Langage C et aléa, séance 4

École des Mines de Nancy, séminaire d'option Ingénierie Mathématique

Frédéric Sur
http://www.loria.fr/~sur/enseignement/coursCalea/

1 La bibliothèque GMP

Nous allons utiliser pendant cette séance une bibliothèque de calcul sur les nombres avec une précision arbitraire. Il s'agit de la *GNU Multiple Precision Arithmetic Library* (GMP).

Sa page web est ici : `http://gmplib.org/`

Sur la page web du cours figurent la bibliothèque déjà compilée (`libgmp.a`), le fichier d'en-tête correspondant (`gmp.h`) ainsi que la documentation (`gmp-man-4.2.1.pdf`). Copiez `libgmp.a` et `gmp.h` dans le répertoire où vous allez écrire les codes-sources de cette séance. Vous pouvez regarder avec votre éditeur de texte ce que contient `gmp.h`.

Pour utiliser GMP, il faut inclure le fichier d'en-tête par :

```
#include"gmp.h"
```

et compiler votre programme (`essai.c`) par :

```
gcc essai.c -L. -lgmp -o essai.exe
```

où `-L.` signifie que la bibliothèque est à chercher dans le répertoire courant (désigné par `.`) et `-lgmp` qu'il faut utiliser la bibliothèque (*library*) `libgmp`.

Ce qui nous intéresse dans la documentation est la section 5 (pages 29 à 37) concernant les entiers de longueur arbitraire.

Un entier est déclaré par `mpz_t nom_entier;` et il faut obligatoirement l'initialiser à l'aide de `mpz_init(integ);` avant de l'utiliser.

Question : à quoi correspond ce type `mpz_t` ? (cherchez dans `gmp.h`)

Voici le plan de la partie de la documentation de GMP qui nous intéresse :

- section 5.1 : comment déclarer un entier de type `mpz_t`,
- section 5.2 : comment affecter une valeur à un entier de type `mpz_t`,
- section 5.3 : comment combiner initialisation et affectation,
- section 5.4 : fonctions de conversion (pas forcément utiles dans cette séance),
- section 5.5 : fonctions arithmétiques sur les entiers `mpz_t`,
- section 5.6 : fonctions relatives à la division entre entiers,
- section 5.7 : fonctions exponentielles sur les entiers,
- section 5.8 : extraction de racines (pas utile ici),
- section 5.9 : fonctions de théorie des nombres (génération de premiers, tests de primalité, pgcd),
- section 5.10 : comparaison entre entiers de type `mpz_t`.

Le reste ne sera pas utile aujourd'hui, sauf peut-être la section 5.13 pour ceux qui avancent vite sur la génération de nombres aléatoires.

2 La méthode de cryptographie RSA

La méthode de chiffrement RSA a été proposée en 1978 par Rivest, Shamir et Adleman. Il s'agit d'une méthode à clef publique : tout le monde peut crypter des messages à l'aide de la clef publique, mais seul celui

qui possède la clef privée de déchiffrement peut décrypter. Concrètement, pour qu'une personne utilise ce système, elle doit créer un couple (clef publique, clef privée), puis distribuer la clef publique à ses interlocuteurs. Ceux-ci crypteront leurs messages à l'aide la clef publique, que la personne pourra décrypter à l'aide de sa clef privée (qu'elle est la seule à détenir).

La méthode RSA est basée sur des propriétés des groupes finis. Comme on va le voir, sa robustesse repose principalement sur la difficulté de factoriser des entiers de grande taille.

2.1 Principe

Soient p et q deux nombres premiers[1] (secrets), et $n = pq$.

Soit c un élément inversible[2] du groupe $\mathbb{Z}/\phi(n)\mathbb{Z}$, où $\phi(n)$ est l'indicatrice d'Euler de n. Dans ce cas on rappelle que $\phi(n) = (p-1)(q-1)$.

Les entiers c et n sont publics, et c est appelé clef de chiffrement.

Le chiffrement d'un message[3] M est : $C(M) = M^c \mod n$.

Le déchiffrement nécessite de connaître l'inverse d de c dans $\mathbb{Z}/\phi(n)\mathbb{Z}$. L'élément d est appelé clef privée. Le déchiffrement d'un message chiffré M' s'effectue par $D(M') = M'^d \mod n$.

Il s'agit effectivement d'un déchiffrement car :

$$D(C(M)) = C(M)^d = M^{c \cdot d} = M^{1+k \cdot \phi(n)} = M \mod n.$$

Cette dernière égalité est basée sur le théorème de Lagrange appliqué au groupe $\mathbb{Z}/\phi(n)\mathbb{Z}$: si M et n sont premiers entre eux, alors (par définition de $\phi(n)$) $M \in \mathbb{Z}/\phi(n)\mathbb{Z}$ et donc $M^{\phi(n)} = 1 \mod n$. Dans ce cas il est immédiat que $M^{1+k \cdot \phi(n)} = M \mod n$. Le cas où M et n ne sont pas premiers entre eux est un peu plus technique.

2.2 Deux questions à résoudre avant d'aborder la suite

1. Comment trouver une clef de chiffrement c connaissant $\phi(n)$? Indication : souvenez-vous du théorème de Bezout et de la notion de pgcd.
2. Comment générer la clef de déchiffrement d à partir de c et $\phi(n)$? Indication : cherchez dans la documentation de GMP l'algorithme d'Euclide étendu.

2.3 Remarques

La robustesse de RSA repose sur le fait que pour inverser c modulo $\phi(n)$, il faut d'abord déterminer la valeur de $\phi(n)$, ce qui revient à factoriser n. À l'heure actuelle on ne connaît pas d'algorithme efficace pour factoriser un entier (mais un tel algorithme existe peut-être !). On choisit p et q de manière à ce que leur longueur soit au moins de l'ordre de 512 bits (i.e. p et q de l'ordre de 2^{512}). Ceci permet de résister aux tentatives de factorisations connues sur les ordinateurs actuels. Mais ceci n'assure évidemment pas que les données codées pourront rester secrètes dans le futur.

D'autre part, l'implémentation effective de RSA nécessite un certain nombre de précautions pour limiter les possibilités d'attaques : choix de la clef de déchiffrement d assez longue, bonne représentation du message à chiffrer...

[1] Remarquons que l'utilisation de RSA nécessite de disposer d'un générateur de nombres premiers.

[2] i.e. il existe $d \in \mathbb{Z}/\phi(n)\mathbb{Z}$ tel que $c \cdot d = 1 \mod \phi(n)$.

[3] supposé appartenir à $\mathbb{Z}/n\mathbb{Z}$: il "suffit" de représenter une chaîne de caractère par un nombre dans $\mathbb{Z}/n\mathbb{Z}$; en fait cette représentation est un problème à part entière si on veut assurer la robustesse effective de la méthode.

Exercice 1 (RSA)

Écrivez des fonctions permettant de crypter et décrypter selon la méthode RSA, à l'aide de la bibliothèque GMP. Faites des essais. Vous pouvez coder une chaîne de caractère de manière naïve par : 'blanc' $\leftarrow 00$, 'a'$\leftarrow 01$, 'b'$\leftarrow 02$, etc.

À toute fin utile, voici un couple de premiers possibles, avec une clef de chiffrement très simple :
$p = 378668090616600572642192533 97$
$q = 1152921504606846803$
$c = 3$

Utilisez les générateurs de nombres premiers de GMP pour créer d'autres clefs.

Petit jeu : distribuez votre n et c (par mail) à vos voisins, et demandez-leur de vous envoyer des messages chiffrés. Déchiffrez-les.

---

3 Méthode ρ de Pollard

Le nombre d'opérations pour trouver un facteur premier d'un entier n par la méthode du crible d'Ératosthène nécessite dans le pire cas de l'ordre de $\sqrt{n}$ opérations. Nous allons voir un algorithme probabiliste plus efficace, appelé *méthode ρ de Pollard*.

3.1 Algorithme

Voici la description de la méthode ρ en « pseudo-code » (f est ici une fonction « bien choisie » que l'on spécifiera plus loin) :

1. $x = 2, y = 2\,; d = 1$.
2. Tant que $d = 1$:
   (a) $x \leftarrow f(x) \mod n$
   (b) $y \leftarrow f(f(y)) \mod n$
   (c) $d \leftarrow \text{pgcd}(x - y, n)$
   (d) Si $d > 1$ alors retourner d

Si la valeur de retour est $d < n$, alors on a trouvé un diviseur de n, et on peut éventuellement recommencer avec n/d pour trouver d'autres diviseurs.

Par contre si la valeur de retour est $d = n$, on est dans un cas d'échec où on n'a pas identifié de diviseur strict de n. On peut recommencer en modifiant la fonction f ou les valeurs de x et y.

La fonction f est choisie de manière à ce que les itérés successifs x et y par f soient assimilés à des nombres aléatoires répartis uniformément à la fois dans $\mathbb{Z}/n\mathbb{Z}$ et dans $\mathbb{Z}/p\mathbb{Z}$ où p est un diviseur premier de n inférieur à $\sqrt{n}$. En pratique, on peut choisir $f(X) = X^2 + 1$.

3.2 Justification

Voici un argument heuristique pour voir en quoi le cas $d = n$ est peu probable (et donc en quoi l'algorithme donne une réponse généralement intéressante).

Soient k éléments $x_1, x_2, \ldots x_k$ tirés uniformément dans $\mathbb{Z}/n\mathbb{Z}$. Quelle est la probabilité p_k pour que deux d'entre eux soient égaux ? La probabilité pour qu'ils soient tous distincts est :

$$1 - p_k = \frac{n(n-1)\ldots(n-(k-1))}{n^k} = \prod_{i=1}^{k-1} (1 - i/n)\,.$$

En utilisant l'approximation $\log(1-x) \sim -x$ pour x proche de 0 (donc ici k petit devant n) :

$$1 - p_k \sim \exp\left(\frac{-k(k-1)}{2n}\right).$$

D'où :

$$k \sim \sqrt{-2\log(1-p_k)} \cdot \sqrt{n}.$$

Autrement dit, le nombre d'éléments à tirer au hasard pour avoir une chance raisonnable (par exemple $p_k > 1/2$) d'obtenir deux éléments égaux est de l'ordre de $\sqrt{n}$ (et même de l'ordre de $1.18\sqrt{n}$ pour $p_k = 1/2$).

Revenons à la méthode ρ de Pollard. Rappelons que la fonction f est choisie de manière à ce que les x et y soient répartis uniformément dans $\mathbb{Z}/n\mathbb{Z}$ et dans $\mathbb{Z}/p\mathbb{Z}$ où p est un diviseur inférieur à $\sqrt{n}$ de n. Supposons que $d = n$. Alors $x - y = 0 \mod n$, soit $x = y \mod n$. Donc le nombre d'itérations de l'algorithme est de l'ordre de $\sqrt{n}$.

Mais comme $x = y \mod n$, on a aussi $x = y \mod p$. Donc le nombre d'itérations pour que l'algorithme retourne $d = n$ est de l'ordre de $\sqrt{p}$.

Comme $\sqrt{p} < \sqrt[4]{n} << \sqrt{n}$, il est peu probable que $d = n$. (C'est une argument heuristique !)

Exercice 2 (Méthode ρ de Pollard)

Implémentez cet algorithme, et testez-le en essayant de factoriser l'entier n intervenant dans le codage RSA. Essayez différentes tailles.

Suite du petit jeu.

1. Décodez les deux messages secrets suivants. Que constatez-vous ?
   (a) 140307796247125975466605573869 68
   avec la clef publique : $n = 32451855365844265714093793686 7027, c = 49908343$
   (b) 212298764044365943343628062129078
   avec la clef publique : $n = 32452072005349547619863710138 3729, c = 349358401$
2. Cassez la clef RSA de votre voisin. Vous connaissez ses c et n qui sont publics, essayez de décoder le message que cherche à lui envoyer un troisième intervenant.

Tracez le graphe du temps de factorisation par la méthode de Pollard en fonction de la taille des nombres premiers intervenant dans la clef RSA. Comparez au crible d'Ératosthène.

---

4 Indications bibliographiques

Pour la justification de l'algorithme de Pollard, et pour des détails sur ce qu'on n'a pas le temps de traiter ici, consultez : P. Naudin et C. Quitté, *Algorithmique algébrique*, Masson 1992.

L'article de Wikipedia sur RSA est intéressant, et donne des indications d'attaques possibles :
`http://fr.wikipedia.org/wiki/Rivest_Shamir_Adleman`

La section 16 de la documentation de GMP donne une idée des problèmes algorithmiques soulevés par le calcul sur les grands entiers.