Chiffrement homomorphe

DUPONT-LATAPIE Ulysse 21981115
ROUGEOLLE Yoan 21953845
YE Nicolas 71814766

Encadré par : MOLIN Pascal

2022 - 2023

Sommaire

1 Introduction

Imaginons que l'on dispose de données confidentielles sur laquelle on souhaite exécuter un algorithme, mais que :

- Soit l'algorithme est détenu par un tiers.
- Soit on ne dispose pas de la puissance de calcul pour exécuter l'algorithme.

Une solution est alors de contacter un tiers pour exécuter l'algorithme en question. Cependant, nous ne lui faisons pas confiance et souhaitons que toute information reste confidentielle. Il faut alors que les informations que l'on donne au tiers soient chiffrées. Le tiers exécutera l'algorithme sur des chiffrés et renverra un résultat également chiffré.

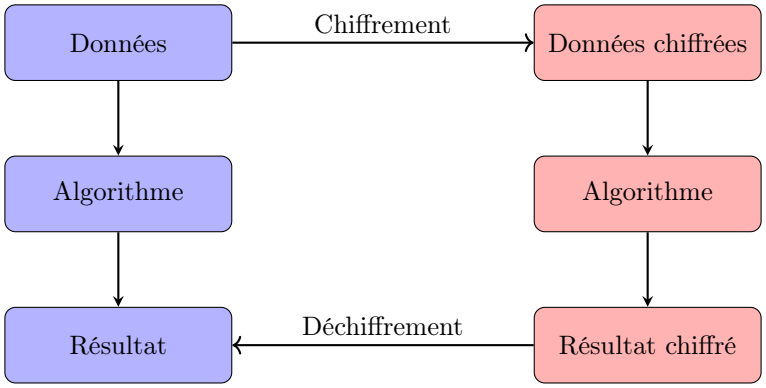


Figure 1: Schéma du principe de la solution

Ce diagramme est le principe même du chiffrement homomorphe. On chiffre nos données, on exécute l'algorithme sur les données chiffrées, puis on déchiffre; le résultat alors obtenu doit être le même que si l'on avait exécuté l'algorithme sur les données d'origine. Ce type de chiffrement pourrait avoir des applications pratiques dans la recherche de génomes dans une séquence ADN que l'on souhaite garder secrète, ou bien dans l'emploi de moteurs de recherche tel que les requêtes utilisateur et réponses du moteur soient masquées.

Lorsqu'un algorithme est conçu, son fonctionnement est généralement décrit sur des objets plus complexes, comme des entiers, des chaînes de caractères, etc. Mais on peut toujours représenter ces algorithmes avec des fonctions booléennes. Dans l'implémentation de ce chiffrement, notre clair est un unique bit. De ce fait, il faut traduire nos algorithmes en fonctions booléennes. Mais cette traduction est difficile : une simple addition d'entiers est déjà compliquée à convertir. Ainsi, les algorithmes cités deviennent excessivement complexes à traduire. Cela va être un des principaux enjeux sur lesquels nous nous attarderons dans la suite.
Nous allons d'abord étudier la construction d'un tel cryptosystème que nous appellerons par la suite FHE (Fully Homomorphic Encryption), en nous appuyant principalement sur la thèse de Gentry de 2009 [1] : nous présenterons ensuite la manière dont nous avons implémenté ce cryptosystème grâce aux travaux de Smart et Vercauteren [4].

2 Cryptosystème employé

2.1 Principe général

On définit le cryptosystème suivant :

Définition 2.1.1. *Soit $(A, +, \times)$ un anneau topologique et $V \subset A$ un voisinage de 0 dans cet anneau. On appelle cryptosystème faiblement homomorphe sur A la donnée de :*

- *$K \subset K_1 \times K_2$ l'espace des couples de clés (publique, privée),*
- *$\mathcal{M} = (\mathbb{F}_2, \oplus, \wedge)$ l'espace des clairs,*
- *A l'espace des chiffrés,*
- *$e : \mathcal{M} \times K_1 \to A \cap V$ la fonction de chiffrement,*

- $d : (A \cap V) \times K_2 \to \mathcal{M}$ *la fonction de déchiffrement,*

tels que pour tout couple $(p_k, s_k) \in K$*:*

- $\forall b \in \mathcal{M}$, $d(e(b, p_k), s_k) = b$.
- $\forall c_1, c_2 \in A \cap V$, *si* $c_1 + c_2 \in A \cap V$, *alors* $d(c_1 + c_2, s_k) = d(c_1, s_k) \oplus d(c_2, s_k)$
- $\forall c_1, c_2 \in A \cap V$, *si* $c_1 \times c_2 \in A \cap V$, *alors* $d(c_1 \times c_2, s_k) = d(c_1, s_k) \wedge d(c_2, s_k)$

Autrement dit, la fonction de déchiffrement est une sorte de morphisme d'anneau de $A \cap V$ vers $\mathbb{F}_2$, mais $A \cap V$ n'est aucunement un anneau. Les opérations d'addition et de multiplication ne sont pas stables dans $A \cap V$. Les chiffrés peuvent alors sortir du voisinage V et on ne pourra donc plus les déchiffrer. Cela nous amène à la notion de bruit :

Définition 2.1.2. *Si de plus A est muni d'une distance, on définit le bruit d'un chiffré comme la distance entre ce chiffré et 0.*

Si ce bruit devient trop grand, le chiffré sort alors du voisinage V. Le but est donc de pouvoir composer les opérations, c'est-à-dire appliquer des fonctions booléennes, tout en maintenant les chiffrés dans $A \cap V$ pour les déchiffrer efficacement.

2.2 SHE

Le SHE (Somewhat Homomorphic Encryption) est un premier niveau de cryptosystème homomorphe. Comme on travaille sur $(\mathbb{F}_2, \oplus, \wedge)$, on peut représenter les fonctions booléennes appliquées à nos clairs comme des circuits de portes logiques $\oplus$ et $\wedge$.

Définition 2.2.1. *Soit* $f : \mathbb{F}_2^n \to \mathbb{F}_2$ *une fonction booléenne. En représentant f comme un arbre (les feuilles étant les monômes correspondant à chacune des variables de notre fonction), on définit la profondeur de f comme étant la profondeur de cet arbre.*

Définition 2.2.2. *Si* $f : \mathbb{F}_2^n \to \mathbb{F}_2$ *une fonction booléenne, on note* $\tilde{f} : A^n \to A$ *le circuit associé à f où les* $\oplus$ *sont remplacé par des* $+$ *et les* $\wedge$ *par des* $\times$*.*

Définition 2.2.3. *Un cryptosystème défini comme dans 2.1.1 est un SHE si et seulement si* $\exists n \in \mathbb{N}^*$, *tel que* $\forall f$ *fonction booléenne de profondeur* $\leq n$, $\forall c_1, c_2, ..., c_k \in Im(e)$, $\tilde{f}(c_1, c_2, ..., c_k) \in A \cap V$.

Remarque : Dans certaines applications, un SHE peut suffire si la profondeur limite est assez grande. Cependant, dans notre cas, on souhaite se détacher de cette limite et créer un FHE (Fully Homomorphic Encryption).

Définition 2.2.4. *Un cryptosystème défini comme dans 2.1.1 est un FHE si et seulement si* $\forall f$ *fonction booléenne,* $\forall c_1, c_2, ..., c_k \in Im(e)$, $\tilde{f}(c_1, c_2, ..., c_k) \in A \cap V$.

En pratique, nous savons construire des SHE, mais ce n'est pas le cas pour les FHE. Cependant, Gentry a apporté dans sa thèse un moyen de transformer un SHE en FHE : le bootstrap.

2.3 Le Bootstrap : passer d'un SHE vers un FHE

Le concept de bootstrapping n'est pas un terme propre au chiffrement homomorphe, ou même à la cryptographie : on le retrouve par exemple dans la conception des langages de programmation. Il désigne la capacité d'un système complexe, fondé sur une certaine base, à poursuivre sa propre construction à partir de lui-même, une fois qu'il est assez avancé. Ici, nous allons bootstrapper un SHE, pour pouvoir construire un FHE. Concrètement, on veut construire une fonction $h : A \to A$ vérifiant :

- $\forall c \in A \cap V, h(c) \in A \cap V$ et $d(h(c), sk) = d(c, sk)$.
- Le bruit de $h(c)$ est inférieur à celui de c.

On appellera cette fonction la fonction de rechiffrement.
L'idée de Gentry est d'utiliser la fonction de déchiffrement pour bootstrapper le SHE ; on évalue de façon homomorphe la fonction de déchiffrement, qui au lieu de s'appliquer sur les bits de la clé privée, s'applique sur les bits chiffrés de cette dernière. Ainsi, pour maintenir les chiffrés dans $A \cap V$, on peut appliquer un rechiffrement entre chaque opération. Cependant, pour pouvoir appliquer ce rechiffrement, il faut déjà que la fonction de déchiffrement soit évaluable dans notre SHE, c'est-à-dire qu'elle ait une profondeur plus petite que la limite du SHE.

Théorème 2.3.1. *Un SHE de profondeur limite $n+1$ ayant une fonction de déchiffrement de profondeur inférieure ou égale à n est bootstappable en un FHE.*

Preuve. Il suffit de faire $n + 1 - \text{profondeur}_{\text{rechiffrement}} \geq 1$ opérations et ensuite d'exécuter le rechiffrement. On répète ce procédé pour continuer l'évaluation de toute fonction f. □

Remarque : Toute la difficulté du bootstrap est d'avoir une fonction de déchiffrement de profondeur inférieure à la limite. Pour résoudre ce problème, on utilise un procédé appelé squashing par Gentry, permettant de construire une nouvelle fonction de déchiffrement ayant la profondeur voulue.

3 Cryptosystème utilisé par Gentry

Nous allons dans cette partie expliquer la construction du cryptosystème utilisé par Gentry pour faire du chiffrement homomorphe. Par la suite , on note $\vec{e_i}$ le i-ième vecteur de la base canonique.

3.1 Réseaux Euclidiens (Lattices)

Nous allons utiliser une structure bien particulière : les réseaux euclidiens sur $\mathbb{R}^n$.

Définition 3.1.1. *Soit $B = \{\vec{b_1}, \vec{b_2}, ..., \vec{b_n}\}$ une famille de vecteurs indépendants de $\mathbb{R}^n$. Le réseau engendré par B est l'ensemble des combinaisons linéaires entières des vecteurs de B :*

$$\mathcal{L}(B) = \{\Sigma_i \vec{b_i} x_i \mid \forall i \in \{1, ..., n\} \ x_i \in \mathbb{Z}\}$$

B est appelée base de ce réseau et est associée à une matrice de n lignes et m colonnes sur laquelle la ligne i représente $\vec{b_i}$. Pour plus de facilité, on supposera par la suite que $n = m$, c'est à dire que l'on n'utilisera que des matrices de rang plein. Par la suite, les vecteurs de la matrice seront **toujours en lignes**.

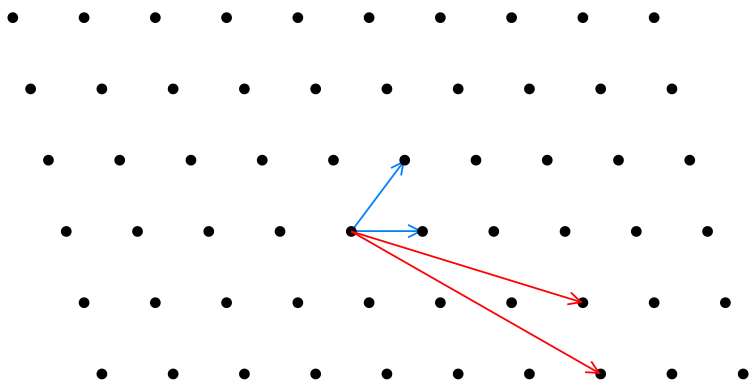


Figure 2: Visualisation d'un réseau sur $\mathbb{R}^2$

Les vecteurs bleus forment une base, tout comme les vecteurs rouges. L'ensemble des points noirs forme le réseau.

Il existe une infinité de bases pour un réseau donné. En effet soit B_1 une base d'un réseau Λ, soit U une matrice unimodulaire (coefficients entiers et déterminant égal à +1 ou -1), alors $B_2 = U \times B_1$ est de nouveau une base de Λ. Ces bases jouent un rôle clé dans le cryptosystème, puisqu'une certaine base va être la clé privée, et une autre sera la clé publique, l'important est de bien les choisir.

A partir d'une matrice représentant une base, il existe une sorte de forme "canonique" de celle-ci, une matrice représentant une autre base, engendrant le même réseau, calculable à partir de toutes les autres qui est

unique : La forme normale d'Hermite ou HNF.

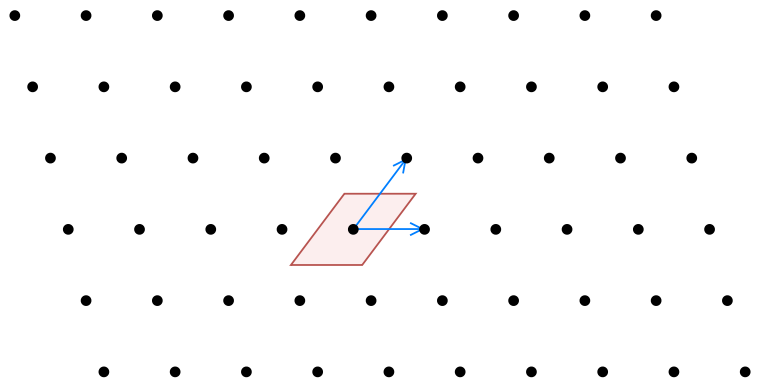
Définition 3.1.2. *Soit Λ un réseau, soit B une base de Λ, il existe une matrice unimodulaire U telle que :*

- *HNF(B) = U × B*
- *HNF(B) est une matrice triangulaire inférieur*
- *La diagonale de HNF(B) est strictement positive, $HNF(B)_{i,i} > 0$ pour tout i*
- *Pour tout $i > j$ $HNF(B)_{i,j} \in [-HNF(B)_{j,j}/2, HNF(B)_{j,j}/2[$*

Un dernier point à aborder sur les réseaux est leur réduction modulo une base B.

Définition 3.1.3. *Le parallélogramme $\mathcal{P}(B)$ pour une certaine base B centré sur l'origine est défini par :*

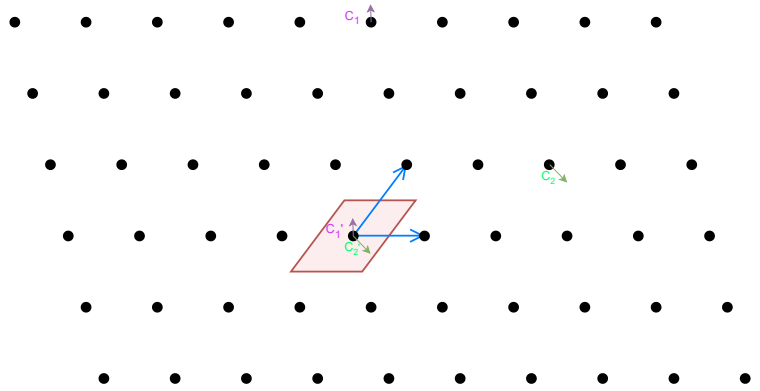
$$\mathcal{P}(B) = \{\Sigma_i \vec{b_i} x_i \mid x_i \in [-1/2, 1/2[\}$$



Soit c un vecteur de $\mathbb{R}^n$ et soit Λ un réseau de $\mathbb{R}^n$ muni d'une base B, calculer c modulo B correspond à chercher un vecteur c' dans $\mathcal{P}(B)$ tel que $c - c' \in \Lambda$. Ceci peut être calculé assez facilement grâce à l'inverse de B.

$$c' = \lceil c \times B^{-1} \rfloor \times B$$

$\lceil \ \rfloor$ correspond à prendre la partie fractionnaire de toutes les composantes du vecteur.



3.2 Cryptosystème de Gentry

3.2.1 Structures

On se place dans l'anneau $R = \mathbb{Z}[X]/f_N(X)$ où $f_N(X)$ est un polynôme unitaire de degré N. La structure de groupe additif de R est isomorphe à celle de $\mathbb{Z}^N$.

Définition 3.2.1. *Soit ϕ l'application linéaire suivante :*

$$\phi : R \to \mathbb{Z}^N$$
$$X^i \mapsto e_{i+1}$$

Cette application fait la correspondance entre polynôme et vecteur de $\mathbb{Z}^N$. Le réseau isomorphe à R est $\mathcal{L}(e_1, ..., e_N) = \phi(R)$.

Définition 3.2.2. *Soit I un idéal de R alors $\phi(I) \subseteq \mathbb{Z}^N$ est un réseau idéal (Ideal Lattice)*

On confondra par la suite $\phi(I)$ et I. Pour plus de facilité, on n'utilisera par la suite que des idéaux principaux, c'est-à-dire engendrés par un seul élément de R, autrement dit tels que $I = (\vec{v}),\ \vec{v} \in R\ \vec{v} \neq 0$. On pourra prendre comme base de ce réseau la base suivante B :

$$B = \{\vec{v_i} = \vec{v} \times X^i \ \ mod\ \ f_N(X) \ \mid \ i \in \{0, ..., N-1\}\}$$

Théorème 3.2.1. *Si $f_N(X)$ est irréductible alors B est une base.*

Preuve. Soit $\vec{w} \in I$, il existe $\vec{a} \in R$ tel que $\vec{w} = \vec{a} \times \vec{v}$. Donc si $a(X) = \Sigma_i a_i X^i$ et $v(X) = \Sigma_i v_i X^i$ alors $w(X) = \Sigma_i a_i \times v(X) \times X^i = \Sigma_i a_i \times \vec{v_i} \mod f_N(X)$, donc B est générateur.
Soit $\lambda_0, \lambda_1, ..., \lambda_{N-1}$ (N-1) scalaires tels que $\sum_i \lambda_i \times \vec{v_i} = 0$. On a alors $\vec{v} \times \sum_i \lambda_i \times X^i \mod f_N(X) = 0$. Comme $f_N(X)$ est irréductible, R est intègre. Comme $\vec{v} \neq 0$, $\sum_i \lambda_i \times X^i \mod f_N(X) = 0$, et comme (1, X, X^2, ..., X^{N-1}) est libre, pour tout i, $\lambda_i = 0$, ainsi B est libre, et est donc une base. □

L'intérêt des réseaux idéaux est qu'en plus de leur stabilité par addition, ils sont dotés d'une structure d'idéal, nous permettant de faire des multiplications, ce qui est utile dans le cadre du chiffrement homomorphe.

3.2.2 Chiffrement / Déchiffrement

Dans ce cryptosystème, nous prenons R défini comme plus haut, et deux idéaux I et J tels que : $R = I + J$ avec J un réseau idéal, et I un "petit" idéal. En pratique $I = (2)$. Pour chiffrer un message m, il y a deux étapes : tout d'abord on associe à m un élément de R/I, que l'on note $\vec{e}$, puis on translate $\vec{e}$ par un vecteur du réseau idéal J.
Dans la pratique, on n'encodera que des bits, donc notre espace de clairs sera $M = \{0, 1\}$. On prend :

- $m \in M$ notre bit à chiffrer
- $I = (2)$
- $J = (\vec{v})$ un idéal principal qui est aussi un réseau
- B un base de ce réseau
- $\vec{m} = (m, 0, 0, ..., 0)$
- $\vec{e} = \vec{m} + 2\vec{r}$ où $\vec{r}$ est un vecteur ayant comme composantes des valeurs dans $\{0, 1, -1\}$ (choisies au hasard).

Soit alors $\vec{x}$ un vecteur aléatoire à composantes entières, on calcule le chiffré $\vec{c} = \vec{x}B + \vec{e}$.

Parlons maintenant de la clé privée et de la clé publique. On comprend que si, à partir de ce $\vec{c}$, on peut retrouver $\vec{e}$, alors retrouver m est assez facile, puisqu'il nous suffit de calculer la première composante du vecteur $\vec{e}$ modulo 2. Il faut donc rendre difficile le passage de $\vec{c}$ vers $\vec{e}$. Pour ça il faut deux bases différentes, une "bonne" (dont les vecteurs sont quasi orthogonaux) qui sera la clé privée, et une "mauvaise" (dont les vecteurs sont quasi confondus) qui sera la clé publique. En effet, on peut chiffrer avec n'importe quelle base, donc une mauvaise base ne permettant pas de déchiffrer convient pour la clé publique. Pour déchiffrer nous allons utiliser la réduction modulo une base.

Avec une bonne base, en faisant la réduction modulo cette base de $\vec{c_1}$ et $\vec{c_2}$, les vecteurs $\vec{c_1'}$ et $\vec{c_2'}$ correspondent bien à nos vecteurs d'origine (ainsi, $\vec{c'}$ correspond à notre vecteur $\vec{e}$)

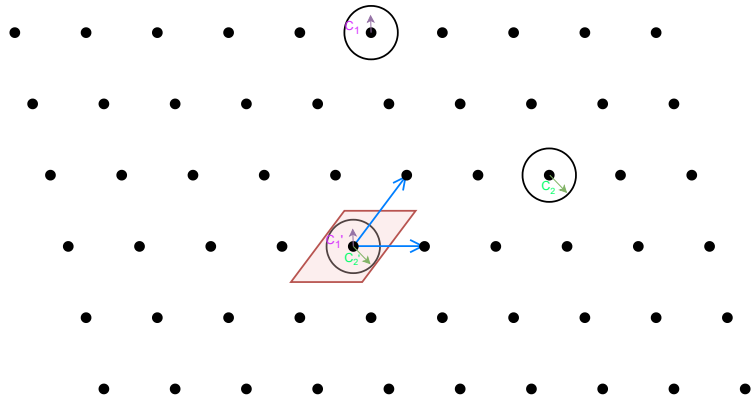


Figure 3: Déchiffrement dans une bonne base

Mais avec une mauvaise base, on se retrouve avec des vecteurs complètement différents, rendant l'utilisation de cette réduction inutilisable. Ce problème est appelé le Bounded Distance Decoding Problem (BDDP).

Théorème 3.2.2. *Bounded Distance Decoding Problem est un problème NP-dur (cf.[6])*

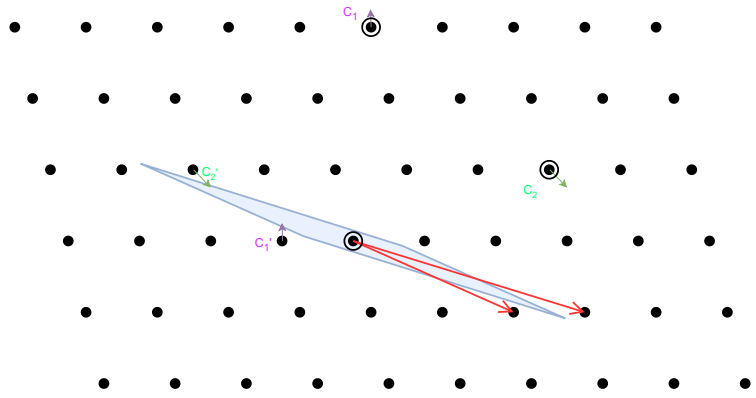


Figure 4: Déchiffrement dans une mauvaise base

On remarque l'importance du cercle contenu dans le parallélogramme, si le rayon de celui-ci est plus petit que la norme du vecteur, alors la réduction modulo la base se passera mal, mais si le rayon est assez grand, on pourra déchiffrer. On appelle ce rayon le rayon de déchiffrement.

Définition 3.2.3. *Soit Λ un réseau de $\mathbb{R}^N$ ayant pour base B. On appelle rayon de déchiffrement $r_{dec}(B)$ le rayon du plus grand cercle contenu dans $\mathcal{P}(B)$*

$$r_{dec}(B) = max\{r \in \mathbb{R}_+ \mid \forall \vec{x} \in \mathbb{R}^N \ tq \ ||\vec{x}|| \leq r \implies \vec{x} \in \mathcal{P}(B)\}$$

Lorsqu'on construit $\vec{e}$ à partir de m, il est très important que sa norme soit plus petite que le rayon de déchiffrement de la base qui est la clé privée, mais plus grand que le rayon de déchiffrement de la clé publique.

Finalement pour déchiffrer, on procède toujours en deux étapes :

- On utilise la bonne base (clé privée) pour calculer la réduction de $\vec{c}$ modulo cette base pour retrouver $\vec{e}$.
- On retrouve le bon représentant de $\vec{m}$ dans $\vec{e}$ ($\vec{e} \in R/I$) : on réduit modulo 2 les composantes de $\vec{m}$, dont on prend la première composante pour avoir m.

En pratique si B est la clé secrète, alors HNF(B) est la clé publique.

3.3 Schéma Faiblement Homomorphe

Théorème 3.3.1. *Il existe des paramètres pour lesquels le schéma de Gentry est faiblement homomorphe.*

Preuve. Soit m_1 et m_2, deux messages clairs. Soient e_1 et e_2 tels que $e_1 \equiv m_1 \pmod I$ et $e_2 \equiv m_2 \pmod I$. On remarque déjà que (comme I est un idéal), $e_1 + e_2 \equiv m_1 + m_2 \mod I$ et $e_1 \times e_2 \equiv m_1 \times m_2 \mod I$

Soit B une base de J, correspondant à la clé privée. Pour chiffrer m_1 et m_2, on calcule $c_1 = e_1 + j_1$ et $c_2 = e_2 + j_2$ avec $j_1 \in J$ et $j_2 \in J$. Alors :

$$\begin{aligned} c_1 + c_2 &= e_1 + e_2 + j_1 + j_2 \\ &= (e_1 + e_2) + j_3 \text{ avec } j_3 \in J \ \ (j_3 = j_1 + j_2) \\ c_1 \times c_2 &= e_1 \times e_2 + e_1 \times j_2 + e_2 \times j_1 + j_1 \times j_2 \\ &= (e_1 \times e_2) + j_4 \text{ avec } j_4 \in J \ \ (j_4 = e_1 \times j_2 + e_2 \times j_1 + j_1 \times j_2) \end{aligned}$$

En admettant qu'on peut construire un réseau tel que $||e_1 + e_2|| \leq r_{dec}(B)$ et $||e_1 \times e_2|| \leq r_{dec}(B)$ on a :

$$\begin{aligned} ||e_1 + e_2|| \leq r_{dec}(B) &\implies (c_1 + c_2) = (e_1 + e_2) \mod J \\ &\implies ((c_1 + c_2) \mod J) \mod I = (e1 + e2) \mod I = m1 + m2 \\ ||e_1 \times e_2|| \leq r_{dec}(B) &\implies (c_1 \times c_2) = (e_1 \times e_2) \mod J \\ &\implies ((c_1 \times c_2) \mod J) \mod I = (e1 \times e2) \mod I = m1 \times m2 \end{aligned}$$

□

Attention ces égalités ne tiennent que si les normes des vecteurs $(e_1 + e_2)$ et $(e_1 \times e_2)$ ne dépassent pas le rayon de déchiffrement, car sinon l'étape de réduction modulo J nous renverra un tout autre vecteur ! C'est pour cela que ce cryptosystème n'est qu'un cryptosystème faiblement homomorphe ; il ne peut qu'évaluer des fonctions sur des chiffrés n'ayant pas une trop grande profondeur.

On expliquera dans la partie sur l'implémentation comment squash la fonction de déchiffrement, afin de pouvoir l'évaluer à travers le cryptosystème sur le chiffré, et ainsi pouvoir réduire la norme de ce vecteur pour ne plus être limités sur le nombre d'opérations applicables sur un chiffré. En d'autres termes, nous verrons plus tard comment bootstrapper le cryptosystème.

4 Implémentation du Cryptosystème

Dans cette partie, nous présentons notre implémentation d'un SHE, que nous améliorons ensuite en FHE. Nous utilisons la variante de Smart-Vercauteren [4].
Ici, notre polynôme $f_N(X)$ est égal à $X^N + 1$ où N est une puissance de 2. (C'est le $2N$-ième polynôme cyclotomique)

4.1 SHE

4.1.1 Génération de Clé

On a vu dans 3.2.2 que les clés privées et publiques forment respectivement une bonne et une mauvaise base de notre réseau euclidien. Pour générer ces bases, on va choisir un vecteur aléatoire $\vec{v}$ et considérer la base

$$\{\vec{v_i} = \vec{v} \times X^i \ \ mod \ \ f_N(X) \ \mid \ i \in \{0, ..., N-1\}\}$$

La matrice de cette base dans la base canonique $(1, X, X^2, ..., X^{N-1})$ est

$$V = \begin{bmatrix} v_0 & v_1 & v_2 & ... & v_{n-1} \\ -v_{n-1} & v_0 & v_1 & ... & v_{n-2} \\ -v_{n-2} & -v_{n-1} & v_0 & ... & v_{n-3} \\ . & . & . & & . \\ . & . & . & . & . \\ . & . & . & & . \\ -v1 & -v2 & -v3 & ... & v0 \end{bmatrix}$$

Ces vecteurs forment une bonne base. V est alors notre clé privée. Mais comme nous pouvons obtenir tous les vecteurs de la base depuis $\vec{v}$, on ne stockera que ce vecteur dans notre mémoire.

Il nous faut maintenant générer la clé publique, pour cela nous avons un bon candidat : la Forme Normale d'Hermite de notre matrice V car toute matrice possède une unique HNF calculable en temps polynomial. Afin de rendre la clé plus compacte, la variante de Smart-Vercauteren (cf. [4]) nécessite que

$$\mathrm{HNF}(V) = \begin{bmatrix} d & 0 & 0 & 0 & 0 & ... & 0 \\ -[r]_d & 1 & 0 & 0 & 0 & ... & 0 \\ -[r^2]_d & 0 & 1 & 0 & 0 & ... & 0 \\ . & . & . & . & & & . \\ . & . & . & & . & & . \\ . & . & . & & & . & . \\ -[r^{N-1}]_d & 0 & 0 & 0 & 0 & ... & 1 \end{bmatrix}$$

avec $d = det(V)$, $[x]_d = x \mod d$ dans $[-d/2; d/2[$ et r une racine de $f_N(X)$ modulo d. On voit en lisant les lignes de $\mathrm{HNF}(V)$ que ses vecteurs forment une mauvaise base. On peut alors l'utiliser comme clé publique. Il faut maintenant trouver les V ayant cette HNF.

Lemme 4.1.1.1. *Soit une matrice V telle que :*

$$V = \begin{bmatrix} v_0 & v_1 & v_2 & ... & v_{N-1} \\ -v_{N-1} & v_0 & v_1 & ... & v_{N-2} \\ -v_{N-2} & -v_{N-1} & v_0 & ... & v_{N-3} \\ . & . & . & & . \\ . & . & . & . & . \\ . & . & . & & . \\ -v1 & -v2 & -v3 & ... & v0 \end{bmatrix}. \text{ Alors HNF(V)} = \begin{bmatrix} d & 0 & 0 & 0 & ... & 0 \\ -[r]_d & 1 & 0 & 0 & ... & 0 \\ -[r^2]_d & 0 & 1 & 0 & ... & 0 \\ . & . & . & & & . \\ . & . & . & & & . \\ . & . & . & & & . \\ -[r^{N-1}]_d & 0 & 0 & 0 & ... & 1 \end{bmatrix},$$

si et seulement si $\mathcal{L}(V)$ contient un vecteur de la forme $\vec{r} = (-r, 1, 0, ..., 0)$.

Preuve : Nécessite 4.1.1.2 à 4.1.1.4. cf.Annexe 6.1.

Définition 4.1.1.2. *On définit w tel que $w(X) \times v(X) = d' \mod f_N(X)$. avec d' une constante. On note W la matrice associée à w.*

Lemme 4.1.1.3. *W est de la forme* $\begin{bmatrix} w_0 & w_1 & w_2 & ... & w_{N-1} \\ -w_{N-1} & w_0 & w_1 & ... & w_{N-2} \\ -w_{N-2} & -w_{N-1} & w_0 & ... & w_{N-3} \\ . & . & . & & . \\ . & . & . & ... & . \\ . & . & . & & . \\ -w1 & -w2 & -w3 & ... & w0 \end{bmatrix}$ *et $V \times W = d' \times Id$*

Preuve. Comme d' est constante et qu'on est sur $\mathbb{Z}[X]/f_N(X)$, d' est le résultat du produit des termes constants de v et w, mais aussi des termes en X^N dans le résultat du produit. Ainsi $d' = v_0 \times w_0 - \sum_{i=1}^{N-1} v_i w_{N-i}$. En appliquant cela à tout $k < N$, on a $d' = \sum_{i+j=k} v_i w_j - \sum_{i+j=k+N} v_i w_j = 0$. On en déduit que $V \times W = d' \times Id$, i.e :

$$\begin{bmatrix} v_0 & v_1 & v_2 & ... & v_{N-1} \\ -v_{N-1} & v_0 & v_1 & ... & v_{N-2} \\ -v_{N-2} & -v_{N-1} & v_0 & ... & v_{N-3} \\ . & . & . & & . \\ . & . & . & ... & . \\ . & . & . & & . \\ -v1 & -v2 & -v3 & ... & v0 \end{bmatrix} \times \begin{bmatrix} w_0 & w_1 & w_2 & ... & w_{N-1} \\ -w_{N-1} & w_0 & w_1 & ... & w_{N-2} \\ -w_{N-2} & -w_{N-1} & w_0 & ... & w_{N-3} \\ . & . & . & & . \\ . & . & . & ... & . \\ . & . & . & & . \\ -w1 & -w2 & -w3 & ... & w0 \end{bmatrix} = \begin{bmatrix} d' & 0 & 0 & ... & 0 \\ 0 & d' & 0 & ... & 0 \\ 0 & 0 & d' & ... & 0 \\ . & . & . & & . \\ . & . & . & ... & . \\ . & . & . & & . \\ 0 & 0 & 0 & ... & d' \end{bmatrix}$$

□

On a alors une importante propriété pour que le système marche :

Lemme 4.1.1.4. $d = d'$

Preuve. cf.Annexe 6.2. □

Grâce à W, nous allons pouvoir calculer r :

Lemme 4.1.1.5. $\forall i \in \{0, ..., N-2\}$, $r = w_{i+1}/w_i$

Preuve. Comme le vecteur $(-r, 1, 0, ..., 0)$ est dans la base de HNF(V), il est en l'occurrence un vecteur de notre réseau. Alors $\exists \vec{y}$ tel que $\vec{y} \times V = (-r, 1, 0, ..., 0)$. Donc, en multipliant l'égalité par W à droite :

$$\begin{aligned}\vec{y} \times V \times W &= (-r, 1, 0, ..., 0) \times W \\ &= -r \times (w_0, w_1, ..., w_N - 1) + (-w_{N-1}, w_0, ..., w_{N-2}) \\ &= \vec{y} \times d \times Id = d \times \vec{y}\end{aligned}$$

En réduisant modulo d, on obtient : $-r \times (w_0, w_1, ..., w_N - 1) + (-w_{N-1}, w_0, ..., w_{N-2}) = 0 \mod d$. D'où $(-w_{N-1}, w_0, ..., w_{N-2}) = r \times (w_0, w_1, ..., w_N - 1) \mod d$. On peut ainsi trouver $r = w_{i+1}/w_i \mod d$, $i \in \{0, ..., N-2\}$ □

Remarque : On ne peut calculer r qu'à condition que l'un des w_i soit inversible modulo d. Dans le cas où aucun d'entre eux n'est inversible, le choix de $\vec{v}$ n'est pas bon.

En pratique, pour vérifier que $\vec{r}$ est dans $\mathcal{L}(V)$, comme $(-w_{N-1}, w_0, ...) = r \times (w_0, w_1, ...) \mod d$, on a par induction que r vérifie $r^N + 1 = 0 \mod d$.

Pour générer une clé, il faut donc :

- Tirer un vecteur v au hasard.
- Calculer $\vec{r}$.
- Si $\vec{r} \notin \mathcal{L}(V)$, on recommence.
- Sinon, on a un bon candidat pour générer les clés et on stocke v et (d, r) comme étant respectivement la clé privée et la clé publique.

4.1.2 Chiffrement

Pour chiffrer un message m (qui est un bit) avec notre clé publique représentée par le couple (d, r) et un vecteur de bruit $\vec{u} = (u_0, ..., u_{N-1}), u_i \in \{-1, 0, 1\}$, on calcule $\vec{a} = 2\vec{u} + m \times \vec{e_1}$ pour obtenir le chiffré $\vec{c}$:

$$\begin{aligned}\vec{c} &= \vec{a} \mod \vec{B} \\ &= \lceil \vec{a} \times B^{-1} \rfloor \times B \ (cf.\ 3.1)\end{aligned}$$

avec $B^{-1} = \frac{1}{d} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & ... & 0 \\ [r]_d & d & 0 & 0 & 0 & ... & 0 \\ [r^2]_d & 0 & d & 0 & 0 & ... & 0 \\ . & . & . & . & & & . \\ . & . & . & & . & & . \\ . & . & . & & & . & . \\ [r^{N-1}]_d & 0 & 0 & 0 & 0 & ... & d \end{bmatrix}$

Malheureusement, chiffrer de cette manière coûte cher du fait des multiplications matricielles. Nous cherchons donc un moyen moins coûteux de chiffrer.

Lemme 4.1.2.1. $\vec{c} = (c, 0, ..., 0)$ *avec* $c = [a(r)]_d$

Preuve. En posant $\vec{a} = (a_0, ..., a_{N-1})$, on a :
$\lceil \vec{a} \times B^{-1} \rfloor \times B = \lceil (s/d, a_1, a_2, ..., a_{N-1}) \rfloor \times B$ avec $s = \sum_{i=0}^{N-1} a_i [r^i]_d = a(r) \mod d$. Alors :

$$\begin{aligned}\vec{c} &= (\lceil s/d \rfloor, \lceil a_1 \rfloor, ..., \lceil a_{N-1} \rfloor) \times B \\ &= (\frac{[a(r)]_d}{d}, 0, ..., 0) \times B \\ &= ([a(r)]_d, 0, ..., 0)\end{aligned}$$

On en déduit ainsi que $\vec{c}$ est de la forme $(c, 0, ..., 0)$ avec $c = [a(r)]_d$. □

Pour chiffrer un bit m, il suffit donc, au lieu de passer par de grandes multiplications matricielles, de :

- Calculer $\vec{a}$
- Prendre son polynôme associé a et l'évaluer en r.
- Ramener le résultat modulo d dans $[-d/2; d/2[$

4.1.3 Déchiffrement

Soit $\vec{c} = (c, 0, .., 0)$ un chiffré. Nous avions déjà récupéré ce chiffré en effectuant une réduction modulo la mauvaise base. Ainsi pour récupérer le vecteur $\vec{a}$ sur lequel on a appliqué notre mauvais modulo, il suffit de lui appliquer le bon modulo :

$$\begin{aligned}\vec{a} &= \vec{c} \mod V \\ &= \lceil \vec{c} \times V^{-1} \rfloor \times V \\ &= \lceil \vec{c} \times \frac{W}{d} \rfloor \times V\end{aligned}$$

De manière naïve, on trouve ensuite notre clair en calculant $\vec{a} \mod 2$. Mais, tout comme avec le chiffrement, nous devons trouver une solution moins coûteuse pour déchiffrer.

Lemme 4.1.3.1. $\forall i \in \{0, ..., N-1\}$, $[cw_i]_d = mw_i \mod 2$

Preuve. On rappelle que le vecteur $\vec{a}$ est géométriquement un petit vecteur de bruit s'appliquant sur un point de notre réseau euclidien que l'on note $\vec{y}$. On peut alors décomposer notre chiffré $\vec{c} = \vec{y} \times V + \vec{a}$. Alors :

$$\begin{aligned}\vec{a} &= \lceil (\vec{y} \times V + \vec{a}) \times V^{-1} \rfloor \times V \\ &= \lceil \vec{y} + \vec{a} \times V^{-1} \rfloor \times V \\ &= \lceil \vec{a} \times V^{-1} \rfloor \times V \text{ (car } \vec{y} \text{ est a composantes entières)}\end{aligned}$$

Pour que cette dernière égalité soit vraie, il faut que $\vec{a} \times V^{-1} = \lceil \vec{a} \times V^{-1} \rfloor$, ce qui n'est vrai que si toute composante de $\vec{a} \times V^{-1}$ est inférieure à $1/2$ en valeur absolue.
De plus, comme $\vec{a} = \lceil \vec{c} \times V^{-1} \rfloor \times V = (\vec{a} \times V^{-1}) \times V$, on a que

$$\begin{aligned}\lceil \vec{c} \times V^{-1} \rfloor &= \vec{a} \times V^{-1} \\ &= \vec{a} \times \frac{W}{d} \quad (1) \\ &= \lceil \vec{c} \times \frac{W}{d} \rfloor \\ &= \frac{[c \times W]_d}{d} \quad (2)\end{aligned}$$

D'où l'égalité, en multipliant (1) et (2) par d : $[c \times W]_d = \vec{a} \times W$.
Alors d'une part :

$$[c \times W]_d = ([cw_0]_d, ..., [cw_{N-1}]_d)$$

D'autre part:

$$\vec{a} \times W = 2\vec{u} \times W + m \times (w_0, ..., w_{N-1})$$

Donc :

$$\begin{gathered}([cw_0]_d, ..., [cw_{N-1}]_d) = m \times (w_0, ..., w_{N-1}) \mod 2 \text{ i.e.} \\ [cw_i]_d = mw_i \mod 2, \forall i \in \{0, ..., N-1\}\end{gathered}$$

□

Ainsi, pour déchiffrer, il faut :

- Avoir un coefficient impair w_i de W
- Calculer $[cw_i]_d \mod 2$ pour retrouver le clair

4.2 FHE

Dans cette partie nous allons voir comment transformer notre SHE en un FHE.

4.2.1 Pourquoi Squash ?

Actuellement notre cryptosystème est limité en nombre d'opérations effectuables sur nos chiffrés (i.e. la profondeur des fonctions évaluables est limitée), car à chaque opération, la norme de notre vecteur représentant le chiffré grossit et peut devenir plus grande que le rayon de déchiffrement, rendant ce même déchiffrement incorrect. Comme dit précédemment, Gentry a proposé d'utiliser la fonction de déchiffrement de façon homomorphe dans le cryptosystème afin de réduire ce "bruit" qui est la norme du vecteur représentant le chiffré : c'est ce qu'on appelle le rechiffrement (ou recrypt). Ce qu'il faut bien comprendre, c'est la correspondance entre les opérations sur les clairs et les opérations sur les chiffrés :

$$c_1 \text{ XOR } c_2 = [c_1 + c_2]_d$$
$$c_1 \text{ AND } c_2 = [c_1 \times c_2]_d$$

Si l'on veut faire un XOR sur les bits chiffrés dans c_1 et c_2, on effectue une addition modulo d sur les chiffrés. Similairement, si l'on veut faire un AND sur les bits chiffrés dans c_1 et c_2, on effectue une multiplication modulo d sur les chiffrés.

Cependant, cela induit le problème difficile de l'évaluation de cette fonction de déchiffrement au sein de notre cryptosystème. Actuellement, nous sommes limités en nombre d'opérations, surtout au niveau des multiplications ; ce sont en effet les multiplications qui font grossir la norme de nos vecteurs.
Actuellement pour déchiffrer, nous effectuons l'opération suivante :

$$\text{Dec(c)} = [cw_i]_d \mod 2$$

Malheureusement, cette multiplication coûte extrêmement cher. Ici, on peut la voir comme la multiplication de deux entiers, mais comme notre cryptosystème utilise des bits, il faut plutôt la voir comme une multiplication de deux nombres binaires, à effectuer comme à l'école primaire en n'utilisant que les opérations XOR et AND (les deux seules opérations faisables) ; sachant qu'à l'évaluation de cette fonction à travers le cryptosystème, les bits de w_i seront chiffrés (avec w_i qui a un nombre de bits assez grand), rendant cette multiplication vraiment coûteuse au niveau du nombre de multiplications (le nombre de AND). On détaillera plus précisément par la suite comment effectuer cette multiplication dans le FHE final. De plus, la réduction modulo d est également une opération complexe : nous devons donc réduire la complexité et le degré de cette fonction de déchiffrement pour espérer l'évaluer à travers le cryptosystème. C'est ce que Gentry appelle "squasher" le circuit de la fonction de déchiffrement.

4.2.2 "Squasher" la fonction de déchiffrement

Première étape : diviser la clé privée (w_i) en plusieurs entiers. Soit S un grand entier et s un petit entier. On pose $T = \{t_1, t_2, ..., t_S\}$ un ensemble d'entiers de cardinal S généré aléatoirement tel qu'il existe un sous-ensemble I de $\{1, 2, ..., S\}$ de cardinal s vérifiant :

$$\sum_{k \in I} t_k = w_i$$

Une autre façon de réécrire cette somme est de poser la fonction σ définie pour tout $i \in \{1, 2, ..., S\}$ par :

$$\sigma(i) = \begin{cases} 1 & \text{si } i \in I \\ 0 & \text{sinon} \end{cases}$$

On obtient donc :

$$\sum_{k=1}^{S} \sigma(k) t_k = w_i$$

Cette fonction σ devient alors notre nouvelle clé privée (représentée par un vecteur de 0 et de 1), et l'ensemble T est ajouté à la clé publique. Cette phase est appelée splitKey par Gentry.

T sera publique, accessible par tous et contient des entiers permettant de reconstruire la clé privée, pouvant réduire la sécurité du cryptosystème. Cependant en prenant s assez grand et S bien supérieur à s, grâce au théorème suivant, la sécurité est preservée.

Théorème 4.2.1. *Le Sparse Subset Sum Problem (SSSP) est un problème difficile. (cf.[7])*

Deuxième étape : étendre le chiffré. Pour cela, avant de déchiffrer, on effectue l'opération suivante sur le chiffré c : on calcule tous les y_i pour tout $i \in \{1, ..., S\}$ avec $y_i = c \times t_i \mod d$ (cette fois-ci entre [0, d). Cette opération sera utile pour des raisons pratiques. Notons que le résultat final avant le modulo 2, sera de toute façon réduit dans [-d/2, d/2)). Le chiffré devient donc une liste de taille S calculable par celui effectuant les calculs sur le chiffré. On déchiffre alors via :

$$\left[\sum_{i=1}^{S} \sigma(i) y_i\right]_d \mod 2 = \left[\sum_{i=1}^{S} \sigma(i) t_i c\right]_d \mod 2 = [w_i c]_d \mod 2$$
$$= \mathrm{Dec(c)}$$

Toutes ces opérations nous permettent d'un peu réduire le nombre total de multiplications, car $\sigma(i) \times y_i$ coûte beaucoup moins cher comme $\sigma(i)$ est un bit. Mais il nous reste encore 2 problèmes : la somme et la réduction modulo d.

Notation. $\lfloor \ \rceil$ signifiera que l'on arrondit à l'entier le plus proche (à ne pas confondre avec $\lceil \ \rfloor$ signifiant que l'on ne garde que la partie fractionnaire)

Nous allons transformer l'opération de déchiffrement petit à petit pour obtenir une formule plus viable.

$$\begin{aligned}
\mathrm{Dec(c)} &= \left[\sum_{i=1}^{S} \sigma(i) y_i\right]_d \mod 2 \\
&= \left(\sum_{i=1}^{S} \sigma(i) y_i\right) - d \times \left\lfloor \frac{\sum_{i=1}^{S} \sigma(i) y_i}{d} \right\rceil \mod 2 \\
&= \left(\sum_{i=1}^{S} \sigma(i) y_i\right) - d \times \left\lfloor \sum_{i=1}^{S} \sigma(i) \frac{y_i}{d} \right\rceil \mod 2 \\
&= \left(\bigoplus_{i=1}^{S} \sigma(i) \langle y_i \rangle_2\right) \oplus \langle d \rangle_2 \times \left\langle \left\lfloor \sum_{i=1}^{S} \sigma(i) \frac{y_i}{d} \right\rceil \right\rangle_2
\end{aligned}$$

Le membre gauche est très facile à calculer de façon homomorphe, étant uniquement un XOR de AND, de complexité très faible. En revanche, le membre de droite, constitué de fractions, d'une somme et d'un arrondi, pose un problème plus délicat.
Les fractions $\frac{y_i}{d}$ seront tout d'abord approximées par des $z_i \in [0, d)$. En se donnant un paramètre ρ qui indique le nombre de bits après la virgule, on approxime $\frac{y_i}{d}$ par ce z_i, où z_i n'a que ρ bits après la virgule, égaux à ceux de $\frac{y_i}{d}$. Il faut choisir ρ tel que son arrondi fournira le même résultat que si l'approximation n'avait pas eu lieu. Cette approximation est très utile, car avec un paramètre ρ assez petit, l'addition de tous ces termes nous coûtera pas beaucoup.

Lemme 4.2.1. *Avec $\rho = \lceil log_2(s+1) \rceil$, si la distance entre le chiffré c et un point du réseau est inférieur à $\frac{1}{(s+1)}$ du rayon de déchiffrement, alors*

$$\left\lfloor \sum_{i=1}^{S} \sigma(i) \frac{y_i}{d} \right\rceil = \left\lfloor \sum_{i=1}^{S} \sigma(i) z_i \right\rceil$$

Preuve. cf.Annexe 6.3. □

Maintenant que l'on a réglé ces fractions, il y a cette somme qu'il faut gérer. Pour ça, il faut réimplémenter l'algorithme de l'addition d'entiers sous forme de circuit, en n'utilisant que des XOR et des AND.

4.2.3 GSA : Grade-School Addition

L'algorithme d'addition reprend la méthode que l'on apprend en école primaire : on pose nos nombres (en binaire) en lignes et on effectue des XOR avec retenue sur les colonnes que nous numérotons de droite à gauche. Notre système de retenue se fait ici grâce aux polynômes symétriques :

Définition 4.2.1. *Soit $X_1, ..., X_n$ n variables, on définit $e_k(X_1, ..., X_n)$ la somme de tous les k-uplets qu'on peut former avec $X_1, ..., X_n$. Autrement dit :*

$$e_k(X_1, ..., X_n) = \sum_{1 \leq i_1 < ... < i_k \leq n} X_{i_1} ... X_{i_k}$$

Remarque : La retenue qu'une colonne j envoie à une colonne $i > j$ se calcule en évaluant $e_{2^{i-j}}$ avec les éléments de la colonne j.

On peut ainsi effectuer l'addition de la manière suivante:

- On balaye les colonnes de droite à gauche.
- Pour chaque colonne, on calcule le XOR.
- On calcule toutes les retenues que la colonne engendre sur celles à gauche de cette dernière.
- On ajoute ces retenues dans les colonnes en question.
- On passe à la colonne suivante (à gauche).

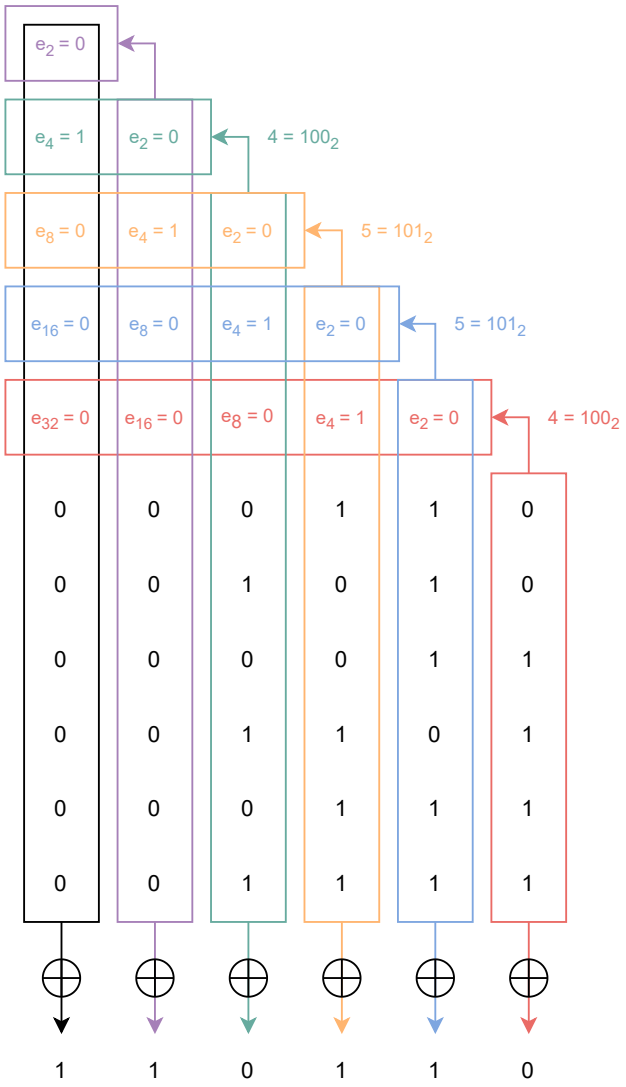


Figure 5: Application du GSA sur un exemple concret

Notre objectif étant d'arrondir la somme, nous n'avons besoin que des ρ bits de précision après la virgule et du chiffre des unités, soit uniquement de $\rho + 1$ colonnes. Une fois la somme calculée, il suffit d'effectuer un XOR entre les deux bits autour de la virgule pour obtenir l'arrondi modulo 2 : comme le bit après la virgule représente 1/2, s'il vaut 1 on peut XOR 1 à l'unité, et sinon ne rien faire de plus.

Lemme 4.2.2. *En prenant $\rho = \lceil log_2(s+1) \rceil$, le nombre total de multiplications est en $O(s \times S)$*

Preuve. Le nombre de multiplications est borné par :

$$S \times 2^{\rho-1} + \sum_{k=1}^{\rho-1} (S+k) \times 2^{\rho-k}$$

Comme $\rho = \lceil log_2(s+1) \rceil$, cette somme est un $O(s \times S)$ □

En pratique, S est très grand, ce qui implique beaucoup de multiplications ; voyons comment réduire ce nombre.

4.2.4 Dernière Optimisation

Maintenant que l'on a réglé le problème de l'addition et de l'arrondi avec le modulo 2, notons que l'on effectue encore beaucoup de multiplications, car on additionne S termes. Le but ici est de réduire ce total d'additions.

Pour faire cela, nous allons changer l'ensemble T, l'ensemble d'entiers de cardinal S contenant un sous-ensemble

de cardinal s tel que la somme de ces éléments nous redonne w_i, et donc notre façon d'encoder la clé privée. Au lieu d'avoir un ensemble T, nous en détiendrons s, de cardinal S. Soit $T_1, ..., T_s$, ces ensembles d'entiers générés aléatoirement. Dans chacun de ces ensembles, un seul élément va permettre de reconstruire w_i.
Plus formellement, on pose pour tout $i \in \{1, ..., s\}$ et pour tout $j \in \{1, ..., S\}$, $t_{i,j}$ le j-ème entier de T_i. On pose la matrice σ de s lignes et S colonnes (qui sera notre nouvelle clé privée) telle que :

$$\forall i \in \{1, ..., s\}, \exists! j \in \{1, ..., S\} \text{ tel que : } \sigma_{i,j} = 1, \text{ sinon } \sigma_{i,j} = 0$$

Chaque ligne de la matrice σ ne contient qu'un seul 1, chaque autre coefficient valant 0. Cette nouvelle matrice vérifie :

$$\sum_{i=1}^{s} \sum_{j=1}^{S} \sigma_{i,j} \times t_{i,j} = w_i$$

Ceci est notre nouvelle façon de diviser la clé.

Pour étendre le chiffré, le procédé est très similaire : on pose pour tout $i \in \{1, ..., s\}$ et pour tout $j \in \{1, ..., S\}$, $y_{i,j} = c \times t_{i,j}$.

Notre nouveau système de déchiffrement découle alors de :

$$\begin{aligned}
\text{Dec(c)} &= \left[\sum_{i=1}^{s} \sum_{j=1}^{S} \sigma_{i,j} y_{i,j} \right]_d \mod 2 \\
&= \left(\sum_{i=1}^{s} \sum_{j=1}^{S} \sigma_{i,j} y_{i,j} \right) - d \times \left\lfloor \frac{\sum_{i=1}^{s} \sum_{j=1}^{S} \sigma_{i,j} y_{i,j}}{d} \right\rfloor \mod 2 \\
&= \left(\sum_{i=1}^{s} \sum_{i=1}^{S} \sigma_{i,j} y_{i,j} \right) - d \times \left\lfloor \sum_{i=1}^{s} \sum_{j=1}^{S} \sigma_{i,j} \frac{y_{i,j}}{d} \right\rfloor \mod 2 \\
&= \left(\bigoplus_{i=1}^{s} \bigoplus_{j=1}^{S} \sigma_{i,j} \langle y_{i,j} \rangle_2 \right) \oplus \langle d \rangle_2 \times \left\langle \left\lfloor \sum_{i=1}^{s} \bigoplus_{j=1}^{S} \sigma_{i,j} \frac{y_{i,j}}{d} \right\rfloor \right\rangle_2
\end{aligned}$$

Certes, ce procédé a augmenté la taille de la clé privée, mais il réduit le nombre d'additions à s au lieu de S : un gain considérable en complexité.

Lemme 4.2.3. *En prenant $\rho = \lceil log_2(s+1) \rceil$, alors le nombre total de multiplications est en $O(s^2)$*

Preuve. Le nombre de multiplications est borné par :

$$s \times 2^{\rho-1} + \sum_{k=1}^{\rho-1} (s+k) \times 2^{\rho-k},$$

le nombre de multiplications est donc en $O(s^2)$. □

4.2.5 Recrypt

Maintenant que l'on a un nouveau circuit pour le déchiffrement, nous allons l'utiliser pour réduire la norme de nos chiffrés, pour ainsi pouvoir faire un infinité d'opérations sur ces chiffrés. Ainsi, pour évaluer de façon homomorphe le circuit de déchiffrement, au lieu d'utiliser la clé de secrète directement, on utilisera la matrice chiffrée à partir de σ, cette matrice faisant partie de la clé publique. Lorsque l'on applique un XOR, on effectue une addition modulo d, lorsque l'on applique un AND, on effectue une multiplication modulo d. Ce comportement est maintenu lorsque l'on exécute le GSA avec les polynômes symétriques. Cette fonction évaluée de façon homomorphe devient notre fonction Recrypt, permettant de réduire la norme nos chiffrés et ainsi d'obtenir un FHE.

5 Expérimentations réalisées

5.1 Génération de clés différentes

En effectuant des tests, on remarque tout d'abord qu'il faut générer des grands coefficients pour v afin appliquer le Recrypt sans "perdre" le bit qui était chiffré. Nous avons donc paramétré nos tests avec :

- la dimension autour de 2^{10}
- les coefficients du polynôme v entre -2^{256} et $+2^{256}$.

Nous avons alors constaté que la génération de clés prenait beaucoup trop de temps. Notre objectif dans cette section est donc de présenter un moyen de réduire ce temps de calcul, avec de la parallélisation et un autre méthode plus efficace de génération.

L'opération coûteuse dans la génération de clé est le déroulement de l'algorithme d'Euclide étendu adapté sur $\mathbb{Z}[X]$, pour calculer w tel que $v \times w = d \mod X^N + 1$. Plus précisément, la taille assez grande des coefficients de v cause des coefficients beaucoup plus gros sur w durant le déroulement de l'algorithme. Pour réduire ces coefficients, nous pouvons essayer de dérouler notre algorithme d'Euclide étendu sur des $\mathbb{F}_p$, p premier. Sur chacun de ces $\mathbb{F}_p$ que l'on traitera en parallèle, nous appliquerons l'algorithme d'Euclide étendu, qui nous permettra ensuite grâce au théorème des restes chinois de retrouver nos d et w modulo le produit des premiers utilisés. Si le produit de ces premiers est plus gros que notre d, nous avons bien retrouvé notre d dans $\mathbb{Z}$, de même pour les coefficients de w, par unicité de la solution modulo le produit des premiers utilisés.

Cette méthode pose cependant un problème : si nous trouvons dans $\mathbb{F}_p$ v et w tels que $v \times w = constante \mod p$, on peut multiplier par l'inverse de cette constante, ainsi $v \times w' = 1 \mod p$, avec $w' = w \times constante^{-1}$, ce qui ne donne aucune information sur d. Nous ne pouvons donc pas directement paralléliser de cette façon ; il faut utiliser des propriétés connues sur d.

Nous savons que d est le déterminant de la matrice V qui a une forme particulière. Le déterminant de V se calcule donc à partir de racines $2N$-ième primitives de l'unité. Soit ξ une racine primitive $2N$-ième de l'unité, nous avons :

$$d = \prod_{k=0}^{N-1} v(\xi^{2k+1})$$

Il faut donc trouver des racines $2N$-ième primitives de l'unité dans des $\mathbb{F}_p$, avec des p premiers bien choisis.

Lemme 5.1.1. *Soit p premier, α un générateur de $\mathbb{F}_p^\times$. Si $p \equiv 1 \mod 2N$, alors $\alpha^{(p-1)/2N}$ est une racine primitive $2N$-ième de l'unité.*

Preuve. L'ordre multiplicatif de α est $(p-1)$, donc l'ordre de $\alpha^{p-1/2N}$ est $2N$. Si il existe un entier $k < 2N$ tel que $(\alpha^{(p-1)/2N})^k \equiv 1 \mod p$, alors l'ordre de α est strictement inférieur à $(p-1)$, ce qui est absurde. □

On a $\phi(p-1)$ générateurs du groupes des inversibles de $\mathbb{F}_p$, ce qui constitue un bon nombre de candidats pour choisir α, donc en pratique, nous tirons aléatoirement un élément de $\mathbb{F}_p$ β, puis nous vérifions que $\beta^{(p-1)/N} \equiv -1 \mod p$, dans ce cas β est bien une racine $2N$-ième primitive de l'unité.

Ainsi nous pouvons calculer $d \mod p$, et puis grâce au théorème des restes chinois retrouver notre d (tant que le produit des premiers utilisé est plus gros que notre d). Ainsi, nous pouvons paralléliser les calculs sur chaque $\mathbb{F}_p$; la réduction modulo p des calculs permet de plus de drastiquement réduire la taille de nos entiers et d'ainsi les accélérer.

Il nous faut maintenant retrouver les coefficients de w : nous pouvons pour cela réutiliser la méthode avec l'algorithme d'Euclide, et comme nous connaissons cette fois-ci $d \mod p$, nous n'avons plus le problème cité plus haut. Une autre méthode pour retrouver les coefficients est la suivante (on utilise une méthode trouvé dans [3], partie 4) :

Lemme 5.1.2. *En posant $g = \sum_{i=0}^{N-1} \prod_{j \neq i} v(\xi^{2j+1})$, alors $w_0 = \frac{g}{N}$.*

Preuve.

$$g = \sum_{i=0}^{N-1} \prod_{j \neq i} v(\xi^{2j+1}) = \sum_{i=0}^{N-1} \frac{\prod_{j=0}^{N-1} v(\xi^{2j+1})}{v(\xi^{2i+1})} = \sum_{i=0}^{N-1} \frac{d}{v(\xi^{2i+1})} = \sum_{i=0}^{N-1} w(\xi^{2i+1})$$

La dernière égalité vient du fait que pour tout $i \in \{1, ..., N-1\}$, ξ^{2i+1} est une racine de $X^N + 1$ (car N est une puissance de 2), et donc comme nous avons que $v(x) \times w(x) + (X^N + 1) \times U(X) = d$, pour un certain $U \in \mathbb{Z}[X]$, pour toutes racines de $X^N + 1$, $v(\xi^{2i+1}) \times w(\xi^{2i+1}) = d$

$$\sum_{i=0}^{N-1} w(\xi^{2i+1}) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} w_j (\xi^{2i+1})^j = \sum_{j=0}^{N-1} w_j \sum_{i=0}^{N-1} (\xi^j)^{2i+1}$$

Pour $j = 0$, nous avons $w_0 \times N$. Pour $j \neq 0$, ξ^{2j} est une racine N-ième de l'unité différente de 1, on a donc :

$$\sum_{i=0}^{N-1} (\xi^j)^{2i+1} = \xi^j \times \sum_{i=0}^{N-1} (\xi^{2j})^i = \xi^j \times \sum_{i=0}^{N-1} (\xi^{2j})^i = \xi^j \times \frac{(\xi^{2j})^N - 1}{\xi^{2j} - 1} = 0$$

Ainsi $g = N \times w_0$. □

On peut retrouver w_i de la même façon, à partir cette fois-ci non pas de $v(x)$, mais de $v(x) \times x^i \mod X^N + 1$. En pratique, nous n'avons besoin que de w_0 et de w_1, du calcul de $r = \frac{w_0}{w_1} \mod d$, et de calculer $w_{i+1} \times r = w_i \mod d$ pour retrouver tout le polynôme, puisque l'on se place dans le cadre où notre polynôme w vérifie cette propriété : si à la fin $r^n + 1 \neq 0 \mod d$, v n'était pas un bon choix, et nous pouvons recommencer en tirant un autre v.

Après quelques tests, cette méthode permet, avec une machine disposant de beaucoup de cœurs, d'accélérer la partie de génération de clés.

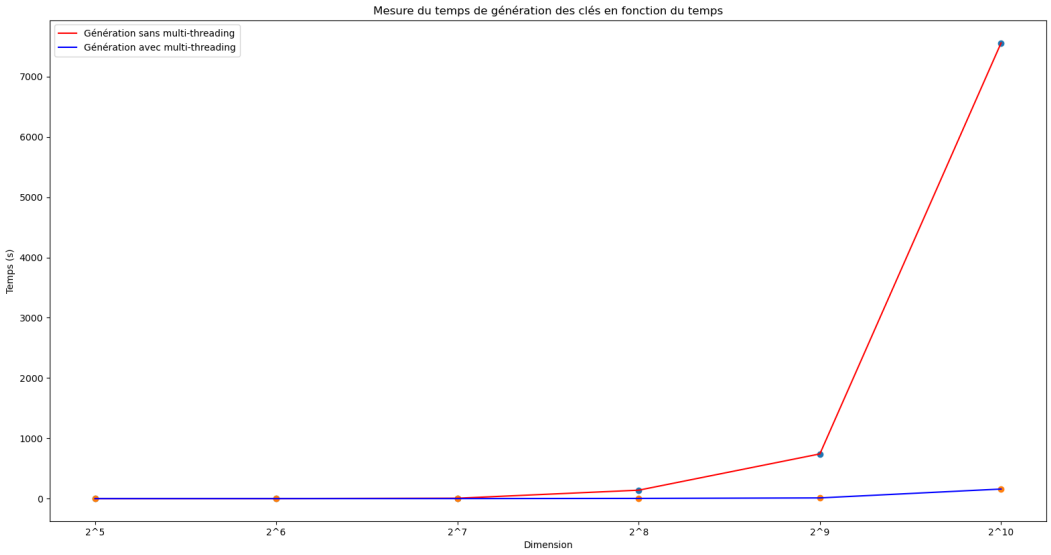


Figure 6: Tests réalisés sur une machine ayant 128 cœurs, avec

Les coefficients de v dans l'intervalle $[-2^{256}, +2^{256}]$. En abscisse, la dimension de l'espace sur lequel les clés ont été générées.

5.2 Chiffrer plusieurs bits

Notre objectif de base ici est de faire de l'arithmétique sur des entiers que nous avons chiffrés. Pour ce faire, il nous faut chiffrer plusieurs bits.

Une première façon à laquelle nous avons pensé est la suivante :
Soit un entier m valant $(b_1, b_2, b_3, ..., b_n)$ écrit en binaire, c'est à dire que $m = \sum_{i=0}^{n} b_{i-1} 2^i$. Originalement, lorsque nous n'avions qu'un bit b, nous lui associions un vecteur $\vec{e} = b \times \vec{e_1} + 2\vec{r}$ où $\vec{r}$ est un vecteur aléatoire

avec pour composantes 0, 1 ou -1. Dans le cas de plusieurs bits, nous pouvons faire la chose suivante : à m nous associons le vecteur $\vec{e} = \sum_{i=1}^{n} b_i \times \vec{e_i} + 2\vec{r}$, avec $\vec{r}$ un vecteur également aléatoire avec pour composantes 0, 1 ou -1. L'étape suivante est similaire, avec $\vec{e}$ qui est également un polynôme, que nous évaluons en r (pas le vecteur, mais le paramètre dans la HNF) ; nous obtenons ainsi un chiffré chiffrant plusieurs bits.

Le souci de cet encodage est que les opération de XOR et de AND sur les chiffrés ne sont pas celles voulues sur les entiers, et sont compliquées à réadapter pour notre utilisation. L'addition va faire un XOR bit à bit, et le AND va faire une opération complètement différente de ce que nous voulons. En effet soient m_1 et m_2 qui en binaire sont respectivement $(b_1, b_2, ..., b_n)$ $(d_1, d_2, ..., d_n)$, la multiplication de leur chiffré va faire la multiplication de leur polynôme binaire associé ($\sum_i b_i X^i$ et $\sum_i d_i X^i$) modulo $X^N + 1$, ce qui n'est pas voulu, mais peut être utile si l'on souhaite faire de l'arithmétique sur des polynômes binaires.

Cette solution ne se prête donc pas à de l'arithmétique sur des entiers. Nous allons donc, comme avec le recrypt, chiffrer indépendemment les bits d'un entier, puis réécrire les algorithmes d'addition et de multiplication de l'école primaire en manipulant des vecteurs de chiffrés.

5.3 Résultats, statistiques sur notre implémentation

Un problème dont nous avons vite pris conscience en implémentant le cryptosystème est que sans accès à la clé privée, nous ne pouvons pas savoir quand il faut appeler recrypt. Nous devons alors déterminer expérimentalement une valeur à partir de laquelle nous appliquerons un recrypt sur nos chiffrés. Pour cela il nous faut d'abord introduire quelques notions. On notera $||.||$ la norme euclidienne sur $\mathbb{R}^N$. Nous utiliserons la même notation pour la norme d'une matrice, qui est définie comme étant la plus grande norme des vecteurs de la base (dans notre cas ils sont en ligne). Gentry montre dans sa thèse (cf. Lemme 7.6.1 de [1] p.86) l'égalité suivante :

Théorème 5.3.1. *Soit B une base de notre réseau Λ,*

$$r_{dec}(B) = 1/(2 \times ||(B^{-1})^T||)$$

Pour une base B sous forme matricielle avec r_{Dec} le rayon du cercle inclus dans $\mathcal{P}(B)$, nous pouvons répéter l'expérience suivante, en connaissant notre clé privée :

- On prend un bit aléatoire
- On lui applique des opérations (surtout des multiplications)
- On traque la norme du chiffré pour savoir quand il faut recrypt
- On compte le nombre d'opérations qu'il faut en moyenne avant de devoir recrypt

Ceci nous permet d'obtenir expérimentalement notre valeur à partir de laquelle recrypt nos chiffrés, valeur qui dictera le comportement du tiers ne pouvant pas traquer la norme (car ne connaissant pas la clé privée).

Voici ce que nous obtenons :

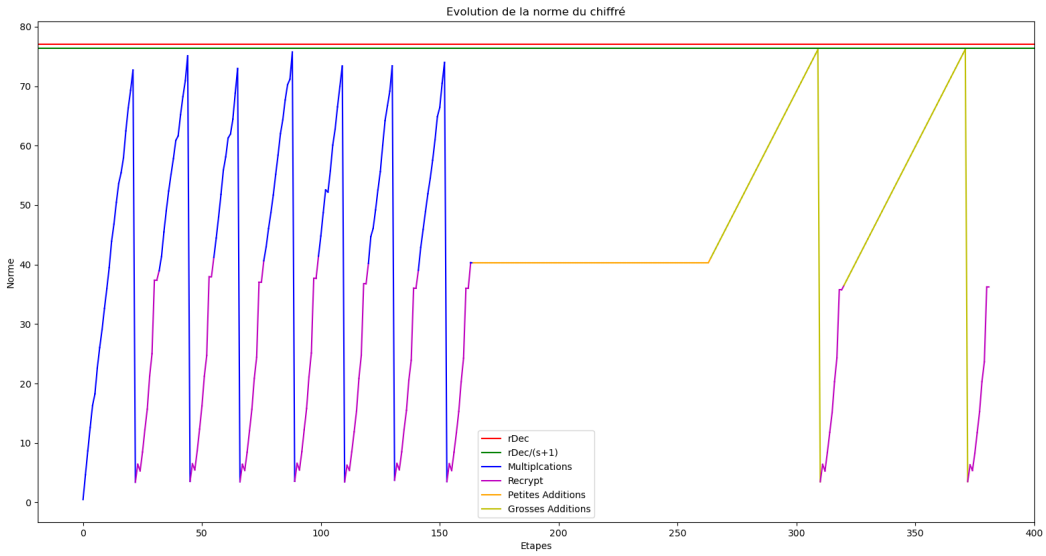


Figure 7: Paramètres standard

L'axe des ordonnées n'est pas exactement la valeur de la norme du chiffré, mais le logarithme en base 10 de celle-ci ; ceci permet un graphe plus explicite. En rouge apparaît la valeur du rayon de déchiffrement, et en vert celle du rayon de déchiffrement divisé par (s + 1). En effet, on rappelle que pour que notre fonction de rechiffrement fonctionne, il faut que la norme du chiffré soit inférieure au rayon de déchiffrement divisé par (s + 1).

En bleu est représentée l'évolution de la norme du chiffré lors d'une multiplication avec un chiffré de norme très petite : on remarque que la norme grandit très vite. En magenta, on peut suivre la création d'un nouveau chiffré à partir de presque 0 lors du rechiffrement. Cela permet d'observer qu'à la fin de la phase rechiffrement, on obtient bien un chiffré de norme plus basse qu'au début, ce qui nous confirme l'obtention d'un FHE. Remarquons que si la courbe en magenta dépassait la ligne verte, alors cela invaliderait notre système en tant que FHE valide. La partie en orange représente la phase d'addition : la somme de plusieurs chiffrés de norme très petites (de même norme que lorsqu'on faisait nos multiplications) s'avère être que de norme quasi constante. C'est bien les multiplications qui font augmenter la norme. Enfin, la partie en jaune modélise des additions avec des chiffrés de plus grosse norme, concrètement le chiffré est sommé à lui-même.

Ce graphique a été obtenu avec pour paramètres :

- Dimension : 2^6
- Taille des coefficients de v comprise entre -2^{256} et $+2^{256}$
- $s = 15$

Le nombre de multiplications minimum avant qu'il ne faille effectuer un rechiffrement est de 61. En augmentant la dimension, ce nombre ne change pas. Cependant en augmentant la taille des coefficients de v, en prenant l'intervalle $\left[-2^{512}, +2^{512}\right]$, on passe à 186 multiplications.

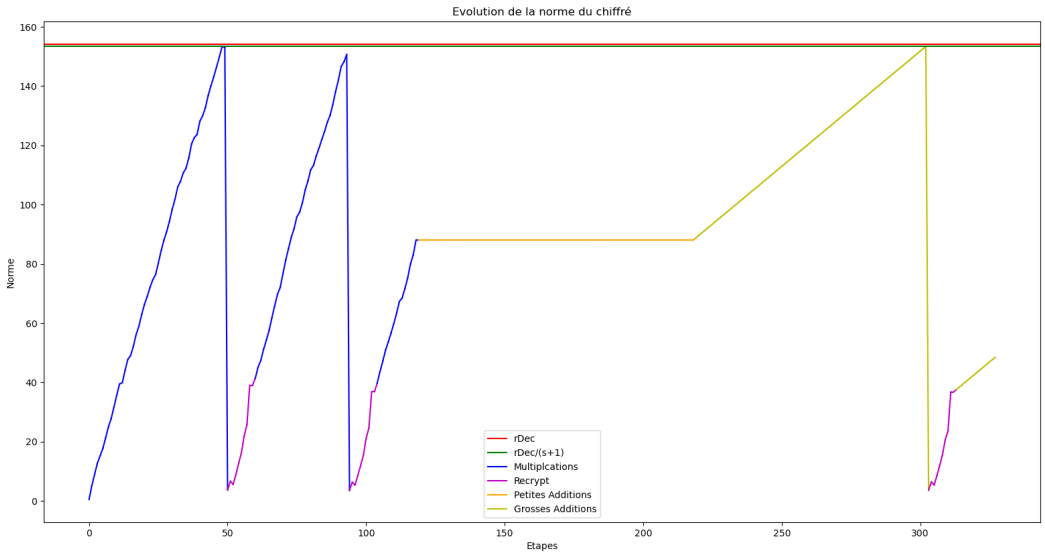


Figure 8: Taille des coefficients de v augmenté

Sur le graphe, on a peut être l'impression que la norme après le rechiffrement est plus petite, mais c'est le rayon de déchiffrement qui a doublé. Cela paraît cohérent, puisqu'augmenter la taille des coefficients de v augmente la rayon de déchiffrement sans trop augmenter la profondeur du la fonction de rechiffrement.

Diminuer s permet de réduire la norme du chiffré à la fin du rechiffrement, mais réduire ce paramètre implique également de diminuer la sécurité pour un gain minimal, car le nombre de multiplications obtenu entre chaque rechiffrement est de 80 (contre 61 avant).

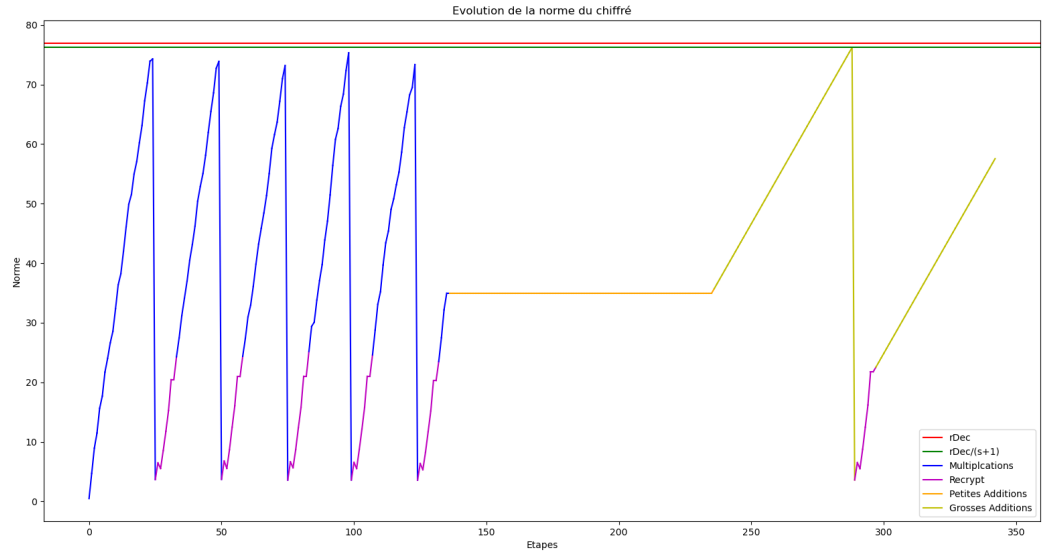


Figure 9: Diminution du paramètre s (subset)

Ici $s = 7$. L'augmenter ne changera pas grand chose non plus, car la norme finale du chiffré dépend majoritairement de la précision de nos flottants pendant le recrypt. Cette précision vaut $\lceil log_2(s+1) \rceil$.

6 Annexes

6.1 Preuve de 4.1.1.1

On rappelle le lemme 4.1.1.1 :

Lemme 6.1.1. *Soit une matrice V telle que :*

$$V = \begin{bmatrix} v_0 & v_1 & v_2 & ... & v_{N-1} \\ -v_{N-1} & v_0 & v_1 & ... & v_{N-2} \\ -v_{N-2} & -v_{N-1} & v_0 & ... & v_{N-3} \\ . & . & . & & . \\ . & . & . & . & . \\ . & . & . & & . \\ -v1 & -v2 & -v3 & ... & v0 \end{bmatrix}. \text{ Alors HNF(V)} = \begin{bmatrix} d & 0 & 0 & 0 & ... & 0 \\ -[r]_d & 1 & 0 & 0 & ... & 0 \\ -[r^2]_d & 0 & 1 & 0 & ... & 0 \\ . & . & . & & & . \\ . & . & . & & & . \\ . & . & . & & & . \\ -[r^{N-1}]_d & 0 & 0 & 0 & ... & 1 \end{bmatrix}$$

si et seulement si $\mathcal{L}(V)$ contient un vecteur de la forme $\vec{r} = (-r, 1, 0, ..., 0)$.

Preuve. ($\Longrightarrow$) Soir B la HNF de V. Comme les lignes de B et de V engendrent le même réseau et que $(-r, 1, 0, ..., 0)$ est une ligne de B, il appartient à $\mathcal{L}(B) = \mathcal{L}(V)$.

($\Longleftarrow$) Tout d'abord, on sait que $d \times \vec{e_1} \in \mathcal{L}(V)$ car $(w_0, ..., w_{N-1}) \times V = (d, 0, ..., 0)$. Supposons qu'il existe un vecteur $\vec{r} = (-r, 1, 0, ..., 0) = -r \times \vec{e_1} + \vec{e_2} \in \mathcal{L}(V)$. Alors sans perdre de généralité, $r \in [-d/2; d/2[$ (on peut soustraire à $\vec{r}$ autant de fois qu'il le faut $d \times \vec{e_1}$ et on aura toujours un vecteur de la même forme appartenant à notre réseau). Montrons par induction que $\forall i \in \{1, ..., N-1\}$, $\vec{r_i} = -r_i \times \vec{e_1} + \vec{e_{i+1}} \in \mathcal{L}(V)$.
Pour $i = 1$, on pose $\vec{r_1} = \vec{r} \in \mathcal{L}(V)$.
Soit $i \in \{1, ..., N-2\}$. Supposons $\vec{r_i} \in \mathcal{L}(V)$, montrons que $\vec{r_{i+1}} \in \mathcal{L}(V)$.
Comme $\vec{r_i} \in \mathcal{L}(V)$ et que $\mathcal{L}(V)$ est stable par rotation (ou multiplication par X : c'est en fait un shift), on a que $-r_i\vec{e_2} + \vec{e_{i+2}} \in \mathcal{L}(V)$. Alors :

$$\begin{aligned} -r_i\vec{e_2} + \vec{e_{i+2}} + r_i\vec{r} &= -r_i\vec{e_2} + \vec{e_{i+2}} - (r_i \times r)\vec{e_1} + r_i\vec{e_2} \\ &= -(r_i \times r)\vec{e_1} + \vec{e_{i+2}} \end{aligned}$$

En réduisant modulo d, dans $[-d/2; d/2[$ on a $[-(r_i \times r)]_d\vec{e_1} + \vec{e_{i+2}} = -[r^{i+1}]_d\vec{e_1} + \vec{e_{i+2}} = \vec{r_{i+1}} \in \mathcal{L}(V)$. Ainsi, on construit la matrice

$$B = \begin{bmatrix} d & 0 & 0 & 0 & 0 & ... & 0 \\ -[r_1]_d & 1 & 0 & 0 & 0 & ... & 0 \\ -[r_2]_d & 0 & 1 & 0 & 0 & ... & 0 \\ . & . & . & . & & & . \\ . & . & . & & . & & . \\ . & . & . & & & . & . \\ -[r_{N-1}]_d & 0 & 0 & 0 & 0 & ... & 1 \end{bmatrix}$$

qui engendre un sous-réseau de V, mais comme $det(B) = det(V)$, $\mathcal{L}(B) = \mathcal{L}(V)$. De plus B est bien la HNF de V car B a la bonne forme, la HNF est unique et on n'a effectué que des opérations élémentaires sur les lignes pour l'obtenir. □

6.2 Preuve de 4.1.1.4

On rappelle le lemme 4.1.1.4 :

Lemme 6.2.1. $d = d'$

Preuve. La preuve est conséquence directe des deux lemmes suivants. □

Lemme 6.2.2. $det(V) = \prod_{\xi_k} v(\xi_k)$

Preuve. En s'inspirant des matrices circulantes, V est diagonalisable dans $\mathbb{C}$. En posant :

$$J = \begin{bmatrix} 0 & 1 & 0 & ... & 0 \\ 0 & 0 & 1 & ... & 0 \\ . & . & . & & \\ . & . & . & ... & \\ . & . & . & & \\ 0 & 0 & 0 & ... & 1 \\ -1 & 0 & 0 & ... & 0 \end{bmatrix}$$

on constate que que V est un polynôme en J de degré $N-1$. De plus X^N+1 est le polynôme minimal annulateur de J, il est irréductible car c'est le $2N$-ième polynôme cyclotomique et il est égal au produit des monômes annulateurs des racines primitives $2N$-ième de l'unité. Soit alors ξ_k une des racines $2N$-ième de l'unité.

Le vecteur est $X_k = \begin{bmatrix} 1 \\ \xi_k \\ \xi_k^2 \\ ... \\ \xi_k^{N-1} \end{bmatrix}$ est un vecteur propre de J associé à la valeur propre ξ_k. Si l'on prend tous les vecteurs propres associés à des valeurs propres distinctes, on obtient une base de $\mathbb{C}^N$. Comme V est un polynôme en J, il est aussi diagonalisable sur les mêmes vecteurs propres que J. Ses valeurs propres sont alors :

$$\lambda_k = \sum_{j=0}^{N-1} v_i \xi_k^j = v(\xi_k)$$

Comme le déterminant est égal au produit des valeurs propres, $d = det(V) = \prod_{\xi_k} v(\xi_k)$. □

Lemme 6.2.3. $d' = \prod_{\xi_k} v(\xi_k)$

Preuve. Par définition $d' = Res(v, f_N)$. On peut exprimer f_N en fonctions des ses racines ξ_j, quitte à se placer dans une extension K du corps des fraction de notre anneau.

$$f_N = \prod_j (X - \xi_j)$$

[5] nous donne que :

$$d' = Res(v, f_N) = \prod_j v(\xi_j)$$

□

6.3 Preuve de 4.2.1

On rappelle le lemme 4.2.1 :

Lemme 6.3.1. *En prenant $\rho = \lceil log_2(s+1) \rceil$, si la distance entre le chiffré c et un point du réseau est inférieure à 1/(s + 1) du rayon de déchiffrement alors*

$$\left\lfloor \sum_{i=1}^{S} \sigma(i) \frac{y_i}{d} \right\rceil = \left\lfloor \sum_{i=1}^{S} \sigma(i) z_i \right\rceil$$

Preuve. On remarque d'abord que $|z_i - \frac{y_i}{d}| \leq 2^{-(\rho+1)}$. Comme dans $\sum_i \sigma(i) \frac{y_i}{d}$ il n'y a que s termes (car seulement s des $\sigma(i)$ sont non nuls) alors :

$$\left| \sum_i \sigma(i) \frac{y_i}{d} - \sum_i \sigma(i) z_i \right| \leq \frac{s}{2^{-(\rho+1)}}$$

En prenant $\rho = \lceil \log_2(s+1) \rceil$. On a alors que :

$$\left| \sum_i \sigma(i) \frac{y_i}{d} - \sum_i \sigma(i) z_i \right| \leq \frac{s}{2^{-(\rho+1)}} \leq \frac{s}{2(s+1)}$$

Maintenant si nous gardons, dans ce cryptosystème, la distance entre le chiffré c et un point du réseau inférieur à $\frac{1}{s+1}$ du rayon de déchiffrement (ce qui est possible si notre cryptosystème est bootstrappable), alors :

$$\begin{aligned}
[w_i c]_d &< \frac{d}{2(s+1)} \\
\left[\sum_i \sigma(i) y_i\right]_d &< \frac{d}{2(s+1)} \\
\left|\sum_i \sigma(i) \frac{y_i}{d}\right| &< \frac{1}{2(s+1)}
\end{aligned}$$

Ainsi comme la distance entre $\sum_i \sigma(i)\frac{y_i}{d}$ et l'entier le plus proche est strictement inférieure à $\frac{1}{2(s+1)}$, et comme celle entre $\sum_i \sigma(i) z_i$ et $\sum_i \sigma(i)\frac{y_i}{d}$ est inférieure à $\frac{s}{2(s+1)}$, alors la distance entre $\sum_i \sigma(i) z_i$ et ce même entier est strictement inférieur à $\frac{s}{2(s+1)} + \frac{1}{s+1} = \frac{1}{2}$. Ainsi ces 2 sommes seront arrondies au même entier. □

7 Bibliographie

[1] A fully Homomorphic Encryption Scheme, Craig Gentry, 2009
[2] Implementing Homomorphic Encryption, Valentin Dalibard, 2011
[3] Implementing Gentry's Fully-Homomorphic Encryption Scheme, Craig Gentry & Shai Halevi, 2011
[4] Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes, N.P.Smart & F.Vercauteren
[5] Préparation à l'agrégation - Résultants, Antoine Chambert-Loir
[6] On Bounded Distance Decoding, Unique Shortest Vectors, and the Minimum Distance Problem, Vadim Lyubashevsky & Daniele Micciancio
[7] On the sparse subset sum problem from Gentry-Halevi's implementation of fully homomorphic encryption, Moon Sung Lee