et s'explique par le fait qu'il existe quelques chaînes dont la taille est de l'ordre de l'entier k. Par exemple, comme précédemment mentionné, en prenant $g = k-1$, l'algorithme d'Euclide soustractif appliqué à $(k-g, g)$ s'exécutera en $k-3$ étapes, ce qui donnera une CAE de taille $k-1$. De même, si p est un "petit diviseur" de $k-1$, premier avec k, la décomposition en fractions continues de $(k-p)/p$ donne $[(k-1-p)/p, p]$ ce qui conduit à une CAE de taille $p + (k-1-p)/p$. En particulier, pour tout k impair, il y a toujours une CAE de taille

$(k+1)/2$.

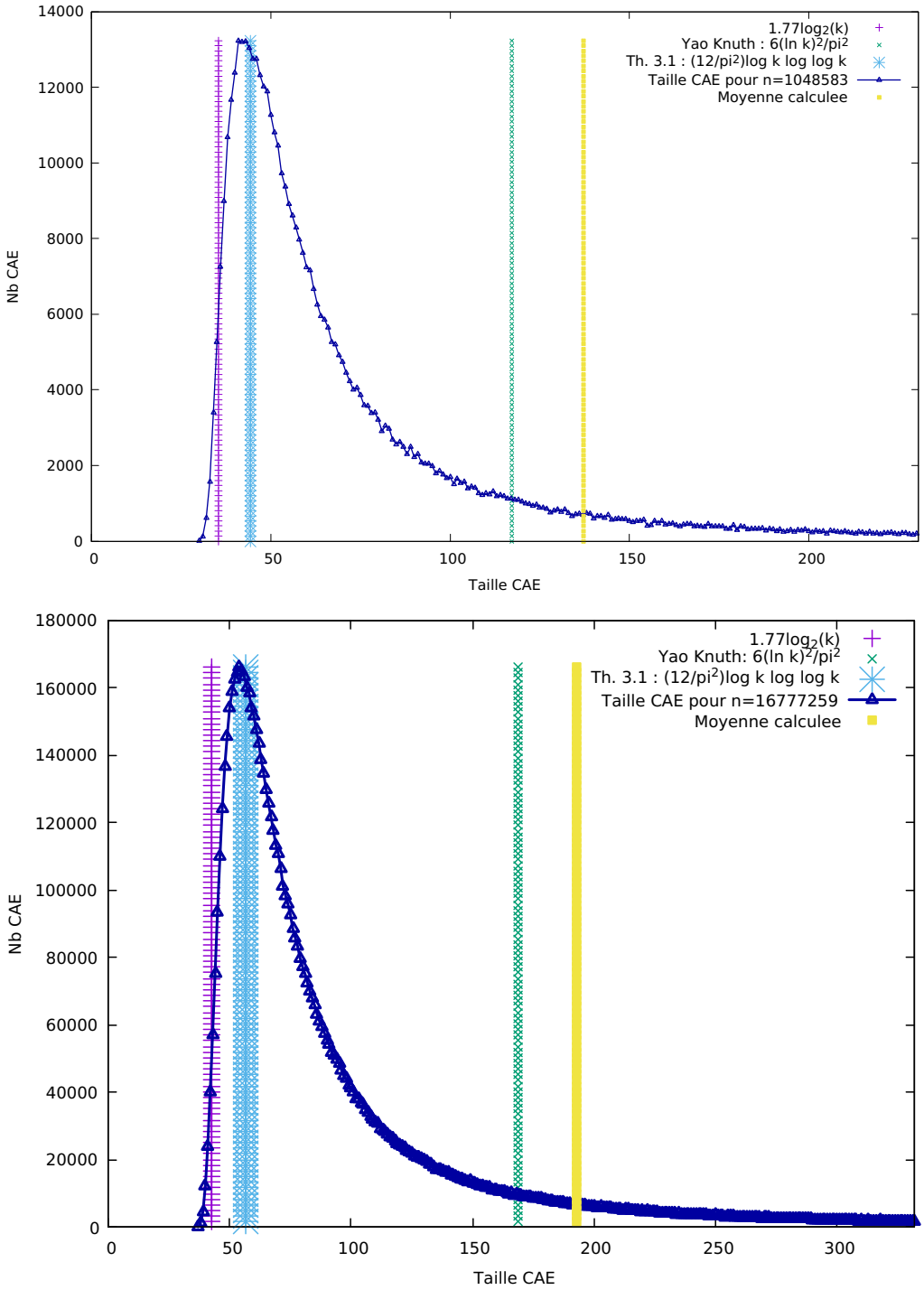


Figure 1.5 – Répartition estimée de la taille des CAE pour $n = 1048583$ (21 bits) et $n = 16777259$ (25 bits).

Pour de nombreuses chaînes, la taille est plutôt de l'ordre de $\mathcal{O}(\log_2 k)$ comme on peut le constater dans les figures 1.5 et 1.6. Pour la figure 1.5, nous avons engendré toutes les CAE pour les entiers $n = 1048583$ (21 bits) et $n = 16777259$ (25 bits). Pour la figure 1.6, nous avons engendré aléatoirement 2^{24} CAE pour l'entier de 256 bits :

$n = 57896044618658097711785492504343953926634992332820282019728792003956564820063$.

Sur chacune de ces figures nous avons fait apparaître, les droites verticales d'équation

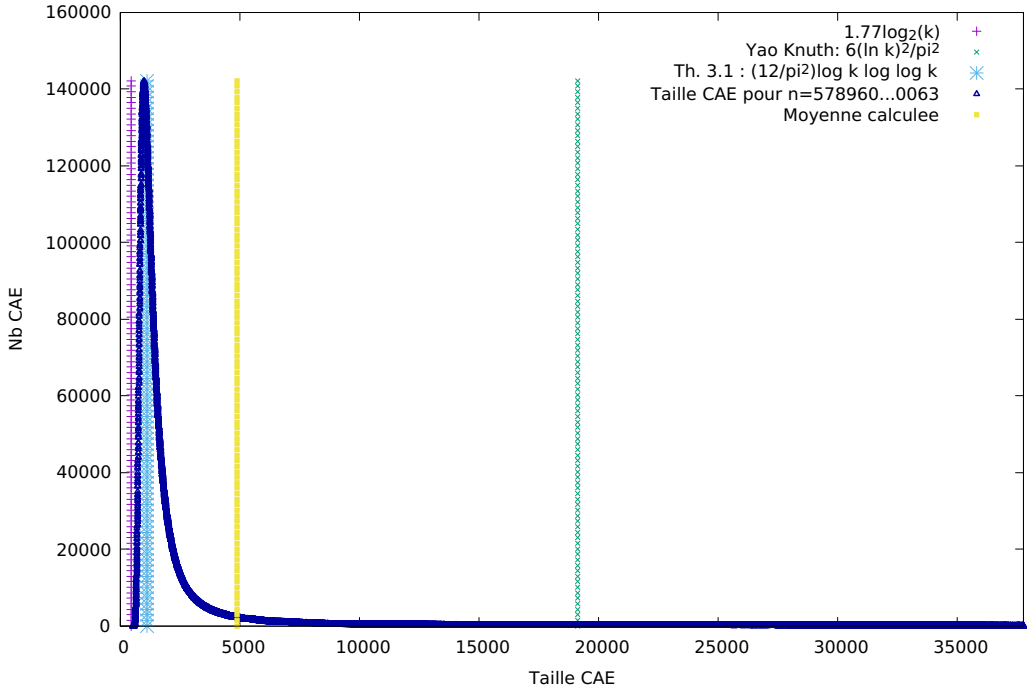


Figure 1.6 – Répartition estimée de la taille des CAE pour $n = 57896044618658097711785492504343953926634992332820282019728792003956564820063$ (256 bits).

$1.77 \log_2(k)$ (cf. prop 1.5), $6(\ln k)^2/\pi^2$ (estimation de Yao-Knuth) et $\frac{12}{\pi^2} \log k \log\log k$ (cf. théorème 1.1). L'intervalle choisi pour l'axe des abscisses est $[0, 12(\ln k)^2/\pi^2]$.

En 2011, un nouveau résultat sur la valeur moyenne de la somme des quotients partiels de a/d, notée $S_d(a)$, pour d fixé et $1 \leqslant a \leqslant d$, permet d'obtenir une "meilleure" estimation de la taille de la plupart des CAE :

Théorème 1.1 ([Ruk11])**.** *Soit $g(d) \in \mathbb{R}^+$ une fonction croissante non bornée telle que*

$g(d) \leqslant \sqrt{\log\log d}$. *Pour tout* $d > 2$,

$$\frac{1}{d} \times \mathrm{card}\left\{ a \in [1,d] \text{ t.q. } \left| S_d(a) - \frac{12}{\pi^2} \log d \log\log d \right| \geqslant g(d) \log d \sqrt{\log\log d} \right\} \ll \frac{1}{g^2(d)} .$$

Une interprétation "rapide" de ce résultat semble indiquer que pour un entier d fixé, il y a "peu" d'entiers a tel que la somme des quotients partiels de a/d s'éloigne de plus de $\log d \log\log d$ de la valeur $\frac{12}{\pi^2} \log d \log\log d$. Notons qu'ici on s'intéresse aux fractions de la forme a/d avec $a < d$. Cependant, il est facile de voir que $S_d(a) = S_a(d)$ puisque si $[q_1, \ldots, q_s]$ est la décomposition en fractions continues de d/a, alors $[0, q_1, \ldots, q_s]$ est la décomposition en fractions continues de a/d. Dans notre cas, on s'intéresse à la somme des quotients partiels de $g/(k-g)$. Or,

$$\frac{k}{g} = 1 + \frac{k-g}{g} .$$

Donc, si $[1, q_1, \ldots, q_s]$ est la décomposition en fractions continues de k/g, alors $[0, q_1, \ldots, q_s]$ est celle de $(k-g)/g$ et donc $[q_1, \ldots, q_s]$ est celle de $g/(k-g)$. On peut donc utiliser le résultat précédent pour $a = g$ et $d = k$ pour avoir une estimation de $S_{k-g}(g)$.

Dans un article non publié [Mon02], Peter Montgomery introduit la définition de chaîne de Lucas en référence aux fonctions de Lucas utilisées dans les algorithmes de factorisation ou de tests de primalités. L'étude de ces chaînes aboutit en particulier à une conjecture sur la taille minimale des CAE.

Définition 1.6. *Une chaîne de Lucas de longueur s calculant un entier k est une suite croissante d'entiers définie par :*

- $a_{-1} = 0, a_0 = 1, a_s = n$,
- $\forall i \in [2, r]$, $\exists j, k, m$ *tels que* $a_i = a_j + a_k$ *et* $a_m = a_j - a_k$ *avec* $-1 \leqslant k, m \leqslant j < i$.

Exemple : La suite $0, 1, 2, 3, 4, 7, 10, 17$ est une chaîne de Lucas de longueur 6.

Montgomery souligne que la suite des entiers obtenus lors du calcul du pgcd de k et de g aboutit toujours à une suite de Lucas si k et g sont premiers entre eux (cf. paragraphe 5 de [Mon02]). Mais ce n'est pas la seule façon de construire une chaîne de Lucas. A titre d'exemple, la suite $0, 1, 2, 3, 5, 6, 11, 17$ est aussi une chaîne de Lucas permettant de calculer l'entier 17 mais ce n'est pas une chaîne d'additions euclidienne. En effet, l'entier 6 est obtenu en faisant $3 + 3$, ce qui n'est pas possible dans une CAE. Dans cette chaîne l'entier 6 n'a pas pu être obtenu en faisant $5 + 1$, car il aurait alors fallu que $4 = 5 - 1$ fasse partie de la chaîne.

La notion de chaîne d'additions euclidienne apparaît donc pour la première fois dans le papier de Montgomery en tant que sous-ensemble des chaînes de Lucas. Si on supprime le terme a_{-1}, cette méthode de construction proposée par Montgomery produit une chaîne d'additions euclidienne. On peut donc utiliser les résultats sur la longueur des chaînes de Lucas pour en déduire des résultats sur la longueur des CAE. Notamment, Montgomery s'intéresse dans son article à la longueur des chaînes de Lucas obtenues en utilisant l'algorithme d'Euclide, par conséquent à la longueur des CAE. Malheureusement, le problème de l'estimation de la longueur des chaînes de Lucas reste un problème difficile et la plupart des résultats connus à ce jour sont des conjectures.

Proposition 1.5. *Soit $L(n)$ la taille de la plus petite chaîne de Lucas calculant un entier n et $L^{CAE}(n)$, la taille de la plus petite chaîne d'additions euclidienne calculant ce même entier :*

- $\log_2 n \leqslant L(n) \leqslant 2\log_2 n$ *([Mon02, paragraphe 2])*,
- $L^{CAE}(n) \leqslant 1.77\log_2 n + \mathcal{O}(1)$ *([Mon02, conjecture, corollaire 14])*.

A ces résultats nous pouvons rajouter le résultat suivant :

Proposition 1.6

Pour tout entier n, $L^{\mathrm{CAE}}(n) \geqslant \frac{1}{\log_2 \phi}\log_2(n-1) - \frac{1}{3}$, où $\phi = \frac{1+\sqrt{5}}{2}$.

Démonstration. Soit n un entier et soit s la taille de la plus petite CAE calculant n. Cette CAE est représentée par une séquence binaire de taille $s-2$ et l'entier n est obtenu en utilisant l'algorithme 2 appliqué à cette séquence. Le plus grand entier que l'on peut obtenir via une CAE de taille s correspond à une séquence de $s-2$ zéros ($s-2$ "grands pas"). En effet, ceci engendre la suite de Fibonacci ; le dernier couple obtenu lors du déroulement de l'algorithme 2 vaut (F_s, F_{s+1}) et l'entier calculé vaut $F_{s+2} = F_s + F_{s+1}$ (cf. [Her+10, Proposition 1]). Ainsi, on a $F_{s+2} \geqslant n$. Etant donné que

$$\frac{\phi^{s+2}}{\sqrt{5}} + 1 \geqslant F_{s+2}\,,$$

on en déduit

$$\log_2\left(\frac{\phi^{s+2}}{\sqrt{5}}\right) \geqslant \log_2(n-1)\,,$$

soit

$$(s+2)\log_2\phi \geqslant \log_2(n-1) + \log_2\sqrt{5}\,.$$

Etant donné que $\log_\phi \sqrt{5} - 2 \geqslant -\frac{1}{3}$, on en déduit :

$$s \geqslant \frac{1}{\log_2\phi}\log_2(n-1) - \frac{1}{3}\,.$$

□

En considérant que $\frac{1}{\log_2 \phi} \simeq 1.44$, ceci nous permet d'estimer que la plus petite chaîne calculant un entier n est de taille au moins $1.44 \log_2 n$.

1.3.4 Recherche de CAE courtes

Pour les chaînes de Lucas obtenues à partir de l'algorithme d'Euclide, Montgomery précise que la recherche d'une chaîne courte pour un entier k revient à chercher l'entier g qui minimise la somme des quotients partiels du développement en fractions continues de $g/(k-g)$. Il propose pour heuristique de choisir g proche de k/ϕ. En effet :

Proposition 1.7

Soit $(c_i)_{i\in\mathbb{N}}$ une séquence binaire représentant une CAE calculant un entier k. Cette séquence se termine par n zéros, si et seulement si l'entier g choisi pour générer cette séquence satisfait :

- $k\frac{F_{n+2}}{F_{n+3}} < g < k\frac{F_{n+1}}{F_{n+2}}$, si n est pair,
- $k\frac{F_{n+1}}{F_{n+2}} < g < k\frac{F_{n+2}}{F_{n+3}}$, sinon.

Démonstration. cf. [HV10, Corollaire 1]. □

Etant donné que :

$$\lim_{n\to\infty} \frac{F_n}{F_{n+1}} = \frac{1}{\phi},$$

cette heuristique se base sur le fait que pour $g \simeq k/\phi$, un certain nombre des premiers quotients partiels du développement en fractions continues de $\frac{g}{k-g}$ seront égaux à 1. Si les autres quotients partiels ne sont pas trop "gros", la somme des quotients partiels (donc la taille de la CAE associée) sera "petite". Les résultats expérimentaux montrent que pour les entiers utilisés classiquement dans le domaine de la cryptographie sur les courbes elliptiques (256 à 512 bits pour un échange Diffie-Hellman), le coût de recherche d'une chaîne de taille "petite" peut être non négligeable (cf. [Mél07a, page 57]).

1.3.4.1 Fractions continues à quotients partiels uniformément bornés

Remarquons que si $[q_1, \ldots, q_s]$ est la décomposition en fractions continues de $g/(k-g)$ alors la décomposition en fractions continues de g/k vaut $[0, 1, q_1, q_2, \ldots, q_s]$. En effet $(k-g)/g = [0, q_1, \ldots, q_s]$ et $k/g = 1 + (k-g)/g$. Une heuristique naturelle pour minimiser la somme des quotients partiels de $g/(k-g)$ pourrait donc consister à chercher un entier g pour lequel les quotients partiels de g/k soient tous bornés par une constance c "petite".

Ce dernier problème a largement été étudié et si certains résultats d'existence ont pu être déterminé, il n'existe malheureusement pas de méthode de construction afin d'obtenir un tel entier g. Voici un bref résumé des résultats dans ce domaine. Notons

$$\mathcal{R}_c = \left\{ \frac{g}{k} = [q_1, \ldots, q_s], \quad 0 < g < k, \quad \text{pgcd}(g,k) = 1, \quad q_i \leqslant c, \ \forall\ 1 \leqslant i \leqslant s \right\}$$

et

$$\mathcal{D}_c = \{k \in \mathbb{N} \mid \exists g \in \mathbb{N} \text{ t.q.} \frac{g}{k} \in \mathcal{R}_c\}\,.$$

En 1972, S.K. Zaremba énonce la conjecture suivante :

Proposition 1.8 (Conjecture de Zaremba [Zar72, p. 69 et 76])**.** *Il existe un entier $c \in \mathbb{N}$ tel que $\mathcal{D}_c = \mathbb{N}$.*

Dans son papier, S.K. Zaremba suggère que l'entier c vaut 5. En 1976, la conjecture (pour $c = 5$) est vérifiée par Itshak Borosh [Bor76] pour les entiers inférieurs à 10000. Il constate de plus que pour environ la moitié des entiers inférieurs à 50000, l'entier c vaut 2.
Plus tard, D. Knuth établit que pour les entiers compris entre 10000 et 3200000, l'entier c vaut 3 [BN83, p. 69]. Ceci est de même conjecturé en 1978 par Harald Niederreiter [Nie78, p. 990], ce dernier suggère même que pour k suffisamment grand, l'ensemble des entiers k pour lesquels $c = 2$ est de densité non nulle.
En 1993, Thomas W. Cusick prouve que pour tout entier $k > 1$, il existe un entier g premier avec k tel que les quotients partiels de g/k soient tous bornés par $3 \log_2 k$ [Cus93].

Essentiellement, on peut classer les recherches dans ce domaine en deux catégories :

- la recherche d'entiers k pour lesquels il existe une constante c explicite et un entier g tel que les quotients partiels de g/k soient tous bornés par c,
- la recherche d'une constante c et du plus "grand" sous-ensemble S de $\mathbb{N}$ pour lequel la conjecture de Zaremba restreint à cet ensemble soit vraie (i.e. $\mathcal{D}_c = S$).

Pour la première catégorie les résultats obtenus sont les suivants :

— pour $k = 2^i$ ou $k = 3^i$, il existe un entier g tel que $c = 3$ (Harald Niederreiter, 1986 [Nie86]),
— pour $k = 5^i$, il existe un entier g tel que $c = 4$ (Harald Niederreiter, 1986 [Nie86]),
— pour $k = 6^i$, il existe un entier g tel que $c = 5$ (M. Yodphotong et V. Laohakosol, 2002 [YL02]),
— pour $k = 7^{j2^n}$, avec $n \geqslant 0$ et $j \in \{1, 3, 5, 7, 9, 11\}$, il existe une entier g tel que $c = 3$ (Takao Komatsu, 2005 [Kom05]).

Concernant la deuxième catégorie :

— en 2014, Jean Bourgain et Alex Kontorovich prouvent qu'il existe une constante A et un sous-ensemble $S \subset \mathbb{N}$ de densité un, tel que $\mathcal{D}_A = S$. Ils montrent qu'il suffit de prendre $A = 50$ pour satisfaire la preuve [BK14, Th. 1.2 page 2],
— en 2015, Shinn Yih Huang améliore le résultat précédent en prouvant qu'il existe un sous-ensemble $S \subset \mathbb{N}$ de densité 1 tel que $\mathcal{D}_5 = S$ [Hua15].

De ce dernier résultat nous pouvons en déduire la proposition évidente suivante :

Proposition 1.9

Il existe un sous-ensemble S de $\mathbb{N}$, de densité 1, tel que pour tout entier $k \in S$, $L^{\mathrm{CAE}}(k) \leqslant 5\lfloor \log_\phi((3-\phi)k)\rfloor$.

Démonstration. Il suffit d'utiliser le fait que pour $g < k$, le calcul du pgcd de g et $k-g$ avec l'algorithme d'Euclide multiplicatif nécessite au plus $\lfloor \log_\phi((3-\phi)k)\rfloor$ étapes ([Knu97, corollaire L, page 360]). Ainsi, le nombre de quotients partiels de $g/(k-g)$ est borné par cette quantité. □

1.3.4.2 Des quotients partiels uniformément bornés vers les quotients partiels bornés en moyenne

L'existence d'une constante universelle c et d'un entier g tel que les quotients partiels $[0, 1, q_1 \ldots, q_t]$ de g/k soient tous bornés par c nous assure l'existence d'une CAE de taille au plus tc calculant l'entier k puisque $\sum_{i=1}^{t} q_i \leqslant tc$. Remarquons que cette dernière inégalité peut se réécrire :

$$\frac{1}{t}\sum_{i=1}^{t} q_i \leqslant c\,.$$

Ce qui peut s'interpréter de la façon suivante : plutôt que de chercher un entier g tel que tous les quotients partiels de g/k soient uniformément bornés par une même constante c, on peut élargir le choix aux entiers g tel que la moyenne des quotients partiels soit bornée par c. Cette problématique a été étudiée par Joshua N. Cooper en 2004. L'auteur s'est particulièrement intéressé au cas $c = 2$ et démontre le résultat suivant :

Proposition 1.10 ([Coo06, Corollaire 4])**.** *Soit $n \geqslant 2$, il existe un sous-ensemble $S \subset [1, n]$ tel que :*

— *$card(S) \geqslant n^{\log 2/\log(1+\sqrt{2})-o(1)}$,*
— *$\forall j \in S$, $\exists g < j$, tel que $(g, j) = 1$ et la moyenne des quotients partiels de g/j soit bornée par 2.*

Comme précédemment, on peut en déduire trivialement la proposition suivante :

Proposition 1.11

Soit $n \geqslant 2$, il existe un sous-ensemble $S \subset [1, n]$ tel que :
- $\text{card}(S) \geqslant n^{\log 2/\log(1+\sqrt{2})-o(1)}$,
- $\forall k \in S,\ L^{\text{CAE}}(k) \leqslant 2\lfloor \log_\phi((3-\phi)k)\rfloor$.

Démonstration. Pour tout k dans S, il existe un entier g tel que si $[q_1, \ldots, q_t]$ est la décomposition en fractions continues de g/k, alors

$$\frac{1}{t}\sum_{i=1}^{t} q_i \leqslant 2\,.$$

D'après [Knu97, corollaire L, page 360], on a

$$t \leqslant \lfloor \log_\phi((3-\phi)k)\rfloor$$

donc

$$\frac{1}{\lfloor \log_\phi((3-\phi)k)\rfloor}\sum_{i=1}^{t} q_i \leqslant 2\,.$$

d'où

$$\sum_{i=1}^{t} q_i \leqslant 2\lfloor \log_\phi((3-\phi)k)\rfloor\,.$$

□

Ainsi dans l'intervalle $[1, n]$ il y a de l'ordre de $n^{0.786}$ entiers k qui sont de bons candidats pour obtenir des chaînes de taille au plus $2t$ où t est le nombre de quotients partiels de g/k. Pour $n = 2^{256} - 1$, ceci donne une probabilité de l'ordre de 3.22×10^{-17} de choisir un tel entier k.

1.4 Chaînes d'additions euclidiennes et courbes elliptiques

Soit P un point d'une courbe elliptique et k un entier. Dans sa thèse, Nicolas Méloni propose la méthode suivante afin de calculer kP :

1. Trouver la représentation binaire $(c_i)_{i=1\ldots t}$ d'une CAE calculant l'entier k.
2. Appliquer l'algorithme 7 afin d'obtenir kP.

Dans cet algorithme, à la $i^{\text{ème}}$ étape, le couple de points (U_1, U_2) obtenu est de la forme (vP, uP) où v et u sont les entiers obtenus lorsque l'on applique l'algorithme 2, page 10, à la suite $(c_i)_{i=1\ldots t}$. Ainsi, si k est l'entier calculé par la chaîne d'addition représentée par la suite $(c_i)_{i=1\ldots t}$, le point calculé par l'algorithme 8 vaut kP. Dans [Mél07a, page 41], il est précisé

Algorithm 7 CAE_Point_Mul($(c_1, \ldots, c_t)$, P)

Require: P
Ensure: $Q = kP$
1: $(U_1, U_2) \leftarrow (P, [2]P)$
2: **for** $i = 1 \ldots t$ **do**
3: **if** $c_i = 0$ **then**
4: $(U_1, U_2) \leftarrow \text{ZADDU}(U_2, U_1)$
5: **else**
6: $(U_1, U_2) \leftarrow \text{ZADDU}(U_1, U_2)$
7: **end if**
8: **end for**
9: $(U_1, U_2) \leftarrow \text{ZADDU}(U_1, U_2)$
10: **return** U_2

Algorithm 8 ZADDU(P, Q)

Require: $P(X_0, Y_0, Z)$ and $Q(X_1, Y_1, Z)$
Ensure: Met à jour X_0, Y_0, X_1, Y_1 et Z de façon à ce que (X_0, Y_0, Z) et (X_1, Y_1, Z) soient des représentants de P et $P + Q$.
1: $A \leftarrow X_1 - X_0$
2: $Z \leftarrow Z.A$
3: $A \leftarrow A^2$
4: $X_0 \leftarrow X_0.A$
5: $A \leftarrow X_1.A$
6: $Y_1 \leftarrow Y_1 - Y_0$
7: $B \leftarrow Y_1^2$
8: $X_1 \leftarrow B - X_0 - A$
9: $A \leftarrow A - X_0$
10: $Y_0 \leftarrow Y_0.A$
11: $B \leftarrow X_0 - X_1$
12: $Y_1 \leftarrow Y_1.B - Y_0$

comment les formules de doublement en coordonnées jacobiennes permettent d'obtenir un point $\tilde{P}$ équivalent à P et possédant la même coordonnée Z que $[2]P$. Cet algorithme effectue $t+1$ appels à la fonction ZADDU dont l'exécution nécessite 5 multiplications et 2 élévations au carré dans $\mathbb{Z}/p\mathbb{Z}$ (cf. Algorithme 8).

Pour la suite, nous nous baserons sur la base de données du site https://hyperelliptic.org/EFD/, recensant le coût des formules d'addition et de doublement pour différents modèles de courbes et différents systèmes de représentation des coordonnées des points d'une courbe. La lettre M représentera une multiplication et S une élévation au carré. On se placera dans l'hypothèse où le coût d'une élévation au carré est estimé être de l'ordre de 0.8 fois le coût d'une multiplication.

En supposant que le point P est donné en coordonnées affines (i.e $Z = 1$), le tableau 1.2 résume le coût du calcul de kP par l'algorithme 7, **une fois la séquence binaire** $(c_1, \ldots, c_t)$ **obtenue**. Ce tableau exprime la complexité en fonction de la taille s de la CAE associée à la séquence $(c_i)_{i=1\ldots t}$. Rappelons que l'on a $s = t + 2$.

Calcul de $[2]P$ [1]	$1M + 5S$
Boucle **for**	$(s-2)(5M + 2S)$
Calcul final	$5M + 2S$
Total	$M(6.6s - 1.6)$

Table 1.2 – Coût du calcul de kP par une CAE de taille s calculant k ($S = 0.8M$).

Remarquons que dans l'algorithme 7, le branchement conditionnel qui dépend de la valeur c_i aboutit à l'exécution de la même séquence d'instructions, en l'occurrence le code la fonction ZADDU. Ceci confère donc à cet algorithme une protection naturelle envers les attaques de type SPA. Pour juger de l'efficacité de ce dernier, il convient donc de le comparer à d'autres méthodes de calcul de point résistantes aux attaques de type SPA. A titre d'exemple, et afin de justifier par la suite notre contribution dans ce domaine, nous allons déterminer pour quelles valeurs de s cet algorithme est compétitif par rapport à l'échelle de Montgomery (cf. [Mon87] et Algorithme 9) et à GLV-SAC (cf. [FLS15] et Algorithme 10) qui est la version résistante aux attaques de type SPA de GLV. Concernant l'échelle de Montgomery, la méthode n'est vraiment efficace que lorsqu'elle est appliquée sur les courbes de Montgomery. Dans ce cas, une addition de points coûte $4M + 2S$, l'opération de doublement coûte $2M + 2S$ exception faite du premier doublement qui coûte $1M + 2S$ si le point P est en coordonnées affines [2]. La table 1.3 résume le coût total de la multiplication scalaire utilisant l'échelle de Montgomery. Soit k un entier de 256 bits, et s la taille de la CAE calculant k, afin que l'algorithme 7 soit plus performant que cette

1. https://hyperelliptic.org/EFD/g1p/auto-shortw-jacobian.html#doubling-mdbl-2007-bl
2. https://hyperelliptic.org/EFD/g1p/auto-montgom-xz.html

Algorithm 9 Echelle de Montgomery

Require: P, $k = (k_{n-1}, \ldots, k_0) \in \mathbb{N}$
Ensure: $Q = kP$
1: $R_0 \leftarrow P$, $R_1 \leftarrow [2]P$
2: **for** $i = n - 2 \ldots 0$ **do**
3: $\quad b \leftarrow k_i$
4: $\quad R_{1-b} \leftarrow R_{1-b} + R_b$
5: $\quad R_b \leftarrow 2R_b$
6: **end for**
7: **return** R_0

	Montgomery	GLV-SAC
Nb op.	$M(6\ell - 5) + S(4\ell - 2)$	$\ell/2(8M + 12S)$
Total M	$(9.2\ell - 6.6)M$	$8.8\ell M$

Table 1.3 – Coût du calcul de kP pour l'échelle de Montgomery et GLV-SAC, k un entier de ℓ bits.

méthode, il faut que :

$$6.6s - 1.6 \leqslant 9.2 \times 256 - 6.6\,,$$

ce qui donne $s \leqslant 356$. Or, d'après la proposition 1.6, la plus petite CAE calculant un entier k de 256 bits est de taille supérieure à $\log_2(k-1)/\log_2\phi - 1/3 \simeq 368.4$. Concer-

Algorithm 10 GLV-SAC(k, PP)

Require: $PP = (P, P + \lambda P)$, $((x_j, \ldots, x_0), (y_j, \ldots, y_0))$ la décomposition SAC de k (cf. Algorithmes 5 et 8 de [Dos+18]).
Ensure: $Q = kP$
1: $Q \leftarrow (X_{PP[|y_j|]}, \text{sign}(x_j).Y_{PP[|y_j|]})$
2: $j \leftarrow j - 1$
3: **while** $(j \geqslant 0)$ **do**
4: $\quad Q \leftarrow 2Q$
5: $\quad Q \leftarrow Q + (X_{PP[|y_j|]}, \text{sign}(x_j).Y_{PP[|y_j|]})$
6: **end while**
7: **return** Q

nant GLV, la méthode peut être appliquée sur toute courbe possédant un endomorphisme "facilement" calculable [GLV01]. En d'autres termes, la méthode GLV est efficace dès qu'il existe un entier $\lambda < \text{ord}(P)$ tel que λP soit "facilement" calculable (en 1 ou 2 multiplications sur $\mathbb{Z}/p\mathbb{Z}$ par exemple). Dans l'algorithme 10, les points P et λP sont donnés en coordonnées affines et le point Q est calculé en coordonnées jacobiennes. Chaque étape de l'algorithme nécessite donc 1 doublement en coordonnées jacobiennes $(1M + 8S)$ et

une addition entre un point en coordonnées jacobiennes et 1 point en coordonnées affines $(7M + 4S)$ [3]. Si l'entier k est de taille ℓ, le réencodage SAC renvoie 2 tuples de taille $\ell/2$. Selon la décomposition GLV de l'entier k, l'algorithme 10 peut nécessiter une dernière addition du point Q avec le point P (cf. [FLS15, paragraphe 3]). La table 1.3 ne prend pas en compte cette dernière. Ainsi pour que l'algorithme 7 soit plus efficace que GLV-SAC, pour un entier k de 256 bits, il faut trouver une CAE de taille s telle que :

$$6.6s - 1.6 \leqslant 256 \times 8.8\,,$$

ce qui donne $s \leqslant 341$.

En conclusion, même si l'on disposait d'un algorithme pour trouver des CAE "courtes", on ne disposerait pas d'une méthode de calcul de kP qui serait compétitive face aux standards actuels. De plus, les résultats du paragraphe précédent semblent indiquer que le nombre d'entiers k pouvant être représentés par une CAE "courte" est très faible.

Avantage CAE/GLV

Afin de nuancer cette conclusion négative, on notera cependant que le calcul de kP via une CAE peut être effectué sur n'importe quelle courbe de Weierstraß ne possédant pas de propriétés particulières (hormis celles classiquement requises pour toute courbe utilisée en cryptographie). Cet algorithme de calcul de kP peut donc s'appliquer à une large famille de courbes. Ce n'est pas le cas de GLV qui nécessite de disposer d'une courbe muni d'un endomorphisme "facilement" calculable.

Concernant l'échelle de Montgomery, l'algorithme 9 peut être utilisé sur d'autres courbes que celles de Montgomery mais ceci nécessite une adaptation des formules d'addition et de doublement ce qui pénalise les performances [BJ02]. Une addition de points nécessite alors 7 multiplications sur le corps de base auxquelles il faut ajouter 3 multiplications par des constantes. L'opération de doublement nécessite quant à elle 7 multiplications sur le corps de base et 2 multiplications par des constantes. Si on omet les multiplications par les constantes, le calcul de kP via une CAE, pour un entier k de 256 bits sera plus efficace dès lors que :

$$6.6s - 1.6 \leqslant 14 \times 256\,,$$

ce qui donne $s \leqslant 543$.

3. https://hyperelliptic.org/EFD/g1p/auto-shortw-jacobian.html

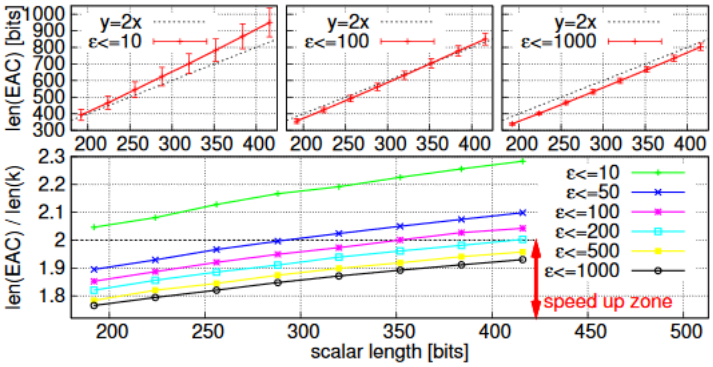


Figure 1.7 – Analyse statistique de la taille des CAE obtenues via l'heuristique de Montgomery [Pro+15].

Avantage CAE/Montgomery

Il suffit donc de trouver des CAE de taille de l'ordre de deux fois la taille de k pour obtenir un algorithme régulier plus performant que l'échelle de Montgomery appliqué à une courbe quelconque. En prenant en compte l'heuristique de Montgomery (cf. paragraphe 1.3.4), on observe expérimentalement qu'en parcourant l'intervalle $[\frac{k}{\phi} - 100, \frac{k}{\phi} + 100]$ pour choisir l'entier g, on obtient assez "facilement" de telles chaînes (cf. fig. 1.7 extraite de [Pro+15]).

Dans le domaine de la cryptographie elliptique, de nombreux protocoles manipulent des données dont la taille varie entre 192 et 521 bits. S'il on admet que pour cet intervalle de taille, il est "facile" d'obtenir une CAE de taille $2t$ pour représenter un entier de taille t, alors le calcul de kP par une CAE sera plus efficace que tout algorithme nécessitant δ opérations sur $\mathbb{F}_p$ par bit de l'entier k (toujours sous l'hypothèse qu'une multiplication et une élévation au carré ont le même coût) dès lors que :

$$6.6s - 1.6 \leqslant \delta t\,,$$

ce qui donne $\delta \geqslant 13.2 - \frac{1.6}{t}$.

CAE et autres algorithmes réguliers de multiplication scalaire

Ainsi, parmi les algorithmes réguliers de calcul de kP, ceux dont le nombre d'opérations par bit du scalaire k est au moins de 14 sont potentiellement moins efficace que l'algorithme 7 (page 26). Le lecteur intéressé par cette famille d'algorithmes (algorithmes réguliers et utilisables sur une courbe de Weierstraß quelconque) en trouvera un panorama dans [Riv11]. Le tableau 1 de ce papier permet de voir que l'algorithme 7 est potentiellement plus efficace qu'une partie d'entre eux. D'autant plus que certains de ces algorithmes nécessitent parfois des pré-calculs et le stockage de plusieurs points, ce qui n'est pas le cas de l'algorithme 7.

1.5 Notre contribution sur l'utilisation des CAE pour le calcul de kP

Trouver une chaîne "courte" pour représenter un entier k devient coûteux en temps pour les tailles d'entiers utilisés actuellement en cryptographie elliptique. Potentiellement, plus k est grand et plus la recherche d'une chaîne d'additions euclidienne proche de la taille minimale (afin de concurrencer les standards actuels) peut s'avérer prendre plus de temps que le calcul de kP lui même ! D'où l'idée de considérer le problème à l'envers. Partir de la représentation binaire (engendrée aléatoirement) d'une CAE (dont on contrôle la longueur) et regarder l'entier k obtenu. Ceci a-t-il un sens ?

1.5.1 CAE Diffie-Hellman

Si on considère le protocole d'échange de clés de Diffie-Hellman (version courbe elliptique), deux entiers k et ℓ sont engendrés et les points kP, ℓP et $k\ell P$ sont calculés. Seul P est fixe et il est important que k et ℓ soient choisis aléatoirement dans un ensemble suffisamment grand. On se propose donc de modifier le déroulement du protocole de Diffie-Hellman comme décrit dans l'algorithme 11. Afin de justifier les lignes 5 et 6, appelons,

Algorithm 11 CAE Diffie-Hellman

Require: P un point
Ensure: Calcul d'un point commun K

1: A engendre aléatoirement la représentation binaire $c^{(1)}$ d'une CAE "courte"
2: B engendre aléatoirement la représentation binaire $c^{(2)}$ d'une CAE "courte"
3: A calcule Q_1=CAE_Point_Mul($c^{(1)}$, P) et l'envoie à B
4: B calcule Q_2=CAE_Point_Mul($c^{(2)}$, P) et l'envoie à A
5: A calcule K=CAE_Point_Mul($c^{(1)}$, Q_2)
6: B calcule K=CAE_Point_Mul($c^{(2)}$, Q_1)

en reprenant les notations de [Her+10, définition 2] :

$$\begin{array}{rccc} \chi & : & \{0,1\}^n & \longrightarrow & \mathbb{N} \\ & & (c_1,\ldots,c_n) & \longmapsto & k \end{array}$$

l'application qui associe à la représentation binaire d'une CAE, l'entier calculé par cette dernière à partir de l'algorithme 2 (page 10). La ligne 3 de l'algorithme correspond alors au calcul de $\chi(c^{(1)})P$ et la ligne de 4 à celui de $\chi(c^{(2)})P$. La ligne 5 calcule le point $\chi(c^{(1)})Q_2$ qui correspond à $\chi(c^{(1)})\chi(c^{(2)})P$ et la ligne 6 calcule le point $\chi(c^{(2)})Q_1$ qui correspond à $\chi(c^{(2)})\chi(c^{(1)})P$.

Remarque

Nous reviendrons plus en détail, à la toute fin de ce chapitre, sur la version CAE Diffie-Hellman.

Procéder de cette façon à l'échange de clés de type Diffie-Hellman soulève deux problèmes.

1.5.1.1 Problème 1

Notons t la taille de $c^{(1)}$ et $c^{(2)}$. Comme cela a déjà été précisé dans la démonstration de la proposition 1.6, l'entier le plus grand que l'on peut obtenir correspond à une séquence de t zéros qui permet de calculer F_{t+4}. Du fait de la non injectivité de la fonction χ (cf. prop. 1.3), les entiers obtenus forment donc un sous-ensemble S de $[1, F_{t+4}]$. Le premier problème concerne la difficulté du calcul du logarithme discret de kP lorsque k est choisi dans le sous-ensemble des entiers représentables par une CAE de taille $t+2$ (donc par une séquence binaire de taille t). Ce problème correspond au problème nommé CDLP (Constrained Discrete Logarithm Problem).

CDLP

Données : g un générateur d'un groupe G, $S \subset [1, \mathrm{card}(G)[$, et g^x pour $x \in S$.
Question : Calculer x.

Tout comme pour le problème classique du logarithme discret, il n'existe pas d'algorithmes spécifiques pour résoudre CDLP sur le groupe des points d'une courbe elliptique. La résolution de CDLP sur un groupe générique nécessite $\Omega(\sqrt{\mathrm{card}(S)})$ opérations de groupe [MMN06]. La complexité de CDLP étant connue, on peut en déduire qu'une attaque visant à retrouver l'entier $\chi(c)$ à partir de $\chi(c)P$ nécessite au moins $\Omega(\sqrt{\mathrm{card}(S)})$

opérations de groupe où S est le sous-ensemble des entiers k pouvant être représentés par une CAE de taille $t+2$. Ceci nous conduit naturellement à notre deuxième problème.

1.5.1.2 Problème 2

De façon très informelle, afin d'assurer un niveau de sécurité de ℓ bits (i.e une complexité d'attaque nécessitant de l'ordre de 2^ℓ opérations de groupe) pour le logarithme discret sur une courbe elliptique, le point P est choisi tel que son ordre soit un nombre premier de taille 2ℓ bits. L'entier k intervenant dans le calcul de kP est choisi aléatoirement dans l'intervalle $[1, \text{ordre}(P)[$ dont la taille est de l'ordre de $2^{2\ell}$. Pour atteindre ce même niveau de sécurité de ℓ bits, quelle valeur doit-on choisir pour l'entier t afin que le cardinal des entiers que l'on peut représenter par une CAE de taille $t+2$ soit de l'ordre de $2^{2\ell}$?

Nombre d'entiers distincts calculables à partir de CAE de taille fixée

En d'autres termes pour un t fixé combien d'entiers distincts obtient-on à partir de la représentation binaire de taille t d'une CAE de taille $t+2$?

Cette question simple à énoncer semble être malheureusement extrêmement compliquée à résoudre (malgré de nombreuses tentatives) du fait de la répartition très "chaotique" des entiers calculés via une séquence binaire représentant une CAE. On peut alors se demander si parmi les CAE d'une taille donnée on ne peut pas identifier un sous-ensemble $\mathcal{M}$ pour lequel on peut calculer une borne inférieure pertinente sur le nombre d'entiers distincts obtenus lorsque l'on restreint l'application χ à $\mathcal{M}$ ou mieux encore pour lequel on sait démontrer que χ restreint à ce sous-ensemble est injective. Parallèlement, il ne faut pas perdre de vue que t doit être suffisamment petit pour que le calcul de $\chi(c)P$ soit plus efficace que le calcul de kP où k est un entier de 2ℓ bits.

1.5.2 Cas du point P fixe

Dans [Her+10] nous proposons trois méthodes de génération de CAE permettant d'assurer un niveau de sécurité donné. Parmi ces 3 méthodes, je ne présenterai dans ce document que la première qui est la seule pour laquelle on peut démontrer un résultat d'injectivité et qui donne de très bons résultats dans le cas d'un point P fixe (cas du protocole Diffie-Hellman lors des deux échanges entre Alice et Bob). De plus, nous avons par la suite étendu ce procédé de calcul de kP au cas où P est variable. La méthode proposée consiste à générer des chaînes de taille $2t$ dans un sous ensemble particulier noté $\mathcal{M}_{0,t}$.

Proposition 1.12

Soit $\mathcal{M}_{0,t}$ l'ensemble des suites binaires de longueur $2t$ dont les t premières composantes valent 0. La restriction de χ à $\mathcal{M}_{0,t}$ est injective.

Démonstration. cf. [Her+10, proposition 3]. □

Pour cet ensemble, on montre facilement les propriétés suivantes :

Proposition 1.13

$\chi(\mathcal{M}_{0,t}) \subset [(t+1)F_{t+2} + F_{t+3}, F_{2t+4}]$, la borne inférieure de l'intervalle est atteinte par la séquence $0^{(t)}1^{(t)}$ et la borne supérieure par la séquence $0^{(2t)}$ (où $j^{(i)}$ représente l'entier j répété i fois). La valeur moyenne d'un entier calculé à partir d'une séquence de $\mathcal{M}_{0,t}$ vaut $(\frac{3}{2})^t F_{t+4}$.

Démonstration. cf. [Her+10, proposition 4]. □

Ainsi, afin de garantir un niveau de sécurité de ℓ bits pour notre version du protocole de Diffie-Hellman, il suffit de tirer aléatoirement les représentations binaires des CAE dans l'ensemble $\mathcal{M}_{0,2\ell}$ (suites de longueur 4ℓ dont les 2ℓ premières composantes sont nulles). Evidemment, il faudra choisir une courbe possèdant un point P d'ordre strictement supérieur à $F_{4\ell+4}$ afin que l'injectivité de χ permette d'obtenir $2^{2\ell}$ points distincts de la courbe.
Ceci nous amène à la réflexion suivante : soit $\phi = \frac{1+\sqrt{5}}{2}$, étant donné que $F_{4\ell+4}$ est l'entier le plus proche de $\varphi^{4\ell+4}/\sqrt{5}$, et que $F_{4\ell+4} \leqslant \varphi^{4\ell+4}/\sqrt{5}$ on peut en conclure que :

$$\lfloor \log_2 F_{4\ell+4} \rfloor + 1 = \lfloor (4\ell+4)\log_2 \varphi - \log_2 \sqrt{5} \rfloor + 1 = \lfloor \ell(4\log_2 \varphi) + 4\log_2 \varphi - \log_2 \sqrt{5} \rfloor + 1\,.$$

Ainsi la taille de $F_{4\ell+4}$ est environ de $\lfloor 2.78\ell - 1.62 \rfloor + 1$, ceci combiné avec le théorème de Hasse donnant un encadrement sur le nombre de points d'une courbe, nous indique que pour obtenir une sécurité de ℓ bits, il va falloir travailler sur un corps premier $\mathbb{F}_p$ où la taille de p est de l'ordre de $\lfloor 2.78\ell - 1.62 \rfloor + 1$, là où les méthodes "classiques" utilisent un entier premier p de taille 2ℓ bits.

Sécurité/Taille du corps

De façon plus informelle, à niveau de sécurité équivalent, l'utilisation des CAE construites à partir de l'ensemble $\mathcal{M}_{0,2\ell}$ a pour conséquence de devoir manipuler des entiers de taille environ 1.4 fois plus importante que les méthodes "classiques".

Il subsiste deux points importants à mentionner pour assurer le bon déroulement du protocole qui n'apparaissent pas de façon explicite dans [Her+10]. L'algorithme de multiplication scalaire (algorithme 7, page 26) n'est valide que dans le cas où l'ensemble des couples de points intermédiaires (U_1, U_2) sont telles que $U_1 \neq U_2$, $U_1 \neq \mathcal{O}_E$ et $U_2 \neq \mathcal{O}_E$. En effet, dans le cas contraire les formules de calcul de `ZADDU` sont incorrectes.

Proposition 1.14

Soit $E(\mathbb{F}_p)$ une courbe et P un point d'ordre strictement supérieur à $F_{4\ell+4}$. Pour toute séquence binaire $(c_i)_{i=1\ldots4\ell}$, les points intermédiaires (U_1, U_2) calculés dans l'algorithme 7 sont distincts 2 à 2 et sont tous différents du point $\mathcal{O}_E$.

Démonstration. Les points (U_1, U_2) sont de la forme (vP, uP) où (v, u) est l'un des couples d'entiers obtenu lorsque l'on applique l'algorithme 2, page 10 à la séquence $(c_i)_{i=1\ldots4\ell}$. D'après la proposition 1.1, page 7, on a $v < u$ pour tous les couples obtenus. De plus, on a toujours $v \leqslant F_{2\ell+2}$ et $u \leqslant F_{2\ell+3}$ (cf. [Her+10, proposition 1]). On ne peut donc jamais obtenir le point $\mathcal{O}_E$ puisque P est d'ordre $4\ell + 4$, et on a bien $vP \neq uP$ pour tout couple (v, u). □

En tirant aléatoirement une suite $(0, \ldots, 0, c_1, \ldots, c_t)$ de longueur $2t$ dans l'ensemble $\mathcal{M}_{0,t}$, les t premiers pas effectués sont des "grands pas". Etant donné que l'on part d'un point P fixe, ceci signifie que les t premières étapes de l'algorithme 7 page 26 conduisent toujours au calcul du couple $(U_1, U_2) = (F_{t+2}P, F_{t+3}P)$, à partir duquel on continue l'algorithme pour les éléments $(c_{t+1}, \ldots, c_{2t})$. Dans le cas d'un point P fixe et pour des suites de longueur $2t$ choisies dans $\mathcal{M}_{0,t}$, on peut donc en pratique précalculer les point $F_{t+2}P$ et $F_{t+3}P$ et ne considérer que les t derniers éléments de la suite en appliquant l'algorithme 12. Dans le tableau 2 de [Her+10, page 248] une comparaison est effectuée entre la mé-

Algorithm 12 CAE_Point_Mul_Fixe($(c_1, \ldots, c_t)$, $F_{t+2}P$, $F_{t+3}P$)

Require: $F_{t+2}P$, $F_{t+3}P$
Ensure: $Q = kP$
1: $(U_1, U_2) \leftarrow (F_{t+2}P, F_{t+3}P)$
2: **for** $i = 1 \ldots t$ **do**
3: **if** $c_i = 0$ **then**
4: $(U_1, U_2) \leftarrow \text{ZADDU}(U_2, U_1)$
5: **else**
6: $(U_1, U_2) \leftarrow \text{ZADDU}(U_1, U_2)$
7: **end if**
8: **end for**
9: $(U_1, U_2) \leftarrow \text{ZADDU}(U_1, U_2)$
10: **return** U_2

Coût : $(t+1)(5M + 2S)$

thode utilisant les CAE et la méthode Comb pour un niveau de sécurité de 80 bits. La méthode Comb [LL94] est l'une des façons les plus efficaces de calculer kP lorsque le point P est fixe et ses performances sont directement liées au nombre de points que l'on s'autorise à précalculer. De plus cette méthode possède une variante dans laquelle le nombre d'opérations effectuées est constant ce qui est un prérequis pour la résistance aux attaques SPA [Fen+06]. Dans ce cas en stockant 2^{h-1} points précalculés, la méthode proposée dans [Fen+06] réalise le calcul de kP en $\lceil \frac{t}{h} \rceil - 1$ doublements de points et $\lceil \frac{t}{h} \rceil$ additions de points, où t est la taille de k. Nous redonnons dans le paragraphe suivant une description de la méthode Comb.

Dans la table 1.4 nous comparons les coûts théoriques (opérations et stockage) du calcul de kP en utilisant une CAE avec les coûts théoriques pour la méthode Comb résistante aux attaques de type SPA. La colonne intitulée CAE (X, Y)-only correspond à une version de l'algorithme 8 page 26 où la coordonnée Z n'est pas calculée. En effet, on peut remarquer que dans la procédure `ZADDU`, la mise à jour des coordonnées X_0, Y_0 et X_1 et Y_1 ne dépend pas de Z. L'algorithme 7 page 26 peut donc être entièrement exécuté sans considérer la coordonnée Z, ce qui permet de gagner une multiplication par tour de boucle. Nous noterons cette procédure `ZADDU_XY`. Dans ce cas, seules les coordonnées (X, Y) de kP sont obtenues. Ceci n'est pas forcément un inconvénient. A titre d'exemple, à partir du point $Q(X, Y, Z)$ calculé par le protocole Diffie-Hellman, si on souhaite dériver une clé de 128 bits pour l'AES, le standard SP 800-56C du NIST [BCD20, page 22, étape 6] se résume essentiellement à extraire les 128 premiers bits de $H(1||X.Y.Z||infos)$, où $X.Y.Z$ est la concaténation des bits représentant les entiers X, Y et Z, *infos* est une suite d'octets indépendante de X, Y et Z, et H est une fonction de hachage (SHA256 par exemple). Si on ne détient que les coordonnées X et Y, on peut donc tout à fait dériver une clé de 128 bits. Dans le cas où la coordonnée Z est indispensable, il est possible de la recalculer ([Mél07a, page 55]).
Depuis la parution de [Her+10], le NIST a mis à jour ses recommandations concernant les niveaux de sécurité pour la cryptographie elliptique :
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-186-draft.pdf
Dans le tableau 1.5, nous donnons pour les niveaux de sécurité 128 et 192, la taille des entiers manipulés pour la méthode Comb-SPA et celle des entiers intervenants dans le calcul de points à partir d'une CAE, ainsi que le nombre de multiplications devant être effectuées sur le corps de base lors du calcul de kP, et le nombre de points stockés. Ces résultats montrent que la version (X, Y)-only des CAE nécessite moins de multiplications que les versions de la méthode Comb où l'on stocke 2 ou 4 points.

	CAE	CAE (X,Y)-only	Comb-SPA
Coût sur $E(\mathbb{F}_p)$	$(2\ell+1)$ ZADDU	$(2\ell+1)5$ ZADDU_XY	$\lceil\frac{2\ell}{h}\rceil-1$ doublements, $\lceil\frac{2\ell}{h}\rceil$ additions
Coût sur $\mathbb{F}_p$	$6.6(2\ell+1)M$	$5.6(2\ell+1)M$	$7.4(\lceil\frac{2\ell}{h}\rceil - 1)M + 10.2\lceil\frac{2\ell}{h}\rceil M$
Stockage	2 points	2 points	2^{h-1} points

Table 1.4 – Coûts CAE et méthode Comb pour une sécurité de ℓ bits, P fixe, $S = 0.8M$.

	Niveau de sécurité					
	128			192		
	Stockage	$\log_2(p)$	Nb. Mult.	Stockage	$\log_2(p)$	Nb. Mult.
CAE	2	358	1696	2	533	2541
CAE (X,Y)-only	2	358	**1439**	2	533	**2156**
Comb-2	2	256	2245	2	384	3372
Comb-4	4	256	1506	4	384	2245
Comb-8	8	256	1119	8	384	1682
Comb-16	16	256	908	16	384	1348

Table 1.5 – Nombre de multiplications ($S = 0.8M$) pour le calcul de kP, P fixe.

1.5.2.1 Une comparaison plus appropriée

Si le nombre de multiplications effectuées dans la version CAE est inférieur à celui qui est effectué dans les méthodes Comb-2 ou Comb-4, il ne faut pas oublier que l'on travaille sur des entiers plus grands (358 bits au lieu de 256 bits par exemple). Dès lors, il faut prendre aussi en compte le surcoût induit par une multiplication entre 2 entiers de 2.8ℓ bits par rapport à une multiplication entre 2 entiers de 2ℓ bits. D'un point de vue pratique, ce surcoût dépend de l'architecture utilisée (8-bits, 32-bits, 64-bits, cryptoprocesseur dédié), du coût de la gestion mémoire et aussi de l'algorithme de multiplication utilisé. Dans [Her+10], nous avons identifié plusieurs contextes pour lesquels la méthode CAE reste plus efficace que la méthode Comb-2 malgré le surcoût induit par la manipulation d'entiers de taille plus importante :

— sur architecture 32-bits ou 64-bits en utilisant la méthode CIOS (Coarsely Integrated Operand Scanning) qui est une implémentation efficace de la multiplication de Montgomery [KAK96],
— sur architecture 32-bits ou 64-bits en utilisant la librairie multiprécision GnuMP,
— sur le crypto-processeur 128-bits Nescrypt de la société STMicroElectronics.

1.5.2.2 De l'efficacité de la méthode CAE (X, Y)-only d'un point de vue "pratique"

Les résultats obtenus dans [Her+10] peuvent être généralisés à toute architecture dès lors qu'une contrainte spécifique sur le surcoût de la multiplication est vérifiée. En effet considérons le nombre de multiplications effectuées dans les méthodes CAE $(X, Y)-$only et Comb-2 :

	$S = 0.8M$	$S = M$
CAE (X, Y)-only	$5.6(2\ell + 1)$	$6(2\ell + 1)$
Comb-2	$17.6\ell - 7.4$	$20\ell - 9$

Table 1.6 – Nombre de multiplications effectuées pour les méthodes CAE (X, Y)-only et Comb-2, sécurité de ℓ bits, P fixe.

Pour un niveau de sécurité $\ell \geqslant 128$ bits, le ratio Comb-2/CAE est :
— supérieur à $\frac{17.5\ell}{5.6(2\ell+1)} \simeq 1.56$ pour $S = 0.8M$,
— supérieur à $\frac{19.9\ell}{6(2\ell+1)} \simeq 1.65$ pour $S = M$.
La méthode Comb-2 effectue donc au moins 1.5 fois plus de multiplications que la méthode CAE (X, Y)-only. Ainsi :

Efficacité CAE (X, Y)-only versus Comb-2

Pour un niveau de sécurité de ℓ bits, la méthode CAE (X, Y)-only est plus performante que la méthode Comb-2 si le coût de la multiplication de 2 entiers de 2.8ℓ bits est strictement inférieur à 1.56 fois le coût de la multiplication de 2 entiers de 2ℓ bits, si $S = 0.8M$, ou strictement inférieur à 1.65 fois le coût de la multiplication de 2 entiers de 2ℓ bits si $S = M$.

La table 1.7 donne à titre d'exemple le nombre de cycles nécessaires pour effectuer une multiplication modulaire entre deux entiers de 256 bits et 358 bits en utilisant :
— la version bas niveau non documentée de GnuMP de la multiplication modulaire de Montgomery,
— la version de la multiplication modulaire de Montgomery présente dans OpenSSL,
— les fonctions bas niveau de multiplication et de réduction modulaire de GnuMP (version 6.2.1),

— les fonctions standards de multiplication et de réduction modulaire de GnuMP (version 6.2.1),
— les fonctions standards de multiplication et de réduction modulaire de OpenSSL (version 1.1.1j).

Les mesures ont été effectuées en utilisant la même démarche que celle présente dans la plateforme d'évaluation SUPERCOP (System for Unified Performance Evaluation Related to Cryptographic Operations and Primitives) : `https://bench.cr.yp.to/supercop.html`. La table 1.7 donne la valeur médiane du nombre de cycles nécessaires pour une multiplication modulaire. Notons qu'il n'existe pas de fonctions spéciales dans les libraires utilisées pour réaliser une élévation au carré, les résultats correspondent donc au contexte $S = M$. Il apparaît clairement que la contrainte énoncée plus haut sur le surcoût de la multiplication modulaire de 2 entiers de 358 bits par rapport à la multiplication modulaire de 2 entiers de 256 bits est satisfaite.

	256 bits	358 bits	Ratio $\times_{358}/\times_{256}$
`gnu_mp_mont`	199	268	1.35
`openssl_mont`	216	344	1.6
`gnu_mp_low`	325	513	1.57
`gnu_mp`	421	602	1.42
`openssl`	1121	1682	1.5

Table 1.7 – Nombre de cycles pour la multiplication d'entiers de 256 et 358 bits, cas $S = M$, gcc 9.3.0.

1.5.2.3 De l'efficacité de la méthode CAE (X, Y)-only d'un point de vue "théorique"

Sur une architecture où les mots machine sont stockés sur w bits, un entier de b bits sera stocké sur $t = \frac{b}{w}$ mots (nous ne considérons pas la partie entière supérieure de cette valeur, afin de simplifier les estimations qui vont suivre). D'un point de vue théorique, la multiplication de deux entiers représentés sur t mots nécessite t^2 multiplications entre mots (de façon plus précise, il faudrait tenir compte de la gestion des retenues et du problème du débordement mémoire). La multiplication de deux entiers représentés sur $t + \delta$ mots nécessitent $t^2 + 2\delta t + \delta^2$ multiplications entre mots. Ainsi le rapport entre le coût d'une multiplication entre deux entiers codés sur $t+\delta$ mots par rapport au coût d'une multiplication entre deux entiers codés sur t mots peut être estimé à :

$$1 + \frac{2\delta t + \delta^2}{t^2}.$$

Afin que la méthode CAE (X, Y)-only soit moins coûteuse que la méthode Comb-2, il suffit donc que :

(a) pour $S = 0.8M$, $\frac{2\delta t+\delta^2}{t^2} \leqslant 0.56$,

(b) pour $S = M$, $\frac{2\delta t+\delta^2}{t^2} \leqslant 0.65$,

On cherche donc les solutions de

$$\begin{aligned} 25\delta^2 + 50\delta t - 14t^2 &\leqslant 0 \qquad (S = 0.8M) \\ 20\delta^2 + 40\delta t - 13t^2 &\leqslant 0 \qquad (S = M) \end{aligned}$$

Le discriminant Δ de ces deux inéquations vaut :

$$\Delta = (10t\sqrt{39})^2 \ (S = 0.8M), \quad \Delta = (4t\sqrt{165})^2 \ (S = M)\,.$$

On obtient alors :

$$\begin{aligned} \delta &\leqslant t(-1 + \tfrac{\sqrt{39}}{5}) \simeq 0.249t \qquad (S = 0.8M) \\ \delta &\leqslant t(-1 + \tfrac{\sqrt{165}}{10}) \simeq 0.284t \qquad (S = M) \end{aligned}$$

Considérons une architecture de k bits, le tableau 1.8 donne pour un nombre de mots machine t et $k = 64$, le nombre de mots machine δ qu'il ne faut pas dépasser pour que les contraintes (a) ou (b) soient satisfaites. Ainsi, pour une sécurité de ℓ bits, afin que la méthode CAE (X, Y)-only soit moins coûteuse que la méthode Comb-2, il suffit que

$$\left\lceil \frac{\lfloor 2.8\ell \rfloor}{k} \right\rceil - \left\lceil \frac{2\ell}{k} \right\rceil = \delta\,. \tag{1.3}$$

En prenant en compte les résultats de la table 1.8, la table 1.9 donne, dans le contexte $S = M$, et pour une architecture 64 bits, la liste exhaustive théorique des couples de la forme $(t, \lfloor 1.4t \rfloor)$ pour lesquels la méthode CAE (X, Y)-only qui effectue des multiplications sur des entiers de $\lfloor 1.4t \rfloor$ bits est moins coûteuse que la méthode Comb-2 qui effectue des multiplications sur des entiers de t bits, le tout pour un niveau de sécurité de $\lfloor t/2 \rfloor$ bits. Nous n'avons obtenu aucun couple pour une architecture 8 bits et seulement 5 pour une architecture 32 bits : (225, 315), (226, 316), (227, 317), (228, 319), (229, 320).

1.5.2.4 La Méthode Comb

Cette méthode de calcul de kP dûe à C.H. Lim et P.J. Lee [LL94] est l'une des plus efficaces dans le cas où le point P est fixe car elle tire partie d'un certain nombre de

$S = 0.8M$		$S = M$	
t	δ	t	δ
4	0	4	1
5	1	5	1
6	1	6	1
7	1	7	1
8	1	8	2
9	2	9	2
10	2	10	2

Table 1.8 – Surcoût δ à ne pas dépasser sur une architecture 64 bits en fonction du nombre de mots $t\}$.

Nb mots de 64 bits	(taille classique, taille CAE)
4	(193, 270), (194, 271), (195, 273), (196, 274), (197, 275), (198, 277), (199, 278), (200, 280), (201, 281), (202, 282), (203, 284), (204, 285), (205, 287), (206, 288), (207, 289), (208, 291), (209, 292), (210, 294), (211, 295), (212, 296), (213, 298), (214, 299), (215, 301), (216, 302), (217, 303), (218, 305), (219, 306), (220, 308), (221, 309), (222, 310), (223, 312), (224, 313), (225, 315), (226, 316), (227, 317), (228, 319), (229, 320)
5	(257, 359), (258, 361), (259, 362), (260, 364), (261, 365), (262, 366), (263, 368), (264, 369), (265, 371), (266, 372), (267, 373), (268, 375), (269, 376), (270, 378), (271, 379), (272, 380), (273, 382), (274, 383)
8	(449, 628), (450, 630), (451, 631), (452, 632), (453, 634), (454, 635), (455, 637), (456, 638), (457, 639)

Table 1.9 – Couples $(t, 1.4t)$ pour lesquels la méthode CAE (X, Y)-only sur des entiers de $1.4t$ bits est plus efficace que la méthode Comb-2 sur des entiers de t bits.

pré-calculs. Nous en redonnons ici une description ce qui nous permettra de corriger une "légère" erreur dans le papier d'origine concernant le nombre moyen d'opérations nécessaires.

L'idée de base est de découper l'entier k de ℓ bits en h paquets de a bits avec $a = \lceil \frac{\ell}{h} \rceil$. On a donc :

$$k = \sum_{i=0}^{h-1} k^{(i)} 2^{ia}, \quad (k^{(i)}, \text{paquet de } a \text{ bits}) .$$

Chaque paquet $k^{(i)}$ de a bits est ensuite découpé en v paquets de b bits avec $b = \lceil \frac{a}{v} \rceil$,

$$k^{(i)} = \sum_{j=0}^{v-1} k^{(i,j)} 2^{jb}, \quad (k^{(i,j)}, \text{paquet de } b \text{ bits}) .$$

Et on a :

$$k^{(i,j)} = \sum_{\lambda=0}^{b-1} k^{(i,j,\lambda)} 2^{\lambda}, \quad k^{(i,j,\lambda)} \in \{0,1\} .$$

Ainsi

$$\begin{array}{rcl} kP & = & (\sum_{i=0}^{h-1} (\sum_{j=0}^{v-1} (\sum_{\lambda=0}^{b-1} k^{(i,j,\lambda)} 2^{\lambda}) 2^{jb}) 2^{ia}) P \\ & = & \sum_{\lambda=0}^{b-1} 2^{\lambda} (\sum_{j=0}^{v-1} (\sum_{i=0}^{h-1} k^{(i,j,\lambda)} 2^{jb+ia}) P) . \end{array}$$

Posons :

- $K_{0,0} = \mathcal{O}_E$,
- $K_{0,\delta} = (\sum_{i=0}^{h-1} \delta_i 2^{ia}) P$, pour tout $\delta = \sum_{i=0}^{h-1} \delta_i 2^i \neq 0$,
- $K_{j,\delta} = 2^{jb} \mathbb{K}_{0,\delta}$ pour $1 \leqslant j \leqslant v-1$ et pour tout $\delta = \sum_{i=0}^{h-1} \delta_i 2^i \neq 0$,
- $\mu^{(j,\lambda)} = \sum_{i=0}^{h-1} k^{(i,j,\lambda)} 2^i$, pour $0 \leqslant j \leqslant v-1$ et pour tout $0 \leqslant \lambda \leqslant b-1$,

Remarquons que $\forall 0 \leqslant j \leqslant v-1$, $K_{j,0} = \mathcal{O}_E$. On a :

$$K_{j,\mu^{(j,\lambda)}} = (\sum_{i=0}^{h-1} k^{(i,j,\lambda)} 2^{jb+ia}) P ,$$

et

$$kP = \sum_{\lambda=0}^{b-1} 2^{\lambda} (\sum_{j=0}^{v-1} K_{j,\mu^{(j,\lambda)}}) .$$

En supposant que l'on puisse stocker les $(2^h - 1)v$ points $K_{j,\delta}$, ce dernier résultat montre que l'on peut calculer kP en utilisant l'algorithme classique "double-and-add" où, à chaque étape, il faudra calculer $\sum_{j=0}^{v-1} K_{j,\mu^{(j,\lambda)}}$ et ne considérer dans cette somme que les points $K_{j,\mu^{(j,\lambda)}}$ tels que $\mu^{(j,\lambda)} \neq 0$ (cf. algorithme 13). Afin d'obtenir une évaluation précise du coût de cet algorithme, il faut s'intéresser de plus près au découpage effectué. (cf. figure 1.8). Chaque bloc de taille a a été découpé en v blocs de taille b avec $b = \lceil \frac{a}{v} \rceil$. Ainsi on a $bv \geqslant a$ et donc le dernier bloc de taille b débute par $bv - a$ bits nuls (en partant de la gauche). Or

$$\mu^{(v-1,\lambda)} = k^{(0,v-1,\lambda)} + k^{(1,v-1,\lambda)} \times 2 + \cdots + k^{(h-1,v-1,\lambda)} \times 2^{h-1} .$$

Ceci correspond à prendre pour chaque bloc de taille a, le bit qui se trouve en position λ dans le dernier bloc de taille b et de former l'entier dont la décomposition en base 2 correspond à la concaténation de ces bits. Comme les $bv - a$ premiers bits (en partant de

Algorithm 13 Méthode comb

Require: b, v, $K_{j,\delta}$, $0 \leqslant j \leqslant v-1$, $0 < \delta < 2^h$.
Ensure: $Q = kP$

```
Q = O_E        /* point à l'infini */
for λ = b − 1 down to 0 do
    Q = 2Q
    for j = v − 1 down to 0 do
        if μ^(j,λ) ≠ 0 then
            Q = Q + K_{j,μ^(j,λ)}
        end if
    end for
end for
return Q
```

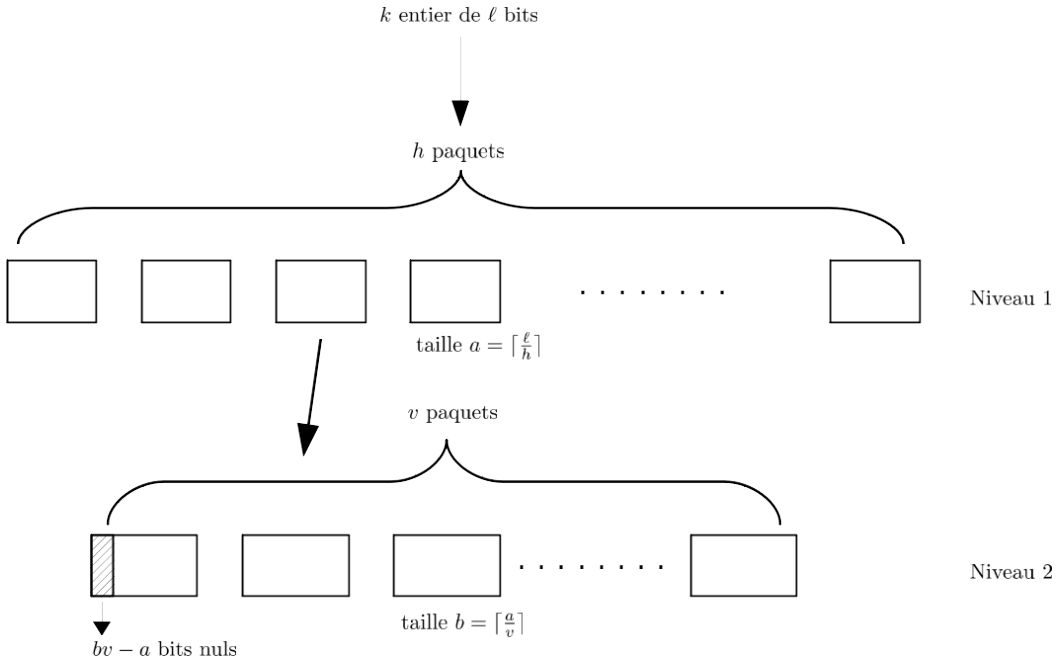


Figure 1.8 – Découpage de la méthode Comb.

la gauche) de chacun de ces derniers blocs de taille b valent 0, on a donc :

$$\mu^{(v-1,\lambda)} = 0 \quad \forall b + a - bv \leqslant \lambda \leqslant b - 1\,.$$

On peut donc scinder la boucle sur la variable λ en 2 parties : la première concerne les $bv-a$ premiers pas de la boucle principale pour lesquels il est inutile de considérer le cas $j = v-1$.

La deuxième concerne les pas suivants. De plus, le point Q étant initialisé avec le point à l'infini, le premier doublement est inutile. L'addition qui suit est réalisée pour $j = v - 2$ (d'après la remarque précédente) et consiste donc à initialiser Q par $K_{v-2,\mu^{(v-2,b-1)}}$ (cf. algorithme 14). L'analyse de l'algorithme 14 permet d'obtenir la complexité suivante :

Algorithm 14 Méthode comb $2^{\text{ème}}$ version

Require: b, v, $K_{j,\delta}$, $0 \leqslant j \leqslant v-1$, $0 < \delta < 2^h$.
Ensure: $Q = kP$

```
1:  Q = K_{v-2,μ^(v-2,b-1)}
2:  for j = v − 3 down to 0 do
3:      if μ^(j,λ) ≠ 0 then
4:          Q = Q + K_{j,μ^(j,λ)}
5:      end if
6:  end for
7:  for λ = b − 2 down to b + a − bv do
8:      Q = 2Q
9:      for j = v − 2 down to 0 do
10:         if μ^(j,λ) ≠ 0 then
11:             Q = Q + K_{j,μ^(j,λ)}
12:         end if
13:     end for
14: end for
15: for λ = b + a − bv − 1 down to 0 do
16:     Q = 2Q
17:     for j = v − 1 down to 0 do
18:         if μ^(j,λ) ≠ 0 then
19:             Q = Q + K_{j,μ^(j,λ)}
20:         end if
21:     end for
22: end for
23: return Q
```

— ligne 4 : au plus $v - 2$ additions de points,
— ligne 8 : $bv - a - 1$ doublements de points,
— ligne 11 : au plus $bv^2 - bv - av + a - v + 1$ additions de points,
— ligne 16 : $b + a - bv$ doublements de points,
— ligne 19 : au plus $bv + av - bv^2$ additions de points.

Ainsi le calcul de kP nécessite exactement $b - 1$ doublements de points et au plus $a - 1$ additions de points. Les entiers $\mu^{(j,\lambda)}$ pour $j \neq v - 1$ valent 0 si chacun des bits $k^{(i,j,\lambda)}$ vaut 0 pour $0 \leqslant i \leqslant h - 1$ ce qui arrive avec une probabilité de $\frac{1}{2^h}$. On peut donc estimer

qu'en moyenne le calcul de kP nécessite :

$$\frac{2^h-1}{2^h}(a-1) \text{ additions de points } + (b-1) \text{ doublements de points}\,.$$

Afin de limiter le nombre de points à stocker (et le nombre de calculs à effectuer), en pratique le découpage de niveau 2 (v blocs de taille b) n'est pas effectué. C'est à dire que l'on se place dans le cas $v = 1$ et donc $b = a$. On découpe donc k en h paquets de a bits, avec $a = \lceil \frac{\ell}{h} \rceil$ (cf. figure 1.9). On a donc :

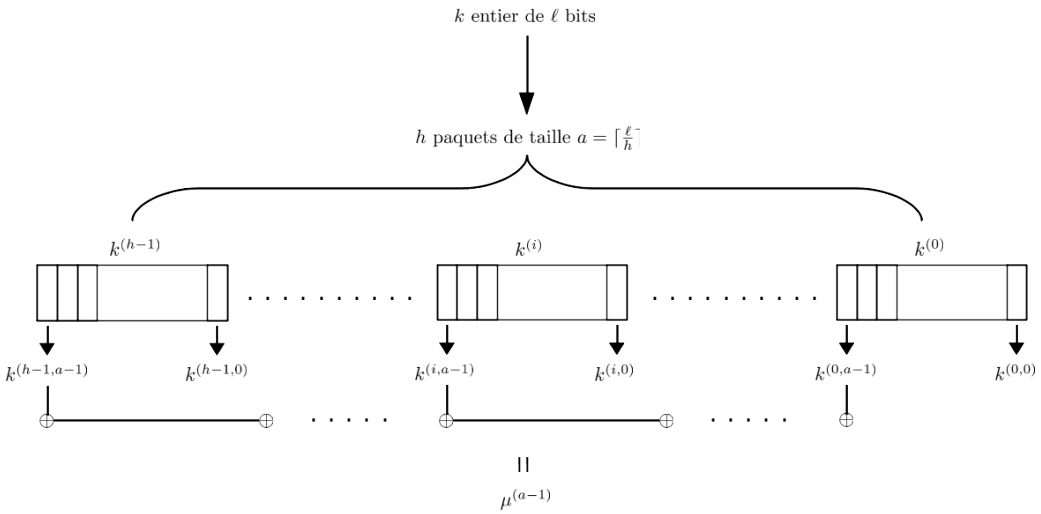


Figure 1.9 – Découpage en pratique de la méthode Comb.

$$k = \sum_{i=0}^{h-1} k^{(i)} 2^{ia}, \quad (k^{(i)}, \text{paquet de } a \text{ bits})\,,$$

et

$$k^{(i)} = \sum_{\lambda=0}^{a-1} k^{(i,\lambda)} 2^{\lambda}, \quad k^{(i,\lambda)} \in \{0,1\}\,.$$

Ainsi

$$\begin{aligned} kP &= \big(\textstyle\sum_{i=0}^{h-1} \big(\sum_{\lambda=0}^{a-1} k^{(i,\lambda)} 2^{\lambda}\big) 2^{ia}\big) P \\ &= \textstyle\sum_{\lambda=0}^{a-1} 2^{\lambda} \big(\sum_{i=0}^{h-1} k^{(i,\lambda)} 2^{ia} P\big)\,. \end{aligned}$$

Posons :

- $K_\delta = \big(\sum_{i=0}^{h-1} \delta_i 2^{ia}\big) P$, pour tout $\delta = \sum_{i=0}^{h-1} \delta_i 2^i$,
- $\mu^{(\lambda)} = \sum_{i=0}^{h-1} k^{(i,\lambda)} 2^i$, pour tout $0 \leqslant \lambda \leqslant a-1$.

Il n'y a ici aucune raison que $\mu^{(a-1)}$ qui représente la somme pondérée des bits de poids forts de chacun des h paquets soit égale à 0. On a :

$$K_{\mu^{(\lambda)}} = \left(\sum_{i=0}^{h-1} k^{(i,\lambda)} 2^{ia} \right) P \,,$$

et

$$kP = \sum_{\lambda=0}^{a-1} 2^{\lambda} K_{\mu^{(\lambda)}} \,.$$

En supposant que l'on puisse stocker les $(2^h - 1)$ points K_δ, on obtient alors, en réadaptant l'algorithme 13, la méthode de calcul décrite dans l'algorithme 15. La complexité est de $(a-1)$ doublements de points et au plus $(a-1)$ additions de points. Comme pour le découpage général, la probabilité pour que $\mu^{(\lambda)}$ soit égal à 0 vaut $\frac{1}{2^h}$, ainsi la complexité moyenne de l'algorithme est de :

$$\frac{2^h - 1}{2^h}(a-1) \text{ additions de points } + (a-1) \text{ doublements de points} \,.$$

Algorithm 15 Méthode comb, version découpage niveau 1

Require: h, a, K_δ, $1 \leqslant \delta \leqslant 2^h$
Ensure: $Q = kP$
1: $Q = K_{\mu^{(a-1)}}$
2: **for** $\lambda = a - 2$ **down to** 0 **do**
3: $\quad Q = 2Q$
4: $\quad$ **if** $\mu^{(\lambda)} \neq 0$ **then**
5: $\quad\quad Q = Q + K_{\mu^{(\lambda)}}$
6: $\quad$ **end if**
7: **end for**
8: **return** Q

1.5.2.5 Un biais statistique ?

Nous terminons cette partie en précisant que l'ensemble des entiers k calculables par une CAE de taille n contient asymptotiquement en moyenne deux fois plus d'entiers impairs que d'entiers pairs.

Proposition 1.15

Soit $(c_i)_{i=1\ldots n}$ la représentation binaire d'une CAE calculant un entier k, on a

. si n est impair, $\Pr(k \text{ impair}) = \frac{2^{n+1}-1}{3.2^n}$ et $\Pr(k \text{ pair}) = \frac{2^n+1}{3.2^n}$
. si n est pair, $\Pr(k \text{ impair}) = \frac{2^{n+1}+1}{3.2^n}$ et $\Pr(k \text{ pair}) = \frac{2^n-1}{3.2^n}$

Démonstration. Partant du couple $(1, 2)$, les couples d'entiers (v,u) obtenus lors du déroulement de l'algorithme 2 page 10 vérifient pgcd(u,v)=1. Ainsi, à chaque étape de l'algorithme on ne peut être que dans l'un des trois états suivants :

1. v et u sont impairs,
2. v est impair et u est pair,
3. v est pair et u est impair.

La matrice M ci-dessous donne la probabilité m_{kj} de transition d'un état j vers un état k en supposant que pour tout i , $\Pr(c_i = 1)=\frac{1}{2}$.

$$M = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ 1 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

En diagonalisant cette matrice, on obtient

$$M^n = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & -2 \\ 1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & (-1/2)^n \end{pmatrix} \begin{pmatrix} 1/3 & 1/3 & 1/3 \\ 1 & 0 & -1 \\ 2/3 & -1/3 & -1/3 \end{pmatrix}$$

Partant de l'état initial $X_0 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$, on obtient donc après n étapes le vecteur de probabilités :

$$X_n = \begin{pmatrix} -\frac{1}{3}\left(-\frac{1}{2}\right)^n + \frac{1}{3} \\ \frac{2}{3}\left(-\frac{1}{2}\right)^n + \frac{1}{3} \\ -\frac{1}{3}\left(-\frac{1}{2}\right)^n + \frac{1}{3} \end{pmatrix}.$$

L'entier k étant égal à la somme des éléments du dernier couple (v, u) obtenu on en déduit que

$$\Pr(k \text{ pair}) = -\frac{1}{3}\left(-\frac{1}{2}\right)^n + \frac{1}{3} \quad \text{et} \quad \Pr(k \text{ impair}) = \frac{1}{3}\left(-\frac{1}{2}\right)^n + \frac{2}{3}.$$

Selon la parité de n, on obtient les résultats énoncés. □

1.6 Cas d'un point P variable

1.6.1 Généralisation du résultat d'injectivité

L'application χ, définie page 32, associe à toute représentation binaire $(c_1, \ldots, c_n)$ d'une CAE, l'entier k correspondant. Avec les notations de la page 8, on a donc

$$\chi(c) = (1,2) \prod_{i=1}^{n} S_{c_i} \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Dans la section précédente, nous avons vu qu'en partant du couple (F_{n+2}, F_{n+3}), au lieu du couple $(1,2)$, alors cette application était injective ; ce qui n'est pas le cas si l'on part de $(1,2)$ car comme on l'a démontré précédemment, on a par exemple $\chi(c_1, \ldots, c_n) = \chi(c_n, \ldots, c_1)$. Une question naturelle est de se demander s'il existe d'autres couples (a,b) pour lesquels en partant de (a,b) l'application est injective. Notons pour la suite

$$\begin{array}{rccl} \chi_{a,b} & : & \{0,1\}^s & \longrightarrow & \mathbb{N} \\ & & (c_1, \ldots, c_n) & \longmapsto & (a,b) \prod_{i=1}^{n} S_{c_i} \begin{pmatrix} 1 \\ 1 \end{pmatrix}. \end{array}$$

Remarquons que si a et b ne sont pas premiers entre eux alors $\chi_{a,b}(c) = d\chi_{a',b'}(c)$ avec d =pgcd(a,b) et pgcd(a',b')=1. Ainsi, on ne considèrera pour la suite que les couples (a,b) où a et b sont premiers entre eux. Dans [Dos+18, Proposition 2], nous établissons le résultat suivant :

Proposition 1.16

Soient a et b deux entiers premiers entre eux. Si $a > F_{n+2}$ ou $b > F_{n+2}$, l'application $\chi_{a,b}$ est injective.

Cette proposition repose sur un lemme important ([Dos+18, Lemme 1]) lorsque l'on souhaite utiliser ce résultat d'injectivité dans le contexte des courbes elliptiques :

Lemme 1.17

Soit μ, l'application définie par

$$\begin{array}{rccl} \mu : & \{0,1\}^n & \longrightarrow & \mathbb{N}^2 \\ & c & \mapsto & \begin{pmatrix} x \\ y \end{pmatrix} = \prod_{i=1}^{n} S_{c_i} \begin{pmatrix} 1 \\ 1 \end{pmatrix}. \end{array}$$

Pour tout $c \in \{0,1\}^n$, on a

$$(x \leqslant F_{n+2} \text{ et } y \leqslant F_{n+1}) \quad \text{ou} \quad (x \leqslant F_{n+1} \text{ and } y \leqslant F_{n+2})\,.$$

Si le résultat de la proposition 1.16 permet d'affirmer que pour tout $c \neq c'$, $\chi_{a,b}(c) \neq \chi_{a,b}(c')$ (avec $a > F_{n+2}$ ou $b > F_{n+2}$), il ne permet pas de conclure qu'en partant du couple de points (aP, bP), on obtient à partir de l'algorithme 12, page 35, un point $\chi_{a,b}(c)P$ différent du point $\chi_{a,b}(c')P$. Si $\chi_{a,b}(c)$ et $\chi_{a,b}(c')$ sont tous les deux strictement inférieurs à l'ordre du point P, l'injectivité au niveau du calcul des points est assurée. C'est à ce niveau qu'intervient le lemme 1.17. Il permet d'établir qu'en partant d'un couple (a,b) l'entier k calculé satisfait :

$$k \leqslant aF_{n+2} + bF_{n+1} \quad \text{ou} \quad k \leqslant aF_{n+1} + bF_{n+2}\,.$$

On en déduit ainsi le corollaire suivant [Dos+18, Corollaire 1] :

Corollaire 1.18

Soit $E(\mathbb{F}_p)$ une courbe elliptique et P un point d'ordre N. Soit $n \geqslant 2$, et a et b, deux entiers tels que :

- pgcd(a,b)=1,
- $a > F_{n+2}$ ou $b > F_{n+2}$,
- $aF_{n+2} + bF_{n+1} < N$ et $aF_{n+1} + bF_{n+2} < N$.

En appliquant l'algorithme 12, page 35, à tous les éléments de $\{0,1\}^n$, en prenant pour points de départ (aP, bP) au lieu de $(F_{n+2}P, F_{n+3}P)$, on obtient 2^n points distincts.

1.6.2 CAE et courbes munies d'un endomorphisme "facilement calculable"

Dans le contexte où le point P est variable, partir de (aP, bP) afin de calculer $\chi_{a,b}(c)P$ n'a aucun intérêt puisque cela nécessite de calculer tout d'abord aP et bP. Le corollaire 1.18 permet de choisir $a = 1$ si $b > F_{n+2}$; il ne reste donc plus qu'à devoir effectuer le calcul de bP.

En 2001, Gallant, Lambert et Vanstone [GLV01] propose d'utiliser un endomorphisme Φ "facilement calculable" de la courbe E afin de découper le calcul de kP en

$$k_1 P + k_2 \Phi(P), \qquad \text{avec} \quad \max\{|k_1|, |k_2|\} = \mathcal{O}(\sqrt{n})\,.$$

Soit N l'ordre de P, si Φ est un endomorphisme de la courbe et si λ est une racine modulo N du polynôme caractéristique de Φ, alors $\Phi(P) = \lambda P$. Le polynôme caractéristique d'un

endomorphisme étant de degré 2, on le notera pour la suite $X^2 + rX + s$. Dans [GLV01], les auteurs proposent un algorithme pour obtenir le couple (k_1, k_2) à partir de l'entier k. Le point $k_1P + k_2\lambda P$ peut alors être calculé en utilisant une méthode classique d'exponentiation simultanée [Möl01]. L'efficacité de ce procédé (classiquement appelé *méthode GLV*) repose sur l'hypothèse que le calcul de λP peut se faire de façon "efficace" (idéalement il doit être moins coûteux que 5 doublements de points [GLV01]). Dans [Coh10, paragraphe 7.2.3], des exemples d'endomorphisme "facilement calculables" (1 seule multiplication sur $\mathbb{F}_p$) sont donnés (on note $\mathcal{O}$ l'élément neutre de la loi de groupe définie sur la courbe) :

- pour $p \equiv 1 \pmod 3$ et la courbe E d'équation $y^2 = x^3 + b$, l'application Φ définie par $\Phi(x, y) = (\beta x, y)$ et $\Phi(\mathcal{O}) = \mathcal{O}$ est un endomorphisme avec β un élément d'ordre 3. Si P est un point d'ordre N, alors pour tout $Q \in \langle P \rangle$ on a $\Phi(Q) = \lambda Q$ où λ est une racine modulo N de $X^2 + X + 1$.
- pour $p \equiv 1 \pmod 4$ et la courbe E d'équation $y^2 = x^3 + ax$, l'application Φ définie par $\phi(x, y) = (-x, \alpha y)$ et $\Phi(\mathcal{O}) = \mathcal{O}$ est un endomorphisme avec α un élément d'ordre 4. Si P est un point d'ordre N, alors pour tout $Q \in \langle P \rangle$ on a $\Phi(Q) = \lambda Q$ où λ est une racine modulo N de $X^2 + 1$.

Soit E une courbe munie d'un endomorphisme Φ "facilement calculable", tel que $\Phi(P) = \lambda P$, le calcul de bP devient négligeable si on choisit $b = \lambda$. Cependant, à ce stade il n'y a aucune raison pour que la valeur λ satisfasse la troisième condition du corollaire 1.18, à savoir $F_{n+1} + bF_{n+2} < N$ (cette dernière condition implique que $F_{n+2} + bF_{n+1} < N$, puisque $b > 1$). Les conditions établies dans ce corollaire permettent d'assurer que tous les points $\chi_{a,b}(c)P$ pour $c \in \{0, 1\}^n$ vérifient $\chi_{a,b}(c) < \mathrm{ord}(P)$, ce qui, combiné avec l'injectivité de $\chi_{a,b}$ permet d'établir que l'on obtient bien 2^n points distincts. Mais ceci n'est pas une condition nécessaire pour obtenir ce résultat. En effet, pour montrer que l'on obtient bien 2^n points distincts, il suffit de prouver que :

$$\forall (c, c') \in \{0, 1\}^n \times \{0, 1\}^n,\ c \neq c' \Rightarrow \chi_{a,b}(c) \neq \chi_{a,b}(c') \bmod \mathrm{ord}(P)\,. \tag{1.4}$$

La proposition suivante, issue d'un résultat intermédiaire établi dans la section 2.1 de [SCQ03], va nous permettre d'établir les conditions permettant d'assurer la relation (1.4) pour $(a, b) = (1, \lambda)$.

Proposition 1.19. *Soit P un point d'ordre N d'une courbe E muni d'un endomorphisme Φ de polynôme caractéristique $X^2 + rX + s$, tel que $\Phi(P) = \lambda P$. Soit $(k_1, k_2) \in \mathbb{Z}^2$, si $k_1 + k_2\lambda \equiv 0 \pmod N$, alors*

$$(k_1, k_2) \neq (0, 0) \Rightarrow \max(|k_1|, |k_2|) \geqslant \sqrt{\frac{N}{1 + |r| + s}}\,.$$

On obtient alors le résultat suivant :

Proposition 1.20

En considérant les notations de la proposition précédente, si $N > F_{n+2}^2(1+|r|+s)$, l'algorithme 12, page 35, appliqué à tous les éléments de $\{0,1\}^n$, en prenant pour points de départ $(P, \Phi(P))$ au lieu de $(F_{n+2}P, F_{n+3}P)$, calcule 2^n points distincts. De plus l'ensemble des points intermédiaires (U_1, U_2) calculés lors du déroulement de l'algorithme vérifient bien $U_1 \neq U_2$, $U_1 \neq \mathcal{O}_E$ et $U_2 \neq \mathcal{O}_E$.

Démonstration. Soit $c \in \{0,1\}^n$, en partant de $(P, \lambda P)$, l'algorithme 12 calcule le point $\chi_{1,\lambda}(c)P = ((1,\lambda)\mu(c))P$. Il existe donc $(k_1, k_2) \in \mathbb{N}^2$ tels que $\chi_{1,\lambda}(c) = k_1 + k_2\lambda$, avec $\mu(c) = \begin{pmatrix} k_1 \\ k_2 \end{pmatrix}$. Supposons que pour $c' \in \{0,1\}^n$, $c' \neq c$, on ait

$$k_1 + k_2\lambda \equiv k_1' + k_2'\lambda \pmod{N},$$

on a alors

$$(k_1 - k_1') + (k_2 - k_2')\lambda \equiv 0 \pmod{N}.$$

Or, d'après le lemme 1.17, page 48, on a $\forall i \in \{1,2\}$, $k_i \leqslant F_{n+2}$ et $k_i' \leqslant F_{n+2}$, donc $|k_i - k_i'| \leqslant F_{n+2}$. La contrainte imposée sur l'ordre N du point P implique que

$$\forall i \in \{1,2\},\ |k_i - k_i'| < \sqrt{\frac{N}{1+|r|+s}}.$$

D'après la proposition précédente, ceci implique que $k_1 = k_1'$ et $k_2 = k_2'$, i.e. $\mu(c) = \mu(c')$. L'injectivité de μ (cf. [Dos+18, Proposition 1]) permet de conclure que l'on calcule bien 2^n points distincts.

Il est important à ce stade de s'assurer que les points intermédiaires U_1 et U_2 sont eux aussi bien tous distincts. Dans le cas contraire la formule d'addition ZADDU serait incorrecte. De même, si l'un des deux points est égal à l'élément neutre, la formule d'addition n'est plus valable. Les points intermédiaires U_1 et U_2 sont de la forme vP et uP où v et u sont deux entiers obtenus en partant de $(1,\lambda)$ et en lisant un certain nombre a et b ($a < b$) de bits de la séquence c (sauf pour P, λP et $(\lambda+1)P$) (cf. fig. 1.10).

Par exemple, dans la figure 1.10, pour le couple de points $((\lambda+1)P, (\lambda+2)P)$, on a $\lambda + 2 = \chi_{1,\lambda}(1)$ et $\lambda + 1$ est obtenu directement à partir de $(1,\lambda)$. Pour le couple de points $((2\lambda+1)P, (3\lambda+1)P)$ (obtenu en lisant la séquence 010), on a $2\lambda + 1 = \chi_{1,\lambda}(0)$ et $3\lambda + 1 = \chi_{1,\lambda}(01)$. Etant donné que les points P, λP et $(\lambda+1)P$ sont distincts 2 à 2, si on suppose qu'il existe deux points U_1 et U_2 tels que $U_1 = U_2$, il convient d'étudier les cas suivants :

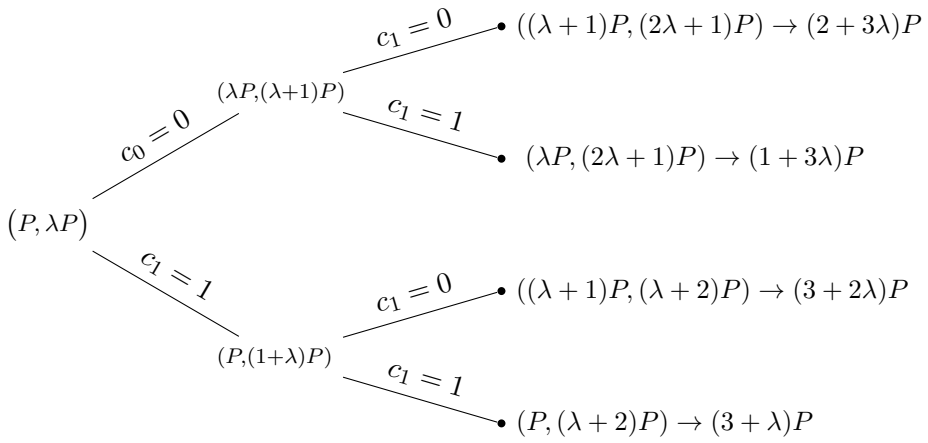


Figure 1.10 – Déroulement du calcul d'une CAE en partant de $(P, \lambda P)$.

Cas 1 : il existe $a \geqslant 1$ tel que $P = \chi_{1,\lambda}(c_1 \dots c_a)P$. On en déduit l'existence de 2 entiers k_1 et k_2 tels que $1 \equiv k_1 + k_2\lambda \pmod N$, i.e.

$$k_1 - 1 + k_2\lambda \equiv 0 \pmod N .$$

Cas 2 : il existe $a \geqslant 1$ tel que $\lambda P = \chi_{1,\lambda}(c_1 \dots c_a)P$. On en déduit l'existence de 2 entiers k_1 et k_2 tels que $\lambda \equiv k_1 + k_2\lambda \pmod N$, i.e.

$$k_1 + (k_2 - 1)\lambda \equiv 0 \pmod N .$$

Cas 3 : il existe $a \geqslant 1$ tel que $(\lambda+1)P = \chi_{1,\lambda}(c_1 \dots c_a)P$. On en déduit l'existence de 2 entiers k_1 et k_2 tels que $1 + \lambda \equiv k_1 + k_2\lambda \pmod N$, i.e.

$$k_1 - 1 + (k_2 - 1)\lambda \equiv 0 \pmod N .$$

Cas 4 : il existe $a \geqslant 1$ et $b > a$ tels que $\chi_{1,\lambda}(c_1 \dots c_a)P = \chi_{1,\lambda}(c_1 \dots c_b)P$. On en déduit l'existence de 4 entiers k_1, k_2, k'_1 et k'_2 tels que $k_1 + k_2\lambda \equiv k'_1 + k'_2\lambda \pmod N$, i.e.

$$(k_1 - k'_1) + (k_2 - k'_2)\lambda \equiv 0 \pmod N .$$

Etant donné que $\forall i \in \{1, 2\}$, $k_i \leqslant F_{n+2}$ et $k'_i \leqslant F_{n+2}$, on a donc en particulier $|k_i|$, $|k_i - 1|$ et $|k_i - k'_i|$ bornées par F_{n+2}, et la contrainte imposée sur N nous permet de déduire que toutes ces valeurs sont strictement bornées par $\sqrt{\frac{N}{1+|r|+s}}$. La proposition 1.19 page 50 [SCQ03, section 2.1] permet une fois encore de conclure que :
Cas 1 : $k_1 = 1$ et $k_2 = 0$,
Cas 2 : $k_1 = 0$ et $k_2 = 1$,

Cas 3 : $k_1 = 1$ et $k_2 = 1$,
Cas 4 : $k_1 = k_1'$ et $k_2 = k_2'$.
Dans les trois premiers cas, on en déduit que :

$$\begin{pmatrix} k_1 \\ k_2 \end{pmatrix} = \prod_{i=1}^{a} S_{c_i} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ ou } \begin{pmatrix} 0 \\ 1 \end{pmatrix} \text{ ou } \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

ce qui est impossible car la multiplication par S_0 ou S_1 du vecteur $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ fait croître au moins l'une des 2 composantes. Dans le dernier cas, étant donné que les matrices S_0 et S_1 sont inversibles, on obtient

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \prod_{i=a+1}^{b} S_{c_i} \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

ce qui une fois encore est impossible car au moins une matrice S_0 ou S_1 intervient dans cette expression. Ainsi on a toujours $U_1 \neq U_2$. Finalement, supposons que $U_1 = \mathcal{O}_E$ (resp. $U_2 = \mathcal{O}_E$), ceci signifie qu'il existe k_1 et k_2 tels que $k_1 + k_2\lambda \equiv 0 \pmod{N}$. Les conditions données sur N et la proposition 1.19 page 50 permettent de conclure que $k_1 = k_2 = 0$, ce qui est impossible. □

1.6.3 Encore une question de taille

Pour assurer une sécurité de ℓ bits (et donc calculer $2^{2\ell}$ points distincts), il faut choisir une courbe et un point P d'ordre $N > F_{2\ell+2}^2(1 + |r| + s)$. En utilisant le fait que $F_{2\ell+2} = \lfloor \frac{\varphi^{2\ell+2}}{\sqrt{5}} \rfloor$ avec $\varphi = \frac{1+\sqrt{5}}{2}$, il suffit de choisir $N > \frac{\varphi^{4\ell+4}}{5}(1 + |r| + s)$. Pour des valeurs r et s telles que $1 + |r| + s \leqslant 4$, on peut donc choisir $N > \frac{4}{5}\varphi^{4\ell+4}$. Ainsi la taille de l'entier N est égale à $\lceil 4\ell \log_2 \varphi + 4 \log_2 \varphi + 2 - \log_2(5) \rceil$, ce qui donne une taille égale à $\lceil 2.777\ell + 2.455 \rceil$. Le théorème de Hasse nous permet de déduire à quelques bits près la taille du corps sur lequel on devra effectuer les calculs si le cofacteur de la courbe vaut 1 (cf. table 1.10), il faudra sinon prendre en compte la valeur de ce cofacteur. Concernant le niveau de sécurité, on peut donc énoncer une condition analogue à ce que l'on avait constaté dans le cas d'un point P fixe.

Niveau de sécurité	128	192	256
Taille min. du corps	358	536	714

Table 1.10 – Taille minimale du corps pour un niveau de sécurité donné, lorsque $1 + |r| + s \leqslant 4$.

Sécurité/Taille du corps

À niveau de sécurité équivalent, l'utilisation des CAE dans le contexte d'un point P variable et d'une courbe munie d'un endomorphisme "facilement" calculable a pour conséquence de devoir manipuler des entiers de taille environ 1.4 fois plus importante que les méthodes "classiques".

1.6.4 Performances théoriques obtenues

Notre algorithme de multiplication scalaire permet donc de calculer un certain point kP pour une courbe elliptique de Weierstraß, définie sur $\mathbb{F}_p$, munie d'un endomorphisme "facilement" calculable. Il semble donc naturel de comparer ses performances avec la méthode GLV qui possède exactement les mêmes caractéristiques. L'algorithme ZADDU est directement lié à l'utilisation du système de coordonnés jacobiennes, ce qui impose que notre méthode de calcul ne peut être utilisé que sur des courbes de Weierstraß. Concernant GLV, on peut très bien utiliser les courbes d'Edwards afin de tirer partie des formules d'addition de points qui sont particulièrement efficaces [His+08]. A ce stade, il convient de mentionner deux points importants :

Encodage du scalaire k

La méthode GLV nécessite une première étape de décomposition du scalaire k afin de trouver les entiers k_1 et k_2 tels que $k = k_1 + k_2\lambda \pmod N$, puis un "réencodage" des entiers k_1 et k_2 (joint sparse form [Sol01]) afin d'utiliser les algorithmes d'exponentiation simultanée. Bien que cette étape soit négligeable dans le cas où l'entier k est fixé, il doit être pris en compte lors de multiples calcul de kP avec k et P qui varient.

Résistance aux attaques SPA

La méthode GLV ainsi que le principe de réencodage ne sont pas résistants à des attaques de type SPA. Une version sécurisée (nommée GLV-SAC par la suite) est décrite dans [FLS15].

Dans [Dos+18], nous avons donc comparé les résultats de notre méthode avec la méthode GLV sécurisée appliquée à une courbe de Weierstraß (W-GLV-SAC) et la méthode GLV sécurisée appliquée à une courbe d'Edwards (TED-GLV-SAC). La table 1.11 donne les coûts théoriques des différentes méthodes. Dans le contexte où $S = M$ cette table permet d'établir que :

Méthode	Coût	Taille du corps
CAE	$(2\ell+1)(5M+2S)$	2.8ℓ
CAE (X,Y)-only	$(2\ell+1)(4M+2S)$	2.8ℓ
TED-GLV-SAV	$\ell \times (10M+4S)$	2ℓ
W-GLV-SAC	$\ell \times (9M+9S)$	2ℓ

Table 1.11 – Coût de la multiplication scalaire, P variable, ℓ bits de sécurité.

— W-GLV-SAC effectue environ $\frac{18}{14} \simeq 1.29$ fois plus de multiplications que la méthode CAE et $\frac{18}{12} = 1.5$ fois plus de multiplications que la méthode CAE (X,Y)- only,
— TED-GLV-SAC effectue environ le même nombre de multiplications que la méthode CAE et $\frac{14}{12} \simeq 1.17$ fois plus de multiplications que la méthode CAE (X,Y)- only.

Sur une plateforme logicielle ($S = M$), au vu du surcoût induit par une multiplication modulaire de deux entiers de 358 bits (cf. table 1.7, page 39), il semble donc que la méthode CAE (X,Y)-only puisse dans certains cas donner de meilleurs résultats que la méthode GLV-SAC et qu'il n'y ait aucun espoir d'être plus efficace que TED-GLV-SAC. Précisons que les données de la table 1.11 n'intègrent pas le coût de l'encodage du scalaire k pour GLV.

1.6.5 Performances pratiques obtenues

Dans [Dos+18], nous avons montré qu'en utilisant les librairies de calcul multiprécision classiques, la méthode CAE (X,Y)-only constituait une alternative sérieuse à GLV. Ce choix d'utiliser des librairies mutltiprécision "grand public" peut être critiqué mais force est de constater qu'elles sont employées dans de nombreuses applications cryptographiques de la vie réelle :

— la librairie OpenSSL utilise pour l'ensemble de ses opération arithmétiques la librairie BIGNUM,
— la librairie GnuTLS incluse dans les systèmes d'exploitation des produits de sauvegarde Synology repose quant à elle sur GnuMP,
— depuis 2014, le système Android propose une API Java cryptographique via la librairie Spongy Castle. Cette dernière repose sur la librairie BIGNUM pour l'arithmétique bas niveau.

La table 1.12 présente le ratio du temps d'exécution mesuré des différentes méthodes pour le calcul de 2^{17} multiplications scalaires et pour un niveau de sécurité de 128 bits. Sans surprise, la table 1.12 confirme le fait que la méthode CAE (X,Y)-only est plus efficace que W-GLV-SAC aussi bien sous Android que sur une plateforme x64. Les résultats plus étonnants sont les ratios obtenus sous Android face à la version GLV non sécurisée

Comparaison	Android	Gnu MP, x64
CAE (X, Y)-only / W-GLV (sans encodage)	**0.91**	1.23
CAE (X, Y)-only / W-GLV (avec encodage)	**0.87**	1.18
CAE (X, Y)-only / W-GLV-SAC (sans encodage)	**0.69**	**0.8**
CAE (X, Y)-only / W-GLV-SAC (avec encodage)	**0.67**	**0.86**
CAE (X, Y)-only /TED-GLV (avec encodage)	1.16	1.59
CAE (X, Y)-only /TED-GLV (avec encodage)	1.09	1.53
CAE (X, Y)-only /TED-GLV-SAC (sans encodage)	**0.84**	1.21
CAE (X, Y)-only /TED-GLV-SAC (avec encodage)	**0.82**	1.19

Table 1.12 – Rapport du temps d'exécution entre les méthodes GLV et CAE pour un niveau de sécurité de 128 bits.

et face à la version TED-GLV-SAC. Concernant cette dernière, le paragraphe précédent indique que TED-GLV-SAC effectue 1.17 fois plus de multiplications que CAE (X, Y)-only, mais les multiplications dans CAE (X, Y)-only coûtent 1.5 fois plus chères en utilisant la librairie BIGNUM d'après la table 1.7. Concernant la version non sécurisée de GLV, le nombre de multiplications effectuées pour un niveau de sécurité de ℓ bits est de 12.5 par bit de sécurité (cf. [Dos+18, Table 2]), ce qui est quasiment identique à ce qui est effectué dans la méthode CAE (X, Y)-only. Cette dernière ne devrait donc pas permettre d'obtenir de meilleures performances. Les performances que nous avons obtenues montrent qu'il faut être particulièrement rigoureux lorsque l'on souhaite établir la complexité de la multiplication scalaire et que la simple estimation qui consiste à se focaliser sur le coût de la multiplication dans le corps de base peut ne pas être suffisante selon le contexte d'implémentation.

1.6.5.1 De l'importance d'un décompte précis de toutes les opérations et de la gestion de la mémoire

Lors de l'utilisation de librairies de calcul multiprécision pour manipuler des grands entiers, le "véritable" coût d'une multiplication modulaire dépend de 2 facteurs :

— la complexité calculatoire de l'opération en elle-même qui est quadratique en le nombre de mots machines sur lequel sont représentées les opérandes,

— la gestion de la mémoire pour allouer et manipuler les différents blocs représentant un entier et dont le coût est linéaire en le nombre de blocs à manipuler.

Concernant la libraire Java Spongy Castle, intégrée dans le système Android, les opérations arithmétiques sont déléguées à la librairie BIGNUM écrite en assembleur et en

C, alors que la gestion mémoire est assurée par la machine virtuelle Java. Il en résulte que toute opération réalisant à un instant donné un accès mémoire voit son temps d'exécution global pénalisé par cet accès. La figure 1.11 représente la répartition du temps d'exécution des différentes étapes résultants de l'enchainement des appels des fonctions `BigInteger.multiply` et `BigInteger.mod` de la librairie Spongy Castle. On y voit clairement que la partie "calcul" (`NativeBN.BN_mul` et `NativeBN.BN_mod`) ne représente que 12.7% du coût total. Quand on mesure le ratio du temps d'exécution entre une multiplication modulaire de 2 entiers de 358 bits et 2 entiers de 256 bits, en utilisant les fonctions `BigInteger.multiply` et `BigInteger.mod` fournies par l'API, on trouve une valeur proche de 1.06 ([Dos+18, Table 4]), au lieu de la valeur 1.5 de la table 1.7 page 39. Cette gestion mémoire représentant une partie non négligeable du temps d'exécution de

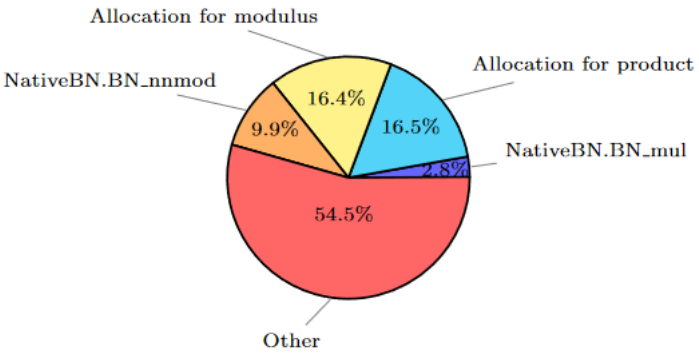


Figure 1.11 – Répartition du temps d'exécution des différentes étapes intervenant dans la multiplication modulaire de 2 entiers de 256 bits avec la librairie Spongy Castle.

toute opération, il est alors indispensable de prendre en compte l'ensemble des calculs effectués dans l'algorithme de multiplication scalaire. La méthode GLV effectue en moyenne ℓ doublements de points et $\ell/2$ additions de points pour un niveau de sécurité de ℓ bits. Les formules à utiliser sont celles qui correspondent au cas où le coefficient a de la courbe $y^2 = x^3 + ax + b$ vaut 0. Chaque addition a pour opérande un point Q intermédiaire (en coordonnées jacobiennes) et le point P en coordonnées affines (i.e $Z_P = 1$). Les tables 1.13 et 1.14 donnent les formules et le coût de ces opérations[2]. Il apparaît dans ce décompte que des additions sont effectuées, en moyenne 10.5 par bit de sécurité, alors qu'il n'y en a que 7 dans ZADDU (cf. page 26). De même, la méthode GLV nécessite en moyenne 7 multiplications par des constantes par bit de sécurité. Aucune opération de ce type n'est effectuée dans ZADDU. Dans la librairie `Big.Integer`, la procédure d'addition utilise les mêmes appels concernant la gestion mémoire que ceux effectués lors d'une multiplication.

2M + 5S + 6A + 3 ∗ 2 + 1 ∗ 3 + 1 ∗ 8

$A = X1^2$
$B = Y1^2$
$C = B^2$
$D = 2 * ((X1 + B)^2 - A - C)$
$E = 3 * A$
$F = E^2$
$X3 = F - 2 * D$
$Y3 = E * (D - X3) - 8 * C$
$Z3 = 2 * Y1 * Z1$

Table 1.13 – Coût et formules de doublement pour GLV.

7M + 4S + 9A + 3 ∗ 2 + 1 ∗ 4

$Z1Z1 = Z1^2$
$U2 = X2 * Z1Z1$
$S2 = Y2 * Z1 * Z1Z1$
$H = U2 - X1$
$HH = H^2$
$I = 4 * HH$
$J = H * I$
$r = 2 * (S2 - Y1)$
$V = X1 * I$
$X3 = r^2 - J - 2 * V$
$Y3 = r * (V - X3) - 2 * Y1 * J$
$Z3 = (Z1 + H)^2 - Z1Z1 - HH$

Table 1.14 – Coût et formules d'addition pour GLV ($Z_2 = 1$).

Dès lors, la seule différence entre les fonctions `BigInteger.multiply` et `BigInteger.add` se situe au niveau des appels de `NativeBN.BN_mul` et `NativeBN.BN_add`. Ainsi le coût d'une addition est pratiquement équivalent à celui d'une multiplication. Concernant les constantes, si la multiplication par 2, 4 ou 8 peut se faire à l'aide d'un ou plusieurs décalages, cette opération s'avère être non négligeable dès lors qu'elle nécessite la gestion des différents blocs mémoire représentant l'opérande à décaler. Ceci explique pourquoi la méthode CAE $(X, Y)-$ only est plus efficace que la méthode GLV classique non sécurisée, évidemment cette efficacité augmente face à la version W-GLV-SAC qui comporte déjà un nombre plus important de multiplications.

La même analyse peut être réalisée en ce qui concerne TED-GLV-SAC. Comme mentionné dans [His+08, section 4.3], le calcul efficace de kP est obtenu en mixant le système de coordonnées d'Edwards $\mathcal{E}$, où chaque point est représenté par un triplet (X, Y, Z), avec le système de coordonnées étendu d'Ewards $\mathcal{E}^e$, où chaque point est représenté par un quadruplet (X, Y, Z, T) avec $T = XY/Z$. L'algorithme de multiplication scalaire (cf. [Dos+18, Algorithme 7]) effectue à chaque itération un doublement suivi d'une addition avec l'un des deux points qui est en coordonnées affines et est donc représenté dans $\mathcal{E}^e$ par $(X, Y, 1, XY)$. Cet enchainement est effectué en mixant les systèmes de coordonnées comme suit :

$$\mathcal{E}^e \leftarrow 2\mathcal{E}$$

2. https://www.hyperelliptic.org/EFD/g1p/auto-shortw-jacobian-0.html

$\mathbf{4M + 4S + 6A + 1 * 2}$

$\mathtt{A} = \mathtt{X1}^2$
$\mathtt{B} = \mathtt{Y1}^2$
$\mathtt{C} = 2\mathtt{Z}_1^2$
$\mathtt{D} = -\mathtt{A}$
$\mathtt{E} = (\mathtt{X1} + \mathtt{Y1})^2 - \mathtt{A} - \mathtt{B}$
$\mathtt{G} = \mathtt{D} + \mathtt{B}$
$\mathtt{F} = \mathtt{G} - \mathtt{C}$
$\mathtt{H} = \mathtt{D} - \mathtt{B}$
$\mathtt{X3} = \mathtt{EF}$
$\mathtt{Y3} = \mathtt{GH}$
$\mathtt{Z3} = \mathtt{FG}$
$\mathtt{T3} = \mathtt{EH}$

Table 1.15 – Coût et formules de doublement pour TED-GLV.

$\mathbf{7M + 8A + 2 * 2}$

$\mathtt{A} = (\mathtt{Y1} - \mathtt{X1})(\mathtt{Y2} + \mathtt{X2})$
$\mathtt{B} = (\mathtt{Y1} + \mathtt{X1})(\mathtt{Y2} - \mathtt{X2})$
$\mathtt{C} = 2\mathtt{Z1T2}$
$\mathtt{D} = 2\mathtt{T1}$
$\mathtt{E} = \mathtt{D} + \mathtt{C}$
$\mathtt{F} = \mathtt{B} - \mathtt{A}$
$\mathtt{G} = \mathtt{B} + \mathtt{A}$
$\mathtt{H} = \mathtt{D} - \mathtt{C}$
$\mathtt{X3} = \mathtt{EF}$
$\mathtt{Y3} = \mathtt{GH}$
$\mathtt{Z3} = \mathtt{FG}$
$\mathtt{T3} = \mathtt{EH}$

Table 1.16 – Coût et formules d'addition pour TED-GLV ($Z_2 = 1$).

$$\mathcal{E} \leftarrow \mathcal{E}^e + \mathcal{E}^e$$

Les tables 1.15 et 1.16 donnent les formules et le coût des opérations de doublement et d'addition[3]. Etant donné que dans la formule de doublement le calcul de T_3 est indépendant de la coordonnée T du point à doubler, on peut donc ne pas effectuer dans la formule d'addition le calcul de T_3, ce qui fait gagner une multiplication et ramène sa complexité à $\mathbf{6M + 8A + 2 * 2}$. Le tableau 1.17 résume le décompte précis intervenant dans les différents protocoles. En considérant le ratio de 1.06 ([Dos+18, Table 4]) entre une multiplication de deux entiers de 358 bits et une multiplication de deux entiers de 256 bits, on peut estimer, dans le contexte $S = M$, la complexité de la méthode CAE (X, Y)-only à $13\ell M_{256} + 15\ell A_{256} + 6M_{358} + 7A_{358}$, où M_i (resp. A_i) représente une multiplication (resp. une addition) entre deux entiers de i bits. Ainsi, la différence entre le nombre d'opérations effectuées dans TED-GLV-SAC et le nombre d'opérations effectuées dans la méthode CAE (X, Y)-only est de l'ordre de $\ell M_{256} - \ell A_{256} + 3\ell D_{256} - 6M_{358} - 7A_{358}$, où D_{256} représente le coût d'une multiplication par 2 sur des entiers de 256 bits. Il s'avère que cette différence est positive en utilisant la librairie `Big.Integer`.

3. [His+08, section 3.2 et 3.3]

Méthode	Multiplications	Carrés	Additions	Multiplications par des constantes	Taille opérandes
CAE	$10\ell + 5$	$4\ell + 2$	$14\ell + 7$	0	2.8ℓ
CAE (X, Y)-only	$8\ell + 4$	$4\ell + 2$	$14\ell + 7$	0	2.8ℓ
GLV	5.5ℓ	7ℓ	10.5ℓ	7ℓ	2ℓ
GLV-SAC	9ℓ	9ℓ	15ℓ	9ℓ	2ℓ
TED-GLV-SAC	10ℓ	4ℓ	14ℓ	3ℓ	2ℓ

Table 1.17 – Comparaison de toutes les opérations effectuées sur le corps de base pour un niveau de sécurité de ℓ bits.

1.7 Un nouvel espoir

Le détail de la preuve de la proposition 1.20 laisse apparaître que la contrainte sur la taille du corps à utiliser pour assurer l'injectivité de l'algorithme de multiplication scalaire provient de la borne supérieure sur les entiers k_1 et k_2 qui est égale à F_{n+2}. Rappelons que $\binom{k_1}{k_2} = \mu(c)$ où c est une une chaîne quelconque de taille n. Afin de diminuer la taille du corps, une stratégie possible serait de travailler avec un sous-ensemble de l'ensemble de toutes les chaînes d'une taille donnée, de façon à diminuer la valeur de la borne supérieure sur k_1 et k_2 , tout en ayant suffisamment d'éléments pour assurer le niveau de sécurité voulu. Rappelons que dans le cas général, la borne supérieure sur k_1 et k_2 est atteinte à partir de la séquence c constituée de n zéros.

Définition 1.7

Soit ℓ et w deux entiers non-nuls. On note $\mathcal{M}_{\ell,\geqslant w}$ l'ensemble des éléments de $\{0,1\}^\ell$ dont le poids de Hamming est supérieur ou égal à w.

Dans [HMV21], nous montrons que pour les éléments $c \in \mathcal{M}_{2n,\geqslant n}$, on a :

Proposition 1.21

Soit $n \geqslant 3$,

$$\max\{\|\mu(c)\|_\infty \mid c \in \mathcal{M}_{2n,\geqslant n}\} = 8\alpha_{n-2} + 11\beta_{n-2}$$

avec $\alpha_m = \frac{(1+\sqrt{2})^m + (1-\sqrt{2})^m}{2}$ et $\beta_m = \frac{(1+\sqrt{2})^m - (1-\sqrt{2})^m}{2\sqrt{2}}$.

On en déduit alors une nouvelle version de la proposition 1.20 page 51 (cf. [HMV21, Théorème 2, Proposition 3]) :

Proposition 1.22

Soin $n \geqslant 3$ et E une courbe munie d'un endomorphisme ϕ dont le polynôme caractéristique est $X^2 + rX + s$. Soit P un point d'ordre N, si

$$N \geqslant (8\alpha_{n-2} + 11\beta_{n-2})^2 (1 + |r| + s)$$

alors, l'algorithme 16 [HMV21, Algorithme 3], appliqué à tous les éléments de $\mathcal{M}_{2n,\geqslant n}$, calcule 2^{2n-1} points distincts. De plus l'ensemble des points intermédiaires (U_1, U_2) calculés lors du déroulement de l'algorithme vérifient bien $U_1 \neq U_2$ et $U_i \neq \mathcal{O}_E$ pour $i \in \{1, 2\}$.

Si on applique cette proposition pour $n = \ell + 1$, on obtient alors un niveau de sécurité de ℓ bits pour notre algorithme de multiplication scalaire ($\ell + \frac{1}{2}$ exactement). Pour cela, il suffit que l'ordre N du point vérifie

$$N \geqslant (8\alpha_{\ell-1} + 11\beta_{\ell-1})^2 (1 + |r| + s)\,.$$

Dans le contexte où $1 + |r| + s \leqslant 4$, il suffit donc de choisir

$$\begin{aligned} N &\geqslant 4\,(8\alpha_{\ell-1} + 11\beta_{\ell-1})^2 \\ &\geqslant 4(64\alpha_{\ell-1}^2 + 176\alpha_{\ell-1}\beta_{\ell-1} + 121\beta_{\ell-1}^2)\,. \end{aligned}$$

Pour ℓ pair (hypothèse classique pour un niveau de sécurité), on a :

- $\frac{(1+\sqrt{2})^{\ell-1}}{2} > \alpha_{\ell-1}$,
- $\alpha_{\ell-1}\beta_{\ell-1} = \frac{(1+\sqrt{2})^{2\ell-2} - (1-\sqrt{2})^{2\ell-2}}{4\sqrt{2}}$, et donc $\frac{(1+\sqrt{2})^{2\ell-2}}{4\sqrt{2}} > \alpha_{\ell-1}\beta_{\ell-1}$,
- $\beta_{\ell-1}^2 = \frac{1}{8}((1+\sqrt{2})^{2\ell-2} + (1-\sqrt{2})^{2\ell-2} + 2)$ et donc $\frac{1}{8}((1+\sqrt{2})^{2\ell-2} + 3) > \beta_{\ell-1}^2$.

On en déduit que

$$16(1+\sqrt{2})^{2\ell-2} + \frac{44}{\sqrt{2}}(1+\sqrt{2})^{2\ell-2} + \frac{121}{8}(1+\sqrt{2})^{2\ell-2} + \frac{363}{8} > 64\alpha_{\ell-1}^2 + 176\alpha_{\ell-1}\beta_{\ell-1} + 121\beta_{\ell-1}^2\,.$$

En remarquant que $19(1+\sqrt{2})^{2\ell-2} > \frac{363}{8}$, on peut donc choisir N tel que :

$$N \geqslant 4\Big((1+\sqrt{2})^{2\ell-2}(16 + \frac{44}{\sqrt{2}} + \frac{121}{8} + 19)\Big).$$

ce qui donne

$$\log_2(N) \geqslant 2.54\ell + 5.8\,.$$

La proposition 1.22 impose de travailler dans l'ensemble $\mathcal{M}_{2n,\geqslant n}$. C'est à dire de consi-

Algorithm 16 EACsmult(X_0,Y_0, c)

Require: $\beta \in \mathbb{F}_p$, P en coordonnées affines
Ensure: Return $Q = (X_1, Y_1, Z)$ le point calculé à partir de $(P, \phi(P))$ et de la séquence binaire c

1: $w \leftarrow \text{poids}(c)$
2: **if** $w \leqslant \text{longueur}(c)/2$ **then**
3: $\quad b \leftarrow 0$
4: **else**
5: $\quad b \leftarrow 1$
6: **end if**
7: $X_1, Y_1, Z \leftarrow X_0.\beta, Y_0, 1$
8: **for** $i = 1 \ldots \text{length}(c)$ **do**
9: $\quad A \leftarrow (X_0 \oplus X_1) \times (c_i \oplus b)$
10: $\quad X_0, X_1 \leftarrow X_0 \oplus A, X_1 \oplus A$
11: $\quad A \leftarrow (Y_0 \oplus Y_1) \times (c_i \oplus b)$
12: $\quad Y_0, Y_1 \leftarrow Y_0 \oplus A, Y_1 \oplus A$
13: $\quad \text{ZADDu}(X_0, Y_0, X_1, Y_1, Z)$
14: **end for**
15: $\text{ZADDu}(X_0, Y_0, X_1, Y_1, Z)$
16: **return** X_1, Y_1, Z

dérer des CAE contenant plus de "petits pas" que de "grands pas". D'un point de vue algorithmique, ceci est très simple à réaliser. Il suffit d'engendrer aléatoirement une séquence binaire de taille $2n$ et de calculer son poids de Hamming. Si ce poids est supérieur à n, il n'y a rien à faire. Sinon ceci signifie que la séquence contient au moins n zéros. Etant donné que décider qu'un petit pas correspond à la valeur 1 et un grand pas à la valeur 0 est un choix totalement arbitraire, dans le cas où le poids de c est inférieur à $2n$, il suffit d'échanger le rôle des "petits pas" et des "grands pas". C'est ce principe qui est appliqué dans l'algorithme 16. L'algorithme initial de multiplication scalaire possède deux défauts. Bien que selon la valeur courante de c_i, on effectue toujours un appel à la fonction ZADDU, et que la même séquence d'instructions est exécutée, sur certaines architectures munies d'un prédicteur de branchement, ce branchement conditionnel dépendant de la valeur de c_i peut être exploité [OKS06]. De plus selon la valeur de c_i, dans l'algorithme 12 page 35, on appelle la fonction ZADDU, soit avec le couple (U_2, U_1), soit avec le couple (U_1, U_2). Cet accès en mémoire à deux données dans un ordre qui dépend d'une donnée secrète peut donner lieu à des attaques de type "cache timing attack" [BH09]. A titre d'exemple, la méthode GLV-SAC est vulnérable à cette catégorie d'attaque. L'algorithme 16 supprime le branchement conditionnel et la dépendance des accès mémoire à la valeur courante c_i, en utilisant une permutation à temps constant qui prépare les arguments d'ap-

pel de ZADDU, ceci en effectuant un certain nombre d'opérations qui accèdent toujours dans le même ordre aux données manipulées (lignes 9 à 12). Cette permutation sécurisée des arguments d'appel de ZADDU est inspirée de [Dül+15].

Le tableau 1.18 donne, pour un niveau de sécurité de ℓ bits, la taille minimale du corps sur lequel les calculs seront effectués, ainsi que la longueur des chaînes intervenants dans la multiplication scalaire. La réduction obtenue sur la taille du corps permet de remettre

Niveau de sécurité	80	96	112	128	192	256	384
Longueur CAE	162	194	126	258	386	514	770
Taille du corps	209	250	290	331	494	657	982

Table 1.18 – Taille minimale du corps et taille des CAE pour un niveau de sécurité donné, lorsque $1 + |r| + s \leqslant 4$.

à jour la remarque sur la sécurité de la multiplication scalaire utilisant la méthode CAE énoncée page 54.

Sécurité/Taille du corps

Dans le contexte d'un point P variable et d'une courbe munie d'un endomorphisme "facilement" calculable, à niveau de sécurité ℓ équivalent, l'utilisation des CAE dont la représentation binaire est choisie dans $\mathcal{M}_{2\ell+2,\geqslant\ell+1}$ a pour conséquence de devoir manipuler des entiers de taille environ 1.28 fois plus importante que les méthodes "classiques".

Remarque importante

Le résultat obtenu sur le lien entre le niveau de sécurité et la taille des entiers manipulés reste valable dans le cas du point P fixe. En effet dans ce cas, on peut précalculer $\Phi(P)$ une fois pour toute (même pour un endomorphisme qui n'est pas forcément "facilement" calculable) et appliquer l'algorithme 12, page 35, en partant de $(P, \Phi(P))$.

1.7.1 Une première conséquence

Dans le contexte d'un point P fixe, l'équation 1.5, page 63 qu'il fallait satisfaire afin que la méthode CAE (X, Y)-only soit plus efficace que la méthode Comb-2 sur une architecture 64 bits, devient

$$\left\lceil \frac{\lfloor 2.56\ell \rfloor}{k} \right\rceil - \left\lceil \frac{2\ell}{k} \right\rceil = \delta \,. \tag{1.5}$$

Ceci permet d'enrichir le tableau 1.9 page 41 qui liste les niveaux de sécurité pour lesquels la méthode CAE (X, Y)-only est moins coûteuse que la méthode Comb-2 sur une architecture 64 bits.

Nb mots de 64 bits	(taille classique, taille CAE)
4	(193, 247), (194, 248), (195, 249), (196, 250), (197, 252), (198, 253), (199, 254), (200, 256), (201, 257), (202, 258), (203, 259), (204, 261), (205, 262), (206, 263), (207, 264), (208, 266), (209, 267), (210, 268), (211, 270), (212, 271), (213, 272), (214, 273), (215, 275), (216, 276), (217, 277), (218, 279), (219, 280), (220, 281), (221, 282), (222, 284), (223, 285), (224, 286), (225, 288), (226, 289), (227, 290), (228, 291), (229, 293), (230, 294), (231, 295), (232, 296), (233, 298), (234, 299), (235, 300), (236, 302), (237, 303), (238, 304), (239, 305), (240, 307), (241, 308), (242, 309), (243, 311), (244, 312), (245, 313), (246, 314), (247, 316), (248, 317), (249, 318), (250, 320)
5	(257, 328), (258, 330), (259, 331), (260, 332), (261, 334), (262, 335), (263, 336), (264, 337), (265, 339), (266, 340), (267, 341), (268, 343), (269, 344), (270, 345), (271, 346), (272, 348), (273, 349), (274, 350), (275, 352), (276, 353), (277, 354), (278, 355), (279, 357), (280, 358), (281, 359), (282, 360), (283, 362), (284, 363), (285, 364), (286, 366), (287, 367), (288, 368), (289, 369), (290, 371), (291, 372), (292, 373), (293, 375), (294, 376), (295, 377), (296, 378), (297, 380), (298, 381), (299, 382), (300, 384)
6	(321, 410), (322, 412), (323, 413), (324, 414), (325, 416), (326, 417), (327, 418), (328, 419), (329, 421), (330, 422), (331, 423), (332, 424), (333, 426), (334, 427), (335, 428), (336, 430), (337, 431), (338, 432), (339, 433), (340, 435), (341, 436), (342, 437), (343, 439), (344, 440), (345, 441), (346, 442), (347, 444), (348, 445), (349, 446), (350, 448)
7	(385, 492), (386, 494), (387, 495), (388, 496), (389, 497), (390, 499), (391, 500), (392, 501), (393, 503), (394, 504), (395, 505), (396, 506), (397, 508), (398, 509), (399, 510), (400, 512)

8	(449, 574), (450, 576), (451, 577), (452, 578), (453, 579), (454, 581), (455, 582), (456, 583), (457, 584), (458, 586), (459, 587), (460, 588), (461, 590), (462, 591), (463, 592), (464, 593), (465, 595), (466, 596), (467, 597), (468, 599), (469, 600), (470, 601), (471, 602), (472, 604), (473, 605), (474, 606), (475, 608), (476, 609), (477, 610), (478, 611), (479, 613), (480, 614), (481, 615), (482, 616), (483, 618), (484, 619), (485, 620), (486, 622), (487, 623), (488, 624), (489, 625), (490, 627), (491, 628), (492, 629), (493, 631), (494, 632), (495, 633), (496, 634), (497, 636), (498, 637), (499, 638), (500, 640)
9	(513, 656), (514, 657), (515, 659), (516, 660), (517, 661), (518, 663), (519, 664), (520, 665), (521, 666), (522, 668), (523, 669), (524, 670), (525, 672), (526, 673), (527, 674), (528, 675), (529, 677), (530, 678), (531, 679), (532, 680), (533, 682), (534, 683), (535, 684), (536, 686), (537, 687), (538, 688), (539, 689), (540, 691), (541, 692), (542, 693), (543, 695), (544, 696), (545, 697), (546, 698), (547, 700), (548, 701), (549, 702), (550, 704)
10	(577, 738), (578, 739), (579, 741), (580, 742), (581, 743), (582, 744), (583, 746), (584, 747), (585, 748), (586, 750), (587, 751), (588, 752), (589, 753), (590, 755), (591, 756), (592, 757), (593, 759), (594, 760), (595, 761), (596, 762), (597, 764), (598, 765), (599, 766), (600, 768)

Table 1.19 – Couples $(t, 1.28t)$ pour lesquels la méthode CAE (X, Y)-only sur des entiers de $1.28t$ bits est plus efficace que la méthode Comb-2 sur des entiers de t bits sur une architecture 64 bits.

On obtient de plus des candidats pour les architectures 8 et 32 bits ce qui n'était pas le cas lorsque nous avions un facteur 1.4 d'expansion sur la taille des objets.

Nb mots de 8 bits	(taille classique, taille CAE)
25	(193, 247), (194, 248), (195, 249), (196, 250), (197, 252), (198, 253), (199, 254), (200, 256)
26	(201, 257), (202, 258), (203, 259), (204, 261), (205, 262), (206, 263), (207, 264)
27	(209, 267), (210, 268), (211, 270), (212, 271), (213, 272)
28	(217, 277), (218, 279), (219, 280)
29	(225, 288), (226, 289), (227, 290), (228, 291), (229, 293), (230, 294), (231, 295), (232, 296)
30	(233, 298), (234, 299), (235, 300), (236, 302), (237, 303), (238, 304)
31	(241, 308), (242, 309), (243, 311), (244, 312)

32	(249, 318), (250, 320), (251, 321), (252, 322), (253, 323), (254, 325), (255, 326), (256, 327)
33	(257, 328), (258, 330), (259, 331), (260, 332), (261, 334), (262, 335), (263, 336)
34	(265, 339), (266, 340), (267, 341), (268, 343), (269, 344)
35	(273, 349), (274, 350), (275, 352)
36	(281, 359), (282, 360), (283, 362), (284, 363), (285, 364), (286, 366), (287, 367), (288, 368)
37	(289, 369), (290, 371), (291, 372), (292, 373), (293, 375), (294, 376)
38	(297, 380), (298, 381), (299, 382), (300, 384)
39	(305, 390), (306, 391), (307, 392), (308, 394), (309, 395), (310, 396), (311, 398), (312, 399)
40	(313, 400), (314, 401), (315, 403), (316, 404), (317, 405), (318, 407), (319, 408)
41	(321, 410), (322, 412), (323, 413), (324, 414), (325, 416)
42	(329, 421), (330, 422), (331, 423), (332, 424)
43	(337, 431), (338, 432), (339, 433), (340, 435), (341, 436), (342, 437), (343, 439), (344, 440)
44	(345, 441), (346, 442), (347, 444), (348, 445), (349, 446), (350, 448)
45	(353, 451), (354, 453), (355, 454), (356, 455), (357, 456)
46	(361, 462), (362, 463), (363, 464), (364, 465), (365, 467), (366, 468), (367, 469), (368, 471)
47	(369, 472), (370, 473), (371, 474), (372, 476), (373, 477), (374, 478), (375, 480)
48	(377, 482), (378, 483), (379, 485), (380, 486), (381, 487), (382, 488)
49	(385, 492), (386, 494), (387, 495), (388, 496)
50	(393, 503), (394, 504), (395, 505), (396, 506), (397, 508), (398, 509), (399, 510), (400, 512)
51	(401, 513), (402, 514), (403, 515), (404, 517), (405, 518), (406, 519), (407, 520)
52	(409, 523), (410, 524), (411, 526), (412, 527), (413, 528)
53	(417, 533), (418, 535), (419, 536), (420, 537), (421, 538), (422, 540), (423, 541), (424, 542)
54	(425, 544), (426, 545), (427, 546), (428, 547), (429, 549), (430, 550), (431, 551), (432, 552)
55	(433, 554), (434, 555), (435, 556), (436, 558), (437, 559), (438, 560)
56	(441, 564), (442, 565), (443, 567), (444, 568)
57	(449, 574), (450, 576), (451, 577), (452, 578), (453, 579), (454, 581), (455, 582), (456, 583)

58	(457, 584), (458, 586), (459, 587), (460, 588), (461, 590), (462, 591), (463, 592)
59	(465, 595), (466, 596), (467, 597), (468, 599), (469, 600)
60	(473, 605), (474, 606), (475, 608), (476, 609), (477, 610), (478, 611), (479, 613), (480, 614)
61	(481, 615), (482, 616), (483, 618), (484, 619), (485, 620), (486, 622), (487, 623), (488, 624)
62	(489, 625), (490, 627), (491, 628), (492, 629), (493, 631), (494, 632)
63	(497, 636), (498, 637), (499, 638), (500, 640)
64	(505, 646), (506, 647), (507, 648), (508, 650), (509, 651), (510, 652), (511, 654), (512, 655)
65	(513, 656), (514, 657), (515, 659), (516, 660), (517, 661), (518, 663), (519, 664)
66	(521, 666), (522, 668), (523, 669), (524, 670), (525, 672)
67	(529, 677), (530, 678), (531, 679), (532, 680), (533, 682), (534, 683), (535, 684), (536, 686)
68	(537, 687), (538, 688), (539, 689), (540, 691), (541, 692), (542, 693), (543, 695), (544, 696)
69	(545, 697), (546, 698), (547, 700), (548, 701), (549, 702), (550, 704)
70	(553, 707), (554, 709), (555, 710), (556, 711), (557, 712)
71	(561, 718), (562, 719), (563, 720), (564, 721), (565, 723), (566, 724), (567, 725), (568, 727)
72	(569, 728), (570, 729), (571, 730), (572, 732), (573, 733), (574, 734), (575, 736)
73	(577, 738), (578, 739), (579, 741), (580, 742), (581, 743), (582, 744)
74	(585, 748), (586, 750), (587, 751), (588, 752), (589, 753), (590, 755), (591, 756), (592, 757)
75	(593, 759), (594, 760), (595, 761), (596, 762), (597, 764), (598, 765), (599, 766), (600, 768)
76	(601, 769), (602, 770), (603, 771), (604, 773), (605, 774), (606, 775), (607, 776)
77	(609, 779), (610, 780), (611, 782), (612, 783), (613, 784)
78	(617, 789), (618, 791), (619, 792), (620, 793), (621, 794), (622, 796), (623, 797), (624, 798)
79	(625, 800), (626, 801), (627, 802), (628, 803), (629, 805), (630, 806), (631, 807), (632, 808)
80	(633, 810), (634, 811), (635, 812), (636, 814), (637, 815), (638, 816)

Table 1.20 – Couples $(t, 1.28t)$ pour lesquels la méthode CAE (X, Y)-only sur des entiers de $1.28t$ bits est plus efficace que la méthode Comb-2 sur des entiers de t bits sur une architecture 8 bits.

Nb mots de 32 bits	(taille classique, taille CAE)
7	(193, 247), (194, 248), (195, 249), (196, 250), (197, 252), (198, 253), (199, 254), (200, 256)
8	(225, 288), (226, 289), (227, 290), (228, 291), (229, 293), (230, 294), (231, 295), (232, 296), (233, 298), (234, 299), (235, 300), (236, 302), (237, 303), (238, 304), (239, 305), (240, 307), (241, 308), (242, 309), (243, 311), (244, 312), (245, 313), (246, 314), (247, 316), (248, 317), (249, 318), (250, 320)
9	(257, 328), (258, 330), (259, 331), (260, 332), (261, 334), (262, 335), (263, 336), (264, 337), (265, 339), (266, 340), (267, 341), (268, 343), (269, 344), (270, 345), (271, 346), (272, 348), (273, 349), (274, 350), (275, 352)
10	(289, 369), (290, 371), (291, 372), (292, 373), (293, 375), (294, 376), (295, 377), (296, 378), (297, 380), (298, 381), (299, 382), (300, 384)
11	(321, 410), (322, 412), (323, 413), (324, 414), (325, 416), (326, 417), (327, 418), (328, 419), (329, 421), (330, 422), (331, 423), (332, 424), (333, 426), (334, 427), (335, 428), (336, 430), (337, 431), (338, 432), (339, 433), (340, 435), (341, 436), (342, 437), (343, 439), (344, 440), (345, 441), (346, 442), (347, 444), (348, 445), (349, 446), (350, 448)
12	(353, 451), (354, 453), (355, 454), (356, 455), (357, 456), (358, 458), (359, 459), (360, 460), (361, 462), (362, 463), (363, 464), (364, 465), (365, 467), (366, 468), (367, 469), (368, 471), (369, 472), (370, 473), (371, 474), (372, 476), (373, 477), (374, 478), (375, 480)
13	(385, 492), (386, 494), (387, 495), (388, 496), (389, 497), (390, 499), (391, 500), (392, 501), (393, 503), (394, 504), (395, 505), (396, 506), (397, 508), (398, 509), (399, 510), (400, 512)
14	(417, 533), (418, 535), (419, 536), (420, 537), (421, 538), (422, 540), (423, 541), (424, 542), (425, 544)
15	(449, 574), (450, 576), (451, 577), (452, 578), (453, 579), (454, 581), (455, 582), (456, 583), (457, 584), (458, 586), (459, 587), (460, 588), (461, 590), (462, 591), (463, 592), (464, 593), (465, 595), (466, 596), (467, 597), (468, 599), (469, 600), (470, 601), (471, 602), (472, 604), (473, 605), (474, 606), (475, 608)
16	(481, 615), (482, 616), (483, 618), (484, 619), (485, 620), (486, 622), (487, 623), (488, 624), (489, 625), (490, 627), (491, 628), (492, 629), (493, 631), (494, 632), (495, 633), (496, 634), (497, 636), (498, 637), (499, 638), (500, 640)

17	(513, 656), (514, 657), (515, 659), (516, 660), (517, 661), (518, 663), (519, 664), (520, 665), (521, 666), (522, 668), (523, 669), (524, 670), (525, 672)
18	(545, 697), (546, 698), (547, 700), (548, 701), (549, 702), (550, 704), (551, 705), (552, 706), (553, 707), (554, 709), (555, 710), (556, 711), (557, 712), (558, 714), (559, 715), (560, 716), (561, 718), (562, 719), (563, 720), (564, 721), (565, 723), (566, 724), (567, 725), (568, 727), (569, 728), (570, 729), (571, 730), (572, 732), (573, 733), (574, 734), (575, 736)
19	(577, 738), (578, 739), (579, 741), (580, 742), (581, 743), (582, 744), (583, 746), (584, 747), (585, 748), (586, 750), (587, 751), (588, 752), (589, 753), (590, 755), (591, 756), (592, 757), (593, 759), (594, 760), (595, 761), (596, 762), (597, 764), (598, 765), (599, 766), (600, 768)
20	(609, 779), (610, 780), (611, 782), (612, 783), (613, 784), (614, 785), (615, 787), (616, 788), (617, 789), (618, 791), (619, 792), (620, 793), (621, 794), (622, 796), (623, 797), (624, 798), (625, 800)

Table 1.21 – Couples $(t, 1.28t)$ pour lesquels la méthode CAE (X, Y)-only sur des entiers de $1.28t$ bits est plus efficace que la méthode Comb-2 sur des entiers de t bits sur une architecture 32 bits.

1.7.2 Une deuxième conséquence

D'un point de vue thérorique le coût d'une multiplication modulaire entre deux entiers de 331 bits étant inférieur à celui de 2 entiers de 358 bits, le surcoût par rapport au produit de deux entiers de 256 bits devrait être inférieur à ce que nous avons pu constater dans [Dos+18] et ceci devrait donc permettre d'obtenir un implémentation en C (et non pas seulement sous Android) de la méthode CAE (X, Y)-only (et éventuellement de la méthode CAE) plus efficace que TED-GLV-SAC. En pratique, il s'avère que du fait de la gestion des grands entiers effectuée par les librairies multi-précisions classiques, le coût de la multiplication modulaire entre 2 entiers de 331 bits est quasiment identique au coût de la multiplication modulaire entre 2 entiers de 358 bits (cf. table 1.22). Dans le système de représentation des entiers AMNS (cf. chapitre suivant), quelque soit les tailles considérées, non seulement les performances sont souvent meilleures que ce que l'on obtient avec les librairies classiques mais de plus le coût de la multiplication modulaire entre 2 entiers de 331 bits est inférieur au coût de la multiplication modulaire entre 2 entiers de 358 bits (cf. table 1.23). Le ratio inférieur à 1.3 que l'on obtient avec le coût d'une multiplication entre deux entiers de 256 bits laisse entrevoir la possibilité d'obtenir une version C de la

	358 bits	331 bits
`gnu_mp_mont`	268	268
`openssl_mont`	344	345
`gnu_mp_low`	513	521
`gnu_mp`	602	608
`openssl`	1682	1706

Table 1.22 – Nombre de cycles pour la multiplication d'entiers de 331 et 358 bits, cas $S = M$, gcc 9.3.0.

	256 bits	331 bits	358 bits	Ratio $\times_{331}/\times_{256}$
AMNS	184	239	296	1.298

Table 1.23 – Nombre de cycles pour la multiplication d'entiers de 256, 331 et 358 bits pour le système de représentation AMNS, cas $S = M$, gcc 9.3.0.

méthode CAE plus efficace que TED-GLV-SAC. Ceci fera l'objet d'un futur travail.

1.7.3 Une troisième conséquence

cf. paragraphe suivant

1.8 Occupation mémoire

Que ce soit dans le cadre d'un point P fixe ou d'un point P variable, la méthode CAE ne nécessite de pouvoir stocker uniquement que 2 points en coordonnées jacobiennes pour obtenir le point final résultant de la multiplication scalaire. Etant donné que ces 2 points ont une coordonnée Z commune, il suffit donc de stocker 5 coordonnées pour démarrer l'algorithme (et uniquement 4 dans le cas de la méthode CAE (X, Y)-only). Il est important de remarquer que dans l'algorithme de multiplication scalaire, quelque soit le couple de points de départ, ce dernier n'est utilisé que lors de la première itération. Ainsi, dans le cas où le point P est variable, le couple $(P, \phi(P))$ peut être écrasé par le couple (U_1, U_2) calculé dans l'algorithme 12 page 35. De même, chaque nouveau couple (U_1, U_2) calculé peut remplacer en mémoire le précédent. Une analyse de l'algorithme d'addition ZADDU (Algorithme 8, page 26) permet de conclure qu'il suffit de deux variables auxiliaires supplémentaires (sur le corps de base) pour procéder à la multiplication scalaire. Dans [HMV21, Algorithme 2], nous avons montré que l'on pouvait en fait utiliser une seule variable auxiliaire pour effectuer l'addition ZADDU. Ainsi, il suffit de 6 registres sur le corps de base (5 pour la méthode (X, Y)-only) pour effectuer une multiplication scalaire complète. Remarquons que la majorité des algorithmes de multiplication scalaire nécessite au moins ce nombre de

		Données		ECSM		
	$log_2(p)$	$\mathbb{F}_p$-registres	Taille(octets)	$\mathbb{F}_p$-registres	Taille(octets)	Total
x-line [CHS14]	128	16	256	28	448	704
Ted-glv-sac	256	6	192	6	192	384
Ted127-glv4 [FLS15]	127	32	508	48	762	1270
FourQ[CL15]	127	4	64	98	1556	1620
Montgomery[Ber06]	256	1	32	6	192	224
CAE [Dos+18]	358	5	224	2	90	314
CAE [HMV21]	**331**	**5**	**207**	**1**	**42**	**249**
CAE (X, Y)-only [Dos+18]	358	4	179	2	90	269
CAE $(\mathbf{X}, \mathbf{Y})$-only [HMV21]	**331**	**4**	**166**	**1**	**42**	**207**

Table 1.24 – Occupation mémoire pour un niveau de sécurité de 128 bits

registres pour stocker au minimum le point de départ P et un point intermédiaire Q. Selon les variantes, d'autres points sont stockés et plusieurs variables auxiliaires sont nécessaires afin d'effectuer le processus de multiplication scalaire. Evidemment, il ne faut pas perdre de vue, qu'à niveau de sécurité ℓ équivalent, la taille des données stockées pour la méthode CAE est plus importante (2.8ℓ ou 2.56ℓ au vu des résultats du paragraphe précédent) que dans les méthodes classiques. Dans [HMV21, Paragraphe 4], nous avons effectué une analyse détaillée du coût mémoire des algorithmes de multiplication scalaire les plus efficaces à l'heure actuelle. Il en ressort que la méthode CAE (X, Y)-only est celle qui requiert le moins d'espace mémoire pour effectuer une multiplication scalaire et ce malgré l'utilisation de données de taille plus importante. La méthode CAE s'avère être elle aussi la méthode de calcul la plus économique en terme de consommation mémoire, exception faite de la méthode de Montgomery qui nécessite 25 octets en moins. La table 1.24 résume les résultats obtenus pour les différentes méthodes étudiées. La colonne "données" contient la taille en octets nécessaire pour stocker le ou les points indispensables à l'initialisation de l'algorithme de multiplication scalaire. La colonne ECSM (Elliptic Curve Scalar Multiplication) contient la taille en octets nécessaire pour stocker les points intermédiaires et/ou les variables intermédiaires utilisées lors du déroulement de l'algorithme de multiplication scalaire.

1.9 Résistance aux injections de faute

Les attaques de type SPA ou DPA font partie de la classe des attaques dites "passives". L'attaquant observe le comportement d'un algorithme, mesure certains évènements et en déduit de l'information sur la donnée secrète manipulée. Un autre type d'attaque, dite "active" consiste à perturber le déroulement de l'algorithme, quitte à ce que ce dernier renvoie un faux résultat, afin d'essayer d'en déduire de l'information sur la donnée secrète. C'est ce que l'on appelle les attaques par injection de faute [BDL97]. En fonction du type

d'attaque (perturbation d'un calcul, perturbation d'un registre mémoire,...), il existe dans la littérature de nombreuses contre-mesures, permettant de renforcer la sécurité d'un algorithme. Dans [Pon+09], les auteurs considèrent un modèle d'attaquant capable de perturber une opération arithmétique effectuée lors d'un calcul de point sur une courbe elliptique. Ils proposent une contre-mesure reposant sur l'existence d'un invariant dans l'algorithme 7 page 26. Nous en donnons ici une version modifiée par rapport à ce qui est décrit dans [DV17], qui nous semble plus naturelle à comprendre.

Au début de l'étape i de l'algorithme 7 page 26, on dispose d'un couple de points (U_1, U_2), provenant de l'étape précédente et que l'on notera $(U_1^{(i-1)}, U_2^{(i-1)})$ pour la suite. Deux cas sont alors à considérér :

1. $c_i = 1$ (petit pas) : posons $U_C = U_2^{(i-1)}$, on a alors à la fin de l'étape $(U_1^{(i)}, U_2^{(i)}) = (U_1^{(i-1)}, U_1^{(i-1)} + U_2^{(i-1)})$ et remarquons que $U_2^{(i)} - U_1^{(i)} = U_C$,
2. $c_i = 0$ (grand pas) : posons $U_C = U_1^{(i-1)}$, on a alors à la fin de l'étape $(U_1^{(i)}, U_2^{(i)}) = (U_2^{(i-1)}, U_1^{(i-1)} + U_2^{(i-1)})$. Remarquons que $U_2^{(i)} - U_1^{(i)} = U_C$.

Ainsi à la fin de l'étape i, afin de vérifier qu'il n'y a pas eu de fautes commises lors des calculs sur $\mathbb{F}_p$, il suffit de vérifier que $U_2^{(i)} - U_1^{(i)} = U_C$ (cf. Algorithme 17 qui est une modification de [DV17, Algorithme 1]). Dans le cas où le point P est variable, il faudra vérifier cette propriété aussi lors du calcul du point $2P$ nécessaire à l'initialisation de l'algorithme. Dans le cas d'un point P fixe, nous supposons que $2P$ est stocké en mémoire. Dans le modèle considéré, l'attaquant ne peut pas modifier cette valeur. La méthode de Pontarelli et al. possède de nombreux inconvénients :

1. elle utilise le système de coordonnées affines, ainsi chaque addition nécessite une inversion I coûteuse sur $\mathbb{F}_p$. Les auteurs montrent dans [Pon+09] que certains des calculs effectués pour obtenir $U_1 + U_2$ (coût I+2M+S+6A) peuvent être réutilisés pour obtenir $U_1 - U_2$, le coût total de ces deux opérations revient alors à I+4M+2S+10A,
2. elle ne tire pas partie de l'utilisation d'une CAE qui combinée avec ZADDU permet d'effectuer l'addition pour un coût de 5M+2S.
3. telle qu'elle est décrite, la méthode est vulnérable aux attaques de type "cache timing" (cf. page 62),
4. finalement, elle ne résiste pas au modèle d'attaquant décrit dans [AKS12].

Revenons sur ce dernier point. Dans [AKS12], les auteurs proposent un modèle d'attaquant capable d'appliquer plusieurs fois une même faute E lors des différents calculs effectués au cours de l'algorithme. Supposons que cet attaquant soit capable lors d'une opération entre 2 points d'ajouter un point E de son choix. Il peut donc choisir un point E et perturber la ligne 12 de l'algorithme. Notons $\tilde{U}_2$ le point ainsi obtenu. On a donc $\tilde{U}_2 = U_2 + E$.

Algorithm 17 Méthod de Pontarelli et al. pour la détection de fautes.

Require: P (et $2P$ dans le cas P fixe)
Ensure: $Q = \chi(c)P$
1: $Q \leftarrow 2P$ /* uniquement dans le cas P variable */
2: $U_D \leftarrow Q - P$ /* uniquement dans le cas P variable */
3: **if** $U_D \neq P$ **then** /* uniquement dans le cas P variable */
4: **return** $(Q = 0, \text{error} = 1)$;
5: **end if**
6: **for** $i = 1 \ldots \text{length}(c)$ **do**
7: **if** $c_i = 1$ **then**
8: $U_C \leftarrow U_2$, $U_1 \leftarrow U_1$
9: **else**
10: $U_C \leftarrow U_1$, $U_1 \leftarrow U_2$
11: **end if**
12: $U_2 \leftarrow U_1 + U_C$
13: $U_D \leftarrow U_2 - U_1$
14: **if** $U_D \neq U_C$ **then**
15: **return** $(Q = 0, \text{error} = 1)$;
16: **end if**
17: **end for**
18: $U_C \leftarrow U_2$, $U_2 \leftarrow U_1 + U_2$
19: $Q \leftarrow U_2$
20: $U_D \leftarrow U_2 - U_1$
21: **if** $U_D \neq U_C$ **then**
22: **return** $(Q = 0, \text{error} = 1)$;
23: **end if**
24: **return** $(Q, \text{error} = 0)$

Supposons alors qu'il fasse de même sur la ligne suivante pour ajouter le point $-E$ au résultat, on a alors

$$\tilde{U}_D = \tilde{U}_2 - U_1 - E = U_2 - U_1 \,.$$

et l'erreur n'est pas détectée.

Dans [DV17], nous montrons comment améliorer la méthode de Pontarelli et al. en utilisant le système de coordonnées jacobiennes et l'addition ZADDU. Le premier problème qui se pose est le calcul efficace de $U_2 - U_1$. Dans [Gou+11], les auteurs proposent une formule d'addition, nommé ZADDC (addition Co-Z conjuguée, cf. [DV17, Algorithme 3]) qui, à partir d'un couple de points (P, Q), calcule le couple (R, S) tel que :

$$R = \overline{P+Q}, S = \overline{P-Q}, Z_R = Z_S \,.$$

La notation $\overline{P+Q}$ (resp. $\overline{P-Q}$) désigne un point dans la même classe d'équivalence que $P+Q$ (resp. $P-Q$). Le coût total de cette formule est de 6M + 3S, soit une multiplication et une élévation au carré de plus que ZADDU.

Remarque importante

Dans les calculs intermédiaires effectués dans ZADDC, les variables intermédiaires W_1 et A_1 [DV17, lignes 2 et 4, Algorithme 3] contiennent les coordonnées (X, Y) du point $\overline{P}$. La variable intermédiaire W_2 [DV17, lignes 3, Algorithme 3] contient la coordonnée X de $\overline{Q}$. Finalement la variable Z_3 [DV17, lignes 9, Algorithme 3] contient la coordonnée Z commune à $\overline{P}$, $\overline{Q}$, $\overline{P+Q}$ et $\overline{P-Q}$. Ceci nous sera utile pour la suite.

Pour la suite, considérons le cas où l'on n'effectue uniquement des petits pas. Au début de l'étape i, on dispose du couple $(U_1^{(i-1)}, U_2^{(i-1)})$. Si on utilise la procédure ZADDC sur ce couple, on obtient les points $\overline{U_1^{(i-1)} + U_2^{(i-1)}}$ et $\overline{U_2^{(i-1)} - U_1^{(i-1)}}$ pour un coût de 6M+3S. Si on dispose d'un point U_C correspondant à $U_2^{(i-2)}$, on peut donc détecter s'il y a eu une erreur à l'étape $i-1$, en vérifiant si $\overline{U_C} \neq \overline{U_2^{(i-1)} - U_1^{(i-1)}}$ (la vérification doit être effectuée dans la même classe d'équivalence que $U_2^{(i-1)} - U_1^{(i-1)}$).

Dans [DV17, Algorithme 5], nous avons défini une procédure mZADDC qui à partir d'un triplet (U_C, U_1, U_2) effectue à l'étape i les opérations suivantes :

1. $U_D \leftarrow \overline{U_2 - U_1}$,
2. Vérifier si $U_D \neq \overline{U_C}$,
3. $U_C \leftarrow \overline{U_2}$,
4. $U_1 \leftarrow \overline{U_1}$,

5. $U_2 \leftarrow \overline{U_1 + U_2}$.

Les étapes 1, 3 et 5 sont effectuées en utilisant les formules de ZADDC appliquées à (U_2, U_1) et en tenant compte de la remarque précédente. L'algorithme 5 proposé dans [DV17] comporte à ce niveau une légère erreur (lignes 14 et 21) car il utilise les formules de ZADDC appliquées à (U_1, U_2), l'algorithme 18 corrige ceci. Pour l'étape 4, une multiplication sup-

Algorithm 18 mZADDC (C, P, Q)

Require: $C(X_C, Y_C, Z_C), P(X_P, Y_P, Z)$ et $Q(X_Q, Y_Q, Z)$.
Ensure: $(C, P, Q) \leftarrow (\overline{Q}, \overline{P}, \overline{P+Q})$, renvoie 0 ssi $\overline{Q-P} = \overline{C}$.

1: $\mathbf{Z} \leftarrow \mathbf{Z}.(\mathbf{X_Q} - \mathbf{X_P})$ #[calcule $Z_{\overline{Q-P}} = Z_{\overline{P+Q}} = Z_{\overline{P}} = Z_{\overline{Q}}$]
2: $Z_C \leftarrow Z_C.(X_Q - X_P)$ #[calcule $Z_{\overline{C}}$]
3: $A \leftarrow (X_Q - X_P)^2$
4: $\mathbf{X_P} \leftarrow \mathbf{X_P}.\mathbf{A}$ #[calcule $X_{\overline{P}}$]
5: $X_Q \leftarrow X_Q.A$ #[calcule $X_{\overline{Q}}$]
6: $X_C \leftarrow X_C.A$ #[calcule $X_{\overline{C}}$]
7: $B \leftarrow Y_Q - Y_P$
8: $E \leftarrow Y_Q + Y_P$
9: $F \leftarrow X_Q - X_P$
10: $\mathbf{Y_P} \leftarrow \mathbf{Y_P}.\mathbf{F}$ #[calcule $Y_{\overline{P}}$]
11: $Y_Q \leftarrow Y_Q.F$ #[calcule $Y_{\overline{Q}}$]
12: $Y_C \leftarrow Y_C.F$ #[calcule $Y_{\overline{C}}$]
13: $Dx \leftarrow E^2 - X_Q - X_P$ #[calcule $X_{\overline{Q-P}}$]
14: $Dy \leftarrow E.(X_Q - Dx) - Y_Q$ #[calcule $Y_{\overline{Q-P}}$]
15: # {vérifie si $\overline{Q-P} = \overline{C}$}
16: $check \leftarrow (Dx \oplus X_C) \vee (Dy \oplus Y_C) \vee (Z \oplus Z_C)$
17: $\mathbf{X_C} \leftarrow \mathbf{X_Q}$
18: $\mathbf{Y_C} \leftarrow \mathbf{Y_Q}$
19: $\mathbf{Z_C} \leftarrow \mathbf{Z}$
20: $\mathbf{X_Q} \leftarrow \mathbf{B^2} - \mathbf{X_Q} - \mathbf{X_P}$ #[calcule $X_{\overline{P+Q}}$]
21: $\mathbf{Y_Q} \leftarrow \mathbf{B}.(\mathbf{X_C} - \mathbf{X_Q}) - \mathbf{Y_Q}$ #[calcule $Y_{\overline{P+Q}}$]
22: **return** *check*

plémentaire est nécessaire pour obtenir la coordonnée Y dans la classe d'équivalence du point U_1. Finalement le point 2 nécessite de calculer le représentant de U_C, ce qui ajoute 3 multiplications supplémentaires. On obtient ainsi un coût total de 10M+3S pour la procédure mZADDC.

On dispose donc d'une méthode qui pour chaque petit pas effectué remplace un triplet (U_C, U_1, U_2) par le triplet $(\overline{U}_2, \overline{U}_1, \overline{U_1 + U_2})$. En considérant qu'un grand pas appliqué à un couple (U_1, U_2) revient à effectuer un petit pas sur le couple (U_2, U_1), ceci signifie qu'a la fin d'une étape i on devrait avoir dans le triplet (U_C, U_1, U_2) courant le triplet $(\overline{U}_1, \overline{U}_2, \overline{U_1 + U_2})$ au lieu de $(\overline{U}_2, \overline{U}_1, \overline{U_1 + U_2})$. En d'autres termes, une fois la procédure mZADDC effectuée si le pas courant était un grand pas, il faut échanger les variables U_C

et U_1 courantes. Afin que cet échange ne dépende pas de la nature du pas courant, nous utilisons une procédure classique de permutation de deux valeurs à temps constant (inspirée de [Dül+15]) qui effectue exactement le même nombre d'opérations et accède toujours dans le même ordre à ses données d'entrées pour effectuer ou non la permutation [DV17, Algorithme 6]. Au final la combinaison de la procédure mZADDC avec cette permutation sécurisée permet de définir un algorithme de multiplication scalaire (sans branchement conditionnel dépendant du pas courant) qui permet de détecter une erreur commise lors des opérations arithmétiques (Algorithme 19).

Algorithm 19 Algorithme de multiplication scalaire avec une CAE et détection de fautes

Require: P et $2P$ avec $Z_P = Z_{2P}$
Ensure: $Q = \chi(c)P$
1: $(U_C, U_1, U_2) \leftarrow (P, P, 2P)$
2: $check \leftarrow 1$
3: **for** $i = 1 \ldots \text{length}(c)$ **do**
4: $\quad check \leftarrow mZADDC(U_C, U_1, U_2)$
5: $\quad$ **if** $check \neq 0$ **then**
6: $\quad\quad$ **return** $(Q = 0, \text{error} = 1)$;
7: $\quad$ **end if**
8: $\quad \text{SafePerm}(U_1, U_C, c_i)$
9: **end for**
10: $check \leftarrow mZADDC(U_C, U_1, U_2)$
11: **if** $check \neq 0$ **then**
12: $\quad$ **return** $(Q = 0, \text{error} = 1)$;
13: **end if**
14: $Q \leftarrow U_2$
15: # vérification des derniers calculs effectués ligne 10.
16: $check \leftarrow mZADDC(U_C, U_1, U_2)$
17: **if** $check \neq 0$ **then**
18: $\quad$ **return** $(Q = 0, \text{error} = 1)$
19: **end if**
20: **return** $(Q, \text{error} = 0)$

La table 1.25 résume le coût de la méthode proposée comparativement à celle de Pontarelli et al. Si le coût d'une inversion est plus important que celui de six multiplications et un carré, nous obtenons une multiplication scalaire plus efficace dans le contexte des détections de faute. En utilisant les benchmarks disponibles dans la librairie cryptographique MIRACL [1], nous avons pu estimer le coût d'une inversion par rapport à celui d'une multiplication et la table 1.26 donne une estimation des gains de notre méthode comparativement à celle de Pontarelli et al (dans le contexte S=M).

Concernant l'attaque mentionnée 72, rappelons qu'à partir d'un même point E, l'atta-

1. https://github.com/miracl/MIRACL

Table 1.25 – Nombre d'opérations par bit des méthodes [Pon+09] et [DV17].

Méthode de Pontarelli et al. [Pon+09]	Notre Méthode [DV17]
1I+4M+2S	10M+3S

Table 1.26 – Estimation des performances de notre méthode en utilisant la librairie MIRACL.

Taille du corps (bits)	Rapport I/M	Nombre estimé de mult. dans la méthode de Pontarelli et al.	Gain (%mult) notre méthode /Pontarelli et al.
192	25	31M	42%
256	26	32M	41%
384	23	29M	45%
512	19	25M	52%

quant perturbe le calcul de $U_1 + U_2$ en ajoutant le point E, puis perturbe le calcul de $U_2 - U_1$ en ajoutant le point $-E$ si bien que l'égalité $U_D = U_C$ est vérifiée. Supposons que ce même attaquant perturbe à l'étape $i-1$ le point 5 de notre méthode, calculant ainsi $\overline{U_1 + U_2} + E$. Supposons que l'on soit dans le cas d'un petit pas , le triplet en entrée de l'étape i vaut donc $(\overline{U}_2, \overline{U}_1, \overline{U_1 + U_2} + E)$. L'attaquant doit alors être en mesure de rajouter une quantité E' telle que :

$$\overline{\overline{U_1 + U_2} + E - \overline{U_1}} + E' = \overline{\overline{U_2}}\,.$$

A ce stade, nous sortons du contexte du modèle de l'attaquant capable d'ajouter et de retirer une même valeur E en différents points de l'algorithme où des calculs sont effectués. En effet, il apparaît ici que le point E' permettant de réaliser l'égalité ci-dessus dépend du coefficient multiplicateur utilisé pour travailler dans une classe d'équivalence. Or ce coefficient dans le contexte de ZADDC vaut $X_{U_2} - X_{U_1}$, il change donc à chaque itération. Il faudrait donc que l'attaquant soit en mesure à chaque itération de l'algorithme d'observer les valeurs des points U_1 et U_2 pour construire le point E'.

1.10 CAE Diffie-Hellman

Dans le paragraphe 1.5.1, page 31, nous avons présenté une version de Diffie-Hellman adaptée à la méthode CAE. Remarquons que l'algorithme proposé (Algorithme 11) peut s'appliquer en prenant pour point de départ $(P, 2P)$ (proposition 1.14, page 35) ou $(P, \lambda P)$ (proposition 1.20, page 51). Dans l'algorithme 11, la clé commune ainsi calculée , lignes 5 et 6, vaut

$$K = \chi_{1,\alpha}(c^{(1)})\chi_{1,\alpha}(c^{(2)})P\,.$$

avec $\alpha = 2$ ou $\alpha = \lambda$ en fonction de la version choisie. A ce stade, il est important de se rappeler que la formule d'addition ZADDU renvoie en fonction du couple (P, Q) donné en entrée, non pas le couple $(P, P+Q)$, mais un couple (P', Q') tel que P et P' sont dans la même classe d'équivalence et Q' et $P+Q$ sont dans la même classe d'équivalence. Par conséquence :

Une clé commune à équivalence près !

Dans la ligne 5 de l'algorithme 11, page 31 :

- Alice obtient un point $K^{(a)}$ dans la même classe d'équivalence que $K = \chi_{1,\alpha}(c^{(1)})\chi_{1,\alpha}(c^{(2)})P$,
- Bob obtient un point $K^{(b)}$ dans la même classe d'équivalence que $K = \chi_{1,\alpha}(c^{(2)})\chi_{1,\alpha}(c^{(1)})P$,

et il n'y a aucune raison pour que $K^{(a)}$ et $K^{(b)}$ soient un même représentant de K.

1.10.1 Cas de la méthode CAE standard

Dans le cas où l'on a utilisé la procédure d'addition ZADDU avec les 3 coordonnées (X, Y, Z), on dispose donc de 2 points (X_A, Y_A, Z_A) et (X_B, Y_B, Z_B) qui sont dans la même classe d'équivalence que le point K. Ainsi la clé partagée peut être obtenue à partir du point affine correspondant $(X_A/Z_A^2, Y_A/Z_A^3) = (X_B/Z_B^2, Y_B/Z_B^3)$.

1.10.2 Cas de la méthode CAE $(X, Y)-$only

En utilisant la méthode CAE (X, Y)-only pour réaliser les étapes de la méthode de Diffie-Hellman, Alice dispose à la fin du protocole d'un couple (X_A, Y_A) et Bob d'un couple (X_B, Y_B) qui sont tous 2 les coordonnées (X, Y) d'un point équivalent au point $K = (X_K, Y_K, Z_K)$. Dans sa thèse, Nicolas Méloni explique comment il est possible de retrouver la valeur de Z^2 à partir des coordonnées (X, Y) calculées par la méthode ZADDU dans sa version (X, Y)-only [Mél07a, page 55]. Ceci permet d'obtenir la valeur commune

$\tilde{K} = X_A/Z_A^2 = X_B/Z_B^2$. La formule proposée n'est valide que dans le cas où les coefficients a et b de l'équation $y^2 = x^3 + ax + b$ sont non nuls. Or, dans le contexte d'une courbe munie d'un endomorphisme facilement calculable, nous travaillons sur des courbes avec $a = 0$ ou $b = 0$. Il est cependant possible dans ce contexte de calculer une valeur commune entre Alice et Bob.

Rappelons qu'en coordonnées jacobiennes, l'équation de la courbe est $Y^2 = X^3 + bZ^6$ ($a = 0$) ou $Y^2 = X^3 + aXZ^4$ ($b = 0$). Considérons les deux points (X_A, Y_A, Z_A) et (X_B, Y_B, Z_B) dans la même classe d'équivalence que le point K, il existe donc un entier t tel que $X_B = t^2 X_A$, $Y_B = t^3 Y_A$ et $Z_B = tZ_A$ et ces deux points correspondent au même point (x, y) en coordonnées affines avec

$$x = \frac{X_A}{Z_A^2} = \frac{X_B}{Z_B^2} \quad \text{et} \quad y = \frac{Y_A}{Z_A^3} = \frac{Y_B}{Z_B^3}\,.$$

Dans le cas où $a = 0$, on a

$$\frac{X_A^3}{Y_A^2 - X_A^3} = \frac{X_A^3}{bZ_A^6} = \frac{1}{b}\left(\frac{X_A}{Z_A^2}\right)^3 = \frac{1}{b}x^3 = \frac{X_B^3}{Y_B^2 - X_B^3}\,.$$

Dans le cas où $b = 0$, on a

$$X_A^3 \frac{Y_A^4}{(Y_A^2 - X_A^3)^3} = \frac{Y_A^4}{a^3 Z_A^{12}} = \frac{1}{a^3}\left(\frac{Y_A}{Z_A^3}\right)^4 = \frac{1}{a^3}y^4 = X_B^3 \frac{Y_B^4}{(Y_B^2 - X_B^3)^3}\,.$$

Sur les courbes de Montgomery, afin d'optimiser les calculs, il est possible de ne calculer que les coordonnées (X, Z) d'un point kP, et la clé partagée est la coordonnée $x = \frac{X}{Z}$ du point commun calculé. Etant donné que lorsqu'un point (x, y) est sur la courbe , le point $(x, -y)$ satisfait aussi l'équation de la courbe, on divise donc l'espace des clés possibles par 2.
Dans notre cas, à un facteur multiplicatif près, on renvoie soit x^3, soit y^4. Le premier cas correspond aux entiers p tels que $p \equiv 1 \pmod 3$ et le deuxième aux entiers p tels que $p \equiv 1 \pmod 4$ (cf. page 50). A ce stade, un résultat dû à Gauss va nous être utile.

Résidus modulo p

Soit p un nombre premier, $r \geqslant 2$ un entier et $b = \text{pgcd}(p-1, r)$:

1. Pour tout $\alpha \in (\mathbb{Z}/p\mathbb{Z})^\star$, α est un $r^{\text{ième}}$ résidu modulo p si et seulement si $\alpha^{\frac{p-1}{b}} \equiv 1 \bmod p$,
2. il y a exactement $\frac{p-1}{b}$ $r^{\text{ième}}$ résidus dans $(\mathbb{Z}/p\mathbb{Z})^\star$,
3. si $\alpha \in (\mathbb{Z}/p\mathbb{Z})^\star$ est un $r^{\text{ième}}$ résidu modulo p, alors il existe exactement b entiers $\ell \in \mathbb{Z}/p\mathbb{Z}$ tels que $\ell^r \equiv \alpha \bmod p$.

Dans le cas $p \equiv 1 \pmod 3$, on a $b = 3$ et dans le cas $p \equiv 1 \pmod 4$, l'entier b vaut 4. Chaque valeur x^3 possède donc exactement 3 antécédents et chaque valeur y^4 exactement 4. On divise donc dans le pire des cas l'espace des clés par 6 ou 8.

Chapitre 2

Le système de représentation PMNS

2.1 Introduction

Dans le chapitre précédent, nous avons vu comment l'utilisation des chaînes d'additions euclidiennes permet dans certains contextes d'optimiser le calcul d'un point kP sur une courbe elliptique. De façon plus générale, comme cela a déjà été mentionné, le calcul efficace de kP dans $E(\mathbb{Z}/p\mathbb{Z})$ ou de x^k dans $\mathbb{Z}/n\mathbb{Z}$ (n premier ou non) est directement lié à la recherche d'une chaîne d'additions courte calculant l'entier k. Une fois ceci effectuée, une autre voie possible pour optimiser le calcul de kP ou de x^k est de s'intéresser aux opérations arithmétiques sous-jacentes : les additions et multiplications modulaires. Ces deux opérations "élémentaires" sont au cœur de la plupart des protocoles à clé publique actuels . Il en est de même pour plusieurs des candidats finalistes de l'appel à standardisation pour les protocoles cryptographiques post-quantiques, lancé en 2017 par le NIST [1]. Les performances de ces différents protocoles (actuels et futurs) dépendent directement de l'efficacité avec laquelle sont effectuées ces opérations modulaires. Notons de plus que toute optimisation dans le domaine de l'arithmétique modulaire a aussi un impact dans le domaine de la cryptanalyse (factorisation, calcul de logarithme discret). Finalement, dans un contexte plus général, de nombreux algorithmes utilisés en théorie des nombres effectuent des calculs modulaires et peuvent donc bénéficier de toute avancée améliorant l'efficacité de l'arithmétique modulaire.

De nombreux travaux ont été développés pour optimiser les opérations modulaires, on pourra par exemple consulter [BZ10] pour une vue d'ensemble sur le sujet. Une approche générale consiste à calculer l'opération dans $\mathbb{Z}$ puis à effectuer la réduction modulaire. Diverses études ont été conduites afin d'optimiser cette étape de réduction.

Parmi les deux opérations élémentaires que sont l'addition et la multiplication dans

1. https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions

$\mathbb{Z}$, c'est cette dernière qui est la plus coûteuse. La littérature regorge de références sur les différents algorithmes permettant de multiplier des grands entiers efficacement : Karatsuba, Toom-Cook, Schönhage-Strassen et autres variantes. Une fois la multiplication effectuée, utiliser un entier N ayant une forme particulière afin d'effectuer astucieusement la réduction modulo N sans division fait partie des méthodes classiques permettant d'optimiser le calcul global de la multiplication modulaire. L'entier N est alors généralement de la forme $2^m - c$, avec c un entier impair dont la taille est inférieure à un mot machine (pseudo-Mersenne), ou de la forme $f(2^m)$ (Mersenne généralisé) avec f un polynôme de faible degré possédant de "petits" coefficients (typiquement 1 ou -1). On retrouve cette dernière classe dans les standards définis par le NIST pour la cryptographie à base de courbes elliptiques. Plus généralement, tout entier N de la forme $\beta^n - c$ dans un système où les entiers manipulés sont représentés en base β sur n symboles et où c est "petit" permet d'effectuer une réduction modulaire efficace [BZ10]. Dans [Per03], Percival propose une méthode de réduction modulaire efficace pour des entiers N de la forme $a \pm b$ tels que $\prod_{p|ab} p$ est "petit". Remarquons que toutes ces méthodes imposent un choix particulier de N ce qui n'est pas toujours possible dans certaines applications (par exemple l'entier N de la clé publique de RSA).

Pour des modules quelconques, certaines méthodes nécessitent des pré-calculs afin de pouvoir remplacer les divisions intervenants dans l'étape de réduction modulaire par des divisions plus simples à effectuer (typiquement des divisions par une puissance de la base dans laquelle sont représentés les entiers). On citera, parmi les plus célèbres, les méthodes de Barrett et de Montgomery [BZ10, paragraphes 2.4.1 et 2.4.2]. Ces dernières sont efficaces dès lors qu'il est nécessaire d'effectuer plusieurs multiplications modulo un même entier N, le coût des pré-calculs devenant négligeable comparativement au gain obtenu sur le nombre de multiplications effectuées. La méthode de Montgomery comporte de plus une étape de conversion afin de plonger (resp. d'extraire) un entier de $\mathbb{Z}/N\mathbb{Z}$ (resp. du domaine de Montgomery) dans le domaine de Montgomery (resp. vers $\mathbb{Z}/N\mathbb{Z}$). Une version polynomiale de cette méthode étant utilisée par la suite, nous en rappelons ici brièvement le principe.

Soit t tel que $2^{t-1} \leqslant N < 2^t$, et soient a et b deux entiers dans $[0, N[$. L'idée est de rajouter au produit $c = ab$ un multiple de N tel que le résultat soit un multiple de 2^t. En d'autres termes, on cherche un entier q tel que c et $-qN$ correspondent sur les t bits de poids faible, ie. tel que $c + qN \equiv 0 \pmod{2^t}$. De cette équation, on en déduit que l'entier q recherché vaut $c(-N^{-1}) \pmod{2^t}$ (ce qui impose N impair). On peut alors facilement calculer $(c + qN)/2^t$ qui vaut $c2^{-t} \pmod{N}$. Moyennant le pré-calcul de $N' = -N^{-1} \pmod{2^t}$, la méthode de Montgomery permet donc de calculer facilement, à partir des entiers a et b, la quantité $ab2^{-t} \pmod{N}$, en effectuant uniquement 3 multiplications

($c = ab$, cN', qN), une réduction modulo 2^t (extraction des t derniers bits de cN' afin d'obtenir q) et une division par 2^t (shift de t bits de $c+qN$). La conversion dans le domaine de Montgomery consiste à appliquer ce même procédé aux entiers $\tilde{a} = a2^t \pmod N$ et $\tilde{b} = b2^t \pmod N$. On obtient ainsi en sortie la quantité $\tilde{a}\tilde{b}2^{-t} \equiv ab2^t \equiv \widetilde{ab} \pmod N$, ce qui assure la consistance des opérations dans le domaine de Montgomery (nous reviendrons page 98 sur ce problème de consistance).

Finalement, une autre possibilité permettant d'optimiser l'arithmétique modulaire est de considérer un système de représentation des entiers pour lequel les calculs à effectuer seront plus simples. C'est le cas du RNS (Residue Number System [Gar59]) qui exploite l'isomorphisme d'anneaux entre $\mathbb{Z}/N\mathbb{Z}$ et $\mathbb{Z}/m_0\mathbb{Z} \times \mathbb{Z}/m_1\mathbb{Z} \times \cdots \times \mathbb{Z}/m_{n-1}\mathbb{Z}$ où les entiers m_i sont non nuls et vérifient $N = \prod_{i=0}^{n-1} m_i$ et pour tout $i \neq j$, pgcd(m_i,m_j)=1. Chaque élément a de $\mathbb{Z}/N\mathbb{Z}$ est représenté dans ce système par la liste $(a_0, a_1, \ldots, a_{n-1})$ de ses résidus modulo m_i, les calculs modulo m_i peuvent être effectués de façon indépendante (et en parallèle) dans chacun des anneaux $\mathbb{Z}/m_i\mathbb{Z}$, sans se soucier de problèmes de propagation de retenue. La reconstruction du résultat final est assurée par le théorème des restes chinois.

2.2 Du MNS au PMNS en passant par l'AMNS

Dans ce chapitre on s'intéresse à la représentation des éléments de $\mathbb{Z}/p\mathbb{Z}$ afin d'obtenir une réduction modulaire efficace pour un entier p n'ayant pas une forme particulière. Soit n tel que $\beta^{n-1} \leqslant p < \beta^n$ et soit r un entier inférieur à p^2 (ce qui correspond à un contexte classique en cryptographie, l'entier r provenant du résultat de la multiplication de deux entiers strictement inférieurs à p). Dans le système de représentation classique en base β, on a :

$$r = \sum_{i=0}^{2n-2} r_i \beta^i, \qquad 0 \leqslant r_i < \beta\,.$$

Réduire r modulo p revient à écrire r sous la forme

$$r = \sum_{i=0}^{n-1} r_i \beta^i + \beta^n \sum_{i=0}^{n-1} r_{n+i} \beta^i\,.$$

et à ajouter la quantité $t \sum_{i=0}^{n-1} r_{n+i}\beta^i$ à $\sum_{i=0}^{n-1} r_i \beta^i$, où

$$t = \beta^n \pmod p\,.$$

On répète ce procédé jusqu'à obtenir une valeur plus petite que β^n. Selon le cas, il faudra enfin soustraire un certain nombre de fois p (au plus $\lfloor \beta^n/p \rfloor$ fois) à ce résultat intermédiaire

afin d'obtenir le résultat final.

Exemple. *Soit* $p = 251$ *et* $\beta = 10$*, on a donc* $n = 3$ *et* $t = 1000 \pmod{251} = 247$*. Considérons* $r = 27772$*. On a* $r = 772 + 1000 \times 27$*. On calcule alors* $772 + 247 \times 27 = 7441$*. On répète le procédé sur cet entier :* $7441 = 441 + 1000 \times 7 \to 441 + 247 \times 7 = 2170$*. Ceci nous amène à une dernière itération :* $2170 = 170 + 1000 \times 2 \to 170 + 247 \times 2 = 664$*. Il reste à soustraire deux fois la valeur de* p *pour obtenir le résultat voulu :* $664 - 2 \times 251 = 162$*.*

Remarquons que chacune des étapes fait intervenir une addition avec gestion de la propagation de la retenue. On peut effectuer ce même procédé en considérant une version polynomiale de la représentation des entiers. A tout entier $r = \sum_{i=0}^{2n-2} r_i \beta^i$, on peut associer le polynôme

$$R(X) = \sum_{i=0}^{2n-2} r_i X^i \;\; r_i < \beta \,,$$

de sorte que $r = R(\beta)$. Le calcul de r modulo p revient alors à calculer $R(X)$ modulo $E(X)$, où

$$E(X) = X^n - T(X) \;\text{ avec }\; T(\beta) = t \pmod{p} \,.$$

En posant

$$R(X) = \sum_{i=0}^{n-1} r_i X^i + X^n \sum_{i=0}^{n-1} r_{n+i} X^i,$$

la réduction modulo $E(X)$ revient à calculer les coefficients du polynôme

$$S(X) = \sum_{i=0}^{n-1} r_i X^i + T(X) \sum_{i=0}^{n-1} r_{n+i} X^i \,.$$

Suite à cette opération, il est indispensable d'appliquer une étape dite de "réduction" afin que les coefficients de $S(X)$ soient tous bornés par β. Ceci revient donc à trouver un nouveau polynôme $\tilde{R}(X)$, tel que

$$\tilde{R}(\beta) \equiv S(\beta) \equiv R(\beta) \pmod{p} \;\text{ avec }\; 0 \leqslant \tilde{r}_i < \beta \,.$$

On itère ce procédé jusqu'à obtenir un polynôme de degré au plus $n - 1$ qu'il faudra évaluer en β afin d'obtenir le résultat voulu (en effectuant éventuellement au plus $\lfloor \beta^n / p \rfloor$ soustractions par p).

Réduction interne et externe

Dans la suite de ce document, on appellera **réduction externe**, la réduction modulo $E(X)$, et **réduction interne**, l'étape consistant à réduire les coefficients d'un polynôme en fonction d'une certaine borne.

A propos de l'étape de réduction interne

Afin de réduire les coefficients du polynôme $S(X)$, on cherche à lui retrancher un polynôme $Z(X) \in \mathbb{Z}[X]$ de degré au plus $n-1$, de façon à ce que $0 \leqslant s_i - z_i < \beta$ et tel que $S(\beta) - Z(\beta) \equiv S(\beta) \pmod{p}$. Cette dernière relation impose que le polynôme $Z(X)$ soit choisi dans l'ensemble des polynômes de degré au plus $n-1$ qui s'annulent en β modulo p. Cet ensemble peut être associé à un réseau dans $\mathbb{Z}^n$ dont on peut facilement obtenir une base. Ce point sera abordé par la suite dans le paragraphe sur la représentation MNS.

Exemple. *Pour l'exemple précédent, on a*

$$R(X) = 2 + 7X + 7X^2 + 7X^3 + 2X^4$$

et choisissons $E(X) = X^3 - 247$. *On a alors*

$$R(X) = 2 + 7X + 7X^2 + X^3(7 + 2X)\,.$$

La réduction modulo $E(X)$ *revient donc à additionner les polynômes* $2 + 7X + 7X^2$ *et* $247(7 + 2X)$. *On obtient le polynôme*

$$S(X) = 1731 + 501X + 7X^2\,.$$

Le polynôme $Z(X) = 1728 + 500X + 3X^2$ *s'annule en* β *modulo* p *et permet de réduire* $S(X)$ *afin d'obtenir*

$$\tilde{R}(X) = S(X) - Z(X) = 3 + X + 4X^2\,.$$

On a bien $S(\beta) \equiv \tilde{R}(\beta) \pmod{p}$, *et* $\tilde{R}(\beta) = 413$. *Il suffit de soustraire* p *une fois afin d'obtenir le résultat voulu :* 162.

Cet exemple nous permet de constater que certaines des opérations effectuées dans la version polynômiale de la réduction modulaire peuvent être parallélisées (chaque coefficient d'un monôme peut être obtenu indépendamment des autres). De plus, ceci permet

de s'affranchir de la gestion de la propagation de la retenue entre les différents symboles représentant l'entier à réduire. Cependant cette méthode soulève deux autres problèmes.

Problème 1 Tout d'abord, une fois p, n et β fixés, est-on toujours assuré de pouvoir trouver un polynôme $Z(X)$ qui s'annule en β modulo p et qui permet de réduire suffisamment la taille des coefficients de $R(X)$? Si tel est le cas, la recherche du polynôme $Z(X)$ peut-elle s'effectuer de façon "efficace" afin de ne pas pénaliser le processus total de réduction modulaire ?

Problème 2 La réduction modulo $E(X) = X^n - T(X)$ de $R(X)$ s'effectue en calculant

$$S(X) = \sum_{i=0}^{n-1} r_i X^i + T(X) \sum_{i=0}^{n-1} r_{n+i} X^i \,.$$

Selon la forme de $T(X)$ ce calcul peut être "coûteux" notamment en terme de représentation des entiers intervenant dans les calculs intermédiaires. Dans l'exemple que nous avons traité, cela revient à multiplier par un entier pratiquement égal à p les coefficients de $R(X)$. Sur une architecture 64 bits et dans un cadre cryptographique, ceci impose donc une gestion des grands entiers, ce qui va alourdir le processus.

Contrainte sur $E(X)$

L'idéal serait donc de pouvoir travailler avec un polynôme de réduction $E(X)$ qui soit le plus "creux" possible afin de réduire le nombre de multiplications à effectuer et dont les coefficients soient "petits" afin que chaque multiplication soit réalisable "nativement" sur l'architecture cible.

Supposons que l'on souhaite trouver un polynôme $E(X) = X^n - T(X)$ dont les coefficients de $T(X)$ soient aussi bornés par β afin que les multiplications intermédiaires puissent être stockées sur un registre de taille $2\log_2(\beta)$ (ce qui est le cas du type `__int128` proposé par GCC pour les architectures 64 bits). Une fois p et n fixés, il n'est pas toujours possible de facilement trouver un polynôme $E(X)$ et une base β qui conviennent.

A titre d'illustration, reprenons l'exemple précédent où $p = 251$ et $n = 3$. Pour $\beta = 10$, on cherche un polynôme $E(X) = X^3 - (aX^2 + bX + c)$ qui soit "creux", dont les coefficients a, b et c sont bornés par β et qui s'annule en β modulo 251. Une simple recherche exhaus-

tive donne les polynômes $X^3-(4X^2+9X+8)$ et $X^3-(7X^2+4X+9)$ qui ne sont pas creux.

Afin d'élargir les choix possibles pour le couple $(E(X),\beta)$, tout en fournissant des solutions aux problèmes 1 et 2, Jean-Claude Bajard, Laurent Imbert et Thomas Plantard proposent en 2004 le système de représentation MNS [BIP05b] (Modular Number System).

2.2.1 Le système de représentation MNS

Dans ce système de représentation, la "base" γ utilisée est indépendante de la valeur ρ qui borne les coefficients a_i de la représentation d'un élément a de $\mathbb{Z}/p\mathbb{Z}$. Nous donnons une définition de ce système de représentation légèrement différente de la définition initiale mentionnée dans [BIP05b].

Définition 2.1

Modular Number System Soient les entiers $p \geqslant 3$, $n \geqslant 2$, $\gamma \in [1, p-1]$ et $\rho \in [1, p-1]$. Soit $E(X) = X^n - c$, où $c \equiv \gamma^n \pmod{p}$, un MNS (Modular Number System) est un ensemble $\mathfrak{B} \subset \mathbb{Z}[X]$ tel que :

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

Cet ensemble est muni de deux lois de composition interne $\tilde{+}$ et $\tilde{\times}$ tel que

— $A(X)\tilde{+}B(X)$ correspond à l'addition modulo $E(X)$ des polynômes $A(X)$ et $B(X)$.

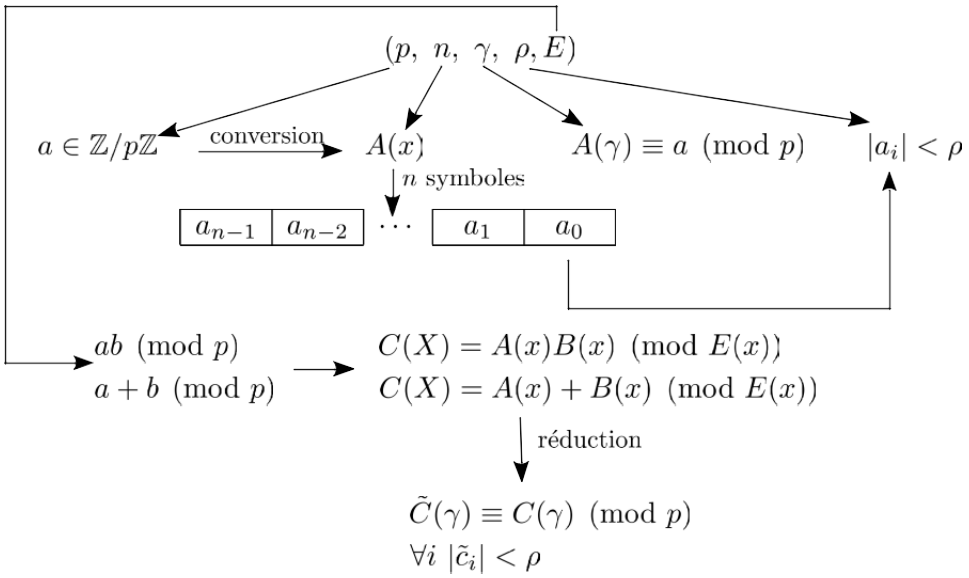
— $A(X)\tilde{\times}B(X)$ correspond à la multiplication modulo $E(X)$ des polynômes $A(X)$ et $B(X)$.

Ces deux opérations **incluent** une étape de **réduction interne** afin d'obtenir un résultat dans l'ensemble $\mathfrak{B}$. L'ensemble $\mathfrak{B}$ est entièrement déterminé par le quintuplet (p, n, γ, ρ, E) (cf. fig. ci-après).

Pour la suite on notera $+$ et $\times$ les opérations $\tilde{+}$ et $\tilde{\times}$.

Exemple. *Pour représenter les éléments de* $\mathbb{Z}/31\mathbb{Z}$ *sur 4 symboles choisis dans l'ensemble* $\{-1, 0, 1\}$*, on peut utiliser le polynôme* $E(X) = X^4 - 2$ *qui admet* $\gamma = 15$ *pour racine dans* $\mathbb{Z}/31\mathbb{Z}$*. On a donc* $p = 31$*,* $n = 4$*,* $\gamma = 15$ *et* $\rho = 2$*. La table 2.1 donne la représentation des éléments de* $\mathbb{Z}/31\mathbb{Z}$ *sous forme de polynômes de degré au plus* 3 *et dont les coefficients sont dans* $\{-1, 0, 1\}$*.*

Cette table permet immédiatement de constater que le système de représentation MNS est redondant. L'entier 12 possède par exemple 5 représentants. Pour que $\mathbb{Z}/p\mathbb{Z}$ puisse être



0	1	2	3	4	5
(0, 0, 0, 0)	(1, 0, 0, 0)	(-1, 1, -1, 1)	(-1, -1, -1, 1) (-1, 0, 0, -1) (-1, 0, 1, 1) (0, 1, -1, 1)	(0, -1, -1, 1) (0, 0, 0, -1) (0, 0, 1, 1) (1, 1, -1, 1)	(1, -1, -1, 1) (1, 0, 0, -1) (1, 0, 1, 1)
6	7	8	9	10	11
(-1, 1, -1, 0)	(-1, -1, -1, 0) (-1, 0, 1, 0) (0, 1, -1, 0)	(0, -1, -1, 0) (0, 0, 1, 0) (1, 1, -1, 0)	(1, -1, -1, 0) (1, 0, 1, 0)	(-1, 1, -1, -1) (-1, 1, 0, 1)	(-1, -1, -1, -1) (-1, -1, 0, 1) (-1, 0, 1, -1) (0, 1, -1, -1) (0, 1, 0, 1)
12	13	14	15	16	17
(0, -1, -1, -1) (0, -1, 0, 1) (0, 0, 1, -1) (1, 1, -1, -1) (1, 1, 0, 1)	(1, -1, -1, -1) (1, -1, 0, 1) (1, 0, 1, -1)	(-1, 1, 0, 0)	(-1, -1, 0, 0) (0, 1, 0, 0)	(0, -1, 0, 0) (1, 1, 0, 0)	(1, -1, 0, 0)
18	19	20	21	22	23
(-1, 0, -1, 1) (-1, 1, 0, -1) (-1, 1, 1, 1)	(-1, -1, 0, -1) (-1, -1, 1, 1) (0, 0, -1, 1) (0, 1, 0, -1) (0, 1, 1, 1)	(0, -1, 0, -1) (0, -1, 1, 1) (1, 0, -1, 1) (1, 1, 0, -1) (1, 1, 1, 1)	(1, -1, 0, -1) (1, -1, 1, 1)	(-1, 0, -1, 0) (-1, 1, 1, 0)	(-1, -1, 1, 0) (0, 0, -1, 0) (0, 1, 1, 0)
24	25	26	27	28	29
(0, -1, 1, 0) (1, 0, -1, 0) (1, 1, 1, 0)	(1, -1, 1, 0)	(-1, 0, -1, -1) (-1, 0, 0, 1) (-1, 1, 1, -1)	(-1, -1, 1, -1) (0, 0, -1, -1) (0, 0, 0, 1) (0, 1, 1, -1)	(0, -1, 1, -1) (1, 0, -1, -1) (1, 0, 0, 1) (1, 1, 1, -1)	(1, -1, 1, -1)
30					
(-1, 0, 0, 0)					

Table 2.1 – Représentation des éléments de $\mathbb{Z}/31\mathbb{Z}$ dans le MNS défini par les paramètres $(31, 4, 15, 2, X^4 - 2)$.

représenté par un MNS, il faut que $(2\rho - 1)^n \geqslant p$, c'est à dire $\rho \geqslant \frac{p^{1/n}+1}{2}$. Le système est

donc redondant dès que

$$\rho > \frac{p^{1/n}+1}{2}\,.$$

2.2.2 Réseau associé à un MNS

Supposons pour l'exemple précédent que l'entier 5 soit représenté par le polynôme $X^3 - X^2 - X + 1$. L'entier 29 est représenté quant à lui par le polynôme $-X^3 + X^2 - X + 1$. Le produit de ces deux représentants modulo $X^4 - 2$ est le polynôme $-X^2 + 2X - 1$. L'étape de réduction interne revient à soustraire à ce dernier :

— soit le polynôme $X^3 - X^2 + 3X - 2$ et obtenir ainsi le représentant de 21 qui vaut $-X^3 - X + 1$,

— soit le polynôme $-X^3 - 2X^2 + 3X - 2$ et obtenir ainsi le représentant de 21 qui vaut $X^3 + X^2 - X + 1$.

Dans les 2 cas, on a soustrait un polynôme qui s'annule en 15 modulo 31, ce qui nous permet d'introduire la notion de réseau lié à un MNS.

Définition 2.2. *Soit $\mathcal{B}$ un MNS défini par le quintuplet (p, n, γ, ρ, E). Le réseau $\mathcal{L}$ associé au MNS $\mathcal{B}$ est le réseau de dimension n engendré par les polynômes de $\mathbb{Z}[X]$ de degré au plus $n-1$ admettant γ pour racine dans $\mathbb{Z}/p\mathbb{Z}[X]$. Ce réseau est engendré par la matrice A suivante, où chaque ligne est la décomposition dans la base canonique $\{1, X, X^2, \ldots, X^{n-1}\}$ d'un polynôme s'annulant en γ modulo p.*

$$A = \begin{pmatrix} p & 0 & \cdots & \cdots & 0 & 0 \\ -\gamma & 1 & \cdots & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & & & \vdots \\ 0 & \cdots & -\gamma & 1 & \cdots & 0 \\ \vdots & & & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \cdots & -\gamma & 1 \end{pmatrix}.$$

Remarque. *Une autre base de $\mathcal{L}$ est donnée par la matrice*

$$\begin{pmatrix} p & 0 & \cdots & \cdots & 0 & 0 \\ -\gamma & 1 & \cdots & \cdots & 0 & 0 \\ -\gamma^2 \bmod p & 0 & 1 & & & \vdots \\ \vdots & & & \ddots & \cdots & 0 \\ \vdots & & & & \ddots & \vdots \\ -\gamma^{n-1} \bmod p & 0 & \cdots & \cdots & \cdots & 1 \end{pmatrix}.$$

Comme on le verra par la suite, ce réseau joue un rôle crucial non seulement d'un point

de vue théorique lorsque l'on s'intéresse au problème de l'existence d'un MNS, mais aussi d'un point de vue pratique lorsque l'on cherche à obtenir une représentation permettant d'effectuer des calculs modulaires efficaces.

2.2.3 Le système de représentation AMNS

Soient $A(X) \in \mathcal{B}$ et $B(X) \in \mathcal{B}$ les représentants des entiers a et b dans le système MNS $\mathcal{B}$ défini par les paramètres (p, n, γ, ρ, E). Afin de calculer $ab \bmod p$, on calcule tout d'abord le produit $R(X) = A(X)B(X)$. La réduction externe (réduction modulo $E(X) = X^n - c$, $c \equiv \gamma^n \bmod p$) de $R(X)$ s'effectue en calculant

$$S(X) = \sum_{i=0}^{n-1} r_i X^i + c \sum_{i=0}^{n-1} r_{n+i} X^i \,,$$

ce qui revient à calculer les quantités $r_i + c r_{n+i}$ pour $i \in [0, n-1]$. Les coefficients de $A(X)$ et $B(X)$ étant bornés par ρ, cette opération nécessite n multiplications entre la constante c et des entiers de taille au plus $\log_2(n\rho^2)$. Afin que ces produits soient "peu coûteux", Jean-Claude Bajard, Laurent Imbert et Thomas Plantard suggèrent de choisir une constante c "petite". Par exemple, sur une architecture k bits sur laquelle il existe une multiplication exacte entre entiers de k bits, choisir c, n et ρ tels que $\log_2(c) < 2k - \log_2(n\rho^2)$, permet d'effectuer nativement chaque multiplication. Si de plus c est une "petite" puissance de 2, ces calculs s'effectuent de façon efficace avec de simples décalages. C'est ainsi qu'est défini le système de représentation AMNS (Adapted Modular Number System).

Définition 2.3 (définition 2,[BIP05b])**.** *Un système de représentation AMNS est un système de représentation MNS défini par les paramètres* (p, n, γ, ρ, E) *pour lequel* $\gamma^n \pmod p$ *est une "petite" constante.*

A ce stade, une question naturelle se pose sur l'existence d'un tel système. Dans [BIP05b], les auteurs proposent la méthodologie suivante pour construire un AMNS :

1. Choisir $\rho = 2^k$ pour une architecture cible de k bits. Ainsi les coefficients des polynômes sont stockés sur des mots machines.
2. Choisir n tel que kn soit de l'ordre de la taille du modulo intervenant dans l'application ciblée.
3. Choisir un "petit" entier c (pour le polynôme de réduction externe $(E(X) = X^n - c)$.
4. Déduire de ces 3 paramètres, l'entier p qui est un diviseur du déterminant d'une certaine matrice dépendant de n, k et c (cf. [BIP05b, eq. 14]).
5. Calculer γ qui est une racine dans $\mathbb{Z}/p\mathbb{Z}$ du pgcd de $x^n - c$ et $2^{k-1} - \sum_{i=0}^{n-1} \zeta_i x^i$, où $(\zeta_0, \ldots, \zeta_{n-1})$ est une représentation dans l'AMNS de 2^{k-1}.

Cette construction amène quelques commentaires :

— tout d'abord l'étape 4 implique que cette méthodologie **ne permet pas** de construire un AMNS pour un entier p fixé, mais seulement pour une taille d'entier donné. En pratique, les auteurs précisent que l'on obtient généralement des entiers de taille légèrement inférieur à nk. Cette construction est donc inutilisable dans le cadre d'un standard cryptographique où l'anneau $\mathbb{Z}/p\mathbb{Z}$ est fixé à l'avance.

— l'étape 5 nécessite la représentation de 2^{k-1} dans l'AMNS afin de calculer γ qui permet de construire l'AMNS ! Il s'agit en fait de choisir aléatoirement une représentation de 2^{k-1} (tirage aléatoire des ζ_i) sans connaître γ jusqu'à obtenir un pgcd avec $x^n - c$ différent de 1. Les coefficients ζ_i choisis doivent être "petits" (cf. ci-après)

Concernant la réalisation efficace de la réduction interne, elle est détaillée dans les algorithmes 2 et 3 de [BIP05b]. Le théorème 1 assure la validité de cette réduction dès lors que les éléments ζ_i satisfont

$$c \sum_{i=0}^{n-1} \zeta_i < 2^{\lfloor (k-1)/2 \rfloor}.$$

Pour conclure, cette construction ne permet pas de savoir en fixant n et $\rho = 2^k$ s'il existe une représentation adéquate de 2^{k-1} assurant la construction d'un AMNS pour une taille n donnée. De plus qu'en est-il de l'existence d'un AMNS pour $\rho \neq 2^k$?

2.2.4 Le système de représentation PMNS

Dans [BIP05a], Jean-Claude Bajard, Laurent Imbert et Thomas Plantard s'intéresse à la question suivante : pour p et n fixés, pour quelles valeurs de ρ peut-on construire un MNS ? Pour répondre à cette question ils introduisent le système de représentation PMNS.

Définition 2.4. *Un système de représentation PMNS est un système de représentation MNS défini par les paramètres (p, n, γ, ρ, E) pour lequel :*

— $\alpha, \beta \in \mathbb{Z}$,

— $E(X) = X^n + \alpha X + \beta$ *est un polynôme irréductible dans* $\mathbb{Z}[X]$.

Remarque. *Le polynôme $E(X)$ étant unitaire, il est primitif dans $\mathbb{Z}[X]$ (son contenu vaut 1), il est donc aussi irréductible dans $\mathbb{Q}[X]$.*

Afin d'obtenir une réduction externe "efficace", les auteurs suggèrent de choisir α et β "petits".

La classe PMNS ainsi définie englobe-t-elle les AMNS ? Pas complètement. Pour les PMNS, il est imposé que $E(X)$ soit irréductible, ce qui n'est pas le cas pour les AMNS.

Remarque. *D'après le corollaire 1.2 de [Bon15], soit $c \in \mathbb{Z}$ tel que $|c| = \prod_{i=1}^{k} p_i^{e_i}$ (décomposition en facteurs premiers), si les pgcd des couples (e_i, n) sont premiers entre eux, alors $X^n + c$ est irréductible dans $\mathbb{Z}[X]$.*

L'existence d'un système de représentation PMNS est donné par le résultat suivant :

Théorème 2.1. *[BIP05a, Théorème 1] Soient $p, n > 1$ deux entiers et $E(X) = X^n + \alpha X + \beta$ un polynôme irréductible dans $\mathbb{Z}[X]$ avec $\alpha,\ \beta \in \mathbb{Z}$. Soit γ tel que $E(\gamma) \equiv 0 \pmod p$, il existe un PMNS de paramètre (p, n, γ, ρ, E) si*

$$\rho \geqslant (|\alpha| + |\beta|)p^{1/n}\,.$$

Remarque. *La démonstration*[2] *comporte une légère "coquille" qui ne remet pas en cause le reste des différents résultats et algorithmes présentés dans le papier. La borne devrait être*

$$\rho \geqslant \frac{n}{2}(|\alpha| + |\beta|)p^{1/n}\,,$$

ce qui en pratique au vu de la taille de p (un entier de 2048 bits par exemple) par rapport à celle de n (4 bits pour une architecture 64 bits) ne change pas grand chose sur la taille des coefficients des représentants. Sans rentrer dans le détail complet de la démonstration, cette dernière utilise le fait que si un vecteur v' appartient au domaine fondamental $\mathcal{H}'$ d'un certain réseau $\mathcal{L}'$ alors

$$\|v'\|_\infty \leqslant \max_{0\leqslant i<n} \|b_i\|_\infty$$

où les vecteurs b_i forment une base de $\mathcal{L}'$. Le domaine fondamental canonique d'un réseau, dont une base est donnée par les vecteurs $(b_i)_{i=0\ldots n-1}$, est défini par

$$\{t \in \mathbb{R}^n \mid t = \sum_{i=0}^{n-1} \mu_i b_i \text{ et } -1 \leqslant \mu_i < 1\}\,.$$

On en déduit donc en fait que

$$\|v'\|_\infty \leqslant n \max_{0\leqslant i<n} \|b_i\|_\infty\,.$$

Le translaté $\mathcal{H}'$ de ce domaine, défini par

$$\mathcal{H}' = \{t \in \mathbb{R}^n \mid t = \sum_{i=0}^{n-1} \mu_i B_i \text{ et } -\frac{1}{2} \leqslant \mu_i < \frac{1}{2}\}$$

est aussi un domaine fondamental, ce qui permet d'affiner la borne précédente :

$$\|v'\|_\infty \leqslant \frac{n}{2} \max_{0\leqslant i<n} \|b_i\|_\infty\,.$$

Ceci explique le facteur $\frac{n}{2}$ qui doit apparaître dans la borne. Dans le paragraphe 2.4.6, je

2. page 8, https://hal-lirmm.ccsd.cnrs.fr/lirmm-00109201/document

propose une borne plus fine pour des polynômes non nécessairement irréductibles.

2.3 A propos de la réduction interne

Soit $\mathcal{B}$ un MNS de paramètre (p, n, γ, ρ, E) et soient $A(X)$ et $B(X)$ des représentants des éléments a et b de $\mathbb{Z}/p\mathbb{Z}$. Posons $C(X) = A(X)B(X) \bmod E(X)$. En considérant le résau $\mathcal{L}$ associé à $\mathcal{B}$ (cf. définition 2.2.3, page 90), la réduction interne revient à trouver dans le réseau $\mathcal{L}$ un polynôme $Z(X)$ qui soit "suffisamment proche" de $C(X)$ de façon à ce que $\|C - Z\|_\infty = \max_{i=0\ldots n-1} |c_i - z_i| < \rho$. Les algorithmes classiques (issus de la théorie des réseaux) pour résoudre le problème CVP_∞ [MG02] ne sont pas adaptés à ce problème car :

— d'une part ils ne sont pas utilisables en pratique [HPS11] si l'on souhaite obtenir un processus de multiplication performant (CVP_∞ est NP-difficile [Emd81]),

— d'autre part, on ne cherche pas forcément le vecteur le plus proche de C, mais un vecteur qui soit à une distance d'au plus ρ.

Il existe à l'heure actuelle quatre algorithmes (polynomiaux en temps) permettant de résoudre efficacement le problème de la réduction interne lorsque l'on se fixe le paramètre p :

1. Babai nearest plane [Bab86, théorème 3.1],
2. Babai rounding technique [Bab86, théorème 3.2],
3. Barret-like [BIP05a],
4. Montgomery-like [NP08].

Jusqu'à très récemment, la méthode Montgomery-like était considérée comme la plus efficace. Des travaux de Nicolas Méloni (en cours de rédaction) montrent que la méthode Babai rounding technique constitue une alternative à ne pas négliger. La table 2.2 résume la complexité de ces différentes méthodes suite aux implémentations, réalisées par Nicolas Méloni [3], des deux algorithmes de Babai .

Méthode	Mult	Add	Shift
Barret-like	$2n^2$	$6n^2 - 3n$	n
Montgomery-like	$2n^2$	$4n^2 - n$	n
Babai Nearest Plane	$2n^2$	$4n^2 - 3n$	$n^2 + n$
Babai Rounding	$2n^2$	$4n^2 - 3n$	$2n$

Table 2.2 – Complexité de la réduction interne.

3. https://plmlab.math.cnrs.fr/melon359/pmns

Complexité de la réduction interne

Bien que moins coûteuse en terme d'additions que Montgomery-like, Babai Nearest Plane nécessite beaucoup plus de décalages et de déplacement mémoires (cf. code source de Nicolas Méloni), ce qui pénalise ses performances globales sur une architecture 64 bits.

Les n décalages effectués dans le Montgomery-like correspondent à la division par ϕ. Sur une architecture 64 bits, ces décalages sont effectués sur des variables de type `__int128` (mot de 128 bits). Pour $\phi = 2^{64}$ le compilateur `gcc` réalise ce décalage en 2 instructions `movq` (déplacement des 64 bits de poids fort du mot de 128 bits vers un registre de 64 bits et transfert de ce dernier vers l'adresse mémoire cible). Parmi les $2n$ décalages effectués dans Babai Rounding, n sont des décalages de h bits de variables de 64 bits avec $h < 64$. Le compilateur `gcc` effectue cette opération en 3 instructions `movq` et 1 instruction `shrq`. Ceci est illustré dans le code ci-dessous. Le coût des décalages dans Montgomery-like est donc de $2n$ instructions élémentaires et de $6n$ instructions élémentaires pour Babai Rounding sur une architecture 64 bits.

Code C

```
__int128 a;
unsigned long int c, d;

...
c = a >> 64;

d = c >> 57;
```

Code Assembleur

```
movq    8+a(%rip), %rdx
movq    %rdx, c(%rip)

movq    c(%rip), %rdx
movq    %rdx, %rax
shrq    $57, %rax
movq    %rax, d(%rip)
```

La table 2.3 donne le nombre de cycles CPU pour les méthodes Babai Nearest Plane, Babai Rounding et Montgomery-like pour l'entier p de 521 bits suivant :
434260661432094042153240807684611919362342866660519438208556213888938779942470310398797678721532919439665321793901239475575152470104910070754656276359929663 7.

$\log_2 p$	n	Méthode	Cycles CPU
521	10	Babai Nearest Plane	500
		Babai Rounding	300
		Montgomery like	304

Table 2.3 – Nombre de cycles pour la réduction interne. Intel Core i7-6700 3.40GHz, `gcc -O3 -funroll-loops`.

Il existe une cinquième méthode de réduction utilisant des tables stockées en mémoire [BIP05a]. L'utilisation de telles tables dans un contexte cryptographique nécessite de mettre en place des contremesures lors de la lecture en table afin d'éviter les fuites d'information [Car+19], ce qui va alourdir le processus de réduction interne.

2.3.1 Montgomery-like

La méthode Montgomery-like proposée par Christophe Negre et Thomas Plantard est une "extension" aux polynômes de la méthode de Montgomery classique. Dans la méthode classique on rajoute un multiple du module $N < 2^k$ (donc un représentant de 0 dans la classe modulo N) à l'entier $c = ab$ de façon à obtenir un multiple de 2^k. L'algorithme renvoie alors la valeur de $c2^{-k} \bmod N$.

Dans [NP08], Christophe Negre et Thomas Plantard proposent une méthode analogue dans le cadre d'un AMNS défini par le polynôme $E(X) = X^n - \lambda$. Il s'agit de rajouter au polynôme $C(X) = A(X)B(X) \pmod{E(X)}$ un multiple d'un polynôme $M(X)$ qui est un représentant de 0 dans l'AMNS, de façon à ce que chacun des coefficients de $C(X) + Q(X)M(X)$ soit un multiple d'un certain paramètre ϕ (on choisira généralement $\phi = 2^k$ où k est la taille des mots sur l'architecture cible). Le calcul de $Q(X)M(X)$ est effectué modulo le polynôme $E(X)$. La réduction consiste alors à calculer $R(X) = (C(X)+Q(X)M(X))/\phi$. La valeur en γ de $C(X) + Q(X)M(X)$ est la même que celle de $C(X)$ (modulo p) et on espère que la division par ϕ réduise suffisamment la taille des coefficients pour être de nouveau dans l'AMNS.
L'algorithme renvoie alors le polynôme $R(X)$ qui vérifie

$$R(\gamma) \equiv C(\gamma)\phi^{-1} \mod p\,.$$

Le polynôme $Q(X)$ recherché doit vérifier

$$C(X) + Q(X)M(X) \equiv 0 \bmod (E(X), \phi)\,.$$

Ainsi

$$Q(X) \equiv -M^{-1}(X)C(X) \bmod (E(X), \phi)\,.$$

Le théorème 1 de [NP08] donne les conditions suffisantes sur ρ, M et ϕ pour que l'algorithme 20 effectue la réduction interne :

$$\rho \geqslant 2|\lambda|n\|M\|_\infty, \quad \phi \geqslant 2|\lambda|n\rho\,.$$

Algorithm 20 Multiplication modulaire dans l'AMNS, [NP08]

Require: $A(X)$ et $B(X) \in \mathcal{B} = (p,\, n,\, \gamma,\, \rho, E)$, et $M'(X) = -M^{-1}(X) \bmod (E(X), \phi)$
Ensure: $R(X) \in \mathcal{B}$ et $R(\gamma) \equiv A(\gamma)B(\gamma)\phi^{-1} \pmod p$

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

Quelques remarques

Les lignes 2 à 5 de l'algorithme constituent le processus de réduction interne. On en déduit que ce processus de réduction est applicable à tout polynôme $C(X) \in \mathbb{Z}[X]$ tel que $\|C\|_\infty < n|\lambda|\rho^2$. Ce résultat sera généralisé dans la partie "Contributions" pour un PMNS quelconque.

Les conditions assurant le bon fonctionnement de l'algorithme imposent de trouver un polynôme M dont la norme infinie soit "petite" afin que ρ ne soit pas "trop grand".

Les conditions d'existence d'un polynôme M inversible modulo $(E(X), \phi)$ seront discutées dans la partie "Contributions".

2.3.2 Babai rounding technique

Soit B une base d'un réseau $\mathcal{L}$ et soit $C = (c_0, c_1, \ldots, c_{n-1})$ un vecteur de $\mathbb{R}^n$. La base B est représentée sous forme matricielle, où chaque ligne B_i $(i = 0, \ldots, n-1)$ correspond à un élément de la base. Il existe $V = (v_0, v_1, \ldots v_{n-1}) \in \mathbb{R}^n$, tel que

$$C = VB\,.$$

On a donc $V = CB^{-1}$. Posons $S = (s_0, \ldots, s_{n-1})$, tel que $s_i = \lfloor v_i \rceil$ (entier le plus proche de v_i). L'algorithme renvoie comme vecteur proche de C, le vecteur $R \in \mathcal{L}$ défini par

$$R = SB\,,$$

i.e

$$R = \sum_{i=0}^{n-1} \lfloor v_i \rceil B_i\,.$$

On a

$$\|C - R\|_\infty = \max_{i=0\ldots n-1} |\sum_{j=0}^{n-1} (v_j - \lfloor v_j \rceil) B_{ji}| \leqslant \frac{1}{2}\|B\|_1\,.$$

Babai Rounding

Cette méthode de réduction peut donc être utilisée pour **tout MNS** (p, n, γ, ρ, E) où le paramètre ρ vérifie

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

la matrice B étant une base du réseau $\mathcal{L}$ constitué des polynômes de degré au plus $n-1$ à coefficients dans $\mathbb{Z}$ qui s'annulent en γ modulo p, et γ étant une racine dans $\mathbb{Z}/p\mathbb{Z}$ de $E(X)$.

2.4 Contributions dans le domaine du système de représentation MNS

Les résultats obtenus dans [DDV20 ; Did+19] avec mes co-auteurs Laurent-Stéphane Didier, Yssouf Dosso, Jérémy Marrez et Nadia El Mrabet concernent :

1. la description complète de la génération des paramètres d'un système de représentation AMNS, ainsi que de l'ensemble des algorithmes nécessaires à son utilisation en pratique,
2. l'introduction d'un nouveau paramètre δ dans la réduction Montgomery-like permet-

tant d'effectuer dans l'AMNS δ additions successives suivies d'une multiplication en effectuant une seule réduction interne pour l'ensemble de ces opérations,

3. la mise en évidence de l'efficacité de ce système de représentation par rapport aux autres standards actuels tels que la librairie GNU MP ou la librairie OpenSSL,
4. une preuve constructive de l'existence du polynôme M indispensable au bon déroulement de la méthode Montgomery-like,
5. une nouvelle définition d'un système de représentation PMNS,
6. l'étude de la redondance du système de représentation PMNS en tant que contre-mesure aux attaques par canaux auxiliaires.

Concernant le point 1, je ne rentrerai pas dans l'ensemble des détails donnés dans [DDV20]. Notamment, je ne détaillerai pas ici les processus de conversion d'un entier de $\mathbb{Z}/p\mathbb{Z}$ vers un représentant de l'AMNS et vice-versa. Je n'aborderai pas non plus le problème de consistance induit par la méthode Montgomery-like : si $A(X)$ et $B(X)$ sont deux représentants des entiers a et b, l'algorithme de multiplication (cf. algorithme 20), ne renvoie pas un représentant de $ab \bmod p$ mais de $ab\phi^{-1} \bmod p$. Ainsi, si on enchaîne plusieurs multiplications, il faudrait garder une trace du nombre de multiplications effectuées afin de retrouver le résultat correct dans $\mathbb{Z}/p\mathbb{Z}$. Pour éviter cela, il faut (tout comme dans le cas de la multiplication de Montgomery classique), "plonger" les entiers dans le domaine de Montgomery (cf. [DDV20, Algorithme 6]).

Parallèlement à ces contributions, je profiterai de ce document pour donner

— une preuve du théorème d'existence d'un PMNS pour un polynôme $E(X)$ quelconque non nécessairement irréductible,
— une généralisation de la réduction interne Montgomery-like applicable à tout polynôme $E(X)$,
— une autre version de la multiplication randomisée pour le système de représentation PMNS,
— quelques remarques concernant le paramètre γ, LLL et le polynôme $E(X)$.

Ces derniers points sont issus d'un travail de recherche en cours développé en collaboration avec Yssouf Dosso. Les deux premiers sont une généralisation de résultats présentés dans sa thèse dans le cadre particulier des AMNS. Le dernier point concerne une nouvelle voie de recherche qui devrait aboutir à l'élaboration de PMNS plus efficaces que les AMNS.

2.4.1 A propos du polynôme $M(X)$ dans la méthode Montgomery-like

Afin d'utiliser la méthode Montgomery-like, il est indispensable de construire un polynôme $M(X)$ qui soit inversible dans $(\mathbb{Z}/\phi\mathbb{Z})[X]/E(X)$. Ceci permet alors de définir le

polynôme $M'(X)$ tel que $M(X)M'(X) = -1 \bmod (E(X), \phi)$.
Dans [NP08], les auteurs suggèrent de choisir $M(X)$ tel que pgcd$(E(X), M(X))$=1. Or ceci n'est pas suffisant. En effet si $M(X)$ est premier avec $E(X)$, il existe un polynôme $N(X) \in \mathbb{Q}[X]$ tel que $M(X)N(X) = 1 \bmod E(X)$. Posons $M'(X) = -N(X)$. Puisque $M(X)N(X) = 1 \bmod E(X)$, on a bien $M(X)M'(X) = -1 \bmod (E(X), \phi)$. Cependant rien ne garantit que les dénominateurs des coefficients de $N(X)$ soient inversibles modulo ϕ et donc que $N(X)$ soit l'inverse de $M(X)$ dans $(\mathbb{Z}/\phi\mathbb{Z})[X]/E(X)$. Or dans la ligne 2 de l'algorithme 20, le polynôme $M'(X)$ est utilisé dans un calcul faisant intervenir une réduction modulo ϕ.

Exemple. *Posons $E(X) = X^5 + X + 1$, $M(X) = X^4 + X^2 + X + 1$ et $\phi = 4$. On a bien pcdg$(E(X), M(X))$=1. Le polynôme $N(X)$ vaut $-\frac{1}{2}X^3 + \frac{1}{2}X^2 - \frac{1}{2}X$ et l'entier 2 ne possède pas d'inverse modulo 4.*

Dans [EG12], les auteurs s'intéressent au cas particulier où $E(X) = X^n + 1$ et $\phi = 2^k$. Ils affirment qu'un polynôme $M(X)$ possède un inverse dans $(\mathbb{Z}/\phi\mathbb{Z})[X]$ si $M(a) \equiv 1 \bmod 2$, $\forall a \in \mathbb{Z}$ (cf. preuve du lemme 4). Cette condition n'est pas suffisante.

Exemple. *Considérons $E(X) = X^6 + 1$, $\phi = 4$ et $M(X) = X^4 + X^2 + 1$. Ce polynôme vérifie bien la condition énoncée et il est inversible modulo $E(X)$. Son inverse vaut $\frac{1}{2}X^2 + \frac{1}{2}$ et l'entier 2 ne possède pas d'inverse modulo 4.*

Dans [DDV20], nous donnons les conditions suffisantes pour assurer l'existence du polynôme $M(X)$ pour tout polynôme $E(X)$ de la forme $X^n - \lambda$. Cette condition d'existence repose sur la notion de résultant de deux polynômes.

Définition 2.5. *Soient $A(X) = a_0 + a_1X + \cdots + a_nX^n$ et $B(X) = b_0 + b_1X + \cdots + b_mX^m$ deux polynômes de $\mathbb{Z}[X]$. Le résultant de $A(X)$ et $B(X)$, noté $Res(A, B)$, est le déterminant de leur matrice de Sylvester définie par*

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

On obtient alors le critère d'existence suivant :

Proposition 2.1

Soient $M(X)$ et $E(X)$ deux éléments de $\mathbb{Z}[X]$ et $\phi = 2^k$ un entier. Si $Res(E, M)$ est impair, alors $M(X)$ possède un inverse dans $(\mathbb{Z}/\phi\mathbb{Z})[X]$.

Démonstration. cf. Proposition 7 et Corollaire 5 de [DDV20]. □

Cette condition est valable pour tout polynôme $E(X)$ non nécessairement de la forme $X^n - \lambda$. Elle permet donc de justifier l'existence du polynôme $M(X)$ pour tout PMNS.

En fonction de la parité de λ, nous donnons alors des conditions suffisantes sur $M(X)$ afin que $Res(E, M)$ soit impair.

Proposition 2.2

Soient $M(X) = m_0 + m_1X + \cdots + m_{n-1}X^{n-1}$ et $E(X) = X^n - \lambda$ deux éléments de $\mathbb{Z}[X]$ et $\phi = 2^k$ un entier :

- si λ est pair et m_0 est impair, alors $Res(E, M)$ est impair,
- si λ est impair et $\text{pgcd}(\overline{M}(X), X^n - 1) = 1$, alors $Res(E, M)$ est impair.

où $\overline{M}(X)$ est le polynôme défini par $\overline{m}_i = mi \bmod 2$.

Démonstration. cf. Propositions 8 et 10 de [DDV20]. □

Ces conditions sont en fait nécessaires et suffisantes (cf. [CEH21, paragraphe C]).

Finalement dans le paragraphe 5.4 de [DDV20], nous montrons comment construire de façon explicite le polynôme $M(X)$.
Dans le cas λ pair, une base réduite du réseau des polynômes s'annulant en γ modulo p contient au moins un élément dont le coefficient constant est impair [DDV20, Proposition 11]. Le choix de $M(X)$ nécessite donc l'énumération d'au plus n polynômes.
Dans le cas λ impair, en partant d'une base particulière des polynômes s'annulant en γ modulo p, on montre qu'il existe alors dans la base réduite une combinaison linéaire à coefficients dans $\{0, 1\}$ donnant le polynôme $M(X)$ qui vérifie $\text{pgcd}(\overline{M}(X), X^n - 1)=1$ [DDV20, Proposition 12]. Dans ce cas, le choix de $M(X)$ nécessite donc l'énumération d'au plus 2^n polynômes. Notons que pour des entiers de 256 bits à 512 bits qui sont typiquement utilisés en cryptographie elliptique, cette recherche exhaustive n'est pas coûteuse.

2.4.2 Quelques additions "gratuites"

La réduction interne est une étape qui a un coût non négligeable (cf. tableau 2.2, page 93). Si elle s'avère être quasiment indispensable après une multiplication, la question se pose pour l'addition. En effet, l'addition de deux polynômes ne va faire grossir chaque coefficient que d'au plus 1 bit. Dans [DDV20], nous nous sommes donc intéressés à la possibilité d'effectuer dans l'AMNS successivement δ additions suivies d'une multiplication tout en ne réalisant qu'un seul appel à la procédure de réduction interne. Plus précisément :

Proposition 2.3

Soient $A(X)$ et $B(X)$ deux éléments d'un AMNS, obtenus après au plus δ additions (sans réduction interne). L'algorithme 20 (page 96) de multiplication appliqué à $A(X)$ et $B(X)$ renvoie un résultat $R(X)$ dans l'AMNS si :

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

Démonstration. cf. [DDV20, Proposition 2]. □

En pratique ce paramètre δ se règle en fonction de l'application cible (par exemple une formule d'addition de points sur une courbe elliptique).

Cette nouvelle condition n'impacte pas la valeur ρ, elle n'a donc pas d'incidence sur la taille des coefficients. Dans le paragraphe 2.4.7, elle sera généralisée à tout PMNS.

2.4.3 Méli-mélo de paramètres

Quand on souhaite construire un AMNS et utiliser la méthode Montgomery-like, un nombre important de paramètres sont à préciser : p, n, λ, ϕ, ρ, δ, $M(X)$, et γ ! Dans [DDV20], nous proposons un ordre et une méthodologie afin de générer l'ensemble de ces paramètres. Plaçons nous dans le cadre d'une implémentation logicielle sur une architecture k bits où la division par 2^k s'effectue en quelques cycles. Les étapes de construction de l'AMNS sont les suivantes :

1. Fixer l'entier p.
2. Fixer la valeur δ (nombre d'additions "gratuites") en fonction de l'application visée (ou choisir $\delta = 0$).
3. Poser $\phi = 2^k$.
4. Choisir n (nombre de symboles pour représenter un entier de $\mathbb{Z}/p\mathbb{Z}$) tel que $2^{nk} \geqslant p$, i.e calculer $n = \lfloor \text{taille}(p)/k \rfloor + 1$.

5. Choisir λ "petit" tel que λ soit un $n^{\text{ième}}$ résidu modulo p (ce choix est toujours possible dans le cas pgcd$(n, p-1)$=1).
6. Extraire une racine γ (modulo p) de $E(X) = X^n - \lambda$.
7. Chercher le polynôme $M(X)$ dans la base réduite du réseau associé aux polynômes de degré au plus $n-1$ qui s'annulent en γ modulo p.
8. Poser $\rho = 2n|\lambda|\|M\|_\infty$.

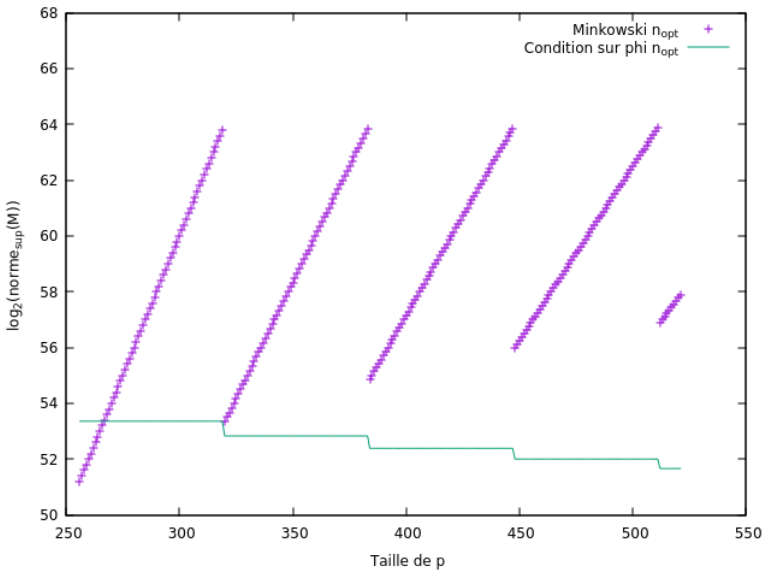


Figure 2.1 – Conditions à respecter sur $\|M(X)\|_\infty$ pour $\delta = 0$, $\lambda = 4$, $n = n_{\text{opt}}$.

Cette construction amène quelques commentaires. Rappelons que le paramètre $\phi = 2^k$ doit vérifier :

$$\phi \geqslant 2n|\lambda|\rho(\delta+1)^2 .$$

On en déduit qu'à l'étape 7, il faut trouver un polynôme $M(X)$ tel que :

$$\|M(X)\|_\infty \leqslant \frac{\phi}{(2n|\lambda|(\delta+1))^2} .$$

L'algorithme LLL dépend de deux paramètres (α, η), avec $\frac{1}{4} < \delta \leqslant 1$ et $\frac{1}{2} \leqslant \eta < \sqrt{\delta}$, et renvoie une base $(b_1, \ldots, b_n)$ du réseau telle que [NS06] :

$$\|b_1\|_2 \leqslant \left(\frac{1}{\alpha - \eta^2}\right)^{\frac{n-1}{4}} p^{\frac{1}{n}} .$$

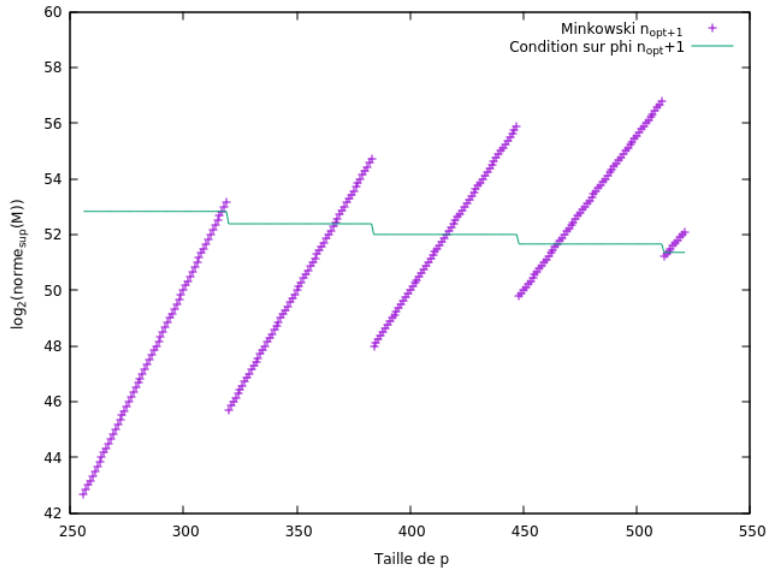


Figure 2.2 – Conditions à respecter sur $\|M(X)\|_\infty$ pour $\delta = 0$, $\lambda = 4$, $n = n_{\text{opt}} + 1$.

Plus le couple (α, η) est proche de $(1, 0.5)$ et plus la borne est petite. On a alors dans ce cas

$$\|b_1\|_2 \lesssim \left(\frac{4}{3}\right)^{\frac{n-1}{4}} p^{\frac{1}{n}}.$$

Dans la librairie Sagemath, on a $\alpha = 0.99$ et $\eta = 0.501$[4]. Dans [NS06], les auteurs constatent qu'en pratique, LLL renvoie une base telle que :

$$\|b_1\|_2 \approx (1.02)^n p^{\frac{1}{n}}.$$

Dans [Odl90, page 14], il est suggéré qu'en petite dimension (ce qui est notre cas), LLL renvoie le vecteur le plus court du réseau. Or, d'après le théorème de Minkowski, ce vecteur est de norme infinie inférieure ou égale à $p^{\frac{1}{n}}$ et en pratique proche de $p^{\frac{1}{n}}$. On ne peut donc pas espérer trouver un "bon" candidat $M(X)$, si $p^{\frac{1}{n}} >> \frac{\phi}{(2n|\lambda|(\delta+1))^2}$.
A titre d'exemple pour un entier p de 256 bits et $k = 64$, supposons $\delta = 0$, $n = 5$, $\lambda = 4$, on doit trouver $M(X)$ tel que $\|M(X)\|_\infty \leqslant 2^{53.35}$ et d'après Minkowski, le réseau contient un élément de norme infinie inférieure ou égale à $p^{1/n} \approx 2^{51}$, on se trouve donc ici dans un contexte favorable.
Cette construction que nous proposons qui consiste à tirer partie du faible coût d'une division par 2^k donne de bons résultats sur une architecture 64 bits dans le cadre de la

4. https://doc.sagemath.org/html/en/reference/matrices/sage/matrix/matrix_integer_dense.html

cryptographie basée sur les courbes elliptiques mais elle devient très pénalisante dès que la taille des entiers augmentent.
Notons n_{opt} la valeur optimale de n calculée à l'étape 4. Si on ne trouve pas de polynômes $M(X)$ satisfaisant les contraintes nécessaires, on peut reprendre la construction pour $n_{\text{opt}}+1$, $n_{\text{opt}}+2$ et ainsi de suite. Mais plus la valeur de n augmente et plus cela pénalise le processus de multiplication. Jusqu'à des entiers de taille 260 bits, on peut espérer trouver un bon candidat pour $n = n_{\text{opt}}$ (cf. figure 2.1, on ne peut pas avoir de "bon" candidat si la ligne verte est en dessous des croix violettes). Dans [DDV20], nous n'avions pas réussi à trouver de PMNS pour un entier de 521 bits pour $n = 9$. Ceci s'explique par le choix de ϕ et la contrainte que cela implique sur $\|M(X)\|_\infty$, la valeur est trop éloignée de ce que donne la borne de Minkowski (cf. figure 2.1). Pour $n = 10$, on est à la limite d'un cas favorable, ce qui explique la difficulté à trouver un "bon" candidat (cf. figure 2.2).
La situation dégénère pour des entiers de type RSA. Pour des entiers de 1024 à 1200 bits environ, il faut choisir $n = n_{\text{opt}} + 5$ (cf. figure 2.3) et pour des entiers de 2048 bits, il faut aller jusqu'à $n = n_{\text{opt}} + 11$ (cf. figure 2.4) ! Si de plus $\delta > 0$ et $\lambda > 4$, on obtiendra une contrainte encore plus forte à respecter sur $\|M(X)\|_\infty$, et ceci aboutira certainement à devoir utiliser une valeur de n trop grande.

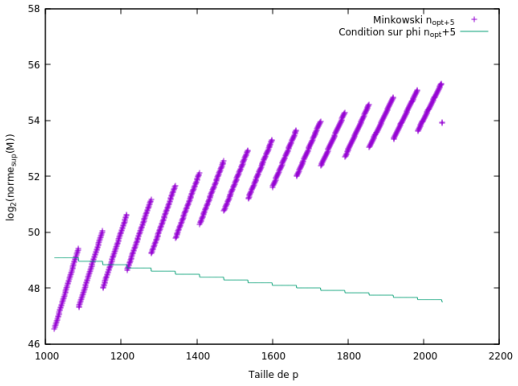


Figure 2.3 – Conditions à respecter sur $\|M(X)\|_\infty$ pour $\delta = 0$, $\lambda = 4$,$n = n_{\text{opt}} + 5$.

2.4.4 Efficacité en pratique des AMNS

Afin de démontrer l'intérêt pratique du système de représentation AMNS, nous avons mesuré le nombre de cycles nécessaires pour effectuer des multiplications modulaires pour des premiers p choisis au hasard. Chaque multiplication a été effectuée en utilisant le système de représentation AMNS, puis en utilisant les librairies OpenSSL et GnuMP. Concernant OpenSSL, la multiplication modulaire proposée par défaut n'utilise pas la

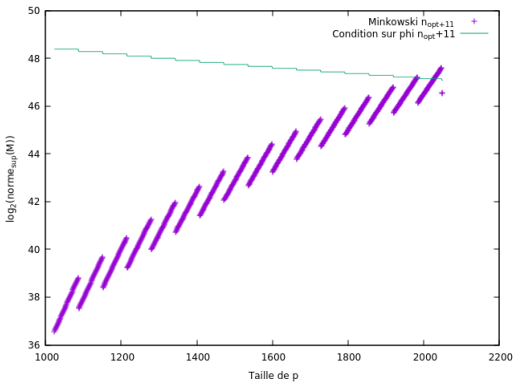


Figure 2.4 – Conditions à respecter sur $\|M(X)\|_\infty$ pour $\delta = 0$, $\lambda = 4$,$n = n_{\text{opt}} + 11$.

méthode de Montgomery (ligne `openssl` de la table 2.4). Nous avons donc recompilé une version de cette librairie afin de pouvoir effectuer une comparaison des AMNS avec la méthode de Montgomery classique (ligne `openssl_mont` de la table 2.4). Concernant GnuMP, outre les fonctions disponibles, il est possible d'utiliser aussi des fonctions dites de "bas niveau" afin d'optimiser les performances des opérations (ligne `gnu_mp_low` de la table 2.4). Parmi ces dernières, il existe une fonction non documentée permettant d'effectuer les multiplications en utilisant la méthode de Montgomery classique (ligne `gnu_mp_mont` de la table 2.4). La table 2.4 donne un aperçu des performances obtenues pour des multiplications modulaires sur des entiers de 256 bits. Les colonnes Q_1, Q_2 et Q_3 correspondent aux premier, deuxième et troisième quartiles. La représentation AMNS s'avère être une

	Q1	Q2	Q3
amns	182	184	186
gnu_mp_mont	180	199	224
openssl_mont	190	216	243
gnu_mp_low	318	325	348
gnu_mp	403	421	437
openssl	1089	1121	1198

Table 2.4 – Nombre de cycles pour la multiplication modulaire de deux entiers de 256 bits, gcc 9.3.0, Intel Core i7-10750H @ 2.6 GHz.

alternative pertinente face aux autres bibliothèques de calcul sur les grands entiers. Il faut cependant modérer cette conclusion. En effet tout dépend de la valeur de n. Pour un même entier p, tant que le nombre de symboles utilisés pour sa représentation AMNS n'excède pas $r + 1$ où r est le nombre de mots utilisés par OpenSSL ou GnuMP pour représenter p,

alors la représentation AMNS surpasse les autres librairies. Dans le cas contraire, les performances sont moins bonnes que la méthode de Montgomery classique. A titre d'exemple, la table 2.5 liste les résultats obtenus pour des entiers de 512 bits où, bien que $n\_opt = 9$, nous n'avons pu construire des AMNS que pour $n = 10$.

	Q1	Q2	Q3
amns	505	510	516
gnu_mp_mont	399	419	437
openssl_mont	377	395	430
gnu_mp_low	672	691	709
gnu_mp	738	758	794
openssl	2269	2331	2428

Table 2.5 – Nombre de cycles pour la multiplication modulaire de deux entiers de 512 bits, gcc 9.3.0, Intel Core i7-10750H @ 2.6 GHz.

2.4.5 Randomisation dans le système de représentation PMNS (redéfini)

Dans [BIP05a], un PMNS est défini comme étant un MNS pour lequel $E(X) = X^n + \alpha X + \beta$ est irréductible dans $\mathbb{Z}[X]$ avec α et β "petits". Comme remarqué page 91, ceci n'inclut pas totalement la classe des AMNS. Dans [Did+19], nous proposons une nouvelle définition des PMNS.

Définition 2.6

Un système de représentation PMNS est un système de représentation MNS défini par les paramètres (p, n, γ, ρ, E) pour lequel $\|E(X)\|_\infty$ est "petite".

Cette définition permet d'inclure les AMNS en tant que sous-classe des PMNS, le polynôme $E(X)$ n'étant plus forcément irréductible.

Nous montrons alors comment tirer partie de la nature redondante du sytème de représentation MNS en général, afin de proposer une possible protection contre les attaques de type DPA dans le contexte des PMNS et des courbes elliptiques. Il existe dans la littérature une multitude de références concernant les contremesures à appliquer pour résister aux attaques DPA dans le contexte de la cryptographie basée sur les courbes elliptiques. Parmi les recommandations classique à appliquer, on retrouve entre autres :

1. la randomisation des coordonnées du point de départ P lors du calcul de kP,
2. la randomisation des valeurs intermédiaires calculées par l'algorithme de multiplication scalaire de façon à garantir que plusieurs exécutions de cet algorithme avec les mêmes données en entrée ne produisent pas le même flux de données observables.

Je ne rentrerai pas dans ce document dans les détails de [Did+19] dans lequel nous montrons :

— comment modifier la procédure de conversion d'un entier vers un représentant du PMNS, afin d'appliquer la contre-mesure 1 ;
— comment modifier la procédure de multiplication et de réduction interne d'un PMNS, afin d'appliquer la contre-mesure 2.

Je reviendrai cependant dans le paragraphe 2.4.8 sur la procédure de multiplication décrite dans [Did+19] afin de mettre en évidence une propriété "indésirable" de cette dernière et d'en proposer une modification répondant mieux aux exigences de la contre-mesure 2.

L'idée générale que nous utilisons dans [Did+19] est de pouvoir associer à un même entier a un ensemble de $(2z+1)^n$ représentants dans le PMNS, z étant un paramètre de la procédure de conversion ou de multiplication. De façon très informelle, afin de changer la valeur d'une donnée d'entrée ou la valeur d'une variable intermédiaire, nous lui ajoutons un représentant de 0 dans le PMNS. Etant donné que pour la réduction Montgomery-like, on dispose d'un polynôme $M(X)$ qui est un représentant de 0 dans le PMNS, les autres représentants que nous utilisons sont de la forme $Z(X)M(X)$ où le polynôme $Z(X)$ est choisi aléatoirement parmi les polynômes de degré $n-1$ et de norme infinie inférieure ou égal à z. Ainsi $Z(X)$ est choisi de façon uniforme dans un ensemble de taille $(2z+1)^n$. Nous donnons alors les conditions sur ρ et ϕ en fonction de z permettant d'assurer que :

— la procédure de conversion renvoie pour un entier a un représentant choisi aléatoirement parmi les $(2z+1)^n$ représentants possibles,
— la procédure de multiplication renvoie pour 2 entiers a et b, un représentant choisi aléatoirement parmi les $(2z+1)^n$ représentants possibles de $ab \bmod p$.

Dans le contexte d'une multiplication scalaire sur une courbe elliptique (calcul de kP), on peut ainsi soit randomiser le point P lors de l'étape de conversion de ses coordonnées dans le PMNS, soit randomiser les multiplications effectuées par l'algorithme de multiplication scalaire, soit combiner ces deux approches. D'un point de vue théorique, l'ajout de randomisation dans le processus de conversion ou de multiplication implique un surplus de calcul de l'ordre de n^2 multiplications et additions d'entiers de k bits (pour $\phi = 2^k$) auquel il faudra ajouter aussi le coût de la génération aléatoire du polynôme $Z(X)$. Ce coût peut avoir un impact non négligeable sur les performances générales. A titre d'exemple, pour le calcul de kP, dans le cadre d'un AMNS, randomiser uniquement le point P augmente le temps d'exécution d'environ 3 à 4 %. Par contre, randomiser le point P ainsi que chaque multiplication modulaire, multiplie par un facteur 2.5 le temps d'exécution total.

2.4.6 Théorème d'existence pour les PMNS (nouvelle et ancienne définition)

Dans cette partie, je donne une nouvelle condition suffisante sur ρ afin de justifier l'existence des PMNS tels que définis dans [Did+19].

Pour deux polynômes $A(X)$ et $B(X)$ quelconques de degré au plus $n-1$, la majoration de $\|A(X)B(X) \bmod E(X)\|_\infty$ joue un rôle essentiel pour établir un critère d'existence d'un PMNS.

Proposition 2.4

Soient $A(X)$ et $B(X)$ deux polynômes quelconques de degré au plus $n-1$ à coefficients dans $\mathbb{R}$, et $E(X)$ un polynôme unitaire de degré n à coefficients dans $\mathbb{Z}$. Soit $\mathcal{E}$ la matrice à $n-1$ lignes et n colonnes dont la $i^{\text{ème}}$ ligne contient les coefficients du polynôme $X^{n+i-1} \bmod E(X)$, et $\mathcal{E}'$ la matrice définie par $\mathcal{E}'_{ij} = |\mathcal{E}_{ij}|$. On a

$$\|A(X)B(X) \bmod E(X)\|_\infty \leqslant w\|A(X)\|_\infty\|B(X)\|_\infty\,,$$

avec

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

Démonstration. Soit $C(X) = A(X)B(X) = c_0 + c_1X + \cdots + c_{n-1}X^{n-1} + c_nX^n + \cdots + c_{2n-2}X^{2n-2}$, et $D(X) = C(X) \bmod E(X)$. On a

$$\begin{aligned} D(X) \;=\; & c_0 + c_1X + \cdots + c_{n-1}X^{n-1} + \\ & c_n(X^n \bmod E(X)) + \cdots + c_{2n-2}(X^{2n-2} \bmod E(X)) \end{aligned}\,.$$

En d'autres termes,

$$(d_0,\ldots,d_{n-1}) = (c_0,\ldots,c_{n-1}) + (c_n,\ldots,c_{2n-2})\mathcal{E}\,.$$

Ainsi,

$$\forall i \in [0,n-1] \quad d_i = c_i + \sum_{j=1}^{n-1} c_{n+j-1}\mathcal{E}_{ji}\,.$$

Or

$$\forall i \in [0,n-1], \quad c_i \leqslant (i+1)\|A(X)\|_\infty\|B(X)\|_\infty\,,$$

et

$$\forall i \in [n,2n-2], \quad c_i \leqslant (2n-(i+1))\|A(X)\|_\infty\|B(X)\|_\infty\,.$$

Donc

$$\forall i \in [0, n-1], \quad d_i \leqslant ((i+1) + \sum_{j=1}^{n-1}(n-j)\mathcal{E}'_{ji})\|A(X)\|_\infty \|B(X)\|_\infty \,.$$

La quantité à droite de cette inégalité étant positive, on en déduit que $\|D(X)\|_\infty \leqslant w\|A(X)\|_\infty\|B(X)\|_\infty$, avec $w = \|(1, 2, \ldots, n) + (n-1, n-2, \ldots, 2, 1)\mathcal{E}'\|_\infty$. □

Quelques w particuliers

Pour $E(X) = X^n - \lambda$, $\lambda \in \mathbb{Z}$, on a

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

Donc, $w = 1 + (n-1)|\lambda|$.

Pour $E(X) = X^n + \alpha X + \beta$, $\alpha, \beta \in \mathbb{Z}$, $\alpha \neq 0$ et $\beta \neq 0$, on a

$$\mathcal{E}' = \begin{pmatrix} |\beta| & |\alpha| & 0 & \ldots & \ldots & 0 \\ 0 & |\beta| & |\alpha| & \ldots & \ldots & 0 \\ \vdots & & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \vdots \\ 0 & \ldots & \ldots & 0 & |\beta| & |\alpha| \end{pmatrix} .$$

Notons $\tilde{w}$ le vecteur dont les composantes sont

$$(1 + \sum_{j=1}^{n-1}(n-j)\mathcal{E}'_{j1}, 2 + \sum_{j=1}^{n-1}(n-j)\mathcal{E}'_{j2}, \ldots, n + \sum_{j=1}^{n-1}(n-j)\mathcal{E}'_{jn}) \,.$$

On a $\tilde{w}_1 = 1 + (n-1)\beta$, $\tilde{w}_i \leqslant \tilde{w}_2 = 2 + (n-1)|\alpha| + (n-2)|\beta|$ pour $2 \leqslant i < n$ et $\tilde{w}_n = n + |\alpha|$. Etant donné que $\beta \neq 0$ et $n \geqslant 2$, on a $\tilde{w}_2 \geqslant \tilde{w}_n$. Ainsi

$$w = \begin{cases} 1 + (n-1)|\beta| & \text{si } |\beta| > 1 + (n-1)|\alpha|, \\ 2 + (n-1)|\alpha| + (n-2)|\beta| & \text{sinon.} \end{cases}$$

Théorème 2.2

Soit $p \geqslant 3$, $\gamma \in \mathbb{Z}$, $E(X)$ un polynôme unitaire tel que $E(\gamma) \equiv 0 \pmod{p}$ et $n = \deg(E)$. S'il existe un polynôme $M(X) \in \mathbb{Z}[X]$ tel que $\text{pgcd}(E(X), M(X)) = 1$ et $M(\gamma) \equiv 0 \bmod p$, alors (p, n, γ, ρ, E) est un PMNS si :

$$\rho > \frac{1}{2} w \|M\|_\infty ,$$

où $w = \|(1, 2, \ldots, n) + (n-1, n-2, \ldots, 2, 1)\mathcal{E}'\|_\infty$, la matrice $\mathcal{E}$ étant définie dans la proposition 2.4.

Démonstration. Considérons la matrice B dont chaque ligne contient les coefficients de $X^i M(X) \bmod E(X)$. Pour tout vecteur $t = (t_0, \ldots, t_{n-1}) \in \mathbb{Z}^n$, les composantes de $t.B$ sont les coefficients de $(\sum_{i=0}^{n-1} t_i X^i) \times M(X) \bmod E(X)$. Les lignes de B sont linéairement indépendantes dans $\mathbb{Z}^n$. En effet, soit $t \in \mathbb{Z}^n$ tel que $t.B = (0, \ldots, 0)$, on en déduit que $(\sum_{i=0}^{n-1} t_i X^i) \times M(X) \equiv 0 \bmod E(X)$. Etant donné que $\text{pgcd}(E(X), M(X)) = 1$, on a alors $t = (0, \ldots, 0)$. Ainsi B est la base d'un certain réseau $\mathfrak{L}$.
Notons B_i la $i^{\text{ème}}$ ligne de B et considérons

$$H' = \{t \in \mathbb{R}^n \mid t = \sum_{i=0}^{n-1} \mu_i B_i \text{ et } -\frac{1}{2} \leqslant \mu_i < \frac{1}{2}\} .$$

H' est une région fondamentale du réseau $\mathfrak{L}$ en tant que translaté de la région fondamentale canonique définie par

$$\{t \in \mathbb{R}^n \mid t = \sum_{i=0}^{n-1} \mu_i B_i \text{ et } 0 \leqslant \mu_i < 1\} .$$

La théorie des réseaux stipule que

$$\forall\, v \in \mathbb{R}^n, \exists\, !(t, y) \in H' \times \mathfrak{L} \text{ tel que } v = t + y .$$

Soit a un élément quelconque $\mathbb{Z}/p\mathbb{Z}$, posons $A = (a, 0, \ldots, 0)$. Il existe donc un couple unique $(t, y) \in H' \times \mathfrak{L}$ tel que $A = t + y$. Posons $\tilde{A} = A - y$, alors $\|\tilde{A}\|_\infty = \|t\|_\infty$. Or il existe $(\mu_0, \ldots, \mu_{n-1})$ tel que $t = \mu.B$. Ainsi les composantes de t correspondent aux coefficients de $\mu(X)M(X) \bmod E(X)$. D'après la proposition 2.4,

$$\|\tilde{A}\|_\infty = \|\mu(X)M(X) \bmod E(X)\|_\infty \leqslant w \|\mu(X)\|_\infty \|M(X)\|_\infty .$$

Etant donné que $\|\mu(X)\|_\infty \leqslant \frac{1}{2}$, on en déduit que

$$\|\tilde{A}\|_\infty \leqslant \frac{1}{2} w \|M(X)\|_\infty .$$

Soit $\tilde{A}(x)$ le polynôme dont les coefficients sont donnés par le vecteur $\tilde{A}$. On a $\tilde{A}(X) = A(X) - Y(X)$ avec $Y(X) = \sum_{i=0}^{n-1} y_i X^i = Q(X)M(X) \bmod E(X)$ pour un certain polynôme $Q(X)$ (étant donné que $y \in \mathfrak{L}$). Or $M(\gamma) \equiv 0 \pmod p$, et $E(\gamma) \equiv 0 \pmod p$, donc $Y(\gamma) \equiv 0 \pmod p$. On en déduit donc que $\tilde{A}(\gamma) \equiv a \pmod p$.
En résumé, pour tout $a \in \mathbb{Z}/p\mathbb{Z}$, il existe un polynôme $\tilde{A} \in \mathbb{Z}[X]$ tel que $\tilde{A}(\gamma) \equiv a \pmod p$ et $\|\tilde{A}(X)\|_\infty \leqslant \frac{1}{2} w \|M(X)\|_\infty$. Donc si $\rho > \frac{1}{2} w \|M(X)\|_\infty$, le tuple (p, n, γ, ρ, E) définit bien un PMNS. □

Ce théorème permet d'obtenir une borne plus fine pour l'existence des PMNS tels qu'ils sont définis dans l'article fondateur [BIP05a].

Corollaire 2.5

Soient $p, n > 1$ deux entiers et $E(X) = X^n + \alpha X + \beta$ un polynôme irréductible dans $\mathbb{Z}[X]$ avec $\alpha,\ \beta \in \mathbb{Z}$. Soit γ tel que $E(\gamma) \equiv 0 \pmod p$, il existe un PMNS de paramètre (p, n, γ, ρ, E) si

- $\rho > \frac{1}{2}(1 + (n-1)|\beta|)p^{1/n}$ dans le cas où $|\beta| > 1 + (n-1)|\alpha|$,
- $\rho > \frac{1}{2}(2 + (n-1)|\alpha| + (n-2)|\beta|)p^{1/n}$, sinon.

Démonstration. Considérons le réseau $\mathfrak{L}$ constitué des polynômes de degré au plus $n-1$ qui s'annulent en γ modulo p (cf. définition 2.2, page 89). Ce réseau étant de volume égal à p, il existe d'après le théorème de Minkowski, un polynôme $M \in \mathfrak{L}$ tel que $\|M(X)\|_\infty \leqslant p^{1/n}$. Etant donné que $E(X)$ est irréductible, on en déduit que $\operatorname{pgcd}(E(X), M(X)) = 1$. Ainsi, d'après le théorème 2.2 le tuple (p, n, γ, ρ, E) définit un PMNS si

$$\rho > \frac{1}{2} w p^{1/n} ,$$

avec (cf. page 110)

$$w = \begin{cases} 1 + (n-1)|\beta| & \text{si } |\beta| > 1 + (n-1)|\alpha|, \\ 2 + (n-1)|\alpha| + (n-2)|\beta| & \text{sinon.} \end{cases} .$$

□

2.4.7 Montgomery-like pour les PMNS

Dans cette partie, je donne les conditions suffisantes sur ρ et ϕ afin de pouvoir utiliser la réduction interne Montgomery-like pour un PMNS quelconque défini par le tuple (p, n, γ, ρ, E) (cf. définition 2.6).

Tout comme pour la méthode Montgomery-like proposée dans le cadre des AMNS dans [NP08], on suppose l'existence de deux polynômes $M(X)$ et $M'(X)$ tels que :

— $M(X)$ est un représentant de l'entier 0 dans le PMNS,

— $M'(X)$ est un polynôme de degré au plus $n-1$ tel que $M(X)M'(X) = -1 \bmod (E(X), \phi)$.

Soient $A(X)$ et $B(X)$ deux éléments du PMNS . Soit $C(X) = A(X)B(X) \bmod E(X)$. D'après la proposition 2.4, page 108, on a

$$\|A(X)B(X) \bmod E(X)\|_\infty \leqslant w\|A(X)\|_\infty\|B(X)\|_\infty < w\rho^2\,,$$

avec

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

la matrice $\mathcal{E}'$ étant définie dans la proposition 2.4.

Algorithm 21 Réduction interne Montgomery-like pour un PMNS (RedCoeff)

Require: $C(X)$ tel que $\|C(X)\| < w\rho^2$, et $M'(X) = -M^{-1}(X) \bmod (E(X), \phi)$

Ensure: $R(X)$ est un représentant dans le PMNS de l'entier $C(\gamma)\phi^{-1} \pmod p$

1: $Q(X) \leftarrow C(X) \times M'(X) \bmod (E(X), \phi)$

2: $T(X) \leftarrow Q(X) \times M(X) \bmod E(X)$

3: $R(X) \leftarrow (C(X) + T(X))/\phi$

4: return $R(X)$

Proposition 2.6

Soit $C(X)$ un polynôme de degré au plus $n-1$ de $\mathbb{Z}[X]$. L'algorithme 21 renvoie un représentant $R(X)$ de $C(\gamma)\phi^{-1}$ dans le PMNS si :

$$\|C(X)\| < w\rho^2, \quad \rho \geqslant 2w\|M(X)\|_\infty, \quad \text{et } \phi \geqslant 2w\rho\,.$$

Démonstration. Le polynôme $Q(X)$ (ligne 1 de l'algorithme) vérifie $\|Q(X)\|_\infty < \phi$. On a

$$\|C(X) + Q(X)M(X) \bmod E(X)\|_\infty \leqslant \|C(X)\|_\infty + \|Q(X)M(X) \bmod E(X)\|_\infty\,.$$

Si

$$\|C(X)\|_\infty < w\rho^2\,,$$

étant donné que

$$\|Q(X)M(X) \bmod E(X)\|_\infty < w\phi\|M\|_\infty\,,$$

la méthode de réduction renvoie un résultat dans le PMNS si

$$w(\rho^2 + \phi\|M\|_\infty) \leqslant \rho\phi\,,$$

c'est à dire

$$\rho \geqslant \frac{w(\rho^2 + \phi\|M\|_\infty)}{\phi} = \frac{w\rho^2}{\phi} + w\|M\|_\infty\,.$$

Il suffit donc que

$$\frac{w\rho^2}{\phi} \leqslant \frac{\rho}{2} \quad \text{et} \quad w\|M\|_\infty \leqslant \frac{\rho}{2}\,,$$

pour que l'inégalité précédente soit satisfaite. On obtient ainsi :

$$\phi \geqslant 2w\rho \quad \text{et} \quad \rho \geqslant 2w\|M\|_\infty\,.$$

□

Lorsque $E(X)$ est irréductible, dans le théorème 2.2, on peut choisir pour $M(X)$ n'importe quel polynôme de degré au plus $n-1$ qui s'annule en γ modulo p. Pour minimiser la valeur de ρ, on a donc intérêt à choisir un vecteur court dans la base réduite du réseau engendré par les polynômes de degré au plus $n-1$ s'annulant en γ. C'est ce choix qui est effectué lorsque l'on utilise la méthode Montgomery-like pour la réduction interne. Ainsi, par rapport à la borne théorique donnée par le théorème 2.2, l'utilisation de la méthode de Montgomery-like contraint à choisir un paramètre ρ qui est environ 4 fois plus "gros".

Tout comme pour l'AMNS, il est possible avec les PMNS d'effectuer δ additions successives suivies d'une multiplication en ne faisant appel qu'une seule fois au processus de réduction interne. Plus précisément :

Proposition 2.7

Soient $A(X)$ et $B(X)$ deux éléments d'un PMNS, obtenus après au plus δ additions (sans réduction interne). L'algorithme 21 appliqué à $C(X) = A(X)B(X) \bmod E(X)$

renvoie un résultat $R(X)$ dans le PMNS si :

$$\rho \geqslant 2w|||M\|_\infty \quad \text{et} \quad \phi \geqslant 2w\rho(\delta+1)^2\,.$$

Démonstration. On a

$$\|A(X)\|_\infty < (\delta+1)\rho \quad \text{et} \quad \|B(X)\|_\infty < (\delta+1)\rho\,.$$

En suivant le déroulement de la preuve de la proposition 2.6, la méthode de réduction renvoie un résultat dans le PMNS si

$$w(\rho^2(\delta+1)^2 + \phi\|M\|_\infty) \leqslant \rho\phi\,,$$

c'est à dire

$$\rho \geqslant \frac{w(\rho^2(\delta+1)^2 + \phi\|M\|_\infty)}{\phi} = \frac{w\rho^2(\delta+1)^2}{\phi} + w\|M\|_\infty\,.$$

Il suffit donc que

$$\frac{w\rho^2(\delta+1)^2}{\phi} \leqslant \frac{\rho}{2} \quad \text{et} \quad w\|M\|_\infty \leqslant \frac{\rho}{2}\,,$$

pour que l'inégalité précédente soit satisfaite. On obtient ainsi :

$$\phi \geqslant 2w\rho(\delta+1)^2 \quad \text{et} \quad \rho \geqslant 2w\|M\|_\infty\,.$$

□

Afin de satisfaire l'inégalité précédente, on peut aussi considérer un paramètre t tel que :

$$\frac{w\rho^2(\delta+1)^2}{\phi} \leqslant \frac{\rho}{t} \quad \text{et} \quad w\|M\|_\infty \leqslant \frac{(t-1)\rho}{t}\,.$$

On obtient alors

$$\phi \geqslant tw\rho(\delta+1)^2 \quad \text{et} \quad \rho \geqslant \frac{t}{t-1}w\|M\|_\infty\,.$$

L'avantage est d'obtenir un paramètre ρ proche de $w\|M\|_\infty$ au détriment de ϕ qui va grossir en fonction de t. Or il ne faut pas oublier que sur une architecture k bits, on souhaite prendre $\phi = 2^k$ pour des raisons de performance.

2.4.8 Nouvelle procédure de multiplication randomisée pour un PMNS

L'algorithme de multiplication randomisée que nous avons proposé dans [Did+19] (cf. Algorithme 22) possède un inconvénient. Il a été développé afin d'assurer que si l'on exécute plusieurs fois l'algorithme avec les mêmes données d'entrée, les calculs intermédiaires fournissent des résultats différents. Cependant, le masquage du polynôme $B(X)$ par $J(X)$ (ligne 4) n'a aucune influence sur le calcul effectué à la ligne 6 de l'algorithme comme je le démontre dans la proposition suivante :

Proposition 2.8

Soient $C(X) = \sum_{i=0}^{n-1} c_i x^i$, $Z(X) = \sum_{i=0}^{n-1} z_i x^i$ ($z_i \geqslant 0$) et $Q(X) = C(X)M'(X) \bmod (E(X), \phi) = \sum_{i=0}^{n-1} q_i x^i$. Si $\forall\ i \in [0, n-1]$, $z_i \leqslant q_i$, alors **RedCoeff**$(C + ZM)$= **RedCoeff**(C) (cf. Algorithme 21, page 112).

Démonstration. Posons $\tilde{C}(X) = C(X) + Z(X)M(X)$. La première étape de la réduction interne Montgomery-like consiste à calculer le polynôme $\tilde{Q}(X) = \tilde{C}(X)M'(X) \bmod (E(X), \phi)$. Ainsi,

$$\tilde{Q}(X) = (C(X)M'(X) \bmod (E(X), \phi) + Z(X)M(X)M'(X) \bmod (E(X), \phi)) \bmod \phi\,.$$

Etant donné que $M(X)M'(X) = -1 \bmod (E(X), \phi)$, on en déduit que $\tilde{Q}(X) = Q(X) - Z(X) \bmod \phi$. Par définition de $Q(X)$, tous ses coefficients sont strictement inférieurs à ϕ. Par hypothèse, $0 \leqslant z_i \leqslant q_i$, ainsi la réduction modulo ϕ n'a pas d'impact sur le calcul de $Q(X) - Z(X) \bmod \phi$. On en déduit donc que :

$$\tilde{Q}(X) = Q(X) - Z(X)\,.$$

La dernière étape de la réduction interne consiste à calculer $\tilde{C}(X) + \tilde{Q}(X)M(X) \bmod E(X)$. On a donc,

$$\tilde{C}(X) + \tilde{Q}(X)M(X) = C(X) + Z(X)M(X) + Q(X)M(X) - Z(X)M(X)\,.$$

On en déduit donc que :

$$\tilde{C}(X) + \tilde{Q}(X)M(X) = C(X) + Q(X)M(X) \bmod E(X)\,,$$

ce qui correspond exactement à la dernière étape de la réduction interne du polynôme $C(X)$. □

Dans l'algorithme 23, nous proposons une modification (ligne 6) afin que les conditions

Algorithm 22 Multiplication de Montgomery randomisée [Did+19]

Require: $\mathcal{B} = (p, n, \gamma, \rho, E)$ et $A(X), B(X) \in \mathcal{B}$
Ensure: $R(\gamma) = A(\gamma)B(\gamma)\phi^{-1} \mod p$
Données : ϕ, $z \in \mathbb{N}$, $M(X) \in \mathcal{B}$, tel que : $M(\gamma) \equiv 0 \pmod p$ et pgcd(ϕ, résultant($E(X)$, $M(X)$)) $= 1$, $M'(X) = -M(X)^{-1} \mod (E(X), \phi)$.

1: $Z(X) \leftarrow$ **randPoly**(z)
2: $J(X) \leftarrow Z(X) \times M(X) \mod E(X)$
3: $B'(X) \leftarrow B(X) + J(X)$
4: $C'(X) \leftarrow (A(X) \times B'(X)) \mod E(X)$
5: $R'(X) \leftarrow$ **RedCoeff**$(C'(X)) + 2 \times J(X)$
6: **return** $R'(X)$

de la proposition 2.8 ne soient pas vérifiées.

Algorithm 23 Nouvelle multiplication randomisée de Montgomery

Require: $\mathcal{B} = (p, n, \gamma, \rho, E)$ et $A(X), B(X) \in \mathcal{B}$
Ensure: $R(\gamma) = A(\gamma)B(\gamma)\phi^{-1} \mod p$
Données : ϕ, $z \in \mathbb{N}$, $M(X) \in \mathcal{B}$, tel que : $M(\gamma) \equiv 0 \pmod p$ et pgcd(ϕ, résultant($E(X)$, $M(X)$)) $= 1$, $M'(X) = -M(X)^{-1} \mod (E(X), \phi)$.

1:
2: $Z(X) \leftarrow$ **randPoly**(z)
3: $J(X) \leftarrow Z(X) \times M(X) \mod E(X)$
4: $B'(X) \leftarrow B(X) + J(X)$
5: $C(X) \leftarrow (A(X) \times B'(X)) \mod E(X)$
6: $Q(X) \leftarrow C(X) \times M'(X) \bmod (E(X), \phi)$
7: $T(X) \leftarrow (Q(X) + \phi Z(X)) \times M(X) \bmod E(X)$
8: $S(X) \leftarrow (C(X) + T(X))/\phi$
9: $R(X) \leftarrow S(X) + J(X)$
10: **return** $R(X)$

Théorème 2.3

Soit $\mathcal{B} = (p, n, \gamma, \rho, E)$ un système de représentation PMNS. Soit $z \in \mathbb{N}$ un paramètre définissant la valeur absolue maximale des coefficients d'un polynôme engendré par la procédure de tirage aléatoire `randPoly`. Soit $M(X) \in \mathcal{B}$ un polynôme tel que : $M(\gamma) \equiv 0 \pmod p$ et pgcd(ϕ, résultant($E(X)$, $M(X)$)) $= 1$, où $\phi = 2^k$. Posons $m = \|M\|_\infty$ et soient $A(X)$ et $B(X)$ deux éléments de $\mathcal{B}$. L'algorithme 23 renvoie un représentant de $A(\gamma)B(\gamma)\phi^{-1} \pmod p$ choisi de façon uniforme en fonction de z parmi $(2z+1)^n$ représentants possibles si :

$$\rho \geqslant 2wm(2z+1) \quad \text{et} \quad \phi \geqslant \max(2w\rho z, 2w(\rho + wmz)),$$

avec
$$w = \|(1, 2, \ldots, n) + (n-1, n-2, \ldots, 2, 1)\mathcal{E}'\|_\infty ,$$
la matrice $\mathcal{E}'$ étant définie dans la proposition 2.4, page 108.

Démonstration. Etant donné que $M(\gamma) \equiv 0 \pmod p$, on a $J(\gamma) \equiv 0 \pmod p$, $T(\gamma) \equiv 0 \pmod p$ et $B'(\gamma) \equiv B(\gamma) \pmod p$. On en déduit :

$$R(\gamma) \equiv C(\gamma)\phi^{-1} \pmod p \equiv A(\gamma)B(\gamma)\phi^{-1} \pmod p .$$

On a :

$$\begin{aligned} R(X) &= \frac{C(X) + T(X)}{\phi} + J(X) \\ &= \frac{(A(X)B'(X) + Q(X)M(X) + \phi Z(X)M(X)) \bmod E(X)}{\phi} + Z(X)M(X) \bmod E(X) \\ &= \frac{(A(X)B'(X) + Q(X)M(X)) \bmod E(X)}{\phi} + 2J(X) . \end{aligned}$$

Ainsi,

$$\|R(X)\|_\infty \leqslant \frac{1}{\phi}(\|A(X)B'(X) \bmod E(X)\|_\infty + \|Q(X)M(X) \bmod E(X)\|_\infty) + 2\|J(X)\|_\infty .$$

De plus :

- $\|B'(X) \bmod E(X)\|_\infty < \rho + wzm$,
- $\|A(X)B'(X) \bmod E(X)\|_\infty < w\rho(\rho + wzm)$,
- $\|Q(X)M(X) \bmod E(X)\|_\infty < w\phi m$,
- $\|J(X)\|_\infty \leqslant wzm$.

Ainsi,
$$\|R(X)\|_\infty < \frac{w\rho(\rho + wzm)}{\phi} + wm(1 + 2z) .$$

Donc si,
$$\frac{w\rho(\rho + wzm)}{\phi} \leqslant \frac{\rho}{2} \quad \text{et} \quad wm(1 + 2z) \leqslant \frac{\rho}{2} ,$$
on aura bien $\|R(X)\|_\infty < \rho$. Ces deux conditions sont équivalentes à

$$\rho \geqslant 2wm(2z + 1) \quad \text{et} \quad \phi \geqslant 2w(\rho + wmz) .$$

Supposons à présent qu'il existe deux polynômes distincts $Z(X)$ et $\tilde{Z}(X)$ tels que $R(X) = \tilde{R}(X)$. On a :

$$\begin{aligned} R(X) &= \frac{(A(X)B(X) + A(X)J(X) + Q(X)M(X) + 2\phi Z(X)M(X)) \bmod E(X)}{\phi} \\ &= \frac{A(X)B(X) \bmod E(X) + M(X)(A(X)Z(X) + Q(X) + 2\phi Z(X)) \bmod E(X)}{\phi}. \end{aligned}$$

et

$$\begin{aligned} \tilde{R}(X) &= \frac{(A(X)B(X) + A(X)\tilde{J}(X) + \tilde{Q}(X)M(X) + 2\phi \tilde{Z}(X)M(X)) \bmod E(X)}{\phi} \\ &= \frac{A(X)B(X) \bmod E(X) + M(X)(A(X)\tilde{Z}(X) + \tilde{Q}(X) + 2\phi \tilde{Z}(X)) \bmod E(X)}{\phi}. \end{aligned}$$

Etant donné que le pgcd de $M(X)$ et de $E(X)$ vaut 1 dans $\mathbb{Q}[X]$, l'égalité $R(X) = \tilde{R}(X)$ entraîne :

$$A(X)Z(X) + Q(X) + 2\phi Z(X) = A(X)\tilde{Z}(X) + \tilde{Q}(X) + 2\phi \tilde{Z}(X) \bmod E(X).$$

Ainsi,

$$Q(X) - \tilde{Q}(X) = A(X)(\tilde{Z}(X) - Z(X)) \bmod E(X) + 2\phi(\tilde{Z}(X) - Z(X)).$$

Pout tout $i \in [0, n-1]$, on a $|q_i - \tilde{q}_i| < \phi$. D'autre part, on a $2\phi|\tilde{z}_i - z_i| \geqslant 2\phi$ puisque $Z(X) \neq \tilde{Z}(X)$. Enfin,

$$\|A(X)(\tilde{Z}(X) - Z(X)) \bmod E(X)\|_\infty \leqslant 2w\rho z.$$

Etant donné que par hyptothèse, $\phi \geqslant 2w\rho z$, on en déduit que la valeur absolue de tous les coefficients du polynôme $A(X)(\tilde{Z}(X) - Z(X)) \bmod E(X) + 2\phi(\tilde{Z}(X) - Z(X))$ est au moins égal à ϕ, ce qui est impossible. On a donc $Z(X) = \tilde{Z}(X)$. □

2.4.9 Quelques mots sur γ, LLL , $E(X)$ et la réduction interne

2.4.9.1 A propos de γ

Dans le système de représentation AMNS, la valeur γ est une racine de $X^n - \lambda$ modulo p. Cette valeur n'existe que si λ est un résidu $n^{\text{ième}}$ modulo p. Dans le cas où p est premier, un théorème dû à Gauss permet de caractériser les résidus $n^{\text{ième}}$ modulo p.

Théorème 2.4 (Gauss)**.** *Soit p un nombre premier et $n \geqslant 2$ un entier. Soit $b = pgcd(p-1, n)$, alors :*

1. *Pour tout entier* $a \in (\mathbb{Z}/p\mathbb{Z})^\star$, *a est un résidu* $n^{\text{ième}}$ *modulo p si et seulement si* $a^{\frac{p-1}{b}} = 1 \bmod p$,
2. *il y a exactement* $\frac{p-1}{b}$ *résidus* $n^{\text{ième}}$ *modulo p dans* $(\mathbb{Z}/p\mathbb{Z})^\star$,
3. *si* $a \in (\mathbb{Z}/p\mathbb{Z})^\star$ *est un résidu* $n^{\text{ième}}$ *modulo p, il existe b éléments* $x \in (\mathbb{Z}/p\mathbb{Z})^\star$ *tels que* $x^n = a \bmod p$.

On peut donc optimiser le code de la fonction `build_amns_candidates_for_n`[5] afin de filtrer les valeurs λ pour lesquelles on va chercher à extraire les racines $n^{\text{ième}}$. En effet, cette extraction est effectuée en utilisant la méthode `nth_root` de la classe des corps finis de la librairie SageMath. Or cette méthode se base sur un algorithme proposé dans [Joh99] qui ramène ce calcul à un problème de résolution de logarithme discret dans un certain corps fini. Selon la taille de ce corps, le calcul peut être assez long et un timeout a été placé dans le code afin de ne pas pénaliser le processus total de génération. Ce timeout peut éventuellement éliminer un candidat λ pour lequel l'algorithme d'extraction n'a pas pu terminer dans le temps imparti.

Concernant l'extraction de la racine $n^{\text{ième}}$, on peut assez facilement optimiser ce calcul pour les tailles utilisées en cryptographies, grâce au résultat suivant [Zeu19].

Proposition 2.9. *Soit p un nombre premier,* $n \geqslant 2$, λ *un résidu* $n^{\text{ième}}$ *modulo p et* $b = pgcd(p-1, n)$. *Soit k le plus grand entier tel que :*
— $\lambda^{\frac{p-1}{b^k}} = 1 \bmod p$,
— $pgcd(\frac{p-1}{b^k}, n) = 1$,
alors λ^u *est une racine* $n^{\text{ième}}$ *de* λ *où u est l'inverse de n modulo* $\frac{p-1}{b^k}$.

Pour un premier p de 256 bits, on choisira $n = 5$ pour construire un AMNS. Ainsi $b = 1$ ou $b = 5$:
— Si $b = 1$, 5 est inversible modulo $p-1$ et tout entier $\lambda \neq 0$ admet pour racine $5^{\text{ième}}$, λ^u avec u l'inverse de 5 modulo $p-1$,
— Si $b = 5$, et si 25 ne divise pas p, alors d'après la proposition précédente, λ^u est une racine $5^{\text{ième}}$ de λ où u est l'inverse de 5 modulo $\frac{p-1}{5}$,
— Si $b = 5$, et $\text{pgcd}(\frac{p-1}{5}, 5) = 5$, soit k le plus grand entier tel que $\text{pgcd}(\frac{p-1}{5^k}, 5) = 1$, il faut chercher λ parmi les $5^{\text{ième}}$ résidus modulo p qui satisfont $\lambda^{\frac{p-1}{5^k}} = 1 \bmod p$ et appliquer la proposition précédente.

Pour un premier p de 521 bits (taille du NIST), on choisira $n = 10$ pour construire un AMNS. Extraire une racine $10^{\text{ième}}$ peut se faire en commençant par extraire les racines

5. ligne 167 de https://github.com/arithPMNS/generalisation_amns/blob/master/amns_generator/amns_generator.sage

carrées de λ (ce qui est immédiat dans le cas $p \equiv 3 \mod 4$ et peut être effectué de façon efficace avec l'algorithme de Tonelli–Shanks dans le cas $p \equiv 1 \mod 4$). Il suffit alors d'extraire une racine $5^{\text{ième}}$, en utilisant ce qui a été décrit précédemment.

Pour un module $m = pq$ de 2048 bits (RSA), on procèdera au calcul de la racine $n^{\text{ième}}$ modulo p et modulo q et on utilisera le théorème des restes chinois. Pour cette taille, on choisira $n = 37$ (cf. figure 2.3, page 104). Ce cas se traite comme le cas $n = 5$.

Une fois que l'on dispose d'une racine, on peut facilement trouver les $b - 1$ autres racines (pour $b > 1$). En effet soit μ un non-résidu $n^{\text{ième}}$ modulo p, i.e :

$$\mu^{\frac{p-1}{b}} \neq 1 \bmod p\,,$$

étant donné que $b = \text{pgcd}(n, p-1)$, il existe un entier s tel que $n = bs$, ainsi

$$(\mu^{\frac{p-1}{b}})^n = (\mu^{p-1})^s = 1 \bmod p\,.$$

On en déduit que $\eta = \mu^{\frac{p-1}{b}}$ est une racine $n^{\text{ième}}$ de l'unité modulo p. Si z est une racine $n^{\text{ième}}$ de λ, alors $r\eta^i$ est aussi une racine $n^{\text{ième}}$ de λ pour $i = 1, \ldots, b-1$. Pour $i \neq j$, on a bien $\eta^i \neq \eta^j \bmod p$. En effet supposons $i > j$ et $\eta^i = \eta^j \bmod p$, on a alors

$$\frac{p-1}{b}(i-j) \equiv 0 \mod (p-1)\,.$$

Or $i - j < b$, donc $i = j$.

Est-il facile de trouver un non-résidu $n^{\text{ième}}$ modulo p ? D'après le théorème de Gauss, il y en a $\frac{(b-1)(p-1)}{b}$ dans $(\mathbb{Z}/p\mathbb{Z})^\star$, soit une probabilité de $\frac{b-1}{b}$ d'en choisir un aléatoirement.

Parmi les racines γ de $E(X)$, on ne peut pas choisir n'importe laquelle. Rappelons que pour représenter les éléments de $\mathbb{Z}/p\mathbb{Z}$, il faut que

$$\sum_{i=0}^{n-1} a_i\gamma^i \in [-\frac{p-1}{2}, \frac{p-1}{2}], \qquad a_i \in]-\rho, \rho[\,.$$

On a

$$\sum_{i=0}^{n-1} a_i\gamma^i < \rho\frac{\gamma^n - 1}{\gamma - 1}\,.$$

De plus, on sait que pour pouvoir représenter tous les entiers de $\mathbb{Z}/p\mathbb{Z}$, il faut que $\rho \geqslant \frac{p^{\frac{1}{n}}+1}{2}$.

Posons $\rho = \frac{p^{\frac{1}{n}}+1}{2} + \varepsilon$. Si

$$\gamma < \frac{(p-1)^{\frac{1}{n}}}{(p^{\frac{1}{n}}+1+2\varepsilon)^{\frac{1}{n}}},$$

alors,

$$\frac{\gamma^n - 1}{\gamma - 1} < \gamma^n < \frac{(p-1)}{(p^{\frac{1}{n}}+1+2\varepsilon)}.$$

Donc $\rho\frac{\gamma^n-1}{\gamma-1} < \frac{p-1}{2}$ (i.e, on ne peut pas représenter tous les entiers). Il faut donc que $\gamma \geqslant \frac{(p-1)^{\frac{1}{n}}}{(p^{\frac{1}{n}}+1+2\varepsilon)^{\frac{1}{n}}}$, par exemple on pourra choisir γ tel que

$$\gamma \geqslant p^{\frac{n-1}{n^2}}.$$

2.4.9.2 A propos de LLL

Lorsque $E(X) = X^n - \lambda$ et que $\mathrm{pgcd}(n, p-1) > 1$, nous avons mentionné dans [DDV20, Corollaire 4] que l'on pouvait toujours choisir $\lambda = 1$ afin d'assurer l'existence d'une racine γ. Il s'agit malheureusement d'un choix non judicieux. En effet :

— si n est impair, $X^n - 1 = (X-1)(X^{n-1} + \cdots + 1)$,
— si n est pair ($n = 2k$), $X^n - 1 = (X^k+1)(X^k-1)$.

Dans les 2 cas, $\gamma = 1$ est une racine de $X^n - 1$ mais elle ne peut pas être choisie (cf. paragraphe précédent). Ainsi, dans le cas n impair, γ est aussi une racine de $X^{n-1} + \cdots + 1$. En observant le code source de LLL disponible dans Sagemath, on s'aperçoit que pour construire la base réduite $B = (b_0, \ldots, b_{n-1})$ des polynômes de degré au plus $n-1$ s'annulant en γ, LLL utilise le polynôme $X^{n-1} + \cdots + 1$ (en effet il est de norme courte). Or le volume du réseau initial étant égal à p, le déterminant de la matrice B doit être égal à p. De l'inégalité de Hadamard en norme euclidienne, on peut déduire que pour la norme infinie, on doit avoir :

$$|\det(B)| \leqslant n^{n/2} \prod_{i=0}^{n-1} \|b_i\|_\infty.$$

Dans le cas n impair, on a $\|b_0\|_\infty = 1$ et l'inégalité devient :

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

Pour les $n-1$ vecteurs restants, LLL renvoie des éléments dont la norme infinie est grosso modo de même amplitude et donc , du fait de l'inégalité précédente, pour chaque b_i on a

plus ou moins (cf. Exemple ci-après) :

$$\|b_i\|_\infty \approx p^{\frac{1}{n-1}} .$$

Etant donné que dans le cas λ impair, on effectue des combinaisons linéaires à coefficients dans $\{0,1\}$ des éléments de la base pour trouver un bon candidat $M(X)$ (cf. page 100), ces combinaisons vont aboutir sur des polynômes dont la norme infinie est plus grande que $p^{\frac{1}{n}}$ et le processus de création de l'AMNS va échouer.
Dans le cas n pair, la situation est encore plus défavorable. En effet, γ est soit une racine de $X^{n/2}+1$, soit une racine de $X^{n/2}-1$ et LLL va au moins utiliser les polynômes $X^{n/2}+1$ (resp. $X^{n/2}-1$) et $X^{n/2+1}+X$ (resp. $X^{n/2+1}-X$) afin de construire sa base réduite. Le reste des éléments de la base sera au moins de norme infinie de l'ordre de $p^{\frac{1}{n-2}}$ (cf. Exemple ci-après).

Exemple. *Soit*

$p = 310676533131899150725766439265066894964723343390951162713165895469306003877 41$

(premier de 256 bits). Pour $n = 5$, on a pgcd$(p-1, n)$=5 et

$\gamma = 23778754090254715608711494035379691072332159340055163065131523238455428054288$

est une racine $5^{\text{ième}}$ modulo p. Le réseau renvoyé par LLL est le suivant :

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 4478994071079223566 & -8424929086432683102 & -2697845918095368817 & 6544917095126594593 & 98863838322233762 \\ 6544917095126594593 & 98863838322233762 & 4478994071079223566 & -8424929086432683102 & -2697845918095368817 \\ 8424929086432683102 & 2697845918095368817 & -6544917095126594593 & -98863838322233762 & -4478994071079223566 \\ 2697845918095368817 & -6544917095126594593 & -98863838322233762 & -4478994071079223566 & 8424929086432683102 \end{pmatrix}$$

Le logarithme en base 2 de la norme infinie de chacun des 4 derniers vecteurs vaut : 62.87.

Pour le même entier p et $n = 6$, on a pgcd$(p-1, n)$=6 et

$\gamma = 19053258714271554136360749459891604195163415187008890704713244954725661616 82$

est une racine $6^{\text{ième}}$ modulo p. Le réseau renvoyé par LLL est le suivant :

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ -47997843284970993509178902426840500445 & \dots & \dots & \dots & \dots & 53460334507842253168491809011188073 18 \\ 53343876735755218826028083327959307763 & \dots & \dots & \dots & \dots & 47997843284970993509178902426840500446 \end{pmatrix}$$

Le logarithme en base 2 de la norme infinie de chacun des 2 derniers vecteurs vaut :

125.327.

L'algorithme LLL est à ce niveau "trop efficace", car il construit une base avec certains vecteurs ayant une norme trop petite. On souhaiterait plutôt obtenir une base dont chaque élément soit de norme proche de $p^{\frac{1}{n}}$.

2.4.9.3 A propos de $E(X)$

Depuis l'article fondateur de 2005, les recherches se sont focalisées sur le système de représentation AMNS, car ce dernier permet d'effectuer une réduction externe efficace. En effet, le polynôme $E(X)$ étant de la forme $X^n - \lambda$, calculer $S(X) = R(X) \bmod E(X)$ pour un polynôme $R(X)$ de degré au plus $2n-2$ revient à calculer les quantités $r_i + \lambda r_{n+i}$ pour $i \in [0, n-1]$ (cf. paragraphe 2.2.3, page 90). La quantité λ étant supposée "petite", la multiplication de chaque coefficient r_i peut être "gratuite" (cas où λ est une puissance de 2), ou bien peut être effectuée en utilisant l'instruction native de multiplication des entiers disponible sur l'architecture utilisée.
Cependant, d'après la proposition 2.6, page 112, afin d'appliquer la méthode Montgomery-like pour la réduction interne, il faut que $\rho \geqslant 2w\|M\|_\infty$, avec $w = 1 + (n-1)|\lambda|$ (cf. page 109). Ainsi, choisir une puissance de 2 pour λ ne permet pas un large choix de candidats, on se cantonnera à 2, 4, 8 et éventuellement 16 pour éviter que ρ ne devienne trop grand. De plus la condition sur ϕ impose de trouver un polynôme $M(X)$ tel que $\|M(X)\|_\infty \leqslant \frac{\phi}{4w^2}$, ce qui fait apparaître un facteur λ^2 dans le dénominateur de cette expression et impose une forte contrainte sur le polynôme $M(x)$. Etant donné que $\lambda = 1$ ne peut être choisi (cf. page 121, paragraphe 2.4.9.1), on a donc

$$w \geqslant 2n - 1\,.$$

La question légitime qui se pose est de déterminer s'il existe d'autres polynômes $E(X)$ pour lesquels :

- la réduction externe puisse être effectuée efficacement,
- $w < 2n - 1$.

Voici à titre d'exemple, les résultats d'une recherche en cours. On se place dans le cas où n est pair et considérons le polynôme $E(X) = X^n + X^{n/2} + 1$. Soit $R(X)$ un polynôme de degré au plus $2n-2$ à réduire modulo $E(X)$. Cette réduction externe peut être effectuée en calculant (cf. page 108) :

$$(r_0, \ldots r_{n-1}) + (r_n, \ldots, r_{2n-2})\mathcal{E}$$

où $\mathcal{E}$ est une matrice à $n-1$ lignes et n colonnes dont la $i^{\text{ème}}$ ligne contient les coefficients

du polynôme $X^{n+i} \bmod E(X)$ (en considérant que les lignes sont indexées à partir de 0) :

— pour $i \in [0, \frac{n}{2}-1]$, $X^{n+i} = -X^{\frac{n}{2}+i} - X^i$, étant donné que $X^n = -X^{\frac{n}{2}} - 1 \bmod E(X)$,

— pour $i = \frac{n}{2}$, on a $X^{n+\frac{n}{2}} = -X^n - X^{\frac{n}{2}} = 1 \bmod E(X)$,

— pour $i \in [\frac{n}{2}+1, n-2]$, $X^{n+i} = X^{i-\frac{n}{2}}$.

La matrice $\mathcal{E}$ est donc de la forme suivante :

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

où I_t est la matrice identité de taille t, et $0_{\frac{n}{2}-1,\frac{n}{2}+1}$ est la matrice nulle à $\frac{n}{2}-1$ lignes et $\frac{n}{2}+1$ colonnes. La réduction modulo $E(X)$ peut être effectuée en suivant l'algorithme 24. Cette réduction ne fait intervenir que des additions et des soustractions, ce qui peut

Algorithm 24 Réduction modulo $X^n + X^{\frac{n}{2}} + 1$

Require: n pair, $R(X)$ un polynôme de degré au plus $2n-2$
Ensure: $S(X) = R(X) \bmod E(X)$

```
for i ∈ [0, n/2 − 2] do
    s_i ← r_i − r_{n+i} + r_{3n/2+i}
end for
s_{n/2−1} ← r_{n/2−1} − r_{3n/2−1}
for i ∈ [n/2, n − 1] do
    s_i ← r_i − r_{n/2+i}
end for
return S
```

être effectué de façon très efficace et ce qui diminue de plus les problèmes de débordement induits par une multiplication.

Qu'en est-il de la valeur w associé à ce polynôme ?

Rappelons que cette valeur est calculée en considérant la matrice $\mathcal{E}'$ dont les coefficients sont la valeur absolue des coefficients de $\mathcal{E}$, i.e.

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

Elle est définie par (cf. proposition 2.4, page 108) :

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

Notons $\tilde{w}$ le vecteur dont les composantes sont $i + \sum_{j=1}^{n-1}(n-j)\mathcal{E}'_{ij}$ pour $i \in [1, n-1]$. On

a :

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

— $\tilde{w}_{\frac{n}{2}} = \frac{n}{2} + (n - \frac{n}{2}) = n$,

— $\tilde{w}_i = i + n - (i - \frac{n}{2}) = \frac{3n}{2}$ pour $i \in [\frac{n}{2}, n-1]$.

On en déduit donc que $w = \frac{3n}{2}$, ce qui pour $n > 2$ donne une valeur qui est inférieure à ce que l'on obtient pour un AMNS.

Le polynôme $E(X)$ choisi n'est pas le seul à vérifier cette propriété et on peut donc à priori trouver des PMNS qui donneront de meilleures performances que les AMNS. Notamment, par rapport au constat effectué sur l'écart entre n_{opt} et la valeur réelle de n permettant de construire des AMNS pour des entiers de taille 1024 ou 2048 (cf. paragraphe 2.4.3, page 101), on obtient des résultats plus intéressants. La condition sur ϕ donne :

$$\|M\|_\infty \leqslant \frac{\phi}{9n^2}\,.$$

On peut alors trouver (pour $\delta = 0$) des PMNS pour représenter des entiers de 1024 bits avec $n = n_{opt} + 3$ au lieu de $n = n_{opt} + 5$ (cf. figure 2.5), et des entiers de 2048 bits avec $n = n_{opt} + 8$ au lieu de $n = n_{opt} + 11$ (cf. figure 2.6).

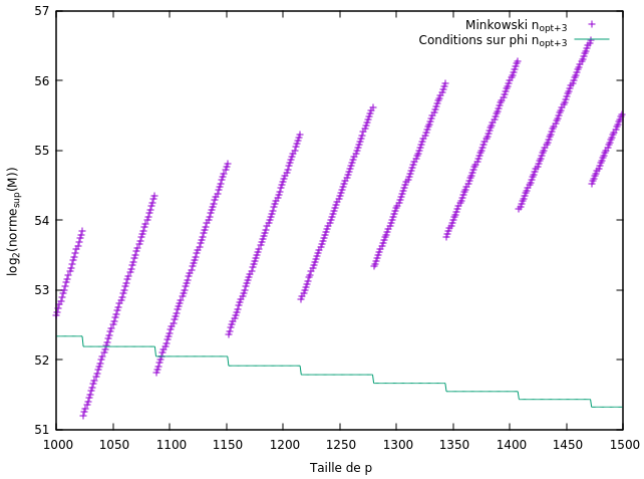


Figure 2.5 – Conditions à respecter sur $\|M(X)\|_\infty$ pour $\delta = 0$, $n = n_{\text{opt}} + 3$.

2.4.10 Quand les AMNS rencontrent les CAE

A ce stade, une suite naturelle de ce travail est d'intégrer la multiplication PMNS dans notre algorithme de calcul de point utilisant les CAE. D'autant plus que, d'après la table 1.23 page 70, la multiplication entre 2 entiers de 331 bits dans ce système de représentation

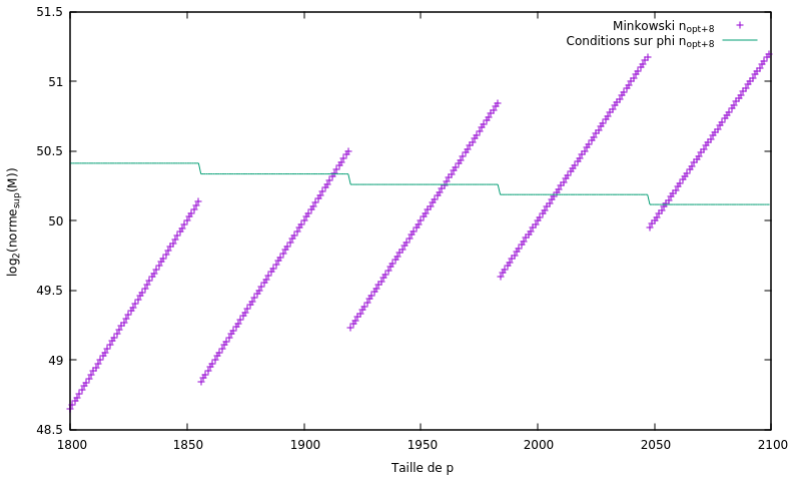


Figure 2.6 – Conditions à respecter sur $\|M(X)\|_\infty$ pour $\delta = 0$, $n = n_{\text{opt}} + 8$.

a un surcoût par rapport à la multiplication de 2 entiers de 256 bits qui n'est pas pénalisant. Dans [HMV21], nous avons réalisé une version non optimisée d'un calcul de kP avec la méthode CAE en utilisant la librairie GnuMP. Pour la courbe d'équation $y^2 = x^3 + 3$ sur $\mathbb{F}_p$ avec $p = 2^{331} - 36301$, nous avons obtenue une valeur médiane de 1201950 cycles pour un calcul de kP en $(X, Y)-$ only sur un processeur Intel Core i5-6500, 3.2 Ghz, gmp 6.2.1 et gcc 9.3.0.

Les premiers tests réalisés avec les AMNS sont encourageants, pour le premier de 332 bits

`p=765538298106951451734757489336228286644514418482466856852587204622626761593413905192864253127897907 3`

et la courbe définie par $y^2 = x^3 + 6x$, on effectue un échange de Diffie-Hellman complet en $(X, Y)-$only (avec reconstruction de la clé commune, cf paragraphe 1.10.2, page 78) en 215000 cycles environ sur un Intel Core i9-10980XE. Ce travail est pour l'instant en cours de développement.

Bibliographie

Chaînes d'additions euclidiennes

[AKS12] Kahraman Daglar Akdemir, Deniz Karakoyunlu et Berk Sunar, « Non-linear error detection for elliptic curve cryptosystems », in : *IET Information Security* **6**(1), mar. 2012, page(s): 28-40.

[Bah06] Hatem M. Bahig, « Improved Generation of Minimal Addition Chains », in : *Computing* **78**(2), 2006, page(s): 161-172.

[Bal+12] Brian Baldwin et al., « Co-Z ECC scalar multiplications for hardware, software and hardware–software co-design on embedded systems », in : *Journal of Cryptographic Engineering* **2**, 2012, page(s): 221-240.

[BCD20] Elaine B. Barker, LiLy Chen et Richard Davis, *Recommendation for Key-Derivation Methods in Key-Establishment Schemes*, Special Publication 800-56C rev. 2, NIST SP, août 2020, URL : https://doi.org/10.6028/NIST.SP.800-56Cr2.

[BBB94] François Bergeron, Jean Berstel et Srečko Brlek, « Efficient computation of addition chains », in : *Journal de Théorie des Nombres de Bordeaux* **6**(1), 1994, page(s): 21-38, ISSN : 12467405, 21188572.

[Ber06] Daniel Julius Bernstein, « Curve25519 : New Diffie-Hellman Speed Records », in : *Public Key Cryptography - PKC 2006*, sous la dir. de Moti Yung et al., Berlin, Heidelberg : Springer Berlin Heidelberg, 2006, page(s): 207-228, ISBN : 978-3-540-33852-9.

[BDL97] Dan Boneh, Richard Allan DeMillo et Richard Jay Lipton, « On the Importance of Checking Cryptographic Protocols for Faults », in : *Advances in Cryptology — EUROCRYPT '97*, sous la dir. de Walter Fumy, Berlin, Heidelberg : Springer Berlin Heidelberg, 1997, page(s): 37-51, ISBN : 978-3-540-69053-5.

[Bor76] Itshak Borosh, « Rational continued fractions with small partial quotients », in : *Notices Amer. Math. Soc.*, Abstract 731-10-29 **23**, 1976, page(s): A-52.

[BN83] Itshak Borosh et Harald Niederreiter, « Optimal multipliers for pseudo-random number generation by the linear congruential method », in : *BIT Numerical Mathematics volume* **23**, 1983, page(s): 65-74.

[BK11] Jean Bourgain et Alex Kontorovich, « On Zaremba's conjecture », in : *Comptes Rendus Mathematique* **349**(9), 2011, page(s): 493-495.

[BK14] Jean Bourgain et Alex Kontorovich, « On Zaremba's conjecture », in : *Annals of Mathematics* **180**(1), 2014, page(s): 137-196, ISSN : 0003486X, URL : http://www.jstor.org/stable/24522920.

[Bra39] Alfred T. Brauer, « On addition chains », in : *Bull. Amer. Math. Soc.* **45**, 1939, page(s): 736-739.

[BJ02] Éric Brier et Marc Joye, « Weierstraß Elliptic Curves and Side-Channel Attacks », in : *Public Key Cryptography*, sous la dir. de David Naccache et Pascal Paillier, Berlin, Heidelberg : Springer Berlin Heidelberg, 2002, page(s): 335-345.

[BH09] Billy Bob Brumley et Risto M. Hakala, « Cache-Timing Template Attacks », in : *Advances in Cryptology – ASIACRYPT 2009 : 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan*, Springer Berlin Heidelberg, 2009, page(s): 667-684, ISBN : 978-3-642-10366-7, DOI : 10.1007/978-3-642-10366-7_39.

[Coo06] Joshua N. Cooper, « Continued fractions with partial quotients bounded in average », in : *Fibonacci Quarterly* **44**(4), 2006, page(s): 297-301.

[CHS14] Craig Costello, Huseyin Hisil et Benjamin Smith, « Faster Compact Diffie–Hellman : Endomorphisms on the x-line », in : *Advances in Cryptology – EUROCRYPT 2014*, sous la dir. de P. Q. Nguyen et E. Oswald, Berlin, Heidelberg : Springer Berlin Heidelberg, 2014, page(s): 183-200.

[CL15] Craig Costello et Patrick Longa, « Four$\mathbb{Q}$: four-dimensional decompositions on a $\mathbb{Q}$-curve over the Mersenne prime. », English, in : *Advances in Cryptology – ASIACRYPT 2015, Auckland, New Zealand*, Berlin : Springer, 2015, page(s): 214-235, DOI : 10.1007/978-3-662-48797-6_10.

[Cus93] Thomas W. Cusick, « Zaremba's Conjecture and Sums of the Divisor Function », in : *Mathematics of Computation* **61**(203), 1993, page(s): 171-176.

[DV17] Fangan-Yssouf Dosso et Pascal Véron, « Cache timing attacks countermeasures and error detection in Euclidean addition chains based scalar multiplication algorithm for elliptic curves », in : *IOLTS 2017*, 2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS),

Thessaloniki, Greece : IEEE, juil. 2017, page(s): 163-168, DOI : 10.1109/IOLTS.2017.8046212.

[Dos+18] Fangan-Yssouf Dosso et al., « Euclidean addition chains scalar multiplication on curves with efficient endomorphism », in : *Journal of Cryptographic Engineering* **8**, 2018, page(s): 351-367, DOI : 10.1007/s13389-018-0190-0.

[DLS81] Peter J. Downey, Benton L. Leong et Ravi Sethi, « Computing Sequences with Addition Chains », in : *SIAM J. Comput.* **10**(3), 1981, page(s): 638-646.

[FLS15] Armando Faz-Hernández, Patrick Longa et Ana H. Sánchez, « Efficient and secure algorithms for GLV-based scalar multiplication and their implementation on GLV-GLS curves (extended version) », in : *J. Cryptographic Engineering* **5**(1), 2015, page(s): 31-52, DOI : 10.1007/s13389-014-0085-7.

[Fen+06] Min Feng et al., « Signed MSB-Set Comb Method for Elliptic Curve Point Multiplication », in : *Information Security Practice and Experience*, sous la dir. de Kefei Chen et al., Berlin, Heidelberg : Springer Berlin Heidelberg, 2006, page(s): 13-24.

[GLV01] Robert P. Gallant, Robert J. Lambert et Scott Alexander Vanstone, « Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms », in : *Advances in Cryptology — CRYPTO*, t. 2139, LNCS, Springer, 2001, page(s): 190-200.

[GJ16] R. Raveen Goundar et Marc Joye, « Inversion-free arithmetic on elliptic curves through isomorphisms », in : *J. Cryptographic Engineering*, 2016, page(s): 187-199.

[GJM10] R. Raveen Goundar, Marc Joye et Atsuko Miyaji, « Co-Z Addition Formulae and Binary Ladders on Elliptic Curves - (Extended Abstract) », in : *CHES*, 2010, page(s): 65-79.

[Gou+11] R. Raveen Goundar et al., « Scalar multiplication on Weierstrass elliptic curves from Co-Z arithmetic », in : *J. Cryptographic Engineering*, 2011, page(s): 161-176.

[GKP94] Ronald Lewis Graham, Donald Ervin Knuth et Oren Patashnik, *Concrete Mathematics : A Foundation for Computer Science*, 2nd, USA : Addison-Wesley Longman Publishing Co., Inc., 1994, ISBN : 0201558025.

[Hen92] Doug Hensley, « Continued fraction Cantor sets, Hausdorff dimension, and functional analysis », in : *Journal of Number Theory* **40**(3), 1992, page(s): 336-358, DOI : https://doi.org/10.1016/0022-314X(92)90006-B.

[HMV21] Fabien Herbaut, Nicolas Méloni et Pascal Véron, « Compact variable-base ECC scalar multiplication using Euclidean addition chains », in : *SECRYPT 2021 - Proceedings of the 18th International Conference on Security and Cryptography, 06-08 July, 2021*, sous la dir. de ..., SciTePress, 2021, page(s): 178-183, DOI : 10.5220/0005014501780183, URL : https://doi.org/10.5220/0005014501780183.

[HV10] Fabien Herbaut et Pascal Véron, « A public key cryptosystem based upon euclidean addition chains », in : *SETA 2010*, t. 6338, Paris, France : Springer, sept. 2010, page(s): 284-297, URL : https://hal.inria.fr/hal-00674252.

[Her+10] Fabien Herbaut et al., « Random Euclidean Addition Chain Generation and Its Application to Point Multiplication », in : *INDOCRYPT 2010*, t. 6498, Hyderabad, India : Springer, déc. 2010, page(s): 238-261, DOI : 10.1007/978-3-642-17401-8_18, URL : https://hal.inria.fr/hal-00674251.

[His+08] Huseyin Hisil et al., « Twisted Edwards Curves Revisited », in : *Advances in Cryptology - ASIACRYPT 2008*, sous la dir. de Josef Pieprzyk, Berlin, Heidelberg : Springer Berlin Heidelberg, 2008, page(s): 326-343, ISBN : 978-3-540-89255-7.

[Hua15] Shinn Yih Huang, « An Improvement to Zaremba's Conjecture », in : *Geometric and Functional Analysis* **25**, 2015, page(s): 860-914.

[KAK96] Cetin Kaya Koc, Tolga Acar et Burton S. Kaliski, « Analyzing and comparing Montgomery multiplication algorithms », in : *IEEE Micro* **16**(3), 1996, page(s): 26-33, DOI : 10.1109/40.502403.

[Kit18] Scott Kitterman, *Cryptographic Algorithm and Key Usage Update to DomainKeys Identified Mail (DKIM)*, RFC 8301, jan. 2018, DOI : 10.17487/RFC8301, URL : https://rfc-editor.org/rfc/rfc8301.txt.

[Knu97] Donald Ervin Knuth, *The Art of Computer Programming : Fundamental Algorithms*, 3rd, t. 2, Addison Wesley, juil. 1997.

[Kom05] Takao Komatsu, « On a Zaremba'S conjecture for powers », in : *Sarajevo journal of mathematics* **1**(13), 2005, page(s): 9-13.

[LL94] Chae Hoon Lim et Pil Joong Lee, « More Flexible Exponentiation with Precomputation », in : *Advances in Cryptology — CRYPTO '94*, sous la dir. d'Yvo G. Desmedt, Berlin, Heidelberg : Springer Berlin Heidelberg, 1994, page(s): 95-107, ISBN : 978-3-540-48658-9.

[Mél07a] Nicolas Méloni, « Arithmétique pour la Cryptographie basée sur les Courbes Elliptiques », thèse de doct., Université de Montpellier, France, 2007.

[Mél07b] Nicolas Méloni, « New Point Addition Formulae for ECC Applications », in : *Arithmetic of Finite Fields*, t. 4547, LNCS, Springer Berlin / Heidelberg, 2007, page(s): 189-201.

[MMN06] Ilya Mironov, Anton Mityagin et Kobbi Nissim, « Hard Instances of the Constrained Discrete Logarithm Problem. », in : *ANTS*, sous la dir. de Florian Hess, Sebastian Pauli et Michael E. Pohst, t. 4076, Lecture Notes in Computer Science, Springer, 14 nov. 2006, page(s): 582-598, isbn : 3-540-36075-1.

[Möl01] Bodo Möller, « Algorithms for Multi-exponentiation », in : *Selected Areas in Cryptography*, sous la dir. de Serge Vaudenay et Amr M. Youssef, Berlin, Heidelberg : Springer Berlin Heidelberg, 2001, page(s): 165-180, isbn : 978-3-540-45537-0.

[Mon87] Peter Lawrence Montgomery, « Speeding the Pollard and elliptic curve methods of factorization », in : *Mathematics of Computation* **48**, 1987, page(s): 243-264, issn : 0025–5718.

[Mon02] Peter Lawrence Montgomery, *Evaluating recurrences of form $X_{m+n} = f(X_m, X_n, X_{m-n})$ via Lucas chains*, 2002, url : https://cr.yp.to/bib/1992/montgomery-lucas.pdf.

[Nie78] Harald Niederreiter, « Quasi-Monte Carlo methods and pseudo-random numbers », in : *Bull. Amer. Math. Soc.* **84**(6), nov. 1978, page(s): 957-1041, url : https://projecteuclid.org:443/euclid.bams/1183541461.

[Nie86] Harald Niederreiter, « Dyadic fractions with small partial quotients », in : *Monatshefte für Mathematik* **101**, 1986, page(s): 309-315.

[OKS06] Acıiçmez Onur, Çetin Kaya Koc et Jean-Pierre Seifer, « Predicting Secret Keys via Branch Prediction », in : *Proceedings of CT-RSA'07*, CT-RSA'07, San Francisco, CA : Springer-Verlag, 2006, page(s): 225-242.

[Pon+09] Simon Pontarelli et al., « Error detection in addition chain based ECC Point Multiplication », in : *2009 15th IEEE International On-Line Testing Symposium*, juin 2009, page(s): 192-194.

[Pro+15] Julien Proy et al., « Full Hardware Implementation of Short Addition Chains Recoding for ECC Scalar Multiplication », in : *Compas : Conférence d'informatique en Parallélisme*, t. 4547, 2015.

[Res99] Eric Rescorla, *Diffie-Hellman Key Agreement Method*, RFC 2631, juin 1999, doi : 10.17487/RFC2631, url : https://rfc-editor.org/rfc/rfc2631.txt.

[Riv11] Matthieu Rivain, *Fast and Regular Algorithms for Scalar Multiplication over Elliptic Curves*, https://eprint.iacr.org/2011/338.pdf, 2011.

[Ruk11] Maria Gennadievna Rukavishnikova, « The law of large numbers for the sum of the partial quotients of a rational number with fixed denominator », in : *Mathematical Notes* **90**(3), 2011, page(s): 418-430.

[Sch37] Arnold Scholz, « Aufgabe 253 », in : *Jahresbericht der deutschen Mathematiker-Vereinigung* **47**, 1937, page(s): 41-42.

[Sch75] Arnold Schönhage, « A lower bound for the length of addition chains », in : *Theoretical Computer Science* **1**(1), 1975, page(s): 1-12, ISSN : 0304-3975, DOI : https://doi.org/10.1016/0304-3975(75)90008-0.

[SCQ03] Francesco Sica, Mathieu Ciet et Jean-Jacques Quisquater, « Analysis of the Gallant-Lambert-Vanstone Method Based on Efficient Endomorphisms : Elliptic and Hyperelliptic Curves », in : *Selected Areas in Cryptography*, sous la dir. de Kaisa Nyberg et Howard Heys, Berlin, Heidelberg : Springer Berlin Heidelberg, 2003, page(s): 21-36, ISBN : 978-3-540-36492-4.

[Sil09] Joseph H. Silverman, *The Arithmetic of Elliptic Curves, 2nd Edition*, Graduate Texts in Mathematics, Springer Verlag, 2009.

[Sol01] Jerome A. Solinas, *Low-Weight Binary Representations for Pairs of Integers*, rapp. tech., 2001, URL : http://www.cacr.math.uwaterloo.ca/techreports/2001/corr2001-41.ps.

[Sub89] Mathukumalli Subbarao, « Addition chains - some results and problems », in : *Number theory and applications*, 1989, page(s): 555-574.

[WJ68] Evert Wattel et G.A. Jensen, *Efficient calculation of powers in a semigroup*, Stichting Mathematisch Centrum. Afdeling Zuivere Wiskunde. ZW. 1968-001, Stichting Math. Centrum, 1968.

[Yao76] Andrew Chi-Chih Yao, « On the Evaluation of Powers », in : *SIAM Journal on Computing* **5**(1), 1976, page(s): 100-103.

[YK75] Andrew Chi-Chih Yao et Donald Ervin Knuth, « Analysis of the substractive algorithm for greatest common divisors », in : *Proc. Nat. Acad. Sci. USA* **72**(12), déc. 1975, page(s): 4720-4722.

[YL02] Monrudee Yodphotong et Vichian Laohakosol, « Proofs of Zaremba's Conjecture for powers of 6 », in : *Proc. Internat. Conf. Algebra Appl.* 2002, page(s): 278-282.

[Zar72] Stanisław Krystyn Zaremba, « La Méthode des “Bons Treillis” pour le Calcul des Intégrales Multiples », in : *Applications of Number Theory to Numerical Analysis*, Academic Press, 1972, page(s): 39-119, ISBN : 978-0-12-775950-0, DOI : https://doi.org/10.1016/B978-0-12-775950-0.50009-1.

Le système de représentation PMNS

[Bab86] László Babai, « On Lovász' lattice reduction and the nearest lattice point problem », in : *Combinatorica* **6**, mar. 1986, page(s): 1-13, DOI : 10.1007/BF02579403.

[BIP05a] Jean-Claude Bajard, Laurent Imbert et Thomas Plantard, « Arithmetic operations in the polynomial modular number system », in : *17th IEEE Symposium on Computer Arithmetic (ARITH'05)*, 2005, page(s): 206-213, DOI : 10.1109/ARITH.2005.11.

[BIP05b] 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, page(s): 159-169, ISBN : 978-3-540-30564-4.

[Bon15] Nicolae Ciprian Bonciocat, « Schönemann–Eisenstein–Dumas-Type Irreducibility Conditions that Use Arbitrarily Many Prime Numbers », in : *Communications in Algebra* **43**(8), 2015, page(s): 3102-3122, DOI : 10.1080/00927872.2014.910800.

[BZ10] Richard Peirce Brent et Paul Zimmermann, *Modern Computer Arithmetic*, Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, 2010, DOI : 10.1017/CBO9780511921698.

[Car+19] Sébastien Carré et al., « Cache-Timing Attack Detection and Prevention », in : *Constructive Side-Channel Analysis and Secure Design*, sous la dir. d'Ilia Polian et Marc Stöttinger, Cham : Springer International Publishing, 2019, page(s): 13-21, ISBN : 978-3-030-16350-1.

[CEH21] Titouan Coladon, Philippe Elbaz-Vincent et Cyril Hugounenq, « MPHELL : A fast and robust library with unified and versatile arithmetics for elliptic curves cryptography (extended version) », in : *ARITH 2021*, Transactions on Emerging Topics in Computing, Torino, Italy, juin 2021, URL : https://hal.archives-ouvertes.fr/hal-03284677.

[DDV20] Laurent-Stéphane Didier, Fangan-Yssouf Dosso et Pascal Véron, « Efficient modular operations using the adapted modular number system », in : *Journal of Cryptographic Engineering*, jan. 2020, doi : 10.1007/s13389-019-00221-7, url : https://hal.archives-ouvertes.fr/hal-02486345.

[Did+19] Laurent-Stéphane Didier et al., « Randomization of Arithmetic over Polynomial Modular Number System », in : *26th IEEE International Symposium on Computer Arithmetic*, t. 1, Proceedings of the 2019 IEEE 26th Symposium on Computer Arithmetic, Kyoto, Japan : IEEE Computer Society, juin 2019, page(s): 199-206, doi : 10.1109/ARITH.2019.00048.

[EG12] Nadia El Mrabet et Nicolas Gama, « Efficient Multiplication over Extension Fields », in : *WAIFI 2012*, Ghent, Belgium, juil. 2012, doi : 10.1007/978-3-642-31662-3_10, url : https://hal.archives-ouvertes.fr/hal-01197178.

[Emd81] Peter van Emde-Boas, *Another NP-complete partition problem and the complexity of computing short vectors in a lattice*, Report. Department of Mathematics. University of Amsterdam, Department, Univ., 1981, url : https://books.google.fr/books?id=tCQiHQAACAAJ.

[Gar59] Harvey L. Garner, « The Residue Number System », in : *Papers Presented at the the March 3-5, 1959, Western Joint Computer Conference*, IRE-AIEE-ACM '59 (Western), San Francisco, California : Association for Computing Machinery, 1959, page(s): 146-153, isbn : 9781450378659, doi : 10.1145/1457838.1457864, url : https://doi.org/10.1145/1457838.1457864.

[HPS11] Guillaume Hanrot, Xavier Pujol et Damien Stehlé, « Algorithms for the Shortest and Closest Lattice Vector Problems », in : *Coding and Cryptology*, sous la dir. d'Yeow Meng Chee et al., Berlin, Heidelberg : Springer Berlin Heidelberg, 2011, page(s): 159-190, isbn : 978-3-642-20901-7.

[Joh99] Anna M. Johnston, « A Generalized qth Root Algorithm », in : *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '99, Baltimore, Maryland, USA : Society for Industrial et Applied Mathematics, 1999, page(s): 929-930, isbn : 0898714346.

[MG02] Daniele Micciancio et Shafi Goldwasser, *Complexity of Lattice Problems*, USA : Kluwer Academic Publishers, 2002, isbn : 0792376889.

[NP08] Christophe Negre et Thomas Plantard, « Efficient Modular Arithmetic in Adapted Modular Number System Using Lagrange Representation », in : *Information Security and Privacy*, sous la dir. d'Yi Mu, Willy Susilo et Jen-

nifer Seberry, Berlin, Heidelberg : Springer Berlin Heidelberg, 2008, page(s): 463-477, isbn : 978-3-540-70500-0.

[NS06] Phong Quang Nguyen et Damien Stehlé, « LLL on the Average », in : *Algorithmic Number Theory*, sous la dir. de Florian Hess, Sebastian Pauli et Michael Pohst, Berlin, Heidelberg : Springer Berlin Heidelberg, 2006, page(s): 238-256, isbn : 978-3-540-36076-6.

[Odl90] Andrew Michael Odlyzko, « The rise and fall of knapsack cryptosystems », in : *In Cryptology and Computational Number Theory*, A.M.S, 1990, page(s): 75-88.

[Per03] Colin Percival, « Rapid Multiplication Modulo the Sum and Difference of Highly Composite Numbers », in : *Mathematics of Computation* **72**(241), 2003, page(s): 387-395, issn : 00255718, 10886842, url : http://www.jstor.org/stable/4099997.

[Zeu19] Thomas Zeugmann, *Taking Discrete Roots in the Field $\mathbb{Z}_p$ and in the Ring $\mathbb{Z}_{p^e}$*, rapp. tech., Hokkaido University, 2019, url : https://www-alg.ist.hokudai.ac.jp/~thomas/TCSTR/tcstr_19_83/tcstr_19_83.pdf.