Rapport de Projet

Le cryptosystème NTRU

Marion Candau

Sous la direction de Mme Christine Bachoc

14 mars 2011

Université Bordeaux 1
Master 2 Cryptologie et Sécurité Informatique

Table des matières

Introduction

La cryptographie moderne est composée de deux branches : la cryptographie symétrique et la cryptographie asymétrique. Pour chiffrer et déchiffrer avec la première, il faut connaître la clé secrète. Les cryptosystèmes les plus connus dans la cryptographie symétrique sont l'AES (2002), IDEA (1990) et 3DES (2005). Ce type de cryptographie peut présenter évidemment un danger car il faut échanger les clés. Dans la deuxième branche, appelée également cryptographie à clé publique, les cryptosystèmes les plus connus sont le cryptosystème RSA (1977) basé sur la difficulté de factoriser les grands nombres entiers, le cryptosystème El Gamal (1985) basée sur le problème du logarithme discret, le cryptosystème ECC (Elliptic Curve Cryptography, 1985) basé sur le problème du logarithme discret sur une courbe elliptique et enfin, l'objet de ce projet, le cryptosystème NTRU (1996) basé sur le problème du plus court vecteur non nul d'un réseau.

Ce dernier a été présenté à la rump session de Crypto'96, la conférence annuelle sur la cryptographie par Jeffrey Hoffstein, Jill Pipher et Joseph Silvermann. L'acronyme NTRU peut référencer à la fois les expressions « Number Theorist aRe Us » et « Number Theory Research Unit » . Quant à la signature basée sur NTRU, elle a tout d'abord été nommée NSS (NTRU Signature Scheme) et proposée à la rump-session de Crypto 2000. Son principal avantage (et aussi son inconvénient) était qu'elle ne reposait que sur l'information disponible immédiatement à partir de la clé privée, soit la moitié d'une base réduite, ce qui a été exploité par Gentry et Szydlo à la rump-session de Crypto 2001 pour casser le système. Un nouvel algorithme de signature appelé NTRUSign a donc été présenté à la rump-session d'Asiacrypt 2001. En 1996, les créateurs de NTRU ainsi que Daniel Lieman ont fondé la société Ntru Cryptosystems qui a lancé NTRU sur le marché. En 2009, l'entreprise a été acquise par Security Innovation [5].

Aujourd'hui, NTRU est utilisé entre autres dans la RFID (Radio Frequency Identification), le e-commerce, l'authentification biométrique, la communication directe vers les véhicules et la protection des produits industriels et pharmaceutiques contre les contrefaçons.

Objectifs

Les objectifs de ce projet sont d'étudier en théorie et en pratique le cryptosystème NTRU à partir notamment du document [1] et donc d'implémenter un programme qui réalise le chiffrement et la signature, et d'étudier la sécurité de ce système et notamment les attaques possibles.

Plan du projet

Dans le premier chapitre, nous verrons les bases mathématiques de NTRU c'est-à-dire les réseaux euclidiens. Dans la deuxième partie seront abordés les différents algorithmes qui composent le système NTRU. Puis, dans la troisième partie, nous verrons les techniques de réductions de réseaux afin de pouvoir attaquer NTRU. La sécurité de NTRU sera étudiée dans la quatrième partie. Puis, dans les cinquièmes et sixièmes chapitres, nous verrons successivement l'implémentation des algorithmes de chiffrement et de signature et les résultats qui en découlent. Enfin, ce que l'on peut conclure de ce projet sera abordé dans la dernière partie.

Chapitre 1

Les réseaux euclidiens

La sécurité du crypto-système NTRU étant basée sur le problème de trouver le plus court vecteur d'un réseau, quelques définitions sur les réseaux sont préalablement nécessaires avant d'introduire le cryptosystème.

1.1 Définitions

Définition : Réseau

Un réseau L est un sous-groupe discret additif de $\mathbb{R}^n$, c'est-à-dire qu'il existe un $\epsilon > 0$ tel que pour tout $v \in L$, et tout $w \in \mathbb{R}^n$, $w \neq v$, si $||v-w|| < \epsilon$ alors w n'appartient pas au réseau L. On peut également définir un réseau de la manière suivante :
Si $v_1, v_2, \ldots, v_k$ sont des vecteurs linéairement indépendants de $\mathbb{R}^n$ alors un réseau L est l'ensemble :

$$L = \left\{ \sum_{i=1}^{k} a_i v_i \mid \forall i \in \{1, \ldots, k\},\ a_i \in \mathbb{Z} \right\}$$

C'est le réseau engendré par $v_1, v_2, \ldots, v_k$.

Définition : Bases et dimension d'un réseau

Si L est un réseau tel que $L = \left\{ \sum_{i=1}^{k} a_i v_i \mid \forall i \in \{1, \ldots, k\},\ a_i \in \mathbb{Z} \right\}$ et $v_1, v_2, \ldots, v_k$ sont linéairement indépendants alors $v_1, v_2, \ldots, v_k$ est une base de L et L est de dimension k. Pour toute autre base $w_1, w_2, \ldots, w_m$, on a $m = k$.
Un réseau L admet plusieurs bases. Si $w_1, w_2, \ldots, w_k$ est une autre base de L alors la matrice de passage de $(w_1, w_2, \ldots, w_k)$ à $(v_1, v_2, \ldots, v_k)$ est à cœfficients entiers et a pour déterminant ± 1.
Si $k = \dim(L) = n = \dim(\mathbb{R}^n)$ on dit que L est de rang plein. On représente un réseau par sa base dans une matrice $k \times n$ en exprimant en ligne les

coordonnées de vecteurs de la base de L dans la base canonique de $\mathbb{R}^n$.

$$B = \begin{pmatrix} \leftarrow & b_1 & \rightarrow \\ \leftarrow & b_2 & \rightarrow \\ & \vdots & \\ \leftarrow & b_k & \rightarrow \end{pmatrix}$$

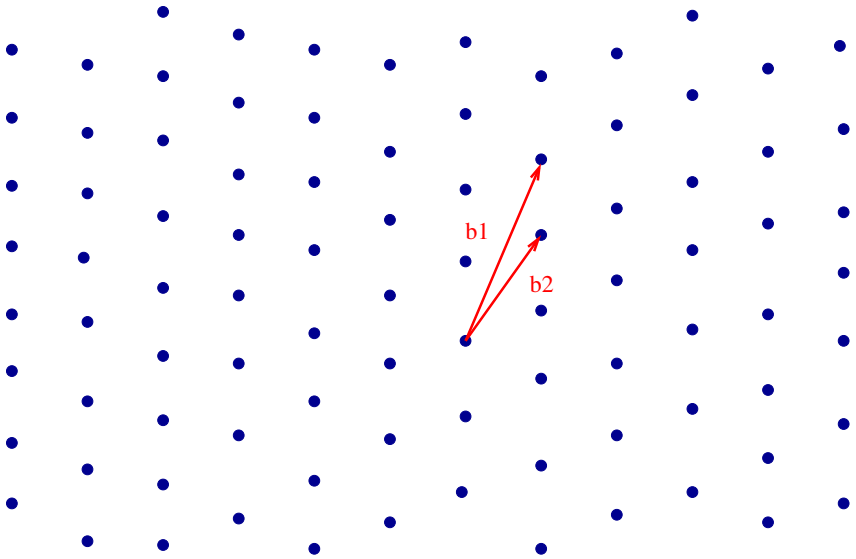

FIG. 1.1.0.1 – Réseau de dimension 2 engendré par (b_1, b_2)

Définition : Matrice de Gram et déterminant d'un réseau

Pour définir cette matrice, nous avons besoin du produit scalaire de deux vecteurs de $\mathbb{R}_n$ qui est défini comme suit :

$$\langle x, y \rangle = \sum_{i=1}^{n} x_i y_i$$

Ainsi la matrice de Gram de L est la matrice des produits scalaires d'une base $B = (b_1, b_2, \ldots, b_k)$ de L :

$$G = \langle b_i, b_j \rangle_{1 \leqslant i,j \leqslant k}$$

Matriciellement, cela correspond à :

$$G = B \times B^T = \begin{pmatrix} \leftarrow & b_1 & \rightarrow \\ \leftarrow & b_2 & \rightarrow \\ & \vdots & \\ \leftarrow & b_k & \rightarrow \end{pmatrix} \times \begin{pmatrix} \uparrow & \uparrow & & \uparrow \\ b_1 & b_2 & \ldots & b_k \\ \downarrow & \downarrow & & \downarrow \end{pmatrix}$$

Le déterminant de cette matrice appelé déterminant de Gram est noté $\det G$ et est strictement positif.
On appelle déterminant du réseau :

$$\det(L) = \sqrt{\det(G)}$$

Le déterminant est un invariant du réseau qui ne dépend pas du choix de la base de L. En effet, si $B' = PB$ est une autre base alors $G = B'B'^T = PBB^TP^T$ et $\det(G) = (\det(P))^2 \times \det(BB^T) = 1 \times \det(G)$.
Si le réseau L est de rang plein alors B est une matrice carrée $n \times n$ donc $\det(G) = (\det(B))^2$ et $\det(L) = |\det(B)|$.
Géométriquement, le déterminant de L est le volume du parallélotope formé par les vecteurs de base c'est-à-dire :

$$\det(L) = Vol\left(\left\{\sum_{i=1}^{k} x_i b_i,\ 0 \leqslant x_i \leqslant 1\right\}\right)$$

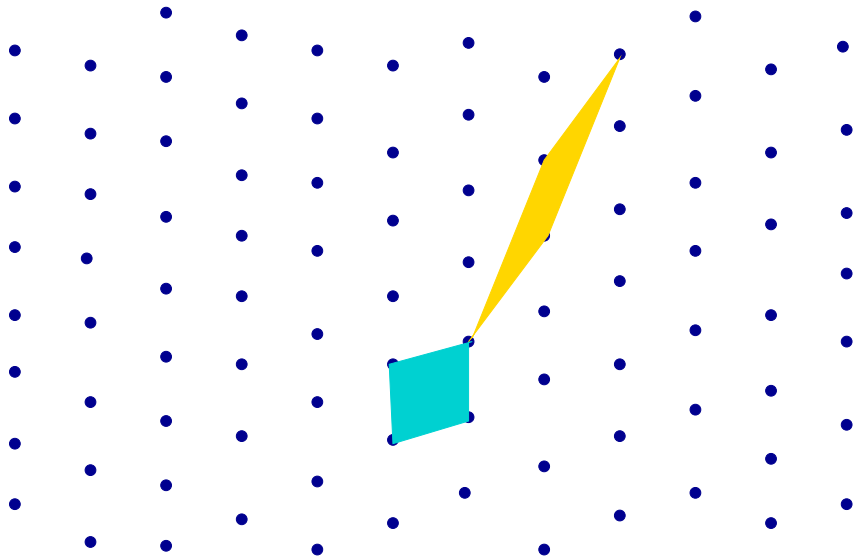

FIG. 1.1.0.2 – Deux parallélotopes d'un même réseau ont le même volume

Définition : minimum de L

Pour définir le minimum d'un réseau, on considère $\| \ . \ \|$, la norme euclidienne sur $\mathbb{R}^n$ définie par :

$$\|x\| = \sqrt{\langle x, x\rangle} = \sqrt{\sum_{i=1}^{n} x_i^2}$$

Le minimum d'un réseau est :

$$\lambda_1(L) = \min L = \min_{x \in L \setminus \{0\}} \|x\|$$

L'ensemble des vecteurs minimaux de L est l'ensemble fini :

$$S(L) = \{x \in L \mid \|x\| = \min(L)\}$$

Définition : Constante d'Hermite

$\det(L)$ et $\min(L)$ restent inchangées si on fait agir une transformation orthogonale sur le réseau L. De plus, si on transforme L par une homothétie $x \mapsto \lambda x$, $\lambda > 0$, on obtient $\det(\lambda L) = \lambda^n \det(L)$ et $\min(\lambda L) = \lambda \min(L)$. D'où, le quotient :

$$\gamma(L) = \frac{\min(L)^2}{\det(L)^{\frac{2}{n}}}$$

appelé la constante d'Hermite du réseau L est invariant par les similitudes.

1.2 Problèmes du plus court vecteur et du plus proche vecteur

A l'aide de toutes ses définitions, on va pouvoir formuler les deux problèmes suivants.

1. Le problème du plus court vecteur, appelé en anglais Shortest Vector Problem (SVP), est le problème suivant :
   Étant donné une base B d'un réseau L, trouver un vecteur non nul de L le plus petit possible c'est-à-dire trouver $v \neq 0 \in L$ tel que $\|v\|$ est minimale.
2. Le problème du plus proche vecteur, appelé en anglais Closest Vector Problem (CVP), est le problème suivant :
   Étant donné une base B d'un réseau L et un vecteur $w \notin L$, trouver le vecteur $v \in L$ le plus proche de w, c'est-à-dire trouver $v \in L$ tel que $\|v - w\|$ est minimale.
   Ces deux problèmes sont NP-difficiles.

   La question que l'on pourrait se poser est de quelle taille est le vecteur le plus court en fonction du déterminant et de la dimension de L ?
   D'après un théorème, la constante d'Hermite associée à chaque réseau de dimension n donc notée γ_n est la plus petite valeur qui satisfait :

   $$\|v_{plus\ court}\|^2 \leqslant \gamma_n \det(L)^{\frac{2}{n}}$$

   De plus, Hermite montra que $\gamma_n \leqslant \left(\frac{4}{3}\right)^{\frac{n-1}{2}}$.
   Pour n grand, on ne connaît pas la limite exacte de la taille du plus

court vecteur, ne connaissant pas la valeur exacte de γ_n, mais une heuristique probabiliste due à Gauss permet d'en avoir une idée.
Une variante de l'heuristique de Gauss dit que, pour un réseau L fixé et une boule de rayon r centrée en 0, quand r tend vers l'infini, le rapport entre le volume de la boule et $\det(L)$ se rapprochera du nombre de points de L dans la boule. Mais plus n est grand, plus l'erreur créée par les points du réseau près de la surface de la boule est grande. Cela est encore plus problématique pour les petites valeurs de r. Cependant, pour quelle valeur de r a-t-on $\frac{Vol(B)}{\det(L)}$ qui approche 1 ? Cela nous donne en quelque sorte, une valeur attendue pour r, le plus petit rayon où le nombre prévu de points de L avec une longueur inférieure à r est égal à 1. Nous obtenons que cette valeur est, pour n grand, d'environ :

$$r = \sqrt{\frac{n}{2\pi e}}(\det(L))^{\frac{1}{n}}$$

Ainsi, si L est un réseau de dimension n, la longueur attendue du vecteur le plus court selon Gauss est :

$$\sigma(L) = \sqrt{\frac{n}{2\pi e}}(\det(L))^{\frac{1}{n}}$$

Cette valeur peut être utile dans la quantification de la difficulté de localiser les vecteurs courts d'un réseau. Elle peut être considérée comme la longueur probable du vecteur le plus court d'un réseau aléatoire de déterminant et de dimension donnés.
Un argument heuristique identique à celui-ci peut être utilisé pour analyser les CVP.
Des problèmes secondaires, néanmoins importants, proviennent de SVP et CVP. On pourrait chercher un vecteur $v \in L$ non nul satisfaisant :

$$\|v\| \leqslant \varphi(n)\|v_{plus\ court}\|$$

avec φ une fonction croissant lentement de n la dimension de L. Par exemple, pour une constante K fixée, on pourrait essayer de trouver $v \in L$ tel que :

$$\|v\| \leqslant K\sqrt{n}\|v_{plus\ court}\|$$

et de manière similaire pour CVP. Ces généralisations sont connues sous le nom d'approximation du plus court ou du plus proche vecteur, ou ASVP et ACVP.

Les bases des réseaux étant posées, on va pouvoir décrire le cryptosystème NTRU et les deux algorithmes NTRUEncrypt pour le chiffrement et NTRUSign pour la signature.

Chapitre 2

Le cryptosystème NTRU

Les opérations de NTRU s'effectuent dans l'anneau :

$$\mathcal{R} = \frac{\mathbb{Z}[X]}{X^N - 1}$$

Les éléments f de $\mathcal{R}$ sont ainsi représentés par :

$$f = (f_0, f_1, \ldots, f_{N-1}) = \sum_{i=0}^{N-1} f_i X^i$$

La multiplication dans cet anneau correspond à la multiplication polynômiale, c'est-à-dire le produit de convolution défini par :

$$(f * g)(X) = \sum_{i=0}^{N-1} h_i X^i$$

Avec :

$$\forall\ 0 \leqslant k \leqslant N-1, \quad h_k = \sum_{i+j=k \mod n} f_i g_j = \sum_{i=0}^{k} f_i g_{k-i} + \sum_{i=k+1}^{N-1} f_i g_{N+k-i}$$

2.1 NTRUEncrypt

2.1.1 Paramètres

NTRUEncrypt a les paramètres suivants :
- N le degré du polynôme qui définit l'anneau de base de NTRU.
- q appelé le grand modulo.
- p appelé le petit modulo qui peut être un entier ou un polynôme.
- D_f et D_g les espaces de clés privées qui sont en réalité des ensembles de petits polynômes dans lesquels les clés privées sont choisies.

– D_m l'espace des messages en clair qui est un ensemble de petits polynômes qui représentent le message à chiffrer. En pratique, les coefficients des polynômes qui sont dans D_m sont dans $\{0, \ldots, p-1\}$.
– D_r l'espace des valeurs aléatoires qui est un ensemble de polynômes dans lequel une valeur temporaire aléatoire est choisie lors du chiffrement.
– une fonction qui permet de centrer les éléments modulo q et ainsi permettre une meilleure performance lors du déchiffrement

Selon les versions, les ensembles D_f, D_g et D_r sont égaux aux ensembles :

$$\mathcal{B}_N(d) = \left\{ f \in \mathcal{R} \mid f = \sum_{i=1}^{d} X^{n_i},\ 0 \leqslant n_1 < \ldots < n_d \leqslant N \right\}$$

Donc les éléments de $\mathcal{B}_N(d)$ ont exactement d cœfficients à 1 et le reste à 0. Ou :

$$\mathcal{T}_N(d) = \left\{ f \in \mathcal{R} \mid f = \sum_{i=1}^{d} X^{n_i} - \sum_{j=1}^{d-1} X^{m_j},\ n_i \neq m_j \right\}$$

Ainsi, les éléments de $\mathcal{T}_N(d)$ ont d cœfficients à 1, $d-1$ cœfficients à -1 et le reste à 0.

2.1.2 Utilisation de NTRUEncrypt

Supposons que Bob veut envoyer un message m à Alice. Tout d'abord, Alice doit générer une clé publique h et des clés privées f et f_p. Puis, Bob doit chiffrer le message m en un message chiffré e grâce à h et l'envoyer à Alice. Enfin, Alice doit déchiffrer e avec f et f_p et retrouver m.

2.1.2.1 Génération des clés

Alice choisit N, p et q et applique l'algorithme suivant pour générer les clés privées et publique.

1: Choisir aléatoirement des polynômes f et g dans D_f et D_g respectivement.
2: Inverser $f \mod q$ pour obtenir f_q et inverser $f \mod p$ pour obtenir f_p (plus de détails en partie 2.1.5)
3: $h \leftarrow p * g * f_q \mod q$
4: **Retourner** la clé publique, h, et les clés privées f et f_p

2.1.2.2 Chiffrement

Bob doit tout d'abord transformer son message m en un polynôme de D_m et ensuite suivre l'algorithme suivant :

1: Choisir aléatoirement un polynôme $r \in D_r$.

2: $e \leftarrow r * h + m \mod q$
3: **Retourner** e

2.1.2.3 Déchiffrement

Alice doit utiliser les clés f et f_p dans la procédure suivante :

1: $a \leftarrow f * e$
2: Recentrer les cœfficients dans un intervalle de longueur q (avec la fonction définie dans la partie 2.1.4)
3: $m \leftarrow f_p * a \mod p$
4: **Retourner** m

2.1.2.4 Exactitude du déchiffrement

On a la relation :

$$a = f * e \mod q$$

Or $e = r * h + m \mod q$ et $h = p * g * f_q \mod q$. D'où :

$$\begin{aligned} a &= f * e \mod q \\ a &= f * (r * h + m) \mod q \\ a &= f * r * (p * g * f_q) + f * m \mod q \\ a &= p * r * g + f * m \mod q \end{aligned}$$

Comme $p * r * g + f * m$ est effectué avec des polynômes qui ont de petits coefficients, avec un choix approprié de paramètres et de la fonction qui recentre les cœfficients modulo q (partie 2.1.4), on a une égalité dans $\mathcal{R}$ c'est-à-dire sans modulo q. En réduisant $a \mod p$, on obtient $a = f * m \mod p$. Alors

$$f_p * a = f_p * f * m = m \mod p$$

2.1.3 D'un point de vue des réseaux

Toutes ces opérations de génération de clés, de chiffrement et de déchiffrement peuvent être vue d'un point de vue des réseaux. En effet, NTRU est basé sur le réseau :

$$M_{h,q} = \{(u, v) \in \mathcal{R}^2 \mid v = u * h \mod q\}$$

L'opération de génération des clés consiste donc à créer ce réseau à partir de la clé publique h et tel que le vecteur (f, g), choisi aléatoirement, soit dans le réseau. On fait donc $h = f^{-1} * g \mod q$[1] et ainsi on a :

$$g = f * h \mod q$$

[1] On considère ici que l'opération de multiplication par p est effectuée lors du chiffrement.

Et ainsi (f, g) appartient au réseau de NTRU.

L'opération de chiffrement consiste à créer un point qui ne sera pas sur le réseau mais qui sera proche d'un point du réseau. Pour cela, on choisit un point aléatoire du réseau, le point $(r, r*h \mod q)$ et on le somme avec $(0, m)$ pour s'éloigner de ce point du réseau. On a ainsi le point $(r, e = r * h + m \mod q) \in \mathbb{R}^{2N}$.

Pour déchiffrer le message, il faut donc résoudre un CVP. Alice possède le vecteur f tel que $h = f^{-1} * g$, elle calcule donc $(0, a) = (0, f) * (r, e)$ et récupère ainsi la distance au point du réseau (r, e) convolé à f c'est-à-dire $f * m$ (car le reste va « disparaître » avec le modulo p). Elle obtient m en calculant $f_p * a \mod p$.

2.1.4 Définition de la fonction qui recentre les cœfficients modulo q

La fonction qui recentre les cœfficients modulo q est capitale pour déchiffrer correctement. En effet, elle permet de passer d'une égalité modulo q à une égalité dans $\mathbb{Z}$ et ainsi permettre d'avoir le bon résultat lorsque l'on effectue une réduction modulo p. Cette fonction a besoin des paramètres suivants :

- N, q et p
- $a(1)$ avec $a = f * e$ lors du déchiffrement
- $f(1)$, $g(1)$, $f_q(1)$ et $r(1)$ qu'Alice envoie à Bob.

A l'aide de ces valeurs, on calcule :

$$I = f_q(1) \times (a(1) - p(1) \times r(1) \times g(1)) \mod q$$

Puis, on calcule :

$$J = \frac{p(1) \times r(1) \times g(1) + I \times f(1)}{N}$$

L'intervalle pour recentrer les cœfficients sera alors $\left[J - \frac{q}{2}, J + \frac{q}{2}\right]$. On aura ainsi a avec des cœfficients dans $\mathbb{Z}$. Notons a' ce a recentré. Cependant, dans certains cas, les cœfficients ne seront pas « recentrables » . Cela arrive si pour $a' \in \mathbb{Z}$:

$$||a'||_\infty = \max_{0 \leqslant i < N} a'_i - \min_{0 \leqslant i < N} a'_i \geqslant q$$

Ainsi le déchiffrement échouera.

2.1.5 Inversion

Les paramètres de NTRUEncrypt imposent que $PGCD(p, q) = 1$ dans $\mathcal{R}$, mais rien n'impose que q ou p soit premier, q est même souvent en

pratique une puissance de 2. Pour inverser ($f \mod q$) sur $\mathcal{R}$, il faut donc d'abord s'assurer que $f \mod q$ est inversible. On s'assure de cela en calculant $PGCD(f \mod q, X^N - 1)$ et s'il est différent de 1 alors $f \mod q$ n'est pas inversible sur $\mathcal{R}$.
Si q est une puissance de 2, on va d'abord calculer l'inverse de $f \mod 2$. On va utiliser l'algorithme 1 pour calculer l'inverse de $f \mod q$ avec q premier.

Algorithme 1 Inversion de $f \mod q$ avec q premier dans $\mathcal{R}$

Entrées: a l'élément à inverser, N et q le modulo des cœfficients de a
Sorties: $a^{-1} \mod q$ sur $\mathcal{R}$

1: $c = 0$, $b = 1$, $f = a$, $g = X^N - 1$, $k = 0$
2: **Boucler**
3: **Tant que** $f_0 = 0$ **Faire**
4: $f(X) = f(X)/X$ et $c(X) = c(X) * X$
5: $k = k + 1$
6: **Fin tant que**
7: **Si** degré de $f = 0$ **Alors**
8: **Retourner** $f_0^{-1} * b * x^{N-k}$
9: **Fin si**
10: **Si** degré de $f <$ degré de g **Alors**
11: échanger f et g et échanger b et c
12: **Fin si**
13: $u = f_0 \times g_0^{-1}$
14: $f = f - u * g$ et $b = b - u * c$
15: **Fin boucle**

Puis on va utiliser l'algorithme 2 pour avoir $f^{-1} \mod q$.

Algorithme 2 Inversion de $a \mod q$ quand $q = 2^l$ dans $\mathcal{R}$

Entrées: a l'élément à inverser, N et q le modulo des cœfficients de a
Sorties: $a^{-1} \mod q$ sur $\mathcal{R}$

1: $l = \log_2 q$
2: $s = 2$
3: $b = a^{-1} \mod 2$ puis $b = b \mod 2^l$
4: **Tant que** $s < 2^l$ **Faire**
5: $s = s^2$
6: $b = b * (2 - a * b)$
7: **Fin tant que**

Cet algorithme calcule bien $f^{-1} \mod X^N - 1$. Tout d'abord si q est premier, l'idée est que l'algorithme commence avec $(f, g) = (a, m)$ avec

$m = X^N - 1$ que l'on va multiplier à droite par les matrices :

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad B = \begin{pmatrix} X^{-1} & 0 \\ 0 & 1 \end{pmatrix} \quad C_u = \begin{pmatrix} 1 & 0 \\ -u & 1 \end{pmatrix}$$

On obtient donc :

$$(f,g)A = (g,f) \quad (f,g)B = (X^{-1}f, g) \quad (f,g)C_u = (f - ug, g)$$

La première transformation a lieu à l'étape 4 de l'algorithme, la deuxième à l'étape 11, et la troisième à l'étape 14. De plus, à l'étape 14, u est choisie pour que $f - ug$ soit divisible par X. L'algorithme produit une séquence de transformations $D_1, D_2, \ldots, D_r$ où chaque D_i est l'un des A, B, ou C_u tel que :

$$(a,m)D_1D_2 \ldots D_{r-1}D_r = (\alpha, *)$$

avec α un entier modulo q non nul. Malheureusement, les cœfficients du produit $D_1D_2 \ldots D_r$ ne sont pas polynômiaux car B a un X^{-1}. Appelons k le nombre de fois où B apparaît dans le produit $D_1D_2 \ldots D_r$. Alors $X^k D_1D_2 \ldots D_r$ a des cœfficients polynômiaux tel que :

$$X^k D_1D_2 \ldots D_r = \begin{pmatrix} a' & * \\ m' & * \end{pmatrix}$$

La multiplication à gauche de (a,m) donne :

$$\begin{aligned} (aa' + mm', *) &= (a,m)\begin{pmatrix} a' & * \\ m' & * \end{pmatrix} \\ &= (a,m)X^k D_1D_2 \ldots D_r \\ &= X^k(\alpha, *) \end{aligned}$$

Donc nous avons :

$$aa' = \alpha X^k \mod m$$

Pour construire cette valeur a', l'algorithme applique les transformations $D_1, D_2, \ldots, D_r$ à $(b,c) = (1,0)$ sauf qu'à la place de B il applique XB. Si B est appliqué k fois, à la fin de l'algorithme la valeur de (b,c) est :

$$(b,c) = (1,0)X^k D_1D_2 \ldots D_r = (1,0)\begin{pmatrix} a' & * \\ m' & * \end{pmatrix} = (a', *)$$

La valeur b satisfait donc à la fin de l'algorithme :

$$ab = \alpha X^k \mod m$$

Comme la valeur de α est simplement f_0, nous avons $a^{-1} = f_0^{-1} X^{N-k} b$.

L'algorithme se termine car à chaque tour de boucle, le degré de $f+g$ diminue d'au moins 1 jusqu'à ce que f devienne une constante. Il se termine donc après au plus $deg(a)+deg(m)$ itérations c'est-à-dire $N-1+N = 2N-1$ dans notre cas.

Puis, si $q = 2^l$, on calcule d'abord $f^{-1} \mod 2$, puis une méthode simple basée sur l'itération de Newton nous donne $f^{-1} \mod 2^l$.

2.1.6 Schéma de padding : NAEP

Pour se protéger contre les attaques à chiffrés choisis, un schéma de padding approprié doit être défini. Ce schéma, dont nous allons donner les grandes lignes, est décrit plus en détail en [7] et donne des preuves de sécurité dans le modèle de l'oracle aléatoire.
NAEP utilise deux fonctions de hachage :

$$G : \{0,1\}^{N-l} \times \{0,1\}^l \longrightarrow D_r \quad H : \{0,1\}^N \longrightarrow \{0,1\}^N$$

Avant de chiffrer un message $M \in \{0,1\}^{N-l}$, on doit le « remplir » en utilisant NAEP et on doit donc utiliser les fonctions suivantes :

$$compress(x) = (x \mod q) \mod 2$$

$$B2P : \{0,1\}^N \longrightarrow D_m \cup \text{"}erreur\text{"}, \quad P2B : D_m \longrightarrow \{0,1\}^N$$

Le rôle de la fonction *compress* est simplement de réduire la taille des entrées pour la fonction de hachage H pour gagner de l'efficacité pratique. La fonction $B2P$ convertit une suite de bits en un polynôme binaire et retourne « erreur » si la suite de bits ne remplit pas les conditions appropriées. La fonction $P2B$ convertit un polynôme binaire en une suite de bits.
L'algorithme de chiffrement intégrant le padding est maintenant l'algorithme suivant :

1: Choisir au hasard b tel que $b \in \{0,1\}^l$
2: $r \leftarrow G(M, b)$
3: $m \leftarrow B2P((M||b) \oplus H(compress(r * h)))$
4: **Si** $B2P$ retourne « erreur » **Alors**
5: aller à l'étape 1
6: **Fin si**
7: $e \leftarrow r * h + m \mod q$
8: **Retourner** e

Pour déchiffrer un message e, on doit effectuer l'algorithme suivant.

1: $a \leftarrow f * e \mod q$
2: Recentrer les cœfficients dans $[A, A+q-1]$
3: $s \leftarrow e - m$
4: $M||b \leftarrow P2B(m) \oplus H(compress(P2B(s)))$

5: $r \leftarrow G(M, b)$
6: **Si** $r * h = s \mod q$ et $m \in D_m$ **Alors**
7: **Retourner** M
8: **Sinon**
9: **Retourner** « chiffré invalide »
10: **Fin si**

Pour mettre en œuvre NAEP, on utilise SVES-3, qui a les spécificités suivantes :

- Pour permettre des messages de longueur variable, un codage sur un octet de la longueur du message en octets est ajouté au message. Le message est complété de zéros pour remplir le bloc du message.
- La fonction de hachage G qui est utilisé pour produire r prend pour entrées M, b, un identifiant (OID) qui identifie le schéma de padding et l'ensemble des paramètres et une chaîne h_{trunc} dérivée de la troncature de la clé publique afin d'avoir une longueur de l_h bits.

SVES-3 inclue h_{trunc} dans G tel que r dépend de la clé publique spécifique. Même si un attaquant trouve un (M, b) qui donne un r avec une bonne probabilité d'erreur de déchiffrement, (M, b) ne s'appliquerait qu'à une seule clé publique et ne pourrait pas être utilisée sur d'autres clés publiques. Néanmoins, les paramètres actuellement recommandés n'ont pas d'erreur de déchiffrement et donc il n'y a pas besoin de l'entrée h_{trunc} dans G. On utilise donc actuellement SVES-3 mais avec $l_h = 0$.

Nous allons maintenant voir l'algorithme de signature nommé NTRUSign.

2.2 NTRUSign

2.2.1 Paramètres

Les paramètres de NTRUSign sont :

- N le degré du polynôme qui définit l'anneau de base de NTRU.
- q le modulo des cœfficients des polynômes
- D_f et D_g l'espace des polynômes f et g utilisés dans la génération des clés. D_f et D_g sont construits à partir de $\mathcal{T}_N(d)$ comme défini dans NTRUEncrypt.
- $||\ .\ ||$ une norme, le plus souvent la norme centrée euclidienne.
- $\mathcal{N}$ la « limite » de la norme utilisée pour vérifier la signature.
- β le facteur d'équilibrage pour la norme. On a $0 < \beta \leqslant 1$.
- H une fonction de hachage utilisée pour signer.
- B le nombre de perturbations dans la signature

2.2.2 Utilisation de NTRUSign

Supposons qu'Alice veut envoyer un document D à Bob en le signant. Tout d'abord, Alice doit générer une clé publique h et un ensemble de clés privées $\{f_i, f'_i, h_i\},\ 0 \leqslant i \leqslant B$ avec B le nombre de perturbations de la signature. Puis, Alice doit signer le document D et générer ainsi la signature $[D, r, s]$ grâce à $\{f_i, f'_i, h_i\}$ et l'envoyer à Bob. Enfin, Bob doit vérifier la signature $[D, r, s]$ grâce à h et s'assurer ainsi que c'est bien Alice qui lui a envoyé ce document.

2.2.2.1 Génération des clés

Alice doit choisir le nombre de perturbations de la signature $B \geqslant 0$, et suivre l'algorithme suivant.

Générer B bases privées et une base publique du réseau :
$i \leftarrow B$
Tant que $i \geqslant 0$ **Faire**
  Choisir f et g dans D_f et D_g respectivement.
  Trouver de petits $F, G \in \mathcal{R}$ tel que $f * G - F * g = q$ comme expliqué dans 2.2.3.
  $f_i \leftarrow f$, $f'_i \leftarrow g$ et $h_i \leftarrow f_i^{-1} * F \mod q$
  $i \leftarrow i - 1$
Fin tant que
Retourner la clé publique h_0 et les clés privées $(f_i, f'_i, h_i),\ 0 \leqslant i \leqslant B$.

2.2.2.2 Signature

Pour signer le document, Alice a besoin du document D, d'une fonction de hachage H, de la limite de la norme $\mathcal{N}$ et du facteur d'équilibrage β ainsi que de l'ensemble des clés privées $\{f_i, f'_i, h_i\}$ et de la clé publique h puis elle applique l'algorithme suivant.

$r \leftarrow 0$
$s \leftarrow 0$ et $i \leftarrow B$
Transformer r en un ensemble de bits.
$m_0 \leftarrow H(D||r)$ avec $||$ la concaténation.
$m \leftarrow m_0$
Perturber le point en utilisant les clés privées :
Tant que $i \geqslant 1$ **Faire**
  $x \leftarrow -\frac{m}{q} * f'_i$
  $y \leftarrow \frac{m}{q} * f_i$
  $\epsilon_x \leftarrow -\{x\} = \lfloor x \rceil - x$
  $\epsilon_y \leftarrow -\{y\} = \lfloor y \rceil - y$
  $s_i \leftarrow \epsilon_x * f_i + \epsilon_y * f'_i$
  $m \leftarrow s_i * (h_i - h_{i-1}) \mod q$

14: $s \leftarrow s + s_i$
15: $i \leftarrow i - 1$
16: **Fin tant que**
17: *Signer le point perturbé en utilisant la clé publique :*
18: $x \leftarrow -\frac{m}{q} * f'_0$
19: $y \leftarrow \frac{m}{q} * f_0$
20: $\epsilon_x \leftarrow -\{x\} = \lfloor x \rceil - x$
21: $\epsilon_y \leftarrow -\{y\} = \lfloor y \rceil - y$
22: $s_0 \leftarrow \epsilon_x * f_0 + \epsilon_y * f'_0$
23: $s \leftarrow s + s_0$
24: *Vérifier la signature :*
25: $b \leftarrow ||(s, \beta(s * h - m_0 \mod q))||$
26: **Si** $b \geqslant \mathcal{N}$ **Alors**
27: $r \leftarrow r + 1$
28: aller à l'étape 2
29: **Sinon**
30: **Retourner** $[D, r, s]$
31: **Fin si**

2.2.2.3 Vérification de la signature

Pour vérifier la signature, Bob a besoin de la même fonction de hachage H, de la même norme, de la même limite de norme $\mathcal{N}$, du même facteur d'équilibrage β, de la clé publique h d'Alice et bien sûr du triplé $[D, r, s]$ envoyé par Alice et il doit suivre la procédure suivante.

1: Transformer r en un ensemble de bits
2: $m \leftarrow H(D||r)$
3: $b \leftarrow ||(s, \beta(s * h - m \mod q))||$
4: **Si** $b < \mathcal{N}$ **Alors**
5: **Retourner** valide
6: **Sinon**
7: **Retourner** invalide
8: **Fin si**

Un des points importants de l'algorithme de génération des clés est de trouver $f * G - g * F = q$ avec F, G de petits polynômes. La partie suivante va nous indiquer la procédure pour les calculer.

2.2.3 Comment trouver $f * G - g * F = q$ avec F, G de petits polynômes ?

2.2.3.1 Trouver $f * G - g * F = q$

L'anneau sur lequel est basé NTRU est $\mathcal{R} = \dfrac{\mathbb{Z}[X]}{X^N - 1}$ qui est un réseau naturel de dimension N associé à chaque élément $r = \sum_{i=0}^{N} r_i X^i \in \mathcal{R}$, à savoir celui généré par la matrice suivante :

$$M_r = \begin{pmatrix} r_0 & r_1 & \dots & r_{N-1} \\ r_{N-1} & r_0 & \dots & r_{N-2} \\ \vdots & & \ddots & \vdots \\ r_1 & r_2 & \dots & r_0 \end{pmatrix}$$

L'application φ qui associe à chaque polynôme de $\mathcal{R}$ sa matrice circulante dans $M_n(\mathbb{Z})$ est définie par :

$$\varphi\left(\sum_{i=0}^{N} r_i X^i\right) = M_r$$

C'est un isomorphisme d'anneau.
Pour $q \in \mathbb{Z}$ et $h \in \mathcal{R}$, l'ensemble :

$$M_{h,q} = \{(u, v) \in \mathcal{R}^2 \mid v = u * h \mod q\}$$

est un réseau de dimension $2N$.

La mesure naturelle dans $\mathcal{R}$ est la norme euclidienne centrée des vecteurs des cœfficients :

$$\|r\|^2 = \sum_{i=0}^{N-1} r_i^2 - \left(\frac{1}{N}\right)\left(\sum_{i=0}^{N-1} r_i\right)^2$$

La norme des éléments de $M_{h,q}$ ou plus généralement de $(u, v) \in \mathcal{R}^2$ est la norme euclidienne par composant :

$$\|(u, v)\|^2 = \|u\|^2 + \|v\|^2$$

On pose $h = f^{-1} * g \mod q$ afin que (f, g) soit dans $M_{h,q}$.

Il y a une base évidente de $M_{h,q}$ qui est $\{(1, h), (0, q)\}$. Si en plus, $(f, g) \in M_{h,q}$, il est naturel de se demander si on peut trouver un autre vecteur $(F, G) \in M_{h,q}$ tel que la paire $\{(f, g), (F, G)\}$ est aussi une base de $M_{h,q}$. Cela est possible si et seulement si

$$PGCD(resultant(f, X^N - 1), resultant(g, X^N - 1)) = 1$$

La stratégie générale pour compléter la base de $M_{h,q}$ est de projeter f et g dans $\mathbb{Z}$ par le résultant. La définition du résultant de deux polynômes $P(X) = a_n \prod_{r=1}^{n}(X - \alpha_r)$ et $Q(X) = b_m \prod_{j=1}^{m}(X - \beta_j)$ sur une extension du corps permettant de scinder les polynômes est :

$$Res(P, Q) = a_n^m \prod_{r=1}^{n} Q(\alpha_r)$$

Dans notre cas, on pose :

$$Q(X) = f(X) = \prod_{j=1}^{N-1}(X - \beta_j)$$

$$P(X) = X^N - 1 = \prod_{r=1}^{N}(X - \alpha_r) = (X - 1)\phi(X)$$

avec $\phi(X) = \sum_{i=0}^{N-1} X^i \in \mathcal{R}$ et on a ainsi :

$$R_f = \prod_{r=1}^{N-1} f(\alpha_r)$$

Or $\phi(X)$ est un polynôme cyclotomique donc les α_r sont des racines de l'unité. En ajoutant la racine de $(X - 1)$ on obtient le groupe des racines de l'unité :

$$\mathbb{U}_N = \left\{1, e^{\frac{2i\pi}{N}}, e^{\frac{4i\pi}{N}}, \ldots, e^{\frac{(2N-2)i\pi}{N}}\right\}$$

On peut construire le morphisme de groupe :

$$\begin{array}{rccl} \varphi & : & \mathbb{U}_N & \rightarrow & \mathcal{R} \\ & & \alpha_r & \mapsto & x^r \mod \phi \end{array}$$

On vérifie que $\varphi(\alpha_r) \times \varphi(\alpha_j) = \varphi(\alpha_r \alpha_j)$.
En effet :

$$\alpha_r \alpha_j = \alpha_{(r+j) \mod N}$$

Donc :

$$\begin{array}{rcl} \varphi(\alpha_r \alpha_j) & = & \varphi(\alpha_{(r+j) \mod N}) \\ & = & x^{(r+j) \mod N} \mod \phi = x^{r+j} \mod \phi \\ & = & (x^r \mod \phi) \times (x^j \mod \phi) \\ & = & \varphi(\alpha_r) \times \varphi(\alpha_j) \end{array}$$

D'où :

$$R_f = \prod_{r=1}^{N-1} f(\alpha_r) = \prod_{i=1}^{N-1} f(x^i) \mod \phi \in \mathbb{Z}$$

Par conséquent, on peut définir :

$$\rho_f = \prod_{i=2}^{N-1} f(x^i) \mod \phi$$

et ρ_g de manière similaire. On connaît maintenant que pour certains $k_f, k_g \in \mathbb{Z}[X]$:

$$\rho_f f + k_f(X^N - 1) = R_f$$
$$\rho_g g + k_g(X^N - 1) = R_g$$

Si R_f et R_g sont premiers entre eux dans les entiers, on peut maintenant utiliser l'algorithme d'Euclide étendu pour trouver α, $\beta \in \mathbb{Z}$ tel que $\alpha R_f + \beta R_g = 1$ et dans ce cas là nous avons :

$$(\alpha\rho_f)f + (\beta\rho_g)g = 1 + k(X^N - 1)$$

Donc si nous faisons $F = -q\beta\rho_g$ et $G = q\alpha\rho_f$ alors

$$f * G - g * F = q$$

Notation : La matrice :

$$\begin{pmatrix} 1 & a \\ c & b \end{pmatrix}$$

avec $a \in \mathcal{R}$ et $b, c \in \mathbb{Z}$ désigne la matrice $2N \times 2N$:

$$\left(\begin{array}{cccc|cccc} 1 & 0 & \dots & 0 & a_0 & a_1 & \dots & a_{N-1} \\ 0 & 1 & \dots & 0 & a_{N-1} & a_0 & \dots & a_{N-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & a_1 & a_2 & \dots & a_0 \\ \hline c & 0 & \dots & 0 & b & 0 & \dots & 0 \\ 0 & c & \dots & 0 & 0 & b & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & c & 0 & 0 & \dots & b \end{array}\right)$$

Théorème :

Si $f, g, F, G \in \mathcal{R}$ satisfont l'équation précédente, prenons $h = f^{-1} * g \mod q$ et prenons $M_{h,q}$ comme le réseau généré par la matrice $\begin{pmatrix} 1 & h \\ 0 & q \end{pmatrix}$ alors :

1. $\begin{pmatrix} f & g \\ F & G \end{pmatrix}$ forme une base de $M_{h,q}$
2. Si $F', G' \in \mathcal{R}$ satisfait $f * G' - g * F' = q$ alors il existe un élément $c \in \mathcal{R}$ tel que $F' = F + c * f$ et $G' = G + c * g$

Preuve :

1. Il suffit de voir que la matrice de passage B suivante a ses cœfficients dans $\mathcal{R}$ et a son déterminant égal à 1 :

$$\begin{aligned} B &= \begin{pmatrix} 1 & h \\ 0 & q \end{pmatrix} \begin{pmatrix} f & g \\ F & G \end{pmatrix}^{-1} \\ &= \frac{1}{q} \begin{pmatrix} 1 & h \\ 0 & q \end{pmatrix} \begin{pmatrix} G & -g \\ -F & f \end{pmatrix} \\ &= \begin{pmatrix} \frac{G-F*h}{q} & \frac{-g+f*h}{q} \\ -F & f \end{pmatrix} \end{aligned}$$

   $h = f^{-1} * g \mod q$ assure que $\frac{-g+f*h}{q} \in \mathcal{R}$ et l'équation $f*G-g*F = q$ implique que $F^{-1}*G = f^{-1}*g \mod q$ donc nous avons aussi $\frac{G-F*h}{q} \in \mathcal{R}$. Cela prouve que B a ses cœfficients dans $\mathcal{R}$ et on peut facilement calculer que :

$$\det(B) = \frac{f*G - g*F}{q} = 1$$

   Donc B^{-1} a aussi ses cœfficients dans $\mathcal{R}$.
   Par conséquent, $\begin{pmatrix} 1 & h \\ 0 & q \end{pmatrix}$ et $\begin{pmatrix} f & g \\ F & G \end{pmatrix}$ génèrent le même réseau.

2. Observons tout d'abord que $F' * G = F * G' \mod q$ donc $F^{-1} * G = F'^{-1} * G' = f^{-1} * g = h \mod q$. Posons $c = \frac{F'*G-G'*F}{q}$. Alors :

$$\begin{aligned} F + c*f &= F + \frac{F'*G*f - G'*F*f}{q} \\ &= F + \frac{F'*(q+g*F) - F*(q+g*F')}{q} \\ F + c*f &= F' \end{aligned}$$

   De manière similaire, on montre que $G + c*g = G'$ ce qui complète la preuve de ce théorème.

   Bien que F et G complètent une base de $M_{h,q}$, ils ont généralement des cœfficients très grands. Cependant, on peut supprimer n'importe quel $\mathcal{R}$-multiple du vecteur (f, g) de (F, G) et avoir encore une base de $\mathcal{R}$. Dans la prochaine partie, nous allons voir comment trouver ce $\mathcal{R}$-multiple idéal pour réduire.

2.2.3.2 Réduire F et G

Pour réduire F, il suffit de trouver un $k \in \mathcal{R}$ tel que $F_{red} = \|F - k*f\|$ est petite. Nous savons que $f^{-1} = \frac{\rho_f}{R_f} \in \frac{\mathbb{Q}[X]}{X^N-1}$, et si on prend k égal à

$l = F * f^{-1} \in \frac{\mathbb{Q}[X]}{X^N - 1}$, F_{red} devrait être égal à zéro. Pour obtenir $k \in \mathcal{R}$ nous prendrons $k = \lfloor l \rceil$ l'arrondi de l vers son plus proche entier.
On peut réduire G de la même manière, en calculant $G - k * g$. En effet,

$$\begin{aligned} & f * G - g * F = q \\ \Leftrightarrow \quad & f * G - g * (F - k * f + k * f) = q \\ \Leftrightarrow \quad & f * G - g * (F - k * f) - g * k * f = q \\ \Leftrightarrow \quad & f * (G - k * g) - g * (F - k * f) = q \end{aligned}$$

Cependant dans NTRUSign, on préfère utiliser $h = F * f^{-1} \mod q$ plutôt que $h = g * f^{-1} \mod q$ pour avoir une meilleure efficacité. Cela consiste donc à inverser les rôles de F et de g et ainsi :

$$\begin{pmatrix} 1 & h \\ 0 & q \end{pmatrix} \quad et \quad \begin{pmatrix} f & F \\ g & G \end{pmatrix}$$

génèrent le même réseau.
Ce changement n'affecte pas l'équation $f * G - F * g = q$, et ainsi tout ce qui a été montré précédemment est valable avec cette nouvelle clé publique h.

2.2.4 Pourquoi NTRUSign fonctionne ?

On a montré dans la partie 2.2.3 que

$$\begin{pmatrix} f & F \\ g & G \end{pmatrix} \quad et \quad \begin{pmatrix} 1 & h \\ 0 & q \end{pmatrix}$$

étaient des bases d'un même réseau.

La signature consiste à trouver un point proche du point du message $(0, m)$. Pour cela, on exprime le point cible comme une combinaison réelle des vecteurs de la base, et on trouve un point proche du réseau en arrondissant les cœfficients réels afin d'obtenir des combinaisons entières des vecteurs de la base. L'erreur introduite par ce processus sera la somme des erreurs d'arrondi de chacun des vecteurs de la base, et une erreur d'arrondi est par définition comprise entre $-\frac{1}{2}$ et $\frac{1}{2}$. Dans NTRUSign, les vecteurs de la base sont tous de même longueur alors l'erreur attendue introduite par $2N$ arrondis de ce type sera $\sqrt{\frac{N}{6}}$ fois cette longueur.

Dans NTRUSign, la base privée est choisie telle que $\|f\| = \|g\|$ et $\|F\| \sim \|G\| \sim \sqrt{\frac{N}{12}}\|f\|$. Cette approximation des normes est obtenue avec le théorème 2.2.4.

Théorème[2] **:**
Soient $X_1, \ldots, X_N$ des variables aléatoires indépendantes uniformément distribuées dans $\left[-\frac{B}{2}, \frac{B}{2}\right]$ et soit $Y = X_1^2 + \ldots + X_N^2$. Alors la moyenne et l'écart-type de Y sont donnés par :

$$\mu(Y) = \frac{NB^2}{12} \quad et \quad \sigma(Y) = \frac{B^2}{6}\sqrt{\frac{N}{5}}$$

On voit qu'avec $B = 1$, ce théorème implique que la norme euclidienne de l est $\sqrt{\frac{N}{12}}$ et la norme euclidienne centrée sera un peu moins et donc en utilisant la pseudo-propriété multiplicative on a $\|F - k * f\| \approx \sqrt{\frac{N}{12}}\|f\|$. L'erreur attendue en signant sera par conséquent

$$\sqrt{\frac{N}{6}}\|f\| + \beta\sqrt{\frac{N}{6}}\|f\|\sqrt{\frac{N}{12}}\|f\| = \sqrt{\frac{N}{6}}\|f\| + \beta\frac{N}{6\sqrt{2}}\|f\|$$

En revanche, un attaquant qui utiliserait seulement la clé publique produira probablement une signature avec N cœfficients incorrects, et ces cœfficients seront distribués aléatoirement modulo q. L'erreur attendue dans la génération de la signature avec la clé publique sera donc :

$$\beta\sqrt{\frac{N}{12}}q$$

Il est donc clair qu'il est possible de choisir $\|f\|$ et q tels que la connaissance de la base privée permet la création de plus petites erreurs de signature que la connaissance de la seule base publique. Par conséquent, en veillant à ce que l'erreur de signature soit plus petite que celle attendue par la production d'une signature connaissant seulement la base publique, le destinataire peut vérifier que la signature a été produite par le propriétaire de la base privée et est donc valide.

2.2.5 Perturbations de la signature

On peut voir que lorsque $B = 0$, NTRUSign décrit simplement la résolution d'un ACVP en utilisant la limite de la norme $\mathcal{N}$ dans $M_{h,q}$. Cependant, dans le réseau NTRU, il suffit de donner la première coordonnée de s comme la signature, car tous les points sont de la forme $(s, h * s + kq)$ et le k qui fait que le point est proche de $(0, m)$ autant que possible, peut être facilement obtenu par une réduction modulo q. Nous verrons dans la section 4.4 que NTRUSign n'est pas sans-divulgation et qu'un attaquant peut faire la moyenne d'un nombre de signatures fini pour obtenir la clé privée. On montrera également que les moyennes impliquées convergent assez rapidement pour un algorithme de signature basé seulement sur un seul ACVP.

[2]Une preuve de ce théorème est donnée dans [8]

Pour avoir un schéma pratique, on doit augmenter le nombre de signatures nécessaires.
Pour un algorithme de signature général, on propose que le signataire hache tout d'abord le document D pour obtenir un point m, puis perturbe ce point en $m' = m + \epsilon$ avec ϵ qui est différent pour chaque message et inconnu du vérificateur. Le signataire signe alors m' et si ϵ est assez petit, une signature de m' sera aussi une signature valide de m. Par conséquent, dans NTRUSign, on propose que les perturbations soient générées en utilisant un processus de signature dans un ou plusieurs réseaux secrets de géométrie similaire au réseau public.
En pratique, l'utilisation de la perturbation est déjà spécifiée dans l'algorithme de signature précédemment énoncé. On démarre avec un point $(0, m)$ et on trouve un point proche (s_B, t_B) à celui-ci dans le réseau généré par $\{(1, h_B), (0, q)\}$. On a seulement besoin de la première coordonnée car $t_B = s_B * h_B \mod q$.
Pour chaque base B_i, on veut trouver le point le plus proche (s_i, t_i) de (s_{i+1}, t_{i+1}) dans le réseau généré par $\{(1, h_i), (0, q)\}$. On peut de nouveau transformer (s_B, t_B) par un vecteur du réseau proche d'un point $(0, m')$ par

$$m' = t_B - s_B * h_{B-1} \mod q = s_B(h_B - h_{B-1}) \mod q$$

Si on considère seulement la première coordonnée alors la différence entre le point donné et le point du réseau est s_i pour chaque $i = B \dots 0$. Donc on peut définir :

$$s = \sum_{i=0}^{B} s_i$$

et le point final de la signature est $(s, h_0 * s \mod q)$.

Maintenant que les deux algorithmes de chiffrement et de signature sont posés, nous allons voir s'il est possible de les attaquer. Pour cela, des notions préalables de réduction des réseaux sont nécessaires, c'est à quoi le chapitre suivant est consacré.

Chapitre 3

Réduction des réseaux

Certaines attaques sur NTRU sont des attaques sur le réseau, c'est-à-dire qu'on utilise un algorithme pour tenter d'approcher le plus court vecteur du réseau et ainsi retrouver les clés privées. Cet algorithme est l'algorithme LLL, dont nous allons voir dans la suite sur quoi il est basé et comment il fonctionne. Pour cela, nous avons d'abord besoin d'introduire l'orthogonalisation de Gram-Schmidt.

3.1 Orthogonalisation de Gram-Schmidt

Soit $\{b_1, \ldots, b_d\}$ linéairement indépendants dans $\mathbb{R}^n$.
Posons :

- $b_1^* = b_1$
- $b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^*$ avec :
  - $\mu_{i,i} = 1$
  - $\forall\ i\ tel\ que\ 2 \leqslant i \leqslant d,\ \mu_{i,j} = \dfrac{\langle b_i, b_j^* \rangle}{\|b_j^*\|^2}$

Alors $(b_1^*, \ldots, b_d^*)$ est l'orthogonalisé de Gram-Schmidt de la base $(b_1, \ldots, b_d)$.

Elle satisfait les propriétés suivantes :

- Elle est orthogonale c'est-à-dire $\forall\ i \neq j,\ \langle b_i^*, b_j^* \rangle = 0$.
- Pour tout $i \geqslant 1$, $\{b_1^*, \ldots, b_i^*\}$ engendre le même sous-espace vectoriel que $\{b_1, \ldots, b_i\}$.
- Pour tout i, $b_i^* - b_i \in Vect(b_1, \ldots, b_{i-1})$.

Interprétation matricielle

On a la relation matricielle suivante :

$$\begin{pmatrix} \leftarrow & b_1 & \rightarrow \\ & \vdots & \\ \leftarrow & b_d & \rightarrow \end{pmatrix} = \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ \mu_{i,j} & & 1 \end{pmatrix} \times \begin{pmatrix} \leftarrow & b_1^* & \rightarrow \\ & \vdots & \\ \leftarrow & b_d^* & \rightarrow \end{pmatrix}$$

On a donc $B = UB^*$ et $\det(U) = 1$ avec U à cœfficients réels. Si $(b_1, \ldots, b_d)$ est une base d'un réseau L, on utilise G la matrice de Gram, on a $G = BB^T$ d'où

$$\det(G) = \det\left(BB^T\right) = \det\left(B^*B^{*T}\right) = \det\begin{pmatrix} \|b_1^*\|^2 & & 0 \\ & \ddots & \\ 0 & & \|b_d^*\|^2 \end{pmatrix}$$

Donc $\det(G) = \prod_{i=1}^{d} \|b_i^*\|^2$ et $\det(L) = \prod_{i=1}^{d} \|b_i^*\|$. On va définir une notion de base réduite en comparant la base d'un réseau à son orthogonalisé de Gram-Schmidt. On a la définition suivante.

Définition : base réduite au sens d'Hermite

Si $(b_1, \ldots, b_d)$ est une base d'un réseau L, $(b_1^*, \ldots, b_d^*)$ est son orthogonalisé de Gram-Schmidt et si on note $\mu_{i,j}$ les cœfficients de Gram-Schmidt alors $(b_1, \ldots, b_d)$ est réduite au sens d'Hermite si :

- $|\mu_{i,j}| \leqslant \frac{1}{2}$ (condition de taille)
- $\forall\ i \leqslant 2,\ \|b_i^*\|^2 \leqslant \frac{3}{4}\|b_{i-1}^*\|^2$ (condition de quasi-orthogonalité)

Hermite donne un algorithme pour réduire une base selon les conditions de la définition, mais il n'est pas polynômial. La condition de Lovàcz est un relâchement de la condition de quasi-orthogonalité, qui peut-être obtenu par l'algorithme LLL en complexité polynômiale.

3.2 Algorithme LLL

L'algorithme LLL doit son nom à ses auteurs, Lenstra, Lenstra et Lovàcz qui l'ont écrit en 1982.

Définition : base LLL-réduite

Avec les notations de la définition précédente, une base $\{b_1, \ldots, b_d\}$ d'un réseau L est dite LLL-réduite si elle vérifie les conditions suivantes :

- $\forall\ j < i,\ |\mu_{i,j}| \leqslant \frac{1}{2}$
- Pour tout $i \leqslant 2$, $\|b_i^*\|^2 \geqslant \left(\frac{3}{4} - \mu_{i,i-1}^2\right)\|b_{i-1}^*\|^2$

De plus, elle satisfait les propriétés du théorème suivant :

Théorème :

Soit $\{b_1, \ldots, b_d\}$ une base LLL-réduite, alors :

1. $\det(L) \leqslant \prod_{i=1}^{d} \|b_i\| \leqslant 2^{\frac{d(d-1)}{4}} \det(L)$
2. $\forall\ j \leqslant i,\ \|b_j\|^2 \leqslant 2^{i-1}\|b_i^*\|^2$
3. $\|b_1\| \leqslant 2^{\frac{d-1}{4}} (\det L)^{\frac{1}{d}}$
4. $\|b_1\|^2 \leqslant 2^{d-1} \min(L)^2$

Preuve :

La définition de l'orthogonalisation de Gram-Schmidt nous permet d'avoir la relation 3.1 :

$$\|b_i\|^2 = \sum_{j=1}^{i} \mu_{i,j}^2 \|b_j^*\|^2 \tag{3.1}$$

qui montre que $\|b_i\| \geqslant \|b_i^*\|$ d'où :

$$\det(L) = \prod_{i=1}^{d} \|b_i^*\| \leqslant \prod_{i=1}^{d} \|b_i\|$$

La définition d'une base LLL-réduite montre que :

$$\|b_i^*\|^2 \geqslant \frac{\|b_{i-1}^*\|^2}{2}$$

D'où, on a la relation 3.2 :

$$\forall\ i > j,\ \|b_i^*\|^2 \geqslant \frac{\|b_j^*\|^2}{2^{i-j}} \tag{3.2}$$

En remplaçant dans l'équation 3.1, on obtient :

$$\begin{aligned} \|b_i\|^2 &\leqslant \left(1 + \frac{1}{4}\left(2 + 2^2 + \ldots + 2^{i-1}\right)\right) \|b_i^*\|^2 \\ &\leqslant \frac{2^{i-1}+1}{2} \|b_i^*\|^2 \\ &\leqslant 2^{i-1} \|b_i^*\|^2 \end{aligned}$$

Ce qui démontre les points 1 et 2.
De plus, $\|b_1\|^2 = \|b_1^*\|^2 \leqslant 2^{i-1}\|b_i^*\|^2$. En multipliant chaque coté des inégalités on obtient le point 3 :

$$\begin{aligned} \|b_1\|^{2d} &\leqslant 2^{\frac{d(d-1)}{2}} \prod_{i=1}^{d} \|b_i^*\|^2 \\ &\leqslant 2^{\frac{d(d-1)}{2}} \det(L)^2 \end{aligned}$$

$$\Rightarrow \|b_1\|^d \leqslant 2^{\frac{d(d-1)}{4}} \det(L)$$

Soit $x \in L$, $x \neq 0$, on a :

$$x = x_1 b_1 + \ldots + x_d b_d$$

avec $x_i \in \mathbb{Z}$ et on a également :

$$x = \lambda_1 b_1^* + \ldots + \lambda_n b_n^*$$

avec $\lambda_i \in \mathbb{R}$.
Soit i_0 le plus grand indice tel que $\lambda_{i_0} \neq 0$. Alors :

$$\lambda_{i_0} = x_{i_0} \quad et \quad \|x\|^2 = \lambda_{i_0}^2 \|b_{i_0}^*\|^2 + \sum_{j<i_0} \lambda_j^2 \|b_j^*\|^2 \geqslant \|b_{i_0}^*\|^2 \geqslant \frac{\|b_1\|^2}{2^{i_0-1}}$$

grâce à la relation 3.2 pour la dernière inégalité. On a donc :

$$2^{i_0-1}\|x\|^2 \geqslant \|b_1\|^2$$

Si on prend $x \in S(L)$ et $i_0 \leqslant d$, on obtient le point 4.

Algorithme 3 Algorithme LLL

Entrées: une base $(b_1, \ldots, b_d)$ d'un réseau L
Sorties: la base LLL-réduite

1: Calculer tous les cœfficients de Gram-Schmidt $\mu_{i,j}$
2: **Pour** $i = 2$ à d **Faire**
3: **Pour** $j = i - 1$ à 1 **Faire**
4: $b_i \leftarrow b_i - \lfloor \mu_{i,j} \rceil b_j$
5: **Pour** $k = 1$ à j **Faire**
6: $\mu_{i,k} = \mu_{i,k} - \lfloor \mu_{i,j} \rceil \mu_{j,k}$
7: **Fin pour**
8: **Fin pour**
9: **Fin pour**
10: **Si** $\exists j$ tel que j ne vérifie pas la condition de Lovàcz **Alors**
11: Échanger b_j et b_{j+1} et retourner à l'étape 2
12: **Fin si**

Montrons que cet algorithme termine en temps polynômial et pour ceci, posons le réseau L_j engendré par $b_1, \ldots, b_j$ et posons $D_j = \det(L_j)^2 = \prod_{i=1}^{j} \|b_j^*\|^2$. Les valeurs de $(D_1, D_2, \ldots, D_d)$ ne changent que dans l'étape 11 de l'algorithme, lorsque l'on échange b_{j-1} et b_j. Lors de cet échange, les réseau $L_1, \ldots, L_{j-2}$ et $L_j, \ldots, L_d$ restent inchangés, seul L_{j-1} est modifié. En effet, on garde $b_1^*, \ldots, b_{j-2}^*$ et b_{j-1}^* devient :

$$b_{j-1}^{'*} = b_j^* + \mu_{j,j-1} b_{j-1}^*$$

et donc $\|b^*_{j-1}\|^2$ devient :

$$\|b'^*_{j-1}\|^2 = \|b^*_j\|^2 + \mu^2_{j,j-1}\|b^*_{j-1}\|^2$$

Donc D_{j-1} est remplacé par :

$$D'_{j-1} = D_{j-1}\frac{\|b^*_j\|^2 + \mu^2_{j,j-1}\|b^*_{j-1}\|^2}{\|b^*_{j-1}\|^2}$$

Si on passe par l'étape 11 de l'algorithme, c'est parce que la condition de Lovàcz n'est pas vérifiée, c'est-à-dire :

$$\|b^*_j\|^2 < \left(\frac{3}{4} - \mu^2_{j,j-1}\right)\|b^*_{j-1}\|^2$$

D'où :

$$\frac{\|b^*_j\|^2 + \mu^2_{j,j-1}\|b^*_{j-1}\|^2}{\|b^*_{j-1}\|^2} < \frac{3}{4}$$

On a donc $D'_{j-1} < \frac{3}{4}D_{j-1}$.

Ainsi chaque passage à l'étape 11 de l'algorithme multiplie le produit $D_1D_2\dots D_d$ par un facteur au plus égal à $\frac{3}{4}$. Il suffit donc de montrer que ce produit est minoré par une constante ne dépendant que du réseau L.
La proposition 3.2 va nous permettre de démontrer cela.

Proposition :
Si L est un réseau de dimension d, la constante d'Hermite :

$$\gamma(L) = \frac{\min(L)^2}{\det(L)^{\frac{2}{d}}}$$

est majorée par une constante ne dépendant que de d.

Preuve :
Cette constante d'Hermite mesure la densité Δ de l'empilement de sphères associé au réseau L. Une sphère associée au réseau est centrée en un point du réseau comme on peut le voir sur le schéma 3.2.0.1.

La réunion de ces sphères est :

$$\mathcal{E} = \bigcup_{x\in L} B(x, R)$$

Le rayon maximum pour que ces sphères soient d'intérieurs disjoints est :

$$R = \frac{1}{2}\min(L)$$

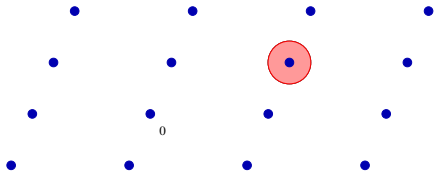

FIG. 3.2.0.1 – Schéma d'une sphère dans un réseau

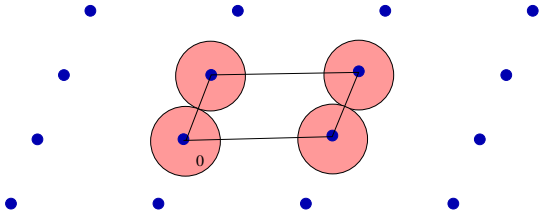

FIG. 3.2.0.2 – Sphères avec un rayon maximum afin d'être d'intérieurs disjoints

comme on peut le voir sur le schéma 3.2.0.2.

Soit $\mathcal{P}$ le parallélotope fondamental construit sur une base $(b_1, \ldots, b_d)$:

$$\mathcal{P} = \{x_1 b_1 + \ldots + x_d b_d \mid 0 \leqslant x_i \leqslant 1\}$$

Le volume de P est égal à $\det(L)$. Le volume de $\mathcal{P} \cap \mathcal{E}$ est égal au volume d'une sphère de rayon R (schéma 3.2.0.3), c'est-à-dire $r^d \pi_d$ où π_d est le volume de la sphère de rayon 1 et de dimension d.

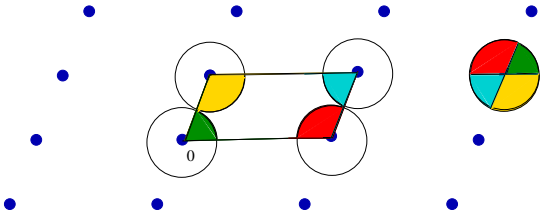

FIG. 3.2.0.3 – Illustration de l'égalité $vol(\mathcal{P} \cap \mathcal{E}) = vol(B(x, r))$

On a :

$$\Delta = \frac{vol(\mathcal{P} \cap \mathcal{E})}{vol(\mathcal{P})} \leqslant 1$$

D'où :

$$\frac{r^d \pi_d}{\det(L)} \leqslant 1$$

Ce qui conduit à :

$$\left(\frac{\min L}{(\det L)^{\frac{1}{d}}}\right)^d \frac{\pi_d}{2^d} \leqslant 1$$

On a donc montré que :

$$\gamma_j = \sup_{L \subset \mathbb{R}^j} \gamma(L)$$

est fini. On a donc pour les réseaux L_j :

$$D_j \geqslant \left(\frac{\min(L_j)^2}{\gamma_j}\right)^j \geqslant \left(\frac{\min(L)^2}{\gamma_j}\right)^j$$

et ainsi le produit $D_1 \dots D_d$ est bien minoré par une constante positive ne dépendant que du réseau L.

Les bases de l'algorithme LLL étant posées, nous allons maintenant voir la sécurité de NTRU et les attaques réalisables sur ce système.

Chapitre 4

Sécurité de NTRU

Comme vu précédemment, la sécurité de NTRU repose sur les problèmes CVP et SVP dans un réseau. Nous allons donc tout d'abord voir les attaques basées sur la résolution de ces problèmes.

4.1 Attaque sur les réseaux

La première attaque sur le réseau NTRU est due à Coppersmith et Shamir [9] qui ont proposé en 1997 une attaque contre la clé publique pour les petites valeurs du paramètre N.
Dans NTRUEncrypt, on a :

$$h = p * f_q * g \mod q \Rightarrow f * h * p^{-1} = g \mod q$$

Notons $h' = h * p^{-1} \mod q$.
Donc il existe un polynôme $u \in \mathcal{R}$ tel que :

$$f * h' - q * u = g$$

On considère l'ensemble :

$$\mathcal{A} = \{(f, g) \in \mathcal{R}^2 \mid \exists\ u \in \mathcal{R},\ f * h' - q * u = g\}$$

On voit clairement que $\mathcal{A}$ est un réseau, et sous forme matricielle on a donc :

$$(f, -u) * \begin{bmatrix} 1 & h' \\ 0 & q \end{bmatrix} = (f, g)$$

Dans la clé publique h, les polynômes f et g ont des petits cœfficients et $(f, g) \in \mathcal{A}$. Alors en appliquant l'algorithme LLL au réseau $\mathcal{A}$, on obtient une base réduite dans laquelle les premiers vecteurs sont assez courts, et on peut ainsi déterminer f et g.

En 1999, Alexander May a introduit [10] d'autres réseaux se basant sur les propriétés des clés privées, afin d'attaquer NTRUEncrypt avec des paramètres plus élevés.

Dans le cas de NTRUSign, la situation est plus difficile pour un attaquant car les petits vecteurs cibles sont beaucoup plus proches de l'heuristique de Gauss (partie 1.2) et donc plus difficiles à trouver dans la pratique par la technique de réduction des réseaux. Il est évident que n'importe quelle base réduite peut être utilisée pour signer, mais trouver une telle base est un problème très difficile pour N grand.

4.2 Attaques exhaustives

Dans NTRUEncrypt, les clés secrètes f et g sont des polynômes avec peu de cœfficients et qui sont tous égaux à ± 1. Or, $h = p * g * f_q \mod q$, d'où :

$$f * h = p * g \mod q \quad et \quad p * h^{-1} * g = f \mod q$$

Une première attaque exhaustive peut consister à choisir un $f \in D_f$, à calculer $f * h \mod q$ et à tester si l'on trouve un polynôme de la forme $p * g \mod q$ avec $g \in D_g$. Le nombre de possibilités pour choisir f est :

$$|D_f| = \left(\begin{array}{c} N \\ d_f \end{array} \right)$$

Par exemple, avec $N = 251$ et $d_f = 72$, on a :

$$|D_f| = 1,187 \times 10^{64}$$

Une autre recherche exhaustive consiste à choisir des valeurs de $g \in D_g$, à calculer $p * h^{-1} * g \mod q$ et à tester si le polynôme obtenu est dans D_f. Le nombre de possibilités pour g est alors de :

$$|D_g| = \left(\begin{array}{c} N \\ d_g \end{array} \right)$$

Enfin, une troisième attaque exhaustive concerne le message chiffré e. On a :

$$e = r * h + m \mod q \Rightarrow e - r * h = m \mod q$$

Cette attaque consiste à choisir un polynôme $r \in D_r$, à calculer $e - r * h \mod q$ et à tester si le polynôme obtenu est dans D_m. Le nombre total de possibilités est :

$$|D_r| = \left(\begin{array}{c} N \\ d_r \end{array} \right)$$

Toutes ces attaques ne peuvent fonctionner que si les clés privées sont mal choisies, ce qui n'est pas le cas en pratique car elles sont choisies de façon aléatoire.

4.3 Sécurité contre la contrefaçon des signatures

On considère ici la difficulté de créer une signature spécifique sans connaître la clé privée. Considérons un usurpateur qui choisit un petit s avec l'espoir que $h * s - m \mod q$ aura aussi des petits cœfficients. En moyenne, ces cœfficients seront plus ou moins aléatoires modulo q, et en utilisant le théorème 2.2.4, la moyenne de la norme d'une fausse signature sera $q\sqrt{\frac{N}{12}}$. Comme la valeur asymptotique de q est $O(N)$, la signature contrefaite aura une norme de $O\left(N^{\frac{3}{2}}\right)$, ce qui est dans le même ordre de grandeur que la norme d'une bonne signature.
Pour NTRUSign sans perturbations et avec les paramètres $(N, q, d_f, \beta) = (127, 256, 31, 0.4)$, en pratique, les signatures générées ont une norme de 140. La moyenne des normes des fausses signatures peut être estimée à 330. Donc la sécurité de NTRUSign semble résider dans ces constantes.

Une mesure plus fine de la difficulté de forger une signature avec cette méthode se traduit par l'écart type entre la moyenne des normes de fausses signatures et la moyenne des normes des vraies signatures. En utilisant toujours le théorème 2.2.4 on a :

$$\frac{\mu(\|Faux\|^2) - \mu(\|Vraie\|^2)}{\sigma(\|Faux\|^2)} \approx \sqrt{\frac{5N}{4}}\left(1 - \frac{Nd_f}{6q^2}\right)$$

Donc pour un rapport fixé de $\frac{Nd}{6q^2} < 1$, l'écart-type entre une fausse signature et une vraie croît comme $O(\sqrt{N})$.

4.4 Attaque sur les fuites d'information des messages signés

Nous considérons tout d'abord une signature sans perturbations. Dans ce cas, un attaquant qui possède une transcription de plusieurs signatures valides a une liste de paires de polynômes de la forme :

$$s = \epsilon f + \epsilon' g \quad et \quad h * s - m = \epsilon F + \epsilon' G$$

avec les cœfficients de ϵ et ϵ' dans $\left[-\frac{1}{2}, \frac{1}{2}\right]$. En d'autres termes, les signatures sont à l'intérieur d'un parallélotope dont les cotés sont les bons vecteurs de

la base. Le défi de l'attaquant est de découvrir un coté du parallélotope.

Comme les ϵ sont aléatoires, leur moyenne est de 0. Pour construire une attaque sur la moyenne des s et des $h * s - m$, l'attaquant doit trouver quelque chose qui ne donne pas une moyenne de 0. Pour avoir cela, il utilise le « réciproque » de s et $h * s - m$. Ce qu'on nomme « réciproque » d'un polynôme a est le polynôme :

$$\bar{a}(X) = a(X^{-1}) = a_0 + \sum_{i=1}^{N-1} a_{N-i} X^i$$

Nous posons :

$$\hat{a} = a * \bar{a}$$

et donc $\hat{a}$ est de la forme :

$$\hat{a} = \sum_{k=0}^{N-1} \left(\sum_{i=0}^{N-1} a_i a_{i+k} \right) X^k$$

En particulier, $\hat{a_0} = \sum_i a_i^2$. Cela signifie que si un attaquant fait la moyenne de toutes ses transcriptions de $\hat{s}$ et $h * \hat{s} - m$, les termes croisés disparaissent et l'attaquant retrouve :

$$\mu(\epsilon)(\hat{f} + \hat{g}) = \frac{N}{12}(\hat{f} + \hat{g})$$

pour s et de la même façon pour $h * s - m$, avec $\mu(x)$ la moyenne de x. Nous nous référerons au produit d'une mesure avec son « réciproque » par le terme « second moment » . Dans le cas de NTRUSign sans perturbation, retrouver le second moment d'une transcription révèle la matrice de Gram de la base privée. En pratique, il apparaît que des informations importantes sur la matrice de Gram sont divulguées après 10000 signatures. Nguyen et Regev [11] ont lancé une attaque sur NTRUSign sans perturbations qui permet de retrouver la matrice de Gram et ainsi la clé privée avec une transcription de 70000 signatures environ. Ce résultat a été amélioré à seulement 400 signatures [12] et donc l'utilisation de NTRUSign sans perturbations est fortement déconseillée. Évidemment, il a fallu faire quelque chose pour réduire la divulgation d'informations et c'est le rôle joué par les perturbations.

Dans le cas de B perturbations, l'espérance de $\hat{s}$ et $h * \hat{s} - m$ est :

$$E(\hat{s}) = \left(\frac{N}{12} \right) (\hat{f_0} + \hat{g_0} + \ldots + \hat{f_B} + \hat{g_B})$$

et

$$E(h * \hat{s} - m) = \left(\frac{N}{12} \right) (\hat{f_0} + \hat{g_0} + \ldots + \hat{f_B} + \hat{g_B})$$

Le second moment n'est plus une matrice de Gram mais la somme de $(B+1)$ matrices de Gram. De plus, les signatures ne se situent pas dans un parallélotope mais à l'intérieur de la somme de $(B+1)$ parallélotopes.
Cela complique la tache d'un attaquant. La meilleure technique connue pour $B=1$ est de calculer :

$$\mu(\hat{s}),\ \mu\left(\hat{s}^2\right),\ \mu\left(\hat{s}^3\right)$$

Comme, par exemple, $\mu(\hat{s})^2 \neq \mu(\hat{s}^2)$, l'attaquant peut utiliser l'algèbre linéaire pour éliminer f_1 et g_1 et retrouver la matrice de Gram, après quoi l'attaque de [11] peut être utilisée pour retrouver la clé privée.
On peut estimer τ le nombre de signatures nécessaires pour retrouver $\mu\left(\hat{s}^3\right)$. Pour cela, on considère un attaquant qui tente de retrouver $\mu\left(\hat{s}^3\right)$ en faisant la moyenne de τ signatures et en l'arrondissant vers l'entier le plus proche. Cela lui donnera une réponse raisonnablement correcte quand l'erreur dans un certain nombre de cœfficients (au moins la moitié) est plus petite que $\frac{1}{2}$. Pour calculer la probabilité qu'un cœfficient a une erreur inférieure à $\frac{1}{2}$, on écrit $\frac{12}{N}\hat{s}$ comme un terme principal plus une erreur, avec le terme principal qui tend vers $\hat{f}_0 + \hat{g_0} + \hat{f}_1 + \hat{g_1}$. L'erreur converge vers 0 avec environ la même vitesse que celle de la convergence du terme principal vers sa valeur attendue. Si la probabilité qu'un cœfficient donné est au-delà de $\frac{1}{2}$ de sa valeur prévue est plus petite que $\frac{1}{2N}$ alors on peut s'attendre qu'au moins la moitié des cœfficients soient arrondis à leur valeur correcte.

La vitesse de convergence d'une erreur et sa dépendance à τ peut être estimée par une application de la technique de Chernoff-Hœffding. On peut ainsi estimer que pour $B=1$, l'attaquant a besoin de plus de 2^{30} signatures.

Nous allons maintenant voir l'implémentation de NTRUEncrypt et NTRUSign.

Chapitre 5

Implémentation

Le choix du langage dans lequel j'ai écrit le programme qui implémente NTRU fut essentiellement déterminé par le fait que l'anneau de base de NTRU est un anneau quotient, et je me suis donc dirigée vers un langage orienté mathématique, et j'ai choisi le langage de script **GP** du système **PARI/GP**[13]. Mon premier choix s'était tout d'abord porté sur le langage **Magma**, mais celui-ci s'est révélé très peu pratique lorsque les éléments passaient d'un anneau à un autre. Les fonctions écrites dans ce langage sont à utiliser en ligne de commande depuis l'interpréteur gp.
Le programme implémente les algorithmes de NTRUSign, NTRUEncrypt et une attaque sur NTRUEncrypt avec un petit paramètre N.

5.1 Structure générale du programme

Le programme se compose d'un fichier principal NTRU.gp qui peut être considéré comme la méthode `main` du programme et de fichiers annexes contenant des fonctions. Ces fichiers sont :

- NTRUEncrypt.gp, qui, comme son nom l'indique, contient toutes les fonctions nécessaires à l'algorithme de chiffrement de NTRU.
- NTRUSign.gp, qui comporte toutes les fonctions nécessaires à l'algorithme de signature.
- attackEncrypt.gp qui implémente une attaque sur NTRUEncrypt avec de petits paramètres.
- inversion.gp, le fichier contenant les fonctions d'inversion sur $\mathcal{R}$ modulo p ou q.
- useful.gp, qui contient toutes les fonctions annexes nécessaires au bon fonctionnement du programme.

5.2 Représentation des données

Le cryptosystème NTRU manipule des polynômes dans l'anneau $\mathcal{R} = \frac{\mathbb{Z}(X)}{X^N - 1}$ par conséquent, tous les polynômes sont de degré inférieur à N. De plus, l'opération de multiplication dans cet anneau est la convolution mod $X^N - 1$. Voyons sur un exemple, comment cette convolution opère.

Exemple de convolution de deux polynômes

Posons $a = 4 + 5X + 7X^2$, $b = 5 + 3X + 2X^2$ et $N = 3$ et considérons les vecteurs de leurs cœfficients c'est-à-dire $[4, 5, 7]$ et $[5, 3, 2]$. Alors leur produit est :

$$\begin{aligned} a \times b &= 4 \times [5, 3, 2] \\ &+ 5 \times [2, 5, 3] \\ &+ 7 \times [3, 2, 5] \\ \\ &= [20, 12, 8] \\ &+ [10, 25, 15] \\ &+ [21, 14, 35] \\ \\ &= [20 + 10 + 21, 12 + 25 + 14, 8 + 15 + 35] \\ &= [51, 51, 58] \end{aligned}$$

Si de plus, les cœfficients de a sont seulement des 0 et des 1, on a avec $a = [1, 0, 1]$ et $b = [5, 3, 2]$:

$$\begin{aligned} a \times b &= 1 \times [5, 3, 2] \\ &+ 0 \times [2, 5, 3] \\ &+ 1 \times [3, 2, 5] \\ \\ &= [5, 3, 2] \\ &+ [3, 2, 5] \\ \\ &= [5 + 3, 3 + 2, 5 + 2] \\ &= [8, 5, 7] \end{aligned}$$

Chaque cœfficient du produit est seulement la somme de d_i nombres avec d_i le nombre de cœfficients égaux à 1 de a. La fonction `prod_conv(f,g)`

permet de calculer le produit de convolution lorsque ni f et ni g n'est de la forme binaire, et `prod_conv_opt(f,g)` effectue le même calcul lorsque f est binaire. Ces fonctions utilisent `turnvect(vect,t)` qui permet de faire « tourner » le vecteur de t rangs vers la droite et ainsi simuler l'opération $f \mapsto f(X) * X^t \mod X^N - 1$.

Il a donc été choisi de représenter les éléments de $\mathcal{R}$ sous la forme d'un vecteur de leurs cœfficients, de taille N, pour ces raisons de performances. Néanmoins pour certaines opérations, les éléments sont reconvertis en polynômes, puis l'opération est effectuée sur ces polynômes et enfin remis sous la forme vectorielle. Ainsi, les fonctions `vecttopol` et `poltovect` permettent de passer d'une forme à l'autre. De plus, pour récupérer le degré de f lorsqu'il est sous forme vectorielle, on utilise la fonction `degreevect`. Enfin, on a parfois besoin d'avoir un élément de $\mathcal{R}$ modulo un autre polynôme (de degré $< N$) que $X^N - 1$, la fonction `vectmodulopol` effectue cela.

5.3 Implémentation de NTRUEncrypt

Le fichier NTRUEncrypt.gp est composé de 6 fonctions. Avant de passer en revue ces fonctions, nous allons voir comment l'opération de convolution peut être optimisée grâce à la forme de f et de r.

5.3.1 Forme optimisée de f et r

Les clés privées de NTRUEncrypt doivent être de petits polynômes mais pas trop petits, car sinon on facilite les attaques exhaustives et les attaques par réduction de réseau. En effet, lorsque le vecteur le plus court est beaucoup plus court que la longueur attendue du plus court vecteur de Gauss $\sigma(L)$ alors LLL a plus de facilités à trouver le vecteur le plus court. On ne peut donc pas améliorer la vitesse de la convolution en diminuant le nombre de 1 de f. Mais il existe un moyen pour améliorer la performance de la convolution qui est de former f en combinant plusieurs petits polynômes, c'est-à-dire créer f à partir de f_1, f_2 tel que :

$$f = f_1 * f_2$$

Par exemple, si $N = 13$ et $f_1 = [1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]$, $f_2 = [1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]$ on a :

$$\begin{aligned} f &= [1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0] * [1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0] \\ &= [1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0] \end{aligned}$$

Et ainsi f a neuf 1 et quatre 0. Pour calculer $f * a$, on aurait donc besoin de 9 additions, mais en calculant d'abord $y = f_2 * a$ puis $f_1 * y$ on fait $3 + 3 = 6$

additions. Dans NTRUEncrypt, on va donc prendre

$$f = f_1 * f_2 + f_3$$

avec $d_{f_1} = d_{f_2} = d_{f_3}$ et $d_f = d_{f_1}^2 + d_{f_1}$. Et au lieu de prendre $r \in D_r$, on va choisir r tel que :

$$r = r_1 * r_2 + r_3$$

avec $d_{r_1} = d_{r_2} = d_{r_3}$ et $d_r = d_{r_1}^2 + d_{r_1}$. Ainsi la multiplication par f prendra $3d_{f_1}$ additions par cœfficients au lieu de $d_{f_1}^2 + d_{f_1}$, de même pour r.

De plus, pour réduire le nombre d'inversions de $f \mod q$ et $f \mod p$, on va choisir $f = 1 + pF$ et ainsi $f^{-1} \mod p = 1$. On va ainsi supprimer également l'opération $a * f^{-1} \mod p$ effectuée lors du déchiffrement.
En résumé, on aura :

$$f = 1 + p(f_1 * f_2 + f_3)$$

et

$$r = r_1 * r_2 + r_3$$

et toutes les opérations de multiplication par f ou r seront de la forme :

$$f * a = (1 + p(f_1 * f_2 + f_3)) * a = a + f_1 * (f_2 * (a * p)) + f_3 * (a * p)$$

et

$$r * a = r_1 * (r_2 * a) + r_3 * a$$

Dans le programme, le booléen `optimized` permet de choisir entre la version non optimisée (avec $f \in D_f$) et la version optimisée (avec f de la forme précédemment citée). L'utilisateur a donc le choix entre les deux formes. Cependant, pour réaliser l'attaque basée sur les réseaux, c'est la version non optimisée qui est utilisée car une attaque sur la version optimisée n'est pas basée sur le même problème[1] .

5.3.2 Choix des paramètres

La première fonction a pour signature :
`parameters_encrypt(level,p,optimized)`
Elle permet de choisir les paramètres en fonction du niveau de sécurité voulu (`level`) et de p. On peut voir la correspondance dans le tableau 5.3.2.1 où oui correspond à f est sous forme optimisée et non le contraire. Ces différents paramètres sont issus de la page 29 de [14]. De plus, $d_r = d_f$.

[1] Si f est de la forme $1 + pF$, la meilleure attaque sur la clé privée consiste à résoudre un CVP.

niveau de sécurité	N	q		df		dg	
		si $p = 2$	si $p = 3$	oui	non	oui	non
très faible	53	67	64	2	7	6	7
faible	127	127	128	4	23	20	23
standard	251	293	256	8	71	72	71
élevé	491	967	1024	15	241	240	241
très élevé	787	2027	2048	21	461	462	461

FIG. 5.3.2.1 – Différents niveaux de sécurité de NTRUEncrypt

5.3.3 Fonction de génération des clés

La deuxième fonction a pour signature :
`gen_key(df,dg,N,p,q,optimized,{verbose = 0})`
et implémente l'algorithme de génération des clés de NTRUEncrypt de la partie 2.1.2.1.
Les entiers `df`, `dg`, `N`, `p`, `q` sont naturellement les paramètres de NTRU.

Dans cette fonction, f (ou f_1, f_2, f_3) est choisie dans D_f. On a vu que, dans NTRUEncrypt, D_f pouvait être égal aux ensembles $\mathcal{T}_N(d_f)$ ou $\mathcal{B}_N(d_f)$ définis dans 2.1.1. Comme l'algorithme traite les deux cas $p = 2$ et $p = 3$, il a été choisi de prendre $D_f = \mathcal{B}_N(d_f)$, et ainsi avoir f sous forme binaire, invariante modulo 2 ou 3. g est également choisi dans $\mathcal{B}_N(d_g)$ pour les mêmes raisons. C'est la fonction `BNd(N,d)` qui permet de choisir aléatoirement un élément de cette forme.

Bien sûr, dans le cas de la forme optimisée, l'inverse de f modulo p n'est pas calculée, car $f^{-1} \mod p = 1$. On inverse donc seulement $f \mod q$ à l'aide des fonctions `invmodprime`, `invmodpower2` et `invmod2` du fichier inversion.gp qui effectuent les algorithmes 1, 2 et un algorithme spécifique pour l'inversion modulo 2 plus optimisé que l'algorithme général d'inversion modulo un nombre premier. Cependant si f_0 est une puissance de 2 dans le cas $p = 3$, f ne sera pas inversible modulo q qui est une puissance de 2. Dans ce cas, on choisit un autre élément f.

Dans la fonction qui recentre les éléments modulo q (partie 2.1.4), on constate que les éléments $f(1)$, $f_q(1)$ et $g(1)$ sont nécessaires. C'est pour cela, qu'à la fin de la fonction `gen_key`, on calcule ces éléments. Dans le cas où f n'est pas sous forme optimisée, $f(1) = d_f$ et $g(1) = d_g$. Mais dans

l'autre cas, $g(1)$ est toujours égal à d_g mais $f(1) \neq d_f$. Toutefois,

$$f(1) = \sum_{i=0}^{N-1} f_i \quad avec\ f = (f_0, \ldots, f_{N-1})$$

La fonction `vectsum` permet d'effectuer cette opération et ainsi de calculer $f(1)$ et $f_q(1)$.

La fonction `gen_key` retourne un vecteur contenant la clé publique h, la clé privée (f, f_p) dans le cas basique et (f_1, f_2, f_3) dans le cas optimisé. Elle retourne également $f(1)$ et $f_q(1)$ appelées `sumf` et `sumfq` dans le programme.

5.3.4 Fonction de chiffrement

La troisième fonction est la fonction de chiffrement, implémentant l'algorithme 2.1.2.2 dont la signature est :
`encryption(dr,N,m,h,q,p,optimized,{verbose = 0})`
Comme dans la génération des clés, les entiers `N`, `p`, `q` et `dr` sont les paramètres de NTRUEncrypt. L'entier `m` représente le message à chiffrer de taille N et h représente la clé publique du destinataire. Cette fonction n'implémente pas le padding NAEP car construire des fonctions de hachage est difficile, et cela ne fait pas partie des objectifs de ce projet.

Comme dans la fonction de génération des clés, r est créé à partir de la fonction `BNd(N,dr)` ou r est créé à partir de r_1, r_2 et r_3 tous trois créé avec cette dernière fonction. Puis le message chiffré est calculé comme dans 2.1.2.2 en utilisant la forme optimisée de r le cas échéant. Enfin la fonction qui recentre les cœfficients ayant besoin de $r(1)$, cette valeur est calculée et un vecteur contenant le message chiffré et $r(1)$ est retourné.

5.3.5 Fonction de déchiffrement

Les quatrième et cinquième fonctions sont :
`center(N,p,q,sumr,dg,sumf,sumfq,A,{verbose = 0})` et
`decryption(f1,f2,f3,f,fp_inv,e,N,q,p,sumr,dg,sumf,sumfq,optimized,{verbose = 0})`
La première implémente la fonction recentrant les éléments modulo q définie dans 2.1.4 et la deuxième exécute l'algorithme de déchiffrement de la partie 2.1.2.3. La fonction `decryption` prend en entrée la clé privée (f_1, f_2, f_3) ou (f, f_p) selon l'optimisation voulue, les paramètres de NTRU, $r(1)$ (noté `sumr`), $g(1) = dg$, $f(1)$ (noté `sumf`), $f_q(1)$ (noté `sumfq`) et bien sûr le texte chiffré `e`. Elle calcule $f * e$ comme dans l'algorithme de la partie 2.1.2.3 et appelle la fonction `center` avec le paramètre `A` $= f*e$. Cette dernière recentre les cœfficients modulo q et renvoie a. Puis selon l'optimisation choisie, la

fonction de déchiffrement calcule $f_p * a$ ou réduit simplement $a \mod p$. Le résultat de ce dernier calcul est renvoyé comme étant le déchiffrement du texte chiffré e.

5.3.6 Fonction combinant le chiffrement ou le déchiffrement de plusieurs messages

La dernière fonction du fichier NTRUEncrypt.gp est une fonction qui a la signature suivante :
`en_de_crypt_several(message,N,dr,h,q,p,f1,f2,f3,f,fp_inv,sumr,dg,sumf,sumfq,bool_en_de,{verbose = 0})`
Le booléen `bool_en_de` s'il est à vrai, permet de choisir de chiffrer plusieurs messages clairs et s'il est à faux, de déchiffrer plusieurs textes chiffrés. La notion de plusieurs messages est en fait un message de taille kN que l'on passe en argument à la fonction (via `message`). Puis ce message est séparé en k messages de taille N grâce à la fonction `divided_several_piece`. Puis en fonction du booléen `bool_en_de`, chaque message est chiffré ou déchiffré, puis re-concaténer en un message de taille kN. Cette fonction renvoie dans les deux cas, ce message, plus l'entier $r(1)$ dans le cas du chiffrement.

5.4 Implémentation de NTRUSign

Le fichier NTRUSign.gp contient 6 fonctions nécessaires à la signature d'un document. Notons également qu'il a été choisi $p = 3$.

5.4.1 Choix des paramètres

La première fonction est celle du choix des paramètres, elle a pour signature :
`parameters_sign(level)`
Les paramètres sont choisis selon le niveau de sécurité voulu (`level`). Le tableau 5.4.1.1 donne les paramètres en fonction du niveau de sécurité. Ces paramètres sont quelques uns des paramètres du tableau 4 de [15]. On a choisi $\beta = 0.4$, $d_f = d_g$ et les différents $\mathcal{N}$ ont été calculés en faisant la moyenne des normes des signatures valides.

5.4.2 Fonction de génération des clés

Les fonctions `find_FG(f,g,N,q,{verbose = 0})`, `reduce_FG(F,G,N,q,Rf,rhof,g,phi,{verbose = 0})` et `Sgen_key(N,q,df,dg,B,{verbose = 0})` sont les fonctions impliquées dans la génération des clés. La fonction `Sgen_key` prend en entrée les paramètres de NTRU (N, q, d_f et d_g), le nombre de perturbations de la signature B, ainsi que l'argument optionnel `verbose`. Cette fonction applique l'algorithme de la partie 2.2.2.1.

niveau de sécurité	N	q	d_f	$\mathcal{N}$
très faible	53	64	6	40
faible	67	128	12	70
standard	127	256	31	140
élevé	223	256	32	220
très élevé	313	512	50	380

FIG. 5.4.1.1 – Paramètres de NTRUSign en fonction du niveau de sécurité

Pour chaque i, avec i allant de B à 0, on va créer un réseau. Les éléments de $\mathcal{R}$, f et g sont choisis dans $\mathcal{T}_N(d)$ avec $d = d_f$ et $d = d_g$, car on a choisi $p = 3$. Le choix aléatoire de ces éléments est effectué par la fonction `TNd(N,d)`. Après le choix de ces éléments, la fonction `find_FG` est appelée avec f et g pour arguments. Cette fonction applique la procédure de la partie 2.2.3.1. Pour cela, elle calcule R_f et R_g les résultants de f et $X^N - 1$ et de g et $X^N - 1$ en utilisant la fonction `power_evaluation(f,N,i)` qui calcule $f(x^i)$ mod $\phi(X)$. Cette dernière utilise la relation suivante en notant $f(x^i) = f^i$:

$$f_0^i = f_0 \quad et \quad \forall i > 0,\ f_k^i = f_{\frac{k}{i} \mod N} \tag{5.1}$$

En effet, si f est sous forme polynômiale, chaque $x^j,\ j > 0$ qui compose f est transformé en $x^{ij \mod N}$ car $x^N = 1$. Et $x_0 = 1$ n'est pas modifié par l'opération $x \mapsto x^i$. Donc $\forall j > 0$, $f_j = f_{ji \mod N}^i$ et en posant $k = ij$ on obtient la relation 5.1. Cette relation est différente dans le programme car les indices de `gp` commencent à 1.

En même temps que R_f et R_g, sont calculés ρ_f et ρ_g. Si R_f et R_g ne sont pas premiers entre eux, la fonction retourne un vecteur d'erreur et de nouveaux f et g sont choisis aléatoirement. Sinon on calcule F et G comme indiqué dans la partie 2.2.3.1 et on appelle la fonction `reduce_FG`. Celle-ci va réduire F et G en suivant la méthode de la partie 2.2.3.2 et retourner F et G réduit. Enfin la fonction `Sgen_key` va retourner la clé publique h_0 et les clés privées (f_i, f_i', h_i) avec $0 \leqslant i \leqslant B$.

5.4.3 Fonction de signature

La fonction de signature est :
`signing(N,q,D,f,fprim,h,Nbound,beta,{verbose = 0})`
Elle met en pratique l'algorithme de la partie 2.2.2.2. `D` est le document à signer, `f,fprim,h` la clé privée (si $B = 1$, ce sont des vecteurs composés de 2 clés privées f_0 et f_1 dans le cas de f). `Nbound` est $\mathcal{N}$ la limite de la norme et `beta` est le facteur d'équilibrage β.

Le document est en théorie haché avec un ensemble de bits r, cependant, construire une fonction de hachage est fastidieux, donc le hachage n'est pas implémenté. Mais signer des bits avec NTRUSign ne donne pas des signatures « aléatoires » , donc une pseudo fonction de hachage est implémentée, `hash(vect,N,q)`, permettant de répartir les éléments de `vect` modulo q et ainsi avoir des signatures imprévisibles.
Une autre différence entre l'algorithme théorique et l'implémentation est le choix de l'ensemble de bits r. En théorie, on démarre à $r = 0$ et on augmente r si la signature trouvée n'est pas valide, dans le sens où :

$$\|(s, \beta(h * s - m_0))\| \geqslant \mathcal{N}$$

En pratique, il a été choisi de prendre r de façon aléatoire car ainsi la probabilité que la signature soit valable est plus élevée. Cependant, si après 256 signatures, la signature n'est pas valide, l'algorithme s'arrête et demande à l'utilisateur s'il veut retester d'autres r ou s'il veut quitter cette signature et revenir au menu de départ de NTRUSign.

Pour calculer la norme de $(s, \beta(h*s-m_0))$, on utilise la fonction `normdoublevect(f,g)` qui calcule :

$$\sqrt{\|f\|^2 + \|g\|^2}$$

en utilisant la fonction `normvect(f)` pour calculer $\|f\|$ et $\|g\|$ les normes centrées euclidiennes de f et g.

Cette fonction de signature renvoie le triplé $[D, intr, s]$ avec D le document signé, $intr$ l'entier correspondant à l'ensemble de bits r, et s la signature.

5.4.4 Fonction de vérification

La fonction de vérification a la signature suivante :
`verif(N,q,D,intr,s,h,Nbound,beta,{verbose = 0})`
avec `D,intr,s` le triplé du document signé et `h` la clé publique. Elle applique l'algorithme de la partie 2.2.2.3 et utilise les mêmes fonctions que la signature pour « hacher » et calculer la norme.

Elle renvoie `true` si la signature est valide et `false` sinon.

5.5 Implémentation d'une attaque sur le réseau NTRU

Le réseau NTRU considéré est le réseau de la clé publique $h = p * g * f^{-1}$ mod q, avec $N = 53$, $q = 64$, $p = 3$ et $df = dg = dr = 7$, il correspond à

un très faible niveau de sécurité. Le fichier `attackEncrypt.gp` contient les fonctions nécessaires à cette attaque.

La fonction `lattices(h,q,p)` construit la matrice des bases du réseaux c'est-à-dire :

$$\begin{pmatrix} 1 & h' \\ 0 & q \end{pmatrix}$$

avec $h' = h * p^{-1} \mod q$.

Ensuite, la fonction `inBND(g)` teste si un élément de $\mathcal{R}$ est sous forme binaire.

Enfin, la procédure principale de l'attaque nommée `attack`, qui effectue la procédure suivante :

- génère une clé publique h
- construit le réseau associé à cette clé publique
- réduit le réseau à l'aide de la fonction `qflll` de **PARI/GP**
- extrait les N premiers éléments de la première ligne de la matrice du réseau réduit comme correspondant à f ou à une des rotations de f (ou à l'opposé d'une rotation, si c'est le cas, on prend $-f$). Notons f_{ret} cet élément trouvé grâce à la réduction du réseau.
- teste si $g = f_{ret} * h * p^{-1} \mod q$ est sous forme binaire et si $g(1) = dg$.
- Si ce n'est pas le cas, on prend la rotation suivante de f_{ret}, sinon on teste si un message chiffré se déchiffre correctement avec cet élément f_{ret}.
- Si le message est égal au message chiffré puis déchiffré c'est que f_{ret} peut déchiffrer tous les messages

Remarque : On ne retrouve pas nécessairement f, mais un élément qui peut déchiffrer tous les messages chiffrés. Coppersmith et Shamir ont montré dans [9] qu'un élément satisfaisant certaines conditions de normes pouvait remplacer f dans le déchiffrement (il suffit que f_{ret} soit aussi court que f).

Cette attaque a été testée sur le niveau de sécurité supérieur, mais la première ligne de la matrice du réseau réduit est un vecteur qui contient seulement un 1 situé après le Nème élément. Si on suit la méthode précédente on trouve donc $f = 0$. Si on choisit une autre ligne, on peut trouver un f binaire, mais g n'est pas binaire ou $g = 0$. On peut donc supposer que $N = 127$ est trop grand pour cette attaque.

5.6 Fonctions du programme principal

Le programme principal nommé `NTRU({verbose = 0})` du fichier NTRU-.gp permet à l'utilisateur de choisir tous les paramètres nécessaires soit au chiffrement (et au déchiffrement), soit à la signature (et à la vérification) d'un message. De plus, une troisième option permet de voir l'attaque précédemment citée sur le réseau NTRU. L'argument optionnel `verbose` est choisi à l'appel de ce programme principal en appelant soit `NTRU()`, soit `NTRU(1)` pour le mode verbose.

La fonction `input()` de `gp` permet de lire une chaîne de caractères interprétée comme une expression GP à partir de l'entrée standard à savoir le clavier. Ainsi la fonction `recover_value_input(printing,vec_choice)` affiche une question à l'utilisateur (argument `printing`) et vérifie que le choix de l'utilisateur est bien un des choix possibles rentrés dans le vecteur `vec_choice`. De plus, la fonction `recover_message(N,q,bool_en_de)` permet de lire un vecteur qui, selon le booléen bool_en_de, sera interprété comme un message clair ou un message chiffré tout en vérifiant qu'il a les bonnes propriétés. De même pour la fonction `recover_document_sign(N,q,doc_-sign)` utilisée dans le cadre de NTRUSign. Enfin, deux fonctions ne sont utilisées que dans le mode verbose, il s'agit de `display_pol` et `display_vect` qui permettent l'affichage de polynômes, respectivement de vecteurs, sans toutefois les afficher dans leur ensemble lorsque ceux-ci contiennent de nombreux termes, seuls ceux de plus haut degré et de plus bas degré sont affichés dans le cas des polynômes, et les premiers et derniers éléments dans le cas des vecteurs.

Nous allons dans le prochain chapitre parler des résultats de cette implémentation de NTRU.

Chapitre 6

Résultats

Le tableau 6.0.0.1 donne les temps de calcul des différents algorithmes de NTRUEncrypt en fonction de p, de la forme optimisée (c'est ce à quoi correspondent les colonnes « oui » et « non ») et du niveau de sécurité.

niveau sécurité	p	génération		chiffrement		déchiffrement	
		oui	non	oui	non	oui	non
très faible	2	24ms	52ms	0ms	0ms	0ms	8ms
très faible	3	32ms	48ms	0ms	0ms	0ms	4ms
faible	2	177ms	232ms	0ms	4ms	4ms	24ms
faible	3	160ms	280ms	0ms	4ms	4ms	28ms
standard	2	677ms	881ms	8ms	20ms	12ms	108ms
standard	3	629ms	1128ms	8ms	24ms	8ms	108ms
élevé	2	2612ms	3376ms	28ms	148ms	32ms	473ms
élevé	3	3028ms	4989ms	28ms	148ms	28ms	473ms
très élevé	2	6612ms	8693ms	61ms	441ms	64ms	1276ms
très élevé	3	7677ms	12736ms	64ms	445ms	60ms	1268ms

FIG. 6.0.0.1 – Différents temps de calcul de NTRUEncrypt sur un ordinateur équipé d'un processeur Intel Core 2 Duo T5800 à 2×2,00 GHz

On constate tout d'abord que la génération des clés est un algorithme qui nécessite beaucoup plus de temps que les autres, mais cela n'est pas important car la génération de la clé n'est effectuée en théorie qu'une seule fois. On constate également que le choix de $p = 3$ n'est plus rapide (dans la forme optimisée) que pour les niveaux de sécurité faible et standard (du moins dans la génération des clés), alors qu'il est choisi en théorie pour des raisons de performance.

La différence de temps entre la forme optimisée et la forme non optimisée

est assez importante notamment dans le déchiffrement, où elle est jusqu'à 20 fois plus rapide. La génération de la clé s'effectue 130% plus rapidement en mode optimisé qu'en mode non optimisé et le chiffrement 7 fois plus rapidement. Le gain est donc important.

En ce qui concerne NTRUSign, les temps de calcul sont présentés dans le tableau 6.0.0.2 en fonction du niveau de sécurité et du nombre de perturbations, 0 ou 1.

niveau sécurité	perturbé	génération	signature	vérification
très faible	non	157ms	28ms	8ms
très faible	oui	292ms	33ms	8ms
faible	non	329ms	36ms	8ms
faible	oui	637ms	72ms	12ms
standard	non	2408ms	100ms	20ms
standard	oui	4949ms	193ms	28ms
élevé	non	15537ms	276ms	52ms
élevé	oui	29950ms	557ms	60ms
très élevé	non	57580ms	549ms	100ms
très élevé	oui	106743ms	1088ms	112ms

FIG. 6.0.0.2 – Différents temps de calcul de NTRUSign sur un ordinateur équipé d'un processeur Intel Core 2 Duo T5800 à 2×2,00 GHz

On constate que les algorithmes de génération de clés et de signatures sont environ deux fois plus long dans le cas perturbé, ce qui est normal car on génère un réseau de plus et on perturbe le point.

Comme dans NTRUEncrypt, la génération des clés est beaucoup plus longue que les autres algorithmes, mais elle n'est effectuée qu'une fois alors que la signature et la vérification sont supposées être utilisées plusieurs fois.

Enfin, l'attaque sur le réseau NTRU avec un très faible niveau de sécurité se réalise en environ 10 secondes, celles-ci étant principalement le temps de réduction du réseau.

Des améliorations sont possibles dans ce programme comme implémenter de vraies fonctions de hachage et ainsi implémenter le padding NAEP. De plus, p et q doivent être premiers entre eux sur $\mathcal{R}$ mais rien n'oblige p à être un entier, il peut aussi être un polynôme. Le tutoriel [4] explique comment s'exécute l'algorithme avec $p = 2 + X$.

Chapitre 7

Conclusion

NTRUEncrypt est un algorithme de chiffrement très rapide et est donc une vraie alternative à RSA et aux autres algorithmes à clé publique. En effet, le produit de convolution opérant sur deux polynômes a un coût en $O(n^2)$ opérations. De plus, les polynômes ont des petits cœfficients ce qui simplifie et accélère les calculs. Un autre avantage concernant le futur de la cryptographie à clé publique est que NTRU résiste encore contre les attaques des ordinateurs quantiques contrairement à RSA, El Gamal et ECC qui seront cassés. En effet, les problèmes sur lesquelles repose la sécurité de NTRUEncrypt n'ont pas pour l'instant de solutions basées sur la théorie quantique.

NTRU fournit également NTRUSign, un algorithme de signature qui, s'il n'est pas sans divulgation, est utilisable en pratique car avec une perturbation, l'information divulguée est très faible et il faut un très grand nombre de signatures pour pouvoir exploiter cette information. NTRU est considéré sûr par le standard IEEE P1363.1 [16].

En ce qui concerne le programme de ce projet, il effectue bien les algorithmes de chiffrement et de signature NTRU ainsi qu'une attaque sur une implémentation de NTRUEncrypt avec un petit paramètre, sans toutefois implémenter la fonction de hachage nécessaire à NTRUSign, ni le padding NAEP, qui permet d'éviter les attaques à chiffrés choisis. Cependant, une implémentation matérielle permettrait d'améliorer sensiblement les temps de calcul.

Dans le futur, si un ordinateur quantique est construit, RSA, El Gamal et ECC seront cassés et NTRU deviendrait probablement l'algorithme standard de cryptographie à clé publique.

Bibliographie

[1] Jeff Hoffstein, Nick Howgrave-Graham, Jill Pipher, William Whyte *Practical lattice-based cryptography : NTRUEncrypt and NTRUSign* (http://www.securityinnovation.com/cryptolab/pdf/lll25.pdf)

[2] Christine Bachoc *RÉSEAUX ET CRYPTOGRAPHIE*, Master CSI, UE Cryptanalyse (http://www.math.u-bordeaux1.fr/~bachoc/Enseignements/Cryptanalyse/LLL.pdf)

[3] Abderrahmane Nitaj *NTRU et ses variantes, sécurité et applications* (http://www.math.unicaen.fr/~nitaj/Ntru.pdf)

[4] *The NTRU Public Key Cryptosystem - A Tutorial* (http://securityinnovation.com/pdf/Ntru_Public_Key_Cryptosystem_Tutorial.pdf)

[5] *Security Innovation* (http://securityinnovation.com/)

[6] Joseph Silvermann *Almost Inverses and Fast NTRU Key Creation*, NTRU Cryptosystems Technical Report n°14, 1999 (http://securityinnovation.com/cryptolab/pdf/NTRUTech014.pdf)

[7] Nick Howgrave-Graham, Joseph Silvermann, Ari Singer, William Whyte *NAEP : Provable security in the presence of decryption failures* (http://securityinnovation.com/cryptolab/pdf/NAEP.pdf)

[8] Jeffrey Hoffstein, Nick Howgrave-Graham, Jill Pipher, Joseph Silvermann, William Whyte *NTRUSign : Digital Signatures Using the NTRU Lattice* (http://securityinnovation.com/cryptolab/pdf/NTRUSign_RSA.pdf)

[9] Don Coppersmith, Adi Shamir *Lattice Attacks on NTRU*, Advances in Cryptology - EUROCRYPT '97, Lecture Notes in Computer Science, 1997, Volume 1233/1997, p.52-61

[10] Alexander May *Cryptanalysis of NTRU* (http://www.cits.rub.de/imperia/md/content/may/paper/cryptanalysisofntru.ps)

[11] Phong Nguyen and Oded Regev *Learning a Parallelepiped : Cryptanalysis of GGH and NTRU Signatures*, EUROCRYPT '06 (http://www.cs.tau.ac.il/~odedr/papers/gghattack.pdf)

[12] Phong Nguyen *A Note on the Security of NTRUSign*, 2006 (http://eprint.iacr.org/2006/387.pdf)

[13] *PARI/GP* (http://pari.math.u-bordeaux.fr/)

[14] William Whyte *Choosing Parameter Set for NTRUEncrypt with NAEP and SVES-3*, RSA Conference 2005 (http://securityinnovation.com/cryptolab/pdf/CRYP-303A-Whyte.pdf)

[15] Jeff Hoffstein, Nicholas Howgrave-Graham, Jill Pipher, Joseph Silvermann, William Whyte *Performance Improvements and a Baseline Parameter Generation Algorithm for NTRUSign*, In Proc. of Workshop on Mathematical Problems and Techniques in Cryptology, 2005, p.99-126 (http://eprint.iacr.org/2005/274.pdf)

[16] *IEEE P1363.1 Public-Key Cryptographic Techniques Based on Hard Problems over Lattices*, Juin 2003 (http://grouper.ieee.org/groups/1363/lattPK/index.html)