Implémentation d'un Cryptosystème Homomorphe par les Réseaux Euclidiens

Nicolas Quero, Reda Souadi
Encadré par Pr. Pascal Molin
Université de Paris Diderot
5 Rue Thomas Mann, 75013 Paris

Juin 2020

Chapitre 1

Introduction

Le chiffrement homomorphe est un domaine de recherche actif qui a fait de gros progrès cette dernière décennie. Son utilité repose sur un principe simple, permettre à autrui de travailler sur ses données sans pouvoir récupérer d'informations à un quelconque instant et dont seul vous pourrez obtenir le résultat du travail. Les applications imaginables sont variées :
- un moteur de recherche qui vous indiquerait les pages web que vous recherchez sans même savoir quelle suite de mots vous avez écrite - permettre à un service de cloud de traiter vos données sensibles de manière sécurisée et confidentielle
- analyser un texte sans pouvoir lire son contenu
- ...

Commençons par un exemple, supposons qu'un client dispose d'une entrée x et qu'un serveur contient un algorithme qui permet de calculer une fonction f, le client cherche à calculer $f(x)$ sans divulguer d'informations sur x, et de la même façon le serveur ne veut pas donner l'algorithme de calcul de f ou bien l'algorithme de calcul de f est lent donc l'utilisateur n'a pas d'intérêt à l'exécuter sur sa machine.

Une solution à ce problème a été pensée par R. Rivest, L. Adleman, et M. Dertouzos il y a plus de 40 ans dans un papier intitulé *On data banks and privacy homomorphisms*. La solution qu'ils ont appelé Privacy homomorphisms, consiste à chiffrer l'entrée x et d'envoyer le chiffré c au serveur qui va ensuite évaluer la fonction f sur le chiffré c. Le résultat sera renvoyé au client qui va le déchiffrer pour ensuite obtenir $f(x)$. La question qui se pose est de savoir

s'il existe des cryptosystèmes qui permettent un tel traitement de données.

Dans ce même papier il a été observé que dans le cryptosystème RSA qui, pour un message clair $x \in (\mathbb{Z}/n\mathbb{Z})^\times$, est chiffré en $c = x^e$ et est déchiffré en calculant $c^d = x^{ed}$ où $e \times d = 1\,[\varphi(n)]$ donc $c^d = x$. Alors qu'en prenant deux messages $x, y \in (\mathbb{Z}/n\mathbb{Z})^\times$ et leurs chiffrés respectifs $c_1 = x^e$ et $c_2 = y^e$ si on note $c_3 = c_1 \times c_2 = x^e \times y^e = (x \times y)^e$, le déchiffrement de c_3 qui est $c_3^d = (x \times y)^{ed} = x \times y$. Donc RSA permet de faire des produit de chiffrés de façon homomorphe.

On dit que RSA est un cryptosystème partiellement homomorphe, puisqu'il ne peut évaluer que quelques fonctions. En revanche les cryptosystèmes homomorphes qui permettent d'évaluer toutes les fonctions calculables sont appelés des cryptosystèmes totalement homomorphes. Le but de ce projet est de construire et d'implémenter un tel cryptosystème.

1.1 Quelques applications

L'application la plus simple et intuitive du chiffrement homomorphe est de permettre à un utilisateur de faire des calculs dans le cloud avec des données chiffrés, donc par exemple la société qui vend le service cloud ne peut pas espionner le client et ne pourra pas lire les données utilisés pour le calcul ni les résultats du calcul.

Une autre application très intéressante est de permettre aux utilisateurs d'une base de données d'envoyer des requêtes chiffrés et d'avoir des résultats chiffrés sans que le serveur sache quelle est la requête ni quels sont les résultats renvoyés à l'utilisateur. Ceci peut être étendu aux bases de données SQL par exemple [1].

Le chiffrement homomorphe peut aussi être utilisé dans les preuves à divulgation nulle de connaissance.

1.2 Un peu de théorie

Définition 1 (Chiffrement homomorphe). Soit $\mathcal{M}$ l'espace des messages clairs, Ψ l'espace des chiffrés, $\mathcal{C}$ l'espace des circuits évaluables.
Un chiffrement homomorphe à clé publique $\mathcal{E}$ est le quadruple de procedures

(KeyGen$_\mathcal{E}$, Encrypt$_\mathcal{E}$, Decrypt$_\mathcal{E}$, Evaluate$_\mathcal{E}$) tel que :
KeyGen$_\mathcal{E} : \mathbb{N} \to SK \times PK$
Encrypt$_\mathcal{E} : PK \times \mathcal{M} \to \Psi$
Decrypt$_\mathcal{E} : SK \times \Psi \to \mathcal{M}$
Evaluate$_\mathcal{E} : PK \times \mathcal{C} \times \Psi \to \Psi$

Le cryptosystème doit être correcte, soit un message $m \in \mathcal{M}$ et deux chiffrés $\psi_1, \psi_2 \in \Psi$ on doit avoir :

$\text{Decrypt}_\mathcal{E}(sk, \text{Encrypt}_\mathcal{E}(pk, m)) = m$
$\forall f \in \mathcal{C} \text{Decrypt}_\mathcal{E}(Eval_\mathcal{E}(pk, f, \psi_1, \psi_2)) = f(\text{Decrypt}_\mathcal{E}(sk, \psi_1), \text{Decrypt}_\mathcal{E}(sk, \psi_2))$

Il y a aussi une autre condition qu'on appelle la compacité qui doit être respectée par les cryptosystèmes homomorphes, c'est d'exiger que la complexité de déchiffrage d'un chiffré qui a été évalué par une fonction f ne dépende pas de la fonction f, la longueur du chiffré ne doit pas non plus dépendre de f.

1.3 Exemple d'un cryptosystème homomorphe

Ce cryptosystème qui est partiellement homomorphe est dû à Dijk, Gentry, Halevi et Vaikuntanathan, ce cryptosystème est symétrique, basé sur l'arithmétique et supporte à la fois les additions et multiplications.
La clé privé est un grand entier impair p, pour chiffrer un bit b, on tire au hasard deux entiers q et r tel que $|r| \ll p$ on a le chiffré $c = pq + 2r + b$. Pour déchiffrer c on fait deux réductions modulo : $b = (c \bmod p) \bmod 2$.
Soit $c_1 = pq_1 + 2r_1 + b_1$, $c_2 = pq_2 + 2r_2 + b_2$ les chiffrés de b_1, b_2 et soit $c^\times$, c^+ respectivement le produit et l'addition de c_1 et c_2.

$$\begin{aligned} c^+ &= p(q_1 + q_2) + 2(r_1 + r_2) + (b_1 + b_2) \\ &= p(q') + 2r' + (b_1 \oplus b_2) \end{aligned}$$

$r' = r_1 + r_2 + 1$ quand $b_1 = b_2 = 1$ sinon $r' = r_1 + r_2$

$$\begin{aligned} c^\times &= p(q_1 c_2 + c_1 q_2) + 2(b_1 r_2 + b_2 r_1 + 2r_1 r_2) + (b_1 \times b_2) \\ &= q'' p + 2r'' + (b_1 b_2) \end{aligned}$$

Les r représentent un bruit, ces deux opérations contribuent à l'augmentation du bruit et à partir du moment où $2r$ s'approche de p le déchiffrage ne se passe plus correctement. Quoi qu'ils soient les q, r choisis lors du chiffrement il existe un certain nombre d'opérations qu'on ne peut pas dépasser sinon on ne pourra plus déchiffrer. Donc ce cryptosystème n'est pas complètement homomorphe il est dit *Somewhat Homomorphic*.

La sécurité de ce cryptosystème repose sur le problème d'approximation du PGCD qu'on croit difficile pour les bons paramètres. Ce système relativement simple de *Somewhat Homomorphic Encryption* n'est cependant pas aussi performant que d'autres systèmes. La raison principale est qu'il demande une clé publique d'une taille de $O(\lambda^{10})$ (où λ est le paramètre de sécurité) ce qui donnait en pratique une clé publique d'une taille de 2^{46} bits pour satisfaire les conditions de sécurité il y a 10 ans (donc sûrement encore plus aujourd'hui). Ce système est l'un des systèmes homomorphes les plus populaires de par sa simplicité, et malgré son impraticabilité, étant donné que l'on dispose maintenant d'un système permettant de réaliser un certain nombre d'opérations, on peut introduire la notion de *Bootstrapping* .

1.4 Bootstrapping

La définition qu'on a donnée d'un cryptosystème homomorphe n'exige pas que ce dernier soit totalement homomorphe, d'ailleurs l'exemple ci-dessus ne l'est pas à cause de la croissance rapide du bruit en faisant des opérations.

Craig Gentry a introduit dans sa thèse sur les cryptosystèmes homomorphes une technique qu'il a appelé *Bootstrapping*, cette technique consiste à transformer un cryptosystème qui a une capacité homomorphe limitée en un système totalement homomorphe.

Pour cela l'idée est de rechiffrer les chiffrés qui deviennent trop bruités. Cela peut être fait en mettant dans la clé publique un chiffré de la clé privé, ensuite pour rechiffrer c on évalue la procédure de déchiffrage appliquée à c et le chiffré de la clé privé, le résultat de cette évaluation sera un chiffré c' qui chiffre le même message que c, i.e $\text{Decrypt}_{\mathcal{E}}(sk, c') = \text{Decrypt}_{\mathcal{E}}(sk, c)$.

Pour pouvoir faire du Bootstrapping le cryptosystème doit être capable d'éva-

luer sa fonction de déchiffrage augmenté i.e la fonction de déchiffrage composé avec une autre (ou d'autres) fonction, de façon que les compositions de cette version augmenté puisse permettre de calculer toute fonction calculable. Dans le cas d'un cryptosystème homomorphe qui chiffre des bits une version augmenté de la procédure de déchiffrage serait :

$$D^*_{c_1,c_2} = \text{NAND}(\text{Decrypt}(sk_1, c_1), \text{Decrypt}(sk_2, c_2))$$

Le choix d'une porte NAND vient du fait que toute porte logique peut être construite à l'aide d'une combinaison de portes NAND exclusivement.
C'est cette technique qu'on va utiliser dans ce projet pour aboutir à un système complètement homomorphe. Cette technique a aussi été utilisé pour rendre le cryptosystème donné ci-dessus comme exemple totalement homomorphe dans ce papier [2].

1.5 Squashing

Généralement pour effectuer du bootstrapping sur un cryptosystème $\mathcal{E}$ qui est Somewhat Homomorphic, on trouve que la procédure de déchiffrage $\text{Decrypt}_{\mathcal{E}}$ ne fait pas partie de l'ensemble des circuits qui peuvent être évalués par $\mathcal{E}$, c'est à dire que la capacité homomorphe de $\mathcal{E}$ est faible pour évaluer $\text{Decrypt}_{\mathcal{E}}$. C'est là où intervient le *squashing*, c'est le fait de simplifier la procédure de déchiffrage de $\mathcal{E}$ pour qu'elle fasse partie des circuits évaluables par $\mathcal{E}$, et ceci passe souvent par la modification de la procédure de déchiffrage pour qu'on puisse effectuer une partie des calculs nécessaires pour le déchiffrage en clair, comme ça on aura moins de calculs à faire en utilisant la capacité homomorphe du cryptosystème.

Chapitre 2

Prérequis

2.1 Réseaux euclidiens

Définition 2 (Réseau de $\mathbb{R}^n$). Un réseau L de $\mathbb{R}^n$ est un sous groupe discret de $\mathbb{R}^n$ pour l'addition qui engendre un espace vectoriel de dimension n.

Théorème 1. *Soit L un réseau de $\mathbb{R}^n$, il existe une famille de n vecteurs $(b_i)_i$ dans L, tel que tout élément $v \in L$ se admet une décomposition linéaire unique dans cette famille avec des coefficients entiers.*
$\forall v \in L, \exists!(\lambda_1, \ldots, \lambda_n) \in \mathbb{Z}$ tel que $v = \sum_{i=1}^{n} \lambda_i \times b_i$
On appelle la famille $(b_i)_i$ une base de L.

On peut alors définir un réseau L par sa base :

$$L = \mathcal{L}(b_1, \ldots, b_n) = \left\{ \sum_{i=1}^{n} \lambda_i \times b_i, \lambda_i \in \mathbb{Z} \right\}$$

, Si on note $M \in \mathcal{M}_{n,n}(\mathbb{R})$ la matrice qui a comme vecteurs colonnes une base $(b_i)_i$ de L on a :

$$L = \mathcal{L}(M) = \{M \times x, x \in \mathbb{Z}^n\} = \left\{x \times M^t, x \in \mathbb{Z}^n\right\}$$

L'exemple le plus simple d'un réseau de $\mathbb{R}^2$ est $\mathbb{Z}^n$ qui a comme base $(0, 1)$ et $(1, 0)$. Sa matrice génératrice est la matrice identité.

Proposition 1. *Soit M_1, M_2 deux matrices génératrices d'un réseau L de $\mathbb{R}^n$ alors il existe une matrice unimodulaire U de taille n (càd $U \in \mathcal{M}_{n,n}(\mathbb{Z})$ et $|det(U)| = 1$) telle que : $M_1 \times U = M_2$*

Corollaire 1. *Soit L un réseau de $\mathbb{R}^n$ et M_1, M_2 deux matrices génératrices de L alors $|det(M_1)| = |det(M_2)|$. La valeur absolue du déterminant d'une matrice génératrice de L est indépendant de la matrice elle-même. On peut donc définir $det(L) := |det(B)|$ où B est une matrice génératrice de L.*

Définition 3. Soit L un réseau de $\mathbb{R}^n$ et B une matrice génératrice de L le domaine fondamental de L par rapport à B est l'ensemble $P(B)$:

$$P(B) = \left\{ \sum_{i=1}^{n} \lambda_i \times b_i, \lambda_i \in \left[\frac{-1}{2}, \frac{1}{2} \right[\right\}$$

Le volume du domaine fondamental est appelé le covolume de L.

Proposition 2. *Le covolume de L est égal à $det(L)$*

Proposition 3. *Soit L un réseau de $\mathbb{R}^n$, B une base de L et $v \in \mathbb{R}^n$ alors il existe un unique vecteur $e \in P(B)$ tel que $v - e \in L$.*

Soit $v \in \mathbb{R}^n$ et L un réseau de base B, pour calculer l'unique vecteur d'erreur $e \in P(B)$ tel que $v - e \in L$ (noté $v \bmod B$)

$$e = v \bmod B = v - B\lceil B^{-1}v \rfloor$$

$\lceil x \rfloor$ désigne l'entier le plus proche de x en privilégiant l'entier le plus petit.

2.2 Les réseaux idéaux

On s'intéresse aux réseaux idéaux qui sont un cas spécial des réseaux car en plus de la loi d'addition des réseaux ils sont munis d'une loi de multiplication pour laquelle ils sont stables. Ceci nous intéresse pour construire un crypto-système muni de deux lois.

Soit $f \in \mathbb{Z}[X]$ un polynôme irréductible et unitaire, le quotient $R = \mathbb{Z}[X]/(f)$ est un anneau intègre puisque l'idéal (f) est premier. Soit I un idéal de R, si on fait correspondre à chaque $g \in R$ un vecteur constitué des coefficients de g alors l'ensemble de ces vecteurs est un réseau. C'est à dire que si on prend l'image de I par l'application linéaire φ ci-dessous, on a alors $\varphi(I)$ qui est un réseau de $\mathbb{R}^n$:

$$\varphi(g) : R \to \mathbb{Z}^n$$
$$\sum_{i=0}^{n-1} a_i \times X^i \mapsto (a_0, \ldots, a_{n-1})$$

Proposition 4. *Soit $g \in R = \mathbb{Z}[X]/(f)$ non nul, alors la famille*

$$\varphi(X^i \times g(X))_{i\in[0,n-1]}$$

est une famille libre de n éléments.

Démonstration. Soit $(\lambda_0, \ldots, \lambda_{n-1}) \in \mathbb{Z}^n$ tel que : $\sum_{i=0}^{n-1} \lambda_i X^i g(X) = 0$
On a $g(X) \times \left(\sum_{i=0}^{n-1} \lambda_i X^i\right) = 0 \implies g = 0$ ou $\sum_{i=0}^{n-1} \lambda_i X^i = 0$ car l'anneau R est intègre.
g est non nul, alors $\sum_{i=0}^{n-1} \lambda_i X^i = 0 \implies \forall i, \lambda_i = 0$.
Donc la famille est libre. □

Tout élément $h(X) \in I = (g)$ admet une décomposition linéaire dans la famille $\varphi(X^i g(X))_{i\in[0,n-1]}$. Donc $\varphi(I)$ est bien un réseau de $\mathbb{R}^n$ qui a comme base la famille $\varphi(X^i g(X))_{i\in[0,n-1]}$.

I est alors muni et stable par deux opérations, le produit des polynômes qu'il hérite de l'anneau R ainsi que l'addition qui est la loi du réseau $\varphi(I)$

Chapitre 3

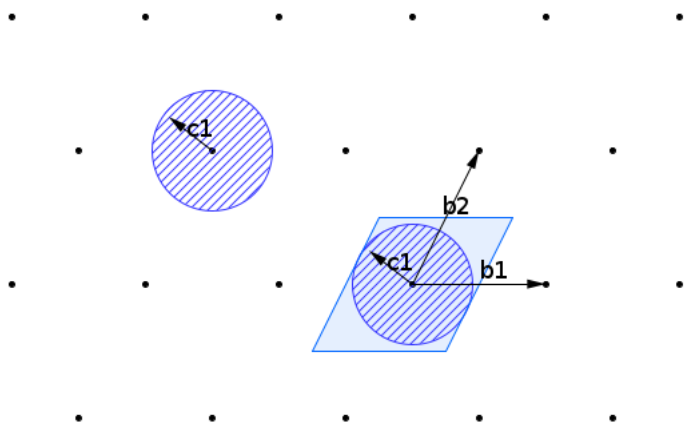
Implémentation d'un système homomorphe au travers des réseaux idéaux

Le cryptosystème qu'on va exposer ici est appelé GGH (Goldreich - Goldwasser - Halevi). Pour cela on aura besoin d'un réseau de $\mathbb{R}^n$ qu'on note L.

3.1 Présentation du système de chiffrement par les réseaux idéaux

Pour crypter un bit $b \in 0, 1$ dans un système homomorphe, nous allons utiliser les propriétés d'anneau d'un réseau idéal pour bénéficier de l'addition et de la multiplication. L'anneau que l'on va considérer ici est $R = \mathbb{Z}[X]/f_n(x)$ où $f_n(x) = x^n + 1$ avec n une puissance de 2.

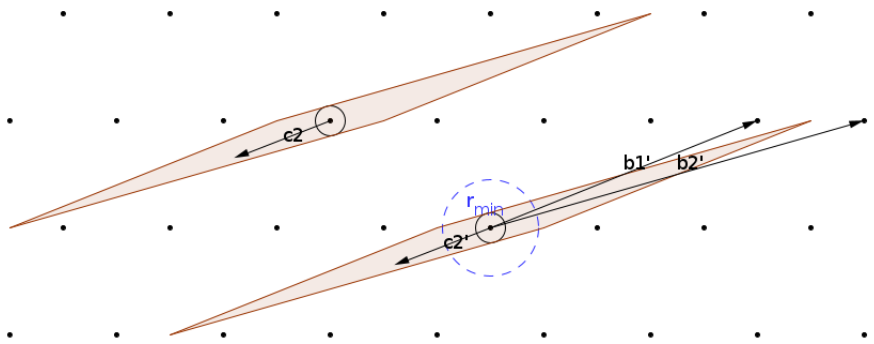
Notre anneau est isomorphe à $\mathbb{Z}_{n-1}[X]$, les éléments de $\mathbb{Z}[X]/f_n(x)$ sont des polynômes de degré $n-1$ donc peuvent être représentés par des vecteurs à n coordonnées. Pour représenter nos vecteurs, on dispose d'un choix infini de bases possibles. Pour cacher une information dans notre système, on va choisir deux bases distinctes, une "bonne" base V qui sera notre clé privée et une "mauvaise" base B qui sera notre clé publique. Pour transmettre une information, on ne va pas transmettre un point du réseau mais un vecteur d'erreur qui éloigne notre message du point du réseau le plus proche, tout en cachant le bit à envoyer dans cette erreur.



Sur le graphique ci-dessus, on a le réseau généré par une bonne base V constituée des vecteurs $b1$ et $b2$. On aperçoit ensuite le parallélogramme de centre $(0,0)$ constituant l'ensemble des points de $\mathbb{R}^2$ dont le point du réseau le plus proche est $(0,0)$. Par exemple, en prenant un point légèrement à droite du parallélogramme, le point du réseau le plus proche de celui-ci sera le point (1,0), l'arrivée de $b1$.

On peut désormais définir l'opération modulo V dans le cadre de notre réseau. Dans $\mathbb{Z}/2\mathbb{Z}$ par exemple, 0, 2, 4, etc. sont équivalents donc peuvent être interchangés à tout instant et conserver leurs propriétés. Ici, tous les points du réseau sont équivalents modulo V et cette opération nous donnera la distance au point du réseau le plus proche. L'opération modulo V sera par la suite définie par $[\cdot]$.

Sur le graphique, on peut apercevoir c_1 deux fois car après réduction modulo V on a recentré c_1 en $(0,0)$ pour ne garder que le vecteur d'information. Enfin, on considère la distance minimale r_{min} telle que l'on peut toujours décrypter un vecteur dont la distance au point du réseau le plus proche est inférieure à r_{min} et ces points sont situés dans les cercles du graphique. En prenant cette distance minimale en compte, on peut désormais voir la différence fondamentale si l'on choisit une mauvaise base B.



Avec la mauvaise base, on constate instantanément que le rayon de décryptage est bien plus petit. Donc après avoir fixé la distance maximale, on ne peut plus décrypter qu'une partie bien plus faible des chiffrés. Cela vient du fait que la bonne base est relativement orthogonale donc que le parallélogramme est plus compact, et dans une base qui presque colinéaire le parallélogramme est bien plus étiré. Donc notre bonne base doit être quasi-orthogonale contrairement à notre mauvaise base.

3.2 Générer les bases

De la même manière que Craig Gentry, pour construire la bonne base V, on commence par choisir un vecteur aléatoire v à n coordonnées qui peut aussi être considéré comme un polynôme $v(x) = \sum_{i=0}^{n-1} v_i x^i$. On engendre ensuite avec v l'idéal $J = (v) = \{v \times x, x \in R\}$. La base de cet idéal que l'on va considérer est la base constituée des $v_i = \{v(x) \times x^i \bmod f_n(x), i \in \{0, ..., n-1\}\}$. En multipliant $v(x)$ par x, on obtient :

$$\begin{aligned} v(x) \times x &= (v_0 + v_1 x + ... + v_{n-1} x^{n-1}) x \bmod f_n(x) \\ &= v_0 x + ... + v_{n-2} x^{n-1} + v_{n-1} x^n \bmod f_n(x) \\ &= -v_{n-1} + v_0 x + ... + v_{n-2} x^{n-1} \bmod f_n(x) \\ &\text{ car } x^n + 1 = 0 \text{ donc } x^n = -1 \end{aligned} \tag{3.1}$$

On déduit une récurrence assez simple qui nous permet d'écrire la matrice des vecteurs de notre bonne base V :

$$V = \begin{pmatrix} v_0 & v_1 & ... & v_{n-1} \\ -v_{n-1} & v_0 & ... & v_{n-2} \\ \vdots & & \ddots & \vdots \\ -v_1 & -v_2 & ... & v_0 \end{pmatrix} \tag{3.2}$$

Cette base est notre clé privée, pour générer la clé publique, on utilise la forme normale de Hermite :

$$B = \begin{pmatrix} d & 0 & 0 & 0 & ... & 0 \\ -r & 1 & 0 & 0 & ... & 0 \\ -[r^2]_d & 0 & 1 & 0 & ... & 0 \\ \vdots & & & \ddots & & \\ & & & & & \\ -[r^{n-1}]_d & 0 & 0 & ... & 0 & 1 \end{pmatrix} \tag{3.3}$$

où $d = det(V)$, $[\cdot]_d$ est la réduction modulo d, et r est une racine de $f_n(x)$ modulo d. On constate que les vecteurs sont quasiment orientés dans les mêmes directions car ils n'ont à chaque fois que 2 coordonnées qui diffèrent en les comparant deux à deux. On a donc une base qui correspond à ce que l'on voulait pour notre clé secrète. De plus, on peut décrire cette base avec r, n et d uniquement ce qui est très pratique. Pour générer ces paramètres, on peut simplement utiliser l'algorithme d'Euclide étendu pour $v(x) = (v_0, .., v_{n-1})$ et $f_n(x) = X^n + 1$, il nous renverra $w(x)$ le polynôme inverse de $v(x)$ qui représente la première colonne de B, et d qui est en réalité le déterminant de V.

3.3 Implémenter le système de chiffrement

Chiffrer un bit

Pour crypter un bit $b \in \{0, 1\}$ avec une clé publique B étant HNF donc représentable par deux entiers r et d. On commence par choisir aléatoirement un vecteur de bruit $u = (u_0, ..., u_{n-1})$. De la même manière que Dalibard, on va choisir un ϵ petit (qu'on va fixer à 15) puis fixer ϵ valeurs de u à 1 ou -1 aléatoirement et on fixe les valeurs restantes à 0.

On pose $a = 2u + be_1 = (2u_0 + b, 2u_1, ..., 2u_{n-1})$.
Puis on pose $c = a \bmod B = [a \times B^{-1}] \times B$
(NB : $[\cdot]$ donne la distance à l'entier le plus proche, $\left[\frac{7}{5}\right] = \frac{2}{5} = \frac{[7]_5}{5}$)
Comme B est sous la forme normale de Hermite, on a

$$B = \begin{pmatrix} d & 0 & 0 & 0 & ... & 0 \\ -r & 1 & 0 & 0 & ... & 0 \\ -[r^2]_d & 0 & 1 & 0 & ... & 0 \\ \vdots & & & \ddots & & \\ -[r^{n-1}]_d & 0 & 0 & ... & 0 & 1 \end{pmatrix} \tag{3.4}$$

et

$$B^{-1} = \frac{1}{d}\begin{pmatrix} 1 & 0 & 0 & 0 & ... & 0 \\ r & d & 0 & 0 & ... & 0 \\ [r^2]_d & 0 & d & 0 & ... & 0 \\ \vdots & & & \ddots & & \\ [r^{n-1}]_d & 0 & 0 & ... & 0 & d \end{pmatrix} \tag{3.5}$$

$a = (a_0, ..., a_{n-1})$ correspond au polynôme $a(x) = \sum_{i=0}^{n-1} a_i x^i$, et par multiplication de matrices on a :

$$\begin{aligned} [a \times B^{-1}] &= [(a_0 + a_1 r + a_2[r^2]_d + ... + a_{n-1}[r^{n-1}]_d,\ a_1 d,\ a_2 d, ...,\ a_{n-1} d) \times \frac{1}{d}] \\ &= [(\frac{s}{d}, a_1, ..., a_{n-1})] \text{ où } s = \sum_{i=0}^{n-1} a_i [r^i]_d \end{aligned} \tag{3.6}$$

On remarque que $s = a(r) \bmod d$, par conséquent :

$$\begin{aligned} c &= ([\frac{s}{d}], [a_1], ..., [a_{n-1}]) \times B \\ &= (\frac{[a(r)]_d}{d}, 0, ..., 0) \times B \text{ car } a_i \in \{-2, 0, 2\}\ \forall i > 0 \text{ donc } a_i \text{ est entier} \rightarrow [a_i] = 0 \\ &= ([a(r)]_d, 0, ..., 0) \text{ par multiplication par B} \end{aligned} \tag{3.7}$$

Maintenant, le chiffré c peut être représenté par :

$$a(r) \bmod d = [b + 2\sum_{i=0}^{n-1} u_i r^i]_d$$

Déchiffrer un bit

On considère un chiffré $c = (c_0, 0, ..., 0)$ comme défini juste avant. Nous devons faire $a = c \bmod V = [c \times V^{-1}]$ où V est la clé secrète soit la bonne base (Rappel : $a = 2u + be_1$).

On rappelle que

$$V = \begin{pmatrix} v_0 & v_1 & ... & v_{n-1} \\ -v_{n-1} & v_0 & ... & v_{n-2} \\ \vdots & & \ddots & \vdots \\ -v_1 & -v_2 & ... & v_0 \end{pmatrix} \tag{3.8}$$

et alors

$$V^{-1} = \frac{1}{d} W \text{ où } W = \begin{pmatrix} w_0 & w_1 & ... & w_{n-1} \\ -w_{n-1} & w_0 & ... & w_{n-2} \\ \vdots & & \ddots & \vdots \\ -w_1 & -w_2 & ... & w_0 \end{pmatrix} \tag{3.9}$$

donc $a = c \bmod V = [c \times \frac{W}{d}] \times V$ où a est le vecteur d'erreur à retrouver.
$c = y \times V + a$ car c est proche d'un point du réseau engendré par V donc ce point le plus proche est une combinaison linéaire $y \in \mathcal{L}(V)$ et le vecteur d'erreur contenant l'information est l'éloignement à $y \times V$.

alors

$$\begin{aligned} a &= [c \times \frac{W}{d}] \times V \\ &= [y \times V \times \frac{W}{d} + a \times \frac{W}{d}] \times V \\ &= [y + a \times \frac{W}{d}] \times V \end{aligned} \tag{3.10}$$

et comme $y \in \mathcal{L}(V)$, sa distance à un point du réseau est nulle : $[y] = 0$

par conséquent, $a = [a \times \frac{W}{d}] \times V$.

Enfin, on a que $[a \times \frac{W}{d}] = (a \times \frac{W}{d})$ tant que les coordonnées de $a \times \frac{W}{d}$ ont toutes une valeur absolue $< \frac{1}{2}$. La raison est que $[a] = a \bmod V = a$. Si l'une des coordonnées est supérieure à $\frac{1}{2}$ cela impliquerait que le point le plus proche serait dans la direction du vecteur de base associé à cette coordonnée donc a ne serait pas correctement réduit et on n'aurait plus l'égalité $[a \times \frac{W}{d}] = (a \times \frac{W}{d})$.

Donc $a = (a \times \frac{W}{d}) \times V$

On rappelle que $a = [c \times \frac{W}{d}] \times V$

Donc $[c \times \frac{W}{d}] = a \times \frac{W}{d} \iff \frac{[c \times W]_d}{d} = a \times \frac{W}{d} \iff [c \times W]_d = a \times W$

Et $c = (c_0, 0, ..., 0)$

Donc $[c \times W]_d = [c_0 \cdot (w_0, ..., w_{n-1})]_d = ([c_0 w_0]_d, ..., [c_0 w_{n-1}]_d)$

De plus, $a \times W = 2u \times W + be_1 \times W = 2u \times W + b \cdot (w_0, ..., w_{n-1})$

Donc $([c_0 w_0]_d, ..., [c_0 w_{n-1}]_d) = b \cdot (w_0, ..., w_{n-1})$

Enfin $\forall i, [c_0 w_i]_d] = b \cdot w_i \bmod 2$

On peut alors garder une unique coordonnée impaire w_i dans w et décrypter :

$$b = [c_0 w_i]_d \bmod 2$$

3.4 Additionner et multiplier des chiffrés dans ce système

En prenant deux chiffrés $c_1 = y_1 + a_1$ et $c_2 = y_2 + a_2$.
La somme des deux donne un vecteur $c_+ = y_+ + a_+$ où $y_+ = y_1 + y_2 \in \mathcal{L}(V)$ et $a_+ = a_1 + a_2$ qui est la somme des vecteurs d'erreur contenant les bits d'information. La somme de bits est traduite par l'opération logique XOR, $[c_1 + c_2]_d = c_1$ XOR c_2
La multiplication des deux donne le vecteur $c_\times = y_\times + a_\times$ où $y_\times = y_1 y_2 + y_1 a_2 + y_2 a_1$ car on rappelle que nous sommes dans un idéal donc le produit d'un élément du réseau par un autre élément est un élément du réseau. Et $a_\times = a_1 a_2$ le produit des vecteurs d'erreur contenant les bits. La multiplication de bits est traduite par l'opération logique AND, $[c_1 \times c_2]_d = c_1$ AND c_2.
Le problème de ce système est que le vecteur d'erreur croît à mesure que l'on réalise ces opérations. On finira donc par obtenir des vecteurs d'information hors du rayon de décryptage que l'on a vu en 3.1 donc l'information sera perdue.
Pour obtenir une solution à ce problème, Craig Gentry a utilisé la technique du *Bootstrapping* introduite en 1.4, seulement jusqu'ici, la méthode de décryptage est trop lourde pour pouvoir décrypter et recrypter sans ajouter trop de bruit.

3.4.1 Le "Squashing" appliqué à notre fonction de décryptage

Pour améliorer la fonction de décryptage, on va ajouter à la clé publique S ($\approx n$) entiers $x_1, ..., x_S \in \mathbb{Z}_d$ qui permettent de retrouver la clé privée w.

Parmi ces x_i, on en choisira $s = 15$ tels que la somme de ceux-ci nous donnera w mod d. Les indices i_k avec $k = 1, ..., 15$ de ces x_i choisis seront stockés dans un vecteur σ de taille S qui vaudra 0 partout sauf aux coordonnées i_k. σ a donc un poids de Hamming de s et le produit de σ par x_i vaut x_i si c'est bien un des x_i que l'on considère, et 0 sinon. Donc, $w = \sum_{i=1}^{S} \sigma_i x_i \bmod d$.

On rappelle que jusque-là on décryptait grâce à $b = [cw]_d$, donc on peut pré-calculer les $y_i = \langle cx_i \rangle_d$ (réduction modulo d dans l'intervalle [0, d[pour l'implémentation) et on obtient la formule de décryptage suivante :

$$dec(c) = \sum_{i=1}^{S} [\sigma_i y_i]_d \bmod 2$$

Ensuite, on montre que cette fonction peut être exprimée comme suit :

$$\begin{aligned} dec(c) &= \Big[\sum_{i=1}^{S} \sigma_i y_i \Big]_d \bmod 2 \\ &= \Big(\sum_{i=1}^{S} \sigma_i y_i \Big) - d \Big\lceil \frac{\sum_{i=1}^{S} \sigma_i y_i}{d} \Big\rfloor \bmod 2 \\ &= \Big(\sum_{j=1}^{S} \sigma_i y_i \Big) - d \Big\lceil \sum_{i=1}^{S} \sigma_i \frac{y_i}{d} \Big\rfloor \bmod 2 \end{aligned} \tag{3.11}$$

Comme nous réalisons des additions réduites modulo 2, nous pouvons simplement remplacer les sommes de bits par des XOR et aboutir à

$$dec(c) = \Big(\bigoplus_{i=1}^{S} \sigma_i \langle y_i \rangle_2 \Big) \oplus \Big\langle \Big\lceil \sum_{i=1}^{S} \sigma_i \frac{y_i}{d} \Big\rfloor \Big\rangle_2$$

(d est pair donc d mod 2 = 1)

En considérant un paramètre de précision ρ, on note z_i l'approximation de chaque quotient $\frac{y_i}{d}$ de ρ bits après le 0 (par exemple, 0.101 est une approximation de 3 bits), et $\lceil \cdot \rfloor$ arrondit à l'entier le plus proche. Pour effectuer l'arrondi, il suffit de faire le XOR du bit avant et du bit après le point binaire car 0.1.. est arrondi à 1 (à l'image de 0.5 en base 10), 1.0.. est arrondi à 1, 0.0.. est arrondi à 0 et 1.1.. est arrondi à 0. Donc le XOR de ces deux bits est suffisant pour arrondir.

C.Gentry a prouvé que si on garde le bruit du vecteur suffisamment petit pour que la distance de c au réseau soit inférieure à $1/s + 1$ du rayon de décryptage alors la distance de wc eu plus proche multiple de d est bornée par $d/(s+1)$, donc $dist_{reseau}([wc]_d) = abs\Big(\Big[\sum_{i=1}^{S} \sigma_i y_i \Big]_d \Big) < \frac{d}{2(s+1)}$.

et alors $abs\Big(\Big[\sum_{i=1}^{S} \sigma_i \frac{y_i}{d} \Big] \Big) < \frac{1}{2(s+1)}$.

Intéressons-nous au quotient y_i/d. $y_i \in [0, ..., d-1]$ donc le quotient est compris entre 0 et 1. On admettra que l'on a l'égalité

$$\left\lceil \sum_{i=1}^{S} \sigma_{i,j} \frac{y_{i,j}}{d} \right\rfloor = \left\lceil \sum_{i=1}^{S} \sigma_{i,j} z_{i,j} \right\rfloor.$$

En implémentant les sommes de la manière que nous venons de présenter, on aura besoin d'effectuer environ $s \cdot S$ multiplications pour effectuer cette opération. Aussi, nous devons chiffrer S les bits de σ et donc nous retrouver avec S chiffrés en plus. Pour pallier à ces problèmes, *Gentry* a proposé une variante où il n'y a plus un vecteur σ mais s vecteurs σ_k chacun de poids 1 donc associé à un unique x_k.
Chaque x_k représente un ensemble B_k de S entiers $B_k = \{< x_k \cdot R^i >_d | i \in \{0, ..., S-1\}$ où R est un paramètre du système et $< \cdot >_d$ est la réduction modulo d dans l'intervalle $[0, d-1]$. On fera désormais référence aux éléments de B_k par $x_k(i)$.

Ces gros ensembles B_k sont les indices de la clé w. Pour cela on choisit un unique indice i_k dans chaque B_k tel que $\sum_{k=1}^{s} x_k(i_k) = w \bmod d$. En particulier, cela veut dire que $b = [cw]_d = [\sum c x_k(i_k)]_d$.

On veut désormais créer une nouvelle clé associée à cet indice de manière à ce qu'on soit capable de recrypter notre chiffré tout en pouvant le décrypter à la fin (donc effectuer le bootstrapping). On définit les nouveaux vecteurs contenant S bits chacun : $\sigma_1, ..., \sigma_s$ tels que $\sigma_k(i) = 1$ si $i = i_k$ et 0 sinon. Ces vecteurs contiennent l'emplacement des i_k qui nous permettent de retrouver la clé w et peuvent être cryptés individuellement et on peut revenir à la formule précédente.

Soit un chiffré c, on calcule tous les entiers $y_{i,j} = \langle c x_i(j) \rangle_d$.

Alors, $dec(c) = \left(\bigoplus_{i=1}^{s} \bigoplus_{j=0}^{S-1} \sigma_{i,j} \langle y_{i,j} \rangle_2 \right) \oplus \left\langle \left\lceil \sum_{i=1}^{s} \bigoplus_{j=0}^{S-1} \sigma_{i,j} z_{i,j} \right\rfloor \right\rangle_2$.

A partir de là, Gentry a implémenté la version optimisée de l'addition en passant par les polynômes symétriques pour n'effectuer que $O(s^2)$ multiplications. Il faut aussi crypter les σ_k en des polynômes de degré $log(S)$ et implémenter la manière de retrouver les indices des x_k secrète en multipliant des chiffrés ajoutés à la clé secrète.

3.5 Sécurité du cryptosystème

Pour que un attaquant qui a une clé publique B_{pk} qui est la base HNF du réseau, puisse casser un chiffrer $c = a \bmod B_{pk}$ pour un $a = 2r + be_1 \in \mathbb{R}^n$ où $r \in \{0,1\}^n$ et donc retrouver b, il faut qu'il retrouve a. On sait que il existe un vecteur $v \in L$ tel que $v + a = c$, en effet le vecteur v est le point de L le plus proche de c car la distance euclidienne entre ces deux points en dimension n est inférieure à $\sqrt{2(n+1)}$ ($\|v - c\|_2 \leq \sqrt{2(n-1)+3} < \sqrt{2(n+1)}$) et pour la norme infinie ($\|v\|_\infty = \max_i |v_i|$) la< distance est inférieure à 3. Le problème est alors de trouver le vecteur $v \in L$ le plus proche de c on aura alors $a = c - v$. Ce problème est connu sous le nom de *Closest vector problem (CVP)*, il est NP-difficile et même son approximation par une constante α est NP-difficile [3].

Trouver a peut aussi être vu comme une instance du problème α-*Bounded distance decoding problem (BDDP)* qui pour une base B d'un réseau L et un vecteur c tel que $d(c, L) < \alpha\lambda(L)$ où $\lambda(L)$ est la longueur du plus petit vecteur de L et il faut trouver $v \in L$ le plus proche de c. Il faut savoir que on a toujours une garantie de $\alpha \leq 1/2$, car pour tout vecteur c il ne peut avoir qu'un seul vecteur de L à une distance strictement inférieure à $\lambda(L)/2$. On sait que ce problème est NP-difficile pour $\alpha > 1/\sqrt{2}$, ce pendant la difficulté pour les autres α est encore un problème ouvert. Le calcul de $\lambda(L)$ est un problème connu sous le nom de *Shortest vector problem (SVP)* et il a été prouvé NP-difficile [3].

Une autre façon de casser le chiffré c est de tester tous les vecteurs a possibles et voir celui qui correspond à c. Notons n la dimension de L il y a 2^{n+1} vecteurs a à tester.

Bibliographie

[1] *Tutorials on the Foundations of Cryptography*. Springer, 2017.

[2] Marten van Dijk, Craig Gentry, Shai Halevin, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. https ://eprint.iacr.org/2009/616.pdf.

[3] *Encyclopedia of cryptography and security*. Springer, 2005.

[4] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, 2011.

[5] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme, 2011. https ://eprint.iacr.org/2010/520.pdf.

[6] Alice Silverberg. Fully homomorphic encryption for mathematicians. https ://eprint.iacr.org/2013/250.pdf.

[7] Valentin Dalibard. Implementing homomorphic encryption. https ://www.cl.cam.ac.uk/ ms705/projects/dissertations/2011-vd241-ihe.pdf.