L'algorithme LLL et ses applications en cryptographie

Abderrahmane Nitaj
Laboratoire de Mathématiques Nicolas Oresme
Université de Caen, France
nitaj@math.unicaen.fr
http://www.math.unicaen.fr/~nitaj

20 janvier 2014

Résumé

Depuis son invention en 1982 par Lenstra, Lenstra et Lovász, l'algorithme LLL a joué un grand rôle dans plusieurs domaines, en particulier en théorie algorithmique des nombres et en cryptographie. Dans ce mini cours, on explicite quelques unes de ses applications en cryptographie. On montre ainsi comment utiliser l'algorithme LLL pour casser des instances particulières du cryptosystème de Diffie et Hellman basé sur le problème du sac à dos et du cryptosystème NTRU basé sur l'anneau des polynômes modulaires. Ces applications sont illustrées par l'utilisation des systèmes de calcul Magma et Maple.

1 Introduction

La réduction des réseaux consite à transformer un réseau quelconque en un réseau dans lequel les vecteurs sont assez courts et presque orthogonaux. C'est un problème classique en mathématiques qui remonte à Lagrange et Gauss pour les réseaux de rang 2. En 1982, Lenstra, Lenstra et Lovász [9] ont inventé un algorithme très efficace pour la réduction des réseaux ayant des dimensions supérieures. Cet algorithme est connu sous le nom LLL et a été utlisé pour résoudre un grand nombre de problèmes. Il a ainsi été utilisé pour factoriser les polynômes, chercher des vecteurs courts dans un réseau, résoudre des équations diophantiennes, cryptanalyser des instantces particulières de plusieurs cryptosystèmes, parmi lesquels on peut citer RSA [13],

Merkle-Hellman [1] et NTRU [6, 11, 12]. Pour un apperçu de quelques unes des applications de LLL, on peut consulter par exemple [15] et [4]. Pour la théorie de la réduction des réseaux et de l'algorithme LLL, on peut consulter [7] ou encore [2].

Dans ce mini-court, on s'interesse à l'application de l'algorithme LLL en cryptographie et plus particulièrement à la cryptanalyse du cryptosystème proposé en 1982 par Merkle et Hellman [1] et à la cryptanalyse de quelques instances simples du cryptosystème NTRU proposé en 1998 par Hoffstein, Pipher et Silverman. Chacune de ces applications est illustrée par des exemples et des programmes implémentés à l'aide des systèmes de calcul Magma et Maple.

On donne maintenant une présentation rapide des réseaux et de l'algorithme LLL.

Soit $B = \{b_1, ..., b_n\}$ une famille de vecteurs linéairement indépendants de $\mathbb{R}^m$. Le réseau engendré par la base B est l'ensemble

$$\mathcal{L}(B) = \left\{ \sum_{i=1}^{n} x_i b_i \quad | \quad x_i \in \mathbb{Z} \right\}.$$

On dit alors que B est une base de $\mathcal{L}$ et que sa dimension est n. Si B' est une autre base de $\mathcal{L}$, alors il existe une matrice carrée $n \times n$ T à coefficients dans $\mathbb{Z}$ telle que $\det(T) = \pm 1$ et $B' = BT$. Si $x = (x_1 \cdots, x_n)$ et $y = (y_1 \cdots, y_n)$ sont deux vecteurs de $\mathbb{R}^n$, le produit scalaire de x et y est défini par

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

Définition 1.1 (**Déterminant**).
Soit $B = \{b_1, ..., b_n\}$ une base d'un réseau $\mathcal{L}$ et $[\langle b_i, b_j \rangle_{1 \leq i,j \leq n}]$ la matrice carrée formé par les produits scalaires $\langle b_i, b_j \rangle_{1 \leq i,j \leq n}$. Le déterminant de $\mathcal{L}$ est

$$\det(\mathcal{L}) = \left(\det\left[\langle b_i, b_j \rangle_{1 \leq i,j \leq n}\right]\right)^{\frac{1}{2}}.$$

Si B' est une autre base de $\mathcal{L}$, alors il existe une matrice carrée $n \times n$ T à coefficients dans $\mathbb{Z}$ telle que $\det(T) = \pm 1$ et $B' = BT$. Ceci montre que le déterminant d'un réseau est indépendant de sa base. La longueur d'un vecteur x est sa norme

$$\|x\| = (\langle x, x \rangle)^{\frac{1}{2}} = \left(\sum_{i=1}^{n} x_i^2 \right)^{\frac{1}{2}}.$$

Le problème de la réduction d'un réseau $\mathcal{L}$ consiste à chercher une base B' qui admet de bonnes propriétés : les vecteurs sont presques orthogonaux entre eux et assez courts, c'est-à-dire ayant des normes assez petites. Il existe en fait plusieurs sortes de réduction du même réseau.

Définition 1.2 (**La réduction de Minkowski**).
Une base $B = \{b_1, ..., b_n\}$ d'un réseau $\mathcal{L}$ est dite réduite au sens de Minkowski si les deux conditions suivantes sont satisfaites :

- *Le vecteur b_1 est un plus court vecteur de $\mathcal{L}$.*
- *Pour tout i avec $1 \leq i \leq n$:*
  - *la famille $\{b_1, ..., b_{i-1}, b_i\}$ est libre,*
  - *le vecteur b_i a une norme minimal dans l'espace engendré par $\{b_1, ..., b_i\}$,*
  - *la famille $\{b_1, ..., b_i\}$ peut être prolongée en une base de $\mathcal{L}$.*

Les bases réduites au sens de Minkowski ont de très bonnes propriétés mais leur utilisation en pratique est très difficile car il n'existe pas d'algorithme polynômial connu pour déterminer une telle base pour un réseau $\mathcal{L}$ donné.

En 1982, Lenstra, Lenstra et Lovász ont inventé l'algorithme polynômial LLL pour réduire un réseau en déterminant une base dont les vecteurs ont de bonnes propriétés. Cette base est alors dite réduite au sens de Lenstra, Lenstra et Lovász. La définition et les principales propriétés d'une base réduite au sens de Lenstra, Lenstra et Lovász sont liées à la notion de base orthogonale au sens de Gram-Schmidt :

Théorème 1.3 (**Méthode d'orthogonalisation de Gram-Schmidt**).
Soit V un sous-espace vectoriel de dimension n et $(b_1 \cdots, b_n)$ une base de V. On considère la famille de vecteurs $(b_1^* \cdots, b_n^*)$ définie par

$$b_1^* = b_1, \quad b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^*,$$

avec pour $j < i$

$$\mu_{i,j} = \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle}.$$

Alors $(b_1^* \cdots, b_n^*)$ est une base orthogonale de l'espace de V.

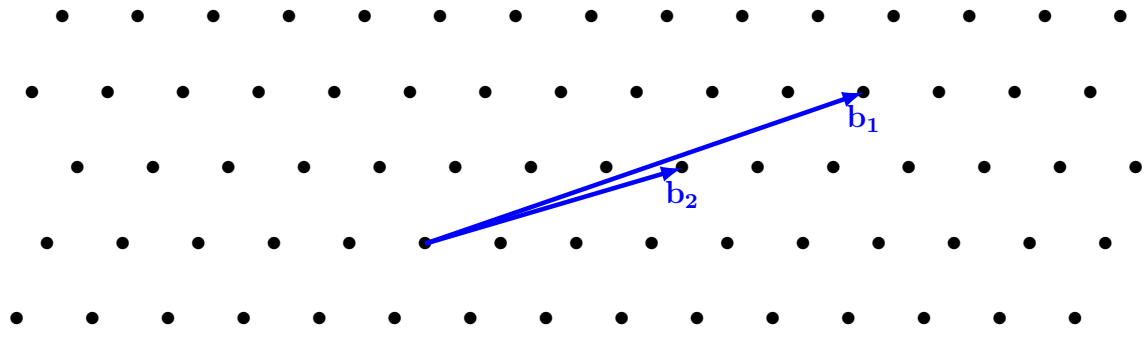

FIGURE 1 – Un réseau engendré par la *mauvaise* base (b_1, b_2)

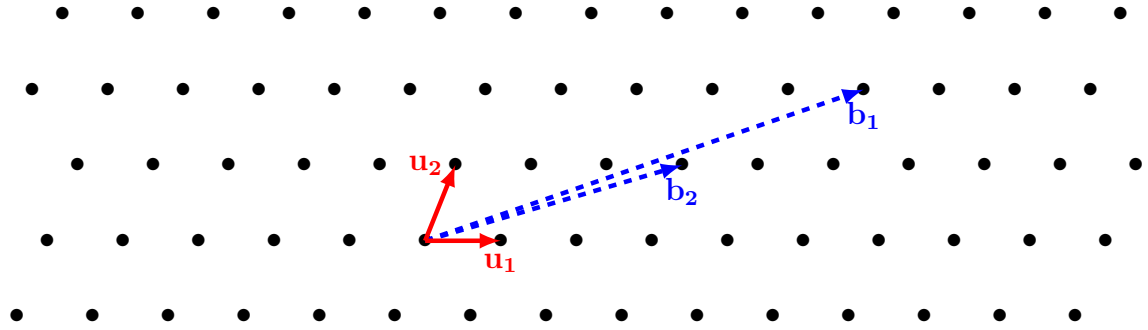

FIGURE 2 – Le même réseau engendré par une *bonne* base (u_1, u_2)

.

Définition 1.4 (La réduction de Lenstra, Lenstra et Lovász).
Une base $(b_1 \cdots, b_n)$ est LLL-réduite si, la base $(b_1^* \cdots, b_n^*)$ produite par la la méthode d'orthogonalisation de Gram-Schmidt vérifie

$$|\mu_{i,j}| \leq \frac{1}{2}, \quad pour \quad 1 \leq j < i \leq n, \tag{1}$$

$$\frac{3}{4}\|b_{i-1}^*\|^2 \leq \|b_i^* + \mu_{i,i-1}b_{i-1}^*\|^2, \quad pour \quad 1 < i \leq n. \tag{2}$$

Les propriétés des bases réduites au sens de LLL sont nombreuses. Du point de vue des longueurs des vecteurs, on peut citer les suivantes.

Théorème 1.5. *Soit $(b_1 \cdots, b_n)$ une base LLL-réduite et $(b_1^*, \cdots, b_n^*)$ la base orthogonale associée par la méthode de Gram-Schmidt. Alors*

1. $\|b_j^*\|^2 \leq 2^{i-j}\|b_i^*\|^2$ *pour* $1 \leq j \leq i \leq n$.
2. $\det(L) \leq \prod_{i=1}^{n} \|b_i\| \leq 2^{\frac{n(n-1)}{4}} \det(L)$.
3. $\|b_j\| \leq 2^{\frac{i-1}{2}}\|b_i^*\|$ *pour* $1 \leq j \leq i \leq n$.
4. $\|b_1\| \leq 2^{\frac{n-1}{4}}(\det(L))^{\frac{1}{n}}$.
5. *Pour tout vecteur non nul* $v \in L$, $\|b_1\| \leq 2^{\frac{n-1}{2}}\|v\|$.

2 Application au problème du sac à dos

2.1 Introduction au problème du sac à dos

En 1982, Merkle et Hellman ont proposé un protocole cryptographique rapide et facile à mettre en oeuvre. Ce protocole est basé sur le problème du sac à dos.

Problème du sac à dos

Soit $n \geq 2$ un nombre entier. Soient a_i, $1 \leq i \leq n$, et S des nombres entiers fixés. Le problème du sac à dos consiste en la recherche d'une solution $(x_1, \cdots, x_k) \in \{0,1\}^n$ de l'équation :

$$a_1x_1 + \cdots + a_nx_n = S.$$

Le problème du sac à dos est un problème NP-complet, mais certaines instances sont facile à résoudre. Une instance facile à résoudre est basée sur la notion de suites supercroissantes.

Définition 2.1. *Une suite supercroissante est une suite $(a_1, \cdots, a_n)$ telle que pour tout k avec $2 \leq k \leq n$, on a $b_k > \sum_{i=1}^{k-1} b_i$.*

Un exemple simple est la suite $(a_1, \cdots, a_n)$ définie pour tout $i \geq 0$ par $a_i = b^i$ où $b \geq 2$ est un entier. En effet

$$\sum_{i=1}^{k-1} a_i = \sum_{i=1}^{k-1} b^i = \frac{b^k - 1}{b-1} < b^k = a_k.$$

Dans ce même exemple, avec $b = 2$, la représentation d'un entier non nul S sous la forme $a_1x_1 + \cdots + a_nx_n = S$ avec $x_i \in \{0,1\}$ est tout simplement la décomposition binaire de $S - 1$. Ce principe s'étend en fait à toutes les suites supercroissantes.

Proposition 2.2. *Soit $(a_1, \cdots, a_n)$ une suite supercroissante et S un entier tel que $S = a_1x_1 + \cdots + a_nx_n$ avec $x_i \in \{0,1\}$. Alors l'algorithme suivant permet de déterminer la solution $(x_1, \cdots, x_n) \in \{0,1\}^n$.*

Dans Maple 12, le programme suivant permet déterminer la solution de l'équation $S = a_1x_1 + \cdots + a_nx_n$ avec $x_i \in \{0,1\}$ où $(a_1, \cdots, a_n)$ est une suite supercroissante. Dans ce programme, on pose $a = \{a_1, \cdots, a_n\}$.

Algorithm 1 : Résolutions du problème sac à dos pour une suite supercroissante

Entrée: La suite supercroissante $(a_1, \cdots, a_n)$ et un entier $S = a_1x_1 + \cdots + a_nx_n$ avec $x_i \in \{0, 1\}$.

Sortie: La solution $(x_1, \cdots, x_n) \in \{0, 1\}^n$.

1: $i = n$.
2: **while** $i \geq 1$ **do**
3: **if** $S \geq b_i$ **then**
4: $x_i = 1$,
5: $S = S - a_i$,
6: **else**
7: $x_i = 0$.
8: **end if**
9: $i = i - 1$.
10: **end while**
11: Sortir $(x_1, \cdots, x_n)$.

Programme (Maple 12)	Commentaires
`sol:=proc(a,S)`	`<--- procédure sol`
`  n:=nops(a);`	`<--- n:=nombre d'éléments de a`
`  Z:=S;`	`<--- Changement de variable;`
`  i:=n;`	`<--- initialisation de i`
`  while i > 0 do`	`<--- Début de la boucle`
`  if Z >= a[i] then`	`<--- Test`
`  x[i] := 1;`	`<--- x[i] = 1`
`  Z := Z-a[i];`	`<--- S := S-a[i]`
`  else x[i] := 0;`	`<--- Sinon x[i] := 0`
`  end if;`	`<--- Fin du test`
`  i := i-1;`	`<--- Décrémentation de i`
`  end do;`	`<--- Fin de la boucle`
`  for i from 1 to n do`	`<--- Sortie de la solution`
`  L := [seq(x[i], i = 1 .. n)];`	
`  end do;`	
`end proc;`	

On peut tester ce programme avec l'exemple suivant

$$a = \{4, 6, 11, 25, 50, 110\}, \quad S = a[1] + a[2] + a[3] + a[6] = 131.$$

Programme (Maple 12)	Commentaires
a:={4, 6, 11, 25, 50, 110};	<--- Liste a
S:=131;	<--- Expression de S
sol(a,S);	<--- procédure sol(a,S);

On obtient alors la solution $(1, 1, 1, 0, 0, 1)$.

MAGMA - Programme 2.3.

Dans Magma, le programme suivant permet déterminer la solution de l'équation $S = a_1x_1 + \cdots + a_nx_n$ avec $x_i \in \{0, 1\}$ où $(a_1, \cdots, a_n)$ est une suite supercroissante. Dans ce programme, on pose $a = \{a_1, \cdots, a_n\}$.

Programme (Magma)	Commentaires
sol := function(a,S)	<--- Fonction sol
Z:=S;	<--- Changement de variable;
n:=#a;	<--- taille de la liste a
x:=[];	<--- initialisation de x
i:=n;	<--- initialisation de i
while 1 le i do	<--- Début de la boucle
if a[i] le Z then	<--- Test
x:=[1] cat x;	<--- Concaténation de 1 à x
Z := Z-a[i];	<--- S := S-a[i]
else x:=[0] cat x ;	<--- Concaténation de 0 à x
end if;	<--- Fin de if
i := i-1;	<--- Décrémentation de i
end while;	<--- Fin de la boucle
return x;	<--- Liste x
end function;	<--- Fin de la fonction

On peut aussi tester ce programme avec le même exemple que dans le programme Maple :

$$a = \{4, 6, 11, 25, 50, 110\}, \quad S = a[1] + a[2] + a[3] + a[6].$$

Programme (Magma)	Commentaires
a:=[4, 6, 11, 25, 50, 110];	<--- Liste a
S:=131;	<--- Expression de S
sol(a,S);	<--- procédure sol(a,S);

On obtient aussi la solution $(1, 1, 1, 0, 0, 1)$.

2.2 Le cryptosystème de Merkle et Hellman

Un des premiers cryptosystèmes à clé publique fût proposé en 1982 par Merkle et Hellman. La sécurité de ce cryptosystème est basée sur la difficulté de résoudre le problème du sac à dos dans les cas difficiles.

Pour utliser le cryptosystème Merkle et Hellman par deux entités **A** et **B**, ils doivent procéder comme suit, où on suppose que **B** veut envoyer un message M à **B**. Pour celà, **A** doit préparer ses clés secrètes et publiques.

Génération des clés par A :

1. Choisir une suite super croissante $(a_1, \cdots, a_n)$.
2. Choisir un entier N tel que $N > \sum_{i=1}^{n} a_i$.
3. Choisir un entier d tel que $\gcd(d, N) = 1$.
4. Calculer $e_i = da_i \pmod N$ pour $1 \leq i \leq n$.
5. Publier la clé publique $(e_1, \cdots, e_n)$.
6. Conserver la clé secrète $(N, d, (a_1, \cdots, a_n))$.

Maintenant, **B** peut coder et envoyer son message M.

Chiffrement par B :

1. Transformer le message M en une suite binaire $X = (x_1, \cdots, x_n) \in \{0, 1\}^n$.
2. Calculer $S = \sum_{i=1}^{n} x_i e_i$.
3. Envoyer le message S à **A**.

En recevant le message chiffré S, **A** peut alors le déchiffrer.

Déchiffrement par A :

1. Calculer $S' \equiv Sd^{-1} \equiv \pmod N$.
2. Résoudre l'équation $S' = \sum_{i=1}^{n} x_i a_i$.
3. Calculer $M = \sum_{i=1}^{n} x_i 2^i$.

La preuve de la justesse de ce cryptosystème est évidente. Pour retrouver le message clair M, **A** connait tous les paramètres sauf M et la suite $(x_1, \cdots, x_n)$. On a alors

$$S' \equiv Sd^{-1} \equiv d^{-1} \sum_{i=1}^{n} x_i e_i \equiv d^{-1} \sum_{i=1}^{n} x_i d a_i \equiv \sum_{i=1}^{n} x_i a_i \pmod N.$$

Puisque la suite $(a_1, \cdots, a_n)$ est supercroissante, alors il est facile de déterminer la solution $(x_1, \cdots, x_n)$ et donc de calculer la somme $M = \sum_{i=1}^{n} x_i 2^i$, ce qui donne le message clair.

MAPLE - Programme 2.4.
Avec Maple 12, le programme suivant illustre une utilisation du cryptosystème de Merkle et Hellman. Dans ce programme, on pose $a = \{a_1, \cdots, a_n\}$, et on suppose que la procédure **sol** 2.1 a été exécutée.

Programme (Maple 12)	Commentaires
`#Génération des clés par A:`	`<--- Génération des clés par A:`
`a := [4, 6, 11, 25, 50, 110];`	`<--- Choix de la suite a`
`n := nops(a);`	`<--- n := #a`
`T := 0:`	`<--- Initialisation de T`
`for i from 1 to n do`	`<--- T=somme des termes de a`
`  T := T+a[i]`	
`end do:`	
`N := (rand(T .. 2*T))();`	`<--- Choix d'un nombre`
`d:=N:`	`  aléatoire dans [T,2T]`
`while gcd(d,N)>1 do`	`<--- Création d'un nombre d`
`  d:=rand(2..N-1)():`	
`end do:`	
`e:=[seq(mod(d*a[i],N),i=1..n)];`	`<--- Création de la suite e`
`# Chiffrement par B`	`<--- Chiffrement par B`
`M:=61;`	`<--- Choix du message M`
`a2:=[seq(2^i, i = 0 .. n-1)];`	`<--- Suite a2=(1,2,...2^{n-1}`
`X :=sol(a2, M);`	`<--- Décomposition de M dans a2`
`S := 0:`	`<--- Initialisation de S`
`for i from 1 to n do`	`<--- S=somme X*e`
`  S := S+X[i]*e[i];`	
`end do:`	
`# Déchiffrement par A`	`<--- Déchiffrement par A`
`S2:=(S*modp(1/d, N)) mod N;`	`<--- Calcul de S2=S/d mod N`
`X2 := sol(a, S2);`	`<--- X2=suite binaire de S2 dans a`
`M2 := 0:`	`<--- Initialisation de M2`
`for i from 0 to n-1 do`	`<--- Calcul du nombre`
`  M2 := M2+X2[i+1]*2^i`	`  M2=somme X2[i+1]*2^i`
`end do:`	
`M2-M;`	`<--- Vérification de l'égalité M2=M`

MAGMA - Programme 2.5.
Dans Magma, on peut programmer facilement aussi le cryptosystème de Merkle et Hellman. Pour celà, on reprend tout d'abord la fonction **sol** 2.1 telle que définie dans 2.1

```
      Programme (Magma)

#Génération des clés par A:
a:= [4, 6, 11, 25, 50, 110];
n:=#a;
T:=0;
for i := 1 to n do T:=T+a[i];end for;
N:=Random(T, 2*T);
d:=N;
while Gcd(N,d) gt 1do
       d:= Random(2, N-1);
end while;
e:=[];
for i := 1 to n do e:=e cat [d*a[i] mod N];end for;
# Chiffrement par B
M:=61;
a2:=[];
for i := 0 to n-1 do a2:=a2 cat [2^i];end for;
X :=sol(a2, M);
S := 0;
for i:=1 to n do S := S+X[i]*e[i];end for;
# Déchiffrement par A
S2:=S*InverseMod(d,N) mod N;
X2 := sol(a, S2);
M2 := 0;
for i:=0 to n-1 do M2 := M2+X[i+1]*2^i;end for;
M2-M;
```

2.3 Application de LLL au problème du sac à dos

Supposons que les entités **A** et **B** sont entrain d'utliser le cryptosystème de Merkle et Hellman. Les paramètres publiques sont donc :

- La clé publique $(e_1, \cdots, e_n)$ de **A**.
- Le message S envoyé par **B** à **A**.

On sait aussi que ces paramètres vérifient une relation secrète $S = \sum_{i=1}^{n} x_i e_i$ pour une suite $(x_1, \cdots, x_n) \in \{0,1\}^n$. On considère le vecteur

$$U = \left(x_1, x_2, \cdots, x_n, \sum_{i=1}^{n} x_i e_i - S \right).$$

On peut écrire alors le vecteur U sous la forme

$$U = \sum_{i=1}^{n} x_i b_i + b_{n+1},$$

avec

$$\begin{array}{rcl} b_1 & = & (1, 0, 0, \cdots, 0, e_1 C), \\ b_2 & = & (0, 1, 0, \cdots, 0, e_2 C), \\ b_3 & = & (0, 0, 1, \cdots, 0, e_3 C), \\ \vdots & \vdots & \qquad\qquad \vdots \\ b_n & = & (0, 0, 0, \cdots, 1, e_n C), \\ b_{n+1} & = & (0, 0, 0, \cdots, 0, -SC), \end{array}$$

où C est un nombre à déterminer plus tard pour garantir le succès de la méthode. Ceci nous amène à considérer le réseau L défini par :

$$L := \left\{ \sum_{i=1}^{n+1} a_i b_i \mid a_i \in \mathbb{Z} \right\}.$$

La matrice associée à ce réseau est alors :

$$M(L) = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & e_1 C \\ 0 & 1 & 0 & \cdots & 0 & e_2 C \\ 0 & 0 & 1 & \cdots & 0 & e_3 C \\ \vdots & \vdots & \vdots & \ldots & \vdots & \vdots \\ 0 & 0 & 0 & \vdots & 1 & e_n C \\ 0 & 0 & 0 & \vdots & 0 & -SC \end{pmatrix}.$$

La matrice $M(L)$ est diagonale inférieure et son déterminant est $\det M(L) = -SC$. Son inverse est :

$$M(L)^{-1} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & \frac{e_1}{S} \\ 0 & 1 & 0 & \cdots & 0 & \frac{e_2}{S} \\ 0 & 0 & 1 & \cdots & 0 & \frac{e_3}{S} \\ \vdots & \vdots & \vdots & \ldots & \vdots & \vdots \\ 0 & 0 & 0 & \vdots & 1 & \frac{e_n}{S} \\ 0 & 0 & 0 & \vdots & 0 & -\frac{1}{SC} \end{pmatrix}.$$

Puisque $U = \sum_{i=1}^{n} x_i b_i + b_{n+1}$, alors U est un vecteur du réseau L et sa norme euclidenne vérifie :

$$\|U\| = \sqrt{x_1^2 + \cdots + x_n^2 + 1} \leq \sqrt{n+1}.$$

Ceci signifie que U est un vecteur court du réseau L. Plus précisement, on sait que l'algorithme LLL produit des vecteurs $(v_1, \cdots, v_n, v_{n+1})$ avec, pour $1 \leq i \leq n+1$:

$$\|v_i\| \leq 2^{\frac{n(n-1)}{4(n-i+1)}} \det(L)^{\frac{1}{n-i+1}}.$$

Pour que le vecteur U soit produit par l'algorithme LLL, il suffit que $U = v_i$ avec $1 \leq i \leq n+1$, ce qui est réalisé si

$$\sqrt{n+1} \leq 2^{\frac{n(n-1)}{4(n-i+1)}} \det(L)^{\frac{1}{n-i+1}} = 2^{\frac{n(n-1)}{4(n-i+1)}} (CS)^{\frac{1}{n-i+1}},$$

ce qui donne

$$C \geq \frac{(n+1)^{\frac{n-i+1}{2}}}{2^{\frac{n(n-1)}{4}} S}.$$

En réduisant le réseau, on peut alors déterminer un vecteur court $V = (y_1, \cdots, y_n, y_{n+1})$. Ce vecteur s'exprime dans la base $(b_1, \cdots, b_{n+1})$ sous la forme

$$V = (y_1, \cdots, y_n, y_{n+1}) = (z_1, \cdots, z_n, z_{n+1}) M(L),$$

ce qui donne

$$(z_1, \cdots, z_n, z_{n+1}) = (y_1, \cdots, y_n, y_{n+1}) M(L)^{-1} = \left(y_1, \cdots, y_n, \frac{1}{S} \sum_{i=1}^{n} y_i e_i - \frac{y_{n+1}}{S} \right).$$

Pour que la suite $(y_1, \cdots, y_n)$ soit une solution pour le problème, on doit avoir $\sum_{i=1}^{n} y_i e_i = S$ et donc $y_{n+1} = 0$, ce qui donne les conditions :

$$\begin{cases} |y_{n+1}| = 0 \\ |y_i| \leq 1 \quad \text{pour } 1 \leq i \leq n \\ \sum_{i=1}^{n} y_i e_i = S. \end{cases} \tag{3}$$

Avec ces conditions, on a

$$\frac{1}{S} \sum_{i=1}^{n} y_i e_i - \frac{y_{n+1}}{S} = 1,$$

et la suite $(z_1, \cdots, z_n)$ est alors une solution $(x_1, \cdots, x_n)$. A l'aide des coefficients $x_1, \cdots, x_n$, on peut alors retrouver le message clair M en calculant $M = \sum_{i=1}^{n} x_1 2^{i-1}$.

Pour illustrer l'application de l'algorithme au problème du sac à dos, on prend l'exemple :

$$e = [205, 119, 281, 56, 112, 171], \qquad S = 825.$$

La matrice formée par les vecteurs de la base du réseau L est donc sous la forme

$$M(L) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 205C \\ 0 & 1 & 0 & 0 & 0 & 0 & 119C \\ 0 & 0 & 1 & 0 & 0 & 0 & 281C \\ 0 & 0 & 0 & 1 & 0 & 0 & 56C \\ 0 & 0 & 0 & 0 & 1 & 0 & 112C \\ 0 & 0 & 0 & 0 & 0 & 1 & 171C \\ 0 & 0 & 0 & 0 & 0 & 0 & -825C \end{pmatrix}.$$

En appliquant l'algorithme LLL sous Maple 12 avec C=10, on obtient la matrice

$$\begin{pmatrix} 0 & 0 & 0 & -2 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ -1 & -3 & 2 & 0 & 0 & 0 & 0 \\ 2 & -3 & 0 & 0 & -2 & 1 & 0 \\ 1 & -2 & -2 & 0 & 1 & -2 & 0 \\ -1 & -1 & -3 & 0 & 0 & 2 & 0 \\ 0 & 0 & 1 & -1 & -2 & 0 & 10 \end{pmatrix}.$$

Seul le vecteur $(1, 0, 1, 1, 1, 1, 0)$ vérifie les conditions 3 et donne

$$(x_1, x_2, x_3, x_4, x_5, x_6) = (1, 0, 1, 1, 1, 1),$$

ce qui permet d'avoir $205 + 281 + 56 + 112 + 171 = 825$, et donc le message secret est

$$M = \sum_{i=1}^{n} x_i 2^{i-1} = 61.$$

MAPLE - Programme 2.6.
Avec Maple 12, l'exemple ci-dessus peut être réalisé comme ci-dessous.

Programme (Maple 12)	Commentaires
restart;	
with(IntegerRelations):	<--- Package LLL et PSLQ:
with(LinearAlgebra):	<--- Matrices et vecteurs
n:=6;	<--- Nombre d'inconnues
S:=825;	<--- Valeur de S
C:=10;	<--- Valeur de C
e[1]:=205*C;e[2]:=119*C;e[3]:=281*C;	<--- Valeurs de la suite e
e[4]:=56*C;e[5]:=112*C;e[6]:=171*C;	
e[7]:=-S*C;	
for i to n+1 do	<--- Construction de
for j to n+1 do	<--- la base b
if i=j then b[i, j]:=1;	
else b[i, j]:=0 ;	
end if;	
end do;	
end do;	<--- Fin
for i to n+1 do	<--- Construction de
b[i, n+1]:=e[i]	la base b
end do;	
for i to n+1 do	
for j to n+1 do	
b[i]:=[seq(b[i, j],j=1..n+1)];	
end do;	
end do;	<--- Fin
ML:=[seq(b[i], i = 1 .. n+1)];	<--- La matrice M(L)
N:=LLL(ML,'integer');	<--- N=Application de LLL
i := 0;	<--- Initialisation
while i < n+1 do	<--- Vérification des
i:=i+1;	<--- conditions:
if N[i,n+1]=0 then	<--- y_{n+1}>0
j := 0;	
while j<n+1 do	
j:=j+1;	
if abs(N[i, j])>1	<--- abs(y_i)=0 ou 1
then j:=n+1	
else if j=n+1 then	
print("La solution est dans :",N[i]);	<--- La solution
i := n+1	
end if;	
end if;	
end do;	
end if;	
end do;	<--- Réponse [1,0,1,1,1,1,0]

MAGMA - Programme 2.7.

Avec Magma, le même exemple peut être réalisé comme ci-dessous.

Programme (Magma)	Commentaires
`n:=6;`	`<--- Nombre de valeurs n=6`
`S:=825;`	`<--- Valeur de S`
`C:=10;`	`<--- Valeur de C`
`e := AssociativeArray();`	`<--- Création de la liste e`
`e[1]:=205*C;e[2]:=119*C;e[3]:=281*C;`	`<--- La suite e`
`e[4]:=56*C;e[5]:=112*C;e[6]:=171*C;`	
`e[7]:=-S*C;`	
`b:=RMatrixSpace(IntegerRing(),n+1,n+1)!0;`	`<--- Création de la base b`
`for i in [1..n+1] do`	`<--- Création de la base b`
`  for j in [1..n+1] do`	
`  if i eq j then b[i][j]:=1;`	
`  else b[i][ j]:=0 ;`	
`  end if;`	
`  end for;`	
`end for;`	`<--- Fin`
`for i in [1..n+1] do`	`<--- Suite de b`
`  b[i][n+1]:=e[i];`	
`end for;`	
`N:=LLL(b) ;`	`<--- N=LLL(b)`
`i := 0;`	`<--- Initiation de i`
`while i lt n+1 do`	`<--- Recherche d'une`
`  i:=i+1;`	`solution vérifiant`
`  if N[i][n+1] eq 0 then`	`les conditions (1)`
`  j := 0;`	
`  while j lt n+1 do`	
`  j:=j+1;`	
`  if Abs(N[i, j])gt 1`	
`  then j:=n+1;`	
`  else if j eq n+1 then`	
`print "La solution est dans  ",N[i];`	
`  i := n+1;`	
`  end if;`	
`  end if;`	
`  end while;`	
`  end if;`	
`end while;`	`<--- Réponse [1,0,1,1,1,1,0]`

3 Application au cryptosystème NTRU

Le cryptosystème NTRU a été présenté en 1996 dans la rump session de Crypto 96 et publié en 1998 [6]. Le domaine de calculs de NTRU est l'anneau des polynômes $\mathbb{Z}[X]/(X^N-1)$ et utilise des réductions modulo deux nombres (ou polynômes) premiers entre eux p et q.

NTRU se compose de deux protocoles. Le protocole de chiffrement-déchiffrement NTRUEncrypt et le protocole de signature NTRUSign. NTRU est aujourd'hui considéré comme fiable par le standard IEEE P1363.1 [8].

3.1 Définitions

Le cryptosystème NTRU utilise plusieurs sortes de paramètres, en particulier les paramètres N, p et q.

Soit N un nombre entier. Les opérations de NTRU ont pour domaine l'anneau des polynômes suivant :

$$\mathcal{P} = \mathbb{Z}[X]/\left(X^N-1\right).$$

Ainsi, les éléments f de $\mathcal{P}$ peuvent être représentés sous la forme

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

L'addition de deux polynômes $f, g \in \mathcal{P}$ se fait de façon naturelle terme à terme de même degrés, alors que la multiplication se fait par un produit de convolution noté ici $*$. Si $f * g = h$ avec $f = \sum_{i=0}^{N-1} f_i X^i$ et $g = \sum_{i=0}^{N-1} g_i X^i$, alors $h = \sum_{i=0}^{N-1} h_i X^i$ avec pour tout $0 \leq k \leq N-1$,

$$h_k = \sum_{i+j \equiv k \mod N} f_i g_j = \sum_{i=0}^{k} f_i g_{k-i} + \sum_{i=k+1}^{N-1} f_i g_{N+k-i}.$$

La table ci-dessous illustre le produit $h = f * g$ avec $N = 7$, $f = \sum_{i=0}^{6} f_i X^i$ et $g = \sum_{i=0}^{6} g_i X^i$.

MAPLE - Programme 3.1.
Avec Maple 12, la procédure suivante permet de calculer $f * g$ dans l'anneau $\mathcal{P}$.

	1	X	X^2	X^3	X^4	X^5	X^6
	f_0g_0	f_0g_1	f_0g_2	f_0g_3	f_0g_4	f_0g_5	f_0g_6
$+$	f_1g_6	f_1g_0	f_1g_1	f_1g_2	f_1g_3	f_1g_4	f_1g_5
$+$	f_2g_5	f_2g_6	f_2g_0	f_2g_1	f_2g_2	f_2g_3	f_2g_4
$+$	f_3g_4	f_3g_5	f_3g_6	f_3g_0	f_3g_1	f_3g_2	f_3g_3
$+$	f_4g_3	f_4g_4	f_4g_5	f_4g_6	f_4g_0	f_4g_1	f_4g_2
$+$	f_5g_2	f_5g_3	f_5g_4	f_5g_5	f_5g_6	f_5g_0	f_5g_1
$+$	f_6g_1	f_6g_2	f_6g_3	f_6g_4	f_6g_5	f_6g_6	f_6g_0
$h=$	h_0	h_1	h_2	h_3	h_4	h_5	h_6

FIGURE 3 – Calcul de $h = f * g$

Programme (Maple)	Commentaires
`star:=proc (N,f,g)`	`<--- Procedure star`
`local k, h, i, s;`	`<--- Variables locales`
`for k from 0 to N-1 do`	`<--- Création de la`
`    h[k] := 0;`	`     suite h_k`
`    for i from 0 to k do`	`<--- Partie i=0..k`
`h[k]:=h[k]+coeff(f,X,i)*coeff(g,X,k-i)`	
`    end do;`	`<--- Fin partie i=0..k`
`    for i from k+1 to N-1 do`	`<--- Partie i=k+1..N-1`
`h[k]:=h[k]+coeff(f,X,i)*coeff(g,X,N+k-i)`	
`    end do;`	`<--- Fin partie i=k+1..N-1`
`end do;`	
`s := 0;`	`<--- Formation du`
`for k from 0 to N-1 do`	`     polynôme h=f*g`
`    s := s+h[k]*X^k`	
`end do;`	`<--- Fin`
`return s;`	`<--- Sortie de h=f*g`
`end proc;`	`<--- Fin de la procédure`

Pour illustrer cette procédure, on considère l'exemple suivant $N = 17$, $f = X^{16} - 1$ et $g = X^{15} + 1$. Le programme suivant permet de calculer $s = f + g$ et $h = f * g$ dans $\mathcal{P} = \mathbb{Z}[X]/(X^{17} - 1)$. On suppose donc que la procédure **star(N,f,g)** est déjà exécutée.

Programme (Maple)	Commentaires
N:=17;	<--- Valeur de N
f:=X^16-1;	<--- Polynôme f
g:=X^15+1;	<--- Polynôme g
h:=star(N,f,g);	<--- h=f*g=-1+X^14-X^15+X^16

MAGMA - Programme 3.2.
Avec Magma, on considère $N = 17$, $f = X^{16} - 1$ et $g = X^{15} + 1$. Le programme suivant permet de calculer $s = f + g$ et $h = f * g$.

Programme (Magma)	Commentaires
P<X>:= PolynomialRing(Integers());	<--- Anneau des polynômes
N:=17;	<--- Exposant
f:=X^16-1;	<--- Polynôme f
g:=X^15+1;	<--- Polynôme g
s:=f+g mod (X^N-1);	<--- somme mod (X^N-1)
h:=f*g mod (X^N-1);	<--- produit mod (X^N-1)
s;	<--- f+g=X^16 + X^15
h;	<--- f*g=X^16 - X^15 + X^14 - 1

Soient $p, q \in \mathcal{P}$ deux entiers premiers entre eux. Généralement, q est de la forme $q = 2^l$ avec $l = \lfloor \log_2 N \rfloor$ et $p = 3$ ou dans certaines versions $p = 2 + X$. On note $\mathbb{Z}_q$ l'anneau des entiers modulo q, c'est à dire $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$. Dans NTRU, $\mathbb{Z}_q$ est représenté par l'intervalle $\left[-\frac{q}{2}, \frac{q}{2}\right[$. En réduisant $\mathcal{P}$ modulo q, on obtient l'anneau

$$\mathcal{P}_q = \mathbb{Z}_q[X]/(X^N - 1).$$

De façon similaire, pour un polynôme $p \in \mathcal{P}$, on définit l'anneau $\mathcal{P}_p$ par

$$\mathcal{P}_p = \mathbb{Z}[X]/\left(X^N - 1, p\right),$$

obtenu en réduisant $\mathcal{P}$ modulo p. Si p est un entier, alors $\mathcal{P}_p = \mathbb{Z}_p[X]/\left(X^N - 1\right)$. Une notion importante dans NTRU est le calcul de l'inverse d'un polynôme f dans $\mathcal{P}_p$ et $\mathcal{P}_q$. L'inverse de f dans $\mathcal{P}_p$ est un polynôme $f_p \in \mathcal{P}_p$ tel que

$$f * f_p \equiv 1 \pmod{p}.$$

De même, l'inverse de f dans $\mathcal{P}_q$ est un polynôme $f_q \in \mathcal{P}_q$ vérifiant

$$f * f_q \equiv 1 \pmod{q}.$$

MAPLE - Programme 3.3.

Avec Maple 12, la procédure suivante permet de calculer l'inverse f_p de f dans $\mathcal{P}_p$. Pour l'algorithme de calcul, on peut consulter [14]. On suppose que la procédure **star(N,f,g)** est déjà exécutée.

```
      Programme  (Maple)

invp := proc (N, p, a)
with(MatrixPolynomialAlgebra);
k := 0;
b := 1;
c := 0;
f := a;
g := X^N-1;
while true do
    f0 := coeff(f, X, 0);
    while f0 = 0 do
        f := expand(f/X);
        c := expand(c*X);
        k := k+1;
        f0 := coeff(f, X, 0)
    end do;
    if degree(f) = 0 then
        b := `mod`(b/f0, p);
        fp := star(N, b, X^(`mod`(-k, N)));
        break
    end if;
    if degree(f) < degree(g) then
        t := f; f := g; g := t;
        t := b; b := c; c := t
    end if;
    f0 := coeff(f, X, 0);
    g0 := coeff(g, X, 0);
    u := `mod`(f0/g0, p);
    f := `mod`(f-u*g, p);
    b := `mod`(b-u*c, p)
end do;
return fp;
end proc:
```

Pour illustrer cette procédure, on considère l'exemple suivant $N = 11$, $p = 3$ et $f = 1 + X - X^3 + X^5 - X^6$. Le programme suivant permet de calculer f_p dans $\mathcal{P}_p$.

Programme (Maple)	Commentaires
N:=11;	
p:=3;	
f:=1+X-X^3+X^5-X^6;	
fp:=invp(N,p,f);	<-- fp=2X^8+2X^7+2X^6+X^5+2X^4+2*X^3+2X^2+X+2
star(N,f,fp) mod p;	<-- vérification f*fp=1 mod p

De même, le programme ci-dessous permet de calculer l'inverse f_q de f dans $\mathcal{P}_q$. Pour l'algorithme de calcul, on peut aussi consulter [14]. On suppose que les procédures **star(N,f,g)** et **inv(N,p,f)** sont déjà exécutées.

```
      Programme  (Maple)

invq := proc (N,p,r,a)
q := p^r;
b := invp(N, p, a);
qq := p;
while qq < q do
    qq := qq*p;
    b := `mod`(star(N, b, 2-star(N, a, b)), q)
end do;
return b;
 end proc:
```

Pour illustrer cette procédure, on considère l'exemple suivant $N = 11$, $p = 2$, $r = 3$ et $f = 1 + X - X^3 + X^5 - X^6$. Le programme suivant permet de calculer f_q dans $\mathcal{P}_q$.

Programme (Maple)	Commentaires
N:=11;	
p:=2;	
r:=3;	
q:=p^r;	
f:=1+X-X^3+X^5-X^6;	
fq:=invq(N,p,r,f);	<-- fq=X^10+7X^9+7X^8+7X^7+5X^6+2X^5+3X^4+4X^3+X+4
star(N,f,fq) mod q;	<-- vérification f*fq=1 mod q

MAGMA - Programme 3.4.
Avec Magma, la fonction suivante permet de calculer l'inverse f_p de f dans $\mathcal{P}_p$. Pour l'algorithme de calcul, on peut consulter [14].

```
      Programme  (Magma)

function invp(N, p, a)
R<X> := PolynomialRing( Integers() );
k := 0; b := 1; c := 0;
f := a; g := X^N - 1;
while true do
    f0 := Evaluate(f, 0);
    while f0 eq 0 do
        f := f div X; c := c * X; k:=k+1;
        f0 := Evaluate(f, 0);
    end while;
    if Degree(f) eq 0 then
        b := (b * Modinv( f0, p )) mod p;
        f1p:= b*X^(-k mod N) mod (X^N-1);
        break;
    end if;
    if Degree( f ) lt Degree( g ) then
        t := f; f := g; g := t;
        t := b; b := c; c := t;
    end if;
    f0 := Evaluate(f, 0);
    g0 := Evaluate(g, 0);
    u := (Modinv( g0, p ) * f0) mod p;
    f := (f - u*g) mod p;
    b := (b - u*c) mod p;
end while;
return f1p;
end function;
```

Pour illustrer cette procédure, on considère l'exemple suivant $N = 11$, $p = 3$ et $f = 1 + X - X^3 + X^5 - X^6$. Le programme suivant permet de calculer f_p dans $\mathcal{P}_p$.

Programme (Magma)	Commentaires
N:=11;	
p:=3;	
f:=1+X-X^3+X^5-X^6;	
fp:=invp(N,p,f);	<-- fp=2X^8+2X^7+2X^6+X^5+2X^4+2*X^3+2X^2+X+2
f1p*f1 mod (X^N-1) mod p;	<-- vérification f*fp=1 mod p

De même, le programme ci-dessous permet de calculer l'inverse f_q de f dans $\mathcal{P}_q$. Pour l'algorithme de calcul, on peut aussi consulter [14]. On suppose que les

procédures **star(N,f,g)** et **inv(N,p,f)** sont déjà exécutées.

```
      Programme  (Magma)

function invq(N,p,r,a)
R<X> := PolynomialRing( Integers() );
q := p^r;
b := invp(N, p, a);
qq := p;
while qq lt q do
    qq := qq*p;
    b :=(b*(2-a*b mod (X^N-1)) mod (X^N-1)) mod q;
end while;
return b;
 end function;
```

Pour illustrer cette procédure, on considère l'exemple suivant $N = 11$, $p = 2$, $r = 3$ et $f = 1 + X - X^3 + X^5 - X^6$. Le programme suivant permet de calculer f_q dans $\mathcal{P}_q$.

```
   Programme  (Magma)

N:=11;
p:=2;
r:=3;
q:=p^r;
f:=1+X-X^3+X^5-X^6;
fq:=invq(N,p,r,f);
star(N,f,fq) mod q;
```

```
   Commentaires

<-- fq=X^10+7X^9+7X^8+7X^7+5X^6+2X^5+3X^4+4X^3+X+4
<-- vérification f*fq=1 mod q
```

NTRU est basé sur la combinaison de plusieurs paramètres qui ont évolué suivant les différentes attaques. Soit d un entier positif. On pose

$$\mathcal{B}(d) = \left\{ f \in \mathbb{Z}_2[X]/\left(X^N - 1\right) \mid f = \sum_{i=1}^{d} X^{n_i},\ 0 \leq n_1 < \cdots < n_d \leq N \right\}.$$

Ceci signifie que les éléments de $\mathcal{B}(d)$ ont exactement d coefficients égaux à 1 et le reste des coefficients est nul. De même, soient d_1 et d_2 deux entiers positifs. On pose

$$\mathcal{T}(d_1, d_2) = \left\{ f \in \mathbb{Z}_3[X]/\left(X^N - 1\right) \mid f = \sum_{i=1}^{d_1} X^{n_i} - \sum_{j=1}^{d_2} X^{m_j},\ n_i \neq m_j \right\}.$$

Autrement dit, $\mathcal{T}(d_1, d_2)$ est l'ensemble des polynômes ayant exactement d_1 coefficients égaux à 1, d_2 coefficients égaux à -1 et le reste des coefficients est nul.

Les paramètres de NTRU sont alors les suivants.

- Un nombre entier N. Ce nombre doit être premier et assez grand.
- Un nombre entier q, généralement de la forme $q = 2^l$ avec $l = \lfloor \log_2(N) \rfloor$.
- Un paramètre p qui est soit un nombre entier p premier avec q soit un polynôme $p = \alpha + \beta X \in \mathbb{Z}[X]$, inversible modulo q. Généralement $p = 2 + X$ ou $p = 2$.
- Un ensemble $\mathcal{L}_f \subset \mathcal{P}$ de polynômes f.
- Un ensemble $\mathcal{L}_g \subset \mathcal{P}$ de polynômes g.
- Un ensemble $\mathcal{L}_m \subset \mathcal{P}$ des messages secrets.
- Un ensemble $\mathcal{L}_r \subset \mathcal{P}$ des polynômes auxiliaires secrets.

Ainsi, suivant les versions de NTRU, les paramètres ci-dessus ont évolué en moyenne tous les trois ans.

Version	p	q	$\mathcal{L}_f$	$\mathcal{L}_g$	$\mathcal{L}_r$	$\mathcal{L}_m$
1998	3	2^k	$\mathcal{T}(d_f, d_f - 1)$	$\mathcal{T}(d_g, d_g)$	$\mathcal{T}(d_r, d_r)$	$\mathcal{T}(d_m, d_m)$
2001	$2 + X$	2^k	$1 + p * F$	$\mathcal{B}(d_g)$	$\mathcal{B}(d_r)$	$\mathcal{B}(d_m)$
2005	2	premier	$1 + p * F$	$\mathcal{B}(d_g)$	$\mathcal{B}(d_r)$	$\mathcal{B}(d_m)$

Les nombres entiers d_f, d_g, d_r et d_m sont fixés pour chaque version. Par exemple, pour une sécurité moyenne, NTRU préconise de prendre $N = 251$, $p = 2$ $q = 128$, $d_f = 72$, $d_f = 71$, $d_r = 72$.

3.2 Utilisation de NTRU

L'utilisation du cryptosystème NTRU se déroule de la façon suivante. Supposons qu'une personne B souhaite envoyer un message M à une personne A. Alors le processus se fait en trois étapes :

1. A doit générer une clé publique h et des clés privées f et f_p.
2. B doit transformer le message clair M en un message chiffré e et envoyer le message chiffré e à A.
3. A doit déchiffrer le message reçu e et retrouver le message clair M.

3.2.1 Générations de clés

Cette étape est réalisée par la personne A qui va recevoir le message M.

Génération des clés :

1. Choisir aléatoirement un polynôme $f \in \mathcal{L}_f$.
2. Calculer l'inverse f_q de f dans $\mathcal{P}_q$, c'est à dire $f * f_q \equiv 1 \pmod{q}$.
3. Calculer l'inverse f_p de f dans $\mathcal{P}_p$, c'est à dire $f * f_p \equiv 1 \pmod{p}$.
4. Choisir aléatoirement un polynôme $g \in \mathcal{L}_g$.
5. Calculer $h \equiv g * f_q \pmod{q}$ dans $\mathcal{P}_q$.

La clé publique est alors h et la clé secrète est (f, f_p). Pour illustrer cette étape, on prend

$$\begin{aligned} N &= 13,\ p = 3,\ q = 8, \\ f &= X^{12} + X^{11} + X^{10} + X^9 + X^8 + X^7 + 1, \\ g &= X^{12} + X^5 - X^4 + X^3 - X^2 + X - 1. \end{aligned}$$

On obtient alors les polynômes suivants :

$$\begin{aligned} f_p &= 2X^{12} + 2X^{11} + 2X^{10} + 2X^9 + 2X^8 + 2X^7 + 2X^5 + 2X^4 + 2X^3 + 2X^2 + 2X, \\ f_q &= X^{12} + X^{11} + X^{10} + X^9 + X^8 + X^7 + 2X^6 + X^5 + X^4 + X^3 + X^2 + X + 2, \\ h &\equiv g * f_q \pmod{q} = 2X^{12} + 2X^{11} + 2X^9 + 2X^7 + 3X^5 + 2X^3 + 2X. \end{aligned}$$

Pour les algorithmes de calcul de f_p et f_q, on peut consulter [10].

3.2.2 Chiffrement

Pour chiffrer un message M, la personne B doit le transformer en un polynôme $m \in \mathcal{L}_m$ et ensuite le transformer en un polynôme chiffré e avec la procédure suivante.

Chiffrement :

1. Choisir aléatoirement un polynôme $r \in \mathcal{L}_r$.
2. Calculer $e \equiv p * r * h + m \pmod{q}$ dans $\mathcal{P}_q$.

Le message chiffré est alors e. On prend maintenant les polynômes suivants :

$$\begin{aligned} m &= X^{10} + X^8 + X^7 + X^4 + X^3 + 1, \\ r &= X^{12} + X^{11} + X^8 + X^7 + 1. \end{aligned}$$

On obtient alors :

$$\begin{aligned} e &\equiv p * r * h + m \pmod{q} \\ &\equiv 5X^{12} + 2X^{11} + 3X^{10} + 2X^9 + 5X^8 + 3X^7 + 2X^6 + 5X^5 + 6X^4 + 4X^3 + 2X. \end{aligned}$$

3.2.3 Déchiffrement

Pour déchiffrer un message $e \in \mathcal{P}_q$, la personne A doit utiliser les clés secrètes f et f_p dans la procédure suivante.

Déchiffrement :

1. Calculer $a \equiv f * e \pmod{q}$ dans $\mathcal{P}_q$ avec, si nécessaire, des coefficients dans l'intervalle $\left[-\frac{q}{2}, \frac{q}{2}\right[$.
2. Calculer $m \equiv f_p * a \pmod{p}$ dans $\mathcal{P}_p$.

En poursuivant l'exemple ci-dessous, on obtient :

$$\begin{aligned} a &\equiv f * e \pmod{q} \\ &\equiv 6X^{12} + 3X^{11} + 6X^{10} + 2X^9 + 3X^8 + 4X^7 \\ &\quad +6X^6 + 6X^5 + 4X^4 + 7X^3 + X^2 + 6X + 3. \end{aligned}$$

Finalement, on obtient :

$$\begin{aligned} m &\equiv f_p * a \pmod{p} \\ &\equiv X^{10} + X^8 + X^7 + X^4 + X^3 + 1, \end{aligned}$$

qui est exactement le message de départ.

3.2.4 Exactitude du déchiffrement

Pour s'assurer de l'exactitude du déchiffrement, on commence par calculer $a \equiv f * e \pmod{q}$ dans $\mathcal{P}_q$ sachant que $e \equiv r * h + m \pmod{q}$ et que $h \equiv p * g * f_q \mod q$. On a

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

Maintenant, soit $a' = p * r * g + f * m$ défini par un calcul exacte dans $\mathcal{P}$ et non dans $\mathcal{P}_q$, c'est à dire sans modulo q. Si les coefficients de a' sont dans un intervalle $[A, A+q[$, alors en ramenant les coefficients de a dans le même intervalle $[A, A+q[$,

on obtient $a = a'$ et en réduisant a modulo p, on obtient $a \equiv f * m \pmod{p}$. Avec l'exemple ci-dessus, on a

$$a' = 6X^{12}+3X^{11}+6X^{10}+2X^9+3X^8+4X^7+6X^6+6X^5+4X^4+7X^3+X^2+6X+3.$$

Les coefficients de a' sont ainsi dans l'intervalle $[0, q]$, ce qui est conforme à la remarque ci-dessus.

MAPLE - Programme 3.5.
Le programme ci-dessous permet de vérifier la validité de NTRU à l'aide de Maple. On suppose que les procédure **star**, **invp** et **invq** sont déjà initialisées.

```
      Programme  (Maple)

N := 13: q := 8: p := 3:
f := X^12+X^11+X^10+X^9+X^8+X^7+1:
g := X^12+X^5-X^4+X^3-X^2+X-1:
fp := invp(N, p, f):
fq := invq(N, 2, 3, f):
h := `mod`(star(N, fq, g), q):
m := X^10+X^8+X^7+X^4+X^3+1:
r := X^12+X^11+X^8+X^7+1:
e := `mod`(star(N, p*r, h)+m, q):
a := `mod`(star(N, f, e), q):
m2 := `mod`(star(N, fp, a), p):
m-m2;
```

La différence entre le message d'origine m est le message $m2$, obtenu après décryptage est nulle, ce qui confirme l'exactitude de NTRU.

MAGMA - Programme 3.6.
Dans Magma, on peut écrire un programme similaire pour vérifier la validité de NTRU. De même, on suppose que les procédures **invp** et **invq** ont été initialisées.

```
      Programme  (Magma)

N := 13;
q := 8;
p := 3;
R<X> := PolynomialRing( Integers() );
f := X^12+X^11+X^10+X^9+X^8+X^7+1;
g := X^12+X^5-X^4+X^3-X^2+X-1;
fp := invp(N, p, f);fp;
fq := invq(N, 2, 3, f);fq;
h:=(g*fq mod (X^N-1)) mod q;
m:=X^10+X^8+ X^7+X^4+X^3+1;
r:=X^12+ X^11 + X^8+X^7+1;
e:=(p*r*h + m) mod (X^N - 1) mod q;
a:=(f*e) mod (X^N - 1) mod q;
m2:=((fp*a) mod (X^N - 1)) mod p;
m-m2;
```

3.2.5 Application de LLL à NTRU (Coppersmith et Shamir)

Juste après la publication de NTRU, Coppersmith et Shamir [3] ont proposé une attaque contre la clé publique pour les petites valeurs du paramètre N. Dans NTRU, la clé publique $h \equiv f_q * g \pmod{q}$ vérifie $f * h \equiv g \pmod{q}$. Alors, il existe un polynôme $u \in \mathcal{P} = \mathbb{Z}/\left(X^N - 1\right)$ tel que

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

On considère alors l'ensemble

$$\mathcal{L} = \left\{(f, g) \in \mathcal{P}^2 \mid \exists u \in \mathcal{P},\ f * h - q * u = g\right\}.$$

On vérifie facilement que $\mathcal{L}$ est un réseau et sous forme matricielle, ceci s'écrit sous la forme :

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

Avec les coordonnées de f, g et h, l'égalité ci-dessus prend la forme

$$\left[\begin{array}{c} f_0 \\ f_1 \\ \vdots \\ \hline\hline f_{N-1} \\ \hline -u_1 \\ -u_2 \\ \vdots \\ -u_{N-1} \end{array}\right] * \left[\begin{array}{cccc||cccc} 1 & 0 & \cdots & 0 & h_0 & h_1 & \cdots & h_{N-1} \\ 0 & 1 & \cdots & 0 & h_{N-1} & h_0 & \cdots & h_{N-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & h_1 & h_2 & \cdots & h_0 \\ \hline\hline 0 & 0 & \cdots & 0 & q & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & q & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & q \end{array}\right] = \left[\begin{array}{c} f_0 \\ f_1 \\ \vdots \\ \hline\hline f_{N-1} \\ \hline g_0 \\ g_1 \\ \vdots \\ g_{N-1} \end{array}\right]$$

Ainsi, la matrice

$$M(L) = \left[\begin{array}{cc} I_N & h_N \\ 0 & qI_N \end{array}\right]$$

du réseau vérifie $\det(M(L)) = q^N$. D'autre part, on suppose que $f \in \mathcal{T}(d_f, d_f - 1)$ et $g \in \mathcal{T}(d_g, d_g)$ avec $d_f = d_g \approx \frac{N}{3}$. Alors

$$\|(f,g)\| = \left(\sum_{i=0}^{N-1} f_i^2 + \sum_{i=0}^{N-1} g_i^2\right)^{1/2} \approx \sqrt{4d_f} \approx 2\sqrt{\frac{N}{3}}.$$

En appliquant l'algorithme LLL au réseau $\mathcal{L}$, on produit un vecteur v_1 assez court, plus précisement

$$\|v_1\| \leq 2^{\frac{2N-1}{4}} \det(L)^{\frac{1}{2N}}.$$

Ainsi, on peut avoir $v_1 = (f,g)$ si la condition suivante est vérifiée :

$$2\sqrt{\frac{N}{3}} \leq 2^{\frac{2N-1}{4}} \det(L)^{\frac{1}{2N}} \quad \Longleftrightarrow \quad 2^{-\frac{2N-5}{2}} N \leq 3q.$$

Pour illustrer l'utilisation de LLL, on prend l'exemple ci-dessous :

$$\begin{aligned} N &= 13, \quad p = 3 \quad q = 8, \\ f &= 1 - X + X^{12}, \\ f_p &= 1 + 2X + 2X^2 + 2X^4 + X^5 + X^6 + X^8 + 2X^9 + 2X^{10} + 2X^{12}, \\ f_q &= 1 + X + X^3 + 7X^4 + 2X^5 + 5X^6 + 5X^7 + 5X^9 + 3X^{10} + 2X^{11} + X^{12}, \\ g &= 1 + X - X^8 + X^{11}, \\ h &= 6X + 3X^2 + 3X^3 + 3X^5 + 5X^6 + 6X^7 + 7X^8 + 6X^9 + X^{10} + 5X^1 1 + 5X^{12} \pmod{q}. \end{aligned}$$

La matrice du réseau est alors

$$M(L) = \left[\begin{array}{cc} I_N & h_N \\ 0_N & qI_N \end{array}\right]$$

où 0_N est la matrice carrée nulle, I_N est la matrice unité et

$$h_N = \begin{bmatrix} 0 & 6 & 3 & 3 & 0 & 3 & 5 & 6 & 7 & 6 & 1 & 5 & 5 \\ 5 & 0 & 6 & 3 & 3 & 0 & 3 & 5 & 6 & 7 & 6 & 1 & 5 \\ 5 & 5 & 0 & 6 & 3 & 3 & 0 & 3 & 5 & 6 & 7 & 6 & 1 \\ 1 & 5 & 5 & 0 & 6 & 3 & 3 & 0 & 3 & 5 & 6 & 7 & 6 \\ 6 & 1 & 5 & 5 & 0 & 6 & 3 & 3 & 0 & 3 & 5 & 6 & 7 \\ 7 & 6 & 1 & 5 & 5 & 0 & 6 & 3 & 3 & 0 & 3 & 5 & 6 \\ 6 & 7 & 6 & 1 & 5 & 5 & 0 & 6 & 3 & 3 & 0 & 3 & 5 \\ 5 & 6 & 7 & 6 & 1 & 5 & 5 & 0 & 6 & 3 & 3 & 0 & 3 \\ 3 & 5 & 6 & 7 & 6 & 1 & 5 & 5 & 0 & 6 & 3 & 3 & 0 \\ 0 & 3 & 5 & 6 & 7 & 6 & 1 & 5 & 5 & 0 & 6 & 3 & 3 \\ 3 & 0 & 3 & 5 & 6 & 7 & 6 & 1 & 5 & 5 & 0 & 6 & 3 \\ 3 & 3 & 0 & 3 & 5 & 6 & 7 & 6 & 1 & 5 & 5 & 0 & 6 \\ 6 & 3 & 3 & 0 & 3 & 5 & 6 & 7 & 6 & 1 & 5 & 5 & 0 \end{bmatrix},$$

En appliquant l'algorithme LLL à la matrice $M(L)$, on obtient une matrice sous la forme

$$\begin{bmatrix} M_f & M_g \end{bmatrix},$$

avec

$$M_f = \begin{bmatrix}
l_1 \to & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 1 & 0 & 0 & 0 \\
l_2 \to & 0 & 0 & 0 & 0 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
l_3 \to & 0 & 0 & 0 & 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
l_4 \to & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 1 & 0 & 0 \\
l_5 \to & 0 & 0 & 0 & 0 & 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
l_6 \to & 0 & -1 & 2 & -1 & 1 & 1 & 0 & 0 & -1 & 0 & 0 & 1 & -1 \\
l_7 \to & -1 & 0 & 2 & -1 & 0 & 2 & 0 & 0 & -1 & -1 & 0 & 1 & 0 \\
l_8 \to & 0 & 1 & -2 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\
l_9 \to & 0 & -1 & 1 & -1 & 0 & 0 & 0 & 1 & 0 & -1 & 1 & 0 & -1 \\
l_{10} \to & 0 & 0 & -1 & 1 & -1 & 1 & 0 & -1 & 1 & 2 & -2 & 0 & 1 \\
l_{11} \to & 0 & 0 & -1 & 1 & 0 & -1 & 1 & 0 & 1 & 0 & -1 & 0 & 1 \\
l_{12} \to & -1 & 0 & 2 & -1 & 2 & 0 & 1 & -1 & 0 & 1 & -1 & 0 & -1 \\
l_{13} \to & 1 & -1 & 0 & 2 & -1 & -1 & 0 & 0 & 0 & 2 & -1 & 1 & 0 \\
l_{14} \to & 2 & -1 & 0 & 0 & 0 & -1 & -1 & 1 & 0 & 0 & 1 & 0 & -1 \\
l_{15} \to & -1 & 1 & -1 & -1 & 0 & 0 & 1 & -1 & 0 & 1 & 0 & -1 & 0 \\
l_{16} \to & 1 & -1 & 1 & -1 & 2 & -1 & -1 & 1 & 0 & 2 & 1 & 0 & -2 \\
l_{17} \to & 0 & 1 & 0 & 0 & 2 & -1 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\
l_{18} \to & 0 & 0 & 0 & -2 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & -1 & -1 \\
l_{19} \to & 0 & 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
l_{20} \to & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 \\
l_{21} \to & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\
l_{22} \to & 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
l_{23} \to & 0 & -1 & 1 & 0 & 0 & 1 & 1 & -1 & 0 & 0 & -1 & 1 & 0 \\
l_{24} \to & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 1 & 0 \\
l_{25} \to & 0 & 0 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
l_{26} \to & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}$$

et

$$M_g = \begin{bmatrix}
l_1 \to & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & -1 & -1 & 0 & 0 & 0 \\
l_2 \to & 1 & 0 & 0 & -1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
l_3 \to & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
l_4 \to & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & -1 & -1 & 0 & 0 \\
l_5 \to & 7 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
l_6 \to & 3 & -1 & 2 & -1 & -2 & 1 & -1 & 1 & -1 & 1 & -1 & -1 & 2 \\
l_7 \to & -1 & -2 & -1 & 1 & 2 & -1 & -3 & 2 & 2 & 0 & 1 & -4 & -2 \\
l_8 \to & -2 & -4 & 3 & -1 & 1 & -3 & 1 & -4 & -3 & 0 & 3 & 0 & 3 \\
l_9 \to & 1 & 0 & -1 & 0 & -1 & 1 & -2 & -5 & -4 & 1 & 3 & 4 & 1 \\
l_{10} \to & 3 & 5 & -1 & -3 & 2 & 1 & 2 & 1 & 1 & -2 & -2 & 2 & 1 \\
l_{11} \to & 1 & 1 & 0 & 0 & -2 & 4 & 2 & 3 & -1 & -4 & -4 & 1 & 1 \\
l_{12} \to & -3 & 2 & 0 & -1 & 2 & -2 & -2 & 1 & -1 & 1 & -1 & -1 & -1 \\
l_{13} \to & 0 & 2 & 0 & 0 & 1 & -2 & 2 & 2 & 2 & -2 & 0 & -1 & 0 \\
l_{14} \to & 0 & 2 & 0 & 0 & -1 & 0 & -2 & 0 & -1 & 1 & -1 & 2 & 0 \\
l_{15} \to & -3 & 1 & 2 & 0 & 0 & -1 & 0 & 1 & 0 & 1 & -2 & -3 & 0 \\
l_{16} \to & 2 & 1 & 1 & 1 & -1 & 0 & 0 & -2 & -2 & 1 & 0 & 1 & 2 \\
l_{17} \to & 2 & 1 & 1 & 1 & -1 & 0 & 0 & -2 & -2 & 1 & 0 & 1 & 2 \\
l_{18} \to & -2 & 1 & 2 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\
l_{19} \to & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
l_{20} \to & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & -1 \\
l_{21} \to & 0 & 1 & 0 & 0 & -1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\
l_{22} \to & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\
l_{23} \to & 0 & -1 & -1 & 1 & 0 & 2 & -1 & 4 & 3 & -1 & -2 & -1 & -1 \\
l_{24} \to & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & -1 & -1 & 0 \\
l_{25} \to & 0 & -1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
l_{26} \to & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0
\end{bmatrix}.$$

Si l'algorithme LLL réussit, les coefficients de f sont parmi les lignes de M_f et ceux de g sont parmi les lignes de M_g. Les lignes 26 de M_f et de M_g nous donnent

$$\begin{aligned}
f(LLL) &= 1 - X + X^{12} = f, \\
g(LLL) &= 1 + X - X^8 + X^{11} = g.
\end{aligned}$$

On retrouve ainsi les clés privées f et g qui ont permi de fabriquer la clé plublique h.

MAPLE - Programme 3.7.
Dans Maple 12, le programme ci-dessous permet de retrouver f et g à partir de h.

Programme (Maple)	Commentaires
`with(IntegerRelations):`	`<--- Package LLL:`
`N := 13;`	`<--- N= 13`
`q := 8;`	`<--- q = 8`
`P := 6*X+3*X^2+3*X^3+3*X^5+5*X^6+6*X^7` `+7*X^8+6*X^9+X^10+5*X^11+5*X^12;`	`<--- Polynôme h`
`for i from 0 to N-1 do` `h[i] := coeff(P, X, i)` `end do:`	`<--- Coefficients de h`
`delta := proc (i, j)` `if i = j then return 1` `else return 0` `end if end proc:`	`<--- delta(i,j)=1 si i=j` `delta(i,j)=0 sinon`
`for i from 0 to N-1 do` `l[i+1]:=seq(delta(i, j),j=0..N-1)` `end do:`	`<--- Matrice I_N`
`for i from 0 to N-1 do` `ll[i+1]:=seq(h[‘mod‘(N-i+j, N)],j=0..N-1)` `end do:`	`<--- Matrice h_N`
`for i from 0 to N-1 do` `b[i+1]:=[l[i+1],ll[i+1]]` `end do:`	`<--- Matrice [I_N H_N]`
`delta2 := proc (i, j)` `if i = j then return q` `else return 0` `end if` `end proc:`	`<--- Matrice 0_N`
`for i from 0 to N-1 do` `l[N+i+1] := seq(delta2(i, j),j=0..N-1)` `end do:`	`<--- Matrice q_N`
`for i from 0 to N-1 do` `b[N+i+1]:=[seq(0,j=0..N-1),l[N+i+1]]` `end do:`	`<--- Matrice [0_N q_N]`
`ML := seq(b[i+1], i = 0 .. 2*N-1):`	`<--- Matrice ML`
`MLL := LLL([ML], 'integer');`	`<--- Application de LLL`

Pour vérifier que toutes les lignes sont des solutions, on peut appliquer le programme ci-dessous et remarquer que $f * P - g \equiv 0 \pmod{q}$.

```
      Programme  (Maple)

for i from 1 to 2*N do
    f:=0;
    for j from 1 to N do
        f:=f+MLL[i,j]*X^(j-1);
    end do;
    g:=0;
    for j from N+1 to 2*N do
        g:=g+MLL[i,j]*X^(j-N-1);
    end do;
    dif:=(star(N, P, f)-g) mod q;
    if (dif=0) then print(i,dif);end if;
end do:
```

Références

[1] R. C. Merkle, M. E. Hellman, Hiding information and signatures in trap door knapsacks, IEEE Transactions on Information Theory, IT-24(5), pp. 525-530, 1978.

[2] H. Cohen, A Course in Computational Number Theory, Graduate Texts in Mathematics, Springer, 1993.

[3] D. Coppersmith and A. Shamir, Lattice attacks on NTRU. In Advances in cryptology—EUROCRYPT '97, volume 1233 of Lecture Notes in Comput. Sci., pages 52–61. Springer, Berlin, 1997.

[4] G. Hanrot, LLL : A Tool for Effective Diophantine Approximation, in : The LLL Algorithm Survey and Applications, Phong Q. Nguyen and Brigitte Vallée Ed. Springer, 2010.

[5] W. Diffie, E. Hellman, New Directions in Cryptography, IEEE Trans- actions on Information Theory, 22, 5 (1976), pp. 644–654.

[6] J. Hoffstein, J. Pipher, and J. H. Silverman, NTRU : A Ring Based Public Key Cryptosystem in Algorithmic Number Theory. Lecture Notes in Computer Science 1423, Springer-Verlag, pages 267–288, 1998.

[7] J. Hoffstein, J. Pipher, and J. H. Silverman, An Introduction to Mathematical Cryptography, Springer-Verlag, UTM, 2008.

[8] IEEE P1363.1 Public-Key Cryptographic Techniques Based on Hard Problems over Lattices, June 2003. IEEE.

[9] A.K. Lenstra, H.W. Lenstra and L. Lovasz, Factoring polynomials with rational coefficients, Mathematische Annalen, Vol. 261, 513—534, 1982.

[10] W. Bosma, J. Cannon, C. Playoust, A. Steel, Solving Problems with Magma. http://magma.maths.usyd.edu.au/magma/pdf/examples.pdf.

[11] NTRU Inc. http://www.ntru.com/

[12] NTRU Inc. http://www.ntru.com/cryptolab/faqs.htm#six

[13] R. Rivest, A. Shamir, L. Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, Communications of the ACM, Vol. 21 (2), 120—126 (1978)

[14] J. H. Silverman, Almost Inverses and Fast NTRU Key Creation," Tech. Rep. 14, NTRU Cryptosystems, Inc., March 1999. Version 1.

[15] D. Simon, Selected applications of LLL in number theory (2008) www.math.unicaen.fr/~simon/maths/lll25_Simon.pdf.