Rapport de projet informatique

Calcul de petites racines d'un polynôme de bas degré à coefficients dans $\mathbb{Z}/n\mathbb{Z}$

CHARTOUNY Maya **NGUYEN Thi Thu Quyen**

Avec Monsieur Micheal Quisquater
Février 2020

Contenus

Avant-Propos

La **cryptographie** est un ensemble de techniques qui permet d'assurer la sécurité des systèmes d'information. En effet, elle doit pouvoir conserver la confidentialité, l'intégrité du message et l'authenticité. Ainsi, elle a pour but de concevoir des systèmes de nature algorithmique visant à assurer la sécurité des communications sur un canal public qui n'est pas totalement sécurisé. Les algorithmes de la cryptographie se divise en deux branches principales: les algorithmes symétriques et les algorithmes asymétriques.
En particulier, le **RSA** (Rivest – Shamir – Adleman) est un cryptosystème asymétrique qui est fortement utilisé pour la transmission de données sécurisée.
Dans le cas où le RSA est utilisé en mode "broadcast" avec un exposant d en lui appliquant une fonction affine (différentes pour chaque envoi) au message il existe des attaques qui permettent de retrouver le message clair. En effet, dans cet article, nous allons nous concentrer sur ce type d'attaque qui revient à calculer une solution d'un polynôme de bas degrés modulo un entier composé.

Nous allons, dans un premier temps, expliquer la théorie de l'algorithme LLL et les méthodes pour la recherche de petite racine d'un polynôme univarié de bas degré à coefficients dans $\mathbb{Z}/n\mathbb{Z}$.
Dans un second temps, nous allons expliquer l'installation et l'utilisation globale de la librairie Flint ainsi que l'implémentation en C de l'algorithme qui permet la recherche de petite racine avec quelques résultats expérimentaux.

Chapitre 1

Théorie

1.1 Contexte

Soit x le message clair, une méthode naïve de chiffrement RSA de x est

$$c \equiv x^e \pmod N$$

où (e, N) est la clé publique.

La question se pose si on peut retrouver x efficacement. En 1996, Coppersmith a prouvé que cela est faisable dans son théorème (Coppersmith (1997)). Ce théorème traite le cas généralisé $p(x) \equiv 0 (\text{mod } N)$ où le $\deg p$ est petit. Plus tard, une nouvelle méthode a été découverte par (Howgrave-Graham (1997)) qui est considérée plus facile à implémenter par rapport à celle de Coppersmith. Ces deux méthodes contribuent aux attaques sur le schéma cryptographique RSA de bas exposants (il suffit de considérer le cas où $p(X) = c - X^e$).
La stratégie des deux méthodes est de trouver un polynôme $r(X) \in \mathbb{Z}[X]$ qui a pour racine x.
La méthode de Coppersmith et la nouvelle méthode (section 1.3) proposent chacune une manière pour construire $r(X)$ en introduisant une famille de polynômes reliés à $p(X)$ et en exploitant des propriétés du réseau euclidien y associé dont le vecteur court (section 1.2).

1.2 Réseaux euclidiens

1.2.1 Notation et définition

Notation: $f = (f_1, ..., f_n) \in \mathbb{R}^n$, la norme de f est :

$$\|f\| = \|f\|_2 = (\sum_{i=1}^{n} f_i^2)^{1/2} =< f, f >$$

où $< \cdot, \cdot >$ désigne le produit scalaire.

Définition 1.2.1. Soient $n \in \mathbb{N}, f_1, ..., f_n \in \mathbb{R}^n$ avec $f_i = (f_{i_1}, ..., f_{i_n})$. Alors,

$$L = \sum_{i=1}^{n} \mathbb{Z} f_i = \{\sum_{i=1}^{n} r_i f_i | r_1, ..., r_n \in \mathbb{Z}\}$$

est le **réseau euclidien** généré par $f_1, ..., f_n$. Si de plus, les f_i sont linéairement indépendants, alors ils forment une **base** de L.
La **norme** de L est $|L| = |\det(f_{i_j})_{i,j=1}^n| \in \mathbb{R}$

Lemme 1.2.1. Soit $N \subseteq M \subseteq \mathbb{R}^n$ deux réseaux euclidiens générés respectivement par $g_1, \cdots, g_n$ et $f_1, \cdots, f_n$ où $f_i = (f_{i_1}, \cdots, f_{i_n})$ et $g_i = (g_{i_1}, \cdots, g_{i_n})$. Alors $\det(f_{i_j})_{1 \leq i,j \leq n}$ divise $\det(g_{i_j})_{1 \leq i,j \leq n}$

Preuve. Pour $1 \leq i, j \leq n, \exists a_{ij} \in \mathbb{Z}$ tel que $g_i = \sum_{1 \leq j \leq n} a_{ij} f_j$, alors $|\det(g_{i_j})| = |\det(a_{ij})|.|\det(f_{i_j})|$. Donc $\det(f_{i_j})_{1 \leq i,j \leq n}$ divise $\det(g_{i_j})_{1 \leq i,j \leq n}$. □

Remarque 1.2.1. La norme de L est indépendante du choix des générateurs. La démonstration est une conséquence du lemme précédent en prenant $N = M$.

1.2.2 Base orthogonale de Gram-Schmidt et base réduite

Définition 1.2.2. Soit L un réseau euclidien et soit $f = (f_1, ..., f_n)$ où $f_i \in \mathbb{R}^n$ pour tout i, une base quelconque du réseau L. On définit f_i^* par la formule suivante:

$$f_i^* = f_i - \sum_{j=1}^{i} \mu_{ij} f_i^*$$

où $\mu_{i,j} = \frac{<f_i, f_j^*>}{\|f_j\|^2}$ pour $1 \leq j \leq i \leq n$.

Alors, $f^* = (f_1^*, ..., f_n^*)$ est appelée **la base orthogonale de Gram-Shcmidt**, notée **GSO**, de la base $f = (f_1, ..., f_n)$.

Remarque 1.2.2. En particulier, on a $f_1^* = f_1$.
De plus, le calcule de la base GSO est en $O(n^3)$ opérations arithmétiques dans $\mathbb{Q}$.

On considère f_i et f_i^* comme étant des vecteurs lignes dans $\mathbb{R}^n$ et on définit les matrices F, F^* et M dans $\mathbb{R}^{n \times n}$ comme suit:

$$F = \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix}, F^* = \begin{pmatrix} f_1^* \\ \vdots \\ f_n^* \end{pmatrix}, M = (\mu_{ij})_{1 \leq i,j \leq n}$$

où $\mu_{ij} = 1$ pour $i \leq n$ et $\mu_{ij} = 0$ pour $1 \leq i < j \leq n$.
Alors M est une matrice triangulaire inférieure avec des uns sur la diagonale. On a donc:

$$F = \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix} = \begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \mu_{n1} & \cdots & 1 \end{pmatrix} \begin{pmatrix} f_1^* \\ \vdots \\ f_n^* \end{pmatrix} = MF^* \tag{1.1}$$

Théorème 1.2.1. Soient $f_1, ..., f_n \in \mathbb{R}^n$ linéairement indépendants du réseau euclidien L, $(f_1^*, ..., f_n^*)$ sa base orthogoanle de Gram-Schmidt et $1 \leq k \leq n$, on a:

1) $\|f_k^*\| \leq \|f_k\|$

2) $f_1^*, ..., f_n^*$ sont deux à deux orthogonales, i.e, $< f_i^*, f_j^* >= 0$ si $i \neq j$

3) $\det \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix} = \det \begin{pmatrix} f_1^* \\ \vdots \\ f_n^* \end{pmatrix} = \prod_{i=1}^n ||f_i^*|| = |L|$

Preuve. Les 3 propriétés sont vérifiées par les définitions et la formule 1.1. □

Théorème 1.2.2. (Inégalité d'Hadamard) Soit $A \in \mathbb{R}^{n \times n}$ une matrice dont les vecteurs lignes $f_1, ..., f_n \in \mathbb{R}^{1 \times n}$ et $B \in \mathbb{R}$ tel que tous les coefficients de A en valeur absolue sont inférieures ou égales à B. Alors

$$|\det A| \leq \|f_1\| \cdots \|f_n\| \leq n^{n/2} B^n$$

Preuve. On assume que $\det A \neq 0$ et que les f_i sont linéairement indépendants.
Soit $(f_1^*, ..., f_n^*)$ la base GSO associée à $(f_1, ..., f_n)$. D'après le théorème précédent on obtient:

$$|\det \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix}| = |\det \begin{pmatrix} f_1^* \\ \vdots \\ f_n^* \end{pmatrix}| = \|f_1^*\| \cdots \|f_n^*\| \leq \|f_1\| \cdots \|f_n\|$$

Finalement, on a $\|f_i\| \leq n^{1/2} B$ pour tout $1 \leq i \leq n$. D'où le résultat. □

Lemme 1.2.2. Soit $L \subset \mathbb{R}^n$ un réseau ayant comme base $(f_1, \cdots, f_n)$ et $(f_1^*, \cdots, f_n^*)$ sa base GSO. Alors pour tout $f \in L \setminus \{0\}$ on a:

$$\|f\| \geq \min\{\|f_1^*\|, \cdots, \|f_n^*\|\}$$

Preuve. Consulter le chapitre 16 von (2013) □

Définition 1.2.3. Un vecteur court est un vecteur dont la norme est petite.

On s'intéreste à trouver un vecteur court dans le réseau euclidien L. Si $(f_1^*, ..., f_n^*)$ en tant que la base GSO de $(f_1, \cdots, f_n)$ est aussi une base pour L, alors d'après le lemme précédent on obtient que l'un des f_i^* est un vecteur court de L.
En général, les f_i^* ne sont pas dans L. Pour remédier à cela, il faut introduire une nouvelle définition qui est celle de la base réduite.

Définition 1.2.4. Soient $f_1, \cdots, f_n \in \mathbb{R}^n$ linéairement indépendants et $(f_1^*, \cdots, f_n^*)$ la base GSO correspondante.
Alors $(f_1, \cdots, f_n)$ est une **base réduite** du réseau euclidien L si:

$$\|f_i^*\|^2 \leq 2\|f_{i+1}^*\|^2 \text{ pour } 1 \leq i < n$$

Théorème 1.2.3. Soit $(f_1, \cdots, f_n)$ une base réduite du réseau $L \subset \mathbb{R}^n$ et $f \in L \setminus \{0\}$, alors:

$$\|f_1\| \leq 2^{(n-1)/2}\|f\|$$

Preuve. D'après la définition de la base réduite on a

$$\|f_1\|^2 = \|f_1^*\|^2 \leq 2\|f_2^*\|^2 \leq 2^2\|f_3\|^2 \leq \cdots \leq 2^{n-1}\|f_n^*\|^2$$

Alors, d'après le lemme 1.2.2

$$\|f\| \geq \min\{\|f_1^*\|, \cdots, \|f_n^*\|\} \geq 2^{-(n-1)/2}\|f_1\|$$

□

Remarque 1.2.3. Les théorème 1.2.2 et 1.2.3 donne des inégalités suivantes comme conséquences directes

$$\|f_1\| \leq 2^{(n-1)/4}(|L|)^{1/n} \tag{1.2}$$
$$\|f_n^*\| \geq 2^{-(n-1)/4}(|L|)^{1/n} \tag{1.3}$$

Preuve. D $||f_1||^{2n} = ||f_1^*||^{2n} \leq ||f_1^*||^2 \cdot 2||f_2^*||^2 \cdots 2^{n-1}||f_n^*|| \leq 2^{\sum_{i=0}^{n-1} i} \prod_{i=1}^{n} ||f_i^*||^2 = 2^{\frac{n(n-1)}{2}}|L|^2$. Alors on a l'inégalité 1.2. On peut obtenir l'inégalité 1.3 d'une façon similaire. □

1.2.3 Algorithme LLL

Nous présentons maintenant un pseudo-code de l'algorithme LLL qui calcule une base réduite d'un réseau euclidien $L \subset \mathbb{Z}^n$ ayant une base arbitraire.

Algorithme 1: Réduction de la base

Input: Vecteurs lignes linéairement indépendants $f_1, \cdots, f_n \in \mathbb{Z}^n$
Output: Une base réduite $(g_1, \cdots, g_n)$ du réseau $L = \sum_{1 \leq i \leq n} \mathbb{Z} f_i \subset \mathbb{Z}^n$

for $i \leftarrow 2$ **to** n **do** $g_i \leftarrow f_i$;
Calculer la GSO $G^*, M \in \mathbb{Q}^{(n \times n)}, i \leftarrow 2$
while $i \leq n$ **do**
  for $j = i-1, i-2, \cdots, 1$ **do** $g_i \leftarrow g_i - \lfloor \mu_{ij} \rceil g_j$; `/* pour mettre à jour la GSO */`
  ;
  if $j > 1$ *and* $\|g_{i-1}^*\|^2 > 2\|g_i^*\|^2$ **then** $i \leftarrow i-1$; `/* échanger` g_i `et` g_{i-1} `et mettre à jour la GSO */`
  ;
  else $i \leftarrow i-1$;
return $g_1, \cdots, g_n$

1.3 Méthodes pour trouver de petites raccines de l'équation modulaire univariée

Théorème 1.3.1. (Coppersmith, 1997) Soit $p(X) \in \mathbb{Z}[X]$ un polynôme unitaire de bas degré k et soit N un entier. Alors on peut trouver efficacement les entiers x tels que $|x| \leq N^{1/k}$ et

$$p(x) = x^k + a_{k-1}x^{k-1} + \cdots + a_0 \equiv 0 \pmod{N} \tag{1.4}$$

Si N est premier?

(1.4) implique que $p(x) = 0$ pour $x \in \mathbb{Z}/N\mathbb{Z}$. Par ailleurs, on a $X^N - X = 0$ pour tout $X \in \mathbb{Z}/N\mathbb{Z}$, donc $x^N - x = 0$. Alors $(X - x)$ est un facteur commun de $p(X)$ et $X^N - X$ dans $\mathbb{Z}/N\mathbb{Z}[X]$. On peut donc trouver $X - x$ en temps polynomial en appliquant l'algorithme de Berlekamp qui cherche le $pgcd(p(X), X^N - X)$.

Si N est composé?

Si N est facile à factoriser, par exemple $N = \prod p_i$. Alors en utilisant la méthode ci-dessus, on peut trouver des x_i tels que $p(x_i) \equiv 0 (\text{mod } p_i)$, ensuite par le théorème du reste chinois on trouvera efficacement $x \equiv x_i (\text{mod } p_i)$ qui est donc la solution de (1.4).

Cependant, dans la cryptographie, N n'est jamais facile à factoriser. Les deux méthodes suivantes utilisent des propriétés des réseaux euclidens spécifiques pour trouver un polynôme $r(X)$ tel que $r(x) = 0$. On cherche donc des solutions de (1.4) parmi les racines entières de $r(X)$.

La nouvelle méthode et la méthode de Coppersmith donne le même polynôme $r(X)$. La raison est dû à la dualité des deux réseaux euclidiens utilisés (Sections 5, 6 de Howgrave-Graham (1997)). Néamoins, la nouvelle méthode est préférable pour l'implémentation. On en verra en prenant un exemple dans la sous-section 1.3.3.

1.3.1 Nouvelle méthode

Remarque 1.3.1. Soit $r(X) = C_k X^d + C_{k-1}X^{d-1} + \cdots + C_0 \in \mathbb{Z}[X]$ et soit $\tilde{X}$ un entier. Alors dans le domaine $|X| \leq \tilde{X}$ on a:

$$\begin{aligned} |r(X)| &\leq |C_d X^d| + |C_{k-1}X^{d-1}| + \cdots + |C_0| \\ &\leq |C_d \tilde{X}^d| + |C_{k-1}\tilde{X}^{d-1}| + \cdots + |C_0| \quad \forall |X| \leq \tilde{X} \end{aligned} \tag{1.5}$$

Soit $h \geq 2$ un entier, $\tilde{X}$ un entier qui sert comme borne pour la solution x. Soit $M_{hk \times hk} = (m_{i,j})_{0 \leq i,j < hk} = (e_{i,j}\tilde{X}^j)_{0 \leq i,j < hk}$ une matrice triangulaire inférieure, i.e $e_{i,j} = 0$ pour $j > i$, les $e_{i,j}$ pour $j \leq i$ satisfont

$$q_i(X) = q_{u,v}(X) := N^{h-1-v}X^u(p(X))^v = \sum_{0 \leq j \leq i} e_{i,j}X^j \quad \text{pour} \begin{cases} v = \lfloor i/k \rfloor \\ u = i - kv \end{cases} \tag{1.6}$$

De plus, on a

$$q_i(x) = q_{u,v}(x) := N^{h-1-v}x^u(p(x))^v \equiv 0 \pmod{N^{h-1}} \tag{1.7}$$

On va construire $r(x)$ comme étant une combinaison linéaire des $q_{u,v}$ en considérant la norme d'un vecteur court du réseau euclidien engendré par les vecteurs lignes de M.

D'abord, il faut vérifier que M soit bien définie. En effet, on a $\deg q_i = \deg q_{u,v} = u + v\deg(p(X)) = u + kv = i$. Alors on a bien vérifié que $q_i(X) = \sum_{0\le j\le i} e_{i,j}X^j$ dans la définition 1.6. Par ailleurs, puisque $p(X)$ est unitaire, $x^u(p(X))^v$ l'est, donc $N^{h-1-v}X^i$ est le monôme dominant de q_i. Donc $m_{i,i} = e_{i,i}\tilde{X}^i = N^{h-1-v}\tilde{X}^i$ et $m_{i,j} = 0$ pour $j > i$. Alors M est une matrice triangulaire inférieure de déterminant

$$\begin{aligned}\det M &= \prod_{i=0}^{hk-1} m_{ii} = N^{\sum_{i=0}^{hk-1}(h-1-\lfloor \frac{i}{k}\rfloor)}\tilde{X}^{\sum_{i=0}^{hk-1}(i)} \\ &= N^{hk(h-1)/2}\tilde{X}^{hk(hk-1)/2}\end{aligned} \tag{1.8}$$

Soit L le réseau euclidien engendré par les vecteurs lignes M_i de M.

$$L = \{y \in \mathbb{Z}^{hk} | y = \sum_{i=0}^{hk-1} n_i M_i, \quad n_i \in \mathbb{Z}\}$$

Soit B la base réduite par l'algorithme LLL de L et b_1 sa première ligne (par LLL, b_1 est un vecteur court). Alors $b_1 = cM$ pour un $c = (c_i)_{1\times hk} \in \mathbb{Z}^{hk}$. Par l'inégalité 1.2 de la section précédente on a en plus la majoration

$$\begin{aligned}||b_1||_2 &\le 2^{(hk-1)/4}|\det L|^{1/(hk)} = 2^{(hk-1)/4}|\det M|^{1/(hk)} \\ &\le 2^{(hk-1)/4}N^{(h-1)/2}\tilde{X}^{(hk-1)/2}\end{aligned} \tag{1.9}$$

Alors

$$\begin{aligned}||b_1||_2 &\ge \frac{1}{\sqrt{hk}}||b_1||_1 = \frac{1}{\sqrt{hk}}||cM||_1 \quad \text{(l'inégalité de Cauchy-Schwarz)} \\ &\ge \frac{1}{\sqrt{hk}}(|\sum_{i=0}^{hk-1} c_i m_{i,0}| + |\sum_{i=0}^{hk-1} c_i m_{i,1}| + \cdots + |\sum_{i=0}^{hk-1} c_i m_{i,hk-1}|) \\ &\ge \frac{1}{\sqrt{hk}}(|\sum_{i=1}^{hk} c_i e_{i,0}| + |(\sum_{i=0}^{hk-1} c_i e_{i,1})\tilde{X}| + \cdots + |(\sum_{i=0}^{hk-1} c_i m_{i,hk-1})\tilde{X}^{hk-1}|)\end{aligned}$$

Prenons

$$r(X) = (\sum_{i=0}^{hk-1} c_i e_{i,0}) + (\sum_{i=0}^{hk-1} c_i e_{i,1})X + \cdots + (\sum_{i=0}^{hk-1} c_i m_{i,hk-1})X^{hk-1} \in \mathbb{Z}[X]$$

. En appliquant la remarque 1.3.1 et la majoration 1.9, on obtient:

$$2^{(hk-1)/4}N^{(h-1)/2}\tilde{X}^{(hk-1)/2} \ge ||b_1||_2 \ge \frac{1}{\sqrt{hk}}|r(x)|, \quad \text{pour } |x| \le \tilde{X}$$

Alors

$$|r(x)| \leq 2^{(hk-1)/4} N^{(h-1)/2} \tilde{X}^{(hk-1)/2} \sqrt{hk} \quad \text{pour } |x| \leq \tilde{X}$$

On veut que

$$|r(x)| \leq N^{h-1} \tag{1.10}$$

Prenons alors

$$\tilde{X} = \tilde{X}(h) = \lfloor (2^{-1/2}(hk)^{-1/(hk-1)}) N^{(h-1)/(hk-1)} \rfloor \tag{1.11}$$

Par ailleurs

$$\begin{aligned}
r(x) &= (\sum_{i=0}^{hk-1} c_i e_{i,0}) + (\sum_{i=0}^{hk-1} c_i e_{i,1})x + \cdots + (\sum_{i=0}^{hk-1} c_i m_{i,hk-1}) x^{hk-1} \\
&= c_0 \sum_{j=0}^{hk-1} e_{0,j} x^j + c_1 \sum_{j=0}^{hk-1} e_{1,j} x^j + \cdots + c_{hk-1} \sum_{j=0}^{hk-1} e_{hk-1,j} x^j \\
&= c_0 e_{0,0} + c_1 \sum_{j=0}^{1} e_{1,j} x^j + \cdots + c_{hk-1} \sum_{j=0}^{hk-1} e_{hk-1,j} x^j, \quad \text{car } e_{i,j} = 0 \text{ pour } j > i \\
&= c_0 q_0(x) + c_1 q_1(x) + \cdots c_{hk-1} q_{hk-1}(x), \quad \text{par 1.6} \\
&= \sum_{i=0}^{hk-1} c_i q_{u,v}(x)
\end{aligned}$$

D'après la définition 1.7, $q_{u,v}(x) \equiv 0 \pmod{N^{h-1}}$, alors

$$r(x) = \sum_{i=0}^{hk-1} c_i q_{u,v}(x) \equiv 0 (\text{mod } N^{h-1}) \tag{1.12}$$

D'après 1.10 et 1.12, on conclut que $r(x) = 0$ pour $|x| \leq \tilde{X}$. Donc x est une racine entière de $r(X)$. Comme on peut trouver toutes racines entières de $r(X)$ par les algorithmes de factorisation de polynômes dans $\mathbb{Z}[X]$ en temps polynomial (comme celles de Zassenhaus et van Hoeij), il reste à vérifier si si qu'elle sont solutions de 1.4.

Remarque 1.3.2. D'après 1.11 on a

$$\tilde{X} \xrightarrow[h \to \infty]{} N^{1/k}$$

Donc il faut préciser que les méthodes concernant cette borne ne trouvent que des solutions du 1.4 jusqu'à $O(N^1/k)$, autremendit des petites racines.

1.3.2 Méthode de Coppersmith

La méthode de Coppersmith utilise un autre réseau euclidien qui est un dual de celui qui est utiliser dans la nouvelle méthode.

Soit $h \geq 2$ un entier, $\tilde{X}$ un entier naturel. Considérons la matrice $\hat{M}^{copp}$ de taille $(2hk - k) \times (2hk - k)$

$$\hat{M}^{copp}_{(2hk-k)\times(2hk-k)} = \left(\begin{array}{c|c} D_{hk \times hk} & A_{hk \times (hk-k)} \\ \hline 0_{(hk-k)\times hk} & D'_{(hk-k)\times(hk-k)} \end{array} \right)$$

où $D_{hk\times hk} = \text{diag }(1, \tilde{X}^{-1}, \cdots, \tilde{X}^{-(hk-1)})$, $D'_{(hk-k)\times(hk-k)} = \text{diag }(N^{\lfloor (k+i)/k \rfloor})_{0\leq i<hk-k}$, et $A_{hk\times(hk-k)} = (a_{i,j})_{\substack{0\leq i<hk,\\ 0\leq j<hk-k}}$ et $a_{i,j}$, où les $a_{i,j}$ satisfonts

$$q_j^{copp}(X) = q_{u,v}^{copp}(X) := X^u(p(X))^v = \sum_{0\leq i\leq j} a_{i,j}X^i \quad \text{pour } \begin{cases} v = \lfloor (k+j)/k \rfloor \\ u = j - kv \end{cases}$$

On constate que $\hat{M}^{copp}$ est une matrice triangulaire supérieur, qui a pour déterminant

$$\begin{aligned} \det(\hat{M}^{copp}) &= \prod_{i=0}^{hk-1} \tilde{X}^{-i} \prod_{i=0}^{hk-k-1} N^{\lfloor (k+i)/k \rfloor} \\ &= N^{hk(h-1)/2}\tilde{X}^{-hk(hk-1)/2} \end{aligned}$$

Par ailleurs, puisque $p(X)$ est unitaire, $q_j(X)$ l'est, donc il apparaît une sous-matrice de diagonale 1 de taille $(hk-k)\times(hk-k)$ dans la matrice $A_{hk\times(hk-k)}$, celle-ci permet de rendre $\hat{M}^{copp}$ à la forme

$$\bar{M}^{copp} = H_1\hat{M}^{copp} = \left(\begin{array}{c|c} M^{copp}_{hk\times hk} & 0_{hk\times(hk-k)} \\ \hline A'_{(hk-k)\times hk} & 1_{(hk-k)\times(hk-k)} \end{array}\right), \quad |\det(H_1)| = 1 \tag{1.13}$$

. Alors

$$|\det(M^{copp})| = |\det(\bar{M}^{copp})| = |\det(\hat{M}^{copp})| = N^{hk(h-1)/2}\tilde{X}^{-hk(hk-1)/2}$$

Posons

$$L^{copp} = \{y \in \mathbb{Z}^{hk} | y = \sum_{i=0}^{hk-1} n_i M_i^{copp}, \quad n_i \in \mathbb{Z}\}$$

le réseau euclidien engendré par les vecteurs lignes M_i^{copp} de M^{copp}. On réduit ensuite $M^{copp}_{hk\times hk}$ à

$$B^{copp}_{hk\times hk} = H_2 M^{copp}_{hk\times hk}, (H_2 \in GL_{hk}(\mathbb{Z})) \tag{1.14}$$

par la réduction de base LLL. Posons $B^*_{hk\times hk}$ (avec des lignes $b_i^*, 0 \leq i \leq hk-1$) la base de L^{copp} après l'orthogonalisation Gram-Schmidt de B^{copp}. D'après l'inégalité 1.3 de la section précédente, on a alors

$$\begin{aligned} ||b^*_{hk-1}||_2 &\geq 2^{-(hk-1)/4}|\det L^{copp}|^{1/(hk)} = 2^{(hk-1)/4}|\det M^{copp}|^{1/(hk)} \\ &\geq 2^{-(hk-1)/4}N^{(h-1)/2}\tilde{X}^{(hk-1)/2} \end{aligned} \tag{1.15}$$

(L^{copp} et L sont en effet duals au sens définit dans l'article Howgrave-Graham (1997).) Puisque $p(x) \equiv 0 \mod N$, il existe y tel que $y = \frac{p(x)}{N} \in \mathbb{Z}$. Considérons le vecteur

$$\begin{aligned} c(x)_{1\times(2hk-k)} &= (1, x, \cdots, x^{hk-1}, -y, -yx, \cdots, -yx^{k-1}, -y^2, -y^2x, \cdots, y^{h-1}x^{k-1}) \\ &= (1, \cdots, x^{hk-1}, -(\frac{p(x)}{N}), -(\frac{p(x)}{N})x, \cdots, -(\frac{p(x)}{N})x^{k-1}, -(\frac{p(x)}{N})^2, \\ &\quad -(\frac{p(x)}{N})^2x, \cdots, (\frac{p(x)}{N})^{h-1}x^{k-1}) \end{aligned} \tag{1.16}$$

qui satisfait

$$c(x)\hat{M}^{copp} = (1, x/\tilde{X}, \cdots, (x/\tilde{X})^{hk-1}, 0, \cdots, 0)_{1\times(2hk-k)} \tag{1.17}$$

Notons $[v]_{hk}$ le vecteur prenant hk premiers composants du $v_{1\times(2hk-k)} = (v_i)$ ayant $v_i = 0$ pour $hk \leq i < 2hk-k$, i.e $[c(x)\hat{M}^{copp}]_{hk} = (1, x/\tilde{X}, \cdots, (x/\tilde{X})^{hk-1})$. Supposons $|x| \leq \tilde{X}$, alors $|x/\tilde{X}| \leq 1$ et

$$\begin{aligned}
\sqrt{hk} &\geq ||c(x)\hat{M}^{copp}|| \quad \text{(l'inégalité de Cauchy-Schwarz)} && (1.18)\\
&= ||c(x)H_1^{-1}\bar{M}^{copp}|| \quad \text{(par 1.13)}\\
&= ||[c(x)H_1^{-1}]_{hk}M^{copp}_{hk\times hk}|| \quad \text{(car } (c(x)H_1^{-1})_{i\geq hk} = 0 \text{ par 1.13 et 1.17)}\\
&= ||[c(x)H_1^{-1}]_{hk}H_2^{-1}B^{copp}_{hk\times hk}|| \quad \text{(par 1.14)}\\
&= ||c'(x)_{1\times hk}B^{copp}_{hk\times hk}|| \quad \text{où } c'(x) = [c(x)H_1^{-1}]_{hk}H_2^{-1} && (1.19)\\
&= ||c''(x)_{1\times hk}B^{*}_{hk\times hk}|| \quad \text{(changement de base)}\\
&\geq ||c''(x)_{hk-1}b^{*}_{hk-1}|| \quad (c''(x)_{hk-1} \in \mathbb{Z})\\
&= |c'(x)_{hk-1}|||b^{*}_{hk-1}|| \quad \text{(car } c'(x)_{hk-1} = c''(x)_{hk-1} \text{ par 1.1)}\\
&\geq |c'(x)_{hk-1}|2^{-(hk-1)/4}N^{(h-1)/2}\tilde{X}^{(hk-1)/2} \quad \text{(par 1.15)}\\
&> |c'(x)_{hk-1}|\sqrt{hk} \quad \text{pour } x \leq \tilde{X} < (2^{-1/2}(hk)^{-1/(hk-1)})N^{(h-1)/(hk-1)}
\end{aligned}$$

Puisque $c'_{hk-1} \in \mathbb{Z}$, cette dernière inégalité implique que $c'_{hk-1} = 0$ pour un bon choix de $\tilde{X}$. D'après 1.19, on a aussi

$$c'(x)_{hk-1} = [c(x)H_1^{-1}]_{hk}((H_2^{-1})^t)_{hk-1} = 0$$

Alors en prenant $r(X) = [c(X)H_1^{-1}]_{hk}((H_2^{-1})^t)_{hk-1} \in \mathbb{Z}[X]$, on a $r(x) = 0$ pour la racine x de 1.4.

1.3.3 Exemple

On traite l'exemple de l'article Howgrave-Graham (1997) pour une illustration plus claire des deux méthodes.

Soient $p(X) = X^2 + 14X + 19 \in \mathbb{Z}[X], N = 35$, alors $k = \deg p = 2$. Choisissons $h = 3$ et $\tilde{X} = \tilde{X}(h) = 2$.

Nouvelle méthode

A l'aide du programme C on obtient des $q_i(X) = q_{u,v}(X) := 35^{3-1-v}X^u(p(X))^v, 0 \leq i < hk = 6, v = \lfloor i/k \rfloor$ et $u = i - kv$

$$\begin{aligned}
q_0 &= q_{0,0} = 1225\\
q_1 &= q_{0,1} = 1225X\\
q_2 &= q_{1,0} = 665 + 490X + 35X^2\\
q_3 &= q_{1,1} = 665X + 490X^2 + 35X^3\\
q_4 &= q_{2,0} = 361 + 532X + 234X^2 + 28X^3 + X^4\\
q_5 &= q_{2,1} = 361X + 532X^2 + 234X^3 + 28X^4 + X^5
\end{aligned}$$

Alors

$$M = \begin{pmatrix} 1225 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1225\tilde{X} & 0 & 0 & 0 & 0 \\ 665 & 490\tilde{X} & 35\tilde{X}^2 & 0 & 0 & 0 \\ 0 & 665\tilde{X} & 490\tilde{X}^2 & 35\tilde{X}^3 & 0 & 0 \\ 361 & 532\tilde{X} & 234\tilde{X}^2 & 28\tilde{X}^3 & \tilde{X}^4 & 0 \\ 0 & 361\tilde{X} & 532\tilde{X}^2 & 234\tilde{X}^3 & 28\tilde{X}^4 & \tilde{X}^5 \end{pmatrix}$$

En réduisant M par LLL on obtient B

$$B = \begin{pmatrix} 3 & 8\tilde{X} & -24\tilde{X}^2 & -8\tilde{X}^3 & -\tilde{X}^4 & 2\tilde{X}^5 \\ 49 & 50\tilde{X} & 0 & 20\tilde{X}^3 & 0 & 2\tilde{X}^5 \\ 115 & -83\tilde{X} & 4\tilde{X}^2 & 13\tilde{X}^3 & 6\tilde{X}^4 & 2\tilde{X}^5 \\ 61 & 16\tilde{X} & 37\tilde{X}^2 & -16\tilde{X}^3 & 3\tilde{X}^4 & 4\tilde{X}^5 \\ 21 & -37\tilde{X} & -14\tilde{X}^2 & 2\tilde{X}^3 & 14\tilde{X}^4 & -4\tilde{X}^5 \\ -201 & 4\tilde{X} & 33\tilde{X}^2 & -4\tilde{X}^3 & -3\tilde{X}^4 & \tilde{X}^5 \end{pmatrix}$$

Alors $r(X) = 3 + 8X - 24X^2 - 8X^3 - X^4 + 2X^5$, on a $r(3) = 0 \equiv p(3) \mod 35$.

Remarque 1.3.3. On constate que la borne $\tilde{X} = \tilde{X}(h) = 2 < 3 = x$ qui est la solution. Ce phénomène se passe souvent, on dit que la borne $\tilde{X}(h)$ est pessimiste.

Méthode de Coppersmith

Calculons des $q_j^{copp}(X) = q_{u,v}^{copp}(X) := X^u(p(X))^v$ avec $0 \leq j \leq hk - k - 1 = 3, v = \lfloor (k+j)/k \rfloor$ et $u = j - kv$.

$$\begin{aligned} q_0 &= q_{0,0} = 19 + 14X + X^2 \\ q_1 &= q_{0,1} = 19X + 14X^2 + X^3 \\ q_2 &= q_{1,0} = 361 + 532X + 234X^2 + 28X^3 + X^4 \\ q_3 &= q_{1,1} = 361X + 532X^2 + 234X^3 + 28X^4 + X^5 \end{aligned}$$

Alors $\hat{M}^{copp}$ est une matrice carré de taille $(2hk - k) = 10$

$$\hat{M}^{copp}_{10\times 10} = \left(\begin{array}{c|c} D_{6\times 6} & A_{6\times 4} \\ \hline 0_{4\times 6} & D'_{4\times 4} \end{array}\right)$$
$$= \left(\begin{array}{cccccc|cccc} \tilde{X}^0 & 0 & 0 & 0 & 0 & 0 & 19 & 0 & 361 & 0 \\ 0 & \tilde{X}^{-1} & 0 & 0 & 0 & 0 & 14 & 19 & 532 & 361 \\ 0 & 0 & \tilde{X}^{-2} & 0 & 0 & 0 & 1 & 14 & 234 & 532 \\ 0 & 0 & 0 & \tilde{X}^{-3} & 0 & 0 & 0 & 1 & 28 & 234 \\ 0 & 0 & 0 & 0 & \tilde{X}^{-4} & 0 & 0 & 0 & 1 & 28 \\ 0 & 0 & 0 & 0 & 0 & \tilde{X}^{-5} & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 35 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 35 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1225 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1225 \end{array}\right)$$

Les uns en rouge permettent de rendre $\hat{M}^{copp}$ à la forme

$$\bar{M}^{copp} = H_1 \hat{M}^{copp} = \left(\begin{array}{c|c} M^{copp}_{6\times 6} & 0_{6\times 4} \\ \hline A'_{4\times 6} & 1_{4\times 4} \end{array}\right), \quad \text{avec } |\det H_1| = 1$$

$$= \left(\begin{array}{cccccc|cccc} 1 & 0 & -19\tilde{X}^{-2} & 266\tilde{X}^{-3} & -3363\tilde{X}^{-4} & 42028\tilde{X}^{-5} & 0 & 0 & 0 & 0 \\ 0 & \tilde{X}^{-1} & -14\tilde{X}^{-2} & 177\tilde{X}^{-3} & -2212\tilde{X}^{-4} & 27605\tilde{X}^{-5} & 0 & 0 & 0 & 0 \\ 0 & 0 & -35\tilde{X}^{-2} & 490\tilde{X}^{-3} & -5530\tilde{X}^{-4} & 58800\tilde{X}^{-5} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -35\tilde{X}^{-3} & 980\tilde{X}^{-4} & -19250\tilde{X}^{-5} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1225\tilde{X}^{-4} & 34300\tilde{X}^{-5} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1225\tilde{X}^{-5} & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & \tilde{X}^{-2} & -14\tilde{X}^{-3} & 158\tilde{X}^{-4} & -1680\tilde{X}^{-5} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \tilde{X}^{-3} & -28\tilde{X}^{-4} & -550\tilde{X}^{-5} & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \tilde{X}^{-4} & -28\tilde{X}^{-5} & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \tilde{X}^{-5} & 0 & 0 & 0 & 1 \end{array}\right)$$

Alors on a obtenu M^{copp}, il reste à trouver H_1 et H_2 définies par 1.13 et 1.14 pour calculer $r(X) = [c(X)H_1^{-1}]_6((H_2^{-1})^t)_5$ avec $[c(X)H_1^{-1}]_6 = (1, X, \frac{p(X)}{35}, \frac{-Xp(X)}{35}, \frac{-p^2(X)}{1225}, \frac{-Xp(X)^2}{1225})$ (les détails se trouvent dans Howgrave-Graham (1997)). Finalement, on retrouve $r(X) = 3 + 8X - 24X^2 - 8X^3 - X^4 + 2X^5$ qui est le même polynôme donné par la nouvelle méthode. L'article Howgrave-Graham (1997) a également expliqué ce phénomène par la *dualité* des deux réseaux euclidiens engrendrées par M et M^{copp}.

1.4 Applications dans RSA

Inspiré du cours *Complexité algébrique et Cryptographie* de **M.Louis Goubin**, nous présentons deux attaques connues dû au calcul efficace de petites racines d'un polynôme de bas degré dans $\mathbb{Z}[X]$.

1.4.1 Attaque de Haståd (1988)

Supposons qu'une personne A veuille envoyer un message à trois personnes différentes B_1, B_2 et B_3 et que les trois ont la même clé publique qui est $e = e_i = 3$. Chacun dispose de sa clé secrète d_i qui est l'inverse de e modulo $\phi(n_i) = (p_i - 1)(q_i - 1)$ où $n_i = p_i q_i$. On a donc le schéma suivant:

$$A \xrightarrow{\quad y_i = x^e \bmod n_i \quad} B_i$$

pour $i \in \{1, 2, 3\}$

L'attaquant C connaît ainsi:

$$\begin{cases} y_1 = x^3 \quad mod\ n_1 \\ y_2 = x^3 \quad mod\ n_2 \end{cases}$$

D'après le théorème du reste chinois, comme le $pgcd(n_1, n_2) = 1$, C connaît alors :
$x^3 \ mod \ (n_1, n_2)$
Donc C connaît:

$$\begin{cases} x^3 & mod \ (n_1, n_2) \\ y_3 = x^3 & mod \ n_3 \end{cases}$$

D'après théorème du reste chinois, comme le $pgcd(n_1, n_2, n_3) = 1$, C connaît alors :
$x^3 \ mod \ (n_1, n_2, n_3)$

Sachant que : $\begin{cases} 0 \leq x < n_1 \\ 0 \leq x < n_2 \\ 0 \leq x < n_3 \end{cases} \Longrightarrow 0 \leq x^3 < n_1 n_2 n_3 \Longrightarrow$ Il connaît $\mathrm{x}^3 \Longrightarrow$ Il connaît x

Remarque:

1) En général, si l'exposant est d, on a besoin de d messages.

2) Si le $pgcd(n_i, n_j) \neq 1$, pour $i \neq j$ et $i, j \in \{1, 2, 3\}$, alors on peut facilement trouver la factorisation de n et donc le message clair.

Amélioration possible: On suppose que chaque destinataire possède (en plus de sa clé publique RSA) un polynôme public:

$$f_i \in \mathbb{Z}/N_i\mathbb{Z}[X]$$

Le chiffrement au déstinataire numéro i est :

$$C_i = f(M)^{e_i} \ mod \ N_i \ (1 \leq i \leq k)$$

Théorème 1.4.1. *Attaque de Haståd améliorée :*

. $N_1, \cdots, N_k$ entiers premiers entre eux deux à deux

. On pose $N = min_{1 \leq i \leq k} N_i$

. $g_i \in \mathbb{Z}/N_i\mathbb{Z}[X]$ $(1 \leq i \leq k)$: k polynômes de degré maximaux d

S'il existe un unique message $M < N_{min}$ tel que $\forall i$, $1 \leq i \leq k$, $g_i(M) = 0 \ mod \ N_i$ et si $k \geq d$ alors on peut trouver M en temps polynomial.

Preuve. On pose $\overline{N} = N_1 \cdots N_k$.
On peut supposer que les g_i sont unitaires (quitte à les multiplier par l'inverse du coefficient dominant)
NB : Si le coefficient dominant n'est pas inversible dans $\mathbb{Z}/N_i\mathbb{Z}[X]$ alors on peut factoriser N_i.

On suppose aussi que les g_i sont tous de degré d (quitte à les multiplier par des monômes)
On définit le polynôme

$$g(x) = \sum_{i=1}^{k} T_i g_i(x) \quad \text{avec } T_i = \begin{cases} 1 & \mod N_i \\ 0 & \mod N_j \quad (j \neq i) \end{cases}$$

T_i est ainsi définit $\mod \overline{N}$ par le théorème du reste chinois.
On a :
· g est unitaire et de degré d.
· $g(M) \equiv 0 \mod \overline{N}$
· $M < N_{\min} \leq \overline{N}^{1/k} \leq \overline{N}^{1/d} \Longrightarrow$ M<$\overline{N}^{1/d} \Longrightarrow \exists \epsilon / M < \overline{N}^{1/d-\epsilon}$
D'après le théorème de Coppersmith on peut trouver M en temps polynomiale. □

Remarque 1.4.1. On peut prendre les $g_i = (f_i)^{e_i} - c_i \equiv 0 \mod N_i$

1.4.2 Attaque de Coppersmith

On chiffre un message M par $C = (2^m M + r)^e \ mod \ N$
Où:· k est le nombre de bits de N
· $m = \lfloor k/e^2 \rfloor$
· M est le message clair d'au plus $k - m$ bits
· r est une valeur alétoire de m bits
· N est de k bits
·(N, e) est la clé publique du RSA

Attaque de Coppersmith: On suppose que l'attaquant peut récupérer deux chiffrés de M:

$$C_1 = M_1^e \ mod \ N \ \text{ avec } M_1 = 2^m M + r_1$$

$$C_2 = M_2^e \ mod \ N \text{ avec } M_2 = 2^m M + r_2$$

où r_1 et r_2 sont aléatoires.

On définit:

$$\begin{cases} g_1(x,y) & = x^e - c_1 \\ g_2(x,y) & = (x+y)^e - c_2 \qquad (j \neq i) \end{cases}$$

Quand $y = r_2 - r_1$ alors M_1 est une racine commune aux deux polynômes, en effet,
$g_1(M_1, r_2 - r_2) = M_1^e - c_1 = 0 \ mod \ \ N$
$g_2(M_1, r_2 - r_1) = (M_1 + r_2 - r_1)^e - c_2 = 0 \ mod \ N$ car $M_1 + r_2 - r_1 = M_2$
Soient

$$h(y) = Res_x(g_1, g_2) \in \mathbb{Z}/n\mathbb{Z}[y]$$

$$\Delta = r_2 - r_1 \text{ est une racine de } h$$

$$h(y) = det \begin{pmatrix} 1 & 0 & \cdots & 0 & -c_1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 & -c_1 & 0 & \cdots & 0 \\ \vdots & & \ddots & & & & & & \\ 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & 0 & -c_1 \\ 1 & ey & \cdots & (y^e - c_2) & 0 & \cdots & 0 & 0 & 0 \\ & & & & \vdots & & & & \\ 0 & \cdots & 0 & 0 & 0 & 1 & ey & \cdots & (y^e - c_2) \end{pmatrix}$$

$\Longrightarrow h(y) = \sum_{\sigma \in S_{2e}} (-1)^{\epsilon(\sigma)} \prod_{i=1}^{2e} A_{i,\sigma(i)}$ avec degré($\prod_{i=1}^{2e} A_{i,\sigma(i)}$)$\leq e^2$
$\Longrightarrow$ degré(h) $\leq e^2$
Or $0 \leq r_1 < 2^m$ avec $m = \lfloor k/e^2 \rfloor \Longrightarrow 0 \leq r_1 < N^{1/e^2}$ (1)
De même, $0 \leq r_2 < N^{1/e^2}$ (2)
(1) et (2) $\Longrightarrow |\Delta| < N^{1/\text{degré}(h)}$
D'après le théorème de Coppersmith on peut trouver Δ en temps polynomial. On sait donc que M_1 est racine de $g_1(x) = x^e - c_1$ et de $g_2(x) = (x + \Delta)^e - c_2$
Il suffit donc de calculer le pgcd(g_1, g_2) pour obtenir M_1.

Chapitre 2

Programmation

2.1 Librairie C Flint

Flint est une librairie dans C qui permet de faire de la théorie des nombres. Actuellement, elle est maintenue par William Hart, son site officiel est : `http://www.flintlib.org/`.

2.1.1 Installation

`FLINT` peut être installée sur différents systèmes d'exploitation et peut être trouvée sur son site officiel. Dans notre cas, on a installé `FLINT` sur macOS via Homebrew. Il suffit d'abord de rentrer sur le site `https://brew.sh/`, ensuite de copier le code d'installation d'Homebrew dans le terminal. Une fois que Homebrew est installé, `brew install flint` installe la librairie `FLINT`.

Hombrew installe automatiquement `gcc, isl, libmpc, nlt, gmp, mpfr, mpir`. Cependent, seules les trois dernières contributent à installer `FLINT` dont `gmp` qui est à son tour indispensable pour `mpfr`. Les détails pour ceux qui veulent installer que ces trois librairies sont disponibles dans `INSTALL` et `README` du package `FLINT` téléchargé du `http://www.flintlib.org/`.

2.1.2 Utilisation globale

`FLINT` est généralement utilisée pour l'arithmétique avec: *des nombres, des polynômes, des séries et des matrices,* sur de nombreux anneaux, comme par exemple les:

- Entiers et rationnels.
- Entiers modulo n.
- Nombres p-adiques.
- Corps finis.

- Nombres réels et complexes.

Les opérations qui peuvent être effectuées comprennent les conversions, l'arithmétique, le calcul des pgcd, la factorisation, la résolution de systèmes linéaires et l'évaluation de fonctions spéciales. De plus, Flint fournit plusieurs méthodes pour faire de l'arithmétique rapidement.

Cette librairie est largement documentée et testée. En effet, elle a été utilisée pour faire des calculs à grande échelle dans la recherche en théorie des nombres, et convient également pour les systèmes d'algèbre informatique. Sage utilise FLINT comme package par défaut pour l'arithmétique polynomiale sur $\mathbb{Z}$, $\mathbb{Q}$ et $\mathbb{Z}/n\mathbb{Z}$ pour des n qui sont petits. Des travaux sont actuellement en cours pour utiliser FLINT dans Singular et Macaulay2.

2.2 Implémentation de la nouvelle méthode

Comme indiqué dans le chapitre précédent, la nouvelle méthode est plus facile à implémenter que la méthode de Coppersmith du fait que les manipulations matricielles sont moins demandées. On va détailler l'implémentation de la nouvelle méthode à l'aide de `FLINT` avec quelques mises en garde.

Remarque 2.2.1. Cette implémentaion n'est pas faite pour un N très grand. L'explication se trouve dans la partie suivante.

Algorithme 2: Calcul de petites racines

Input: Le polynôme $p(X)$, N, et les parametres $h, \tilde{X}$
Output: La solution x

$k \leftarrow \deg(p(X))$;
for $i \leftarrow 0$ **to** $hk - 1$ **do** Calculer q_i;
Construire la matrice M à partir des q_i;
Réduire M par LLL;
Construire $r = \sum c_i q_i$ à partir de la première ligne de la matrice réduite;
Calculer les racines entières x de r. Vérifier si x est une racine de (1.4);
return x

2.2.1 Documentation

On utilise des headers `fmpz_poly.h`, `fmpz_lll.h`, et `fmpz_poly_factor.h` de la librairie FLINT.

- `fmpz_poly.h` fournit les structures qui représentent les polynômes, les matrices à coefficients entiers et les opérations élementaires autorisées sur ces structures.
  - La structure `fmpz_poly_t{fmpz* coeffs; slong length;...}` enregistre un polynôme $p(x) = a_k x^k + a_{k-1}x^{k-1} + \cdots + a_0$ sous la forme:

$$\texttt{k+1} \quad a_0 \;\; a_1 \;\; \ldots \quad a_k(*)$$

Dans cette structure, les coefficients sont limités par la taille du type `fmpz` qui est en effet un `slong` de 32 bits qui peut représenter les entiers dans l'intervalle $[-2.147.483.647, +2.147.483.647]$ (On peut rencontrer l'erreur `segmentation fault` en calculant q_i quand N et h sont grands). Pour utiliser cette structure il faut d'abord la déclarer puis l'initialiser par `fmpz_poly_init()` et finalement la libérer par `fmpz_poly_clear()`. Il y a plusieurs façons d'ajouter des coefficients comme `fmpz_poly_set_str(p, s)` qui passe un `char* s` du syntaxe (*) au polynôme `fmpz_poly_t p` ou `fmpz_poly_read(p)` qui lit `p` depuis le clavier.

  - La structure `fmpz_mat_t{fmpz** entry; ...}` enregistre des coefficients de la matrice dans `entry`. On peut manipuler le coefficient $M_{i,j}$ de la matrice M par `*fmpz_mat_entry(M,i,j)`.
  - Les opération élémentaires sont `fmpz_poly_mul, fmpz_poly_pow, fmpz_poly_scalar_mul_ui`, ... (pour la multiplication, la puissance, la multiplication par un scalaire) des polynômes ou des matrices à coefficients entiers.
  - La fonction `fmpz_poly_evaluate_mod(p,x,N)` permet d'évaluer `p(X) mod N` en `X=x`.

- `fmpz_lll.h` fournit l'algorithme LLL qui fait la réduction de base du réseau euclidien.

  - La variable `fmpz_lll_t fl {delta, eta, rt}` sert à préciser la méthode pour la réduction. La fonction `fmpz_lll_context_init_default(fl)` initialise `delta, eta, rt` par défaut.
  - La fonction `fmpz_lll(M,U,fl)` rend M réduite par U avec l'option LLL `fl`.

- `fmpz_poly_factor.h` fournit des méthodes de factorisation de polynômes dans $\mathbf{Z}[X]$.

  - La structure `fmpz_poly_factor_t fac{slong num; fmpz_poly_t *p; slong *exp}` enregistre `num` termes de la factorisation dans `p` et leurs mulitplicités dans `exp`. On l'initialise par `fmpz_poly_factor_init(fac)` et la libère par `fmpz_poly_factor_clear(fac)`.
  - La fonction `fmpz_poly_factor(fac, r)` enregistre la factorisation du polynôme `fmpz_poly_t r` dans `fmpz_poly_factor_t fac`. Elle sert à trouver des racines de `r`.

2.2.2 Compilation et debugging

La compilation se fait par le `Makefile`.

Le syntaxe du programme compilé est

```
./test7 a b c d
```

- `N`← `a`
- `h`← `b`

```
CC=gcc
CFLAGS=-Wall

GMP_INCLUDE=-I/Users/quyennguyen/Programs/libs/gmp/6.2.1/include
GMP_LIBS=-L/Users/quyennguyen/Programs/libs/gmp/6.2.1/lib -lgmp
FLINT_INCLUDE=-I/usr/local/Cellar/flint/2.7.0/include/flint
FLINT_LIBS=-L/usr/local/Cellar/flint/2.7.0/lib -lflint

test7 : test7.c
	${CC} ${CFLAGS} -o test7 test7.c ${FLINT_INCLUDE} ${FLINT_LIBS}
```

Figure 2.1: `Makefile`

- Si `c = 0` alors $\tilde{X} \leftarrow \tilde{X}(h)$ comme calculé dans (1.11) sinon $\tilde{X} \leftarrow$ `c`
- Si `d = defaut` (Figure: 2.2) alors $p(X) \leftarrow X^2 + 14X + 19$ sinon `d = test` (Figure: 2.3) alors le programme lit $p(X)$ sous le syntaxe (*) depuis le clavier.

```
[quyennguyen@Quyens-MacBook-Air flint % ./test7 35 3 0 defaut
hk   h    k    N    6 3 2 35
X    2
p:   x^2+14*x+19
q0:   1225
q1:   1225*x
q2:   35*x^2+490*x+665
q3:   35*x^3+490*x^2+665*x
q4:   x^4+28*x^3+234*x^2+532*x+361
q5:   x^5+28*x^4+234*x^3+532*x^2+361*x
M:   6 6  1225 0 0 0 0 0 0 2450 0 0 0 0 665 980 140 0 0 0 0 1330 1960 280 0 0 361
 1064 936 224 16 0 0 722 2128 1872 448 32
M:   6 6  3 16 -96 -64 -16 64 49 100 0 160 0 64 -128 50 -20 80 160 32 -201 8 132
-32 -48 32 -83 -142 52 8 32 160 61 32 148 -128 48 128
r:   2*x^5-x^4-8*x^3-24*x^2+8*x+3
3
```

Figure 2.2: `c = 0, d = defaut, x = 3`

Remarque 2.2.2. L'affiachage ne sert qu'à vérifier le bon fonctionnement du programme. On voit bien dans l'exemple suivant (Figure: 2.4) que les coefficients des q_i sont très grandes tels qu'ils dépassent la capacité du type `fmpz`, ce qui mène à une faux représentation par type et l'erreur `Segementation fault` .

Remarque 2.2.3. Comme on voulait profiter des fonctions de `FLINT`, quelques détails du programme ne sont pas optimisés. Par ailleurs, la limitation des tests nous empêche de conclure des résultats statistiques fiables sur la performance du programme. Le code du programme est disponible sur github.

```
mayachartouny@Maya Nouv % ./test7 35 3 3 test
3  19 14 1
hk  h   k   N   6 3 2 35
X   3
p:  x^2+14*x+19
1225
1225*x
35*x^2+490*x+665
35*x^3+490*x^2+665*x
x^4+28*x^3+234*x^2+532*x+361
x^5+28*x^4+234*x^3+532*x^2+361*x
M:  6 6  1225 0 0 0 0 0 0 3675 0 0 0 0 665 1470 315 0 0 0 0 1995 4410 945 0 0 36
1 1596 2106 756 81 0 0 1083 4788 6318 2268 243
M:  6 6  -201 12 297 -108 -243 243 -3 -24 216 216 81 -486 -243 324 -81 -81 324 -
243 193 411 216 81 81 243 387 -63 297 -378 -243 0 -66 399 -36 189 -486 0
r:  x^5-3*x^4-4*x^3+33*x^2+4*x-201
3
```

Figure 2.3: `c = 3, d = test, x = 3`

```
quyennguyen@Quyens-MacBook-Air flint % ./test7 8619 6 8 defaut
hk  h   k   N   12 6 2 8619
X   8
p:  x^2+14*x+19
q0:  0
q1:  0
q2:  5518582289439921*x^2+77260152052158894*x+104853063499358499
q3:  5518582289439921*x^3+77260152052158894*x^2+104853063499358499*x
q4:  640281040659*x^4+17927869138452*x^3+149825763514206*x^2+340629513630588*x+2311
41455677899
q5:  640281040659*x^5+17927869138452*x^4+149825763514206*x^3+340629513630588*x^2+23
1141455677899*x
q6:  74287161*x^6+3120060762*x^5+47915218845*x^4+322406278740*x^3+910389158055*x^2+
1126341935082*x+509535637299
q7:  74287161*x^7+3120060762*x^6+47915218845*x^5+322406278740*x^4+910389158055*x^3+
1126341935082*x^2+509535637299*x
q8:  8619*x^8+482664*x^7+10790988*x^6+122113992*x^5+734942130*x^4+2320165848*x^3+38
95546668*x^2+3310592376*x+1123236699
q9:  8619*x^9+482664*x^8+10790988*x^7+122113992*x^6+734942130*x^5+2320165848*x^4+38
95546668*x^3+3310592376*x^2+1123236699*x
q10:  x^10+70*x^9+2055*x^8+32760*x^7+307410*x^6+1732164*x^5+5840790*x^4+11826360*x^
3+14095245*x^2+9122470*x+2476099
q11:  x^11+70*x^10+2055*x^9+32760*x^8+307410*x^7+1732164*x^6+5840790*x^5+11826360*x
^4+14095245*x^3+9122470*x^2+2476099*x
zsh: segmentation fault  ./test7 8619 6 8 defaut
```

Figure 2.4: `Faux représentations de q0 et q1 et segementation fault`

Réferences

(2013). Short vectors in lattices. In J. von zur Gathen, & J. Gerhard (Eds.) *Modern Computer Algebra*, (pp. 473–502). Cambridge: Cambridge University Press, 3 ed.
URL https://www.cambridge.org/core/books/modern-computer-algebra/short-vectors-in-lattices/EDA3AAE03A5C274C95BC38A4434EC64D

Coppersmith, D. (1997). Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities. *Journal of Cryptology*, *10*(4), 233–260.
URL https://doi.org/10.1007/s001459900030

Howgrave-Graham, N. (1997). Finding small roots of univariate modular equations revisited. In M. Darnell (Ed.) *Crytography and Coding*, Lecture Notes in Computer Science, (pp. 131–142). Berlin, Heidelberg: Springer.