UFR Sciences
M2 Calcul Formel Martin WEIMANN

# RSA, Primalité, Factorisation

Ce chapitre est en grande partie tiré d'un cours de Denis Simon (LMNO, Caen). Commençons par quelques rappels d'arithmétique.

**Théorème 1 (Fermat)** Si $p$ est un nombre premier, alors pour tout $a$ non divisible par $p$, on a

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

*Preuve :* exercice.

**Théorème 2** Si $N = pq$ est le produit de deux nombres premiers distincts, alors pour tout $a$ non divisible par $p$ ni $q$, on a

$$a^{(p-1)(q-1)} \equiv 1 \pmod{N}$$

*Preuve :* exercice.

**Théorème 3** Si $N$ est un entier, alors pour tout $a$ premier avec $N$,

$$a^{\varphi(N)} \equiv 1 \pmod{N}$$

où $\varphi$ est l'indicatrice d'Euler, $\varphi(N)$ est donc le cardinal de $\mathbb{Z}/N\mathbb{Z}^*$.

*Preuve :* exercice.

**Exercice 1** Montrer que si la décomposition de $N$ en facteurs premiers est $N = \prod_{i=1}^{k} p_i^{e_i}$, alors $\varphi(N) = \prod_{i=1}^{k} (p_i - 1) p_i^{e_i - 1}$.

## 1 RSA, 1978

Autrefois, tous les protocoles cryptographiques étaient symétriques. (traduction : avec toutes les méthodes connues pour fabriquer des codes secrets, quand on savait coder un message, on savait aussi le décoder). Autrement dit, la clé pour coder un message était la même que pour décoder.

Avec RSA, ce n'est plus le cas : il y a une clé publique pour coder et une clé secrète pour décoder. Il s'agit donc de cryptographie asymétrique. L'idée de la cryptographie asymétrique est de Diffie et Hellman (1976), et RSA en est la première réalisation concrète.

Le nom du protocole RSA vient du nom de ses trois auteurs : Rivest, Shamir, Adelman.

Protocole RSA :

*Préliminaires* : le ***destinataire*** choisit un fois pour toutes deux nombres premiers $p$ et $q$, puis calcule $N = pq$ et $\varphi = (p-1)(q-1)$. Il choisit aussi $e \in (\mathbb{Z}/\varphi\mathbb{Z})^*$ et calcule $d = e^{-1} \mod \varphi$. Il rend publics les paramètres $N$ et $e$ (dans le journal ou sur sa page internet...) Il garde secrets tous les autres paramètres : $p$, $q$, $\varphi$ et $d$.

*Pour chiffrer un message* : L'***expéditeur*** doit d'abord écrire le message clair sous la forme d'entiers $M \in \mathbb{Z}/N\mathbb{Z}$ (en utilisant par exemple le code ASCII). Pour le chiffrer, il doit calculer $m := M^e \mod N$. Il n'y a plus qu'à expédier le message chiffré $m$.

*Pour déchiffrer un message* : Le destinataire retrouve le message clair en calculant $m^d = M \mod N$.

**Exercice 2** Justifier la validité du protocole RSA.

complexité :

Complexité pour construire les clés : Trouver des nombres premiers $p$ et $q$. Choisir aléatoirement $d \sim N$, premier avec $\varphi(N) = (p-1)(q-1)$. Calculer $e := d^{-1} \mod \varphi$.

Complexité pour coder : Calculer $M^e \mod N$. La complexité est donc en $\tilde{O}(\log^2 N)$ en utilisant l'arithmétique modulaire rapide et l'exponentiation rapide. Le décodage a le même coût.

sécurité :

Il est évident que si on sait factoriser $N$ alors RSA est cassé. En fait, toute la sécurité de RSA repose sur la difficulté de factoriser $N$. En 2009, le record de factorisation pour un entier RSA était 768 bits ($\sim 10^{231}$). En prenant des entiers de taille 2048 bits ($2^{2048} \sim 10^{616}$), on a donc une bonne sécurité.

Conseils pour bien utiliser RSA :

Il y a certains choix des paramètres à éviter. Par exemple, il ne faut pas que $p$ et $q$ soient trop petits, sinon, la factorisation est rapide. De même, il faut que $p$ et $q$ ne soient pas trop proches l'un de l'autre, que $p-1$ et $q-1$ n'aient pas trop de petits facteurs premiers... Les petites valeurs de $d$ sont aussi à éviter.

Il faut absolument garder $\varphi(N)$ secret, car si on sait retrouver $\varphi(N)$ alors on peut factoriser $N$. En effet, on a $\varphi(N) = (p-1)(q-1)$, donc $p+q = N - \varphi(N) + 1$ et $pq = N$. Puisque l'on connait la somme et le produit de $p$ et $q$, on retrouve $p$ et $q$ en résolvant $x^2 - (p+q)x + pq = 0$. Nous verrons plus loin que l'on peut aussi factoriser $N$ si l'on connaît un multiple quelconque de $\varphi(N)$.

Deux applications :

*Échange de clés symétriques* : L'avantage des protocoles symétriques est qu'ils sont particulièrement rapides, et donc adaptés pour transmettre des gros documents. Leur défaut est que l'expéditeur et le destinataire doivent partager une clé : comment faire pour s'accorder sur une clé si l'on ne peut pas préalablement échanger de manière sécurisée ? En utilisant dans une première étape le chiffrement RSA, l'expéditeur peut envoyer de manière sécurisée une clé symétrique au destinataire. Ils ont alors une clé symétrique commune qu'ils peuvent utiliser dans une deuxième étape pour échanger des documents volumineux.

*Signature numérique de documents* : Pour cette application, il faut utiliser ce que l'on appelle une *fonction de hachage*. Une fonction de hachage $h$ est une fonction la plus compliquée possible qui prend en entrée des très grosses données et renvoie un petit résultat. Typiquement, une fonction définie sur $\mathbb{Z}/AZ$ et à valeurs dans $\mathbb{Z}/aZ$, où $A$ est très grand (plusieurs millions de chiffres) et $a$ est de la taille des clés RSA. Il est important que personne ne soit capable de trouver deux valeurs distinctes $m$ et $m'$ telles que $h(m) = h(m')$, bien que cela soit théoriquement possible puisque $A > a$. Il existe plusieurs exemples de bonnes fonctions de hachage (par exemple MD5, SHA...)

Muni d'une fonction de hachage $h$ et d'une clé RSA, voici comment on peut signer un document : Soit $D$ le document à signer et $(p, q, \varphi, d)$ une clé RSA secrète et $(N, e)$ la clé publique correspondante. La signature du document $D$ est la valeur de $s := h(D)^d \mod N$. En effet, toute personne qui lit le document peut calculer non seulement $h(D)$ mais aussi $s^e \mod N$ et constater que les deux valeurs coincident. Seul le propriétaire de la clé RSA est capable d'émettre cette signature.

## 2 Primalité

Le protocole RSA demande en particulier de générer des grands nombres premiers. Il est tout à fait inutile d'essayer de trouver des nombres premiers en factorisant des nombres au hasard. On utilise plutôt des tests de primalité, qui consistent à déterminer si un entier $N \in \mathbb{Z}$ est premier.

Avant d'essayer un algorithme compliqué pour tester la primalité d'un entier, il est souvent préférable de tester qu'il n'est divisible par aucun petit nombre premier. On peut par exemple utiliser la liste des nombres premiers jusqu'à 1000, que l'on obtient (par exemple) avec le crible d'Erathostène que l'on déroule jusqu'à $\sqrt{1000}$.

## 2.1 Test de (pseudo)-primalité de Fermat

Ce test repose sur la conséquence suivante du petit théorème de Fermat :

**Théorème 4** Un nombre entier $N \geq 3$ est premier si et seulement si $a^{N-1} = 1 \mod N$ pour tout nombre entier $1 < a < N$.

*Preuve :* Le sens direct est le petit théorème de Fermat. Pour la réciproque, supposons que $N$ ne soit pas premier et soit $1 < a < N$ un diviseur de $N$ non trivial. Si $a^{N-1} = 1 \mod N$, alors $a$ divisant $N$, on doit aussi avoir $a$ divise 1. Absurde. □

Si $N$ est grand, il est évidemment impossible en pratique de tester ces congruences pour tout $a < N$, la complexité serait alors au moins linéaire en $N$, donc exponentielle en la taille $\log(N)$ de l'entrée. Le test de primalité de Fermat repose sur l'idée suivante : si $N$ est composé, alors il est peu probable que $a^{N-1} = 1 \mod N$ pour une valeur arbitraire de $a$.

**Algorithme 1 (Test de pseudoprimalité de Fermat)** Cet algorithme prend en entrée un entier $N > 3$ et un entier $a$, $1 < a < N - 1$.

(1) `Calculer` $b := a^{N-1} \mod N$.

(2) `Si` $b = 1$ `retourner "`$N$ `est pseudopremier en base` $a$`".`

`Sinon, retourner "`$N$ `n'est pas premier".`

Quand l'algorithme retourne " $N$ n'est pas premier ", on est certain que $N$ n'est pas premier. En revanche, si $N$ est pseudopremier en base $a$, on ne peut pas affirmer qu'il est premier.

Toutefois, un entier $N$ pseudopremier ressemble beaucoup à un nombre premier et le test de Fermat a peu de chances de se tromper. Rien que pour la base $a = 2$, il n'existe que 22 valeurs de $N < 10000$ pour lesquelles le test se trompe. Il a été montré qu'un nombre de 1024 bits pseudopremier en base $a = 2$ a seulement 1 chance sur $10^{41}$ de ne pas être premier !

On peut aussi essayer de borner la probabilité de succès du test de primalité de Fermat lorsque l'on tire $a$ aléatoirement.

**Proposition 1** Soit $N \geq 3$ un nombre composé. S'il existe un entier $a$ premier avec $N$ tel que $a^{N-1} \neq 1 \mod N$, alors le test de primalité de Fermat a une probabilité de se tromper $< 1/2$.

*Preuve :* Il faut montrer que l'ensemble $B = \{b : b^{N-1} = 1 \mod N\}$ des "mauvaises bases" $b \in [1, N-1]$ est de cardinal $\leq (N-1)/2$. On remarque que $B$ est un sous-groupe du groupe multiplicatif $(\mathbb{Z}/N\mathbb{Z})^\times$. Par hypothèse, il existe $a \in (\mathbb{Z}/N\mathbb{Z})^\times \setminus B$, donc $B$ est un sous groupe strict. Par le théorème de Lagrange, $|B| \leq \varphi(N)/2 \leq (N-1)/2$. □

Dans ce cas, appliquer le test de Fermat pour plusieurs valeurs différentes de $a$ augmente exponentiellement vite les chances de succès. Par exemple, pour 10 valeurs aléatoires de $a$, la probabilité de se tromper est $< 1/2^{10} \approx 1/1000$.

**Nombres de Carmichael.** Malheureusement, il existe des nombres composés $N$ (très rares) pour lesquels la proposition ne s'applique pas. Ce sont les nombres $N$ composés qui sont pseudopremiers en toute base $a$ première à $N$, que l'on appelle *nombres de Carmichael*.

*Exemple :* $561 = 3 \cdot 11 \cdot 17$ est le premier nombre de Carmichael. Il est composé, mais il est pseudopremier en toute base $a$ premier avec 561. En effet, dans ce cas le petit théorème de Fermat implique

$$\begin{aligned} a^{560} = (a^2)^{280} &= 1 \pmod 3 \\ a^{560} = (a^{10})^{56} &= 1 \pmod{11} \\ a^{560} = (a^{16})^{35} &= 1 \pmod{17} \end{aligned}$$

et il suit du théorème des restes chinois que $a^{560} = 1 \mod 561$. Dans ce cas, les seules bases pour lesquelles 561 n'est pas pseudopremier sont les bases $a$ non premières à $N$, au nombre de $561 - \varphi(561)$. On a $\varphi(561) = (3-1)(11-1)(17-1) = 320$. La probabilité que le test de Fermat se trompe sur 561 est donc de $320/561 > 1/2$.

Les nombres de Carmichael sont donc les bêtes noires du test de primalité de Fermat. On sait depuis peu (1994, Alford, Granville et Pomerance) qu'il en existe une infinité. Cependant, ils sont très rares ! Il y a par exemple seulement 646 nombres de Carmichael inférieur à $10^9$.

**Exercice 3** Montrer que si $N = p_1 \cdots p_r$ où les $p_i \geq 3$ sont des nombres premiers distincts tels que $p_i - 1$ divise $N - 1$ pour tout $i$, alors $N$ est un nombre de Carmichael (la réciproque est vraie, théorème de Korselt, 1899). En déduire que $75361 = 11 \times 13 \times 17 \times 31$ est un nombre de Carmichael puis montrer que le test de primalité de Fermat a plus de 3 chances sur 4 de se tromper sur ce nombre.

## 2.2 Test de pseudoprimalité forte de Miller–Rabin

**Théorème 5 (Fermat)** Soit $p > 2$ un nombre premier et $p - 1 = 2^s t$ avec $t$ impair. Si $(a, p) = 1$, alors

$$\begin{cases} \text{Soit } a^t & \equiv +1 \mod p \\ \text{Soit } a^{2^i t} & \equiv -1 \mod p \text{ pour un certain } i, 0 \leqslant i \leqslant s-1 \end{cases}$$

**Exercice 4** Montrer ce théorème (*indication*: factoriser $a^{p-1} - 1$).

**Algorithme 2 (Test de pseudoprimalité forte de Miller–Rabin)** Cet algorithme prend en entrée un entier impair $N > 3$ et un entier $a$, $2 \leqslant a \leqslant N - 2$.

(1) Calculer $t$ impair et $s > 0$ tels que $n - 1 = 2^s t$.

(2) Calculer $b := a^t \mod N$.

(3) Si $b = \pm 1$, retourner " $N$ est pseudopremier fort en base $a$ ".

(4) Pour $1 \leqslant i \leqslant s - 1$ faire

$b := b^2$

Si $b = -1$, retourner " $N$ est pseudopremier fort en base $a$ ".

(5) Retourner " $N$ n'est pas premier ".

**Proposition 2** (admis.) Soit $N > 9$ un entier non premier. Alors le nombre d'entiers $1 < a < N-1$ tels que $N$ est pseudopremier fort en base $a$ est $\leqslant \frac{1}{4}\varphi(N)$.

Ce résultat assure en particulier que la probabilité d'échec du test de Miller–Rabin est $\leqslant \frac{1}{4}$. Ainsi, en appliquant 20 fois l'algorithme sur un entier non premier avec des valeurs aléatoires de $a$, on a une probabilité d'erreur inférieure à $\frac{1}{4^{20}} < 10^{-12}$ ce qui est très faible. De plus, le résultat est cette fois inconditionnel, les nombres de Carmichael ne dérogent pas à la règle.

**Exercice 5** Soit $N$ un entier qui est pseudopremier en base $a$ (pour le test de Fermat), mais qui n'est pas pseudopremier fort en base $a$ (pour le test de Miller-Rabin). Expliquer comment on peut factoriser $N$.

### 2.3 Test de primalité de Lucas

**Théorème 6 (Lucas)** Un entier $N$ est premier si et seulement s'il existe $a \in \mathbb{Z}$ tel que $a^{N-1} \equiv 1 \pmod{N}$, mais que $a^{\frac{N-1}{q}} \not\equiv 1 \pmod{N}$, pour tout premier $q$ divisant $N-1$.

Ce résultat donne un certificat qu'un nombre $N$ est premier, à condition que les facteurs premiers de $N-1$ soient connus. Il faut donc factoriser $N-1$, et être certain que les facteurs trouvés sont premiers...

## 3 Factorisation

On ne détaillera dans cette section que la méthode naïve et la méthode basée sur la connaissance (d'un multiple) de l'indicatrice d'Euler $\varphi(N)$.

### 3.1 Factorisation naïve

La méthode naïve consiste à faire des divisions euclidiennes de $N$ par tous les entiers $\sqrt{N}$. On peut améliorer un peu en ne considérant seulement des divisions

- par les nombres premiers
- par 2 et les nombres impairs
- par $2, 3$ et les $\pm 1 \pmod 6$.

Complexité $= \tilde{O}(N^{\frac{1}{2}})$.

Un petit calcul : Combien de temps faut–il pour retrouver $p$ et $q$ par la méthode naïve, connaissant seulement leur produit $M = pq$, sachant que $M$ a 2048 bits ?

Il faut faire environ $2^{1024} \sim 10^{308}$ divisions. Imaginons que l'on dispose d'un ordinateur avec un processeur fonctionnant à 1GHz. Il fait donc $10^{12}$ opérations par seconde. Il s'agit bien sûr d'opérations élémentaires, et pas de divisions de grands entiers, mais imaginons quand même que notre ordinateur fasse $10^{12}$ divisions par seconde. Imaginons qu'arrivent des progrès spectaculaires en matériel informatique et que notre ordinateur fasse $10^{20}$ divisions par seconde. Imaginons aussi que l'on utilise tous les ordinateurs de la planète, soit $10^{10}$ ordinateurs qui travaillent tous en même temps. On peut donc faire $10^{30}$ divisions par seconde. Avec tout ce matériel, il suffirait donc de $10^{308-30} = 10^{278}$ secondes pour terminer le calcul. Rappelons que l'Univers a $13,7$ milliards d'années, soit moins de $10^{18}$ secondes. Avec tout ce matériel, il faut donc seulement $10^{260}$ fois l'âge de l'Univers pour terminer le calcul.

### 3.2 Factorisation en connaissant un multiple de $\varphi(N)$

**Exercice 6** Supposons qu'un entier $N$ secret soit de la forme $N = pq$ avec $p$ et $q$ premiers. Suppososns que l'on arrive à connaître $\varphi(N)$. Comment peut–on utiliser cette information pour retrouver $p$ et $q$ ?

Au-delà de cet exercice, supposons $N$ composé et supposons que l'on connaisse seulement un multiple de $\varphi(N)$. Comment factoriser $N$ ? En fait, la méthode que nous allons décrire marche encore si l'on connait seulement un multiple de l'exposant du groupe $(\mathbb{Z}/N\mathbb{Z})^*$. (dans la plupart des cas difficiles, on passe de l'un à l'autre en multipliant par de petits nombres premiers $(2, 3, 5...)$.

On utilise une idée voisine du test de Miller–Rabin. On prend $1 < a < N-1$ aléatoire. Si $\text{pgcd}(a, N) \neq 1$, on a trouvé un facteur de $N$, c'est gagné. Sinon, $a \in (\mathbb{Z}/N\mathbb{Z})^*$. En supposant connu $M$ multiple de $\varphi(N)$, on a donc $a^M = 1 \mod N$ d'après le théorème d'Euler-Fermat. Ecrivons $M = 2^s t$ avec $t$ impair et notons $b = a^t \mod N$. On a donc $b^{2^s} = 1 \mod N$. On calcule le plus petit indice $i \geq 0$ tel que $b^{2^i} = 1 \mod N$. Si $i = 0$, on ne récupère pas d'information et on réitère avec une autre valeur de $a$. Sinon $N$ divise $b^{2^i} - 1$ et $i > 0$ implique

$$N = \text{pgcd}(N, b^{2^i} - 1) = \text{pgcd}(N, b^{2^{i-1}} - 1)\,\text{pgcd}(N, b^{2^{i-1}} + 1) \qquad (*)$$

Or $N$ ne divise pas $b^{2^{i-1}} - 1$ par définition de $i$. On vérifie que $N$ ne divise pas non plus $b^{2^{i-1}} + 1$ (ce qui arrive avec une probabilité $\geq 1/2$ dès lors que $N$ est composé, car $x^2 - 1 = 0 \mod N$ a alors au moins deux autres solutions que les solutions triviales $\pm 1 \mod N$, cf ci-après). Si c'est le cas, $(*)$ est une factorisation non triviale de $N$, gagné. Sinon, perdu. On recommence avec une autre valeur de $a$.

**Algorithme 3** Factorisation de $N$ connaissant un multiple $M$ de $\varphi(N)$.

(1) `Calculer` $t$ `impair et` $s > 0$[1] `tels que` $M = 2^s t$.

(2) `Choisir` $1 < a < N - 1$ `au hasard.`

(3) `Si` $\text{pgcd}(a, N) > 1$, `retourner` $\text{pgcd}(a, N)$.

(4) `Calculer` $b := a^t \mod N$.

(5) `Pour` $1 \leqslant j \leqslant s - 1$ `faire`

  `Si` $b = \pm 1$, `retourner «aucun facteur trouvé».`

  $B := b^2 \mod N$.

  `Si` $B = 1$, `retourner` $\text{pgcd}(b - 1, N)$.

  $b := B$.

En particulier, on voit que dans RSA, si l'exposant secret $d$ est connu, on peut non seulement jeter la clé $e$, mais aussi le module $N = pq$. En effet, on a $ed - 1 = k\varphi(N)$, donc on peut factoriser $N$ avec l'algorithme précédent en considérant $M = ed - 1$.

**A propos du nombre de solutions de $x^2 - 1 = 0 \mod N$.** Modulo un nombre premier $p$, le polynôme $x^2 - 1 = 0$ admet deux racines $\pm 1$, et ce sont les seules car $\mathbb{Z}/p\mathbb{Z}$ est un corps. Modulo un nombre composé, il y a un grand nombre de racines de $x^2 - 1 = 0$. Pour les construire, on choisit des signes $\pm 1$ modulo chaque facteur premier de $N$, puis on relève avec le Théorème Chinois. Cela donne $2^{\omega(N)}$ racines de $x^2 - 1 = 0$ (cf exo ci-après). Chaque racine $x \neq \pm 1 \mod N$ donne alors une factorisation *non triviale* de $N$

$$N = \text{pgcd}(x^2 - 1, N) = \text{pgcd}(x - 1, N)\,\text{pgcd}(x + 1, N).$$

**Exercice 7 (nombre de racines carrées de 1 modulo $N$)** Soit $N > 1$ un entier. Dans chacun des cas suivants, donner le nombre de solutions de l'équation $x^2 = 1$ modulo $N$.

1. $N$ est un nombre premier impair.
2. $N$ est est une puissance d'un nombre premier impair.

[1] Si $p$ divise $N$ alors $p - 1$ divise $\varphi(N)$. Donc $\varphi(N)$ est pair et $M$ l'est également

3. $N$ est impair quelconque.

4. $N = 2$; $N = 4$; $N = 8$.

5. $N = 2^a$ avec $a \geqslant 3$.

6. $N$ est quelconque.

### 3.3 D'autres algorithmes ? Un bref état de l'art...

Voici quelques résultats de complexité pour la factorisation d'un entier $N$. On rappelle que la complexité doit être interprétée en fonction de la taille de l'entrée $\log(N)$.

- Méthode du log discret de Shanks, 1971 : complexité exponentielle $\tilde{O}(N^{1/3})$
- Méthode $\rho$ de Pollard, 1975 : complexité exponentielle $\tilde{O}(N^{1/4})$
- Méthode de Dixon, 1981 : complexité sous-exponentielle $O(\exp(c \cdot \sqrt{\log N \log\log N}))$

Ces algorithmes peuvent tout à fait être étudiés au niveau M2 (ce sont des thèmes possibles abordés à l'option C). Actuellement, l'algorithme de factorisation ayant la meilleur complexité asymptotique est l'algorithme NFS (*Number Field Sieve, 1993*), dont la complexité sous-exponentielle est de l'ordre de

$$O(\exp(c \cdot (\log N)^{\frac{1}{3}} (\log\log N)^{\frac{2}{3}}))$$

avec $c$ une constante proche de 2.

Il n'existe à ce jour aucun algorithme de factorisation de complexité polynomiale, excepté bien sûr si l'on a de l'information sur certains facteurs de $\varphi(N)$ (exemple la méthode $p-1$ de Pollard, 1974 qui est d'ailleurs une brique élémentaire des algorithmes de complexité sous-exponentielle).

Il a été prouvé que l'ordinateur quantique, s'il voit le jour (probablement plutôt quand que si), permettrait de résoudre la factorisation en temps polynomial (algorithme de Shor, de complexité $O(\log(N)^3)$). Le cryptosystème RSA aura alors du souci à se faire. La cryptographie post-quantique est déjà en route...