Cryptographie et Sécurité
Série 8 : TP Python - RSA

06 Novembre 2019

A rendre exclusivement sur **Moodle**, uniquement sous forme de fichier **.py**, implémenté avec **Python 3**, avant le **13 Novembre 2019 à 17h00**.

Votre code doit être **suffisamment commenté**.

Implémentation de RSA

Le but de cette série est d'implémenter RSA. Pour cela, on aura besoin d'implémenter divers outils et algorithmes :

1. **Un générateur de nombres premiers**, incluant un test de primalité (**test de Fermat**)
2. **L'algorithme d'exponentiation rapide**
3. **L'algorithme étendu d'Euclide**
4. **Un générateur de Clés**
5. **L'Encryption des messages**
6. **La Décryption des messages** de façon efficace avec l'aide du **théorème des restes chinois**.

On va maintenant revoir ces différents algorithmes, puis on donnera une marche à suivre un peu plus détaillée pour l'implémentation.

Générateur de nombres premiers

Pour la génération des clés, on a besoin de générer p et q, deux entiers, premiers, de grande taille.

La méthode qu'on utilise consiste à générer un entier aléatoire, puis à vérifier s'il est premier.

Pour cela, on utilise le test de primalité de Fermat. Si n est le nombre qu'on veut tester, on choisit un nombre aléatoire a entre 2 et n-1 (inclus), et on calcule $a^{n-1} \mod n$. Si le résultat est différent de 1, alors n n'est pas premier (c.f. petit théorème de Fermat). Sinon, n est "probablement premier". On répète ce test avec différentes valeurs de a pour augmenter la confiance qu'on a en le fait que n soit premier.

Exponentiation rapide

Pour calculer plus facilement toutes les exponentiations, que ce soit pour le test de primalité ou l'encryption, on utilise l'exponentiation rapide (Fast Exponentiation) :

On veut calculer $3^{42} \mod 25$. Le principe est de calculer les exposants dont la valeur est une puissance de 2 :

$3^1 \mod 25 = 3$
$3^2 \mod 25 = 9$
$3^4 \equiv 3^2 \cdot 3^2 \equiv 9 \cdot 9 \equiv 81 \mod 25 = 6$
$3^8 \equiv 6 \cdot 6 \equiv 36 \mod 25 = 11$
$3^{16} \equiv 121 \mod 25 = 21$
$3^{32} \equiv 21 \cdot 21 \equiv 441 \mod 25 = 16$

En décomposant $42 = 32 + 8 + 2$ en puissances de 2, on peut calculer aisément :

$$3^{42} \equiv 3^{32} \cdot 3^8 \cdot 3^2 \equiv 16 \cdot 11 \cdot 9 \equiv 1584 \mod 25 = 9$$

Algorithme étendu d'Euclide

On utilise l'algorithme étendu d'Euclide lorsqu'on crée les clés pour calculer l'inverse de l'exposant e modulo $\phi(n)$.

Soit deux entiers a et b, $a \geq b$, cet algorithme nous permet d'obtenir $r = pgdc(a, b)$ et s, t tels que $s \cdot a + t \cdot b = r$ (i.e. les cofficients de Bézout). Si a et b sont premiers entre eux, alors s est l'inverse de a modulo b, et t est l'inverse de b modulo a.

Ici, si $a = \phi(n)$ et $b = e$, on s'intéresse donc à r, qui nous indique si e est premier avec $\phi(n)$, et à t, l'inverse de $e \mod \phi(n)$.

L'algorithme est le suivant ($\div$ est une **division entière** !):

r_0 := a;
r_1 := b;
s_0 := 1;

$s_1 := 0;$
$t_0 := 0;$
$t_1 := 1;$
$q_1 := r_0 \div r_1;$

repeat until $r_{i+1} == 0$
$r_{i+1} := r_{i-1} - q_i * r_i;$
$s_{i+1} := s_{i-1} - q_i * s_i;$
$t_{i+1} := t_{i-1} - q_i * t_i;$
$q_{i+1} := r_i \div r_{i+1};$
end repeat;

return $r_i, s_i, t_i;$

Attention : L'algorithme retourne les coefficients de Bézout, donc des nombres positifs ou négatifs. Il faut donc penser à vérifier que s (resp. t) est bien compris entre 0 et b-1 (resp. a-1), cad que s (resp. t) est bien modulo b (resp. a).

Théorème des restes chinois

On ne considère que le cas spécial ou l'on a que deux congruences (le théorème général fonctionne pour n'importe quel nombre de congruences).

Soit $p, q \in \mathbb{Z}$, $p, q > 1$ et $pgdc(p, q) = 1$. Alors le système d'equations suivant :

$$x = a \mod p$$
$$x = b \mod q$$

possède une unique solution modulo $p \cdot q$, qui est $x = a \cdot q \cdot (q^{-1} \mod p) + b \cdot p \cdot (p^{-1} \mod q)$

Application :

On peut donc précalculer $(q^{-1} \mod p)$ et $(p^{-1} \mod q)$.

Tout ce qu'il reste alors à faire lorsqu'on reçoit un message à décrypter est de calculer son modulo p et q.

On cherche donc :

$$a = c^d \mod p$$
$$b = c^d \mod q$$

Ici, on peut simplifier l'exposant en calculant $\Phi(p) = p - 1$ et $\Phi(q) = q - 1$ (car p et q sont premiers).

Donc $c^{p-1} \equiv 1 \mod p$ (idem avec q), donc on a :

$$a = c^{d \mod p-1} \mod p$$
$$b = c^{d \mod q-1} \mod q$$

Ce qui réduit grandement les exposants. Il ne reste plus qu'à calculer ces deux valeurs, puis introduire le tout dans la formule pour x :

$$x = (a \cdot q \cdot (q^{-1} \mod p) + b \cdot p \cdot (p^{-1} \mod q)) \mod n$$

Remarque : Comme dit précédemment, on peut précalculer dès la génération des clés ($q^{-1} \mod p$) et ($p^{-1} \mod q$), mais également les deux exposants réduits de d qui sont ($d \mod q-1$) et ($d \mod p-1$). Ces 4 valeurs précalculées permettent d'améliorer encore l'efficacité de la décryption.

Détails d'implémentation

Les nombres premiers que vous générez devront être assez grands (au moins d'ordre 10^6).

Votre code doit contenir (Une fonction pour chaque point est un bon découpage):

- L'exponentiation rapide pour calculer la puissance d'un nombre modulo un autre.
- La génération de nombres premiers aléatoires, a partir de nombre aléatoires dont on vérifie la primalité à l'aide du test de Fermat répété plusieurs fois (et en utilisant l'exponentiation rapide pour ces tests).
- L'algorithme étendu d'Euclide pour calculer l'inverse d'un nombre modulo un autre.
- La génération de clés, en créant d'abord p et q premiers et grands avec le générateur de premiers aléatoires que vos aurez crée, puis $n = p \cdot q$, puis en sélectionnant l'exposant d'encryption e petit (et premier avec $\Phi(n)$), puis en utilisant l'algorithme étendu d'Euclide pour calculer l'exposant de décryption $d = e^{-1} \mod \Phi(n)$.
- L'encryption, avec les clés publiques n et e, via l'exponentiation rapide. On considère les messages comme étant déjà sous la forme de nombres (et des nombres plus petits que n).
- La décryption, cette fois en exploitant le fait que le possesseur de la clé privée connait p et q et le théorème des restes chinois pour faciliter la décryption.