Projet de Cryptographie

Mai 2020

PROBLÈME DU SAC À DOS

Kilian DUMAY
Raphaëlle GAUCHÉE

À rendre à M. Riccardo BRASCA

Table des matières

1.1 Introduction

1.2 Énoncé du problème

Les données du problème peuvent s'exprimer en termes mathématiques.

Soit A une liste d'objets x_i, $1 \leq i \leq n$. À chaque objet x_i, on attribue un poids w_i et une valeur p_i. La capacité maximale du sac à dos est notée W. L'objectif est de maximiser la valeur des objets sans dépasser le poids maximal :

$$\max_{x_i \ choisi} \sum_{i=1}^{n} p_i \qquad \text{et} \qquad \sum_{i=1}^{n} w_i \leq W$$

Ce problème est intéressant pour la cryptographie car c'est un problème NP complet, il est donc en principe compliqué de trouver une solution.

1.3 Préambule

1.3.1 Définition

Soit $A = \{a_1, a_2, ..., a_n\}$ une suite d'entiers. A est dit "super-croissante" si et seulement si $\sum_{i=1}^{k} a_i \leq a_{k+1}$

Exemple : $A = \{1, 3, 7, 12, 27, 55, 112, 253\}$

1.4 MERKLE-HELLMAN

MERKLE-HELLMAN est un des premiers cryptosystèmes asymétriques, défini par Ralph MERKLE et Martin HELLMAN en 1978. Il est, contrairement à RSA, à sens unique, c'est-à-dire que la clé publique est utilisée uniquement pour le chiffrement, et la clé privée uniquement pour le déchiffrement. Il ne peut donc pas être utilisé pour un protocole d'authentification. Ce cryptosystème repose sur le problème de la somme de sous-ensembles, un cas spécial du problème du sac à dos.

Pour ce cryptosystème, les clés sont des sacs :
(1) La clé privée contient entre autre un sac supercroissant dit "facile" solvable en temps polynomial et

(2) la clé publique est un sac à dos dit "dur" calculée à partir de la clé privée.

1.4.1 Pratique

1.4.2 Génération des clefs

On choisit une suite d'entiers super-croissante : $\{a_1, a_2, \cdots, a_n\}$.

On choisit un entier N tel que $\sum_{i=1}^{n} a_i \leq N$.

On choisit un entier $A < N$ tel que $\text{pgcd}(A, N) = 1$.

On obtient alors la clef secrète, utile au déchiffrement : $\big(N, A, \{a_1, a_2, \cdots, a_n\}\big)$.

On calcule ensuite les coefficients de la clef publique tels que $b_i \equiv Aa_i \pmod{N}$

Ainsi, on obtient la clef publique, utile au chiffrement : $\{b_1, b_2, \cdots, b_n\}$

1.4.3 Chiffrement

La clé publique permet de chiffrer des messages de longueur n. On considère un mot $w = \big(w_1, w_2, \cdots, w_n\big)$ de longueur n, tel que $w_i \in \{0, 1\}$. Alors :

$$c = \sum_{i=1}^{n} w_i b_i$$

est le message chiffré par la clé publique $\{b_1, b_2, \cdots, b_n\}$.

Le code en Pari GP pour le chiffrement :

```
\\ w est le mot binaire de longueur n qu'on veut envoyer.
\\ publique est la clef publique.

chiffrement(w, publique) = {
    my(c = 0);
    for(i = 1, #w,
        c+ = w[i] * publique[i];
    );
    return(c);
};
```

1.4.4 Déchiffrement

Soit le mot chiffré c. On pose $p \equiv A^{-1}c \pmod{N}$. On peut alors écrire, l'instance du problème du sac à dos :

$p = \sum_{i=1}^{n} x_i a_i$

qui a pour solution, $x_i = w_i$. Le détenteur de la clef privée $\big(N, A, \{a_1, a_2, \cdots, a_n\}\big)$ peut calculer p et résoudre en temps polynomial l'instance du sac à dos pour retrouver le message original $(w_1, w_2 \ldots, w_n)$.

Le code en Pari GP pour le déchiffrement :

```
\\ c est le chiffré
\\ privee est la clef privée
dechiffrement(c, privee) = {
    my(invA = Mod(privee[2], privee[1])^-1);
    my(p = invA * c);
    p = lift(p);

    \\ on a aussi que p = somme(a_i * b_i)
    \\ comme (a_i) est une suite super croissante,
    \\ on peut retrouver les b_i facilement

    x = vector(#privee[1]);

    my(n= #privee[3]); \\ le cardinal
    forstep(i=n, 1, -1,
        if(p >= privee[3][i],
            x[i] = 1; p-= privee[3][i],
            x[i] = 0;
        );
    );
    return (x);
}
```

On voit bien que la résolution se fait en temps polynomiale, en effet il suffit de parcourir une seule fois la liste des entiers de la liste super croissante dans

l'ordre décroissant pour avoir la solution. Comme la suite est super croissante si p est supérieur au plus grand entier de la liste alors cet entier est forcément présent puisque l'ensemble des autres entiers ne le dépassent pas.

Exemple : On choisit une suite d'entiers super-croissante : $\{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8\} = \{1, 3, 7, 19, 41, 72, 183, 333\}$.

On calule la somme des coefficients de la suite $\big(a_n\big)_{n\in\mathbb{N}}$: $\sum_{i=1}^{8} a_i = 669$. Et on choisit un entier N plus grand, on prend $N = 732$.

On choisit un entier $A < N$ tel que $\text{pgcd}(A, N) = 1$. Soit $A = 587$.

On obtient alors la clef secrète : $\big(\{1, 3, 7, 19, 41, 72, 183, 333\}, 732, 587\big)$.

On calcule les coefficients b_i de la clef publique avec $b_i \equiv Aa_i \pmod N$. Ainsi on obtient la clef publique $\{b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8\} = \{587, 297, 449, 173, 643, 540, 549, 27\}$.

On suppose que Bob veut envoyer le message suivant $w = 01001100$ à Alice.

Bob chiffre son message avec la clef publique telle que $c = \sum_{i=1}^{n} w_i b_i$. Ainsi, Bob envoie $c = 1480$ à Alice. En effet,

$w = 01001100$
$c = 0 \times 587$
$\quad + 1 \times 297$
$\quad + 0 \times 449$
$\quad + 0 \times 173$
$\quad + 1 \times 643$
$\quad + 1 \times 540$
$\quad + 0 \times 549$
$\quad + 0 \times 27$
$\quad = 1480$

Alice reçoit $c = 1480$ de la part de Bob. Elle calcule $p = A^{-1}c \pmod N$.

Sachant que $A^{-1} = 419$, on a $p = 419 \times 1480 = 620120 \equiv 116 \pmod{732}$

Alice résout ensuite le sac à dos, dit facile, $p = \sum_{i=1}^{n} x_i a_i$ avec un algorithme glouton et la clef privée $\{1, 3, 7, 19, 41, 72, 183, 333\}$ pour $p = 116$. Elle décompose $p = 116$ en sélectionnant le a_i le plus grand inférieur ou égal à 116. Puis elle recommence jusqu'à obtenir 0 :

$116 - 72 = 44$
$44 - 41 = 3$
$3 - 3 = 0$.

Donc Alice décode p tel que $p = (0, 1, 0, 0, 1, 1, 0, 0)$ où les "1" sont les marqueurs des a_i sélectionés. Alice obtient bien le message que Bob lui a envoyé.

1.5 Algorithme LLL

1.5.1 Définitions utiles

Un Réseau entier est un sous groupe discret de $\mathbb{Z}^n$. Un réseau possède une infinité de bases composées de vecteurs linéairement indépendants à coefficient entier.

Soient $n > 2$, et $b = (b_1, b_2, ..., b_n)$ une base d'un réseau entier.

Prenons $b^* = (b_1^*, b_2^*, ..., b_n^*)$ l'orthogonalisation de la base par le procédé de Gram Schmidt, défini par :

$b_1^* = b_1$

$\mu_{i,j} = \frac{<b_i, b_j^*>}{<b_j^*, b_j^*>}$ Pour $1 \leq j < i, \leq n$

$b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} \times b_j^*$ Pour $1 < i \leq n$

b est une base réduite si $|\mu_{i,j}| \leq \frac{1}{2}$ pour $1 \leq j < i \leq n$ et $||b_i^*||^2 \geq (\frac{3}{4} - \mu_{i,i-1}^2) \times ||b_{i-1}^*||^2$ pour $1 \leq i \leq n$

1.5.2 Cryptanalyse du code Merkle-Hellman

Soient $B = (b_1, b_2, ..., b_n)$ la clef publique d'un chiffre de Merkle-Hellman et c le mot chiffré.

On cherche $x = (x_1, x_2, ..., x_n)$ avec $x_i \in 0, 1$ tel que $< B, x >= c$.

On défini un réseau L, de base

$$\mathrm{V} = \begin{pmatrix} 1 & 0 & 0 & ... & -\mathrm{b}_1 \\ 0 & 1 & 0 & ... & -\mathrm{b}_2 \\ 0 & 0 & 1 & ... & -\mathrm{b}_3 \\ ... & & & & \\ ... & & & & \\ ... & & & & \\ 0 & 0 & 0 & ... & -\mathrm{b}_n \\ 0 & 0 & 0 & ... & c \end{pmatrix}$$

V est donc une matrice carré de taille $n + 1$

On peut alors calculer une base réduite de V, et regardez si un des vecteurs de la base est une solution.

1.5.3 LLL

L'algorithme LLL du nom de ses 3 créateurs est l'algorithme qui permet de calculer une base réduite, il se fait en 2 étapes.

La première consiste à calculer une base orthonormée, on utilise ici le procédé de Gram Schmidt.

```
\\ Soit b une base
LLL(b) = {

    my(n = #b);

    my(b_ortho, mu, B, i, j, k);

    b_ortho = vector(n); \\ la base orthonormée
    b_ortho[1] = b[1];


    \\ tableau simple
    B = vector(n);
    B[1] = scal(b_ortho[1], b_ortho[1]);

    \\ tableau de tableau
    mu = vector(n);
    for(i = 1, n,
        mu[i] = vector(n);
    );


    for(i = 2, n,
        b_ortho[i] = b[i];
        for(j = 1, i-1,
            mu[i][j] = scal(b[i], b_ortho[j])/ B[j];
            b_ortho[i] -=mu[i][j]*b_ortho[j];
        );
        B[i]= scal(b_ortho[i], b_ortho[i]);
    );
```

Une fois qu'on a une base orthonormée, on doit faire en sorte que les conditions de base réduite soient respectées. Pour cela on utilise un algorithme intermédiaire qui ajuste un vecteur :

```
RED(k, l, b, mu) = {
    my(r, j);
    if(abs(mu[k][l]) > 1/2,
        r = floor(0.5 + mu[k][l]);
        b[k] -=r*b[l];
        for(j =1, l-1,
            mu[k][j] -= r*mu[l][j];
        );
```

```
        mu[k][l] -=r;
    );
    return([b, mu]);
}
```

Et une fois le vecteur ajusté, on le compare au vecteur précédent, si les conditions ne sont pas respectées, on échange alors les vecteurs.

```
    \\ Suite LLL
    k = 2;

    my(mu_tmp, B_tmp, b_tmp, mu_k_tmp, t);

    while(k<=n,

        [b, mu] = RED(k, k-1, b, mu);

        if( (B[k]) <  ((3/4)-(mu[k][k-1] ^ 2)) *B[k-1] ,
            mu_tmp = mu[k][k-1];
            B_tmp = B[k] + mu_tmp^2 * B[k-1];
            mu[k][k-1] = mu_tmp*B[k-1]/B_tmp;

            B[k] = B[k-1]*B[k]/B_tmp;
            B[k-1] = B_tmp;

            b_tmp = b[k];
            b[k] = b[k-1];
            b[k-1] = b_tmp;

            if(k>2,
                for(j = 1, k-2,
                    mu_k_tmp = mu[k][j];
                    mu[k][j] = mu[k-1][j];
                    mu[k-1][j] = mu_k_tmp;
                );
            );

            for(i = k+1, n,
                t = mu[i][k];
                mu[i][k] = mu[i][k-1] - mu_tmp*t;
                mu[i][k-1] = t + mu[k][k-1]*mu[i][k];
            );
            k = max(2, k-1),


            \\ else
            forstep(l=k-2, 1, -1,
```

```
                [b, mu] = RED(k,l, b, mu);
            );
            k = k+1;
        );

    );
    return(b);

}
```