UNIVERSITÉ CLAUDE BERNARD LYON1

Le Knapsack et le chiffre de Merkle-Hellman

Clotilde GUILLET & Adrianna YOMBI

2008-2009

But du projet

Comme l'indique le sujet nous allons commencer par l'étude du « ***knapsack problem*** » ou encore le « ***problème du sac à dos*** ». Il est l'un des 21 problèmes NP-complets présentés par **Richard KARP** en 1972. Nous allons introduire la notion de suite super-croissante, puis nous parlerons des différents types d'empilements qui existent : l'**empilement en temps linéaire** et celui en **temps exponentiel**.

Nous nous concentrerons beaucoup plus sur le « ***knapsack algorithm*** » ou encore l' « ***algorithme du sac à dos*** » car il a été le premier à être utiliser pour généraliser le cryptage à clé publique. Pour illustrer cette partie nous parlerons des chiffrements à clés publiques et privées.

Etant donné que cet algorithme est NP-complet nous aborderons brièvement cette notion.

Pour entrer dans le vif du sujet, nous parlerons de son utilisation particulière du « ***knapsack algorithm*** **»** dans le cryptosystème développé par **Ralph MERKLE** et **Martin HELLMAN** en 1978. Pour cette partie nous mettrons en place une application java (applet) qui permettra de crypter puis de décrypter un message donné.

En 1982 **A. SHAMIR**, puis en 1985 **BRICKEL** ont démontré que cet algorithme pouvait être cassé.

Ceci peut être fait grâce à un algorithme de réduction de réseau qui s'exécute en temps polynomial.

Cet algorithme utilisé est le « ***LLL*** » ou encore « ***lattice reduction*** ». Il a été mis en place par **A. LENSTRA**, **H. LENSTRA** et **L. LOVASZ**. Il est inspiré de la classique orthogonalisation de Gram-Schmidt. Il permet de résoudre des problèmes qui peuvent se ramener à la recherche de petits vecteurs dans un réseau.

La seconde partie de notre travail va consister à implémenter cet algorithme.

Introduction

La cryptographie est une méthode permettant de rendre illisible des informations afin de garantir l'accès à un seul destinataire authentifié. La conversion des données s'effectue au moyen d'une clé. Il existe deux types de cryptographie, la cryptographie à clé secrète et la cryptographie à clé publique.

I.1. Notions sur la Cryptographie à clé publique

On parle de cryptographie asymétrique ou encore de cryptographie à clé publique.

Dans ce type de cryptographie, les algorithmes sont publics, mais chaque individu possède un couple de clés : l'une secrète (appelée **clé privée**) lui permettant d'effectuer les opérations que lui seul est sensé être en mesure de faire (signature ou déchiffrement), tandis que sa **clé publique** est diffusée afin de permettre à ses interlocuteurs de mettre en œuvre les opérations réciproques (vérification de signature ou chiffrement de message). Les deux clés ont un rôle « asymétrique », d'où la terminologie.

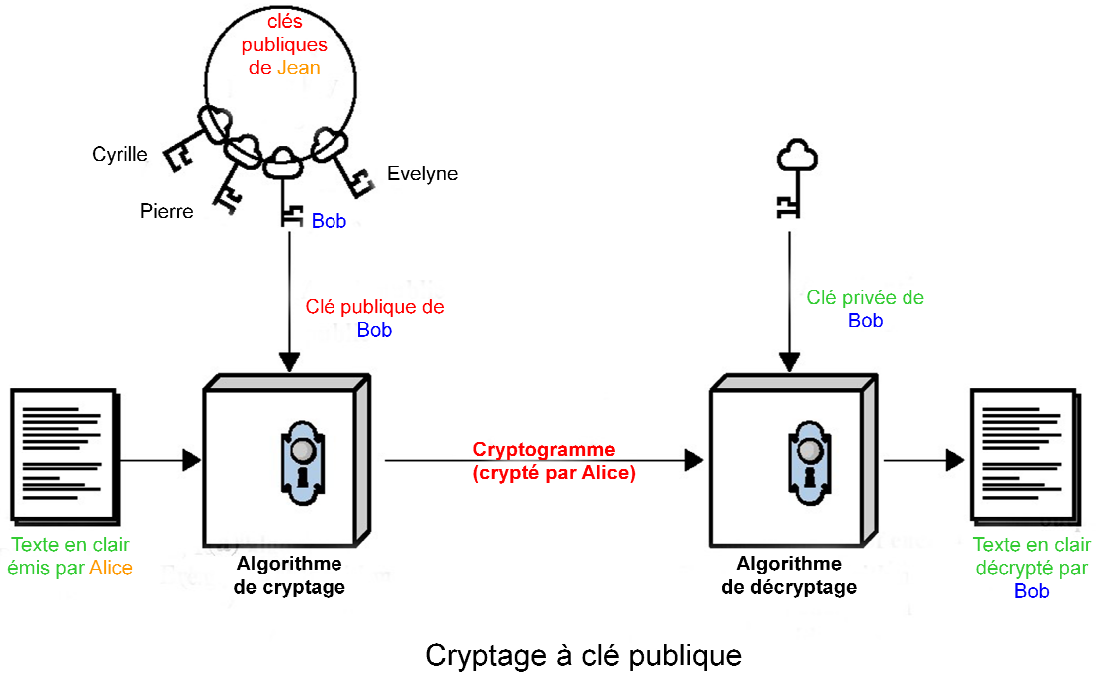


Figure1. Le schéma ci-dessus illustre le déroulement d'un cryptage à clé publique.

I.2. Fonction à sens unique à trappe

Une fonction $f : M \rightarrow C$ est dite **à sens unique** si :

- pour tout x de M, il est **facile** de calculer $f(x)$ (autrement dit, f peut être calculée en temps polynomial) et,
- il est **difficile** de trouver, pour la plupart des $y \in f(M)$ un $x \in M$ tel que $f(x) = y$ à moins d'exécuter un nombre prohibitif d'opérations, ou d'avoir une chance sur laquelle il est déraisonnable de compter.

Le calcul dans le sens inverse sera lui aussi efficace pourvu qu'on dispose d'une information secrète : la **trappe** c'est-à-dire une fonction g telle que $g \circ f = Id$.

La construction des couples (*f, g*) doit être facile et la publication de *f* ne doit rien révéler sur *g*.

C'est là qu'on retrouve formalisée l'idée d'utiliser deux algorithmes différents, un pour chiffrer et un pour déchiffrer.

I.3. Les algorithmes Knapsack

Le premier algorithme généralisé de chiffrement à clé publique est le « Knapsack algorithm » ou encore « l'algorithme du sac à dos ». Il a été développé par **Ralph MERKLE** et **Martin HELLMAN** en **1978**.

Il pouvait seulement être utilisé pour le cryptage, bien que plus tard Adi SHAMIR l'ait adapté pour les signatures digitales.

Les algorithmes Knapsack tiennent leur grand niveau de sécurité du « problème de sac » qui est un problème NP-complet.

I.4. Le problème du sac à dos

C'est un problème qui est très simple. Supposons une pile d'objets, chacun avec un poids différents, est-il possible de mettre quelques uns de ces objets dans un sac à dos tout en sachant que ce dernier à un poids maximum ? Plus formellement, supposons qu'on a une série de valeur M_1, M_2, …, M_n et une somme S, calculez les valeurs des $\boldsymbol{b_i}$ tel que :
$S= b_1M_1+b_2M_2+…+b_nM_n$.

Les valeurs de $\mathbf{b_i}$ peuvent être des 0 ou des 1. Le « un » indique que l'objet est dans le sac et le « zéro » indique qu'il n'y est pas.

Par exemple, les objets doivent peser respectivement 1, 5, 6, 11, 14 et 20. On ne peut que remplir un sac à dos qui pèse 22 en utilisant les poids 5, 6 et 11. Par contre on ne pourra pas faire de même avec un sac qui pèse 24. En général le temps requis pour résoudre ce genre de problème semble croitre exponentiellement avec le nombre d'objet dans la pile.

L'idée soutenue par l'algorithme de Merkle-Hellman est d'encoder un message comme une solution à une série de problèmes de sac à dos.

Un bloc de texte en clair de longueur égale à nombre d'objet dans la pile sélectionnera les objets dans le «sac à dos » (le texte en clair correspondant à la valeur de b), et le texte chiffré serait la somme résultante.

I.5. Les empilements

Il existe deux types de problèmes d'empilement :

– une soluble en temps linéaire,
– une soluble en temps exponentiel.

a) Empilement facile

Si la liste des poids est super-croissante, on utilise un algorithme (appelé glouton *cf. annexe A*) de la manière suivante :

1. Prendre le poids total (il est à signaler que le poids total ici n'est pas la somme de tous les éléments dans la suite) et le comparer avec le plus grand nombre de la suite.

– Si le poids total est inférieur à ce nombre, alors celui-ci n'est pas dans le tas. On recommence l'opération avec le nombre suivant dans le tas (qui, par définition de la suite, sera plus petit)
– Si le poids total est supérieur à ce nombre, alors celui-ci est dans le tas
2. Réduire le poids du tas à créer de ce nombre et passer au plus grand nombre suivant de la suite.
3. Répéter jusqu'à ce que ce soit terminé.
4. Si le poids total a pu être ramené à 0 : il y a une solution.

b) Empilement difficile

Dans le cas présent, on ne connaît pas d'algorithme rapide. Il faut tester méthodiquement toutes les solutions possibles, ce qui, si la suite des poids est suffisamment longue, est impraticable. Ces algorithmes sont en temps exponentiels.

Le cryptosystème de Merkle-Hellman exploite cette propriété. La clé privée est une suite de poids super-croissante. A partir de celle-ci, on calcule la clé publique. Ce calcul consiste à prendre la suite super-croissante, et à la multiplier par (n modulo m), avec m supérieur à la somme de tous les termes de la suite, et n ne devant avoir aucun facteur commun avec m.

Remarque : L'empilement facile peut être transformé pour créer un empilement difficile. Pour la clé publique, on utilisera un empilement difficile qui servira à chiffrer. La clé privée quant à elle, utilisera un empilement facile, qui donne un moyen simple de déchiffrer les messages.

Bien sûr, ceux qui ne connaissent pas la clé privée sont obligés de résoudre le problème d'empilement difficile, ce qui est infaisable en pratique.

I.6. Les suites super- croissantes ou suprecreasing knapsack

Si la liste des poids est une séquence super-croissante alors le résultat du problème de « sac à dos » est facile à résoudre.

Une séquence super-croissante est une séquence dans laquelle chaque terme est plus grand que la somme de tous les termes qui le précédent. Par exemple : {1,3,6,13,27,52} est une séquence super-croissante par contre {1,3,4,9,15 ,25} ne l'est pas.

La solution à un « knapsack » super-croissant est facile à trouver, le procédé est identique à l'algorithme utilisé pour l'empilement facile.

Par exemple, on considère un « knapsack » total ayant un poids de 70 et une séquence de poids de : {2, 3, 6, 13, 27,52}. Le plus grand poids est 52, il est plus petit que 70 alors 52 est dans le « knapasck ». On soustrait 52 de 70, il reste 18.Le prochain plus grand nombre de la séquence est 27, ce dernier est supérieur à 18, ce qui fait qu'il n'est pas dans le « knapsack ». Le prochain poids est 13 qui est inférieur à 18, on soustrait 13 de 18 il restera 5. Le poids suivant est 6 qui est supérieur à 5 alors 6 n'est pas dans le «knapsack ». En continuant ce processus nous montrerons que 2 et 3 sont tous les deux dans le « knapsack » et le poids total est ramené 0, ceci indique que la solution a été trouvée.

Le texte en clair résultant d'un texte chiffré ayant comme valeur 70 serait 110101.

Si la séquence n'est pas super-croissante ou si elle est normale, alors il n'existe pas d'algorithme qui permettent de la résoudre rapidement. La seule manière de déterminer

quels sont les éléments qui sont dans le « knapsack » serait de tester méthodiquement toutes les solutions possibles jusqu'à en trouver une qui soit correcte. Même les algorithmes les plus rapides, tenant compte des différentes heuristiques croient exponentiellement avec le nombre de poids possible dans le « knapsack ». Si on ajoute un poids de plus dans le « knapsack », il faudra deux fois plus de temps pour trouver une solution possible.

I.7. Notions sur les problèmes NP-Complet dans le « knapsack »

Le problème de sac à dos peut être représenté sous une forme décisionnelle en remplaçant la maximisation par la question suivante : un nombre *k* étant donné, existe-t-il une valeur des x_i pour laquelle $\sum_{i=1}^{n} pixi \geq k$, avec respect de la contrainte ?

Il y a un lien entre la version « décision » et la version « optimisation » du problème dans la mesure où s'il existe un algorithme polynomial qui résout la version « décision », alors on peut trouver la valeur maximale pour le problème d'optimisation de manière polynomiale en appliquant itérativement cet algorithme tout en augmentant la valeur de *k*.

D'autre part, si un algorithme trouve la valeur optimale du problème d'optimisation en un temps polynomial, alors le problème de décision peut être résolu en temps polynomial en comparant la valeur de la solution sortie par cet algorithme avec la valeur de *k*. Ainsi, les deux versions du problème sont de difficulté similaire.

Sous sa forme décisionnelle, le problème est NP-complet, ce qui signifie qu'il n'existe pas de méthode générale connue pour construire une solution optimale, à part l'examen systématique de toutes les solutions envisageables. Le problème d'optimisation est NP-difficile, sa résolution est au moins aussi difficile que celle du problème de décision, et il n'existe pas d'algorithme polynomial connu qui, étant donné une solution, peut dire si elle est optimale (ce qui reviendrait à dire qu'il n'existe pas de solution avec un *k* plus grand, donc à résoudre le problème de décision NP-complet).

Les problèmes NP-complet sont dits « calculatoirement difficile ».

II.1. Le chiffre de Merkle-Hellman

Ce chiffre est basé sur la propriété suivante : la clé privée est une séquence de poids super-croissante pour un problème de « knapsack ». La clé publique est une séquence de poids pour un problème de knapsack normal. Avec une même solution (solution unique). Merkle et Hellman ont développé une technique pour convertir un problème de knapsack à séquence super-croissante en un problème de knapsack normal. Pour ce faire, ils ont dû utiliser l'arithmétique modulaire.

Merkle et Hellman ont utilisé la fonction à sens unique suivante pour leur chiffre :

Tout entier x, $0 \leq x \leq 2n - 1$ peut être représenté en binaire sur n bits. On définit $f(x)$ comme le produit scalaire $<A, x>$

II.2. Création d'une clé publique à partir de la clé privée

Elle se déroule de la manière suivante :

- Choisir une séquence super-croissante, par exemple : {2,3, 6, 13, 27,52}.
- Multiplier toutes ces valeurs par un nombre **n modulo m.** Le modulo doit être un nombre plus grand que la somme de ceux qui sont dans la séquence.

  Le modulo devrait être un chiffre qui soit plus grand que la somme de tous les nombres de la séquence (ici la somme totale des nombres est égal à : 103) donc on pourrait choisir ici le chiffre 105 par exemple.

  Le multiplicateur ne doit pas avoir de facteur en commun avec le modulo : nous prendrons comme exemple le chiffre 31. Et l'empilement devrait se faire comme suit :

2*31 mod 105= 62;
3*31 mod 105= 93;
6*31 mod 105= 81;
13*31 mod 105= 88;
27*31 mod 105= 102;
52*31 mod 105= 37;

Et le « knapsack » serait : {62, 93, 81, 88, 102,37}.

II.3. Cryptage

Pour crypter un message binaire :

1. Diviser ce dernier en des blocs égal au nombre d'items qui existe dans la suite de poids du « knapsack » normal.
2. Ensuite, le fait qu'il y'ait un 1 indique que l'élément est présent la suite et un zéro pour indiquer que l'élément est absent.
3. Calculer le poids total pour chaque bloc du message.

Exemple :

Si le message binaire est 011000110101101110, on procèdera à son cryptage comme suit :

Message= 011000 110101 101110
011000 correspond à 93+81= 174
110101 correspond 62+93+88+37=280
101110 correspond 62+81+88+102=333

Ainsi le texte chiffré serait : {174, 280, 333}.

II.4. Décryptage

Le destinataire du message connait la clé privée : la séquence de poids super-croissante pour un problème de « knapsack », ainsi que les valeurs de n et m qui ont été utilisé pour la transformer.

Pour décrypter le message, ce destinataire doit déterminer n^{-1} grâce à ***l'algorithme d'Euclide étendu***. Les entiers n, n^{-1} et m constituent l'information secrète.

Ensuite multiplier chaque valeur du texte chiffré par n^{-1} mod m et puis partitionner à l'aide de la clé privée (la séquence de poids super-croissante pour un problème de « knapsack ») pour obtenir les valeurs du texte en clair.

Pour notre exemple, nous avons la suite super-croissante de poids qui est {2, 3, 6, 13, 27, 52}, puis nous avons m qui est égal à 105 et n qui vaut 31. Le message chiffré est {174, 280, 333}. Dans ce cas n^{-1} est égal à 61, donc les valeurs du message chiffré doivent être multiplié par 61 mod 105.

174*61 mod 105=9= 3+6 ; qui correspond à 011000.
280*61 mod 105=70= 2+3+13+52 ; qui correspond à 110101.
333*61 mod 105=48= 2+6+13+27; qui correspond à 101110.

Le texte décrypté est 011000 110101 101110.

Remarque : Il est important que le nombre de bits par paquets soit différent de la longueur de la clé. En effet si ce n'est pas le cas, on pourrait attaquer le texte par une **analyse des fréquences** car chaque nombre serait chiffré par le même nombre.

III. Cassage du cryptosystème de Merkle-Hellman

En 1982, Shamir découvre une attaque permettant d'obtenir le message clair en ne se servant que de la clé publique et du chiffre issu du cryptosystème de Merkle-Hellman, en utilisant l'algorithme de la réduction des réseaux.

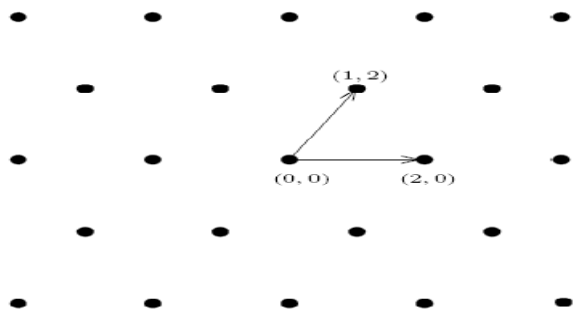
III.1. Réduction de réseau

Afin d'attaquer le cryptosystème de Merkle et Hellman, un algorithme appelé ***réduction de réseau*** est utilisé. Il est inspiré de la classique orthogonalisation de Gram-Schmidt, et possède un comportement encore mal compris au sens où il fournit souvent des résultats bien meilleurs que ceux attendus !

Afin de le décrire, il faut introduire un objet mathématique appelé **réseau**.

Un réseau est un groupement d'un ensemble de points distancés entre eux. D'un point de vue mathématique, un réseau est une combinaison linéaire à coefficients entiers des vecteurs de la base représentative de ce réseau.

Voici un exemple d'un réseau $\mathbb{Z}(2,0)\oplus\mathbb{Z}(1,2)$:



Le réseau $\mathbb{Z}(2,0) \oplus \mathbb{Z}(1,2)$.

Considérons *n* vecteurs b_i à *n* dimensions dont les composantes sont par exemple des réels.

On considère l'ensemble des vecteurs de $\mathbb{R}^n$ engendrés en additionnant un nombre quelconque, mais entier, de chaque vecteur. Cet ensemble est appelé ``***réseau engendré par la base b1, b2,..., bn***'' et est noté *L*.

Il peut être simplement défini par :

$$L = \mathbb{Z} \times b_1 + \mathbb{Z} \times b_1 + \ldots + \mathbb{Z} \times b_n = \left\{ \sum_{i=1}^{n} r_i \times b_i \ : \ (r_1, r_2 \ldots, r_n) \in \mathbb{Z}^n \right\}$$

Une ***base*** d'un réseau est un ensemble de vecteurs linéairement indépendants qui engendrent le réseau.

L'idée de la réduction de réseau est de calculer une nouvelle base engendrant le même réseau que (b_1, b_2,..., b_n) mais dont les vecteurs sont plus courts et ``plus orthogonaux''.

III.2. L'algorithme LLL

Le principal algorithme de réduction de réseau, appelé **LLL**, a été proposé en 1982 par **A. Lenstra**, **H. Lenstra** et **L. Lovász** avec comme objectif de factoriser des polynômes. L'idée élémentaire est que, étant donné une base d'un réseau, on continue à engendrer la même base en échangeant des vecteurs ou en additionnant une combinaison linéaire à coefficients entiers de vecteurs à un autre vecteur de la base, de manière à construire une base réduite.

Cet algorithme fonctionne selon le principe suivant. LLL est une combinaison de Gauss dans un hyperplan et de Gram Schmidt. On essaye d'appliquer Gauss à des sous-réseaux projetés de dimension 2. Plus précisément, on commence par projeter b_i et b_{i+1} orthogonalement à l'espace engendré par (b_1, …, b_{i-1}) et on effectue une étape de Gauss sur le réseau projeté.

Les opérations de translation et d'échange dans l'espace projeté sont relevées et s'appliquent en fait sur b_i et b_{i+1} eux-mêmes. De plus, si nécessaire, le nouveau vecteur obtenu est aussi translaté par rapport à (b_1, …, b_{i-1}) pour se rapprocher de son projeté.

Comme dans l'algorithme de Gauss, il y a deux types d'opérations : des translations et des échanges de vecteurs. Les translations se limitent à déplacer un vecteur parallèlement à ses prédécesseurs. Les échanges n'agissent que sur des vecteurs voisins.

Dans cet algorithme, on note entre crochets le produit scalaire habituel, c'est-à-dire :

$$<(a_1,a_2,...,a_n),(b_1,b_2,...,b_n)>=\sum_{i=1}^{n} a_i \times b_i$$

On note également $\lfloor x \rceil$ l'entier le plus proche de x. L'algorithme nécessite de plus l'emploi de deux matrices à n lignes et n colonnes notées μ et b^*. Enfin, si b est une matrice, b_i désigne le vecteur de la ligne i et $b_{i,j}$ désigne la composante située à l'intersection de la ligne i et de la colonne j.

Le principal intérêt de cette réduction est qu'elle permet de trouver un vecteur court du réseau, au sens de la norme euclidienne, qui apparaît en résultat comme élément de la base réduite. Rien ne garantit que l'on trouve le vecteur non nul le plus court mais, expérimentalement, l'algorithme LLL est très performant et permet d'obtenir des vecteurs très courts. En particulier, si un réseau possède un vecteur ``anormalement'' court, il y a de grandes chances qu'on l'obtienne grâce à cet algorithme (*cf. annexe B pour le code de l'algorithme*).

III.3. Cryptanalyse du système de Merkle et Hellman

L'algorithme LLL va nous permettre de cryptanaliser le chiffre de Merkle-Hellman en résolvant un problème de sous-ensembles (SSP) de faible densité.

On définit la densité d'un n-uplet (a_1, …, a_n) comme :

$$D= n/ (\max \{ \log(a_i) : 1 \leq i \leq n \})$$

L'algorithme LLL nous permet de ramener le problème de la somme de sous-ensembles à celui de la recherche d'un vecteur court dans un réseau. LLL construit une base réduite qui contient un vecteur dont la norme est à un facteur 2 près celle du plus court vecteur du réseau.

En pratique, LLL trouve un vecteur qui est bien meilleur et LLL peut trouver une solution au problème de la somme de sous-ensembles pourvu que ce vecteur soit plus court que la plupart des vecteurs non nuls du réseau.

Attaquer le cryptosystème de Merkle et Hellman va consister, étant donné un chiffre c et une clé publique (a_1, a_2, ..., a_n) à trouver (m_1, m_2,..., m_n) ∈ $\{0,1\}^n$ tel que $\sum_{i=1}^{n} m_i \times a_i = c$,

c'est-à-dire à simplement retrouver le message à partir du chiffre et de la clé publique.

a) Définition du réseau

Considérons le réseau engendré par les lignes de la matrice suivante :

$$\begin{pmatrix} 1 & 0 & 0 & \dots & 0 & -a_1 \\ 0 & 1 & 0 & \dots & 0 & -a_2 \\ 0 & 0 & 1 & \dots & 0 & -a_3 \\ \dots & & & \dots & \dots & \\ \dots & & & \dots & \dots & \\ \dots & & & \dots & \dots & \\ 0 & 0 & 0 & \dots & 1 & -a_n \\ 0 & 0 & 0 & \dots & 0 & c \end{pmatrix}$$

Notons b_i la ligne i de la matrice. Si $\sum_{i=1}^{n} m_i \times a_i = c$, il est alors clair que :

$$\sum_{i=1}^{n} mi \times bi - bn + 1 = (m1, m2, \dots, mn, 0)$$

Puisque la dernière composante est égale à $\sum_{i=1}^{n} m_i \times a_i - c$. La norme euclidienne de ce vecteur est donc inférieure à $(n)^{1/2}$ (borne atteinte dans le pire des cas si tous les m_i sont égaux à 1). On observe donc qu'à une solution du problème de sac à dos est associé un vecteur particulièrement court du réseau. Si LLL est capable de trouver ce vecteur court, il sera possible d'en déduire les m_i, c'est-à-dire de casser le cryptosystème de Merkle et Hellman.

III.4. Exemple de réduction de réseau

Considérons le problème de sac à dos suivant : les a_i valent 366, 385, 392, 401, 422, 437 et la valeur cible est c=1215.

Nous allons donc réduire le réseau défini par les vecteur-lignes de la matrice suivante :

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & -366 \\ 0 & 1 & 0 & 0 & 0 & 0 & -385 \\ 0 & 0 & 1 & 0 & 0 & 0 & -392 \\ 0 & 0 & 0 & 1 & 0 & 0 & -401 \\ 0 & 0 & 0 & 0 & 1 & 0 & -422 \\ 0 & 0 & 0 & 0 & 0 & 1 & -437 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1215 \end{pmatrix}$$

L'application de LLL fournit le réseau suivant :

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

Le premier vecteur, le plus court de la nouvelle base, fournit la solution. En effet, la dernière composante est nulle, ce qui indique que l'on a trouvé une combinaison linéaire nulle à coefficients entiers des a_i et de c.

De plus, la structure très particulière de la matrice définissant initialement le réseau fait que ces coefficients valent respectivement (0, 0,-1,-1,-1, 0,0) pour les a_i. Nous en déduisons que 392+401+422 est un multiple de 1215. Il est alors immédiat de vérifier que l'on a bien 392+401+422=1215.

Notons que LLL trouve également de nombreux autres vecteurs courts du réseau mais que ces derniers ne correspondent à aucune solution en termes de problème de sac à dos. Rappelons enfin que, bien que cet algorithme fonctionne très bien, il peut ne pas trouver de solution, même s'il en existe une.

Remarque

Pour illustrer cette partie nous avons réalisé le cassage du chiffre de Merkle-Hellman avec le logiciel ***Maple***. Son fichier d'exécution ainsi qu'une page html attestant du résultat obtenu sera joint à notre rapport. *(cf annexe C)*

IV. Conclusion

Le problème du sac à dos illustre la construction d'un système de chiffrement à clé publique au moyen d'un problème NP-complet. Il s'agit du chiffre de Merkle-Hellman.

Ce type de cryptage a été le premier à utiliser le système des clés publiques.

Chaque protagoniste choisit une clé privée (qui doit être une suite super-croissante) et à partir de celle-là, calcule sa clé publique (qui sera connu de tout le monde).

L'expéditeur chiffre son message à l'aide de la clé publique du destinataire. Et ce dernier le déchiffre à l'aide sa clé privée et de deux autres variables, obtenu lors du calcul de la clé publique.

Cependant le chiffrement de Merkle et Hellman (celui du sac à doc) a subi plusieurs attaques fructueuses. En 1978, Herleston a publié un article dans lequel il a remarqué qu'un bit du cryptogramme obtenu par le système du sac à dos peut être révélé. En 1979, Shamir a proposé une cryptanalyse d'un système de sac à doc dans certaines circonstances. Finalement, shamir et Zippel ont trouvé des failles dans la transformation qui permet de reconstruire la suite super-croissante à partir de suite normale. Mais, Shamir reste le premier à avoir utilisé l'algorithme LLL pour casser le cryptosystème de Merkle-Hellman en utilisant l'algorithme linéaire de Lenstra.

L'une des véritables cryptanalyses du chiffre de Merkle-Hellman (technique qui utilise la réduction de réseaux arithmétiques par l'algorithme LLL) a conduit assez rapidement à un abandon des méthodes de chiffrement basées sur des problèmes de sac à dos. Beaucoup de modifications ont été appliqué au problème du sac à dos, surtout au niveau de la sécurité. Il semblerait qu'il y aurait quand même des utilisations pour des petites sécurités basé sur le cryptosystème du knapsack pour protéger les signatures et les connexions. Cependant, cela semble être le futur de nouvelles applications, par exemple, des protocoles qui ont été standardisés sur des machines, avec des moyens de communication qui traversent un réseau. Bien évidemment les recherches avec cette idée évoluent avec une approche complètement différente.

Annexes

Annexe A

Algorithme du glouton

```
pour c de 0 à W faire
  T[0,c] := 0
fin pour

pour i de 1 à n faire
  pour c de 0 à W faire
    si c>=w[i] alors
      T[i,c] := max( T[i-1,c], T[i-1, c-w[i]] + p[i] )
    sinon
      T[i,c] := T[i-1,c]
    fin si
  fin pour
fin pour
```

Annexe B

Algorithme LLL

// 1ere étape : initialisations des variables
1 $b_1^* \leftarrow b_1$, $B_1 \leftarrow \langle b_1^*, b_1^* \rangle$
// 2eme étape : calculs de Gram Schmidt
2 **pour** $i \leftarrow 2$ **à** n **faire**
3 $b_i^* \leftarrow b_i$
4 **pour** $j \leftarrow 1$ **à** $i-1$ **faire**
5 $\mu_{i,j} \leftarrow \langle b_i, b_j^* \rangle / B_j$, $b_i^* \leftarrow b_i^* - \mu_{i,j} b_j^*$
6 $B_i \leftarrow \langle b_i^*, b_i^* \rangle$
// 3eme étape : k est une variable tel que : $b_1, b_2, \ldots, b_{k-1}$ sont réduits ; l'algorithme cherche à modifier b_k pour que $b_1, b_2, \ldots, b_{k-1}$ soient réduits ;
7 $k \leftarrow 2$
// 4eme étape : le vecteur b_k est modifié de manière adéquate de telle sorte que : $|\mu_{k,k-1}| \leq 1/2$ et les $\mu_{k,j}$ sont mis à jour pour $1 \leq j \leq k-1$;
8 **si** $|\mu_{k,k-1}| > 1/2$ **alors**
9 $r \leftarrow \lfloor \mu_{k,k-1} \rceil$, $b_k \leftarrow b_k - r b_{k-1}$
10 **pour** $j \leftarrow 1$ **à** $k-2$ **faire**
11 $\mu_{k,j} \leftarrow \mu_{k,j} - r\mu_{k-1,j}$
12 $\mu_{k,k-1} \leftarrow \mu_{k,k-1} - r$

// 5eme étape : la condition $||b_i^*||^2 > (3/4 - \mu_{i,i-1}^2) \, ||b_{i-1}^*||^2$ pour $1 < i \leq n$ est violée pour i=k. Les vecteurs b_k et b_{k-1} sont échangés et leurs paramètres sont mis à jour. k est

également décrémenté de 1 car seuls $b_1, b_2, \ldots, b_{k-2}$ sont réduits. Sinon b_k est modifié de manière adéquate de telle sorte que $|\mu_{k,k-1}| \leq ½$ pour $1 \leq j \leq k-2$ en conservant $|| b_i^* ||^2 > (3/4 - \mu_{i,i-1}^2) || b_{i-1}^* ||^2$ satisfaite. k est alors incrémenté car $b_1, b_2, \ldots, b_k$ est réduite.

```
13  si Bk<(3/4-μk,k-1²)Bk-1 alors
14      μ←μk,k-1, B←Bk+μ²Bk-1, μk,k-1←μ Bk-1/B, Bk←Bk-1Bk/B, Bk-1←B
15      échanger les vecteurs bk et bk-1
16      si k>2 alors
17          pour j←1 à k-2 faire
18              échanger μk,j et μk-1,j
19      pour i←k+1 à n faire
20          t←μi,k, μi,k←μi,k-1-μ t, μi,k-1←t+μk,k-1μi,k
21      k←max(2,k-1)
22      retourner à l'étape 8
23  sinon
24      pour l←k-2 à 1 faire
25          si |μk,l|>1/2 alors
26              r←⌊μk,l⌉, bk←bk-rbl
27              pour j←1 à l-1 faire
28                  μk,j←μk,j-rμl,j
29              μk,l←μk,l-r
30      k←k+1
31  si k≤ n alors
32      retourner à l'étape 8
33  sinon
34      retourner(b1,b2,...,bn) comme base réduite
```

Annexe C

Résultat cryptanalyse de Merkle-Hellman avec Maple

```
> restart;
//on construit le 6-uplet de base
> a:=[366,385,392,401,422,437];
                    a := [366, 385, 392, 401, 422, 437]

> s:=0;
                              s := 0

> for i from 1 to 6 do s:=s+a[i] od;
                              s := 366

                              s := 751

                              s := 1143
```

$s := 1544$

$s := 1966$

$s := 2403$

//on calcule la somme des éléments et on choisit le module plus grand que la somme (2403)

```
> m:=2405;
```

$m := 2405$

//t est le multiplicateur

```
> t:=61;
```

$t := 61$

//on calcule l'inverse de t par euclide étendu

```
> igcdex(t,m,'u','v');print(u,v);
```

1

276, -7

//u est l'inverse de t, fixé à 61.

```
> b:=map(x->t*x mod m,a);
```

$b := [681, 1840, 2267, 411, 1692, 202]$

```
> readlib(lattice);
```

proc() ... end proc

//on construit un sac à dos arbitraire :

```
> lattice([[1,0,0,0,0,-b[1]],[0,1,0,0,0,-b[2]],[0,0,1,0,0,-
b[3]],[
0,0,0,1,0,-b[4]],[0,0,0,0,1,-
b[5]],[0,0,0,0,0,+b[1]+b[3]]],'integer');
```

[[1, 0, 1, 0, 0, 0], [-2, 0, 3, -3, 1, -2], [0, 2, 1, 4, -1, -3], [-1, -3, 2, 1, -1, 0], [0, 0, 0, -2, -3, 2], [-2, 4, 2, 2, 2, 2]]

//le premier vecteur ligne est solution. « Lattice » est une fonction de Maple qui fait l'équivalent de l'algorithme LLL.

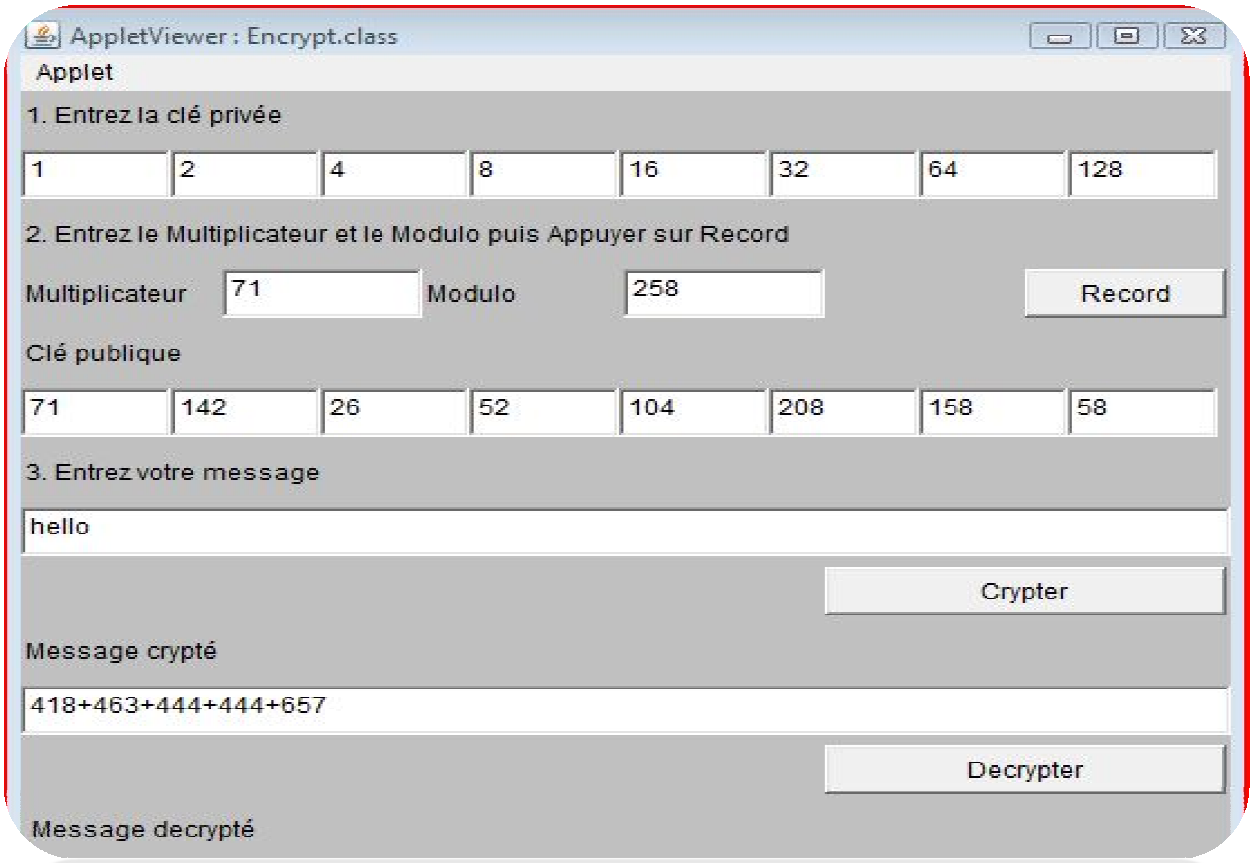
```
> lattice([[1,0,0,0,0,-b[1]],[0,1,0,0,0,-b[2]],[0,0,1,0,0,-
b[3]],[0,0,0,1,0,-b[4]],[0,0,0,0,1,-
b[5]],[0,0,0,0,0,+b[1]+b[2]]],'integer');
```

[[1, 1, 0, 0, 0, 0], [-2, 2, 1, -3, 1, -2], [2, -1, 3, 1, -1, 0], [0, 1, -1, -3, -2, 2], [-1, 1, 1, 2, -4, -1], [-2, 2, 1, -1, 2, 5]]

>

Annexe D

Applet Java pour coder et décoder un message



Explication :

On rentre une clé privée qui doit être super-croissante. On choisit le multiplicateur et le modulo et on clique sur « Record », ce qui nous donne la clé publique.

On écrit notre message et lorsque l'on clique sur « Crypter » cela nous donne le message crypté (calculé à partir de la clé publique). Puis lorsque l'on clique sur « Décrypter », on retrouve bien notre message (calculé à partir de l'inverse du multiplicateur (Algorithme d'Euclide), du modulo, et de la clé privée).

Nous avons essayé de coder l'algorithme LLL, en java, mais pour l'instant il ne fonctionne pas.

Bibliographie

Knapsack Cryptostem par Behdad ESFAHBOD (December 2001).
Applied Cryptography, Second Edition par John WHILEY & Sons.
Handbook for applied cryptography par A. Menezes, P. Van Oorschot, et S. Vanstone, CRC Press, 1996.
On breaking the iterated Merkle-Hellman public-key cryptosystem par Leonard M. Adleman
Etudes et implémentation de l'algorithme LLL par EL MOUTTAKI Mohamed et EL OUATI Youssef
Knapsack problem, algorithm and computer implementation par S. MARTELLO and P.TOTH
Codage, cryptologie et applications par Bruno Martin

Webographie

- http://fr.wikipedia.org/wiki/Probl%C3%A8me_du_sac_%C3%A0_dos
- http://www.math.ucsd.edu/~crypto/Projects/JenniferBakker/Math187/index.html
- http://www.personal.kent.edu/~rmuhamma/Algorithms/algorithm.html
- http://www.cs.bris.ac.uk/Teaching/Resources/COMS21101/lectures/se2-2/img24.html
- http://fr.wikipedia.org/wiki/Probl%C3%A8me_du_sac_%C3%A0_dos
- http://www.bibmath.net/crypto/complements/sacados.php3
- http://www-ath.cudenver.edu/%7Ewcherowi/courses/m5410/ctcknap.html
- http://interstices.info/jcms/c_19213/le-probleme-du-sac-a-dos
- http://www.enseignement.polytechnique.fr/informatique/ARCHIVES/IF/03/pi/poupard/index.html
- http://www.gymnase-verdon.vd.ch/branches/mathematique/cryptographie/nombres/merkle_hellexpl.htm
- http://www.montefiore.ulg.ac.be/~dumont/pdf/Crypto06.pdf
- http://www.math.u-bordeaux.fr/~bachoc/Enseignements/Cryptanalyse/LLL.pdf