III. Chiffrement RSA

Alice veut envoyer un message à Bob de manière sécurisé : à la réception du message seul Bob peut le déchiffrer. Pour ce faire Bob génére de manière aléatoire deux grands nombre premier p et q. Il calcule ensuite leur produit $n = p \times q$ ainsi que $\phi(n) = (p-1) \times (q-1)$. Il choisit ensuite un troisième nombre premier e qui ne divise pas $\phi(n)$ (par convention e vaut souvent 3 ou 65537). A l'aide de l'algorithme d'euclide il calcule $0 \leqslant d < \phi(n)$ tel que $ed \equiv 1 \mod \phi(n)$. Il compose alors deux clés : une publique qu'il transmet à Alice (en clair) et une secrète qu'il conserve pour lui. La clé publique est composée de n et de e. La clé secrète est composée de n et de d. Les autres entiers ne sont plus nécessaires.

A la réception de la clé publique (n, e), Alice chiffre sont message $M \in [0, n-1[$ par $C = M^e \mod n$ (exponentiation modulaire) qu'elle transmet à Bob. Bob déchiffre alors le message C en posont $M' = C^d \mod n$.

Exercice 3.1. Montrer que l'on a bien $M' = M$.

Solution : Comme on a $ed \equiv 1 \mod \phi(n)$, il existe un entier $k \in \mathbb{Z}$ vérifiant $ed = 1 + k\phi(n)$. On a donc

$$\begin{aligned}
M' &\equiv C^d \mod n \\
&\equiv (M^e)^d \mod n \\
&\equiv M^{ed} \mod n \\
&\equiv M^{1+k\phi(n)} \mod n \\
&\equiv M \times \left(M^{\phi(n)}\right)^k \\
&\equiv M \times 1^k && \text{car } M^{\phi(n)} \equiv 1 \mod n \\
&\equiv M \mod n
\end{aligned}$$

Comme M' et M sont des entiers de $[0, n-1]$ la relation $M \equiv M' \mod n$ implique $M = M'$.

Pour pouvoir utiliser le chiffrement RSA il nous faut détailler quelques points :

- comment calculer d à partir de e et $\phi(n)$;
- l'exponentiation modulaire rapide (pour effectuer des exponentiations modulaires efficacement) ;
- la génération de nombre aléatoire (ou presque) ;
- comment tester la primalité (ou presque) d'un nombre de manière efficace.

Pour calculer d il suffit d'appliquer l'algorithme d'Euclide étendu à e et $\phi(n)$. Comme e est premier et ne divise pas $\phi(n)$ les entiers e et $\phi(n)$ sont premiers entre eux. Il existe alors u et v dans $\mathbb{Z}$ vérifiant $u \times e + v \times \phi(n) = 1$. Les entiers u et v sont obtenus à l'aide de l'algorithme d'Eculide étendu. Pour d on prend alors le reste de la diviosion euclidienne de u par $\phi(n)$.

1 Exponentiation modulaire

Exercice 3.2. Caculer de manière naive $4^10 \mod 13$. Quel est le nombre d'étape ?

Solution : On a

$$\begin{aligned}
4^2 &= 4 \times 4 = 16 \equiv 3 \mod 13 \\
4^3 &= 4 \times 4^2 \equiv 4 \times 3 \equiv 12 \mod 13 \\
4^4 &= 4 \times 4^3 \equiv 4 \times 12 \equiv 48 \equiv 9 \mod 13 \\
4^5 &= 4 \times 4^4 \equiv 4 \times 9 \equiv 36 \equiv 10 \mod 13 \\
4^6 &= 4 \times 4^5 \equiv 4 \times 10 \equiv 40 \equiv 1 \mod 13 \\
4^7 &= 4 \times 4^6 \equiv 4 \times 1 \equiv 4 \mod 13 \\
4^8 &= 4 \times 4^7 \equiv 4 \times 4 \equiv 16 \equiv 3 \mod 13 \\
4^9 &= 4 \times 4^8 \equiv 4 \times 3 \equiv 12 \mod 13 \\
4^{10} &= 4 \times 4^9 \equiv 4 \times 12 \equiv 48 \equiv 9 \mod 13
\end{aligned}$$

D'où $4^{10} \equiv 9 \mod 13$. La méthode a nécessité 9 multiplications.

Une autre stratégie permet de réduire le nombre d'étapes. On part de $4^{10} = 4^5 \times 4^5$. Pour calculer 4^{10} il suffit alors de connaitre 4^5. De même on a $4^5 = 4^2 \times 4^2 \times 4$ et $4^2 = 4 \times 4$. On obtient alors :

$$\begin{aligned}
4^2 &= 4 \times 4 = 16 \equiv 3 \mod 13 \\
4^5 &= 4^2 \times 4^2 \times 4 \equiv 3 \times 3 \times 4 \equiv 36 \equiv 10 \mod 13 \\
4^{10} &= 4^5 \times 4^5 \equiv 10 \times 10 \equiv 100 \equiv 9 \mod 13
\end{aligned}$$

Avec cette méthode seule 4 multiplicatins sont nécessaires.

Exercice 3.3. A l'aide de la méthode précédente, imaginer un algorithme `exp_mod_rapide` prenant en entrée tois entiers a,p et n et retournant l'unique entier b vérifiant $b \equiv a^p \mod n$.

Solution :

```
Entier exp_mod_rapide(Entier a,Entier p,Entier n){
  Si p==0 retourner 1
  Entier y = exp_mod_rapide(a,p/2,n)
  Entier b=(y*y)%n
  Si p%2==0 retourner b
  Retourner (b*a)%n
```

2 Générateurs pseudo-aléatoires

Une suite de bits est dite *aléatoire* si elle est imprévisible, c'est-à-dire qu'aucune stratégie effective ne peut mener à un gain infini si l'on parie sur les termes de la suites. Un ordinateur ne peut pas créer de telles suites.

Une suites de bits $(a_n)_{n\in\mathbb{N}}$ est dite *pseudo-aléatoire* si elle est produite par un algorithme et s'il est algorithmiquement "difficile" de prévoir avec une probabilité $\geqslant \frac{1}{2}$ le bit a_{n+1} à partir des premiers bits $a_1, ..., a_n$.

D'un point de vu concret les suites pseudo-aléatoires ont un gros défaut, liés au fait que les ressources d'un ordinateur sont limités.

6. Quel est ce défaut ?

Le générateur pseudo-aléatoire retrouvera le même état interne au moins deux fois et la suite sera donc périodique.

En 1946, John Von Neumann propose le générateur "middle-square". Son principe est le suivant :

1 on choisit en entier n à quatre chiffres

2 on note m le nombre formé des quatre chiffres au milieu de n^2

3 on retourne m

4 on pose $n = m$ et on revient à 2

Exercice 3.4.

– **1.** Essayer cette algorithme avec $n = 1111$.

– **2.** Donner un majorant de la périodicité de la suite retournée.

– **3.** Quel est la périodicité minimale ? Pour quel n est elle atteinte ?

En 1948, Derrick Henry Lehmer à introduit les générateurs congruentiel linéaire. Le principe est le suivant, on choisit des entiers a, c et n que l'on garde secret. Puis on choisi x_0 et on calcul x_{i+1} à partir de x_i, en posant $x_{i+1} = (ax_i + b) \mod n$.

Exercice 3.5. – **1.** Tester ce générateur pour $a = 3$, $b = 4$, $n = 8$ et $x_0 = 5$. – **2.** Quel est la périodicité maximale d'un générateur congruentiel linéaire ?

Supposons que n soit connu, alors pour retrouver a et b il suffit des trois premiers termes x_0, x_1 et x_2. On pose $y_1 = x_1 - x_0$, $y_2 = x_2 - x1$.

Exercice 3.6.

– **1.** Montrer la relation $y_2 \equiv ay_1 \mod n$.

– **2.** Comment retrouver a ? puis b ?

– **3.** Retrouver les paramètres a, b qui on permis de créer la suite $97, 188, 235, 293, 604, 596, 412$ avec $n = 1023$.

Le choix des paramètres a, b et n d'un générateur congruentielle linéaire influence l'efficacité du générateur est doit donc être fait avec précaution. Un bon choix est celui utilisé par `standard minimal` : $a = 16807$ et $n = 2^{31} - 1$.

3 Test de primalité

Donnons sans démonstration le petit théorème de Fermat :

Théorème 3.7. *Si p est un nombre premier et a un nombre non divisible par p on a*

$$a^{p-1} \equiv 1 \mod p$$

La réciproque de ce théorème est fausse : il existe des entiers n non premier tels que pour tout entier a premier avec n on ait $a^{n-1} \equiv 1 \mod n$ ($n = 561$ par exemple). Un tel entier n est appelé nombre de Carmichael.

Supposons que l'on veuille tester si un nombre n est premier. Séléction des témoins $t_0, ..., t_{k-1}$. Si pour une valeur de i dans $\{0, ..., k-1\}$ on a $t_i^{n-1} \not\equiv 1 \mod n$ alors n n'est pas premier Si on ne trouve pas de tel i alors n est probablement premier. Ce test est appelé test de primalité de Fermat.

On a vu précedement que les nombres de Carmickael seraient détectés comme probablement premier par ce test or ils ne sont par premiers. Le test n'est donc pas sûr à 100%. Néanmoins les nombres ce Carmickael sont assez rares et si le nombre de témoins est asses important il est

peut probable de tomber sur un nombre comme étant probablement premier par le test et qui ne soit pas réélement premier.

Nous allons utiliser ce test sur des nombres codés sur 32 avec les témoins $2, 3, 5, 7, 11, 13$ et 17. Parmi les entiers entre 2 et $2^{32} - 1$, exactement 203280702 sont détectés probablement premier. Seulement 481 ne sont pas réellement premier. La liste complète est diponible sur ma page web.