

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)

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

- 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 connaît 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

Générez deux nombres premiers p et $q > 256$ au hasard, en utilisant par exemple les fonctions `nextprime` et `rand`. 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 utilisera la fonction `powmod(a, c, n)` pour calculer $a^c \pmod{n}$.

Exercice 2 : Codage et décodage d'un message

On transforme une chaîne de caractère en une liste d'entiers et réciproquement avec `asc` et `char`. Pour l'appliquer à une liste `l`, on peut utiliser `map(l, powmod, c, n)`. En utilisant la paire de clefs de l'exercice 1, codez un message puis décidez ce message pour vérifier. Décodez le message authentifié situé à l'URL
<http://www-fourier.ujf-grenoble.fr/~parisse/mat249/rsa1>.

Exercice 3 : Puissance modulaire rapide

Pour pouvoir crypter des messages de cette manière, il est nécessaire d'avoir une fonction calculant $a^c \pmod{n}$ rapidement. Comparer le temps de calcul de `a^c mod n` et `powmod(a, c, n)` pour quelques valeurs de a, c, n (on pourra utiliser `time(instruction)` pour connaître le temps d'exécution d'une instruction. Pour comprendre cela, programmez une fonction `puimod` calculant $a^c \pmod{n}$ en utilisant et en justifiant un des algorithmes

- récursif : si $c == 0$ on renvoie 1, si $c == 1$ on renvoie a , sinon, si c est pair, on pose $b = a^{c/2} \pmod{n}$ (calculé par appel récursif) et on renvoie $b \times b \pmod{n}$, et enfin si c est impair, on pose $b = a^{(c-1)/2} \pmod{n}$ (calculé par appel récursif) et on renvoie $b \times b \times a \pmod{n}$
- itératif : on initialise $A \leftarrow a, b \leftarrow 1$, puis tant que $c \neq 0$
 - si c est impair, $b \leftarrow A \times b \pmod{n}$,
 - $c \leftarrow$ quotient euclidien de c par 2,
 - $A \leftarrow A \times A \pmod{n}$
 et on renvoie b .

Exercice 4 : Attaque simple

L'utilisation de 256 valeurs possibles pour a se prête à une attaque très simple : la personne souhaitant décoder un message codé avec une clef publique sans en connaître la clef secrète

calcule simplement 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 5 : Groupement de lettres

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. Pour cela, on groupe 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 $x = 8$. 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, l'instruction `size` permet de connaître la taille d'une chaîne de caractères, on pourra utiliser la fonction d'écriture en base de la feuille d'exercices 2).

Exercice 6 : Sécurité du codage 1

Montrer que la connaissance de $\varphi(n)$ et de n permet de calculer p et q par résolution d'une équation de degré 2. 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 paraît-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 connaître la clef. On pourra utiliser les instructions `seq` pour générer une suite de terme général exprimé en fonction d'une variable formelle, et `scatterplot(1)` qui représente le nuage de points donné par une liste `l` de couples de coordonnées. Observez en particulier les cas où c n'est pas premier avec $\varphi(n)$ et $c = 3$. Conclusions ?

Exercice 8 : Primalité

Pour créer une paire de clefs, il faut générer des nombres premiers et donc être capable de déterminer si un nombre est premier ou non. Un test simple consiste à appliquer la contraposée du petit théorème de Fermat : si $a^p \not\equiv a \pmod p$, alors p n'est pas premier. Ecrire une fonction prenant en argument a et p et renvoyant 0 si p n'est pas premier et 1 si $a^p \equiv a \pmod p$, puis une fonction prenant en argument p et effectuant le test pour toutes les valeurs de $a \in [2, p - 1]$ jusqu'à ce que le test échoue. Si tous les tests réussissent, p est peut-être premier mais ce n'est pas certain : existe-t-il un nombre non premier pour lequel tous les tests réussissent ?