TP Sécurité
Cryptage asymétrique et Hachage

I-Algorithme de RSA

L'algorithme asymétrique de cryptographie RSA est fondé sur l'utilisation d'une paire de clés composée d'une clé publique pour chiffrer et d'une clé privée pour déchiffrer ou signer des données confidentielles. La clé publique correspond à une clé qui est accessible par n'importe quelle personne souhaitant chiffrer des informations. La clé privée permettant le chiffrement est quant à elle réservée à la personne ayant créé la paire de clés.
Lorsque deux personnes, ou plus, souhaitent échanger des données confidentielles, une personne prend en charge la création de la paire de clés, envoie sa clé publique aux autres personnes qui peuvent alors chiffrer les données confidentielles à l'aide de celle-ci puis envoyer les données chiffrées à la personne ayant créé la paire de clés. Cette dernière peut alors déchiffrer les données confidentielles à l'aide de sa clé privée

1-Notations

On note *K* le crypto-système de chiffrement qui représente la **clé privée** : K {d, n}
Et *K*' le crypto-système de déchiffrement qui représente la **clé publique** : K' {n, e}
Avec :

- Le **module de chiffrement** n = p * q
- p et q sont tous les deux des nombres premiers, donc PGCD(p, q) = 1
- L'**exposant de chiffrement** *e*
- L'**exposant de déchiffrement** d
- e * d = 1 mod φ(n)
- φ(n) = (p-1)*(q-1), **l'indicateur d'Euler**

2-Les étapes du RSA

2-1-CREATION DE LA PAIRE DE CLES

Le but de cette première étape est de trouver et générer trois nombre n, a et b correspondant aux clés privées et publiques.

a-Détermination de la clé publique

- Choisir deux nombres premiers p et q (de taille à peu près égale).
- Calculer n = p*q
- En déduire φ(n) = (p-1)*(q-1)
- Choisir un nombre e tel que 1 < e < ϕ(n) et PGCD(e, φ(n)) = 1

Notre clé publique est (e, n).

b-Détermination de la clé privée

Il faut trouver d tel que d*e ≡ 1 mod φ(n) avec d ≡ e^(-1) mod φ(n)
Notre clé privée est (d, n).

2-2-PREPARATION DU MESSAGE ORIGINAL EN BLOCS NUMERIQUES

Cette étape n'est pas générique à tous les cryptages et décryptages RSA. En fait, elle doit être instauré par le générateur des clés qui détermine comment son message textuel sera traduit sous forme numérique.

a- Numérisation

Pour certain, une traduction de la lettre en son équivalent numérique (code ASCII) sera choisi. Pour d'autre, un nombre est associé à chaque lettre. Chaque lettre subit une opération choisie.

b- Découpage

On découpe le message numérique en tranche m ayant moins de chiffres que le nombre de chiffres de n.

2-3-CODAGE ET DECODAGE DU MESSAGE

a-Codage

Chaque bloc constitue un entier x (message encrypté) que l'on code grâce à la clé publique e du récepteur :

On utilise la formule x = y^e mod (n) pour coder le message original.

b-Décodage

La méthode de décodage de l'entier y (message original en clair) correspond à l'inverse de celle de codage :

On utilise la formule y = x^d mod (n) pour décoder le message encrypté.

2-4-PASSAGE DU MESSAGE NUMERIQUE AU MESSAGE ORIGINAL

Cette dernière étape correspond au processus inverse de l'étape 2 (préparation du message). L'idée sera de retrouver à partir de la règle instaurée pour traduire les lettres en chiffres, le message original avec une fonction qui produit le résultat inverse de celle de l'étape 2.

- On retrouve chaque bloc textuel à partir de son bloc numérique décrypté correspondant.
- On découpe les blocs de texte pour retomber sur des mots de la langue originale.

3-Le Travail demandé :

Indications pour l'implémentation

RSA travaille avec de très grands nombres. Il conviendra donc d'utiliser la classe **BigInteger**
On notera que cette classe propose des constantes telles que **ONE** ou **ZERO**, ainsi que les opérations arithmétiques **add, subtract, multiply** etc, et **compareTo** pour la comparaison
De plus, vous pourriez avoir besoin des méthodes suivantes de cette même classe :
probablePrime : qui permet de générer un (grand) nombre qui a une forte probabilité d'être premier.
modInverse : qui permet de calculer l'inverse modulaire d'un nombre (étant donné e et n, trouver d tel que e*d= 1(mod n)).
modPow : qui permet de calculer l'élévation à la puissance modulaire (a^b(mod n))
gcd : qui permet de calculer le pgcd de deux nombres.

```
int bitLength = 1024;
SecureRandom rnd = new SecureRandom();
BigInteger p = BigInteger.probablePrime(bitLength, rnd);
BigInteger q = BigInteger.probablePrime(bitLength, rnd);
```

1) Ecrire une méthode Encrypte (message) qui retourne le message crypté en utilisant la clé publique

2) Ecrire de la même façon Decrypte (message) qui retourne le message décrypté en utilisant la clé privée

III-Hashage :

Écrire un programme Java qui prend en argument un fichier de type quelconque et génère le hashcode associé en utilisant l'algorithme simplifié suivant :
1- décomposer le fichier en matrices de byte de taille 16*100
2- calculer le xor de chaque colonne dans un tableau hash de byte de taille 16
3- répéter 1 et 2 en sommant dans hash jusqu'à la fin de fichier si le fichier est de taille qui n'est pas multiple de 16 compléter par des zéro
Tester votre code sur des fichiers de différente taille

	bit 1	**bit 2**	• • •	**bit *n***
block 1	b_{11}	b_{21}		b_{n1}
block 2	b_{12}	b_{22}		b_{n2}
	•	•	•	•
	•	•	•	•
	•	•	•	•
block *m*	b_{1m}	b_{2m}		b_{nm}
hash code	C_1	C_2		C_n

Figure 3.3 Simple Hash Function Using Bitwise XOR