TP RSA

Rappel du principe de codage RSA :

- Chaque personne souhaitant coder ou signer un message dispose d'une clef privée, un entier s connu de lui seul, et d'une clef publique, une paire d'entiers (c, n).
- n est le produit de 2 nombres premiers p et q, et s et c sont inverses modulo $\varphi(n)$, où $\varphi(n) = (p-1)(q-1)$ (le nombre d'entiers de l'intervalle $[1, n[$ premiers avec n), on a alors (cf. devoir 1)

$$(a^c \pmod n)^s \pmod n = (a^s \pmod n)^c \pmod n = a \pmod n \quad (1)$$

- Pour coder un message à destination d'une personne dont la clef publique est (c, n), on commence par le transformer en une suite d'entiers, On peut par exemple remplacer chaque caractère par un entier compris entre 0 et 255, son code ASCII.
- Puis on envoie la liste des nombres $b = a^c \pmod n$. En principe, seule la personne destinataire connait s et peut donc retrouver a à partir de b en calculant $b^s \pmod n$.
- On peut aussi authentifier qu'on est l'auteur d'un message en le codant avec sa clef privée, tout le monde pouvant le décoder avec la clef publique.

Exercice 1 : Générer une paire de clefs

Génèrez deux grands nombres premiers p et q au hasard, en utilisant par exemple les fonctions `nextprime` et `randint` de Xcas ou le test de Miller-Rabin si votre langage préféré n'a pas de test de primalité
`xcas.univ-grenoble-alpes.fr/forum/viewtopic.php?f=49&t=2582`
ou un test de Fermat ou un test naïf si vous travaillez avec un langage où les entiers sont représentés par des flottants. Calculez $n = p \times q$ puis $\varphi(n) = (p-1)(q-1)$ puis choisissez une clef secrète s inversible modulo $\varphi(n)$ et calculez son inverse c. Vérifiez sur quelques entiers la propriété (1), on pourra utiliser la fonction `pow(a,c,n)` pour calculer $a^c \pmod n$ en Xcas et en Python, ou programmer la puissance modulaire rapide (si elle n'est pas implémentée)

Programme pour l'identité de Bézout (en Xcas on peut aussi utiliser `iegcd(a,b)`) :

```
def bezout(a,b):
    l1=[1,0,a]; l2=[0,1,b]
    while l2[2]!=0:
        q=l1[2]//l2[2]
        l1,l2=l2,[l1[0]-q*l2[0],l1[1]-q*l2[1],l1[2]-q*l2[2]]
    return l1
```

Exercice 2 : Codage et décodage d'un message (sur PC)

On transforme une chaine de caractère en une liste d'entiers et réciproquement (avec `asc` et `char` en Xcas, ou l'application répétée de `ord` et `chr` en Python). Pour le moment on code caractère par caractère, sans s'inquiéter de la sécurité du codage. Pour coder/décoder une liste `l` d'entiers, on peut utiliser `pow(l,c,n)` en Xcas et Python. En utilisant la paire de clefs de l'exercice 1, codez un message puis décodez ce message pour vérifier. Décodez le message authentifié situé à l'URL
`http://www-fourier.ujf-grenoble.fr/~parisse/mat249/rsa1`.

Exercice 3 : Attaque simple

On a vu que le codage monoalphabétique n'est pas une bonne idée, une attaque possible étant la recherche de fréquences, ici on peut utiliser une attaque encore plus simple : la personne souhaitant décoder un message codé avec une clef publique sans en connaitre la

clef secrète calcule la liste des $a^c \pmod n$ pour les 256 valeurs possibles de a et compare au message. Décodez de cette manière le message situé à l'URL
`http://www-fourier.ujf-grenoble.fr/~parisse/mat249/rsa2`

Exercice 4 : Padding aléatoire
Pour parer à cette attaque, on va augmenter le nombre de valeurs possibles de a pour que le calcul de la liste de toutes les puissances des a possibles soit trop long. Plusieurs stratégies sont possibles, l'une d'elle consiste à ajouter à a un multiple aléatoire de 256. Comment la personne qui recoit un message crypté retrouvera-t-elle le message en clair ? Implémenter cette méthode.

Exercice 5 : Groupement de lettres
On peut aussi grouper par paquets de x caractères et on associe à un groupe de caractères l'entier correspondant en base 256. Par exemple, si on prend des groupes de $x = 3$ caractères, "ABC" devient `65*256^2+66*256+67` car le code ASCII de A, B, C est respectivement 65, 66, 67. Donner une condition reliant n et x pour que le décodage redonne le message original. Choisissez une paire de clefs vérifiant cette condition pour $x = 3$ (calculatrices avec entiers représentés par des flottants) ou $x = 8$ (autres). Ecrire un programme de codage et de décodage avec groupement (on commencera par compléter le message original par des espaces pour qu'il soit un multiple de 8 caractères, en Xcas et Python l'instruction `len` permet de connaitre la taille d'une chaine de caractères, en Xcas, on pourra utiliser la fonction `convert(.,base,256)` d'écriture en base 256).

Exercice 6 : Sécurité du codage 1
Vérifiez sur l'exemple de l'exercice 1 que la connaissance de $\varphi(n)$ et de n permet de calculer p et q par résolution d'une équation de degré 2. Si on connait seulement c et s, peut-on retrouver $\varphi(n)$? La sécurité du codage repose donc sur la difficulté de factoriser n. Tester sur des entiers de taille croissante le temps nécessaire au logiciel pour factoriser p et q. Une valeur de n de taille 128 bits, 512 bits, 1024 bits parait-elle suffisante ?

Exercice 7 : Sécurité du codage 2
Le choix de c et de s est aussi important. Pour le comprendre, prenons $p = 11$ et $q = 13$. Représentez pour différentes valeurs de c les points $(a, a^c \pmod n)$, plus le dessin obtenu est aléatoire, plus il sera difficile à une personne mal intentionnée de déchiffrer un message sans connaitre la clef. En Xcas, on pourra utiliser les instructions `seq` pour générer une suite de terme général exprimé en fonction d'une variable formelle, et `scatterplot(l)` qui représente le nuage de points donné par une liste `l` de couples de coordonnées. En Python, on peut utiliser l'instruction `plot` de `matplotlib`. Observez en particulier les cas où c n'est pas premier avec $\varphi(n)$ (comment voit-on que RSA ne fonctionne pas ?) et également le cas $c = 3$.

Exercice 8 : attaque par les restes chinois
Une personne souhaite envoyer le même message x à trois destinataires différents, ayant chacun leur propre clef publique $c = 3, N_1$, $c = 3, N_2$ et $c = 3, N_3$ avec $c = 3$ pour les 3 destinataires. Il envoie donc $y_1 = x^3 \pmod N)_1$, $y_2 = x^3 \pmod N)_2$ et $y_3 = x^3 \pmod N)_3$. Une personne mal intentionnée arrive à intercepter y_1, y_2 et y_3. En appliquant les restes chinois, elle peut en déduire x. Par exemple, retrouver x sans chercher à factoriser les clefs pour

```
46693373016 % 180711261397,
(-111575037168) % 840724735099,
(-18270191368) % 372130013641
```