Projet Algo. S.D. (2015/2016)
Cryptographie à clé publique (RSA) *

A rendre le mercredi 20 janvier 2016

Formation Ingénieurs ITII - CNAM Alsace

Ce projet est à réaliser en binôme. Le langage de programmation à utiliser est le C. Il faudra rendre un rapport imprimé avec une description du travail réalisé, des résultats obtenus ainsi que de l'adéquation des résultats obtenus avec les résultats attendus. Ce rapport contiendra 5 pages maximum et le code complet sera fourni en annexe. On s'attachera à soigner la présentation du code afin qu'il soit le plus lisible possible. Il ne suffit pas d'écrire le code d'une fonction, il faut expliquer (i.e. commenter) les choix que ce soit au niveau de la structure de données ou de l'algorithme choisi(e).

1 Introduction

On va construire un programme permettant de simuler une communication chiffrée utilisant l'algorithme RSA (Rivest-Shamir-Adleman). Après avoir rappelé le principe de fonctionnement du cryptage, on détaille les fonctions à implanter.

2 Principe de fonctionnement du cryptage RSA

2.1 Génération d'un couple d'entiers

On présente un procédé pour générer aléatoirement un couple (clef publique, clef privée). On trouvera une présentation détaillée des principes de l'algorithme de cryptographie RSA ici : `http://fr.wikipedia.org/wiki/Rivest_Shamir_Adleman`.

1. Choisir p et q, deux nombres premiers distincts ;
2. Noter n leur produit, appelé "module de chiffrement" : $n = p \times q$;
3. Calculer l'indicatrice d'Euler de n : $\phi(n) = (p-1) \times (q-1)$;
4. Choisir e, un entier premier avec $\phi(n)$, appelé "exposant de chiffrement" ;
5. Comme e est premier avec $\phi(n)$, il est, d'aprés le théorème de Bachet-Bezout, inversible c'est-à-dire qu'il existe un entier d tel que $e \times d = 1 \ mod \ \phi(n)$. d est l'exposant de déchiffrement.

On appelle habituellement le couple (n, e) la clef publique (celle qui sert à encoder le message) et (n, d) la clef privée (celle qui sert à décoder le message crypté).

2.2 Codage

Si M est un entier inférieur à n représentant un message, alors le message chiffré sera représenté par $C = M^e \ mod \ n$.

*librement inspiré d'un projet proposé à Polytech' Paris-Sud

2.3 Décodage

Pour déchiffrer C, on utilise d, l'inverse de e modulo $\phi(n)$ et on calcule $M' = C^d \ mod \ n$.

3 Opérations arithmétiques de base à implanter

On va commencer par implanter des opérations arithmétiques sur les entiers :

1. déterminer si un entier n est premier ou non ;
2. engendrer un entier n premier avec un entier donné p ;
3. calculer l'inverse $mod \ n$ d'un entier donné x (il s'agit de trouver un entier y tel que $x \times y = 1 \ mod \ n$) ;
4. calculer l'exponentielle modulaire $x \mapsto x^d \ mod \ n$ efficacement (et sans débordement)

On commencera par implanter et valider ces opérations sur les types `int` et `long`. On pourra alors tester l'exemple présenté sur `wikipedia` dans lequel on choisit $p = 3$, $q = 11$. Le produit n vaut donc $3 \times 11 = 33$ et $\phi(n) = 20$. On choisit $e = 3$ (qui est bien premier avec $\phi(n) = 20$) comme exposant de chiffrement. L'exposant de déchiffrement $d = 7$ est l'inverse de 3 modulo 20. On peut vérifier aisément que $e \times d = 3 \times 7 = 1 \ mod \ 20$. Le message à coder est $M = 4$. Il s'agit de coder ce message, puis de le décoder et de vérifier qu'on obtient bien le message initial.

4 Opérations sur les chaînes de caractères

Il faut également proposer un mécanisme d'encodage d'un message. On supposera qu'un message est une chaîne de caractères stockée dans un fichier. Il s'agira d'extraire le message du fichier et de le découper pour qu'il soit représentable par une suite d'entiers tous inférieurs à n.

Supposons que $n > 255 = 2^8 - 1$. Dans ce cas, on peut découper le message caractère par caractère. Chaque caractère est alors représenté par son code ASCII. Evidemment, si n est beaucoup plus grand, on peut regrouper certains caractères entre eux. Si on fait des paquets de p caractères, il est nécessaire que l'entier n choisi soit supérieur à $2^{(8 \times p)} - 1$.

5 Simulations d'attaques et performances

Une fois un prototype mis en place, on cherchera à implanter des attaques visant à décoder un message sans connaître la clé privée. On pourra supposer certaines données du problème connues et on évaluera les informations pertinentes pour casser rapidement le code et donc décoder un message. Par exemple, connaître n ne sert à rien, par contre connaître sa décomposition en 2 facteur p et q peut rendre l'attaque plus facile.

6 Extensions

L'implantation basée sur les types entiers `int` et `long` ne permet pas de choisir des nombres p et q suffissament grands pour rendre inefficaces les attaques. Afin de pouvoir améliorer la robustesse de l'algorithme, on modifiera l'implantation pour utiliser la bibliothèque GMP de calculs sur les grands entiers (représentés par le type `mpz_t`).

Les opérations suivantes pourront être utiles :

initialisation de i	mpz_init(i)
affectation (entier signé) i=1	mpz_set_si(i,1)
affectation (entier non signé) i=1	mpz_set_ui(i,1)
`resultat` = `arg1` + `arg2`	mpz_add(resultat, arg1, arg2)
`resultat` = `arg1` - `arg2`	mpz_sub(resultat, arg1, arg2)
`resultat` = `arg1` × `arg2`	mpz_mul(resulat, arg1, arg2)
$(x, y) \mapsto x^y$	réutiliser la fonction puissance faite en TP
test du signe de n (renvoie 0 si n est nul)	mpz_sgn(n)
comparaison de x et y (renvoie un entier)	mpz_cmp(x,y)
quotient de la division de `arg1` par `arg2`	mpz_cdiv_q(resultat, arg1, arg2)
`resultat` = `arg1` mod `arg2`	mpz_mod(resultat, arg1, arg2)

A Calcul de l'inverse de p modulo n

On cherche à calculer l'inverse q du nombre p modulo n. Cela revient à chercher un entier q tel que $p \times q \ = \ 1 \ mod \ n$. Pour cela, on pourra utiliser la fonction `inverse` donnée ci-dessous.

```
// version simplifiee de l'algorithme d'Euclide etendu
long invers(long r, long u, long v, long r1, long u1, long v1)
{
  if (r1==0) return u;
  else return (invers(r1,u1,v1,r - (r/r1)*r1, u - (r/r1)*u1, v-(r/r1)*v1));
}

long inverse(long p, long n) // calcul de l'inverse p modulo n
{
  return invers(p,1,0,n,0,1);
}
```