Le problème du sac à dos en cryptographie

Deroff Julien Goyat Rémi

18 juin 2007

Table des matières

1 Cryptographie à clé publique **4**
- 1.1 Cryptographie à clé privée/secrète . . . 4
- 1.2 Systèmes à clé publique . . . 6
  - 1.2.1 Description . . . 6
  - 1.2.2 Exemple simple . . . 8
  - 1.2.3 Exemple : RSA . . . 8

2 Le problème du sac à dos **11**
- 2.1 Présentation du problème . . . 11
- 2.2 Description du sac utilisé . . . 12
- 2.3 Algorithmes et complexité . . . 13
  - 2.3.1 Algorithme exaustif . . . 13
  - 2.3.2 Algorithme glouton . . . 13
  - 2.3.3 Algorithme de type programmation dynamique . . . 14

3 Notions de géométrie des nombres **16**
- 3.1 Définitions . . . 16
- 3.2 Théorème de Minowski . . . 20
- 3.3 Réseaux orthogonaux . . . 21
- 3.4 Les bases LLL-réduites . . . 22
- 3.5 Algorithmes . . . 23
  - 3.5.1 Algorithme Gram-Schmidt . . . 23
  - 3.5.2 Algorithme de réduction faible . . . 24
  - 3.5.3 Algorithme LLL . . . 24

4 Le cryptosystème de Merkle-Hellman **26**
- 4.1 Le problème du sac à dos . . . 26
- 4.2 Description du cryptosystème . . . 27
  - 4.2.1 Génération des clés privée et publique . . . 27
  - 4.2.2 Chiffrement et déchiffrement . . . 28
- 4.3 Cryptanalyse du cryptosystème . . . 30

4.3.1 L'attaque de Lagarias-Odlyzko 31
4.3.2 Le réseau de Coster, La Macchia, Odlyzko et Schnorr . 32
4.3.3 Le réseau de Joux et Stern 35
4.4 Expérimentations 37
4.4.1 Le réseau de Lagarias-Odlyzko 37
4.4.2 Le réseau de Coster, La Macchia, Odlyzko et Schnorr . 37
4.4.3 Le réseau de Joux et Stern 37
4.5 Conclusion 38

A Histogrammes **39**

B Extrait du code source Java **47**

Table des figures

1.1 Cryptosystème à clé secrète/privée 5
1.2 Fonction à sens unique 6
1.3 Principe d'un cryptosysteme à clé publique 7

3.1 Un exemple de réseau 16
3.2 Une base quelconque 19
3.3 Une base réduite 23

A.1 Cryptanalyse d'un message de longueur 16 par le réseau de Lagarias-Odlyzko 39
A.2 Cryptanalyse d'un message de longueur 21 par le réseau de Lagarias-Odlyzko 40
A.3 Cryptanalyse d'un message de longueur 31 par le réseau de Lagarias-Odlyzko 41
A.4 Cryptanalyse d'un message de longueur 51 par le réseau de Lagarias-Odlyzko 42
A.5 Cryptanalyse d'un message de longueur 16 par le réseau de Coster, La Macchia, Odlyzko et Schnorr 42
A.6 Cryptanalyse d'un message de longueur 21 par le réseau de Coster, La Macchia, Odlyzko et Schnorr 43
A.7 Cryptanalyse d'un message de longueur 31 par le réseau de Coster, La Macchia, Odlyzko et Schnorr 43
A.8 Cryptanalyse d'un message de longueur 51 par le réseau de Coster, La Macchia, Odlyzko et Schnorr 44
A.9 Cryptanalyse d'un message de longueur 16 par le réseau de Joux et Stern 44
A.10 Cryptanalyse d'un message de longueur 21 par le réseau de Joux et Stern 45
A.11 Cryptanalyse d'un message de longueur 31 par le réseau de Joux et Stern 45
A.12 Cryptanalyse d'un message de longueur 51 par le réseau de Joux et Stern 46

Chapitre 1

Cryptographie à clé publique

1.1 Cryptographie à clé privée/secrète

La cryptographie, l'écriture des secrets, remonterait, comme certains osent le dire, aussi loin que l'écriture elle-même. Ces dernières années, ce domaine est devenu l'objet de très nombreuses études scientifiques tant le besoin de notre société de transmissions d'informations de manière sûre est devenue vitale. Ces systèmes permettant la transmission secrète des données sont appelés cryptosystèmes.

Malgré leurs différences, le fonctionnement des différents systèmes peut être représenté de manière générale. L'émetteur chiffre le message original en message chiffré à l'aide de la clé de cryptage choisie. Puis, le receveur déchiffre le message reçu en ustilisant la clé de cryptage pour obtenir le message original. (Figure 1.1)
Ou de manière symbolique :

```
E(message original) = message chiffré
R(message chiffré) = message original
```

Si une tierce personne intercepte la transmission, elle n'aura accès qu'au message chiffré qui, à priori, lui est impossible à lire ou comprendre.

Pour illustrer, on peut citer l'un des cryptosystèmes les plus connus : le code César. Cet ancien cryptosystème consiste simplement à un décalage de k lettres dans l'alphabet de l'ensemble du message.

```
Par exemple, pour k=3, le message NEVERSURRENDER,
une fois chiffré, est QHYHUVXUUHQGHU.
```

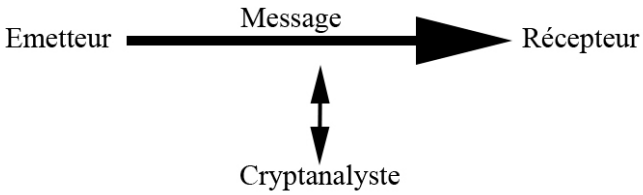

FIG. 1.1 – Cryptosystème à clé secrète/privée

Le code César est basé sur des subsitutions : chaque lettre est substituée par une autre. D'où pour k=3 les substitutions sont les suivantes :

```
Ancien :  ABCDEFGHIJKLMONPQRSTUVWXYZ
Nouveau : DEFGHIJKLMONPQRSTUVWXYZABC
```

La clé de cryptage utilisée dans ce système est l'entier k, correspondant au décalage de k lettres dans l'alphabet.

La méthode de chiffrage $\mathbf{E}_k$ est : décalage de k lettres dans l'alphabet.

La méthode de déchiffrage correspondante $\mathbf{D}_k$ est : décalage inverse de k lettres dans l'alphabet.

Quelques illustrations supplémentaires :

$$\mathbf{E}_{25}(BONJOUR) = ANMINTQ$$

$$\mathbf{E}_3(HELP) = KHOS$$

$$\mathbf{D}_6(SAVOIR) = \mathbf{E}_{20}(SAVOIR) = MUPICL$$

Le cryptosystème César fait partie des systèmes symétriques, ou à clé privée, dont la sécurité repose sur le partage de la connaissance de la clé, que seuls doivent connaître l'émetteur et le récepteur. La symétrie vient de la capacité à déchiffrer un message lorsqu'on sait le chiffrer, et inversement.

Des systèmes actuels tels que DES, AES, ou d'autres, comme le célèbre Enigma, ont été ou sont encore largement utilisés. Et l'on sait depuis 1949, grâce à Claude Shannon, que le chiffrement de Gilbert Vernam, qui consiste à ajouter au message en clair une clé secrète de même longueur n, est parfaitement sûr.

Néanmoins, se pose le problème du partage de la clé de façon absolument sûre.

1.2 Systèmes à clé publique

1.2.1 Description

Pour palier au principal défaut de la cryptographie symétrique, en 1976, Whitfield Diffie et Martin Hellman ont présenté le concept de cryptographie à clé publique (ou cryptographie asymétrique).

Ce concept repose sur l'existence de fonctions à sens unique : une telle fonction est simple à appliquer à un message, mais il est en revanche extrêment coûteux de retrouver ce message une fois transformé.

En fait, les cryptosystèmes à clé publique utilisent des fonctions à sens unique comportant une brèche secrète. Ces fonctions sont très difficiles à inverser, à moins que l'on ne possède une information supplémentaire : la clé privée.

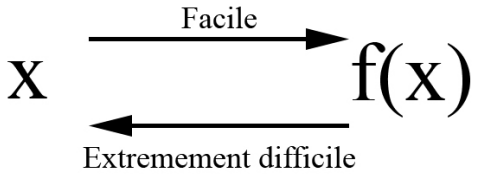

FIG. 1.2 – Fonction à sens unique

On peut le décrire schématiquement de la manière suivante (voir figure 1.3) :

Pour envoyer un message à Julien, Rémi procède ainsi :
– Rémi demande la clé publique à Julien.
– A l'aide de la clé publique, Rémi chiffre le message.
– Rémi envoie le message chiffré à Julien.
– Julien déchiffre alors le message reçu à l'aide de sa clé privée.

FIG. 1.3 – Principe d'un cryptosysteme à clé publique

1.2.2 Exemple simple

On peut illustrer concrètement le concept de cryptographie à clé publique de la manière suivante :

Prenons l'annuaire téléphonique d'une grande ville, il est alors aisé de trouver le numéro d'une personne. Dans l'autre sens, c'est très dur de trouver une personne à partir d'un certain numéro de téléphone.

On utilise le cryptosystème décrit ci-après :

> Pour chaque lettre du message original, on choisit aléatoirement un nom commençant par cette lettre. Le numéro de téléphone correspondant consitue alors le chiffrement de la lettre.
>
> Le chiffrement de HELLO pourrait alors ressembler à :

Lettre	**Nom**	**Lettre chiffrée**
H	Hellman	0298730431
E	El Gamal	0152312457
L	Lenstra	0325794723
L	Lovász	0132870918
O	Odlyzko	0279813300

> On obtient le texte chiffré en mettant à la suite les numéros de téléphone.

Dans cet exemple, la clé publique est l'annuaire et la clé privée est l'annuaire inversé correspondant. Sans l'annuaire inversé, il est extrêmement difficile de déchiffré le message.

Néanmoins, ce premier exemple n'est pas sûr. En effet, l'inversion de l'annuaire peut se faire, par un algorithme de tri, en un temps logarithmique. La fonction inverse n'est pas du tout difficile à calculer.

On a donc besoin de problèmes extrêmement difficiles à résoudre. Selon la théorie de la complexité, ces problèmes sont des problèmes dits NP.

1.2.3 Exemple : RSA

L'un des nombreux problèmes NP (ou NP-complets) utilisés en

cryptographie est la factorisation de grands nombres entiers. Ce problème

est notamment utilisé par le premier cryptosystème à clé publique, RSA,

présenté à l'origine par Rivest, Shamir et Adlemanen.

On décrit RSA de la manière suivante :

Soient p et q deux grands nombres premiers distincts choisis aléatoirement (approx. 100 chiffres en décimal)

Notons

$$n = pq \qquad et \qquad \varphi(n) = (p-1)(q-1)$$

On choisit aléatoirement l'entier d>1 tel que

$$(d, \varphi(n)) = 1$$

et on calcule e, tel que

$$1 < e < \varphi(n)$$

satisfaisant

$$ed \equiv 1(mod\varphi(n)$$

Pour chiffrer un message m, on l'élève à la puissance e et on le

réduit modulo n. Pour déchiffrer un message chiffré c, on l'élève

à la puissance d et on le réduit modulo n.

On a donc,

– pour le chiffrement :

$$c \equiv m^e(modn)$$

– et pour le déchiffrement :

$$m \equiv c^d(modn)$$

$$c^d \equiv (w^e)^d \equiv w(modn)$$

La clé publique est constituée du couple (e,n) et la clé secrète est l'entier d.

A partir de la clé publique, il est très difficile de retrouver d. En effet, puisqu'il est défini tel que

$$ed \equiv 1(mod\varphi(n))$$

on a besoin de $\varphi(n)$ qui lui-même est très difficile à calculer en général, puisqu'il nécessité la factorisation de n qui, on le sait, est un problème NP-complet.

Il existe de nombreux autres problèmes NP-complets se basant notamment sur les graphes (CLIQUE, 3-COL ...), les grammaires formelles ou bien la combinatoire, avec le problème du sac à dos.

Chapitre 2

Le problème du sac à dos

2.1 Présentation du problème

Dans les chapitres suivants, nous allons étudier un cryptosystème basé sur le problème du sac à dos.

Le problème du sac à dos est l'un des problèmes NP-complets utilisés en cryptographie. Ce problème de combinatoire, malgré la simplicité de sa description, est complexe à résoudre.

Intuitivement, on peut formuler le problème du sac à dos de la manière suivante : disposant de plusieurs objets, comment remplir au mieux le sac à dos ?
"Au mieux" peut dépendre de plusieurs critères ou objectifs tels que minimiser le volume restant, maximiser ou minimiser le total des valeurs associées aux différents objects, etc.

On peu alors trouver le problème du sac à dos formulé de différentes façons et selon plusieurs dimensions :
– 2 dimensions : recouvrement d'une surface par des tétraminos (formes tétris)
– 3 dimensions : remplissage d'un volume par différents objets comme, par exemple, remplir des containers.

Le problème peut-être généralisé :
On numérote les objets par i
quad $1 \leq i \leq n$
On note a_i la valeur associée à l'objet i
et C la capacité du sac à dos.

Il peut exister de nombreuses manières de représenter le remplissage d'un sac.
On peut utiliser :

$$x_i = \begin{cases} 0 & \text{si l'objet i n'est pas pris} \\ p & \text{s'il est pris } n \text{ fois} \quad (n > 0) \end{cases}$$

On décrit alors la façon de remplir le sac par le vecteur $X = (x_1, \ldots, x_n)$

Le problème est alors le suivant :

On cherche un vecteur $X = (x_1, \ldots, x_n) \in 0, p^n$ vérifiant :

$$\sum_{1 \leq i \leq n} x_i a_i \ \leq C$$

2.2 Description du sac utilisé

Le problème utilisé par le schema de Merkle et Hellman est le suivant :
On se donne n entiers positifs $a_1, \ldots, a_n$
ainsi qu'un entier c.
Existe-t-il un choix judicieux parmi les a_i pour que leur somme vaille exactement c ?
ou

$$\exists ? \ (x_1, \ldots, x_n) \in 0, 1^n \quad \text{tel que} \quad \sum_{1 \leq i \leq n} x_i a_i = c$$

Ce problème est un problème de type sac à dos de dimension 1 qui peut être intuitivement modélisé par la recherche d'un empilement d'objets de hauteur a_i, afin d'atteindre une hauteur exacte c.

Exemple :
Considérons le 10-uplet :

$$A = (43,\ 129,\ 215,\ 345,\ 903,\ 302,\ 561,\ 1078,\ 697,\ 1525)$$

et le nombre 2887
On note alors que

$$2887 = 345 + 1078 + 561 + 903$$

Nous avons donc une solution pour $c = 2887$ et le 10-uplet A.

Ce problème de dimension 1 peut paraître simple à résoudre. Dans sa forme générale, illustrée ci-dessus, il demande de nombreux calculs.

Remarque. Une forme particulière du problème permet une résolution rapide.
Pour cela, il faut imposer une condition aux a_i : ils doivent former une suite supercroissante, c'est-à-dire :

$$\forall i \quad 1 \leq i \leq n \qquad a_i > \sum_{1 \leq j \leq i-1} a_j$$

Cette forme particulière des a_i garantit une solution unique au problème ainsi qu'une résolution rapide (polynomiale).

2.3 Algorithmes et complexité

Pour résoudre les problèmes de type sac à dos, on dispose de différents algorithmes. Néanmoins, ce problème appartenant à la classe NP, on ne connait pas d'algorithme efficace (polynomial en n) pour le résoudre dans un cas général et il est très improbable qu'un tel algorithme existe.

2.3.1 Algorithme exaustif

Un premier algorithme simple est possible ; il s'agit de calculer et d'essayer toutes les possibilités. Cette approche algorithmique est appelée recherche exaustive. Elle peut être, par exemple réalisée par un graphe d'exploration binaire.
Le principal inconvénient de cette approche est que, si n est suffisament grand (de l'ordre de 50 ou 100), le calcul devient totalement infaisable. En effet, le nombre de combinaisons possibles des x_i est de l'ordre de 2^n (soit près de 10^{15} combinaisons).

2.3.2 Algorithme glouton

On peut notemment utiliser un algorithme de type glouton pour résoudre le problème.

Algorithme 1
Entrée :
– *une liste d'objets* $(a_1, \ldots, a_n)$
– *une capacité* C
Sortie :
– *un vecteur* $(x_1, \ldots, x_n) \in 0, 1^n$

Début

Trier les objets $(a_i)_{1 \leq i \leq n}$ *par ordre décroissant de leur valeur associée*
C_{temp} = *0 Pour* $i = 1$ *à* n *faire*
  si $a_i + C_{temp} \leq C$ *alors*
    $x_i = 1$
  sinon
    $x_i = 0$
  Fin si
Fin pour
Retourner $(x_1, \ldots, x_n)$
Fin

Cet algorithme de complexité polynomiale ne donne pas, dans le cas général, de solution exacte.
Néanmoins, il est très efficace à condition que les a_i forment une suite supercroissante.

2.3.3 Algorithme de type programmation dynamique

Algorithme 2
Entrée :
– *une liste d'objets* $(a_1, \ldots, a_n)$
– *une capacité* C
Sortie :
– *un ensemble des objets* x_i *qui permmettent d'atteindre* C

Début

Pour $t = 1$ *à* C *faire*
  $table[0, t] = \varnothing$
Fin pour
Pour $i = 1$ *à* n *faire*
  Pour $t = 0$ *à* C *faire*
    Si $\sum_{a \in table[i,t]} a \geq a_i$ *alors*
      Si $\sum_{a \in table[i-1,t]} a \leq \sum_{a \in table[i-1,t-a_i]} a + x_i$ *alors*
        $table[i, t] = table[i - 1, t]$
      Sinon
        $table[i, t] = table[i - 1, t - a_i] \cup x_i$
      Fin si

Sinon
$table[i,t] = table[i-1,t]$
Fin si
Fin pour
Fin pour
Retourner $table[n,C]$
Fin

Cet algorithme est de complexité en temps et en espace $O(nC)$. Il est donc rapide à éxécuter.
Néanmoins, son gros inconvénient est l'espace mémoire qu'il requiert. Il est donc inutilisable pour des valeurs de n et C élevées.

En général, il n'y a pas d'algorithme efficace pour la résolution des problèmes de type sac à dos.
On constate cependant que, pour le cas des suites supercroissantes, il est très aisé de résoudre ce problème.
On pourrait donc être tenté d'utiliser cela pour créer un cryptosysteme.

Chapitre 3

Notions de géométrie des nombres

3.1 Définitions

Par la suite, diverses notions de géométrie des nombres sont nécessaires pour la compréhension. La notion de réseau euclidien est étroitement lié à la géométrie des nombres.

Naïvement, un réseaux euclidien peut être représenté par l'ensemble des points d'une grille ou des sommets d'un cristal.

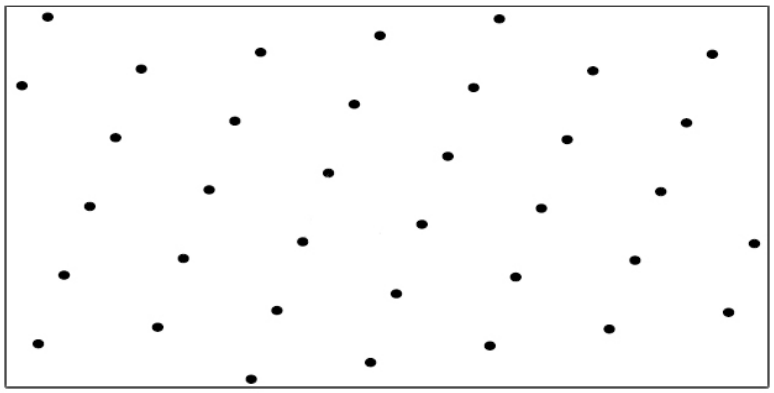

FIG. 3.1 – Un exemple de réseau

Définition 1
Soient $\mathbf{b}_1,\ldots,\mathbf{b}_n \in \mathbb{R}$ *des vecteurs indépendants. Alors l'ensemble*

$$L = \{x | x = \mathbf{u}_1\mathbf{b}_1 + \cdots + \mathbf{u}_n\mathbf{b}_n\} \qquad (\mathbf{u}_1, \ldots, \mathbf{u}_n \; entiers)$$

est appelé un réseau.
On dit que les vecteurs $\mathbf{b}_1,\ldots,\mathbf{b}_n$ *forment une base de ce réseau.*

Théorème 1
Un réseau est un sous-groupe discret de $\mathbb{R}^n$

Preuve. Soit L un réseau de base $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$
Si x et y sont de la forme

$$x = \mathbf{u}_1\mathbf{b}_1 + \cdots + \mathbf{u}_n\mathbf{b}_n$$

$$y = \mathbf{v}_1\mathbf{b}_1 + \cdots + \mathbf{v}_n\mathbf{b}_n$$

alors $x - y$ est de la même forme, d'où $x - y \in L$
Donc L est un groupe addictif.
Considérons le cube W donné par $|\mathbf{x}_i| \leq 1$ $(i = 1, \ldots, n)$
Le réseau L n'a aucun point $\neq 0$ dans LW.
Plus généralement, soit $\mathbf{x}_0$ un point de L, alors L n'a aucun point $x \neq \mathbf{x}_0$ dans $\mathbf{x}_0 + LW$
Maintenant, LW contient une sphère $|x| \leq p$
d'où la distance séparant deux points distincts de L est $\geq p$.
Ceci prouve que L est un ensemble discret.□

Définition 2
La forme quadratique définie positive naturelle associée à la base $\mathbf{b}_1, \ldots, \mathbf{b}_n$ *est :*

$$q(\mathbf{x}_1, \ldots, \mathbf{x}_d) = \|\mathbf{x}_1\mathbf{b}_1 + \cdots + \mathbf{x}_d\mathbf{b}_d\|^2$$

Proposition 1
Soit L *un réseau de* $\mathbb{R}^n$ *alors* L *admet une base,*
c'est-à-dire qu'il existe des vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{R}^n$ *linéairement indépendants tels que* $L = (\mathbf{b}_1, \ldots, \mathbf{b}_d)$.
La cardinalité d *de toutes les bases est identique et s'appelle la dimension (ou le rang) de* L. *On la notera dim* L.

Remarque 1
A un réseau sont donc associés deux dimensions :
– sa propre dimension $d = dim\ L$
– n sa dimension de prolongement
(dimension de son espace de prolongement $\mathbb{R}^n$)
On note que le réseau L peut être transformé en un réseau L' de dimension de prolongement d par isométrie.

On appelle réseaux totaux les réseaux dont la dimension d est maximale (c'est-à-dire $d = n$).

Théorème 2
Soit L un réseau de base $A = \{\mathbf{a}_1, \ldots, \mathbf{a}_n\}$
et soient $\mathbf{b}_1, \ldots, \mathbf{b}_n$ des points de L.
Alors $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ est une base de $L \Leftrightarrow$ il existe une matrice entière U telle que :

$$det\, U = \pm 1 \quad et \quad B = AU$$

Preuve.
$\Rightarrow$ Tout d'abord, on suppose B, une base de L.
Alors $L = AY$ et aussi $L = BY$.
D'où si $A^{-1}B = U$, on a alors $UY = U$ et $U^{-1}Y = Y$.

$$\begin{array}{cc} AY = BY & AY = BY \\ \text{or } B = AU & \text{or } BU^{-1} \\ AY = AUY & BU^{-1}Y = BY \\ Y = UY & U^{-1}Y = Y \end{array}$$

ça implique que U et U^{-1} sont toutes les deux des matrices entières. Alors nécessairement, $det\, U = \pm 1$

$\Leftarrow$ Dans l'autre sens, supposons qu'il existe une matrice U telle que $det\, U = \pm 1$ et $B = AU$. Alors $UY = Y$, puisque U et U^{-1} sont des matrices entières
D'où $BY = AUY = AY = L$ et donc B est une base de L. $\square$

Une conséquence du théorème est que, si B est une base de L, la quantité $|det\, B|$ ne dépend pas du choix de la base B.

Définition 3
Le déterminant $\Delta(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ de la matrice de Gram $(< \mathbf{b_i}, \mathbf{b_j} >)_{1 \leq i,j \leq d}$ est un réel ≥ 0 indépendant de la base B : on l'appelle le discriminant de L, noté $\Delta(L)$.

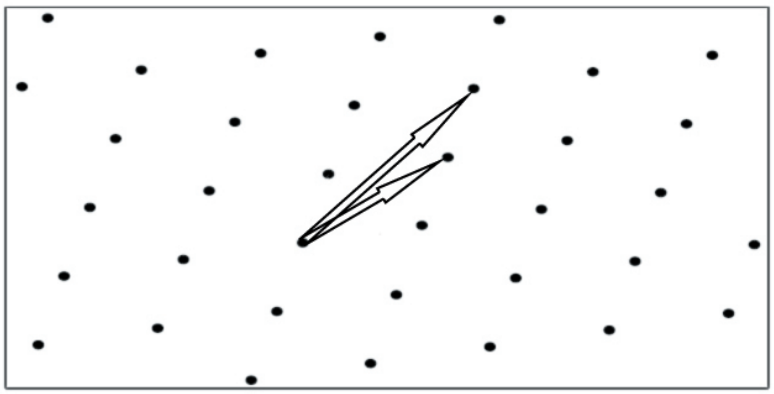

FIG. 3.2 – Une base quelconque

Remarque 2
Si B est une base de L, $\Delta(L) = |{}^tBB|$ car
$Gram(< \mathbf{b_i}, \mathbf{b_j} >)_{1 \leq i,j \leq d} = {}^tBB$.

Définition 4
Soit L un réseau et $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ une base de L.
Le volume ou déterminant du réseau L est défini par :

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

Cette quantité ne dépend pas de la base $\mathbf{b}_1, \ldots, \mathbf{b}_d$.

Proposition 2
Soient les propriétés suivantes :
– *Si $d = n$, alors vol $L = |det(\mathbf{b}_1, \ldots, \mathbf{b}_d)|$*
– *vol $L = \sqrt{\Delta(L)}$*

Remarque 3
Le déterminant n'est autre que le volume au sens de Lebesgue (sens usuel) des "briques élémentaires" du réseau, c'est-à-dire les ensembles de la forme :

$$P_b = \left\{ \sum_{1 \leq i \leq n} x_i b_i | (x_1, \ldots, x_n) \in [0, 1[^n \right\}$$

où les b_i sont une base.

De plus, tout point de $\mathbb{R}^n$ est congru modulo L à un point et un seul de P_b : on dit que P_b est un domaine fondamental pour L.

3.2 Théorème de Minowski

On note μ la mesure de Lebesgue dans $\mathbb{R}^n$. Ainsi, pour toute partie intégrable S de $\mathbb{R}^n$, on notera $\mu(S)$ sa mesure (ou son volume).

Théorème 3 (Minowski)
Soient L un réseau de $\mathbb{R}^n$ et S un sous-ensemble intégrable de $\mathbb{R}^n$ tels que $\mu(S) > vol\ L$.
Il existe alors deux éléments $x, y \in S$ distincts tels que $x - y \in L$.

Preuve. Soient $b = (b_1, \ldots, b_n)$ une base de L et P_b le parallélotope semi-ouvert construit sur b.
Comme P_b est un domaine fondamental pour L, S est la réunion disjointe des $S \cap (h + P_b)$ où $h \in H$
on a donc

$$\mu(S) = \sum_{h \in H} \mu(S \cap (h + P_b))$$

Puisque μ est invariant par translation, on obtient

$$\mu(S \cap (h + P_b)) = \mu((-h + S) \cap P_b)$$

Or les ensemblesne peuvent pas être disjints, car sinon, on aurait :

$$\mu(P_b) \geq \sum_{h \in H} \mu((-h + S) \cap P_b) = \sum_{h \in H} \mu(S \cap (h + P_b))$$

$$\mu(P_b) \geq \mu(S)$$

or par hypothèse, $\mu(S) > vol\ L = \mu(P_b)$.

Il existe donc deux éléments $p, p' \in L$ $(p \neq p')$ tels que

$$P_b \cap (-p + S) \cap (-p' + S) \neg \varnothing$$

On peut donc trouver des élements $x, y \in S$ tels que

$$-p + x = -p' + y$$

puisque $p, p' \in L$ et que L est un sous-groupe addictif, $p - p' \in L$
donc $x - y \in L$ et $x \neg y$ (car $p \neg p'$). $\square$

Corollaire 1
Soient L un réseau de $\mathbb{R}^n$ et S une partie intégrable symétrique par rapport à 0 et convexe de $\mathbb{R}^n$.
Si $\mu(S) > 2^n vol\ L$
alors $S \cap L$ contient un point autre que 0.

Preuve. On applique le théorème à $S' = \frac{1}{2}S$
On a

$$\mu(S') = \frac{1}{2^n}\mu(S) > vol\ L$$

Il existe donc deux points $x, y \in S'$ tels que $y - z \in L$
alors

$$x = y - z = \frac{1}{2}(2y + (-2z))$$

est un point de S (car S symétrique et convexe). Donc $x \in S \cap L$. □

3.3 Réseaux orthogonaux

On note $E(L)$ le sous-espace engendré par L dans $\mathbb{R}^n$.

Définition 5
On note $L^\perp$ le réseau orthogonal de L. Il est défini tel que :

$$L^\perp = E(L)^\perp \cap \mathbb{Z}^n = \{x \in \mathbb{Z}^n \mid \forall y \in L, < x, y >= 0\}$$

On remarque facilement que $L^\perp$ est un sous-groupe de $(Z)^n$. Donc $L^\perp$ est un réseau.
De plus, $dim\ L^\perp = n - dim\ L$.

Définitions 6
On dit que L est un réseau complété si :

$$\overline{L} = (L^\perp)^\perp$$

On dit que L est un réseau complet si :

$$\overline{L} = L$$

Définition 7
On définit L^* le réseau dual de L :

$$L^* = \{x \in \mathbb{Z}^n \mid \forall y \in L\ :\ x.y \in \mathbb{Z}\}$$

Théorème 4
Soit L un réseau complet dans $\mathbb{Z}^n$, alors :

$$det(L^\perp) = det(L)$$

Preuve. On a par définition $L = E(L) \cap \mathbb{Z}^n$ et $L^\perp = E^\perp(L) \cap \mathbb{Z}^n$
D'après Martinet [14],

$$det(L) = \frac{det(L \cap F}{L^* \cap F^\perp} \qquad \text{avec } L = \mathbb{Z}^n \text{ et } F = E(L)$$

on a donc

$$det(\mathbb{Z}^n) = \frac{det(\mathbb{Z}^n \cap E(L))}{det((\mathbb{Z}^n)^* \cap E^\perp(L))}$$

or $(\mathbb{Z}^n)^* = \mathbb{Z}^n$
on a donc

$$det(\mathbb{Z}^n) = \frac{det(\mathbb{Z}^n \cap E(L))}{det(\mathbb{Z}^n \cap E^\perp(L))} = \frac{det(L)}{det(L^\perp)}$$

or $det(\mathbb{Z}^n = 1)$ on a donc pour finir $det(L) = det(L^\perp)$. □

Corollaire 2
Soit L un réseau de $\mathbb{Z}^n$, alors :

$$det((L^\perp)^\perp) = det(L^\perp) = det(\overline{L})$$

3.4 Les bases LLL-réduites

Définition 8 (Orthogolanisation de Gramm-Schmidt)
Etant donnée une base $(b_1, \ldots, b_d)$, on notera $(b_1^* b_d^*)$ la base obtenue par le procédé d'orthogonalisation de Gramm-Schmidt :

$$b_i^* = \begin{cases} b_1 & \text{si } i\text{=}1, \\ b_i - \sum_{1 \le j \le i-1} \mu_{i,j} b_j^* & \text{si } 2 \le i \le d. \end{cases}$$

$$où \quad \mu_{i,j} = \frac{< b_i, b_j^* >}{\left\| b_j^* \right\|^2} \quad \forall j < i$$

Définition 9
On dit qu'une base $(b_1, \ldots, b_d)$ est faiblement réduite si son orthogonalisation de Gramm-Schmidt vérifie :

$$\forall\, 1 \le j < i \le d \qquad |\mu_{i,j}| \le \frac{1}{2}$$

Définition 10
On dit qu'une base $(b_1, \ldots, b_d)$ est LLL-réduite pour le facteur δ si elle est faiblement réduite et si elle satisfait les conditions de Lovász :

$$\forall\, i \quad 2 \le i \le d \qquad \left\| b_i^* + \mu_{i,i-1} b_{i-1}^* \right\| > \delta \left\| b_{i-1}^* \right\|$$

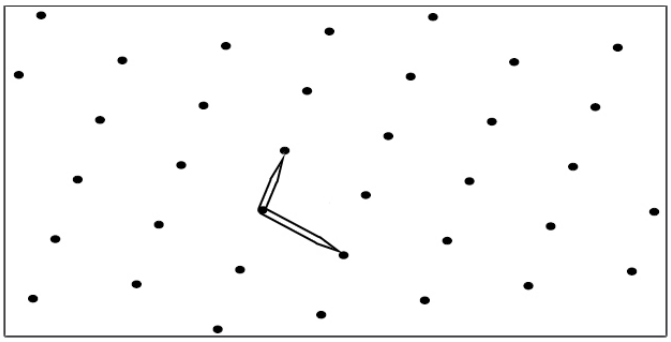

FIG. 3.3 – Une base réduite

3.5 Algorithmes

3.5.1 Algorithme Gram-Schmidt

Algorithme 1 (Gram-Schmidt)
Entrée :
– *une base* $(b_1, \ldots, b_d)$
Sortie :
– *une base* $(b_1^*, \ldots, b_d^*)$
– *une matrice* $M = (\mu_{i,j})$
Début

Pour $i = 1$ *à* d *faire*
  $j = 1$
  $b_i^* = b_i$
  Pour $j = 1$ *à* $i - 1$ *faire*
    $\mu_{i,j} = \frac{<b_i|b_j^*>}{\|b_j\|^2}$
    $b_i^* = b_i^* - \mu_{i,j} b_j^*$
  Fin pour
Fin pour
Retourner $(b_1^*, \ldots, b_d^*)$
Fin

Pour construire la base $(b_1^*, \ldots, b_d^*)$, l'algorithme opère séquentiellement. Tout d'abord, il prend b_1 comme premier vecteur, puis, comme second vecteur, le projeté de b_2 orthogonalement à b_1, etc.

On a alors, pour tout i, b_i^* est le projeté de b_i orthogonalement à l'espace qu'engendre $(b_1, \ldots, b_i - 1)$.

3.5.2 Algorithme de réduction faible

Algorithme 2 (Réduction faible)
Entrée :
– *une base* $(b_1, \ldots, b_d)$
Sortie :
– *une base* $(b_1, \ldots, b_d)$ *faiblement réduite*
Début

Calculer Gram-Schmidt Pour $i = 2$ *à* n *faire*
  Pour $j = i - 1$ *à* 1 *faire*
    $b_i = b_i - \lceil \mu_{i,j} \rfloor \, b_j$
    Pour $k = 1$ *à* j *faire*
      $\mu_{i,k=\mu_{i,k}} - \lceil \mu_{i,k} \rfloor \, \mu_{j,k}$
    Fin pour
  Fin pour
Fin pour
Retourner $(b_1, \ldots, b_d)$
Fin

3.5.3 Algorithme LLL

Algorithme 1 (LLL)
Entrée :
– *une base* $(b_1, \ldots, b_d)$
Sortie :
– *une base* $(b_1, \ldots, b_d)$ *LLL-réduite à un facteur* δ
Début

Réduire faiblement $(b_1, \ldots, b_d)$
S'il existe un j *qui ne satisfait pas la condition de Lovász*
  Alors échanger b_j *et* b_{j+1} *et réappliquer LLL*
Retourner $(b_1, \ldots, b_d)$
Fin

Cet algorithme n'est pas optimal.

Pour des version optimisées de l'algorithme, on obtient une complexité $O(nd^5m^3)$ où

m : nombre max de bits des coordonnées des b_i

n : dimension de l'espace

d : dimension du réseau

En pratique, on peut prendre $\delta = 0.99$ car pour $\delta = 1$ on ne sait pas si l'algorithme reste polynomial.

Chapitre 4

Le cryptosystème de Merkle-Hellman

Le cryptosystème de Merkle-Hellman a été inventé par Ralph Merkle et Martin Hellman en 1978 [5]. C'est un cryptosystème facile à implémenter car il est basé sur le problème du sac à dos.

4.1 Le problème du sac à dos

Le problème du sac à dos est connu pour être NP-Complet. Cependant, ce problème est facile à résoudre lorsque les éléments du sac à dos $(a_1, \ldots, a_n)$ forment une suite super croissante :

Définition 1
La suite $(u_n)_{n\in\mathbf{N}} \in \mathbf{R}^n$ *est* super croissante *lorsque :*

$$\forall n \in \mathbf{N}, u_{n+1} > \sum_{i=0}^{n} u_i$$

Dans ce cas, la résolution du problème est triviale et il peut être résolu de la manière suivante :

Algorithme 1
Entrée :
– *un sac à dos* $(a_1, \ldots, a_n)$
– *une valeur cible* c
Sortie :
– *le message en clair* $(m_1, \ldots, m_n) \in \{0, 1\}^n$

Début

Pour i *de* n *à* 1 *faire*
  Si $c > a_i$ *alors* $c \leftarrow c - a_i$.
  $x_i \leftarrow 1$
Fin pour
Retourner $(m_1, \ldots, m_n)$
Fin

4.2 Description du cryptosystème

Comme tout cryptosystème à clé publique, le cryptosystème de Merkle-Hellman est formé :
– d'une clé publique : une suite d'entiers positifs $(b_1, \ldots, b_n)$
– d'une clé privée : le sac à dos $(a_1, \ldots, a_n)$ super croissant et deux entiers m et w.
L'espace des messages de longueur n est $\{0, 1\}^n$.

4.2.1 Génération des clés privée et publique

1. On choisit un entier positif n suffisament grand (Merkle et Hellman recommandent de prendre $n \simeq 100$)
2. On choisit une suite d'entiers positifs $(a_1, \ldots, a_n)$ qui soit super-croissante.
3. On choisit $m \in \mathbf{N}$ tel que $m > \sum_{i=1}^{n} a_i$
4. On choisit $w \in [1, m-1]$ premier avec m.
5. On calcule $b_i = a_i.w \mod m$ pour $i \in \{1, 2, \ldots, n\}$

La clé publique est $(b_1, \ldots, b_n)$, elle peut être diffusée librement. La clé privée est composée de m, w et du sac à dos $(a_1, \ldots, a_n)$. Comme toute clé privée, elle doit être tenue secrète afin de protéger le cryptosystème.
Exemple réalisé à l'aide de la classe MH.java fournie en Annexe B :
On va chiffrer un message de longueur 6, on a donc besoin de deux sac à dos de cette longueur :
– On choisit les 6 a_i formant le sac à dos facile A :

$$A = (93, 660, 1479, 3218, 6602, 13594)$$

– On choisit $m = 25922 > \sum_{i=1}^{n} a_i = 25646$
– On choisit $w = 10693$ premier avec m et $w < m$

– On calcule $b_i = a_i.w \mod m$ pour $i \in \{1, 2, \ldots, n\}$:

$$B = (9413, 6596, 2527, 11580, 9580, 15988)$$

– La clé publique est $(9413, 6596, 2527, 11580, 9580, 15988)$.
– La clé privée est $(25922, 10693, (93, 660, 1479, 3218, 6602, 13594))$.

4.2.2 Chiffrement et déchiffrement

Chiffrement

Soit $M = (m_1, \ldots, m_n)$ le message en clair.
Le message est chiffré à l'aide de la clé publique en calculant $c = \sum_{i=1}^{n} m_i b_i$.

Suite de l'exemple :
Le message à chiffrer est 51, ce qui donne en binaire 110011 d'où
$M = (1, 1, 0, 0, 1, 1)$.
On chiffre M : $c = \sum_{i=1}^{n} b_i.m_i = 41577$

Déchiffrement

Pour déchiffrer le message, on commence par calculer w^{-1}, l'inverse de w mod m. Ce calcul peut être fait grâce à l'algorithme d'Euclide étendu :

Algorithme 2

Entrée :
– $a, b \in \mathbf{N}$
Sortie :
– $t \in \mathbf{N}$ *tel que* $b.t \equiv 1 \mod n$

Début
$s \leftarrow 0$
$t \leftarrow 1$
$q \leftarrow \lfloor \frac{a}{b} \rfloor$
$r \leftarrow a \mod b$
Tant que $r > 0$ *faire*
$tmp \leftarrow s - q.t$
Si $tmp \geqslant 0$ *alors* $tmp \leftarrow tmp \ mod \ n$ *sinon* $tmp \leftarrow n - ((-tmp) \ mod \ n)$
$s \leftarrow t$
$t \leftarrow tmp$
$n \leftarrow b$
$b \leftarrow r$
$q \leftarrow \lfloor \frac{a}{b} \rfloor$
$r \leftarrow n - q.b$
Fin tant que
Si $b \neq 1$ *alors b n'a pas d'inverse modulo n sinon* $b^{-1} mod n = t$
Fin

Une fois qu'on a obtenu w^{-1}, on calcule $c.w^{-1} \equiv \sum_{i=1}^{n} x_i.b_i.w^{-1} \equiv x_i.a_i(\mod m)$.
Le sac à dos $(a_1, \ldots, a_n)$ étant super-croissant on trouve facilement la solution grâce à l'algorithme décrit dans le paragraphe 4.1 page 26.

Suite et fin de l'exemple :
– $w^{-1} = -5549 = 20373 \mod 25922$
– $d = w^{-1}.c = 20373.41577 = 20949 \mod 25922$
– On résoud le problème du sac à dos (A, d) : $20949 = 13594 + 6602 + 660 + 93$. On retrouve alors le message initial $(1, 1, 0, 0, 1, 1)$.

Afin de renforcer la sécurité du système, il est possible d'appliquer successivement plusieurs cryptosystèmes de Merkle-Hellman . Malheureusement, ce cryptosystème fut rapidement cassé. Dès 1982, Shamir proposa une attaque contre un système composé d'une seule transformation de Merkle-Hellman . Cette méthode ne cassait cependant pas l'application successives de plusieurs transformations et ne put être expérimentée car l'attaque de Shamir reposait sur un algorithme de Lenstra [11] que personne n'avait implémenté.

En 1983, Adleman [12] démontra que l'algorithme de réduction de réseaux LLL pouvait être utilisé à la place de l'algorithme de Lenstra, facilitant ainsi les expérimentations. Brickell [9, 10] montra plus tard comment casser

des cryptosystèmes composés de plusieurs itérations. C'en était fini du cryptosystème inventé par Merkle et Hellman.

4.3 Cryptanalyse du cryptosystème

L'attaque de ce cryptosystème est basée sur l'algorithme LLL de réduction de réseau proposé par Lenstra, Lenstra et Lovász. Etant donné une clé publique $B = (b_1, \ldots, b_n)$ et un chiffré c, l'application de l'algorithme LLL à un certain réseau permet de retrouver le message en clair $M = (m_1, \ldots, m_n)$. Cependant, cette attaque ne fonctionne que sur un sac à dos de faible densité :

Définition 2
La densité d'un sac à dos $(a_1, \ldots, a_n)$ *est égale à*

$$d = \frac{n}{\max_{i=1,\ldots,n} log_2 a_i}$$

Elle correspond au nombre de poids divisé par la taille maximale des poids.

Si $d \leqslant 1$, alors la solution du sac à dos est unique. De plus, si $d \leqslant 0.94$, le problème du sac à dos peut être résolu en utilisant la réduction de réseau. En effet, le problème du sac à dos peut être ramené à un problème de recherche de vecteur court dans un réseau.

Bien que la recherche de vecteurs courts dans un réseau puisse être très difficile en général, les algorithmes connus tels que LLL s'avèrent en réalité bien plus performants que ne l'affirme la théorie. Il est donc probable que la recherche de vecteurs courts soit facile, même si dans le cas général elle est difficile, c'est pourquoi il faut séparer l'efficacité de la recherche du vecteur court et la qualité de la transformation du sac à dos en recherche d'un vecteur court.

Dans ce TER, nous nous intéresserons uniquement aux réseaux pouvant être utilisés et laisserons de côté l'étude de l'efficacité de la recherche en fonction de l'algorithme utilisé. Nous utiliserons uniquement l'algorithme LLL. Etudions maintenant les différents réseaux utilisables :

4.3.1 L'attaque de Lagarias-Odlyzko

Soit $M = (m_1, \ldots, m_n)$ le message en clair correspondant au message chiffré c. L'idée de Lagarias et Odlyzko [4] est de chercher les m_i les plus petits possibles tels que $\sum_{i=1}^{n} m_i b_i - s = 0$. On cherche donc un vecteur court dans le réseau engendré par $b_1, \ldots, b_n, -s$:

$$G = \begin{pmatrix} 1 & 0 & \ldots & 0 & Nb_1 \\ 0 & 1 & \ldots & 0 & Nb_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & Nb_n \\ 0 & 0 & \ldots & 0 & -Nc \end{pmatrix}$$

où N est un entier supérieur à $\sqrt{n}$.
Une solution du problème est associée à un vecteur court du réseau. Ce vecteur ligne g_i est de la forme $(m_1, m_2, \ldots, m_n, 0)$. Si LLL est capable de le trouver, il sera alors possible d'en déduire les m_i et donc de retrouver c et de casser le cryptosystème :

$$c = \begin{pmatrix} m_1 & m_2 & \ldots & m_n \end{pmatrix} \begin{pmatrix} g_{i,1} \\ g_{i,2} \\ \vdots \\ g_{i,n} \end{pmatrix} = m_1 g_{i,1} + m_2 g_{i,2} + \ldots + m_n g_{i,n}$$

Définissons les vecteurs g_i, $i = 1, \ldots, n$ où g_i est la $i^{ème}$ ligne de G et appliquons l'algorithme LLL à $\{g_i\}_{i=1\ldots(n+1)}$. Celui-ci renvoie une nouvelle base $\{\tilde{g}_i\}_{i=1\ldots(n+1)}$. Pour résoudre le sac à dos, on cherche dans cette base un vecteur $\tilde{g}_{i_0}$ tel que :

$$\tilde{g}_{i_0,j} \in \{0, 1\}, j = 1 \ldots n \tag{4.1}$$

$$\tilde{g}_{i_0,n+1} = 0 \tag{4.2}$$

Si ce vecteur n'existe pas, le problème (B, c) n'a pas de solution ou l'algorithme LLL n'a pas trouvé une bonne base réduite sinon $(m_1, \ldots, m_n) = (\tilde{g}_{i_0,1}, \ldots, \tilde{g}_{i_0,n})$.
L'utilisation de ce réseau a pourtant montré ses limites car il ne permet de résoudre que des sacs à dos dont la densité $d < 0.6463$.

Afin d'illustrer cette attaque, nous allons supposer que Sophie dispose de la clé publique B et qu'elle ait intercepté le message chiffré c. Elle peut facilement retrouver le message en clair :

– Sophie doit choisir $N > \sqrt{n} \simeq 2.645$, elle choisit donc $N = 3$.

– Elle crée le réseau de Lagarias et Odlyzko :

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 28239 \\ 0 & 1 & 0 & 0 & 0 & 0 & 19788 \\ 0 & 0 & 1 & 0 & 0 & 0 & 7581 \\ 0 & 0 & 0 & 1 & 0 & 0 & 34740 \\ 0 & 0 & 0 & 0 & 1 & 0 & 28740 \\ 0 & 0 & 0 & 0 & 0 & 1 & 47964 \\ 0 & 0 & 0 & 0 & 0 & 0 & -124731 \end{pmatrix}$$

– Elle applique LLL à cette base du réseau et obtient une base LLL-réduite :

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & -3 & -1 & 2 & 0 & -3 \\ -2 & 0 & 3 & -4 & 0 & 1 & 0 \\ 1 & -5 & 0 & -1 & 2 & 1 & 3 \\ -6 & 4 & -2 & 0 & 2 & 1 & 0 \\ -2 & -3 & 3 & 6 & 4 & 3 & 0 \\ 2 & 2 & 3 & 1 & 2 & -7 & 0 \end{pmatrix}$$

– Elle constate que la première ligne du réseau répond aux critères (4.1) et (4.2), ses six premières coordonnées fournissent donc une solution au problème (B, c). Sophie a réussi à retrouver le message initial sans la clé privée.

4.3.2 Le réseau de Coster, La Macchia, Odlyzko et Schnorr

Pour augmenter la densité critique, il faut éviter que les m_i puissent valoir -1. On cherche donc non plus un vecteur court au sens de la norme usuelle c'est-à-dire dans une boule centrée en O et de rayon minimal, mais un vecteur dans une boule décalée ou dans une famille de boules décalées pour éviter d'inclure la coordonnée -1, ce qui revient à changer de métrique. C'est ce principe que Coster, La Macchia, Odlyzko et Schnorr ont expliqué dans [2, 1].

De ce changement résulte le réseau engendré par la matrice $(n+1) \times (n+1)$ suivante :

$$G' = \begin{pmatrix} 1 & 0 & \dots & 0 & Nb_1 \\ 0 & 1 & \dots & 0 & Nb_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ddots & 1 & Nb_n \\ \frac{1}{2} & \frac{1}{2} & \dots & \frac{1}{2} & Nc \end{pmatrix}$$

où N est un entier supérieur à $\frac{\sqrt{n}}{2}$ et définissons les vecteurs g'_i, $i = 1, \dots, n$ où g_i est la $i^{ème}$ ligne de G'. Soit L' le réseau engendré par $\{g'_i\}_{i=1\dots(n+1)}$. Regardons le vecteur

$$y = m_1 g'_1 + m_2 g'_2 + \dots + m_n g'_n - m_{n+1}$$

Pour $i = 1, \dots, n$, ou bien $m_i = 0$ ou $m_i = 1$ donc y est une combinaison linéaire des $\{g'_i\}_{i=1\dots n}$ et le réseau L' contient y.

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} m_1 + 0 + \dots + 0 - 0.5 \\ 0 + m_2 + 0 + \dots + 0 - 0.5 \\ \vdots \\ 0 + 0 + \dots + 0 - 0.5 \\ t(b_1 m_1 + b_2 m_2 + \dots + b_n m_n) - tc \end{pmatrix} = \begin{pmatrix} m_1 - 0.5 \\ m_2 - 0.5 \\ \vdots \\ m_n - 0.5 \\ 0 \end{pmatrix}$$

Avec $m_i \in \{0, 1\}$, y est un vecteur dont toutes les composantes appartiennent à $\{-0.5, 0.5\}$ sauf la dernière qui vaut 0. Il faut maintenant réduire la base $\{g'_i\}_{i=1\dots(n+1)}$ en utilisant l'algorithme LLL. Soit $\{\tilde{g}'_i\}_{i=1\dots(n+1)}$ la nouvelle base réduite. On cherche un vecteur $\tilde{g}'_i$ dans cette base tel que :

$$\tilde{g'}_{i_0,j} = \pm 0.5, j = 1 \dots n \tag{4.3}$$

$$\tilde{g'}_{i_0,n+1} = 0 \tag{4.4}$$

Supposons maintenant qu'il existe un vecteur $\tilde{g}'_k$ tel que $\tilde{g}'_k = (\pm 0.5, \pm 0.5, \dots, \pm 0.5, 0)$. Voyons comment on peut retrouver la solution du problème à partir de ce vecteur :

- On définit le vecteur $X = (x_1, x_2, \dots, x_n)$ avec $x_i = 0.5 + \tilde{g}'_{k,i}, i = 1, \dots, n$
- On calcule $s = XB = x_1 b_1 + \dots + x_n b_n$
- Si $s = c$, on a trouvé une solution à (B, c) et le texte en clair est le vecteur X.

Sinon :
On re-définit le vecteur $X = (x_1, x_2, \ldots, x_n)$ avec
$x_i = 0.5 - \tilde{g}'_{k,i}, i = 1, \ldots, n$
On re-calcule $s = XB = x_1 b_1 + \ldots + x_n b_n$
Si $s = c$ alors le texte en clair est le vecteur X.

Imaginons que Guillaume veuille également casser le cryptosystème avec le réseau précédent :

– Il prend $N = 3$ car $3 > \sqrt{n} > \frac{\sqrt{n}:}{2}$ et il définit le réseau de Coster, La Macchia, Odlyzko et Schnorr :

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 28239 \\ 0 & 1 & 0 & 0 & 0 & 0 & 19788 \\ 0 & 0 & 1 & 0 & 0 & 0 & 7581 \\ 0 & 0 & 0 & 1 & 0 & 0 & 34740 \\ 0 & 0 & 0 & 0 & 1 & 0 & 28740 \\ 0 & 0 & 0 & 0 & 0 & 1 & 47964 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 124731 \end{pmatrix}$$

– Il réduit cette base à l'aide de LLL. Son logiciel de calcul formel lui renvoie cette base :

$$\begin{pmatrix} -0.5 & -0.5 & 0.5 & 0.5 & -0.5 & -0.5 & 0 \\ -1 & -1 & -2 & 0 & 1 & -1 & -3 \\ -1.5 & 0.5 & 3.5 & -3.5 & 0.5 & 1.5 & 0 \\ 1 & -5 & 0 & -1 & 2 & 1 & 3 \\ -2 & 3 & 2 & 4 & 1 & 3 & 0 \\ -3.5 & 1.5 & -1.5 & -3.5 & -0.5 & -1.5 & 3 \\ 3 & 3 & 3 & 1 & 3 & -6 & 0 \end{pmatrix}$$

– Guillaume constate alors que la première ligne de la base LLL-réduite est la seule qui répond aux deux conditions (4.4) et (4.3).
– Il définit le vecteur X comme dans la première méthode décrite ci-dessus :

$$X = (0, 0, 1, 1, 0, 0)$$

– Il calcule

$$s = XB = (0, 0, 1, 1, 0, 0) \begin{pmatrix} 9413 \\ 6596 \\ 2527 \\ 11580 \\ 9580 \\ 15988 \end{pmatrix} = 2527 + 11580 = 14107 \neq 41577$$

Cela ne correspond pas au c qu'il cherche.

– Il définit alors X selon la deuxième méthode :

$$X = (1, 1, 0, 0, 1, 1)$$

– Il re-calcule

$$s = XB = (1, 1, 0, 0, 1, 1) \begin{pmatrix} 9413 \\ 6596 \\ 2527 \\ 11580 \\ 9580 \\ 15988 \end{pmatrix} = 9413{+}6596{+}9580{+}15988 = 41577 = c$$

Guillaume a lui aussi retrouvé le message en clair.

4.3.3 Le réseau de Joux et Stern

Joux et Stern [3, 1] se sont basées sur le même principe de changement de métrique et ont proposés ce réseau $(n+1) \times (n+2)$:

$$G'' = \begin{pmatrix} n+1 & -1 & \dots & -1 & -1 & Nb_1 \\ -1 & n+1 & \dots & -1 & -1 & Nb_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ -1 & -1 & \dots & n+1 & -1 & Nb_n \\ -1 & -1 & \dots & -1 & n+1 & -Nc \end{pmatrix}$$

où N est un entier positif grand ($N \geqslant n$).

Après application de LLL, on cherche un vecteur $\tilde{g}''_k$ correspondant aux critères suivants :

$$\tilde{g}''_{k,j} \in \{x, y\} \text{ avec } x, y \in \mathbf{Z}, x > 0, y < 0 \text{ et } x - y = n + 2, j = 1, \dots, n+1 \tag{4.5}$$

$$\tilde{g}''_{k,n+2} = 0 \tag{4.6}$$

Pour retrouver le message M à partir de ce vecteur, on effectue les copérations suivantes :
– On définit le vecteur $X = (x_1, x_2, \dots, x_n)$ avec $x_i = \tilde{g}''_{k,i} - y, i = 1, \dots, n$
– On divise chaque composante de X par $x - y$
– On calcule $s = XB = x_1 b_1 + \dots + x_n b_n$
– Si $s = c$, on a trouvé une solution à (B, c) et le texte en clair est le vecteur X.

Sinon :
On re-définit le vecteur $X = (x_1, x_2, \ldots, x_n)$ avec
$x_i = \tilde{g}''_{k,i} - x, i = 1, \ldots, n$
On re-calcule $s = XB = x_1 b_1 + \ldots + x_n b_n$
Si $s = c$ alors le texte en clair est le vecteur X.
Ces deux derniers réseaux permettent de résoudre des sacs à dos dont la densité $d < 0.9408$

Cette fois, c'est Charlie qui veut casser le cryptosystème.
– Il prend $N = 37 > 6$
– Il définit la base du réseau de Joux et Stern :

$$\begin{pmatrix} 7 & -1 & -1 & -1 & -1 & -1 & -1 & 348281 \\ -1 & 7 & -1 & -1 & -1 & -1 & -1 & 244052 \\ -1 & -1 & 7 & -1 & -1 & -1 & -1 & 93499 \\ -1 & -1 & -1 & 7 & -1 & -1 & -1 & 428460 \\ -1 & -1 & -1 & -1 & 7 & -1 & -1 & 354460 \\ -1 & -1 & -1 & -1 & -1 & 7 & -1 & 591556 \\ -1 & -1 & -1 & -1 & -1 & -1 & 7 & -1538349 \end{pmatrix}$$

– Il applique LLL à cette base et obtient cette base LLL-réduite :

$$\begin{pmatrix} 3 & 3 & -5 & -5 & 3 & 3 & 3 & 0 \\ -4 & -4 & -12 & 4 & 12 & -4 & -4 & -37 \\ -13 & 3 & 27 & -29 & 3 & 11 & -5 & 0 \\ -24 & 16 & 0 & 16 & 0 & 16 & 24 & 0 \\ 3 & -45 & -5 & 3 & 27 & 3 & -5 & 0 \\ -26 & 14 & -10 & -10 & 14 & -10 & -26 & 0 \\ -4 & 36 & 12 & 12 & 20 & -36 & 20 & 0 \end{pmatrix}$$

– Seul le premier vecteur ligne répond aux conditions (4.5) et (4.6).
Guillaume pose alors $x = 3$ et $y = -5$
– Charlie définit X comme indiqué au-dessus :

$$(8, 8, 0, 0, 8, 8)$$

– Il divise toutes les coordonnées de X par $x - y = 8$. X devient
$(1, 1, 0, 0, 1, 1)$
– Il calcule alors

$$s = XB = (1, 1, 0, 0, 1, 1) \begin{pmatrix} 9413 \\ 6596 \\ 2527 \\ 11580 \\ 9580 \\ 15988 \end{pmatrix} = 9413+6596+9580+15988 = 41577 = c$$

Charlie vient lui aussi de déchiffrer le message c.
Trois personnes viennent donc de décrypter notre message en utilisant trois réseaux différents. Le cryptosystème de Merkle-Hellman est donc obsolète. Nous allons maintenant comparer l'efficacité des trois réseaux en fonction de la densité du sac à dos et de sa longueur.

4.4 Expérimentations

Pour chaque type réseau, on effectue 80 réductions de réseaux générés aléatoirement, à partir de sacs à dos de longueur et de densité variables.

4.4.1 Le réseau de Lagarias-Odlyzko

D'après les figures A.1 à A.4, on constate que le taux de réussite chute aux environs d'une densité de 0,63 mais uniquement lorsque le message et assez long. On note également que le message de longueur 50 est le plus difficile à cryptanalyser et cela pour n'importe quelle densité. En moyenne, le message en clair a été retrouvé 1 fois sur 2 seulement.

4.4.2 Le réseau de Coster, La Macchia, Odlyzko et Schnorr

Sur les figures A.5 à A.8, on observe qu'on obtient toujours un résultat d'au moins 80%. Ces résultats sont largement meilleurs que ceux du précédent réseau. Cela est dû au fait qu'il permet de résoudre des sac à dos de densité inférieure à 0,94. On obtient toutefois de très bons résultats pour les sacs à dos denses, ceci s'expliquant par la longueur des messages relativement courte. Des tests moins importants sur des mesages de longueur 100 ont montré que le message en clair était retrouvé moins d'une fois sur 3 pour des sacs à dos denses.

4.4.3 Le réseau de Joux et Stern

Les figures A.9 à A.12 indiquent que ce réseau permet d'obtenir des résultats comparables à ceux du réseau précédent. Nénmoins, on observe des aberrances pour les messages de longueur 30 qui sont surement dûes aux conditions expérimentales et à la génération des sacs à dos. Comme pour le réseau précédent, les messages de longueur 100 se sont révélés très difficiles à cryptanalyser : seul un test sur 3 donne un résultat concluant.

4.5 Conclusion

Plusieurs cryptosystème à base de sacs à dos existent, notamment des variantes plus ou moins complexes du système de Merkle et Hellman, telles que le cryptosystème de Qu et Vanstone, ou bien d'autres systèmes, tels celui de Chor et Rivest. Néanmois, à ce jour, tous les cryptosystèmes à base de sac à dos ont été cryptanalysés à l'aide d'algorithmes de réductions de réseaux. Seul le cryptosystème de Naccache et Stern, basé sur une extension du problème de résidu quadratique, tient depuis 1998. Les faiblesses des cryptosystèmes à base de sacs à dos font que RSA reste le cryptosystème à clé publique incontournable.

Annexe A

Histogrammes

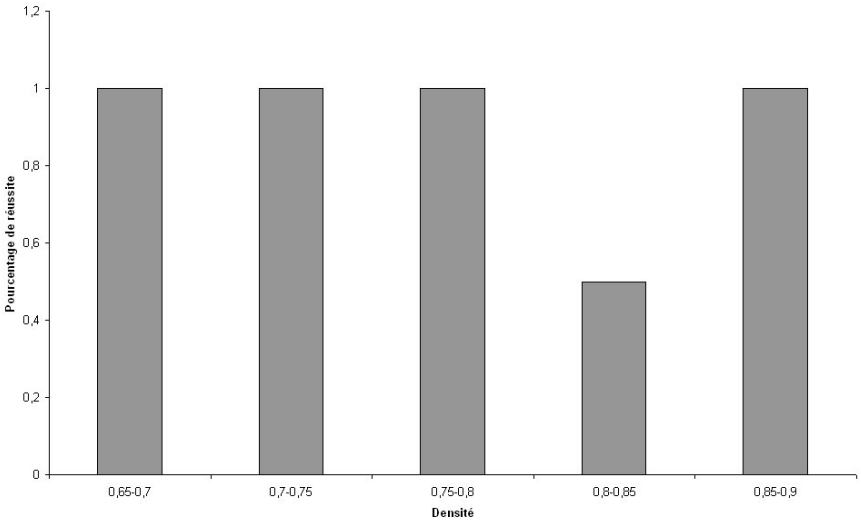

FIG. A.1 – Cryptanalyse d'un message de longueur 16 par le réseau de Lagarias-Odlyzko

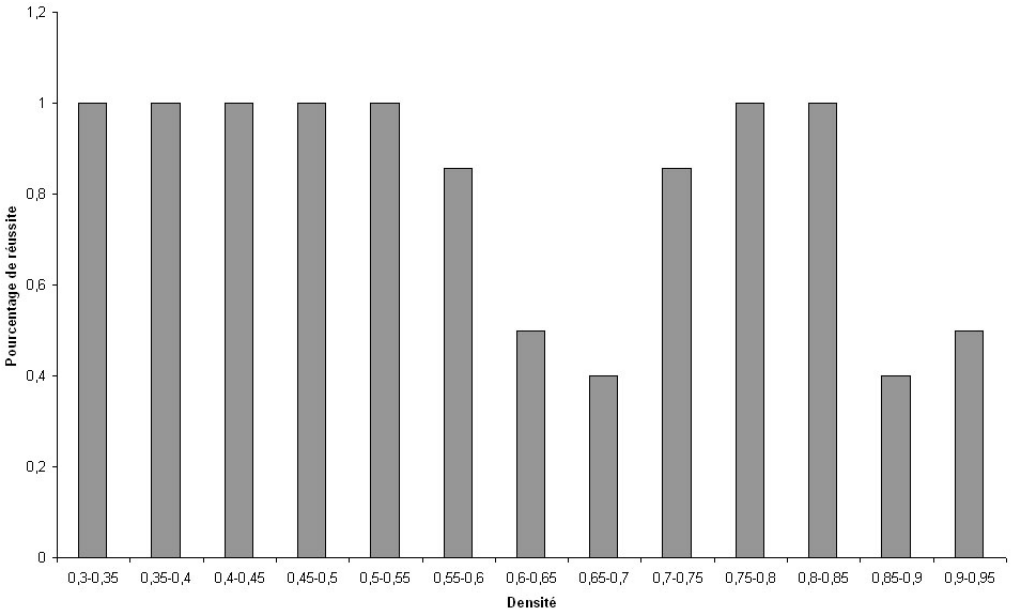

FIG. A.2 – Cryptanalyse d'un message de longueur 21 par le réseau de Lagarias-Odlyzko

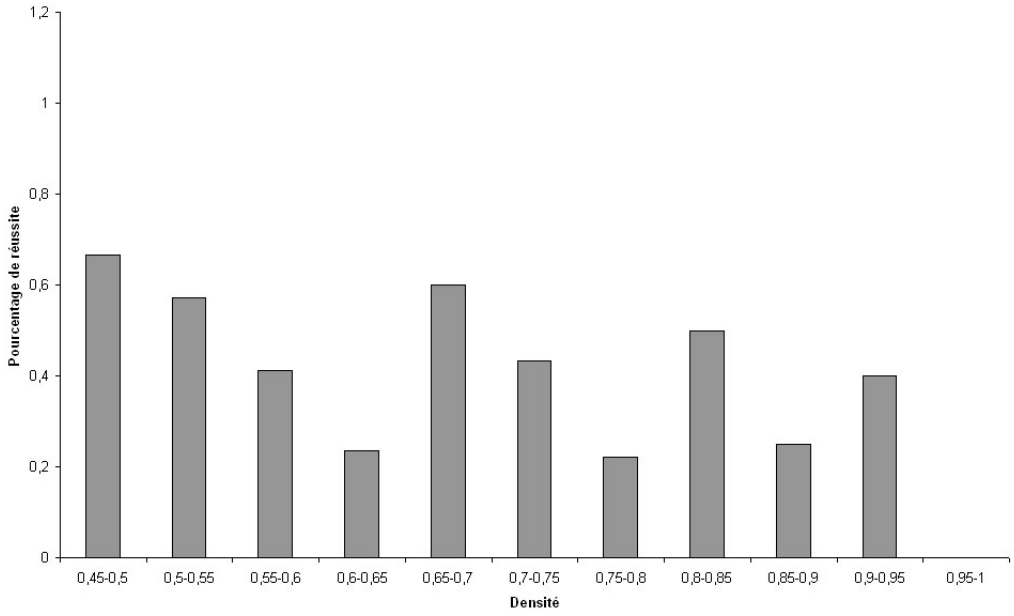

FIG. A.3 – Cryptanalyse d'un message de longueur 31 par le réseau de Lagarias-Odlyzko

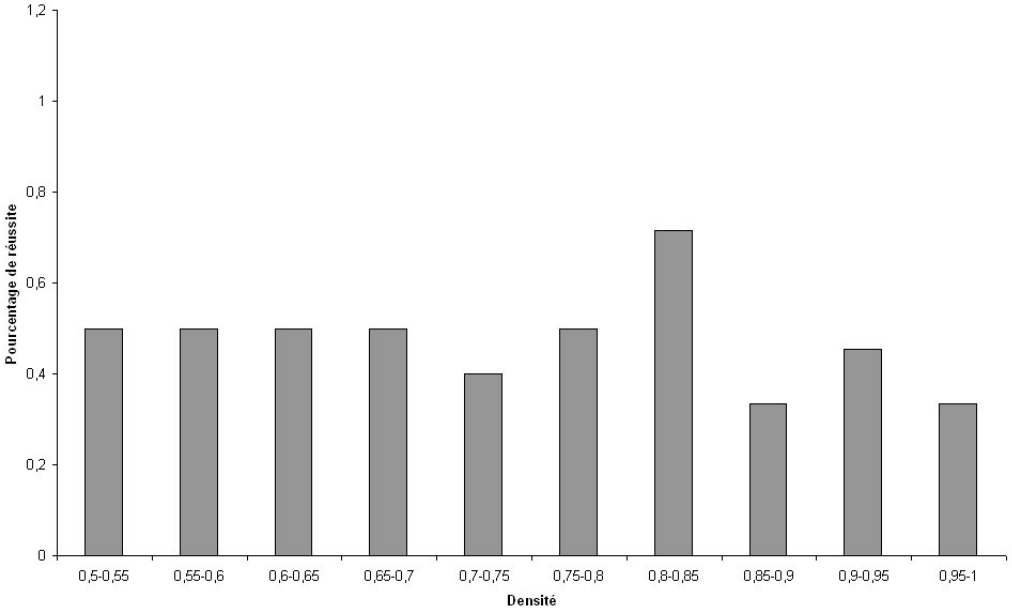

FIG. A.4 – Cryptanalyse d'un message de longueur 51 par le réseau de Lagarias-Odlyzko

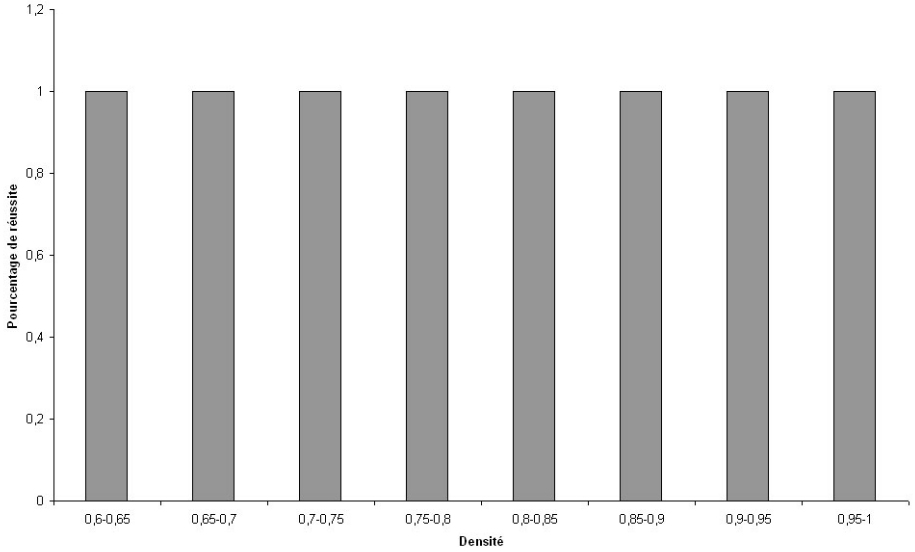

FIG. A.5 – Cryptanalyse d'un message de longueur 16 par le réseau de Coster, La Macchia, Odlyzko et Schnorr

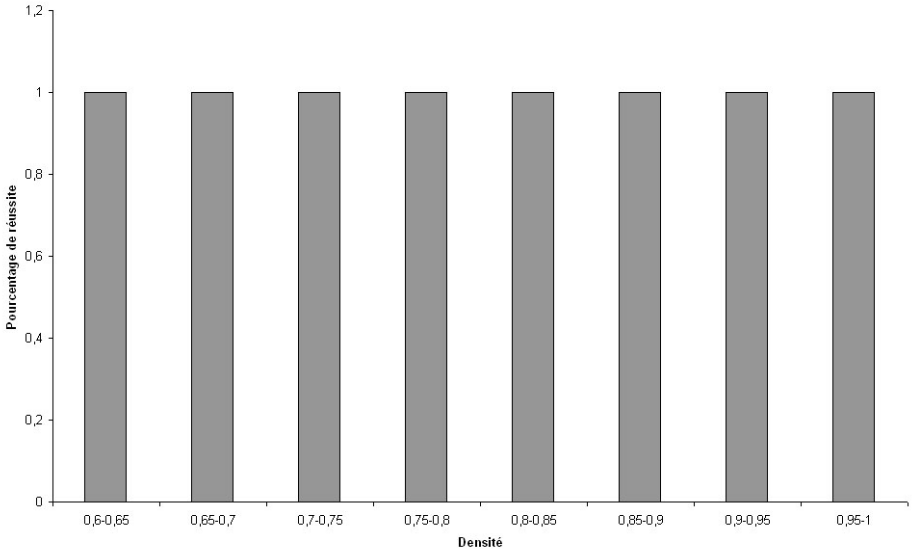

FIG. A.6 – Cryptanalyse d'un message de longueur 21 par le réseau de Coster, La Macchia, Odlyzko et Schnorr

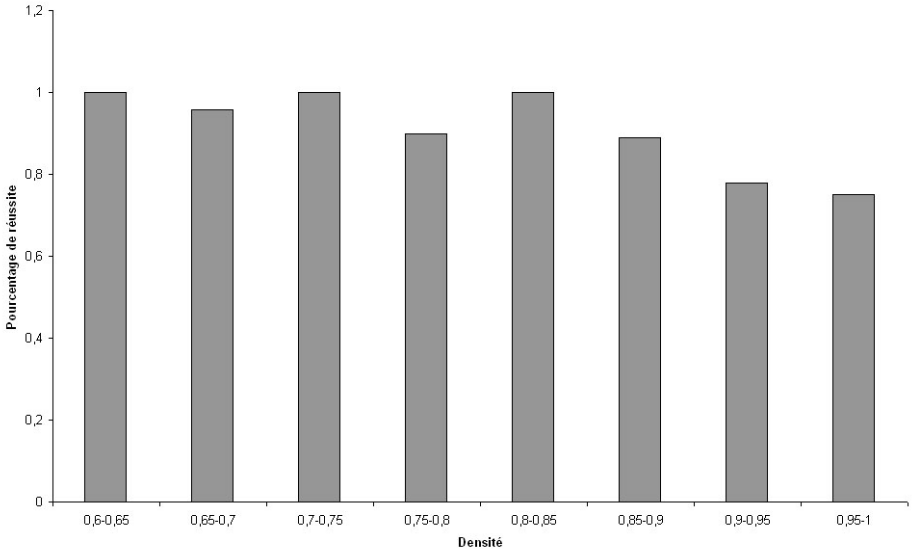

FIG. A.7 – Cryptanalyse d'un message de longueur 31 par le réseau de Coster, La Macchia, Odlyzko et Schnorr

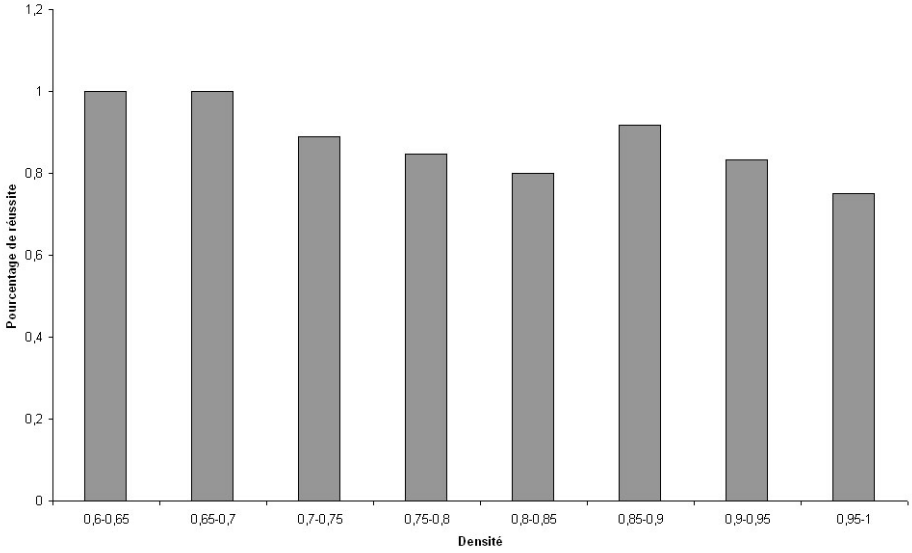

FIG. A.8 – Cryptanalyse d'un message de longueur 51 par le réseau de Coster, La Macchia, Odlyzko et Schnorr

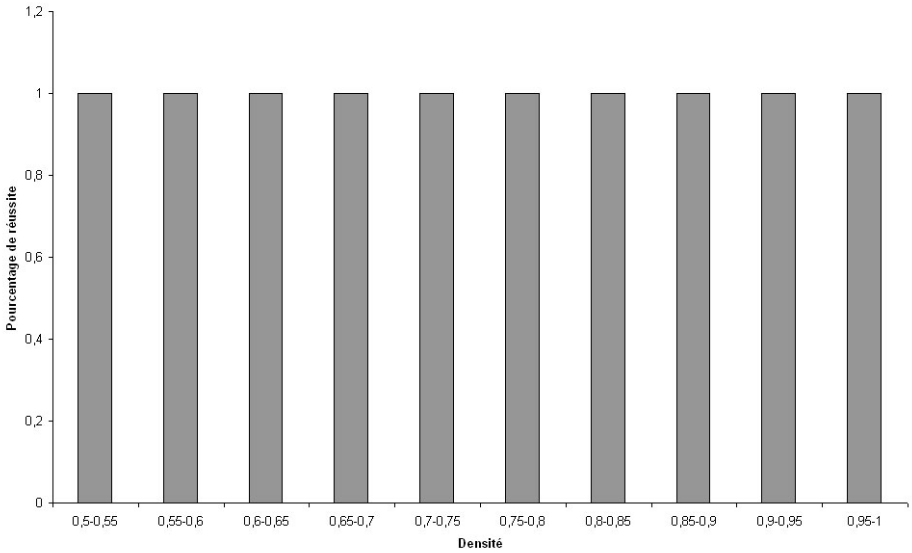

FIG. A.9 – Cryptanalyse d'un message de longueur 16 par le réseau de Joux et Stern

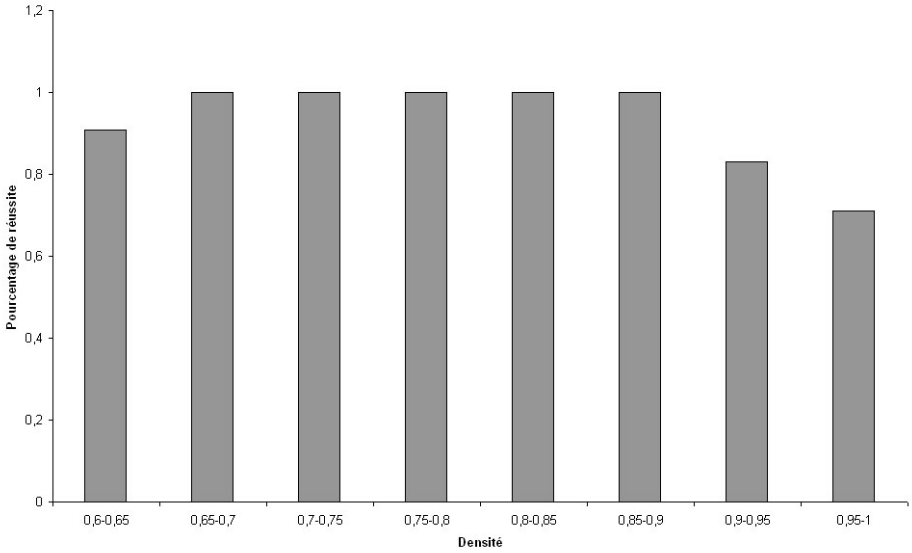

FIG. A.10 – Cryptanalyse d'un message de longueur 21 par le réseau de Joux et Stern

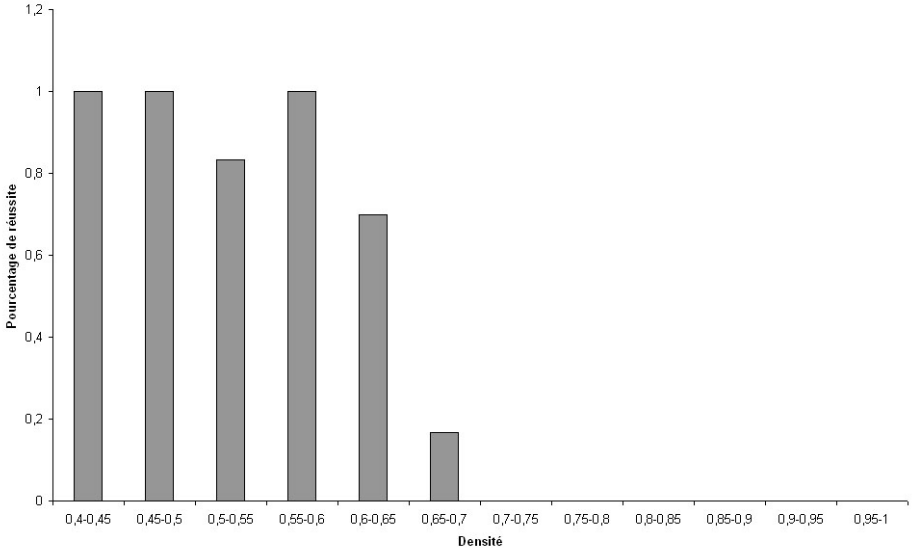

FIG. A.11 – Cryptanalyse d'un message de longueur 31 par le réseau de Joux et Stern

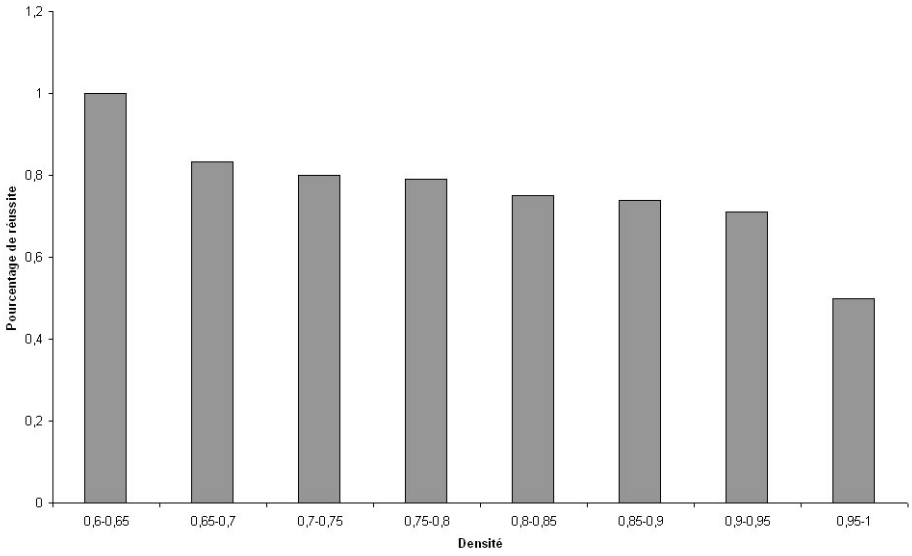

FIG. A.12 – Cryptanalyse d'un message de longueur 51 par le réseau de Joux et Stern

Annexe B

Extrait du code source Java

```
/**
 * @(#)MH.java
 *
 *
 * @author
 * @version 1.00 2007/5/14
 */
import java.util.*;
import java.math.*;
import java.io.*;

public class MH {

        private final static BigInteger zero=BigInteger.ZERO;
        private final static BigInteger one=BigInteger.ONE;
    private static BigInteger m,w,invw,c,d;
    private static int n;
    private static int [] msg;

    public static int [] captureMessage() throws IOException{
        System.out.println("Tapez le message
        a crypter (un entier positif):");

        InputStreamReader lecteur=new InputStreamReader(System.in);
                BufferedReader entree=new BufferedReader(lecteur);
                String message=entree.readLine();
                BigInteger b= new BigInteger(message);

                String s=b.toString(2);
                n=s.length();
                int [] msg=new int [n];
                int lg=b.bitCount();

                String s2;
                char c [] =new char [1];
                for(int i = 0 ;i<n;i++){

                        c[0]=s.charAt(n-i-1);

                        s2=new String(c);
                        msg[i]=Integer.parseInt(s2);
        }
```

```
        return(msg);
    }


        public static void creerCle(int n, BigInteger [] a1,
        BigInteger [] b1,  BigInteger m1, BigInteger w1){

                Random random = new Random();
        a1[0]= new BigInteger(7,random);
//Choisit un BigInteger entre 0 et 128

        a1[0]=a1[0].add(one);
//on ajoute 1 pour eviter le cas a[0]=0

        BigInteger s = a1[0];

        //definition des ai
        for (int i=1;i<n;i++){

                a1[i]= ( a1[i-1].multiply(new BigInteger("2")) ).add(
                new BigInteger(9,random).add(one));

                s = s.add(a1[i]);
        }

        //calcul de m
        m1 = s.add(new BigInteger(10,random));
        m=m1;

        //recherche de w
        w1 = m1.subtract(new BigInteger(18,random));
        while ((m1.gcd(w1).longValue()!=1)
        || (w1.compareTo(new BigInteger("1000"))<0))
                w1=m1.subtract(new BigInteger(18,random));

        w=w1;
        for (int j =0; j<n; j++)
                b1[j]=( a1[j].multiply(w1) ).mod(m1);



        affiche(a1,b1,m1,w1);

        }


    public static void affiche(BigInteger [] a1, BigInteger [] b1,
BigInteger m1, BigInteger w1){
                //affiche la clé privée et la clé publique
        String txta="(";
        String txtb="(";
        String txtpi="(";
        for (int k=0; k<a1.length;k++){
                txta=txta + a1[k] + ",";
                txtb=txtb + b1[k] + ",";
        }
        txta=txta+")";
        txtb=txtb+")";
        System.out.println("La cle publique est b="+txtb);
        System.out.println("La cle prive est (m=" + m1 + ",
         w=" + w1 +",a="+txta) ;
```

```
}

public static BigInteger chiffrer(int [] msg1,BigInteger []b1){
    BigInteger tmp;
    c=zero;
    // on calcule c=Somme des mi*bi
            for(int i=0;i<n;i++){
                    if (msg1[i]==0)
                    tmp=zero;
            else tmp =one;
            tmp=tmp.multiply(b1[i]);
            c=c.add(tmp);
            }

    System.out.println("c: "+c);
    return(c);



}

public static BigInteger dechiffrer(BigInteger [] a1){
    byte [] eps = new byte [n];
    //on calcule w^-1
    invw=w.modInverse(m);

    //on remplace c= somme des mi*bi par d= somme
    //des mi*ai en multipliant par w^-1
    d= (invw.multiply(c)).mod(m);

    //On resout le sac a dos super croissant
    sacADos(d,a1,eps);


    String txteps="(";
    for (int j =0; j<n; j++){
            txteps+=eps[j]+",";
    }
    txteps+=")";

    byte [] msgdecode= new byte [n];
    String txtdecode="(";
    for (int i=0; i<n;i++){
            msgdecode[i]= eps[n-i-1];
            txtdecode+=eps[i]+",";
    }
    txtdecode+=")";
    System.out.println("Le message decode:"+txtdecode);

    //mettre msgdecode en base 10
    BigInteger b= zero;
    BigInteger deux = new BigInteger("2");
    for(int i=0; i<n; i++){
            if (eps[i]==1)
                    b=b.add(deux.pow(i));
    }
    System.out.println("Ce qui donne en base 10:"+b);
    return(b);
}
```

```
public static byte [] sacADos(BigInteger d,
BigInteger [] objs, byte [] eps){
    //methode pour choisir les elements du sac a dos
    //correspondant a la solution
    BigInteger e=d;

    while (e.compareTo(zero)!=0){
            int i=objs.length-1;
            //on part du dernier element du sac a dos
            //(qui est le plus grand car le sac a dos
            //est super croissant)
            //et on cherche le premier qui soit inférieur ou egal a d
            while (i>0 && (e.subtract(objs[i])).compareTo(zero)<0  )
                    {
                            i--;
                    }
            //on place un marqueur sur l'element utilise
            //et on le soustrait a la valeur cible
            eps[i]=1;
            e=e.subtract(objs[i]);

    }
    return(eps);
}

public static void reseau1(BigInteger [] b,BigInteger c,int n){
     //renvoie la syntaxe latex du reseau de Lagarias Odlyzko
    //et la commande Maple appliquant LLL a ce reseau
    double r =Math.sqrt((double)n);
    long N = (long)Math.floor(r)+1;

    String latex="\\begin{pmatrix}";
    String maple="LLL([";
    BigInteger t;
    String p= ""+N;

            int cpt=0;boolean flag;
            for (int i=0; i<=n;i++){
                    flag=false;
                    maple+="[";
                    for(int j =0; j<n;j++){

                            if ((i<n) &&(j==cpt)&&(!flag)){
                                    flag = true;
                                    cpt++;
                                    latex+="1";
                                    maple+="1";
                            }
                            else {latex+="0";maple+="0";}
                            latex+=" & ";
                    /*      if(j!=(n)) */maple+=",";
                    }
                    t = new BigInteger(p);
                            if(i<n){
                                    t=t.multiply(b[i]);}
                            else {t=t.multiply(c);latex+="-";
                                    maple+="-";}

                    latex+=t + "\\\\";
                    maple+=t+"]";
                    if(i!=n) maple+=",";
            }
```

```
            latex+="\\end{pmatrix}";
            maple+="]);";
            System.out.println(latex);
            System.out.println();
            System.out.println(maple);


}

public static void reseau2(BigInteger [] b,BigInteger c,int n){
     //renvoie la syntaxe latex du reseau de Coster La MAcchia Odlyzko
    //et Schnorr et la commande Maple appliquant LLL a ce reseau
    double r =Math.sqrt((double)n)/2;
    long N = (long)Math.floor(r)+1;


    String latex="\\begin{pmatrix}";
    String maple="LLL([";
    BigInteger t;
    String p= ""+N;

            int cpt=0;boolean flag;
            for (int i=0; i<=n;i++){
                    flag=false;
                    maple+="[";
                    for(int j =0; j</*=*/n;j++){

                            if(i==n)
                                    {latex+="\\frac{1}{2}";
                                    maple+="1/2";
                            }
                            else{

                            if ((i<n) &&(j==cpt)&&(!flag)){
                                    flag = true;
                                    cpt++;
                                    latex+="1";
                                    maple+="1";
                            }
                            else {latex+="0";maple+="0";}}
                            latex+=" & ";
                    /*      if(j!=(n)) */maple+=",";

                    }

                    t = new BigInteger(p);
                            if(i<n){
                                    t=t.multiply(b[i]);}
                            else {t=t.multiply(c);}

                    latex+=t + "\\\\";
                    maple+=t+"]";
                    if(i!=n) maple+=",";
            }
            latex+="\\end{pmatrix}";
            maple+="]);";
            System.out.println(latex);
            System.out.println();
            System.out.println(maple);


}
```

```
public static void reseau3(BigInteger [] b,BigInteger c,int n){
     //renvoie la syntaxe latex du reseau de Joux et Stern
    //et la commande Maple appliquant LLL a ce reseau

    long N =((long) n)*((long )n) +1;
    System.out.println(N);
    String latex="\\begin{pmatrix}";
    String maple="LLL([";
    BigInteger t;
    String p= ""+N;

            int cpt=0;boolean flag;
            for (int i=0; i<=n;i++){
                    flag=false;
                    maple+="[";
                    for(int j =0; j<=n;j++){

                            if ((i<=n) &&(j==cpt)&&(!flag)){
                                    flag = true;
                                    cpt++;
                                    latex+=""+(n+1);
                                    maple+=""+(n+1);
                            }
                            else {latex+="-1";maple+="-1";}
                            latex+=" & ";
                    /*      if(j!=(n)) */maple+=",";
                    }
                    t = new BigInteger(p);
                            if(i<n){
                                    t=t.multiply(b[i]);}
                            else {t=t.multiply(c);
                            latex+="-";maple+="-";}

                    latex+=t + "\\\\";
                    maple+=t+"]";
                    if(i!=n) maple+=",";
            }
            latex+="\\end{pmatrix}";
            maple+="]);";
            System.out.println(latex);
                    System.out.println();
            System.out.println(maple);


}

public static void main(String[] args) throws IOException {


            int [] msg = captureMessage();
            n=msg.length;
            String txtm="(";
            for(int i =0; i<n;i++){
                    txtm+=msg[i]+",";
                    }
            txtm+=")";
            System.out.println("Le message en clair:"+txtm);

    BigInteger a [] = new BigInteger [n];
    //sac a dos a
    BigInteger b [] = new BigInteger [n];
```

```
        //sac a dos a*w mod m

                m=w= zero;


        creerCle(n,a,b,m,w);
        chiffrer(msg,b);
        dechiffrer(a);

        }
}
```

Bibliographie

[1] M.J. Coster, A. Joux, B.A. LaMacchia, A.M. Odlyzko, C.P Schnorr, and J. Stern. *An improved low-density subset sum algorithm.* Computational complexity 2, 1992.

[2] M.J. Coster, B.A. LaMacchia, A.M. Odlyzko, and C.P Schnorr. An improved low-density subset sum algorithm. *Advances in Cryptology : Proceedings of Eurocrypt '91*, 1991.

[3] A. Joux and J. Stern. *Improving the critical density of the Lagarias-Odlyzko attack against subset sum problems. Proceedings of Fundamentals of Computation Theory.* Springer LNCS 529, FCT91, 1991.

[4] J.C Lagrias and A.m Odlyzko. *Solving low-density subset problems.* J. Assoc. Comp. Mach., 1985.

[5] R. Merkle and M. Hellman. *Hiding information and signature in trapdoor knapsacks.* IEE Trans. Information, 1978.

[6] P.Q. Nguyen and J. Stern. *Lattice Reduction in Cryptology : an Update*, volume 1838. W.Bosma, Lecture Notes in Computer Science, 2000.

[7] P.Q. Nguyen and J. Stern. *Adapting Density Attacks to Low-Weight Knapsacks.* B.Roy, Lecture Notes in Computer Science, 2005.

[8] Guillaume Poupard. L'étrange sac à dos de merkle et hellman.

[9] E.F.Brickell. *Solving low density knapsacks.* Plenum Press, Proceedings of Crypto '83, 1983.

[10] E.F.Brickell. *Breaking iterated knapsacks.* Springer Verlag, Proceedings of Crypto '84, 1985.

[11] H.W.Lenstra. Jr. integer programming with a fixed number of variables. Technical report, Mathematisch Instituut, Universiteit van Amsterdam, 1981.

[12] L.M.Adleman. *On beaking generalized knapsack public key cryptosystems.* ACM, Proceedings of 15th STOC, 1983.

[13] C. G. Lekkerkerker. *Geometry of Numbers.* North-Holland, 1969.

[14] Jacques Martinet. *Les réseaux parfaits des espaces euclidiens.* Masson, 1996.

[15] Phong Quang Nguyen. *La géométrie des nombres en cryptologie.* PhD thesis, Université de Paris 7, 1999.

[16] Arto Salomaa. *Public-Key Cryptography.* Springer-Verlag, 1990.

[17] Pierre Samuel. *Théorie algébrique des nombres.* Hermann Paris, 1967.