Décryptage de fichiers chiffrés

Sujet proposé par Andreas Enge

enge@lix.polytechnique.fr

Difficulté: moyen (**)

Ce projet est accompagné d'une page de suivi, accessible depuis
http://www.enseignement.polytechnique.fr/profs/informatique/Andreas.Enge/

1 Le contexte

Le problème du logarithme discret est, aux côtés du problème de la factorisation, à la base des systèmes cryptographiques à clef publique les plus utilisés en ce moment. Par exemple, les signatures numériques DSA et le système de chiffrement ElGamal utilisés dans OpenPGP reposent sur la difficulté de ce problème modulo un grand nombre premier p.

Nous rappelons le petit théorème de Fermat : pour un nombre premier p et tout entier g entre 1 et $p-1$, on a

$$g^{p-1} \bmod p = 1.$$

De plus, il y a des entiers (appelés *éléments primitifs*) g tels que toutes les puissances $1, g, g^2, \ldots, g^{p-2}$ sont différentes modulo p et recouvrent donc l'ensemble des entiers entre 1 et $p-1$. Par exemple, pour $p = 11$, on peut choisir $g = 2$ puisque les puissances successives de 2 modulo 11 sont 1, 2, 4, 8, $16 = 5$, 10, $20 = 9$, $18 = 7$, $14 = 3$ et 6 ; la puissance suivante est $12 = 1$, et le cycle se répète. Si on se donne maintenant un autre entier h entre 1 et $p-1$, il existe forcément un entier x (appelé le *logarithme discret de* h *en base* g) tel que

$$g^x \bmod p = h.$$

Vu le théorème de Fermat, cet entier est uniquement déterminé modulo $p-1$, et on peut donc le prendre entre 0 et $p-2$.

L'ancêtre de tous les cryptosystèmes fondés sur le problème du logarithme discret est l'échange de clef inventée par Diffie et Hellman [3], qui permet à deux utilisateurs Alice et Bob d'obtenir une clef de chiffrement commune. Alice choisit un exposant a au hasard entre 0 et $p-1$, calcule $A = g^a \bmod p$ et l'envoie à Bob. Bob agit de façon symétrique en prenant un entier aléatoire b et en envoyant $B = g^b \bmod p$ à Alice. Maintenant, les deux peuvent obtenir $g^{ab} \bmod p$: Alice procède en calculant $B^a \bmod p$, Bob calcule $A^b \bmod p$. Ils peuvent ensuite utiliser ce secret commun pour chiffrer la communication entre eux. Éve souhaite les espionner et a installé du matériel écoutant la ligne reliant les deux. Elle intercepte A et B. Clairement, si elle sait calculer des logarithmes discrets modulo p, elle retrouve a ou b et ainsi g^{ab}, et peut à partir de ce moment décrypter la communication chiffrée.

2 L'algorithme

Un algorithme trivial pour résoudre le problème consiste à tester toutes les valeurs successives de x jusqu'à ce qu'on tombe sur la bonne. Clairement, cette approche n'est plus faisable déjà pour des tailles assez modérées de p. Le but de ce projet est d'implanter un algorithme plus rapide qui permettra d'attaquer des instances dans lesquelles p a plusieurs dizaines de chiffres décimaux. Cet algorithme a été présenté dans [1]. Il procède en trois étapes : dans la première phase, on crée des relations entre quelques nombres bien choisis ; dans la deuxième, on résout un système linéaire pour trouver les logarithmes discrets de ces nombres particuliers ; dans la troisième phase, on calcule le logarithme discret du nombre qui nous intéresse.

2.1 Trouver des relations entre petits nombres premiers

Fixons une borne B. Soit $\mathcal{F} = \{p_1, \ldots, p_n\}$ l'ensemble comprenant tous les nombres premiers ne dépassant pas B ; on l'appelle la *base de facteurs*. Pour des valeurs a_i choisies aléatoirement entre 1 et $p-2$, nous pouvons maintenant calculer $g^{a_i} \bmod p$; si nous avons de la chance, cet entier se décompose en facteurs premiers contenus dans $\mathcal{F}$. Cela se teste en essayant de diviser par les éléments de $\mathcal{F}$. On obtient donc de temps en temps une relation de la forme

$$g^{a_i} \bmod p = \prod_{j=1}^{n} p_j^{e_{ij}}.$$

Passant aux logarithmes, on obtient l'équation linéaire

$$a_i = \sum_{j=1}^{n} e_{ij} \log p_j,$$

dans laquelle les logarithmes des éléments de la base de facteurs sont inconnus. L'étape suivante est de résoudre le système linéaire ainsi formé pour justement trouver ces logarithmes particuliers.

Exemple. Pour $p = 1019$, $g = 3$, $B = 11$ et donc $\mathcal{F} = \{2, 3, 5, 7, 11\}$, nous trouvons (modulo 1019) :

$$3^{66} \bmod 1019 = 770 = 2 \cdot 5 \cdot 7 \cdot 11, \quad 3^{72} = 880 = 2^4 \cdot 5 \cdot 11,$$
$$3^{90} = 660 = 2^2 \cdot 3 \cdot 5 \cdot 11, 3^{108} = 495 = 3^2 \cdot 5 \cdot 11, \quad 3^{128} = 539 = 7^2 \cdot 11,$$
$$3^{134} = 616 = 2^3 \cdot 7 \cdot 11$$

2.2 Algèbre linéaire

Le système linéaire issu de la phase précédente est à résoudre modulo $p-1$, ce qui découle du petit théorème de Fermat. On va procéder par élimination de Gauß. Notons que les entiers modulo $p-1$ forment un anneau : on peut additionner, soustraire et multiplier en faisant les opérations usuelles sur les entiers et en réduisant après modulo $p-1$. Malheureusement, il ne s'agit pas d'un corps, puisque tous les éléments ne sont pas inversibles : on peut inverser un élément a précisément si a et premier avec $p-1$. Dans ce cas, l'algorithme d'Euclide étendu entre a et $p-1$ trouve les coefficients de Bézout s et t tels que

$$sa + t(p-1) = 1;$$

modulo $p-1$, on a donc bien $s = a^{-1}$.

Le problème d'algèbre linéaire peut être résolu en travaillant séparément modulo les diviseurs premiers de $p-1$ et en réassemblant le résultat modulo $p-1$ par le théorème des restes chinois. Dans ce projet, nous allons nous faciliter la vie et prendre des p tels que $p-1=2q$ avec q premier (des premiers de Sophie Germain). En plus, au lieu de choisir un élément g primitif, nous allons prendre un élément g d'ordre q, de sorte que $g^q = 1 \bmod p$, et le cycle des puissances de g a une longueur q au lieu de $p-1$. Ainsi, on peut faire l'algèbre linéaire modulo q, et tout élément différent de 0 devient inversible. Cela permet d'appliquer l'élimination de Gauß telle qu'elle est connue pour l'algèbre linéaire sur les réels. D'abord, on met la matrice sous forme triangulaire (sauf permutation éventuelle des lignes). On choisit un *pivot* différent de 0 dans la première colonne, et en soustrayant des multiples de la ligne correspondante de toutes les autres lignes, on élimine toutes les entrées sauf le pivot. Puis, on choisit un pivot dans la deuxième colonne et on élimine toutes les entrées de la deuxième colonne sauf les deux dans les lignes de pivot sélectionnées jusqu'ici, et ainsi de suite. On travaille en même temps sur le côté droit du système. À la fin, on remonte le système triangulaire et trouve facilement une solution.

Exemple. Continuant l'exemple de la section précédente, il faut résoudre le système linéaire

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 4 & 1 & 0 & 1 \\ 2 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 2 & 1 \\ 3 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 66 \\ 72 \\ 89 \\ 106 \\ 128 \\ 134 \end{pmatrix}$$

modulo 509.

Comme premier pivot, on choisit le 1 dans la première ligne, et on élimine les entrées en dessous dans la première colonne en soustrayant 4 fois la première ligne de la deuxième, 2 fois la première ligne de la troisième et 3 fois la première ligne de la sixième. Cela donne

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 506 & 505 & 506 \\ 0 & 508 & 507 & 508 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 2 & 1 \\ 0 & 506 & 507 & 507 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 66 \\ 317 \\ 466 \\ 106 \\ 128 \\ 445 \end{pmatrix}$$

Dans la deuxième colonne, on choisit le pivot 506, dont l'inverse modulo 509 est 339. On multiplie la deuxième ligne par 339 pour avoir un pivot 1 et soustrait 508 fois cette ligne de la troisième, 1 fois cette ligne de la quatrième et 506 fois cette ligne de la dernière :

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 171 & 1 \\ 0 & 0 & 169 & 0 \\ 0 & 0 & 338 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 2 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 66 \\ 64 \\ 21 \\ 42 \\ 128 \\ 128 \end{pmatrix}$$

On peut d'hores et déjà éliminer la sixième ligne, qui fait doublon avec la cinquième ; de même, en choisissant le pivot 169 dans la troisième colonne, la quatrième ligne s'annulle. On obtient :

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 171 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 66 \\ 64 \\ 223 \\ 191 \end{pmatrix}$$

Remontant ce système triangulaire, on trouve $d = 191$, $c = 223$, $b = 64 - 171 \cdot 223 - 191 = 424$ et $a = 66 - 424 - 223 - 191 = 246$.

2.3 Calculer des logarithmes discrets

La troisième phase fait finalement intervenir le nombre h dont nous cherchons le logarithme discret. De façon similaire à la première étape, on cherche une relation de la forme

$$hg^a = \prod_{j=1}^{n} p_j^{e_j},$$

ce qui implique

$$\log h = \sum_{j=1}^{n} e_j \log p_j - a,$$

équation dans laquelle maintenant les $\log p_j$ sont connus.

Exemple. Pour $h = 101$, on obtient

$$hg^{11} = 245 = 5 \cdot 7^2$$

et donc

$$\log 101 = 2 \cdot 223 + 424 - 11 = 350,$$

et en effet on vérifie aisément que $3^{350} \bmod 1019 = 101$.

3 Travail demandé

Écrivez un programme `Dlog` qui prend comme argument un nombre premier p tel que $q = \frac{p-1}{2}$ est également premier, la base g d'ordre q du logarithme et le nombre A dont le logarithme est cherché. Le programme renverra le logarithme de A modulo p en base g. Par exemple :

```
# java Dlog 1019 3 101
350
```

Puis écrivez un programme `DH` pour cryptanalyser le système de chiffrement dérivé de l'échange de clef à la Diffie–Hellman. Le but est d'obtenir la clef secrète après avoir écouté l'information échangée lors de son établissement. Le programme prendra en argument le nombre premier p, la base g et les deux nombres $A = g^a$ et $B = g^b$ échangés par les deux participants. Il affiche la valeur de g^{ab}. Par exemple :

```
# java DH 1019 3 101 255
303
```

La page de suivi contient des liens vers des fichiers qui ont été chiffrés avec `gpg`. Les clefs utilisées sont dérivées des échanges Diffie–Hellman de la liste suivante. Pour décrypter le fichier `exemple3.txt.gpg`, exécutez la commande

```
gpg -o exemple3.txt -d exemple3.txt.gpg
```

Le logiciel vous demandera le mot de passe, qui n'est rien d'autre que la clef commune Diffie–Hellman, ici, `303`. Jusqu'où pouvez-vous aller ?

p	g	A et B
$10^3 + 19$	3	101
		255
$10^{10} + 259$	3	4819601888
		4585976725
$10^{20} + 763$	3	91735131463472784552
		71943364482341972729
$10^{30} + 1783$	2	170443004030887263495503209797
		796030796541354213297251504 43
$10^{40} + 17407$	2	7844874590252958799279234570696688991100
		7168212572107335215990016933997075374505
$10^{50} + 4483$	3	57729859988238372248223918064862692108319469581386
		71082576999006199136234704359201564628816601360849
$10^{60} + 15919$	2	915432078722549409670942815593824347180000684959197526496947
		625397773938548846252647070659717579933818502198566122612022

3.1 Conseils pratiques

On remarque que les phases 1 et 2 peuvent prendre beaucoup de temps, tandis que la phase 3 est relativement rapide. Prenez donc soin de ne pas répéter les deux premières phases chaque fois que le programme est appelé ; idéalement, le deuxième logarithme pour le même p et g prendra une fraction du temps nécessaire pour le premier. Cela peut être obtenu en stockant des résultats intermédiaires dans un fichier.

Le bon choix de la borne B et donc de la base des facteurs $\mathcal{F}$ s'avère délicat : si B est trop petit, alors la probabilité d'obtenir une relation est très faible. Si B est trop grand, la taille du système linéaire croît ainsi que le nombre de relations requis. La valeur théorique pour un temps de calcul minimal est donnée par $B \sim L_p^{1/2}$ avec

$$L_p = e^{\sqrt{\ln p \ln \ln p}}.$$

Le temps de calcul de l'algorithme est alors de l'ordre de $L_p^{3/2}$. En pratique, il faudra expérimenter et probablement choisir de plus petites valeurs pour B, ne serait-ce que pour faire tenir la matrice en mémoire.

Comme on a vu dans le petit exemple, les relations qu'on obtient ont tendance à être linéairement dépendantes. Pour pouvoir résoudre le système linéaire, il faudra donc créer plus de relations qu'il y a d'éléments dans la base des facteurs.

3.2 Structures de données

Pour manipuler les grands entiers dépassant les 32 ou 64 bits des machines actuelles, Java propose la classe `BigInteger`, qui contient toutes les opérations arithmétiques nécessaires pour le projet.

4 Extensions

Le sujet se prête à une multitude d'améliorations algorithmiques :

- Pour la recherche de relations, il est plus rapide (et même plus facile à programmer...) de parcourir systématiquement les puissances de g au lieu de choisir des puissances aléatoires g^a. Mais attention aux dépendances linéaires systématiques qu'une mauvaise implantation risque d'introduire !
- Un nombre a d'autant plus de chance de se décomposer en petits facteurs premiers qu'il est petit lui-même. L'approche suivante, appelée le *crible linéaire* et introduit dans [2], essaie d'en profiter. Soient $H = \lceil \sqrt{p} \rceil$ et $J =$

$H^2 \bmod p = H^2 - p$. Pour c_1, c_2 petits, on a

$$(H + c_1)(H + c_2) \bmod p = J + (c_1 + c_2)H + c_1c_2$$

de l'ordre de $\sqrt{p}$. Ce nombre a donc plus de chance de fournir des relations que les g^a qui sont en général de l'ordre de p. En revanche, il faut inclure les $H + c_1$ dans la base des facteurs. En théorie, le temps de calcul devient de l'ordre de $L_p^{3\sqrt{2}/4}$; mais est-ce que cette approche gagne en pratique ?
- On peut faire l'algèbre linéaire pour des p tels que $p-1$ n'est plus le double d'un premier. Pour ce faire, on factorise $p - 1 = p_1 \cdots p_k$, résout le système linéaire modulo tous les p_i et remonte la solution par le théorème des restes chinois. En effet, l'algorithme Euclidien permet de trouver des entiers e_i tels que $e_i \bmod p_i = 1$ et $e_i \bmod p_j = 0$ pour $j \neq i$. Si on connaît alors les valeurs $a_i = a \bmod p_i$ d'un entier a modulo tous les p_i, on peut le reconstruire comme $a = \sum a_i e_i$. Si $p-1$ a des facteurs multiples, la reconstruction devient nettement plus compliquée.
- Toute idée est bonne si elle permet d'aller plus loin !

Références

[1] Leonard Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, pages 55–60, New York, 1979. IEEE.

[2] Don Coppersmith, Andrew M. Odlyzko, and Richard Schroeppel. Discrete logarithms in $GF(p)$. *Algorithmica*, 1 :1–15, 1986.

[3] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6) :644–655, November 1976.