Logarithme discret

Florian Bourse

1 Présentation du problème

1.1 Logarithme discret

En cryptographie, la sécurité des données repose sur la difficulté de certains problèmes algorithmiques. Le calcul du logarithme discret fait parti de ces problèmes.

Nous travaillons dans $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$, pour $q = 2^{31} - 1$, nombre premier de Mersenne, c'est-à-dire que tous les résultats seront calculé modulo q. 7 est un générateur du groupe $(\mathbb{Z}_q \setminus \{0\}, \times)$, c'est-à-dire que pour tout élément x, il existe un exposant e tel que $x = 7^e$.

Le logarithme discret d'un nombre h dans une base g est le nombre e tel que $h = g^e$ si il existe. Par exemple, le logarithme discret de 49 en base 7 est 2 car $7^2 = 49$.

1.2 Représentation des nombres

Pour manipuler des grands nombres, la capacité du type `int` n'est pas suffisant. Nous allons donc utiliser des chaînes de caractères, qui en C sont des tableaux de `char` terminés par le caractère nul '\0'.

Pour éviter de recoder une table d'addition ou de multiplication à rallonge, nous allons nous simplifier la vie en utilisant la représentation en binaire des nombres.

Chaque nombre est donc une chaîne de 31 caractères '0' ou '1'. Par exemple, 0 est représenté par

```c
char zero[] = "0000000000000000000000000000000"
```

Pour simplifier la gestion de la mémoire, toutes les fonctions de calcul modifieront le tableau donné en entrée pour y mettre le résultat. Par exemple `add_to(a, b);` calcule la somme modulo q des nombres représentés par `a` et `b` et met le résultat dans `a`. Les fonctions de comparaisons ne modifient pas leur entrée et renvoient un booléen.

1.3 But du projet

Le but du projet est de calculer des logarithmes discrets en base 7 modulo $2^{31} - 1$.

Pour ce faire, nous allons implémenter la comparaison, l'addition, puis la multiplication, et enfin l'exponentiation des entiers modulo q, représenté par 31 caractères '0' ou '1'.

Une fois ces fonctions auxiliaires accomplies, le projet contient 3 niveaux de réussite :

1. Vérifier que `x` est bien le logarithme discret de `target0` ;
2. Calculer le logarithme discret de `target1` ;
3. Calculer le logarithme discret de `target2`.

Un algorithme naïf permet de trouver la deuxième réponse, mais ne sera pas assez efficace pour la troisième.

2 Opposé, addition, multiplication

Le nombre $2^{31} - 1$ a été choisi car il simplifie beaucoup l'implémentation. Lorsque l'on regarde son écriture un binaire, on se rend compte que parmi les nombres sur 31 bits, il n'y a que 0 qui possède 2 écritures.

On peut aussi remarquer que si on obtient une retenue en additionnant 2 nombres, elle représente le nombre 2^{31}, qui est donc égal à 1 modulo q.

Pour la multiplication, on peut utiliser la méthode russe pour éviter les problèmes de dépassement de capacité (car le produit de deux nombres de 31 bits peut faire jusque 62 bits). Pour multiplier un nombre a par $b = \sum_i b_i 2^i$, on peut effectuer les opérations suivantes :

$$\sum_i b_i (a \times 2^i)$$

où les $a \times 2^i$ peuvent être obtenus successivement en commençant à `tmp = a` en doublant `tmp` à chaque itération. On remarquera que les b_i sont la représentation en binaire de b.

Cette méthode peut être adaptée à l'exponentiation pour donner l'algorithme d'exponentiation rapide.

3 Calcul du logarithme discret

3.1 Algorithme naïf

Il s'agit de calculer successivement tous les 7^i et de s'arrêter quand on tombe sur la cible cherchée. Le logarithme discret de la cible est alors i.

3.2 Baby step, giant step

Pour éviter de parcourir tout l'ensemble dans le pire des cas, on fait un compromis entre le temps d'exécution et la mémoire utilisée. Pour un paramètre entier k, on découpe en k parties équilibrées l'espace de recherche $(\mathbb{Z}_q)$ $[0, i_1[, [i_1, i_2[, \ldots, [i_{k-1}, q[$, et on calcule les 7^{i_ℓ} et on les stocke dans un tableau.

Ensuite, on part de la cible t, et on la multiplie par 7 jusqu'à arriver à un des éléments du tableau. Si

$$7^j \times t = 7^{i_\ell}$$

alors le logarithme discret de t en base 7 est $i_\ell - j$.

Si on a découpé l'ensemble en 20 parties par exemple, la définition du tableau pour stocker les résultats des 7^{i_ℓ} pourrait être

```c
char giantsteps[20][32]
```

Avant d'implémenter cette solution, il faut se demander quel est le choix le plus judicieux pour k en analysant le pire des cas.