Université de Nice-Sophia Antipolis — Deug MIAS-MI 1
Algorithmique & Programmation — 2000-2001

Projet

RSA - Les entiers longs

But :
* implanter une classe `EPA` pour manipuler des entiers positifs de taille arbitraire,
* à l'aide de la classe `EPA`, implanter le système de cryptographie **RSA**, du nom de ses inventeurs : Rivest, Shamir et Adleman.

Que faut-il faire

Vous devrez rendre votre projet le mercredi 6 juin 2001. Ce travail consistera en un ensemble de fichiers contenant le code source de votre programme (.java) et un rapport de quelques pages édité sous Word (.doc). Outre la complétude et la robustesse de votre programme, on insistera sur la lisibilitè et la compréhensibilitè du code (indentation, aération, choix des identificateurs, commentaires dans le code, ...).

Vous remettrez votre programme sous forme électronique. Vous fournirez également votre rapport sous forme électronique accompagné si possible d'une version imprimée.

Quoi qu'il en soit n'oubliez pas de mettre vos noms et de garder plusieurs sauvegardes de votre projet. N'accordez que peu de confiance aux disquettes.

Bonne chance !

1 Introduction

1.1 Cryptographie à clef privée

Dans les modèles de cryptographie classiques, dits à clef privée, Alice et Bob choisissent secrètement une clef k qui définit des règles de chiffrement E_k (*Encryption* avec la clef k) et de déchiffrement D_k (*Decryption* avec la clef k).
Dans ces systèmes, une même clef est utilisée pour le chiffrement et le déchiffrement. Le défaut majeur des systèmes à clef privée est qu'ils nécessitent la communication préalable de la clef k par un canal sûr avant la transmission du message chiffré.

1.2 Cryptographie à clef publique

Le principe des systèmes à clef publique est de publier la règle de chiffrement E_k, en s'assurant qu'il est impossible de déduire de E_k la règle de déchiffrement D_k.
On s'affranchit ainsi de la necessité de communiquer une clef secrète par un canal sécurisé.
Ce type de système peut se comparer à la situation où Alice dépose un message dans une boîte aux lettres blindée dont seul Bob possède la clef.

1.3 RSA

Le chiffrement RSA repose sur l'impossibilité *pratique* de décomposer un nombre suffisamment grand en produit de ses facteurs premiers. A ce jour, on n'a pas trouvé d'algorithme polynomial permettant la factorisation d'un entier quelconque.

2 Mise en oeuvre de RSA

Voici comment fonctionne RSA :

1. Bob engendre deux grands nombres premiers p et q
2. Bob calcule $n = pq$ et $\phi(n) = (p-1)(q-1)$
3. Bob choisit un b aléatoire $(1 < b < \phi(n))$ tel que $pgcd(b,\phi(n)) = 1$
4. Bob calcule $a = b^{-1} \bmod \phi(n)$, i.e. l'entier a tel que $ab = 1 \bmod \phi(n)$
5. Bob publie n et b dans un répertoire

On peut alors définir les fonctions de chiffrement E_k et de déchiffrement D_k suivantes :

$$E_k(x) = x^b \bmod n$$

$$D_k(y) = y^a \bmod n$$

Nous ne démontrerons pas que le déchiffrement du message chiffré retourne le message original et nous admettrons que nous avons bien :

$$D_k(E_k(x)) \equiv x \bmod n$$

Nous obtenons donc un système à clef publique $K = (n,p,q,a,b)$ dans lequel :
- les valeurs n et b sont publiques
- les valeurs p, q et a sont secrètes

Il apparait que le chiffrement RSA repose sur des calculs arithmétiques simples (multiplication, quotient et reste, puissance...). Comme nous l'avons dit précédemment, le chiffrement RSA est basé sur la difficulté de la factorisation des grands entiers. Les `int` de Java sont trop petits pour permettre une implantation sûre de RSA. Il est donc nécessaire de créer une structure de donnée qui nous permette de manipuler les grands entiers.
Bien que Java procure une classe appelée `BigInteger`, il vous est demandé, pour ce projet, de créer votre propre classe pour manipuler les grands entiers, ce qui constitue la plus grande partie du travail à effectuer pour l'implantation du système RSA.

3 Représentation des entiers longs - la classe `EPA`

A l'adresse :

`http://www-sop.inria.fr/tick/personnel/Yannis.Bres/Teaching/AlgoProg/`

vous trouverez :

- le squelette de la classe EPA qui vous permettra de gérer les grands entiers
- la documentation des méthodes que vous devez écrire
- un système qui vous permettra de tester votre programme progressivement
- à qui écrire en cas de problème

3.1 Entiers et notion de base

Dans le langage naturel, nous représentons les nombres par une suite de chiffres compris entre 0 et 9. Ces nombres sont représentés en base 10. Ainsi, la succession de chiffres «78953» représente le nombre :

$$3 \cdot 10^0 + 5 \cdot 10^1 + 9 \cdot 10^2 + 8 \cdot 10^3 + 7 \cdot 10^4$$

Par la même processus, la succession de chiffres «243» exprimée en base 8 représente le nombre :

$$3 \cdot 8^0 + 4 \cdot 8^1 + 2 \cdot 8^2 = 3 + 4 \cdot 8 + 2 \cdot 64 = 163$$

De manière générale, à partir d'une série de chiffres c_i exprimés dans une base B, nous sommes capables de calculer la valeur de l'entier correspondant n :

$$n = c_0 \cdot B^0 + c_1 \cdot B^1 + c_2 \cdot B^2 + ...$$

Attention, ce nombre s'écrit à l'envers dans le langage naturel :

$$...c_2 c_1 c_0$$

Nous allons représenter les entiers longs par un tableau de `int` qui représentera les chiffres de notre nombre exprimés dans une certaine base. Pour des raisons algorithmiques, que vous découvrirez en avançant dans votre projet, nous allons exprimer ces entiers longs dans une base B qui sera une puissance de 10 (10, 100, 1000...). D'autre part, il faut que la base soit supérieure ou égale à $2^8 = 256$ (pour pouvoir stocker au moins un octet par chiffre - voir le codage RSA) et inferieure à $\sqrt{2^{31}} \simeq 46341$ (pour la multiplication, il faut absolument que B^2 soit représentable par un `int`) ce qui signifie que vous avez le choix pour B entre 1000 et 10000.

3.2 Décomposition de nombres en série de chiffres

Dans la section précédente, nous avons appris à calculer la valeur d'un nombre à partir de sa représentation en série de chiffre exprimée dans une certaine base. Nous voulons maintenant calculer la série de chiffres en base B qui représente un entier n. Pour cela, reprenons nos exemples précédents. Le nombre 78953 en base 10 se décompose de la manière suivante :

$$c_0 = 78953/10^0 \bmod 10 = 3$$
$$c_1 = 78953/10^1 \bmod 10 = 5$$
$$c_2 = 78953/10^2 \bmod 10 = 9$$
$$c_3 = 78953/10^3 \bmod 10 = 8$$
$$c_4 = 78953/10^4 \bmod 10 = 7$$
$$c_4 c_3 c_2 c_1 c_0 = [78953]_{10}$$

De même, 163 en base 8 :

$$c_0 = 163/8^0 \bmod 8 = 3$$
$$c_1 = 163/8^1 \bmod 8 = 4$$
$$c_2 = 163/8^2 \bmod 8 = 2$$
$$c_2 c_1 c_0 = [243]_8$$

De manière générale, la décomposition d'un entier n en chiffres c_i exprimés dans une base B se calcule par :

$$c_i = n/B^i \bmod B$$

Dans la classe `EPA` Vous pouvez donc construire un objet `EPA` à partir d'un entier n. Pour cela, il suffit d'allouer un tableau de la bonne taille (à quelque chose près, c'est le logarithme de l'entier n en base B) et de remplir les cases du tableau comme indiqué ci-dessus.

3.3 Addition de deux entiers longs

Lorsque vous étiez à l'école primaire, vous avez appris à faire une addition de deux nombres à plusieurs chiffres en posant l'opération et en faisant plusieurs petites additions de chiffres. Nous allons appliquer le même principe pour calculer la somme de deux grands entiers.

$$\begin{array}{cccccc} & {\scriptstyle 1} & & & {\scriptstyle 1} & \\ & & 1 & 0 & 2 & 3 \\ + & & 9 & 7 & 4 & 8 \\ \hline & 1 & 0 & 7 & 7 & 1 \end{array}$$

Il suffit d'ajouter les deux chiffres d'une même colonne et de prendre en compte la retenue. L'addition présentée ci-dessus est une addition en base 10. Voici comment se déroule une addition de deux nombres $M = ...m_2m_1m_0$ et $N = ...n_2n_1n_0$ en base B.

$$\begin{array}{cccccc} {\scriptstyle r_{i+1}} & & {\scriptstyle r_i} & & {\scriptstyle r_{i-1}} & \\ ... & & m_{i+1} & & m_i & ... \\ + & & n_{i+1} & & n_i & \\ \hline & & s_{i+1} = (m_{i+1} + n_{i+1} + r_i) \bmod B & & s_i = (m_i + n_i + r_{i-1}) \bmod B & \\ & & r_{i+1} = (m_{i+1} + n_{i+1} + r_i)/B & & r_i = (m_i + n_i + r_{i-1})/B & \end{array}$$

3.4 Soustraction de deux entiers longs

La soustraction s'effectue de manière comparable à l'addition si ce n'est qu'il faut faire attention aux valeurs négatives dans les calculs. De plus, n'oubliez pas que la classe EPA ne gère que les naturels, c'est à dire les entiers positifs, il est donc impossible d'effectuer la soustraction $A - B$ si $B > A$

3.5 Multiplication de deux entiers longs

La multiplication de deux grands entiers s'inspire elle aussi de la technique de la multiplication apprise à l'école primaire. L'algorithme est plus compliqué que celui de l'addition ou la soustraction car il contiendra deux boucles imbriquées au lieu d'une seule, mais le principe général reste le même : une succession d'opérations de base avec une bonne gestion de la retenue. Voici un exemple pour vous aider à déduire l'algorithme de la multiplication de deux grands entiers.

$$\begin{array}{cccccc} & & 1 & 0 & 2 & 3 \\ & \times & & 1 & 3 & 8 \\ \hline & & 8 & 1 & 8 & 4 \\ & 3 & 0 & 6 & 9 & \\ 1 & 0 & 2 & 3 & & \\ \hline 1 & 4 & 1 & 1 & 7 & 4 \end{array}$$

Si vous avez bien compris les problèmes de base précédents, vous n'aurez pas de mal à adapter l'algorithme en base 10 à l'algorithme en base B. Un schéma trop complexe ne vous aiderait en rien.

4 La classe `EPA`

A partir du squelette de la classe `EPA`, vous devez donc programmer toutes les fonctions de base de l'arithmétique des entiers (addition, soustraction, multiplication, quotient et reste) ainsi que les fonctions plus évoluées (puissance, inverse, pgcd, test de primalité) qui vous permettront l'implantation de RSA.
Notez bien qu'il n'est pas forcément nécessaire d'implanter toutes les méthodes que vous trouverez dans ce squelette : implantez des méthodes telles que `estÉgalÀ1()` uniquement **si vous en avez besoin**.
Vous vous référerez à l'API pour connaître le rôle des méthodes à implanter. Voici quelques indications pour un certain nombre d'entre elles.

- Les constructeurs `private EPA()` et `private EPA( int[] vals )`. Ce sont des constructeurs privés et c'est donc à vous d'en définir le comportement exact. Le premier peut être vide.
- Le constructeur `public EPA( byte[] bytes )` servira lors du codage d'un fichier. Il transforme un tableau d'octets en EPA simplement en copiant les octets contenus dans `val` vers le tableau contenant les chiffres de notre EPA. Les octets sont des nombres codés sur 8 bits (256 valeurs différentes). Ce constructeur ne peut donc construire qu'une sous partie des entiers puisque chaque chiffre ne pourra pas dépasser 255. **Remarque :** ce constructeur impose aux EPA une base supérieure à 255.
- Le constructeur `public EPA( String chaîne )` construit un entier long à partir de sa représentation décimale. `chaîne` doit donc contenir une suite de chiffres de 0 à 9. Le fait d'avoir une puissance de 10 comme base nous permet de construire l'entier en découpant la chaîne en tranches sans effectuer de manipulation arithmétique complexe. Pour une base de 10, on considèrerait les caractères un par un. Pour une base de 100, deux par deux, pour une base de 1000, trois par trois, etc...
- La méthode `public byte[] toByteArray()` transforme l'EPA en tableau d'octets. C'est la méthode inverse du constructeur vu précédemment. L'existence d'une telle méthode peut vous sembler étrange dans la mesure où cela implique que tous les chiffres de votre EPA doivent être compris entre 0 et 255. Mais cela arrive et cette fonction vous sera très utile pour le décodage des fichiers.
- Le calcul du quotient et du reste de la division de a par b exprimés en base B est donné par l'algorithme suivant :

$q \leftarrow 0$
$r \leftarrow a$
$d \leftarrow$ *le premier chiffre de* b
tant que $r \geq b$ **faire**
  si $longueur(r) > longueur(b)$ **alors**
    $n \leftarrow$ *les deux premiers chiffres de* r
    $k \leftarrow longueur(r) - longueur(b) - 1$
  sinon
    $n \leftarrow$ *le premier chiffre de* r
    $k \leftarrow 0$
  finsi
  $t \leftarrow \left\lfloor \frac{n}{d} \right\rfloor$
  si $r < t \cdot b \cdot B^k$ **alors**
    $t \leftarrow \left\lfloor \frac{n}{d+1} \right\rfloor$
  finsi
  $r \leftarrow r - t \cdot b \cdot B^k$
  $q \leftarrow q + t \cdot B^k$
fin tant que
$a = q \cdot b + r$

– Le pgcd entre deux entiers se calcule à l'aide des relations suivantes :

$$pgcd(a,0) = a$$

$$pgcd(a,b) = pgcd(b, a \bmod b)$$

Il s'agit de faire décroitre b jusqu'à 0 en prenant à chaque fois le reste de la division.

– La méthode `inverseModulo( EPA module )` permet de calculer l'inverse d'un entier modulo un nombre. Cette fonction calcule l'entier $a = b^{-1} \bmod n$ c'est à dire l'entier a tel que $ab = 1 \bmod n$. Pour cela, nous allons utiliser l'algorithme d'Euclide étendu donné ci dessous.

$n_0 \leftarrow n$
$b_0 \leftarrow b$
$t_0 \leftarrow 0$
$t \leftarrow 1$
$q \leftarrow \left\lfloor \frac{n_0}{b_0} \right\rfloor$
$r \leftarrow n_0 - q \cdot b_0$
tant que $r > 0$ **faire**
    $temp \leftarrow t_0 - q \cdot t$
    si $temp \geq 0$ **alors**
        $temp \leftarrow temp \bmod n$
    sinon
        $temp \leftarrow n - ((-temp) \bmod n)$
    finsi
    $t_0 \leftarrow t$
    $t \leftarrow temp$
    $n_0 \leftarrow b_0$
    $b_0 \leftarrow r$
    $q \leftarrow \left\lfloor \frac{n_0}{b_0} \right\rfloor$
    $r \leftarrow n_0 - q \cdot b_0$
fin tant que
si $b_0 \neq 1$ **alors**
    b n'a pas d'inverse modulo n
sinon
    $b^{-1} \bmod n = t$
fin si

– Le calcul de $x^a \bmod n$ s'effectue à l'aide de l'algorithme suivant :

$p \leftarrow 1$
$p_2 \leftarrow x$
tant que $a \neq 0$ **faire**
    si a est impair **alors** $p \leftarrow (p \cdot p_2) \bmod n$
    $p_2 \leftarrow p_2^2 \bmod n$
    $a \leftarrow \left\lfloor \frac{a}{2} \right\rfloor$
fin tant que
$x^a \bmod n = p$

– Pour tester si un entier n est premier, vous pouvez utiliser le test de primalité de Miller-Rabin donné par l'algorithme ci-dessous. Etant donné un entier n, on tire un nombre aléatoire a tel que $0 < a < n$. A partir de ceci, l'algorithme est capable de décider si n est premier

avec une probabilité d'erreur inférieure à 1/4. Plus précisément, si l'algorithme répond «n est décomposable» alors n est effectivement décomposable. Si l'algorithme répond «n est premier» alors il y a moins d'une chance sur 4 pour que n ne soit pas réellement premier.

tirer un entier aléatoire a, $0 < a < n$
calculer m et k tels que $n - 1 = m \cdot 2^k$ où m est impair
$b \leftarrow a^m \bmod n$
si $b = 1$ **alors**
    répondre «n est premier» et **fin**
finsi
pour $i = 1$ **jusqu'à** k **faire**
    si $b \equiv -1 \bmod n$ **alors**
      répondre «n est premier» et **fin**
    sinon
      $b \leftarrow b^2 \bmod n$
    finsi
fin pour
répondre «n est décomposable»

Comme nous l'avons dit précédemment, la probabilité d'erreur de cet algorithme n'est inférieure qu'à 1/4. Toutefois, en effectuant ce test p fois et en tirant à chaque coup un nouveau a aléatoire, la probabilité de déclarer n premier par erreur tend vers une probabilité de l'ordre de $1/2^p$ (ce que nous ne chercherons pas à montrer).

Vous avez désormais toutes les clefs pour implanter le système de chiffrement RSA.
Bonne chance !