Notes du cours PO-13502
Cryptage RSA et tests de primalité
2013-2014

B. Ischi

(MaTheX - http://www.mathex.net)

Collège de Candolle

Table des matières

CHAPITRE 1

Notions de base d'arithmétique

Dans ce qui suit, $\mathbb{N} = \{0, 1, 2, 3, \cdots\}$ désigne l'ensemble des nombres naturels et $\mathbb{Z}$ l'ensemble des entiers relatifs. Par ailleurs, les lettres k, l ,m, n, p, q, r, s et t désignent des nombres entiers.

1. Théorème fondamental de l'arithmétique

DÉFINITION 1.1. Un nombre $p \in \mathbb{N}$ est premier s'il possède exactement deux diviseurs.

THÉORÈME 1.2. *Tout entier $n > 1$ est le produit d'un nombre fini de nombres premiers.*

DÉMONSTRATION. Procédons par récurrence sur n. L'énoncé est vrai pour $n = 2$. Supposons maintenant qu'il soit vrai pour tout nombre $\leq n$. Si $n + 1$ est premier, la démonstration est terminée. Sinon, il existe deux nombres $a \neq 1 \neq b$ tels que $n + 1 = ab$. Par hypothèse de récurrence, a et b sont des produits finis de nombres premiers et par conséquent, $n+1$ est également un produit fini de nombres premiers. □

THÉORÈME 1.3 (Euclide, III$^{\text{e}}$ siècle av. J.-C.). *Il existe une infinité de nombres premiers.*

DÉMONSTRATION. Raisonnons par l'absurde: supposons qu'il existe un nombre fini de nombres premiers: $p_1, p_2, \cdots, p_k$. Soit $n = p_1 p_2 \cdots p_k + 1$. En vertu du résultat qui précède, n est un produit fini de nombres premiers. Par conséquent, $p_j \mid n$ pour un certain j entre 1 et k, ce qui est impossible car le reste de la division de n par p_j vaut 1. □

LEMME 1.4 (Euclide). *Si un nombre premier p divise un produit mn, alors alors p divise m ou n.*

DÉMONSTRATION. Suivant Gauss, raisonnons par l'absurde: supposons qu'il existe un nombre premier p et des nombres m et n non divisibles par p tels que $p \mid mn$. Pour p et m fixés, soit n le plus petit nombre vérifiant ces hypothèses. Alors $n < p$. En effet, si ce n'est pas le cas, par division euclidienne, $n = qp + r$ avec $r < p$ et comme $mn = mqp + mr$, il suit $p \mid mr$.

Comme $n < p$, on trouve, par division euclidienne, $p = kn+s$ avec $0 < s < n$ car p est premier. Par conséquent, $ms = mp - mkn$ qui est divisible par p ce qui n'est pas possible, puisque par hypothèse, n est le plus petit nombre non divisible par p tel que $p \mid mn$. □

THÉORÈME 1.5 (Théorème fondamental de l'arithmétique). *Tout nombre entier $n > 1$ se décompose en produit fini de nombres premiers*

$$n = p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k} \text{ où } p_1 < p_2 < \cdots < p_k$$

et cette décomposition est unique.

DÉMONSTRATION. Par le théorème qui précède, nous savons que n se décompose en un produit fini de nombres premiers. Nous devons donc montrer que cette décomposition est unique. Nous pouvons procéder par récurrence sur n. Pour $n = 2$, la décomposition est unique. Supposons que la décomposition soit unique pour tout nombre $\leq n - 1$ et que

$$p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k} = n = q_1^{m_1} q_2^{m_2} \cdots q_j^{m_j} \text{ avec } p_1 < p_2 < \cdots < p_k \text{ et } q_1 < q_2 < \cdots < q_j \ .$$

En vertu du lemme d'Euclide, p_1 divise un des q_i et donc est égal à un des q_i. Si $i \neq 1$, alors

$$p_1^{n_1-1}p_2^{n_2}\cdots p_k^{n_k} = m = q_1^{m_1}\cdots q_i^{m_i-1}\cdots q_j^{m_j} \text{ avec } p_1 < p_2 < \cdots < p_k \text{ et } q_1 < q_2 < \cdots < q_j$$

Comme $m < n$, par hypothèse de récurrence, la décomposition de m est unique, et donc $q_1 = p_1 = q_i$, une contradiction. Par conséquent, $p_1 = q_1$ et

$$p_1^{n_1-1}p_2^{n_2}\cdots p_k^{n_k} = m = q_1^{m_1-1}q_2^{m_2}\cdots q_j^{m_j} \text{ avec } p_1 < p_2 < \cdots < p_k \text{ et } q_1 < q_2 < \cdots < q_j$$

et $m < n$. Par hypothèse de récurrence, $k = j$, $p_1 = q_1$, $p_2 = q_2, \cdots, p_k = q_k$ et $n_1 - 1 = m_1 - 1$, $n_2 = m_2, \cdots, n_k = m_k$, ce qui montre que la décomposition de n est unique. □

2. Théorème de Bézout et algorithme d'Euclide

THÉORÈME 2.1 (Bézout). *Soient a, $b \in \mathbb{Z}$. Alors, il existe s, $t \in \mathbb{Z}$ tels que*

$$\text{pgcd}(a, b) = a \cdot s + b \cdot t$$

DÉMONSTRATION. Notons $d = \text{pgcd}(a, b)$. L'ensemble

$$H = a\mathbb{Z} + b\mathbb{Z} = \left\{ am + bn \;\middle|\; m,\, n \in \mathbb{Z} \right\}$$

est clairement un idéal de $\mathbb{Z}$. Comme $\mathbb{Z}$ est un anneau principal, H est principal. Par conséquent, il existe $d' \in \mathbb{N}$ tel que $H = d'\mathbb{Z}$ et donc il existe des nombres s et t tels que $d' = as + bt$. Comme, par hypothèse, d divise a et b, il suit d divise d'. Par ailleurs, a et $b \in H$, par conséquent d' divise a et b et aussi $d' \leq d = \text{pgcd}(a, b)$. On conclut que $d = d'$. □

Algorithme d'Euclide étendu.

L'algorithme d'Euclide étendu permet de trouver rapidement les nombres s et t. On suppose que $a > b$.

Partie A: On définit

$$\begin{aligned} s_1 &= 1, & t_1 &= 0, \\ s_2 &= 0, & t_2 &= 1, \\ a_1 &= a, & b_1 &= b\ . \end{aligned}$$

Ainsi,

$$\begin{aligned} a_1 &= & s_1 a + & & t_1 b \\ b_1 &= & s_2 a + & & t_2 b \end{aligned}$$

De plus, on note $a_1 = q_1 \cdot b_1 + r_1$, $s = s_1 - q_1 s_2$ et $t = t_1 - q_1 t_2$. Par conséquent,

$$r_1 = a_1 - q_1 b_1 = s \cdot a + t \cdot b$$

Remarquons que,

$$\text{pgcd}(a, b) = \text{pgcd}(a_1, b_1) = \text{pgcd}(b_1, r_1)$$

En effet, si k divise a_1 et b_1, alors k divise $r_1 = a_1 - q_1 b_1$ et si l divise $r_1 = a_1 - q_1 b_1$ et b_1, alors l divise également a_1.

Partie B: On pose

$$\begin{aligned} s_1 &= s_2, & t_1 &= t_2, \\ s_2 &= s, & t_2 &= t, \\ a_1 &= b_1, & b_1 &= r_1\ , \end{aligned}$$

et on recommence l'étape **A**.

L'algorithme s'arrête quand $r_1 = 0$. A ce moment, $\text{pgcd}(a,b) = b_1$ et $s = s_2$ et $t = t_2$.

EXEMPLE 2.2. Calculons de pgcd de 2322 et 654:

$$\begin{aligned}
2322 &= 1\cdot 2322 + 0\cdot 654 \\
654 &= 0\cdot 2322 + 1\cdot 654 \\
360 &= \underbrace{(1\cdot 1 - 3\cdot 0)}_{=1}\cdot 2322 + \underbrace{(1\cdot 0 - 3\cdot 1)}_{=-3}\cdot 654 && (2322 = 3\cdot 654 + 360) \\
294 &= \underbrace{(1\cdot 0 - 1\cdot 1)}_{=-1}\cdot 2322 + \underbrace{(1\cdot 1 - 1\cdot(-3))}_{=4}\cdot 654 && (654 = 1\cdot 360 + 294) \\
66 &= \underbrace{(1\cdot 1 - 1\cdot(-1))}_{=2}\cdot 2322 + \underbrace{(1\cdot(-3) - 1\cdot 4)}_{=-7}\cdot 654 && (360 = 1\cdot 294 + 66) \\
30 &= \underbrace{(1\cdot(-1) - 4\cdot 2)}_{=-9}\cdot 2322 + \underbrace{(1\cdot 4 - 4\cdot(-7))}_{=32}\cdot 654 && (294 = 4\cdot 66 + 30) \\
6 &= \underbrace{(1\cdot 2 - 2\cdot(-9))}_{=20}\cdot 2322 + \underbrace{(1\cdot(-7) - 2\cdot 32)}_{=-71}\cdot 654 && (66 = 2\cdot 30 + 6) \\
0 &= && (30 = 5\cdot 6 + 0)
\end{aligned}$$

Par conséquent, nous trouvons que

$$\text{pgcd}(2322, 654) = 6 = 20\cdot 2322 - 71\cdot 654$$

EXEMPLE 2.3. La fonction pgcd2 du script *Python* donné ci-dessous calcule le pgcd de a et b à l'aide de l'algorithme d'Euclide étendu et donne deux entiers s et t tels que

$$\text{pgcd}(a,b) = a\cdot s + b\cdot t$$

```
1  def pgcd2(a,b):
2      s1=1
3      t1=0
4      s2=0
5      t2=1
6      a1=max(a,b)
7      b1=min(a,b)
8      r1=a1%b1
9      while r1>0:
10         q1=a1//b1
11         s=s1-q1*s2
12         t=t1-q1*t2
13         t1=t2
14         s1=s2
15         t2=t
16         s2=s
17         a1=b1
18         b1=r1
19         r1=a1%b1
20     if a>=b:
21         return [b1,s2,t2]
22     else:
23         return [b1,t2,s2]
24 a=input("a=?")
25 b=input("b=?")
26 print pgcd2(a,b)
```

L'exécution du script donne, par exemple:

```
python exemple.py
a=?2322
b=?654
[6, 20, -71]
```

Complexité de l'algorithme d'Euclide étendu.

La suite des nombres "à gauche" est $a_1,\ b_1,\ r_1,\ r_2,\ \cdots,\ r_n = 0$, où n est le nombre de lignes. On peut la renuméroter comme suit: $R_0 = r_n$, $R_1 = r_{n-1}$, $\cdots$, $R_{n-3} = r_1$, $R_{n-2} = b_1$ et $R_{n-1} = a_1$. On constate que

$$R_k \geq R_{k-1} + R_{k-2} \text{ donc } R_k \geq F_k \cdot \text{pgcd}(a, b)$$

où F_k est le $k^{\text{ème}}$ terme de la suite de Fibonacci:

$$0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ \cdots,\ F_k = F_{k-1} + F_{k-2}$$

Rappelons maintenant quelques résultats bien connus concernant la suite de Fibonacci. On cherche x tel que $F_k = x^k$ et $F_{k+2} = F_{k+1} + F_k$, c'est-à-dire $x^{k+2} = x^{k+1} + x^k$. En divisant par x^k, on trouve $x^2 = x + 1$, ou $x^2 - x - 1 = 0$, c'est-à-dire

$$x = \varphi := \frac{1+\sqrt{5}}{2} \text{ (le nombre d'or) ou } x = \frac{-1}{\varphi} = \frac{1-\sqrt{5}}{2}$$

On pose

$$F_k = \alpha\varphi^k + \beta\left(\frac{-1}{\varphi}\right)^k$$

Alors,

$$\begin{cases} k=0 \ \Rightarrow \ \alpha+\beta=0 \\ k=1 \ \Rightarrow \ \alpha\varphi - \frac{\beta}{\varphi} = 1 \end{cases} \Rightarrow \ \alpha\left(\varphi + \frac{1}{\varphi}\right) = 1$$

$$\Rightarrow \ \alpha = \frac{\varphi}{\varphi^2+1} = \frac{\varphi}{\varphi+2} = \frac{1+\sqrt{5}}{5+\sqrt{5}} = \frac{1+\sqrt{5}}{\sqrt{5}(\sqrt{5}+1)} = \frac{1}{\sqrt{5}} \ \Rightarrow \ \beta = -\frac{1}{\sqrt{5}}$$

De plus, asymptotiquement, la suite de Fibonacci est une suite géométrique:

$$\lim_{k\to\infty}\left(F_k - \alpha\varphi^k\right) = 0$$

Par conséquent, le nombre de lignes de l'algorithme d'Euclide étendu est proportionnel à $\log(\max(a, b))$.

3. Congruences

Définition 3.1. Soient a, b et n des nombres entiers avec $n > 1$. On dit que a **est congru à** b **modulo** n si n divise a-b. On note

$$a \equiv b \pmod{n}$$

Remarque 3.2. La relation de congruence dans $\mathbb{Z}$ est une relation d'équivalence compatible avec l'addition et la multiplication.

Définition 3.3. On définit $\mathbb{Z}/n\mathbb{Z}$ l'ensemble des classes d'équivalences modulo n.

REMARQUE 3.4. $\mathbb{Z}/n\mathbb{Z}$ est un anneau pour l'addition et la multiplication modulo n. Les classes d'équivalence sont notées simplement par leur représentant entre 0 et $n-1$. Par ailleurs, on note parfois $\mathbb{Z}_n$ au lieu de $\mathbb{Z}/n\mathbb{Z}$. Par conséquent, comme ensemble,

$$\mathbb{Z}_n = \{0, 1, 2, \cdots, n-1\}$$

DÉFINITION 3.5. $\mathbb{Z}_n^*$ désigne le groupe des éléments inversible (pour la multiplication modulo n) de $\mathbb{Z}_n$.

PROPOSITION 3.6. *Soient m et n des entiers avec $n > 1$ et $0 \leq m \leq n-1$. Alors, les trois propositions suivantes sont équivalentes:*

(1) *m est un générateur de $\mathbb{Z}_n$*
(2) $\text{pgcd}(m, n) = 1$
(3) *m est inversible dans $\mathbb{Z}_n$*

DÉMONSTRATION. (1) $\Rightarrow$ (3): Si m est un générateur de $\mathbb{Z}_n$, alors il existe un entier k tel que $km = 1 \pmod{n}$ ce qui signifie que m est inversible modulo n.

(3) $\Rightarrow$ (2): Si m est inversible dans $\mathbb{Z}_n$, alors il existe un entier k tel que $km = 1 \pmod{n}$, donc il existe un entier l tel que $km = 1 + ln$, en d'autres termes $km - ln = 1$, ce qui montre que $\text{pgcd}(m, n)$ divise 1, c'est-à-dire que $\text{pgcd}(m, n) = 1$.

(2) $\Rightarrow$ (1): Si $\text{pgcd}(m, n) = 1$, en vertu du théorème de Bézout, il existe des nombres a et b tels que

$$1 = \text{pgcd}(m, n) = am + bn$$

ce qui montre que am est congru à 1 modulo n et donc que m engendre $\mathbb{Z}_n$. □

LEMME 3.7 (chinois). *Soient m, $n \in \mathbb{Z}$. Si $\text{pgcd}(m, n) = 1$, alors il existe un isomorphisme d'anneaux entre $\mathbb{Z}_{mn}$ et $\mathbb{Z}_n \times \mathbb{Z}_m$.*

DÉMONSTRATION. Considérons l'application f de $\mathbb{Z}$ dans $\mathbb{Z}_n \times \mathbb{Z}_m$ définie par

$$f(k) = (k \bmod m, k \bmod n) .$$

C'est un homomorphisme d'anneaux et

$$\text{Ker}(f) = \text{ppcm}(m, n) \cdot \mathbb{Z}$$

Comme par hypothèse, m et n sont relativement premiers, $\text{ppcm}(m, n) = mn$. Par conséquent, l'homomorphisme $\hat{f}$ de $\mathbb{Z}_{mn}$ dans $\mathbb{Z}_n \times \mathbb{Z}_m$ induit par f est injectif. Finalement, comme $\mathbb{Z}_{mn}$ et $\mathbb{Z}_n \times \mathbb{Z}_m$ ont le même nombre d'éléments, $\hat{f}$ est un isomorphisme d'anneaux. □

Algorithme des restes chinois.

L'algorithme des restes chinois permet de trouver une préimage d'un nombre $a \in \mathbb{Z}_n \times \mathbb{Z}_m$. Soient $n_1, \cdots, n_k$ des nombres entiers deux à deux relativement premiers (*i.e.* $\text{pgcd}(n_i, n_j) = 1$, $\forall\ 1 \leq i < j \leq k$) et $n = n_1 n_2 \cdots n_k$. Soient $a_1, \cdots, a_k$ des nombres entiers. On cherche x tel que

$$x \equiv a_1 \pmod{n_1},\ \cdots,\ x \equiv a_k \pmod{n_k}$$

En vertu du lemme chinois, x existe et est unique modulo n.

Pour chaque $1 \leq i \leq k$, en appliquant l'algorithme d'Euclide étendu, on trouve s_i et t_i tels que

$$1 = s_i n_i + t_i \frac{n}{n_i}$$

car n_i et $\frac{n}{n_i} = n_1 n_2 \cdots n_{i-1} n_{i+1} \cdots n_k$ sont relativement premiers. Alors,

$$x = \sum_{i=1}^{k} a_i t_i \frac{n}{n_i}$$

C'est une solution car

$$t_i \frac{n}{n_i} = 1 - s_i n_i \equiv 1 \pmod{n_i} \text{ et } t_i \frac{n}{n_i} \equiv 0 \pmod{n_j} \text{ si } j \neq i$$

EXEMPLE 3.8. Soient $n_1 = 3$, $n_2 = 5$ et $n_3 = 7$ d'une part, et d'autre part $a_1 = 2$, $a_2 = 3$ et $a_3 = 2$. Alors, l'algorithme d'Euclide étendu donne

$$\begin{aligned} 1 &= 12 \cdot 3 - 1 \cdot 35 \\ 1 &= -4 \cdot 5 + 1 \cdot 21 \\ 1 &= -2 \cdot 7 + 1 \cdot 15 \end{aligned}$$

et l'algorithme des restes chinois donne

$$x = -2 \cdot 35 + 3 \cdot 21 + 2 \cdot 15 = 23$$

4. Fonction indicatrice d'Euler

DÉFINITION 4.1. On dit que deux nombres entiers a et b sont **relativement premiers** si $\text{pgcd}(a, b) = 1$.

DÉFINITION 4.2. La fonction $\varphi : \mathbb{N} \to \mathbb{N}$ définie par

$$\varphi(n) = \#P(n) \text{ où } P(n) = \left\{ k \in \mathbb{N} \;\middle|\; k < n \text{ et } \text{pgcd}(k, n) = 1 \right\}$$

et où $\#P(n)$ désigne le nombre d'éléments dans $P(n)$, est appelée la **fonction indicatrice d'Euler**. Elle associe à tout entier naturel n le nombre des entiers naturels inférieurs à n et relativement premiers à n.

REMARQUE 4.3. Remarquons que

$$\varphi(n) = \#\mathbb{Z}_n^*$$

PROPOSITION 4.4. *La fonction indicatrice d'Euler a les propriétés suivantes:*

(1) *Si p est premier, alors $\varphi(p) = p - 1$.*
(2) *Si p est premier et $r \in \mathbb{N}$, alors $\varphi(p^r) = p^{r-1}(p-1)$.*
(3) *Si $\text{pgcd}(m, n) = 1$, alors $\varphi(mn) = \varphi(m)\varphi(n)$.*

DÉMONSTRATION. (1) Si p est premier, alors $P(p) = \{1, 2, \cdots, p-1\}$ et par conséquent, $\varphi(p) = P(p) = p - 1$.

(2) Soit p un nombre premier et $r \in \mathbb{N}$. Alors, en vertu du lemme d'Euclide, tout diviseur de p^r est de la forme p^s avec $0 \leq s \leq r$. Il suit que,

$$[m < p^r \text{ et } \text{pgcd}(p^r, m) \neq 1] \;\Rightarrow\; m = kp$$

Par conséquent,

$$\varphi(p^r) = p^r - \#\{p, 2p, 3p, \cdots, p^2, (p+1)p, \cdots, p^r\} = p^r - p^{r-1} = p^{r-1}(p-1)$$

(3) En vertu du lemme chinois, il existe un isomorphisme d'anneaux entre $\mathbb{Z}_{mn}$ et $\mathbb{Z}_n \times \mathbb{Z}_m$. Par conséquent, $\mathbb{Z}_{mn}^*$ et $(\mathbb{Z}_n \times \mathbb{Z}_m)^*$ ont le même nombre d'éléments. De plus, de manière générale, si A et B sont des anneaux, on a $(A \times B)^* = A^* \times B^*$. Par conséquent,

$$\varphi(mn) = \#\mathbb{Z}_{mn}^* = \#\left(\mathbb{Z}_n \times \mathbb{Z}_m\right)^* = \#\left(\mathbb{Z}_n^* \times \mathbb{Z}_m^*\right) = \#\left(\mathbb{Z}_n^*\right) \cdot \#\left(\mathbb{Z}_m^*\right) = \varphi(m)\varphi(n)$$

□

THÉORÈME 4.5 (Formule d'Euler). *Soient a et n des nombres entiers. Alors*

$$\text{pgcd}(a, n) = 1 \;\Rightarrow\; a^{\varphi(n)} \equiv 1 \;(mod\; n)\;.$$

DÉMONSTRATION. C'est un corollaire du théorème de Lagrange qui affirme que le cardinal de tout sous-groupe d'un groupe fini divise le cardinal du groupe. En effet, soit $a \in \mathbb{Z}_n^*$ (rappelons que $a \in \mathbb{Z}_n^* \;\Leftrightarrow\; \text{pgcd}(a, n) = 1$). Notons m l'ordre de a, c'est-à-dire le plus petit entier m tel que $a^m \equiv 1 \pmod{n}$. Le nombre m est le cardinal du sous-groupe engendré par a. Par conséquent, en vertu du théorème de Lagrange, m divise $\#\mathbb{Z}_n^*$. De plus, $\#\mathbb{Z}_n^* = \varphi(n)$. Il suit que $a^{\varphi(n)} = a^{mk} \equiv 1 \pmod{n}$.

Donnons une preuve directe de la formule d'Euler. Soient $r_1, \cdots, r_{\varphi(n)}$ les nombres entiers inférieurs à n et relativement premiers à n. Comme a est également relativement premier à n, il suit, en vertu du lemme d'Euclide, que les nombres $ar_1, \cdots, ar_{\varphi(n)}$ sont aussi relativement premiers à n. De plus, pour tous $i \neq j$, on a $ar_i \not\equiv ar_j \pmod{n}$. En effet, sinon, on aurait

$$a(r_i - r_j) = kn \;\Rightarrow\; n \mid a \text{ ou } n \mid (r_i - r_j) \;\Rightarrow\; r_i = r_j$$

une contradiction, puisque par hypothèse, $i \neq j$. Par conséquent, les nombres

$$ar_1 \pmod{n}, \cdots, ar_{\varphi(n)} \pmod{n}$$

sont les mêmes nombres que $r_1, \cdots, r_{\varphi(n)}$, éventuellement dans un ordre différent. Il suit que

$$ar_1 \cdots ar_{\varphi(n)} \equiv r_1 \cdots r_{\varphi(n)} \pmod{n} \;\Rightarrow\; a^{\varphi(n)} r_1 \cdots r_{\varphi(n)} \equiv r_1 \cdots r_{\varphi(n)} \pmod{n}$$

$$\Rightarrow\; (a^{\varphi(n)} - 1) r_1 \cdots r_{\varphi(n)} = kn$$

Comme les nombres $r_1, \cdots, r_{\varphi(n)}$ sont premiers à n, il suit, en vertu du lemme d'Euclide, que $a^{\varphi(n)} - 1 \equiv 0 \pmod{n}$, ce qu'il fallait démontrer. □

5. Petit théorème de Fermat

THÉORÈME 5.1 (Petit théorème de Fermat). *Soit p un nombre premier et $a \in \mathbb{N}$. Si $p \nmid a$, alors $a^{p-1} \equiv 1 \pmod{p}$.*

DÉMONSTRATION. C'est un corollaire de la formule d'Euler. Comme $p \nmid a$, p et a sont relativement premiers et, comme nous l'avons montré, $\varphi(p) = p - 1$. Par conséquent, en vertu de la formule d'Euler, il vient

$$a^{p-1} = a^{\varphi(p)} \equiv 1 \pmod{p}$$

□

CHAPITRE 2

Le système cryptographique à clefs publiques RSA

Le système cryptographique RSA a été inventé en 1977 par Ronald Rivest, Adi Shamir et Len Adleman. Il aurait été découvert déjà en 1973 par Clifford Cocks travaillant pour le gouvernement britannique. Son travail était classé secret et il ne le publia pas. Le système RSA est utilisé quotidiennement à travers le monde pour crypter des messages.

FIGURE 1. Ronald Rivest (New York, 1947), Adi Shamir (Tel Aviv, 1952), Leonard Adleman (Californie, 1945)

1. Clefs RSA

DÉFINITION 1.1. Une **clé RSA** consiste en un 5-tuple de nombres (p, q, n, e, d) tels que:

(1) p et q sont des nombres premiers,
(2) $n = pq$
(3) e est relativement premier à $\varphi(n)$,
(4) $de \equiv 1 \pmod{\varphi(n)}$.

Le couple (n, e) est appelé la **clé publique** et le triplet (p, q, d) la **clé privée**. Par ailleurs, le nombre n est appelé le **module** et les nombres e et d respectivement l'**exposant public** et l'**exposant privé**.

Théorème RSA.

Le théorème qui suit est une conséquence du petit théorème de Fermat. Il est la clef de voûte du protocole RSA.

THÉORÈME 1.2 (RSA). *Soit (p, q, n, e, d) une clé RSA et m un nombre entier $< n$. Alors,*

$$m^{ed} \equiv m \pmod{n}$$

Démonstration. Si $p \nmid m$, en vertu du petit théorème de Fermat, $m^{p-1} \equiv 1 \pmod{p}$. Par conséquent, pour tout $k \in \mathbb{N}$,

$$m^{k(p-1)} \equiv (m^{p-1})^k \equiv 1^k \equiv 1 \pmod{p}$$

et donc

$$m^{k(p-1)+1} \equiv m \pmod{p}$$

Remarquons que cette égalité est également vraie si $p \mid m$, car dans ce cas (notons $m = sp$)

$$m^{k(p-1)+1} - m = (sp)^{k(p-1)+1} - sp = p\left(s^{k(p-1)+1}p^{k(p-1)} - s\right)$$

Comme $ed \equiv 1 \pmod{\varphi(n)}$ et $\varphi(n) = (p-1)(q-1)$ (voir plus haut), on trouve, en vertu du petit théorème de Fermat, que

$$m^{ed-1} \equiv m^{(p-1)k_2} \equiv 1^{k_2} \equiv 1 \pmod{p} \text{ et } m^{ed-1} \equiv m^{k_1(q-1)} \equiv 1^{k_1} \equiv 1 \pmod{q}$$

où $(q-1) \mid k_2$ et $(p-1) \mid k_1$. En d'autres termes, il existe des nombres entiers a et b tels que

$$m^{ed-1} - 1 = ap = bq$$

En vertu du théorème fondamental de l'arithmétique, il suit que $p \mid b$ et donc il existe un nombre entier c tels que $b = cp$. Par conséquent,

$$m^{ed-1} - 1 = bq = cpq = cn \;\Rightarrow\; m^{ed-1} \equiv 1 \pmod{n} \;\Rightarrow\; m^{ed} \equiv m \pmod{n}$$

□

2. Le protocole RSA

Roméo veut écrire un message secret à Juliette. Notons,

$$(p_R, q_R, n_R, e_R, d_R)$$

la clé RSA de Roméo et

$$(p_J, q_J, n_J, e_J, d_J)$$

celle de Juliette. Rappelons que les clés publiques (n_R, e_R) et (n_J, e_J) sont connues de tous, elles sont, par exemple, publiées dans un annuaire de clés RSA.

(1) Roméo transforme son message en un nombre $m < n_J$.
(2) Roméo calcule avec la clé publique de Juliette

$$M = m^{e_J} \pmod{n_J}$$

qu'il envoie, par un canal quelconque, à Juliette.
(3) Juliette, en vertu du théorème RSA, calcule avec sa clé privée

$$M^{d_J} \equiv m^{e_J d_J} \equiv m \pmod{n_J}$$

et transforme le nombre m en texte selon l'algorithme inverse à celui utilisé par Roméo pour transformer son texte en nombre.

3. Signature électronique

Le système RSA permet non seulement d'envoyer des messages lisibles uniquement par leurs destinataires, mais aussi d'éviter l'usurpation d'identité grâce au protocole de signature électronique:

(1) Roméo transforme son message en un nombre $m < \min(n_J, n_R)$ et calcule

$$S = m^{d_R} \pmod{n_R}$$

puis

$$M = S^{e_J} \pmod{n_J}$$

qu'il envoie, par un canal quelconque, à Juliette.

(2) Juliette, en vertu du théorème RSA, calcule avec sa clé privée

$$M^{d_J} \equiv S^{e_J d_J} \equiv S \pmod{n_J}$$

puis avec la clé publique de Roméo

$$S^{e_R} \equiv m^{d_R e_R} \equiv m \pmod{n_R}$$

et transforme le nombre m en texte selon l'algorithme inverse à celui utilisé par Roméo pour transformer son texte en nombre.

Si le message est lisible, c'est qu'il a été écrit par Roméo car seul Roméo connaît d_R (ou la personne qui le menace !). Il est facile d'imaginer des variantes de ce protocole de signature électronique comme par exemple: Roméo et Juliette conviennent par un canal non sécurisé d'une signature s et Roméo ajoute à la fin de son message $s^{d_R} \pmod{n_R}$ avant d'encrypter le tout avec la clé publique de Juliette. Juliette, après avoir décrypté tout le message avec sa clé privée, décrypte la signature avec la clé publique de Roméo. Si elle retrouve le mot convenu, elle est certaine que le message a été envoyé par Roméo.

4. Attaquer RSA

Dans cette partie, nous considérons la situation suivante: Roméo écrit à Juliette en utilisant le protocole décrit plus haut. Un espion, Pâris, intercepte ce message. Il sait qu'il est destiné à Juliette et aimerait en connaître le contenu. Bien entendu, s'il réussit à factoriser n_J, il peut décrypter le message.

La sécurité du protocole RSA est basée sur deux postulats:

(1) Décrypter le message est aussi long que factoriser n_J.
(2) Factoriser n_J nécessite un temps rédhibitoire (plusieurs milliers d'années).

Ces postulats ne sont pas toujours vrais. Nous allons voir quelques précautions élémentaires à prendre pour éviter des failles de sécurité dans le protocole RSA. La question est assez complexe et largement ouverte. Avant cela, commençons par quelques remarques reliées au premier postulat.

Remarque 4.1. Si la factorisation de n est connue, alors $\varphi(n)$ se calcule facilement car, comme nous l'avons vu, $\varphi(n) = (p-1)(q-1)$. Réciproquement, si n et $\varphi(n)$ sont connus, la factorisation de n est rapide. En effet,

$$\varphi(n) = (p-1)(q-1) = pq - (p+q) + 1 \;\Rightarrow\; p + q = \varphi(n) - n - 1 \text{ et } pq = n$$

et trouver p revient à résoudre une équation du second degré.

1. Algorithme *Las Vegas* pour factoriser n à partir de d.
Commençons par quelques rappels.

(1) Soit $n = pq$ avec p et q premiers. Alors

$$y \equiv 1 \pmod{n} \;\Leftrightarrow\; [y \equiv 1 \pmod{p} \text{ et } y \equiv 1 \pmod{q}]$$

En effet, d'une part

$$y \equiv 1 \pmod{n} \;\Rightarrow\; y = 1 + kn = 1 + kpq \;\Rightarrow\; [y = 1 + rp \text{ et } y = 1 + sq]$$

et d'autre part, en vertu du lemme d'Euclide,

$$[y \equiv 1 \pmod{p} \text{ et } y \equiv 1 \pmod{q}] \;\Rightarrow\; [y = 1 + rp \text{ et } y = 1 + sq] \;\Rightarrow\; rp = sq$$

$$\Rightarrow\; p \mid s \text{ (et } q \mid r) \;\Rightarrow\; y = 1 + sq = 1 + lpq = 1 + ln \;\Rightarrow\; y \equiv 1 \pmod{n}$$

(2) Soit p un nombre premier. Alors

$$x^2 \equiv 1 \pmod{p} \;\Rightarrow\; x \equiv \pm 1 \pmod{p}$$

En effet, comme $\mathbb{Z}_p^*$ est un corps, le polynôme $x^2 - 1 \in \mathbb{Z}_p^*[x]$ admet au plus deux racines. De manière générale, si $\mathbb{K}$ est un corps et $p \in \mathbb{K}[x]$ est un polynôme de degré m, alors p admet au maximum m racines dans $\mathbb{K}$. Procédons par récurrence sur le degré m de p. Si $m = 1$, c'est évident. Supposons l'affirmation démontrée pour tout polynôme de degré $m - 1$ et soit p un polynôme de degré m et x_0 une racine de p. Par l'algorithme de la division euclidienne, on trouve

$$p(x) = q(x)(x - x_0) + r$$

où q est de degré $m - 1$ et r est de degré 0, c'est-à-dire $r \in \mathbb{K}$. Comme $p(x_0) = 0$ il suit que $r = 0$. Si x_1 est une autre racine de p, alors $p(x_1) = 0 = q(x_1)(x_1 - x_0)$. Comme $(x_1 - x_0) \neq 0$, il suit que x_1 est une racine de q, qui, par hypothèse de récurrence, admet au plus $m - 1$ racines. Par conséquent, p admet au plus m racines.

Remarque 4.2. Soit $n = pq$ avec p et q premiers. Alors,

$$x^2 \equiv 1 \pmod{n} \;\Leftrightarrow\; [x^2 \equiv 1 \pmod{p} \text{ et } x^2 \equiv 1 \pmod{q}]$$

$$\Leftrightarrow\; [x \equiv \pm 1 \pmod{p} \text{ et } x \equiv \pm 1 \pmod{q}]$$

L'équation de départ $x^2 \equiv 1 \pmod{n}$ admet donc quatre solutions. Deux d'entre elles sont triviales

$$[x \equiv 1 \pmod{p} \text{ et } x \equiv 1 \pmod{q}] \text{ ou } [x \equiv -1 \pmod{p} \text{ et } x \equiv -1 \pmod{q}]$$

$$\Rightarrow\; x \equiv \pm 1 \pmod{n}$$

Soit x une solution non triviale, c'est-à-dire $x \not\equiv \pm 1 \pmod{n}$. Alors,

$$x^2 \equiv 1 \pmod{n} \;\Rightarrow\; x^2 - 1 = kn \;\Rightarrow\; (x - 1)(x + 1) = kn \;\Rightarrow\; n \mid (x + 1)(x - 1)$$

mais n ne divise ni $(x + 1)$ ni $(x - 1)$ (car, si, par exemple, $n \mid (x + 1)$, alors, $x + 1 = rn$, et $x = -1 + rn$, ce qui est impossible puisque x est une solution non triviale). Par conséquent, en

vertu du lemme d'Euclide,

$$n \mid (x+1)(x-1) \;\Rightarrow\; pq \mid (x+1)(x-1)$$
$$\Rightarrow\; [(p \mid (x+1) \text{ et } q \mid (x-1)) \text{ ou } (p \mid (x-1) \text{ et } q \mid (x+1))]$$

et donc

$$\text{pgcd}(x+1, n) = p \text{ ou } q \text{ et } \text{pgcd}(x-1, n) = p \text{ ou } q$$

car n ne divise ni $(x+1)$ ni $(x-1)$.

EXEMPLE 4.3. Soit $n = 3 \cdot 5 = 15$. Alors,

$$x \equiv 1 \pmod 3 \;\Rightarrow\; x = \cdots,\ 1,\ 4,\ 7,\ 10,\ \cdots$$
$$x \equiv -1 \pmod 3 \;\Rightarrow\; x = \cdots,\ 2,\ 5,\ 8,\ 11,\ 14,\ \cdots$$
$$x \equiv 1 \pmod 5 \;\Rightarrow\; x = \cdots,\ 1,\ 6,\ 11,\ 16,\ \cdots$$
$$x \equiv -1 \pmod 5 \;\Rightarrow\; x = \cdots,\ 4,\ 9,\ 14,\ 19,\ \cdots$$

Les quatre solutions de l'équation $x \equiv 1 \pmod{15}$, sont 1, -1, 4 et 11. Les deux premières sont triviales. Par ailleurs,

$$\text{pgcd}(4+1, 15) = 5,\ \text{pgcd}(4-1, 15) = 3,\ \text{pgcd}(11-1, 15) = 5,\ \text{pgcd}(11+1, 15) = 3,$$

Pour factoriser n il suffit (!) donc de trouver une solution non triviale de l'équation

$$x^2 \equiv 1 \pmod n$$

Rappelons que $ed \equiv 1 \pmod{\varphi(n)}$ et en vertu du théorème RSA

$$m^{ed} \equiv m \pmod n \text{ et } m^{ed-1} \equiv 1 \pmod n$$

pour tout nombre $m < n$. De plus, comme $\varphi(n)$ est pair, $ed - 1$ est pair et

$$\left(m^{\frac{ed-1}{2}}\right)^2 \equiv 1 \ (\bmod n)$$

Par conséquent, si

$$m^{\frac{ed-1}{2}} \not\equiv \pm 1 \pmod n$$

alors on peut factoriser n en calculant par exemple

$$\text{pgcd}\left(m^{\frac{ed-1}{2}} - 1, n\right)$$

Algorithme type Las Vegas pour trouver une solution non triviale de l'équation $x^2 \equiv 1 \pmod n$ et pour factoriser n connaissant d:

(1) Trouver le nombre entier s tel que

$$ed - 1 = 2^s(2k+1)$$

(2) Choisir au hasard un nombre $0 < m < n$.

(3) Trouver

$$t_0 = \min\left\{t \;\middle|\; m^{2^t(2k+1)} \equiv 1 \pmod{n}\right\}$$

Si $t_0 = 0$, recommencer au point (2). Sinon, définir

$$v_0 = m^{2^{t_0-1}(2k+1)}$$

Ainsi

$$v_0 \not\equiv 1 \pmod{n} \text{ et } v_0^2 \equiv 1 \pmod{n}$$

(4) Si $v_0 \equiv -1 \pmod{n}$, recommencer au point (2). Sinon, calculer

$$\text{pgcd}(v_0 - 1, n)$$

et factoriser n.

Nous allons montrer (Théorème de Miller-Rabin) que le nombre de m qui pour lesquels il faut recommencer au point (2) est majoré par $\frac{\varphi(n)}{4}$. Par conséquent, la probabilité de factoriser n avec un m donné est $\geq \frac{3}{4}$.

Réalisation concrète La fonction pgcd2 du code *Python* donné ci-dessous calcule le pgcd selon l'algorithme d'Euclide étendu. La fonction factorisernavecd est une retranscription de l'algorithme donné ci-dessus. Les nombres p et q sont trouvés grâce à la fonction sympy.ntheory.generate.nextprime du module sympy qui trouve le premier nombre premier plus grand que le nombre passé en paramètre (ici, il est tiré au hasard).

```
1  import random
2  import sympy
3  def pgcd2(a,b):
4      s1=1
5      t1=0
6      s2=0
7      t2=1
8      a1=max(a,b)
9      b1=min(a,b)
10     r1=a1%b1
11     while r1>0:
12         q1=a1//b1
13         s=s1-q1*s2
14         t=t1-q1*t2
15         t1=t2
16         s1=s2
17         t2=t
18         s2=s
19         a1=b1
20         b1=r1
21         r1=a1%b1
22     if a>=b:
23         return [b1,s2,t2]
24     else:
25         return [b1,t2,s2]
26 def factorisernavecd(n,d,e):
27     fact=1
28     passage=0
29     while fact==1:
30         passage=passage+1
31         m=random.randint(1,n)
32         if passage==1:
33             print "\n(",passage,") m=",m
34         else:
35             print "\n(",passage,") m=..."
```

```
36          x=d*e-1
37          s=0
38          while x%2==0:
39              x=x/2
40              s=s+1
41          t=s
42          while pow(m,2**t*x,n)==1 and t>0:
43              t=t-1
44          if t!=0:
45              v0=pow(m,2**(t-1)*x,n)
46              fact=pgcd2(v0-1,n)[0]
47      return fact
48  #~ p=sympy.ntheory.generate.nextprime(random.randint(1,10**300))
49  p=int('23109302447244866061191314588514601780957969698381954186989509784131561596006554164000714001617107587677799\
50  65541640007140016171075876777799357071713554657714787419438229854394164262408 1\
51  23130989654935709832618008242008948149037766699790213009747128890\
52  12729253406096418420955962856562463925869769224955425966994843768 21')
53  #~ q=sympy.ntheory.generate.nextprime(random.randint(1,10**300))
54  q=int('30323438915609234219974308887599502743989167081674860468094052955188548352165067888417755530433466260 3\
55  50678884177555304334662603534991231476882609797325757560800778717889957117835 8\
56  68867931811134670352709598445058973845275369503418165008128990244604 6\
57  31595673051914355687717434175440797359597769059996424700273772954281')
58  n=p*q
59  phin=(p-1)*(q-1)
60  #~ e=random.randint(1,10**20)
61  #~ while pgcd2(e,phin)[0]!=1:
62      #~ e=e+1
63  e=79501184606768945987
64  d=pgcd2(e,phin)
65  d=d[1]%phin
66  #
67  print "\n p=",p,"\n q=",q,"\n e=",e,"\n d=",d,"\n n=",n
68  for j in xrange(0,10,1):
69      fact=factorisernavecd(n,d,e)
70      if fact==p:
71          print "\n fact=p"
72      if fact==q:
73          print "\n fact=q"
```

L'exécution du code donne par exemple, en environ 3 secondes:

```
 p=23109302447244866061191314588514601780957969698381954186989509784131561596006554164000714001617107587677799
3570717135546577147874194382298543941642624081231309896549357098326180082420089481490377666997902130097471288902987279249049127292534060964184209559628565624639258697692249554259669948437682 1
 q=30323438915609234219974308887599502743989167081674860468094052955188548352165067888417755530433466260360534991
2314768826097973257575608007787178899571178358688679318111346703527095984450589738452753695034181650081289902446048555143197
6048555143197315956730519143556877174341754407973595977690599964247002737729542 81
 e= 79501184606768945987
 d=4126577735083856674211751773807519737623185429160593922612242842012286985687241593000693852198921403728470762989366725837816556565947593902802946356094227893666194346114450058215343388030282187067558763628353598078116476019077727163703176394007945898335289712780463468679702183504364463168947678062195438492277581133286211405651593509246514629929190894403005035410439760698680014811904173483103012677296347744113951511200125882805546414882849974207943464164547972747149667045845914867239177630187630760459972683779670848938338358049415459922459954092151011809624289897121233110890597831871714357 23
 n=7007535211413686839044890934904959963406884589858613387774273742637957379493287390135839658644823046723325786984909508923078823325354645607346435929841751212397850585084075751574368626736350202366345123723282465631015276120838951858900481101649510208666995448124357558860722032237178433701134571521656357079341498379333886345721123893878206266298111621275624001790920901469294432501655730092104842441215283191904257286970838955803516454453448826547338757286511030489757387410857320371461648803468897355298049290923140160671389433407996202584191182352936180987254791028369555520047836491689409120701

( 1 ) m=35834543382821675755943579645222379578638423522795099996426804601434657337037068692163887028151936720
1989077171544033485120992297297877513771399717489560004929086424645403919258770569036319366028567419855096147
1861814235919145699601073791830714820785451853193760413859202426916372741978946086545912237008126931402628180
8133278675257801085410294780598124084977141216955300058272548636186179283300899245660948737032593476971966111
7898865915933853147608724826577543842168426731212238421310905522236389049654425207058398508272404477088368968
```

258835810380917503323903300035773442130662561235732032832450 9

(2) m=...

(3) m=...

(4) m=...

fact=q

(1) m=14864750639470751056074344184255748837700205894658269303031690432224292624982996005422993716361620 5
17496020556213450468766406503616974146573168531068732801445820104857210524632216958861214594765221687679543
18883698581951974359744071753652902404936131990231958303038785178641707577554453779441802090332052567479522
31941822960346706692426849912945734831909468497031615152186908899629652634561667027215929298989112889773557
40728983470702043570439116019991980628203940238611098414004513240014101210275721128112169812836496657322745
11831959968691708345731738492889761847215408898885723833893644671248314

(2) m=...

(3) m=...

(4) m=...

(5) m=...

(6) m=...

(7) m=...

(8) m=...

(9) m=...

(10) m=...

fact=p

(1) m=41660599845433310558677383586702075780501729296201584808634895074979542952001716102434283195141386 5
19079991852849363428764412836555794861764930222597186136418876960226206645460654545254330135153102015716 1
95910214140850767841877663244606938500965930735470340958512247326097708174660222998230159431752220874169 24
55424729634604321880053005064885514082947389297861689750313477794726665743989294670112465745913323757077 02
06628270536701904046822327142558258548972767323532922551931854692994985503858071259324808709331840060758 77
14209452403299347327833478651379316688640809400840008758791929744765084791 87

fact=p

(1) m=45689728815286215712323497503099270661459382625463568523800483011199010463444237820296083215327 60
14643837810730782319866657457719054489200256450848065894858349222418365155168319636016359393823504923949 3
31690856800280718826964762279785725650276619727047997430917198813520747693857201094141280139157468873213 1
62756130457757206392554611170500776569578726300278056909127124722240682314556566200275802114001269172727 3
86610855042909484764971999567400074166722036472502232807507118251452070072433663809127361051016685837823 1
87919095961613552114596428733133491549128085279157951973727693154851917059199031 5

fact=p

(1) m=45981431971430091122387453858372403740458806998162970548407492014685418030542201971071161860287 70
07809149020991015231366915457040603713331329235670649936365539820755154145621832379747038860201273265127 6
85517027592433974913407150164025684529898807864416529313243256888772172490756834510818295272312977876744 6
21156805559764561163801312776748968675141071089404700617077377727297505358655982606335362915922097010802 0
20447034503936587470409187583865235369107837804532675515386700993449906705697931602892688043970929785797 4
39498174356860068683387177992765166657456318611953671942829695237423202786435034 4

(2) m=...

fact=q

```
( 1 ) m=347390796819223579351397108395823643968961052115228743327863402454369909864724990527203268415951612
112052161768056133982458945043714519247096444625351767616114314222425091810957194869586932825893351318336048
888306619916816468121113703115495925000187183881869383016391826855980654118128371120148147547210462760835290
709975149330563504634394525321525082235656558324429593835815847983180686985752572397894475238132318809210893
381974426902038527351559480057129269000637422806773212063122889086599184643513399116472174509868987473083279
987991062798689811364914747494343387842360778782806197331728702509047 7

 fact=p

( 1 ) m=565794088820434354966916396954099136053633344669898701016190478635163959346893296332149546908861653210
070170787073442582538936744760800597654854314337646943821266061921298194599290093649170921017705422993183520
592443740585768731398984575780943206441905183713722265929991221760261360054524543305024875048037764957909439
215232900211034192708597523783985040150365985230845282482444230395129691446296943616901118086027517635564329
484842169096217219242558164489492072968458972562853545393310947303124804773076016914607776472292116737914219
597534426532129649965330490434514905470749868028677860254903606198 7

 fact=p

( 1 ) m=256286047955510214077422218891840014024697385440017780423226216344040978863311490668417696843334437098
840687047142420834569712253276222295751196929156102513436431935915190298127481096837027682593292598019472249
725280035707690568084251733677553121424299015698444285254225423031290826292626302570615879208344361023101414
140870691744144317247903030541518084553355866191114289152188129720064812663495034073766983730763045214055548
488856372585334246989479242034431863567578224434106953524965635574200352988223456680908077180406686102972803
607171198521404444004259295498694848336201136108107583936112934 23

( 2 ) m=...

 fact=p

( 1 ) m=423590583765949958388828292313668205197823550495708762645534132582009077034777582376327881552016641929
811228426264730883656178521901011459207165203597708432635843653724225492940656327734881408010487829751104549
083460378497235064909409441532407101974904485956221369491481315684781566569957631177844282857394053944083490
047508869489101325606518267022690146597476276737540602495149168065095919815295001006305683876904144703746246
766315921947593109970915501757814557187781207000672972266085297936361961567450749223666498665279090720019546
555051859754745356003725204625261034388538047914470295489429730

 fact=q

( 1 ) m=108894408918393724846690765313049274737176435982519572824358297977346141738093322020548018611249732229
590336457088606006360845213473260438730323687150158312912631405042675957367917129869145160178429297503895843
182408295679682210482424356830634989989001629046242133653943178810880477155046229872389719481911878094911048
048139243614309938711840289266577641240062014373531665517844795807542165035112801862417496794237174044232345
142014734073939927066262086694969259110176799610019734242854722387687908509156419711522859454025993121033474
892580928156124332469639589839560871285593546803129482952548391 6

( 2 ) m=...

 fact=p
```

2. Précautions à prendre dans le choix de la clé privée.

Pour éviter que n soit facilement factorisable, quelques précautions sont de rigueur au moment du choix de la clé privée.

- Il faut que $|p-q|$ soit grand. En effet, écrivons $q = p+\delta$ et supposons que $\frac{\delta}{p} \ll 1$. Alors,

$$\sqrt{N} = \sqrt{pq} = \sqrt{p(p+\delta)} = p\sqrt{1+\frac{\delta}{p}} \approx p\left(1+\frac{\delta}{2p}\right) = p+\frac{\delta}{2}$$

Par conséquent, si $|p-q|$ est petit, on peut trouver p en partant de $\sqrt{N}$ par un algorithme naïf en δ étapes. Bien entendu, p et q doivent également être suffisamment grands.

- Les nombres p et q ne doivent pas être des nombres premiers "spéciaux" comme par exemple, des nombres premiers de Mersenne de la forme $M_r = 2^r - 1$ avec r premier, ou des nombres premiers de Fermat de la forme $F_n = 2^{2n}+1$. Pour éviter ces nombres, il suffit de choisir un grand nombre p_0 au hasard et de chercher le premier nombre premier $p \geq p_0$. La probabilité de tomber alors sur des nombres premiers "spéciaux" est négligeable.

- **Attaque de Wiener** L'exposant privé d ne doit pas être trop petit. En effet, rappelons que

$$ed \equiv 1 \pmod{\varphi(n)} \;\Rightarrow\; ed = 1 + k'\varphi(n)$$

$$= 1 + k'(p-1)(q-1) = 1 + k'(n-(p+q)+1) \overset{(k=2k')}{=} 1 + k\left(\frac{n+1}{2} - \frac{p+q}{2}\right)$$

$$\Rightarrow\; \frac{2e}{n} = \frac{2}{dn} + \frac{k}{dn}(n+1-p-q) \;\Rightarrow\; \left|\frac{2e}{n} - \frac{k}{d}\right| = \frac{|2+k(1-p-q)|}{dn}$$

Par conséquent,

$$\text{si } \frac{|k(p+q-1)-2|}{n} < \frac{1}{2d} \text{ alors } \left|\frac{2e}{n} - \frac{k}{d}\right| < \frac{1}{2d^2}$$

Cette dernière inégalité montre que $\frac{k}{d}$ est une réduite du développement en fraction continue de $\frac{2e}{n}$. Il suffit alors d'effectuer le développement en fraction continue de $\alpha = \frac{2e}{n}$:

$$\begin{aligned}
&\alpha_0 = \frac{2e}{n},\ a_0 = \lfloor \alpha \rfloor && \Rightarrow\ \frac{p_0}{q_0} = a_0 \\
&\alpha_1 = \frac{1}{\alpha_0 - a_0},\ a_1 = \lfloor \alpha_1 \rfloor && \Rightarrow\ \frac{p_1}{q_1} = a_0 + \frac{1}{a_1} \\
&\alpha_2 = \frac{1}{\alpha_1 - a_1},\ a_2 = \lfloor \alpha_2 \rfloor && \Rightarrow\ \frac{p_2}{q_2} = a_0 + \frac{1}{a_1 + \frac{1}{a_2}} \\
&\vdots && \vdots \\
&\alpha_{k+1} = \frac{1}{\alpha_k - a_k},\ a_{k+1} = \lfloor \alpha_{k+1} \rfloor && \Rightarrow\ \frac{p_{k+1}}{q_{k+1}} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots\ a_k + \frac{1}{a_{k+1}}}}}
\end{aligned}$$

et, pour chaque réduite $\frac{p_k}{q_k}$, de tester si le dénominateur q_k est le d cherché en essayant de factoriser n avec q_k selon la méthode développée plus haut (l'algorithme *Las Vegas*).

Voyons maintenant dans quels cas l'inégalité donnée ci-dessus est satisfaite. Pour commencer, remarquons que si $e < \phi(n)$, alors

$$1 = de - k'\varphi(n) < d\varphi(n) - k'\varphi(n) = (d-k')\varphi(n) \;\Rightarrow\; d - k' > \frac{1}{\varphi(n)} \approx 0$$
$$\Rightarrow\; 0 < k' < d \;\Rightarrow\; 0 < k = 2k' < 2d$$

De plus, si p et q ont la même taille, soit de l'ordre de $\sqrt{n}$ (*i.e.* $p, q < \gamma\sqrt{n}$ avec γ "petit"), alors

$$\frac{|k(p+q-1)-2|}{n} < \frac{2d2\gamma\sqrt{n}}{n} = \frac{4d\gamma}{\sqrt{n}}$$

et

$$\frac{4d\gamma}{\sqrt{n}} < \frac{1}{2d} \;\Rightarrow\; d < \frac{\sqrt[4]{n}}{\sqrt{8\gamma}}$$

Par conséquent,

$$\text{si } d < \frac{\sqrt[4]{n}}{\sqrt{8\gamma}} \text{ alors } \frac{|k(p+q-1)-2|}{n} < \frac{1}{2d}$$

Il existe des améliorations de l'attaque de Wiener, notamment celle de **Boneh-Durfee** qui nécessite seulement $d < Cn^{0.292}$ où C est une constante indépendante de n.

3. Précautions à prendre dans le choix de la clé publique.

- **Attaque contre modules communs** Supposons que deux utilisateurs aient le même module, donc des clés publiques de la forme (n, e_A) et (n, e_B). Premièrement, l'utilisateur A peut déchiffrer les messages destinés à B et réciproquement. De plus, si e_A et e_B sont relativement premiers, alors par l'algorithme d'Euclide étendu, il est facile de trouver deux nombres s et t tels que

$$te_A + se_B = 1$$

Si un utilisateur envoie le même message m à A et B, il calcule

$$M_A = m^{e_A} \pmod{n} \text{ et } M_B = m^{e_B} \pmod{n}$$

Alors, il est facile de retrouver m:

$$M_A^t M_B^s \equiv m^{te_A} m^{se_B} \equiv m^{te_A + se_B} \equiv m \pmod{n}$$

- **Attaque par "broadcast"** Considérons e utilisateurs qui ont le même exposant public e et dont les clés publiques sont respectivement: (n_1, e), $(n_2, e), \cdots, (n_e, e)$. Supposons que $n_1, \cdots, n_e$ soient premiers deux-à-deux (sinon, on peut facilement factoriser n_i en cherchant le pgcd avec l'algorithme d'Euclide étendu). Supposons maintenant qu'un expéditeur envoie un message commun $m < \min\{n_i\,;\, 1 \le i \le e\}$ à ces e utilisateurs et que Pâris intercepte ces e messages. Il connaît alors

$$c_1 \equiv m^e \pmod{n_1},\; c_2 \equiv m^e \pmod{n_2},\; \cdots,\; c_e \equiv m^e \pmod{n_e}$$

et, avec l'algorithme des restes chinois, également

$$c \equiv m^e \pmod{n_1 \cdots n_e}$$

Mais comme $m^e < n_1 \cdots n_e$, on a l'égalité $c = m^e$. Il suffit donc à Pâris de calculer la racine $\sqrt[e]{c}$ pour trouver m.

Pour éviter des attaques en "broadcast", il suffit de prendre e suffisamment grand.

4. Précautions à prendre pour l'encryptage du texte.

- Si chaque lettre d'un message est transformée en nombre (par exemple avec le code ASCII) puis encryptée avec le protocole RSA, alors il est possible de décrypter un message sans connaître la clé privée du destinataire. En effet, il suffit alors à Pâris d'encrypter, avec la clé publique de Juliette, successivement les lettres de l'alphabet pour obtenir un dictionnaire qui pour chaque lettre donne le nombre crypté correspondant. Il est donc indispensable de diviser le texte en blocs de lettres. La taille des blocs est fixée par la grandeur de n, soit, en utilisant le code ASCII (un octet=8 bits par lettre), $\log_{256}(n)$. Donc si n est un nombre décimal de 300 chiffres, on peut diviser le texte en blocs de

$$\log_{256}(10^{300}) = 300 \log_{256}(10) = \frac{300}{\log_{10}(256)} \approx 124 \text{ lettres.}$$

On utilise généralement un nombre n de 1024 bits en base 2 ce qui donne $\log_{10}(2^{1024}) = 1024 \log_{10}(2) \approx 308$ chiffres en base 10.

Si, par exemple, on peut diviser le texte en blocs de 40 lettres, il est judicieux de couper le texte en blocs de 30 lettres et d'ajouter au début ou à la fin de chaque bloc dix lettres choisies aléatoirement. Ainsi, un message ne sera pas encrypté deux fois de suite de la même manière, ce qui protège le protocole contre une attaque du type décrit ci-dessus.

- Le message m doit être relativement premier à $n = pq$, sinon il suffit de calculer $\text{pgcd}(m, n)$ pour trouver p ou q. Remarquons que si le message m n'est pas premier à n, il peut toujours être décodé. Remarquons également que la proportion de messages m premiers à n est donnée par

$$\frac{\varphi(n)}{n} = \frac{(p-1)(q-1)}{pq} = \left(1 - \frac{1}{q}\right)\left(1 - \frac{1}{q}\right)$$

Par conséquent, si, par exemple, p et q sont $\geq 10^{100}$, la proportion de messages (d'entiers $< n$) non premiers à n est $\leq 2 \cdot 10^{-100}$!

5. Attaques par analyse de paramètres physiques.

Une attaque due à **Kocher**, consiste à mesurer le courant consommé lors du décryptage (par exemple dans le cas de cartes à puces). Notons $d = d_0 + d_1 2 + d_2 2^2 + \cdots$. Pour calculer $m^d \pmod{n}$ par l'algorithme donné plus loin, lors du $k^{\text{ème}}$ passage dans la boucle, on effectue une multiplication si $d_k = 0$ et 2 multiplications si $d_k = 1$, ce qui se traduit par une consommation de courant plus grande si $d_k = 1$ et plus petite si $d_k = 0$. Ces variations peuvent être mesurées et peuvent servir à reconstituer d.

6. Attaque par factorisation du module n.

Pour terminer, nous considérons brièvement le problème de la factorisation du module n. Nous avons déjà vu que la méthode naïve consistant à tester tous les nombres entre 2 et la partie entière de $\sqrt{n}$ est en pratique inapplicable. Notons L le nombre de chiffres en base 2 (typiquement $L = 1024$ bits). Alors, le temps d'exécution de cet algorithme est proportionnel à

$$T = C_0 2^{L/2}$$

où C_0 est une constante.

Il existe des algorithmes plus performants. Un des meilleurs donne

$$T = C_0 2^{C_1 \sqrt{\ln(p)\ln(\ln(p))}}$$

où p est le plus petit facteur premier de n et C_0 et C_1 sont des constantes. Dans le cas $p \sim q \sim \sqrt{n}$, il vient

$$T = C_0 2^{C_1 L^\kappa}, \ \kappa < 1$$

Ces algorithmes sont dits **sous-exponentiels**. En pratique, en 2006 il était possible de factoriser en quelques heures un nombres entier de 150 chiffres (soit environ 330 bits) avec plusieurs ordinateurs. De nos jours, il semblerait que la NSA soit capable de factoriser des modules de 1024 bits (environ 300 chiffres en base 10) et il est recommandé de passer à des modules de 2048 bits (environ 600 chiffres en base 10).

Mentionnons qu'avec l'algorithme de **Shor**, il est possible de factoriser un nombre en un temps polynomial en L. Cet algorithme doit fonctionner sur un ordinateur quantique. Malgré les annonces de la société *D-Wave*, un tel ordinateur n'existe pas encore.

Finalement, on ne sait toujours pas s'il existe un algorithme classique qui permette de factoriser un nombre en un temps polynomial en L.

5. Réalisation concrète

Dans cette partie, nous voulons écrire un code *Python* permettant de crypter et décrypter des messages selon le protocole RSA. Nous allons tenir compte explicitement seulement de certaines des précautions préconisées dans la section qui précède.

La confection des clefs RSA sera traitée plus loin. Elle nécessite la construction aléatoire de très grands nombres premiers (environ 150 chiffres) et donc des tests de primalité efficaces.

1. Comment calculer m^e (mod n) ?

L'approche naïve consiste à calculer $m^e = e \cdot e \cdots e \pmod{n}$ de manière récursive comme dans le script *Python* donné ci-dessous:

```
1  def puissance(m,e,n):
2      m0=m%n
3      r=1
4      k=1
5      while k<=e:
6          r=r*m0
7          r=r%n
8          k=k+1
9      return r
10 m=input("m=")
11 e=input("e=")
12 n=input("n=")
13 print "m^e mod n=", puissance(m,e,n)
```

Le langage *Python* possède une commande qui effectue ce calcul: pow(m,e,n). L'exécution sur un ordinateur familial du script donné ci-dessous est "instantanée":

```
1 import random
2 m=random.randint(1,10**300)
3 e=random.randint(1,10**100)
4 n=random.randint(1,10**300)
5 print "m=",m
6 print "e=",e
7 print "n=",n
8 print "m^e mod n=",pow(m,e,n)
```

On trouve, par exemple:

```
python comp.py
m= 308489503258407332617737066722422604683853149003297648332568346838152016886248162168473733
794206721391552179236879881664303408541062815116434264433665322804521403093600083819335471150
386633253748174930676633290024075331018918499792039853664551605427114727318375919465536004435
5688873341005252245836 73
e= 591669389367557160624182203458313572204662453839829857223716341691504683070654605280780517
7713936976
n= 498863120827187581423038137504447251078367733709620146540575038919376162489147584399024092
161282944839132325288829539916205008765998813494019604455692656679705717828558974841445347308
795023507311707149630173563577255407956884333467978713101874452590300991413156725965640205518
84308468255890746864 3649
m^e mod n= 351497298545323868324249020405352376721101804414973518026637665730429852267204328 9
933470561527137504731365548232273371804350053175734509901640307720868444431544369052379701563
640059722712055858945740932666281360901308762873846127462246474550567406239125716779226880613
707375927456809747637490426522 45
```

En revanche, le calcul de $m^e \pmod n$ par la fonction puissance avec des nombres de cette taille ne se termine jamais ! En effet, la fonction puissance effectue $e-1$ passages dans la boucle, donc $e-1$ multiplications. Comme e a 100 chiffres, même à raison de 10^9 multiplications par secondes, il faut environ $3 \cdot 10^{83}$ années pour effectuer ce calcul !

Voici un algorithme plus efficace (c'est un euphémisme !) pour calculer $m^e \pmod n$: on commence par écrire e en base 2

$$e = e_0 2^0 + e_1 2^1 + e_2 2^2 + e_3 2^3 + \cdots e_l 2^l$$

avec l la partie entière du logarithme en base 2 de e et, pour tout i entre 0 et l, $e_i = 0$ ou $e_i = 1$. Ainsi,

$$m^e = m^{e_0+e_1 2+e_2 2^2+e_3 2^3+\cdots e_l 2^l} = m^{e_0 2^0} m^{e_1 2^1} m^{e_2 2^2} m^{e_3 2^3} \cdots m^{e_l 2^l} = \prod_{i\in\omega} m^{2^i}$$

où $\omega = \{0 \leq i \leq l \;\Big|\; e_i = 1\}$. Il suffit donc de commencer par calculer

$$m^2,\ \left(m^2\right)^2,\ \left(\left(m^2\right)^2\right)^2, \cdots, m^{2^l} \pmod n$$

ce qui nécessite $l-1$ mises au carré, et ensuite d'effectuer le produit ci-dessus modulo n, qui nécessite au pire l multiplications. Remarquons que si e est, en base 10, un nombre de 100 chiffres,

$$l \approx \log_2(10^{100}) = 100 \log_2(10) = 100 \frac{1}{\log_{10}(2)} \approx 332$$

Par conséquent, avec cet algorithme, le calcul de $m^e \pmod n$ s'effectue, sur un ordinateur familial, "instantanément".

Voici un script *Python* qui implémente cet algorithme. La fonction est appelée powmaison. Elle fait appel à la fonction tobin qui transforme e en base 2.

```
1  def tobin(e):
2      r=[]
3      while e>0:
4          r.append(e%2)
5          e=e/2
6      return r
7  def powmaison(m,e,n):
8      eb=tobin(e)
9      r=1
10     mp=m%n
11     for e in eb:
12         if e==1:
13             r=r*mp%n
14         mp=mp**2%n
15     return r
16 import random
17 m=random.randint(1,10**300)
18 e=random.randint(1,10**100)
19 n=random.randint(1,10**300)
20 print "m=",m
21 print "e=",e
22 print "n=",n
23 powmen=pow(m,e,n)
24 print "pow(m,e,n)=",powmen
25 powmaison=powmaison(m,e,n)
26 print "powmaison(m,e,n)=",powmaison
27 print "diff=",(powmen-powmaison)
```

L'exécution est "instantanée" et donne

```
python powmaison.py
m= 29883981129719440692697109961273846952244480996172808673766872848039723747653143673056297613797905812173206
31827538330023158611750354813125301219386043639853122417578165434267825872471270058862421244569509552578359124
97794042271699063314327565795397850712742819715400884943482920971652250939658062583
e= 27913040201020624061280319732429813720133860276522509620663492422644545011755498158019868831357529777
n= 30287959833352070151553742432880344188367988802770300387249095975901383934185898315854019386042658302884721 6
27251799845470473256690293399012546130272946517865156835464182105890951082495225493254501123501764997062587307 8
10472038501284003313614535295326146998150098950646831057168584243056144299541783 0
pow(m,e,n)= 22862217038033267671865176272714905804588081346729779623541241675556166161618165840956659527285629 7
62560840471557115486704762203249769366604849306190639490680046065326732377053296923517125113630698104632730972 5
79820827136765785985196306600224324162836128792646255950325533807544577426766071304301002 3
powmaison(m,e,n)= 22862217038033267671865176272714905804588081346729779623541241675556166161618165840956659527 2
85629762560840471557115486704762203249769366604849306190639490680046065326732377053296923517125113630698104632 7
30972579820827136765785985196306600224324162836128792646255950325533807544577426766071304301002 3
diff= 0
```

2. Programme d'encryptage et de décryptage.

Le programme *Python* donné ci-dessous permet de crypter et décrypter un message avec le protocole RSA et d'y insérer une signature électronique. La fonction nombresin vérifie que les nombres donnés par l'utilisateur (n_j, e_j, n_r, $\cdots$) sont bien des nombres entiers. Chaque lettre du message est transformée en un nombre compris entre 0 et 255 (8 bits, soit un octet) conformément au code ASCII. Par conséquent, les lettres accentuées sont remplacées par des lettres sans accents (fonction accents). Le texte est décomposé en blocs de k lettres selon le procédé:

$$\text{exemple} \rightarrow \text{ord('e')}256^0 + \text{ord('x')}256^1 + \text{ord('e')}256^2 + \cdots$$

où ord est la fonction qui à chaque lettre attribue son code ASCII. Sur les k lettres, b sont extraites du texte et r (fonction tailleblocs) sont tirées au hasard afin qu'un même texte ne soit pas encodé deux fois de suite de la même manière (fonction strtobigint dont la réciproque est la fonction biginttostr). Les

fonctions tobin et powmaison ont déjà été commentées plus haut. Chaque bloc de lettres, après avoir été transformé en un nombre, est ensuite encrypté selon le protocole RSA. Le message est constitué de la concaténation de ces nombres dont la taille est “calibrée” (fonctions concatenationblocsdenombres et sa réciproque coupurenombres)

```
1  import unicodedata
2  import random
3  import sys
4  from math import *
5  def nombresin(texte):
6      m=''
7      n=raw_input(texte)
8      while m=='':
9          try:
10             m=int(n)
11         except:
12             print "\nVous devez entrer un nombre entier !\n"
13             n=raw_input(texte)
14     return m
15 def accents(texte):
16     if isinstance(texte,str):
17         texte=unicode(texte,"utf8","replace")
18         texte=unicodedata.normalize('NFD',texte)
19         texte=texte.encode('ascii','ignore')
20         return texte
21 def tailleblocs(n):
22     k=int(log(n)/log(256))
23     r=min(10,k//2)
24     b=k-r
25     print "\nTaille des blocs: "+str(k)+" caracteres"
26     print "Nombre de caracteres aleatoires ajoutes en debut de chaque bloc: "+str(r)
27     print "Nombre de caracteres du texte par bloc: "+str(b)
28     return k,r,b
29 def strtobigint(texte,n):
30     if n<=256:
31         print "n doit etre >256 (et en fait beaucoup plus grand !)"
32     else:
33         k,r,b=tailleblocs(n)
34         l=len(texte)
35         print "\nTaille du texte: "+str(l)
36         espaces=k-l%b
37         print "Nombres d'espaces ajoutes a la fin du texte: "+str(espaces)
38         texte=texte+espaces*" "
39         l=len(texte)
40         m=[]
41         for j in xrange(0,l//b,1):
42             mb=0
43             for i in xrange(0,r,1):
44                 mb=mb+random.randint(0,255)*256**i
45             for i in xrange(r,k,1):
46                 mb=mb+ord(texte[j*b+i-r])*256**i
47             m.append(mb)
48         return m
49 def biginttostr(m,n):
50     if n<=256:
51         print "\n n doit etre >256 (et en fait beaucoup plus grand !)\n "
52         sys.exit()
53     else:
54         k,r,b=tailleblocs(n)
55         texte=''
56         for mb in m:
57             textebloc=''
58             for i in xrange(0,k,1):
59                 textebloc=textebloc+chr(mb%256)
```

```
60                  mb=mb//256
61              texte=texte+textebloc[r::]
62          return texte
63  def tobin(e):
64      r=[]
65      while e>0:
66          r.append(e%2)
67          e=e/2
68      return r
69  def powmaison(m,e,n):
70      eb=tobin(e)
71      r=1
72      mp=m%n
73      for e in eb:
74          if e==1:
75              r=r*mp%n
76          mp=mp**2%n
77      return r
78  def coupurenombres(m,n):
79      l=int(log(n)/log(10))+1
80      lm=len(m)
81      mb=[]
82      for i in xrange(0,lm//l,1):
83          mb.append(int(m[i*l:(i+1)*l]))
84      return mb
85  def concatenationblocsdenombres(m,n,e):
86      mf=''
87      for mb in m:
88          mbf=str(powmaison(mb,e,n))
89          mbf=(int(log(n)/log(10))+1-len(mbf))*'0'+mbf
90          mf=mf+mbf
91      return mf
92  choix=raw_input("\nPour encoder un texte, appuyez sur la touche c et pour decoder sur une autre
        touche: ")
93  if choix=='c':
94      nj=0
95      nj=nombresin("\nCle publique du destinataire: n=")
96      ej=nombresin("Cle publique du destinataire: e=")
97      texte=raw_input("\nMessage (Attention: les accents seront supprimes !):\n")
98      signature=raw_input("\nSignature (sans accents, appuyez seulement sur la touche enter pour un
            message sans signature): ")
99      if signature!='':
100         nr=nombresin("\nVotre cle publique: n=")
101         dr=nombresin("Votre cle privee: d=")
102         ms=strtobigint(signature,nr)
103         texte=texte+'\n\n'+'***'+concatenationblocsdenombres(ms,nr,dr)
104     texte=accents(texte)
105     m=strtobigint(texte,nj)
106     mf=concatenationblocsdenombres(m,nj,ej)
107     print "\nMessage encrypte:\n\n"+mf
108 else:
109     nj=nombresin("\nVotre cle publique: n=")
110     dj=nombresin("Votre cle privee: d=")
111     M=raw_input("\nMessage recu:\n")
112     MB=coupurenombres(M,nj)
113     m=[]
114     for b in MB:
115         m.append(powmaison(b,dj,nj))
116     texte=biginttostr(m,nj)
117     print "Texte decode:\n"
118     print texte
119     if texte.find('***'):
120         tp=texte.split('***')
121         try:
122             ntp=int(tp[1])
```

```
123             except:
124                 ntp=''
125             if ntp!='':
126                 print "\nIl semblerait que le texte contienne une signature electronique."
127                 nr=nombresin("\nCle publique de l'expediteur: n=")
128                 er=nombresin("Cle publique de l'expediteur: e=")
129                 SB=coupurenombres(tp[1],nr)
130                 s=[]
131                 for b in SB:
132                     s.append(powmaison(b,er,nr))
133                 print "\nLa signature est: "+biginttostr(s,nr)
```

Pour tester ce programme, nous devons disposer de clés RSA. Le programme *Python* donné ci-dessous, crée des clés RSA. On commence par tirer p au hasard, et on trouve le premier nombre premier plus grand que p grâce à la méthode ntheory.generate.nextprime du module sympy (`apt-get install python-sym`
La construction d'une telle fonction nécessite l'étude des tests de primalité. Le nombre d est trouvé grâce à la fonction pgcd2 qui utilise l'algorithme d'Euclide étendu.

```
1  import random
2  import sympy
3  def pgcd2(a,b):
4      s1=1
5      t1=0
6      s2=0
7      t2=1
8      a1=max(a,b)
9      b1=min(a,b)
10     r1=a1%b1
11     while r1>0:
12         q1=a1//b1
13         s=s1-q1*s2
14         t=t1-q1*t2
15         t1=t2
16         s1=s2
17         t2=t
18         s2=s
19         a1=b1
20         b1=r1
21         r1=a1%b1
22     if a>=b:
23         return [b1,s2,t2]
24     else:
25         return [b1,t2,s2]
26 p=sympy.ntheory.generate.nextprime(random.randint(1,10**300))
27 q=sympy.ntheory.generate.nextprime(random.randint(1,10**300))
28 n=p*q
29 phin=(p-1)*(q-1)
30 e=random.randint(1,10**20)
31 while pgcd2(e,phin)[0]!=1:
32     e=e+1
33 d=pgcd2(e,phin)
34 d=d[1]%phin
35 print "Cle RSA privee: \n\np="+str(p)+"\n\nq="+str(q)+"\n\nd="+str(d)+"\n\nCle RSA publique\n\nn=
       "+str(n)+"\n\ne="+str(e)
```

On trouve, par exemple (fichier `cle.txt`)

```
Romeo

Cle RSA privee:

p=897153650979531378224611145474583668810250013024351232414128927620501175783244252260934098835129669908222599917534915288338408750727775272006178012780810816889936168703056207550250528352427695526999048717097204693285349951403865814213885460583987005455859942244195203273382083221224845549058023450129
```

```
q=948416782129933377722371651615302728181042158451716714032112676469410341879317903623578847487257528296
977426119540137914495344297578861530479745899857052098815850222263341524371446197808489056194061381635655
841236189495230898904142670749939917018772193536546640774136302438485263529783306649002955271 3

d=495933401006307325217792602691137434326768336189463909017877380480609950005602852567286740887431311723 6
090797243317443785519480138416668733951450309293644861998518722130151350411849790170481809512026895891063
411575785616687894024717718813623909317286694614453708086728907439327984550346905218166239031955730252286
313195103727493951371098552113248201223654803999552848027137824695038015532651491882605375899566722017631
224058862680317933126370638747533647184731128193657151584478415879090903957456819639593924340020613344508
598017294152033434460357213619143515937723325897315488801040740843218722530 47

Cle RSA publique

n=850875578738128501811538810253309434545827974334430331561188492299332764701339337449232788918995759284 56
568571835080313730738524889238797849520194669663236150344264016589437518028588746782184280932040754925271 1
207004404912753967617125249480397595954683913905232084685611260879195052426600537404947354371653316425243 8
899793597185492070516091907634800576948163408747541030107535346593440996890455180277000678019592510040095 1
293170214367985247142845414117683226521155861244422258251149579241212639216734006885333107051689224212517 1
769157514701547318770217266585936638447667294568374230295264578932149977

e=97233094499729231815

Juliette

Cle RSA privee:

p=114211239291959423234009306750775681216183633261855778315470489097410975173661931233197850869828226076 91
083294340425193745580531677873812440995680208741210469389182338505293426023681443710547436145400961288562 8
229147205324951258613171277733889588768909411401226568206687728090835820915159870228152569

q=420257011630823253878710433418597945176035162322458155304860183223580043329958580335694291900940149683 12
674777290679759166019330423397665917344141325897595812230251277701214768841999584179892242609138677642400 4
368049681085303633563479672051840482317938315470605351343430558301013454848682993754087627

d=839520001622114163582453510431414083609187238851298627220227973052367165676401936189249677751211257275 69
530616936665482568107501294472664108312154819043106654274768911162370121579548643434292447039665310996376 0
991928870095685445854311188539374420375641047934416953602251987726326261014136146448009728828022702433320 8
746072889280960519621547653399759602979265902099273853656132627602541168673597961894893030994470066052091 3
172626097203841614053217343456865199391621570497746920061923024625211487319191784046007861759847190704995 4
843255708737006512500918099637099067837530778158967976611470036773269

Cle RSA publique

n=479980741194917291412825724892575091282105959377863052352257135157282003977882528868951259007122681329429
678435915255860711651185615742187593645852221033218214264526766235945087676492514408737158271707392790692 50
797031980764044880551473492625630637906664306563758375703887626466745173309103164188396623878138898712553 79
961444924491127792881686747371247178898809608829704016937758010045030691440930232059852205058678862631493 98
069625613415098317292222754118605902744683333158042261452354474748121154409274233186591198111046700926322 36
262638671420094220237308891971352300977212893315985534536351163763 3

e=64546481480700602045
```

L'encryptage donne (fichier texte.txt)

```
Pour encoder un texte, appuyez sur la touche c et pour décoder sur une autre touche: c

Cle publique du destinataire: n=479980741194917291412825724892575091282105959377863052352257135157282003977882528868951259007122681329429
678435915255860711651185615742187593645852221033218214264526766235945087676492514408737158271707392790692 50
797031980764044880551473492625630637906664306563758375703887626466745173309103164188396623878138898712553 79
961444924491127792881686747371247178898809608829704016937758010045030691440930232059852205058678862631493 98
069625613415098317292222754118605902744683333158042261452354474748121154409274233186591198111046700926322 36
262638671420094220237308891971352300977212893315985534536351163763 3
Cle publique du destinataire: e=64546481480700602045

Message (Attention: les accents seront supprimes !):
```

```
Elle parle ! Oh ! parle encore, ange resplendissant ! Car tu rayonnes dans cette nuit, au-dessus de ma tete, comme
 le messager aile du ciel, quand, aux yeux bouleverses des mortels qui se rejettent en amere pour le contempler, il
 devance les nuees paresseuses et vogue sur le sein des airs !

Signature (sans accents, appuyez seulement sur la touche enter pour un message sans signature): Ton Romeo

Votre cle publique: n=850875578738128501811538810253309434545827974334430331561188492299332764701339337449232788918995759284565685718350803137307385248892387978495201946696632361503442640165894375180285887467821842809320407549252711207004404912753967617125249480397595954683913905232084685611260879195052426600537404947354371653316425243889979359718549207051609190763480057694816340874754103010753534659344099689045518027700067801959251004009512931702143679852471428454141176832265211558612444222582511495792412126392167340068853331070516892242125171769157514701547318770217266585936638447667294568374230295264578932149977
Votre cle privee: d=495933401006307325217792602691137434326768336189463909017877380480609950005602852567286740887431311723609079724331744378551948013841666873395145030929364486199851872213015135041184979017048180951202689589106341157578561668789402471771881362390931728669461445370808672890743932798455034690521816623903195573025228631319510372749395137109855211324820122365480399955284802713782469503801553265149188260537589956672201763122405886268031793312637063874753364718473112819365715158447841587909090395745681963959392434002061334450859801729415203343446035721361914351593772332589731548880104074084321872253047

Taille des blocs: 249 caracteres
Nombre de caracteres aleatoires ajoutes en début de chaque bloc: 10
Nombre de caracteres du texte par bloc: 239

Taille du texte: 9
Nombres d'espaces ajoutes a la fin du texte: 240

Taille des blocs: 248 caracteres
Nombre de caracteres aleatoires ajoutes en debut de chaque bloc: 10
Nombre de caracteres du texte par bloc: 238

Taille du texte: 896
Nombres d'espaces ajoutes a la fin du texte: 66

Message encrypte:

419163066681418288820898803590041739232152086190946984612108003259048968584764411257007561947473542401155541018627297151315308495797089617992067799280041148620335612992387522439712901492783865893259194445191168025626224081219108069194726903006475151837901490509659143584125447264725717816462629174782601081468993515152672948802245378594335658317322891443968967132628179880687344130313832351993416399904925325773017215913445864819211208518861253088238042198749115839718266498769813500773919844431035666755270213660697762458980495318731437158755686388167783429921358886366195650509495669197273037068791601967726441180500465305453833468636953074063686191249288176453745090307353065366481397167228771793886198524124793302297572689267993204305319898325416220877828727806981323689720085140314921344825023591005782785221053452663270169794698977752971685393919741976991278125013600984842686521597086709356710714576014644762007338875099060405695150527028110927365206547395664747539661312762982432844918845087549200125239177437144022537373872378764438053160838992205766630346521759337160692475441972013931154217046804974356139880198583061569545050102262318433669865027151110180038358881710147569191140414869300263174423915916571928021314935896409434598305884872880232273478108292340838248127220574840702771491833075982193827020961886123256947071394825053423123548562076096752590831424121554715084006977323316270805805642477426669015419859239261663154984429754211102080754788382447786698616794598868058795479741584336908582169026456015757126572080999967417619990737491246604268410339686203050324545201500715445764765278426496658673488190558115683923273771702989492641116575274264099487194373446649460289228171816436008761945915342751578724063651916476101038065360496059343434265871069574496832535955194685579179962452445587979446010499286882619353874655167054425808577418947409874186079435447584668060070340375998807526258096635933584275306196351055401302654318969291989124118360284127286024332269071822695792820371221313114587442778000819225541936044239345578037184533003281319500012119210854945999589578938130646081024006876731229993505541523508743918698519267170794425152834190905155644737033099101295931599909818748719668920321290013734732261781295116611747412255239651111086855703523175552567485637596857007018339697213833308909076046109636356642751061815236567443572264855783286828716394215491596930 3
```

Le décryptage donne:

```
Pour encoder un texte, appuyez sur la touche c et pour decoder sur une autre touche:

Votre cle publique: n=47998074119491729141282572489257509128210595937786305235225713515728200397788252886895125900712268132942967843591525586071165118561574218759364585222103321821426452676623594508767649251440873715827170739279069250797031980764044880551473492625630637906664306563758375703887626466745173309103164188396623878138898712553799614449244911277928816867473712471788988096088297040169377580100450306914409302320 5
```

```
985220505867886263149398069625613415098317292222754118605902744683333158042261452354474748121154409274233186591198111046700926362322626386714200942202373088919713523009772128933159855345363511637631
Votre cle privee: d=83952000162211416358245351043141408360918723885129862722022797305236716567640193618924967777512112572756953061693666548256810750129447266410831215481904310665427476891116237012157954864343429244703966531099637609919288700956854458543111885393744203756410479344169536022519877263262610141361464480097288280227024333208746072889280960519621547653399759602979265902099273853656132627602541168673597961894893030994470066052091317262609720384161405321734345686519939162157049774692006192302462521148731919178404600786175984718907049954843255708737006512500918099637099067837530778158967976611470036773269

Message recu:
419163066681418288820898803590041739232152086190946984612108003259048968584764411257007561947473542401155541018627297151315308495797089617992067799280041148620335612992387522439712901492783865893259194445191168025626224081219108069194726903006475151837901490509659143584125447264725717816462629174782601081468993515152672948802245378594335658317322891443968967132628179880687344130313832351993416399904925325773017215913445864819211208518861253088238042198749115839718266498769813500773919844431035666755270213660697762458980495318731437158755686388167783429921358886366195650509495669197273037068791601967726441180500465305453833468636953074063686191249288176453745090307353065366481397167228771793886198524124793302297572689267993204305319898325416220877828727806981323689720085140314921344825023591005782785221053452663270169794698977752971685393919741976991278125013600984842686521597086709356710714576014644762007338875099060405695150527028110927365206547395664747539661312762982432844918845087549200125239177437144022537373872378764438053160838992205766630346521759337160692475441972013931154217046804974356139880198583061569545050102262318433669865027151110180038358881710147569191140414869300263174423915916571928021314935896409434598305884872880232273478108292340838248127220574840702771491833075982193827020961886123256947071394825053423123548562076096752590831424121554715084006977323316270805805642477426669015419859239261663154984429754211102080754788382447786698616794598868058795479741584336908582169026456015757126572080999967417619990737491246604268410339686203050324545201500715445764765278426496658673488190558115683923273771702989492641116575274264099487194373446649460289228171816436008761945915342751578724063651916476101038065360496059343434265871069574496832535955194685579179962452445587979446010499286882619353874655167054425805774189474098741860794354475846680600703403759988075262580966355933584275306196351055401302654318969291989124118360284127286024332269071822695792820371221313114587442778000819225541936044239345578037184533003281319500012119210854945999589578938130646081024006876731229993505541523508743918698519267170794425152834190905155644737033099101295931599909818748719668920321290013734732261781295116611747412255239651111086855703523175552567485637596857007018339697213833308909076046109636356642751061815236567443572264855783286828716394215491596930
Taille des blocs: 248 caracteres
Nombre de caracteres aleatoires ajoutes en debut de chaque bloc: 10
Nombre de caracteres du texte par bloc: 238
Texte decode:

Elle parle ! Oh ! parle encore, ange resplendissant ! Car tu rayonnes dans cette nuit, au-dessus de ma tete, comme le messager aile du ciel, quand, aux yeux bouleverses des mortels qui se rejettent en amere pour le contempler, il devance les nuees paresseuses et vogue sur le sein des airs !

***266122097241625380507018817069583594201392724352492889048049063130803070090229525479372745546709996339218372609180290980705015004244486279703666746832409205046023970909184276165655082553946793707502725377762888234790182279160249779861522769432541403624482882555455292769415961235854663321816494118195408043172253393772188975847506236304334321807336973731917717608071556766536247607324303093820571858708017082172302211236810008515657731593204991838054287984795973684782939713594778672807891792556313910809840752208089897570212881531779709371586952442392357774493474038006867368942362762327120378325321

Il semblerait que le texte contienne une signature electronique.

Cle publique de l'expediteur: n=85087557873812850181153881025330943454582797433443033156118849229933276470133933744923278918995759284565685718350803137307385248892387978495201946696632361503442640165894375180285887467821842809320407549252711207004404912753967617125249480397595954683913905232084685611260879195052426600537404947354371653316416425243889979359718549207051609190763480057694816340874754103010753534659344099689045518027700067801959251004009512931702143679852471428454141176832265211558612444222582511495792412126392167340068853331070516892242125171769157514701547318770217266585936638447667294568374230295264578932149977
Cle publique de l'expediteur: e=97233094499729231815

Taille des blocs: 249 caracteres
Nombre de caracteres aleatoires ajoutes en debut de chaque bloc: 10
Nombre de caracteres du texte par bloc: 239

La signature est: Ton Romeo
```

CHAPITRE 3

Tests de primalité

Dans ce chapitre, nous allons étudier quelques tests de primalité. Le problème est simple: étant donné un nombre $p \in \mathbb{N}$, trouver un algorithme qui détermine si p est premier. Bien entendu, il est souhaitable que cet algorithme nécessite un temps de calcul "raisonnable".

L'histoire des tests de primalité s'étend sur plus de 2000 ans. Nous verrons que certains résultats fondamentaux sont très récents. Au lieu de procéder à une étude chronologique, nous séparons les tests de primalité en deux grandes catégories: les tests déterministes et les tests probabilistes. Nous aborderons seulement brièvement les tests déterministes et nous concentrerons notre attention sur les tests probabilistes. A la fin du chapitre, nous aborderons une application concrète: la production de grands nombres premiers et de clefs RSA. Nous commençons par quelques rappels (sans démonstrations) concernant la répartition des nombres premiers.

1. Répartition des nombres premiers

1. Le crible d'Ératosthène.

Le crible d'Érathosthène (276-194 av. J.-C.) est une méthode simple pour établir la liste des nombres premiers inférieurs à un nombre entier donné $n \geq 2$. On commence par écrire la liste des nombres

$$1,\ 2,\ 3,\ \cdots,\ n-1,\ n$$

et on biffe le 1. Le premier nombre non biffé, 2, est un nombre premier

$$\not{1},\ \underline{\mathbf{2}},\ 3,\ \cdots,\ n-1,\ n$$

Ensuite, on biffe tous les multiples de 2 et le premier nombre non biffé, 3, est un nombre premier:

$$\not{1},\ \underline{\mathbf{2}},\ \underline{\mathbf{3}},\ \not{4},\ 5\ \ \not{6},\ 7,\ \cdots,\ n-1,\ n$$

et on recommence le procédé en biffant tous les multiples de 3.

Par exemple, pour $n = 10$, on trouve

$$\not{1},\ \underline{\mathbf{2}},\ \underline{\mathbf{3}},\ \not{4},\ \underline{\mathbf{5}}\ \ \not{6},\ \underline{\mathbf{7}},\ \not{8},\ \not{9},\ \not{1}0$$

Nous verrons plus bas que cet algorithme est totalement inefficace pour trouver des grands nombres premiers (*i.e.* 300 chiffres en base 10).

2. Le théorème des nombres premiers.

Définition 1.1. Soit $x \in \mathbb{R}_+$. On définit

$$\pi(x) = \#\left\{p \text{ premier} \;\middle|\; p \leq x\right\}$$

le nombre de nombres premiers $\leq$ à x.

En 1896, Hadamard et La Vallée Poussin démontrent indépendamment le théorème suivant, appelé le théorème des nombres premiers.

THÉORÈME 1.2 (des nombres premiers).

$$\lim_{x\to+\infty} \pi(x)\frac{\ln(x)}{x} = 1$$

Pour information, quelques valeurs numériques calculées avec Mathematica :

x	$\pi(x)$	$\frac{x}{\ln(x)}$	$\frac{\pi(x)}{x/\ln(x)}$
10	4	$4,343$	$0,921$
10^2	25	$21,715$	$1,151$
10^3	168	$144,765$	$1,161$
10^4	1229	$1085,736$	$1,132$
10^5	9592	$8685,890$	$1,104$
10^6	78498	$72382,414$	$1,084$

x	$\pi(x)$	$\frac{x}{\ln(x)}$	$\frac{\pi(x)}{x/\ln(x)}$
10^7	664579	$620420,688$	$1,071$
10^8	5761455	$5428681,024$	$1,061$
10^9	50847534	$48254942,434$	$1,054$
10^{10}	455052511	$434294481,903$	$1,048$
10^{11}	4118054813	$3948131653,666$	$1,043$
10^{12}	37607912018	$36191206825,27$	$1,039$

La démonstration originale du théorème des nombres premiers repose sur des méthodes d'analyse complexe que nous n'aborderons pas ici. Des preuves dites 'élémentaires", c'est-à-dire ne faisant pas appel à l'analyse complexe, existent. La première date de 1949 et est due à Erdös et Selberg. Notons que le théorème des nombres premiers a été conjecturé par Gauss en 1792 et par Legendre en 1797.

Il suit du théorème des nombres premiers que si n est un nombre de 150 chiffres en base 10, il existe environ

$$\frac{10^{150}}{\ln(10^{150})} = \frac{10^{150}}{150\ln(10)} \approx \frac{10^{150}}{345} \approx 3\cdot 10^{147}$$

nombres premiers inférieurs à n. Il est donc exclu d'utiliser le crible d'Ératosthène pour trouver des "grands" nombres premiers.

2. Les tests de primalité déterministes

Le test de primalité le plus naïf consiste à tester tous les nombres entre 2 et $n-1$. En fait, il suffit de tester tous les nombres en 2 et la partie entière de $\sqrt{n}$, car si n est composé, disons $n = a\cdot b$, alors a ou $b \leq \sqrt{n}$ (car si a, $b > \sqrt{n}$, alors $n = a\cdot b > \sqrt{n}\sqrt{n} = n$, ce qui est impossible). Ce test est inutilisable pour des "grands" nombres premiers (disons 150 chiffres en base 10) car il faut effectuer environ $\sqrt{10^{150}} = 10^{75}$ tests ! Et même en testant uniquement les nombres premiers inférieurs à $\sqrt{n}$, à supposer qu'on les connaisse tous, l'algorithme est inapplicable, car en vertu du théorème des nombres premiers, il y a environ

$$\frac{\sqrt{10^{150}}}{\ln(\sqrt{10^{150}})} = \frac{10^{75}}{75\ln(10)} \approx 6\cdot 10^{72}$$

nombres premiers inférieurs à $\sqrt{n}$ à tester, soit, à raison de 10^{12} test à la seconde, $6\cdot 10^{60}$ secondes, soit environ $2\cdot 10^{54}$ années, beaucoup plus que l'âge estimé de l'univers !

Notons L le nombres de chiffres en base deux d'un nombre n. En 2002, **M. Agrawal, N. Kayal et N. Saxena** ont publié un algorithme déterministe dont le temps d'exécution est polynomial en L.

Nous verrons que certains algorithmes probabilistes sont en fait déterministes si la conjecture de Riemann est vraie.

3. Tests de primalité probabilistes

1. Le test de Fermat.

En vertu du petit théorème de Fermat, nous savons que si p est premier et si $p \nmid a$, alors $a^{p-1} \equiv 1 \pmod{p}$. Réciproquement, si n n'est pas premier, alors il existe un diviseur a de n avec $2 \leq a < n$. Dans ce cas, a^{n-1} n'est pas congru à 1 modulo n. En effet, si $a^{n-1} \equiv 1 \pmod{n}$, alors n divise $a^{n-1} - 1$ et, comme a divise n, a divise également $a^{n-1} - 1$, ce qui signifie qu'il existe un nombre k tel

$$ak = a^{n-1} - 1 \;\Rightarrow\; a^{n-1} - ak = 1 \;\Rightarrow\; a(a^{n-2} - k) = 1$$

et il suit que a divise 1, ce qui est impossible car par hypothèse, $a \geq 2$.

Définition 3.1. Soit n et a des nombres entiers.

(1) n est un **pseudopremier de Fermat en base** a si a est relativement premier à n et $a^{n-1} \equiv 1 \pmod{n}$.
(2) n est un **nombre de Carmichael** si $a^{n-1} \equiv 1 \pmod{n}$ pour tout a relativement premier à n.

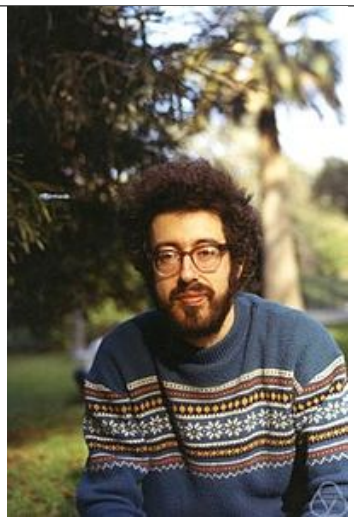
Exemple 3.2. Le nombre $341 = 11 \cdot 31$ est un nombre pseudopremier de Fermat en base 2 (qui sont appelés des nombres de Poulet) comme on peut facilement le vérifier avec Python:

```
>>> pow(2,340,341)
1
```

Par ailleurs, le nombre $561 = 3 \cdot 11 \cdot 17$ est un nombre de Carmichael. Remarquons que s'il existe a tel que $a^{n-1} \equiv 1 \pmod{n}$, alors $a = a_0 + kn$ avec $a_0 < n$ et $a_0^{n-1} \equiv 1 \pmod{n}$. Pour vérifier que 561 est un nombre de Caramichael, il suffit donc de tester tous les a premiers à 561 et plus petits que 561. Le programme suivant teste si 561 est un nombre de Caramichael:

```
1 def pgcd2(a,b):
2     s1=1
```

Pierre de Fermat (∼ 1605-1665)



R. M. Solovay (1938, Brooklyn)

V. Strassen (1936, Düsseldorf)

Tableau 1. Les tests de primalité probabilistes.

```
3       t1=0
4       s2=0
5       t2=1
6       a1=max(a,b)
7       b1=min(a,b)
8       r1=a1%b1
9       while r1>0:
10          q1=a1//b1
11          s=s1-q1*s2
12          t=t1-q1*t2
13          t1=t2
14          s1=s2
15          t2=t
16          s2=s
17          a1=b1
18          b1=r1
19          r1=a1%b1
20      if a>=b:
21          return [b1,s2,t2]
22      else:
23          return [b1,t2,s2]
24  n=4
25  for a in xrange(3,561,1):
26      if  pgcd2(a,561)[0]==1 and pow(a,560,561)!=1:
27          print 'a=',a,'a^560 mod 561=',pow(a,560,561)
```

Test de Fermat Il existe une infinité de nombres pseudopremiers et de nombres de Carmichael mais ils sont rares. C'est cette rareté qui permet de tester si un nombre n est premier en calculant $a^{n-1} \pmod n$ pour quelques valeurs de nombres a. En pratique, on peut prendre les nombres 2, 3, 5 et 7. Si $a^{n-1} \equiv 1 \pmod n$ pour toutes ces valeurs de a, alors n est "probablement premier", sinon il est certainement composé. En utilisant des nombres supplémentaires, on diminue le risque que le test se fasse duper par des "faux premiers".

Le programme *Python* qui suit, fabrique au hasard 1000 grands nombres composés. Il compte et affiche le nombre d'erreurs du test de Fermat.

```
1   import random
2   def testfermat(n):
3       r=True
4       temoins=[2,3,5,7]
5       j=0
6       while r==True and j<=3:
7           a=temoins[j]
8           if pow(a,n-1,n)!=1:
9               r=False
10          j=j+1
11      return r
12  erreurs=0
13  for j in xrange(1,1001,1):
14      n1=random.randint(1,10**150)
15      n2=random.randint(1,10**150)
16      if testfermat(n1*n2):
17          erreurs=erreurs+1
18  print "Nombre d'erreurs:", erreurs
```

On note

$$G_0 := \mathbb{Z}_n^* \text{ et } G_1 := \left\{ a \in G_0 \;\middle|\; a^{n-1} \equiv 1 \pmod n \right\} \subseteq G_0$$

Remarquons que G_1 est un sous-groupe de G_0 et

$$n \text{ est premier } \Rightarrow \; G_1 = G_0$$

Ces définitions seront utilisées pour comparer les différents tests de primalité probabilistes.

REMARQUE 3.3. Commençons par rappeler que si $n = p$ est premier, $\mathbb{Z}_p^* = \{1, 2, \cdots, p-1\}$ alors que si n est composé, $a \in \mathbb{Z}_n^*$ si et seulement si a est relativement premier à n (car il existe $b \in \mathbb{Z}_n^*$ tel que $ab \equiv 1 \pmod{n}$ et $ab \equiv 1 \pmod{n}$ si et seulement si il existe k tel que $ab - kn = 1$, c'est-à-dire si et seulement si pbcd(a, n) divise 1, en vertu du théorème de Bézout).

Par conséquent, si n est composé, l'ensemble des nombres $a \in [2, n-1]$ qui ne passent pas le test de Fermat, mis à part les diviseurs de n, est donné par $G_0 - G_1$. Ces nombres sont appelés **des témoins** (témoins que n est composé). Par ailleurs, les nombres dans G_1 passent le test. Ils sont appelés **des menteurs**. Le défaut du test de Fermat est qu'il existe des nombres composés, les nombres de Carmichael, pour lesquels $G_0 = G_1$, ce qui veut dire que tous les nombres dans G_0 sont des menteurs et qu'il n'y pas de témoins.

Nous allons maintenant présenter des autres tests probabilistes de primalité pour lesquels l'ensemble $G \subseteq G_0$ des nombres a qui passent le test (les menteurs) est toujours strictement inclus dans G_0 si n est composé.

2. Le test de Solovay-Strassen.

Nous commençons par voir (ou revoir) quelques notions concernant le symbole de Legendre, nécessaires à la compréhension du test de Solovay-Strassen.

DÉFINITION 3.4. On définit le **symbole de Legendre** pour $a \in \mathbb{Z}$ et p premier $\neq 2$ par:

$$\left(\frac{a}{p}\right) := \begin{cases} 0 & \text{si } a \equiv 0 \pmod{p} \\ +1 & \text{si } a \text{ est un carré non nul modulo } p \\ -1 & \text{si } a \text{ n'est pas un carré modulo } p \end{cases}$$

Si $n = p_1^{n_1} \cdots p_k^{n_k}$ avec $p_1, \cdots, p_k$ premiers, et n est impair, on définit (Jacobi):

$$\left(\frac{a}{n}\right) := \left(\frac{a}{p_1}\right)^{n_1} \cdots \left(\frac{a}{p_k}\right)^{n_k}$$

LEMME 3.5. *Soit G un groupe cyclique d'ordre r et d'élément neutre e. Soit m un entier et $z \in G$. Soit $k = \text{pgcd}(m, r)$ Notons*

$$S = \left\{ x \in G \;\middle|\; x^m = z \right\}$$

Alors,

$$\left[S \neq \emptyset \;\Rightarrow\; z^{\frac{r}{k}} = e\right] \;\; et \;\; \left[z^{\frac{r}{k}} = e \;\Rightarrow\; \#S = \text{pgcd}(m, r)\right]$$

où $\#S$ désigne le nombre d'éléments de S.

DÉMONSTRATION. Soit g un générateur de G. Alors, il existe un nombre i tel que $z = g^i$. Soit $x = g^j \in G$. Alors, $x^m = z$ si et seulement si $g^{mj} = g^i$, c'est-à-dire, si et seulement si

$$mj = i \pmod{r} \;\Rightarrow\; mj = i + lr$$

Soit $k = \text{pgcd}(m, r)$. Notons $m = m'k$ et $r = r'k$. Alors, $x^m = z$ si et seulement si

$$m'kj = i + lr'k$$

Remarquons que

$$m'kj = i + lr'k \;\Rightarrow\; k \mid i \;\Rightarrow z^{\frac{r}{k}} = g^{i\frac{r}{k}} = (g^r)^{\frac{i}{k}} = e$$

Supposons donc que $k \mid i$ et notons $i = ki'$. Alors, $x^m = z$ si et seulement si

$$m'j = i' + lr'$$

Comme m' et r' sont relativement premiers, en vertu du théorème de Bézout, il existe des entiers u et v tels que

$$um' + vr' = 1$$

Par conséquent,

$$m'j = i' + lr' \;\Leftrightarrow\; um'j = ui' + ulr' \;\Leftrightarrow\; (1 - vr')j = ui' + ulr' \;\Leftrightarrow\; j = ui' + r'(ul + vj)$$

Or,

$$\exists\, l \;\Big|\; j = ui' + r'(ul + vj) \;\Leftrightarrow\; j \equiv ui' \pmod{r'}$$

d'où il suit que j est déterminé modulo r'. Il y a donc $\frac{r}{r'} = k$ solutions:

$$j \equiv ui' \pmod{r},\; j \equiv ui' + r' \pmod{r},\; j \equiv ui' + 2r' \pmod{r},\; \cdots,\; j \equiv ui' + (k-1)r' \pmod{r},$$

□

Corollaire 3.6. *Soit p premier. Alors,*

$$\#\left\{x \in \mathbb{Z}_p^* \;\middle|\; x^m = 1\right\} = \text{pgcd}(m, p-1)$$

Lemme 3.7. *Soit p premier. L'ensemble des carré $(\mathbb{Z}_p^*)^2$ est un sous-groupe de $\mathbb{Z}_p^*$ et $[\mathbb{Z}_p^* : (\mathbb{Z}_p^*)^2] = 2$.*

Démonstration. L'application $f : \mathbb{Z}_p^* \to \mathbb{Z}_p^*$ définie par $f(x) = x^2$ est un morphisme de groupes. Son image est un sous-groupe que nous notons $(\mathbb{Z}_p^*)^2$. Remarquons que si $x^2 = a \pmod p$, alors $(p-x)^2 \equiv a \pmod p$ (**NB**: $p - x \equiv -x \pmod p$). Par ailleurs, comme $\mathbb{Z}_p$ est un corps, l'équation $x^2 = a$ admet au plus deux solutions dans $\mathbb{Z}_p$. Par conséquent, l'ensemble des carrés contient $\frac{p-1}{2}$ éléments. En vertu du théorème de Lagrange, l'ordre de $(\mathbb{Z}_p^*)^2$ divise celui de $\mathbb{Z}_p$, par conséquent l'ordre de $\mathbb{Z}_p/(\mathbb{Z}_p^*)^2$ vaut $\frac{\frac{p-1}{2}}{\frac{p-1}{2}} = 2$. □

Lemme 3.8. *Soit p premier. L'ensemble $H = \left\{a \in \mathbb{Z}_p^* \;\middle|\; a^{\frac{p-1}{2}} \equiv 1 \pmod p\right\}$ est un sous-groupe de $\mathbb{Z}_p^*$ d'ordre $\frac{p-1}{2}$.*

Démonstration. De toute évidence, H est un sous-groupe de $\mathbb{Z}_p^*$. Comme p est premier, nous savons que

$$\#\left\{x \in \mathbb{Z}_p^* \;\middle|\; x^m = 1\right\} = \text{pgcd}(m, p-1)$$

Par conséquent, H contient $\text{pgcd}(\frac{p-1}{2}, p-1) = \frac{p-1}{2}$ éléments. □

Lemme 3.9. *Pour n impair, on a*

(1) $\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right)$
(2) $\left(\frac{a}{n}\right) = 0 \Leftrightarrow \text{pgcd}(a, n) > 1$.

Démonstration.

(1) Commençons par supposer que $n = p$ est premier. Rappelons que $\mathbb{Z}_p - \{0\}$ est un groupe.

Si p divise a ou b, alors, d'une part, $\left(\frac{a}{p}\right) = 0$ ou $\left(\frac{b}{p}\right) = 0$ et, d'autre part, p divise ab de sorte que

$$\left(\frac{ab}{p}\right) = 0 = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$$

Si a et b sont des carrés non nuls mod p (*i.e.* $a \equiv x^2 \pmod p$ et $b \equiv y^2 \pmod p$), alors, d'une part, $\left(\frac{a}{p}\right) = 1 = \left(\frac{b}{p}\right)$ et, d'autre part, $ab \equiv x^2y^2 \equiv (xy)^2 \pmod p$ de sorte que

$$\left(\frac{ab}{p}\right) = 1 = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$$

Si a et ab sont des carrés non nuls mod p (*i.e.* $a \equiv x^2 \pmod p$ et $ab \equiv z^2 \pmod p$), alors, d'une part, $\left(\frac{a}{p}\right) = 1 = \left(\frac{ab}{p}\right)$ et, d'autre part, $b \equiv a^{-1}ab \equiv (x^{-1})^2z^2 \equiv (x^{-1}z)^2 \pmod p$ de sorte que

$$\left(\frac{ab}{p}\right) = 1 = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$$

Finalement, si a et b ne sont pas des carrés mod p, alors, d'une part, $\left(\frac{a}{p}\right) = -1 = \left(\frac{b}{p}\right)$. D'autre part, comme nous l'avons vu, $\mathbb{Z}_p^*/(\mathbb{Z}_p^*)^2$ est de cardinal 2. Soit $x = c^2$ un carré non nul mod p et $[x]$ la classe de x. Alors $a \notin [x]$ et $b \notin [x]$. En effet, si par exemple $a \in [x]$, alors

$$ax^{-1} = d^2 \;\Rightarrow\; a = d^2x = d^2c^2 = (dc)^2$$

une contradiction. Il suit que a et b sont dans la même classe, en d'autres termes

$$ab^{-1} = f^2 \;\Rightarrow\; ab^{-1}b^2 = f^2b^2 \;\Rightarrow\; ab = f^2b^2 = (fb)^2\ .$$

Par conséquent, ab est un carré non nul mod p, de sorte que

$$\left(\frac{ab}{p}\right) = 1 = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$$

La formule pour n composé découle immédiatement de la formule pour n premier.

(2) On note $n = p_1^{n_1} \cdots p_k^{n_k}$. On a

$$\left(\frac{a}{n}\right) = 0 \;\Leftrightarrow\; \exists\ 1 \le i \le k \;\Big|\; \left(\frac{a}{p_i}\right) = 0 \;\Leftrightarrow\; \operatorname{pgcd}(a, n) > 1$$

□

Lemme 3.10 (Euler). *Si p est premier $\neq 2$, alors, pour tout $a \in \mathbb{Z}$,*

$$a^{\frac{p-1}{2}} \equiv \left(\frac{a}{p}\right) \pmod p$$

Démonstration. Supposons que $a \not\equiv 0 \pmod p$. Remarquons qu'en vertu du petit théorème de Fermat, on a

$$\left(a^{\frac{p-1}{2}}\right)^2 \equiv a^{p-1} \equiv 1 \pmod p$$

Comme p est premier, $\mathbb{Z}_p$ est un corps et l'équation $x^2 = 1$ admet au plus deux solutions: 1 et $-1 \equiv p-1 \pmod{p}$ ($(p-1)^2 = p^2 - 2p + 1 \equiv 1 \pmod{p}$). Par conséquent,

$$a^{\frac{p-1}{2}} \equiv \pm 1 \pmod{p}$$

Or, comme p est premier, nous savons que le sous-groupe H de $\mathbb{Z}_p^*$ des éléments a tels que $a^{\frac{p-1}{2}} = 1$ contient $\frac{p-1}{2}$ éléments.

Par ailleurs, l'ensemble $(\mathbb{Z}_p^*)^2$ des carrés est un sous-groupe de $\mathbb{Z}_p^*$ d'ordre $\frac{p-1}{2}$. De plus, si $a = b^2$, alors $a^{\frac{p-1}{2}} = b^{p-1} = 1$. Par conséquent, $(\mathbb{Z}_p^*)^2 \subseteq H$ et comme ces deux ensembles ont le même nombre d'éléments, on a $(\mathbb{Z}_p^*)^2 = H$.

En résumé, si $a \equiv 0 \pmod{p}$, alors $a^{\frac{p-1}{2}} \equiv (l \cdot p)^{\frac{p-1}{2}} \equiv 0 \pmod{p}$. Si $a \equiv b^2 \not\equiv 0 \pmod{p}$, alors $a^{\frac{p-1}{2}} \equiv 1 \pmod{p}$ car $(\mathbb{Z}_p^*)^2 \subseteq H$. Finalement, si a n'est pas un carré modulo p, alors $a^{\frac{p-1}{2}} \not\equiv 1 \pmod{p}$ car $H = (\mathbb{Z}_p^*)^2$. Mais comme

$$\left(a^{\frac{p-1}{2}}\right)^2 \equiv a^{p-1} \equiv 1 \pmod{p} ,$$

il suit que $a^{\frac{p-1}{2}} \equiv -1 \pmod{p}$. □

Lemme 3.11. *Pour n impair, on a*

$$\left(\frac{-1}{n}\right) = (-1)^{\frac{n-1}{2}}$$

Démonstration. Si n est premier, c'est une conséquence du lemme d'Euler. Supposons maintenant que $n = p_1 \cdots p_r$ avec $p_1, \cdots, p_r$ premiers (non tous nécessairement distincts). Alors

$$\left(\frac{-1}{n}\right) = \prod_{i=1}^{r} \left(\frac{-1}{p_i}\right) = (-1)^{\sum_{i=1}^{r} \frac{p_i-1}{2}} = (-1)^h$$

avec h le nombre d'indices i pour lesquels $\frac{p_i-1}{2}$ est impair, c'est-à-dire le nombres d'indices pour lesquels $\frac{p_i-1}{2} = 2k+1 \Rightarrow p_i = 4k+3 \Rightarrow p_i \equiv 3 \pmod{4}$. Par ailleurs, comme n est impair, $n \equiv 1 \pmod{4}$ ou $n \equiv 3 \pmod{4}$. Il suit que pour tout indice $1 \leq i \leq r$, $p_i \equiv 1 \pmod{4}$ ou $p_i \equiv 3 \pmod{4}$. Par conséquent, $n \equiv 3^h \pmod{4}$. Conclusion: $n \equiv 1 \pmod{4}$ si h est pair et $n \equiv 3 \pmod{4}$ si h est impair et on a bien

$$\frac{n-1}{2} \equiv h \pmod{2}$$

□

Définition 3.12. On définit

$$G_2 := \left\{ a \in \mathbb{Z}_n^* \;\middle|\; a^{\frac{n-1}{2}} \equiv \pm 1 \pmod{n} \right\} \text{ et } G_3 := \left\{ a \in \mathbb{Z}_n^* \;\middle|\; a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) \pmod{n} \right\}$$

Remarque 3.13.

(1) G_2 et G_3 sont des sous-groupes de G_0.
(2) On a

$$G_3 \subseteq G_2 \subseteq G_1 \subseteq G_0$$

En effet, si $a \in \mathbb{Z}_n^*$ et $a^{\frac{n-1}{2}} \equiv \pm 1 \pmod{n}$, alors

$$a^{n-1} \equiv \left(a^{\frac{n-1}{2}}\right)^2 \equiv 1 \pmod{n} ,$$

ce qui montre que $G_2 \subseteq G_1$. Par ailleurs, si $a \in \mathbb{Z}_n^*$, alors $a \not\equiv 0 \pmod{n}$, donc $\left(\frac{a}{n}\right) = \pm 1$, ce qui montre que $G_3 \subseteq G_2$.

LEMME 3.14. *On a*

$$G_3 = G_0 \;\Leftrightarrow\; n \textit{ est premier}$$

DÉMONSTRATION. $\Leftarrow$: C'est le lemme d'Euler démontré plus haut.

$\Rightarrow$: Supposons que n ne soit pas premier. Nous devons montrer qu'il existe $a \in \mathbb{Z}_n^*$ tel que $a^{\frac{n-1}{2}} \not\equiv \left(\frac{a}{n}\right) \pmod{n}$. Notons

$$n = p_1^{n_1} \cdots p_r^{n_r}$$

où p_i sont des premiers distincts.

Supposons que $n_1 = 1 = \cdots = n_r$. Soit $z \in \mathbb{Z}_n^*$ qui n'est pas un carré. Par l'algorithme chinois, choisissons (**nb** p_1 et $p_2 \cdots p_r$ sont relativement premiers)

$$a \equiv 1 \pmod{p_2 \cdots p_r} \text{ et } a \equiv z \pmod{p_1}$$

Alors, a n'est pas un carré mod n (car si $a \equiv u^2 \pmod{n}$, alors $a \equiv u^2 \pmod{p_1 p_2 \cdots p_r}$, donc a est un carré modulo p_1). Par conséquent,

$$\left(\frac{a}{n}\right) = -1$$

Par ailleurs,

$$\text{si } a^{\frac{n-1}{2}} \equiv -1 \pmod{n}, \text{ alors } a^{\frac{n-1}{2}} \equiv -1 \pmod{p_2 \cdots p_r} .$$

Comme $a \equiv 1 \pmod{p_2 \cdots p_r}$, il suit que

$$a^{\frac{n-1}{2}} \not\equiv -1 \pmod{n}$$

ce qui montre que $a \in G_0$ mais $a \notin G_3$.

Supposons maintenant que $n_1 \geq 2$. Alors, il existe $a \in \mathbb{Z}_n^*$ d'ordre $p_1(p_1 - 1)$. En effet, rappelons que

$$\mathbb{Z}_n^* \simeq \mathbb{Z}_{p_1^{n_1}} \times \cdots \times \mathbb{Z}_{p_r^{n_r}}$$

De plus, $\mathbb{Z}_{p_1^{n_1}}$ est cyclique d'ordre $\varphi(p_1^{n_1}) = p_1^{n_1}(p_1 - 1)$. Soit g un générateur. Alors,

$$a = g^{\left(p_1^{n_1-2}\right)}$$

est d'ordre $p_1(p_1 - 1)$ car

$$a^{p_1(p_1-1)} = \left(g^{\left(p_1^{n_1-2}\right)}\right)^{p_1(p_1-1)} = g^{p_1^{n_1-1}(p_1-1)} = 1$$

et pour tout $1 \leq s < p_1(p_1 - 1)$,

$$a^s = \left(g^{\left(p_1^{n_1-2}\right)}\right)^s \neq 1$$

Alors,

$$a^{n-1} \neq 1 \text{ car } p_1 \nmid n - 1$$

ce qui montre que $a \in G_0$ mais $a \notin G_3$. □

Remarque 3.15.

(1) En vertu du lemme qui précède, nous savons que si n est composé, G_3 est un sous-groupe différent de G_0. En vertu du théorème de Lagrange, l'ordre de G_3 divise l'ordre de G_0 et $[G_0 : G_3] \geq 2$, ce qui signifie qu'en prenant aléatoirement un a dans G_0, il y a au moins une chance sur deux d'avoir $a \notin G_3$ (c'est-à-dire que a soit un témoin), donc qu'il ne passe pas le test de Solovay-Strassen. Par conséquent, si n passe successivement k fois le test de Solovay-Strassen, on peut affirmer qu'il est premier, avec une probabilité supérieure à

$$1 - \frac{1}{2^k}$$

(2) En théorie analytique des nombres, on peut montrer, sous l'hypothèse de Riemann généralisée, que si tous les $a \in [2, \lfloor 2\log(n)^2 \rfloor]$ passent le test de Solovay-Strassen, alors n est premier. En résumé, si l'hypothèse de Riemann est vraie, on peut transformer le test probabiliste de Solovay-Strassen en un test déterministe dont le temps d'exécution est "raisonnable" (*i.e.* il dépend de manière polynomiale du nombre de digits de n).

3. Le test de Rabin-Miller.

Le test de Rabin-Miller est meilleur que le test de Solovay-Strassen car on peut montrer que la proportion de menteurs est toujours $\leq \frac{1}{4}$ (sauf pour $n = 9$).

Lemme 3.16 (Rabin-Miller). *Soit n impair. On pose $n - 1 = 2^s M$ avec M impair. Si n est premier et a est relativement premier à n, alors soit*

$$a^M \equiv 1 \ (mod\ n)$$

soit il existe un nombre $0 \leq r \leq s - 1$ tel que

$$a^{2^r M} \equiv -1 \ (mod\ n) \ .$$

Leonhard Euler (1707-1783)

M. O. Rabin (1931, Breslau)

G. L. Miller

Tableau 2. Les tests de primalité probabilistes.

DÉMONSTRATION. Si n est premier, alors $\mathbb{Z}_n^*$ contient $n-1$ éléments. Soit $a \in \mathbb{Z}_n^*$. En vertu du théorème de Lagrange, l'ordre de a, que nous notons α, divise $n-1 = 2^s M$. Par conséquent, $\alpha = 2^t M'$ avec $0 \leq t \leq s$, M' impair et $M' \mid M$.

Si $t = 0$, alors $a^{M'} \equiv 1 \pmod{n}$, donc

$$a^M \equiv 1 \pmod{n} .$$

Si $t \geq 1$, alors α est pair. Posons $x = a^{\frac{\alpha}{2}}$. Alors,

$$x^2 \equiv \left(a^{\frac{\alpha}{2}}\right)^2 \equiv a^\alpha \equiv 1 \pmod{n}$$

Comme α est l'ordre de a, $x = a^{\frac{\alpha}{2}} \not\equiv 1 \pmod{n}$. De plus, comme n est premier, $\mathbb{Z}_n$ est un corps et l'équation $x^2 = 1$ admet au plus deux solutions ($x = 1$ et $x = -1 \equiv n-1 \pmod{n}$). Par conséquent,

$$x = a^{2^{t-1}M'} \equiv -1 \pmod{n}$$

Finalement, comme M et M' sont impairs, $M = kM'$ avec k impair. Il suit que

$$a^{2^{t-1}M} \equiv a^{2^{t-1}M'k} \equiv \left(a^{2^{t-1}M'}\right)^k \equiv (-1)^k \equiv -1 \pmod{n}$$

□

Algorithme de Rabin-Miller Soit n un impair.

(1) Ecrire n sous la forme

$$n - 1 = 2^s M \qquad \text{avec } M \text{ impair}$$

(2) Choisir au hasard un entier $a \in [2, n-1]$.
(3) Calculer les éléments de S:

$$a^M \pmod{n},\ a^{2M} \pmod{n},\ a^{2^2 M} \pmod{n},\ \cdots,\ a^{2^{s-1}M} \pmod{n},\ a^{n-1} \pmod{n}$$

(4) Si n passe le test de Rabin-Miller (voir ci-dessous), répondre: "n est probablement premier"; sinon, répondre: "n est composé".

DÉFINITION 3.17. On dit que n **passe le test de primalité de Rabin-Miller en base** a si les deux conditions suivantes sont vérifiées:

(1) $a^{n-1} \equiv 1 \pmod{n}$,
(2) Si $a^M \not\equiv 1 \pmod{n}$, alors il existe $0 \leq r < s-1$ tel que

$$a^{2^r M} \equiv -1 \pmod{n}$$

REMARQUE 3.18. Concrètement, pour vérifier le point (2), on peut chercher le premier élément $a^{2^r t}$ de S qui est congru à 1 modulo n et vérifier si

$$a^{2^{r-1}t} \pmod{n} = -1$$

DÉFINITION 3.19. Un entier n est un pseudopremier fort en base a si n est composé impair et s'il passe le test de primalité de Rabin-Miller.

DÉFINITION 3.20. Soit n un entier composé impair et un entier $a \in [2, n-1]$. Si n ne passe pas le test de primalité de Rabin-Miller en base a, alors on dit que a **est un témoin de** n (*i.e.* a est témoin que n est composé). On note $t(n)$ le nombre de témoins de n et $b(n) := n - 2 - t(n)$ le nombre de bases pour lesquelles le nombre n passe le test.

Le programme *Python* suivant permet d'effectuer le test de Rabin-Miller:

```
1  def decomp(n):
2      s=0
3      M=n
4      while M%2==0:
5          M=M/2
6          s=s+1
7      print "n=2^"+str(s)+"*"+str(M)
8      return M,s
9  def testrabinmiller(n,M,s,a):
10     if n%2==0:
11         return False
12     else:
13         t1=pow(a,n-1,n)
14         if t1!=1:
15             print "a="+str(a)+" a^(n-1) mod n="+str(t1)
16             return False
17         else:
18             t2=pow(a,M,n)
19             if t2!=1 and t2!=n-1:
20                 r=1
21                 t2=pow(a,2**r*M,n)
22                 while t2!=n-1 and r<s-1:
23                     r=r+1
24                     t2=pow(a,2**r*M,n)
25                 if t2!=n-1:
26                     print "a="+str(a)+" a^(2^rM)<>-1"
27                     return False
28                 else:
29                     return True
30             else:
31                 return True
32 n=input("Tapez un nombre entier: ")
33 M,s=decomp(n)
34 for k in xrange(1,n,1):
35     if not testrabinmiller(n,M,s,k):
36         print k," est un temoin de ",n
```

EXEMPLE 3.21. L'exécution du code montre que 9 a 6 témoins:

```
Tapez un nombre entier: 9
n=2^3*1
a=2 a^(n-1) mod n=4
2  est un temoin de  9
a=3 a^(n-1) mod n=0
3  est un temoin de  9
a=4 a^(n-1) mod n=7
4  est un temoin de  9
a=5 a^(n-1) mod n=7
5  est un temoin de  9
a=6 a^(n-1) mod n=0
6  est un temoin de  9
a=7 a^(n-1) mod n=4
7  est un temoin de  9
```

DÉFINITION 3.22. Soit n impair et $n-1 = 2^s M$ avec M impair. On définit

$$S := \left\{ a \in G_0 \;\middle|\; a^M \equiv 1 \pmod{n} \text{ ou } \exists\, r \in [0, s-1] \text{ tel que } a^{2^r M} \equiv -1 \pmod{n} \right\}$$

REMARQUE 3.23.

(1) Si n est premier, alors $S = G_3 = G_2 = G_1 = G_0$. Nous allons montrer que

$$S \subseteq G_3 \subseteq G_2 \subseteq G_1 \subset G_0$$

Il y a égalité partout si et seulement si n est premier.

(2) Si $n \equiv 3 \pmod 4$, alors $S = G_3$ et les tests de Solovay-Strassen et Rabin-Miller sont équivalents. En effet, si $n = 3 + k4$, alors $\frac{n-1}{2} = 1 + k2$, donc $\frac{n-1}{2} = M$ est impair. Par conséquent, $a \in G_0$ appartient à S si et seulement si $a^M = \pm 1 \pmod n$. Par ailleurs, comme $n-3$ est divisible par 4 et comme le symbole de Legendre d'un carré non nul vaut 1, on trouve

$$\left(\frac{a}{n}\right) = \left(\frac{a}{n}\right) \underbrace{\left(\frac{\left(a^{\frac{n-3}{4}}\right)^2}{n}\right)}_{=1} = \left(\frac{a \cdot \left(a^{\frac{n-3}{4}}\right)^2}{n}\right) = \left(\frac{a^{\frac{2}{2}} \cdot a^{\frac{n-3}{2}}}{n}\right) = \left(\frac{a^{\frac{n-1}{2}}}{n}\right)$$

Comme $a^M = \pm 1 \pmod n$, il suit que

$$\left(\frac{a^{\frac{n-1}{2}}}{n}\right) = a^{\frac{n-1}{2}} \pmod n$$

(3) G_1, G_2 et G_3 sont des sous-groupes mais en général, S n'est pas un sous-groupe. En effet, S est stable par inversion. Par ailleurs, si a, $b \in S$ vérifient

$$a^M \equiv b^M \equiv 1 \pmod n \;\Rightarrow\; (ab)^M \equiv 1 \pmod n \;\Rightarrow\; ab \in S$$

De plus, si, par exemple, $a^M \equiv 1 \pmod n$ et $b^{2^r M} \equiv -1 \pmod n$, alors $(ab)^{2^r M} \equiv -1 \pmod n$. De même, si, par exemple, $a^{2^{r'} M} \equiv -1 \pmod n$ et $b^{2^r M} \equiv -1 \pmod n$, avec $r > r'$, alors

$$(ab)^{2^r M} \equiv (-1)(-1)^{2^{r-r'}} \equiv -1 \pmod n$$

Supposons maintenant que

$$a^{2^r M} \equiv -1 \pmod n \text{ et } b^{2^r M} \equiv -1 \pmod n$$

Dans ce cas, on peut avoir $ab \notin S$. Par exemple, si $\epsilon^2 \equiv 1 \pmod n$ mais $\epsilon \not\equiv \pm 1 \pmod n$ et si $a^{2M} \equiv -1 \pmod n$ (*i.e.* $4 \mid n-1 \;\Rightarrow\; n-1 = 4 \cdot k \;\Rightarrow\; n \equiv 1 \pmod 4$). Alors $a\epsilon \in S$ car $(a\epsilon)^{2M} = -1$. Cependant

$$(\epsilon a^2)^M \equiv \epsilon^M a^{2M} \equiv \epsilon^{2t+1}(-1) \equiv -\epsilon \not\equiv \pm 1 \pmod n$$

ce qui montre que $(\epsilon \cdot a) \cdot a \notin S$ alors que $\epsilon a \in S$ et $a \in S$, c'est-à-dire, que S n'est pas un sous-groupe.

PROPOSITION 3.24. *On a*

$$S \subseteq G_3 \subseteq G_2 \subseteq G_1 \subseteq G_0$$

DÉMONSTRATION. Toutes ces inclusions ont déjà été démontrées sauf $S \subseteq G_3$. Si n est premier, nous avons déjà montré que $S = G_0 = G_3$.

Supponsons donc que $n = p_1 \cdots p_k$ avec p_i premier non tous nécessairement distincts. Soit $a \in S$. Nous devons montrer que $a \in G_3$, c'est-à-dire que $a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) \pmod{n}$. Comme $n-1 = 2^s M$, il suit que $a^{\frac{n-1}{2}} = a^{2^{s-1}M}$ qui vaut $-1 \pmod{n}$ si $r = s-1$ (voir définition de S) et $1 \pmod{n}$ si $r < s-1$.

Cherchons maintenant à calculer le symbole de Legendre $\left(\frac{a}{n}\right)$. Si $a^M = 1$, alors, comme M est impair et $\left(\frac{a}{n}\right) = \pm 1$, il suit

$$\left(\frac{a}{n}\right) = \left(\frac{a}{n}\right)^M = \left(\frac{a^M}{n}\right) = \left(\frac{1}{n}\right) = 1$$

et on a bien que

$$a^{\frac{n-1}{2}} = \left(\frac{a}{n}\right)$$

Supposons maintenant que $a^{2^r M} \equiv -1 \pmod{n}$. Ecrivons $p_i - 1 = 2^{s_i} M_i$. Remarquons que comme $p_i \mid n$, il suit que $a^{2^r M} \equiv -1 \pmod{p_i}$. Soit o l'ordre de a modulo p_i. En vertu du théorème de Lagrange, $o \mid p_i - 1 = 2^{s_i} M_i$. Donc $o = 2^m M_i'$ avec M_i' impair divisant M_i et $m \leq s_i$. Remarquons que $a^{2^{r+1}M} \equiv 1 \pmod{p_i}$. Par conséquent, $m \leq r+1$. Comme $a^o \equiv 1 \pmod{p_i}$, $m = r+1$. En effet, si $m \leq r$, alors

$$a^{2^r M} \equiv \left(a^{2^m M_i'}\right)^{2^{r-m}\frac{M_i}{M_i'}} \equiv (1)^{2^{r-m}\frac{M_i}{M_i'}} \pmod{p_i} \equiv 1 \pmod{p_i}$$

une contradiction, car par hypothèse, $a^{2^r M} \equiv -1 \pmod{n}$ et donc $a^{2^r M} \equiv -1 \pmod{p_i}$. Par conséquent, l'ordre de a modulo p_i est de la forme $2^{r+1} M_i'$ avec M_i' impair, ce qui signifie que $a^{2^{r+1} M_i'} \equiv 1 \pmod{p_i}$ et que $a^{2^r M_i'} \not\equiv 1 \pmod{p_i}$. Or, comme p_i est premier, l'équation $x^2 \equiv 1 \pmod{p_i}$ admet au plus deux solution dans $\mathbb{Z}_{p_i}^*$: 1 et $-1 \equiv p_i - 1 \pmod{p_i}$. Nous pouvons en conclure que $a^{2^r M_i'} \equiv -1 \pmod{p_i}$. Remarquons encore que $r+1 \leq s_i$ et que $\frac{M_i}{M_i'}$ est impair. Il suit de ces considérations que, modulo p_i, on a, en vertu du lemme d'Euler,

$$\left(\frac{a}{p_i}\right) \equiv a^{\frac{p_i-1}{2}} \equiv a^{2^{s_i-1}M_i} \equiv \left(a^{2^{s_i-1}M_i'}\right)^{\frac{M_i}{M_i'}} \equiv \begin{cases} 1 & \text{si } s_i > r+1 \\ -1 & \text{si } s_i = r+1 \end{cases} \pmod{p_i}$$

Notons h le nombre d'indices pour lesquels $s_i = r+1$. Alors, comme $n = p_1 \cdots p_k$, il suit que

$$\left(\frac{a}{n}\right) = \prod_{i=1}^{k} \left(\frac{a}{p_i}\right) = (-1)^h$$

Par ailleurs, en notant $\xi = \prod_{i=1}^{k} M_i$ (**NB** ξ est impair)

$$\begin{aligned} n = 1 + 2^s M &= \prod_{i=1}^{k} p_i \\ &\Rightarrow = \prod_{i=1}^{k} (1 + 2^{s_i} M_i) = 1 + \xi h 2^{r+1} + L 2^{r+2} \Rightarrow \; 1 + 2^s M = 1 + h 2^{r+1} \xi + L 2^{r+2} \\ \Rightarrow \; 2^{r+1} \left(M 2^{s-(r+1)} - h\xi\right) = L 2^{r+2} \; &\Rightarrow \; M 2^{s-(r+1)} - h\xi = 2L \; \Rightarrow \; h\xi = M 2^{s-(r+1)} - 2L \end{aligned}$$

Par conséquent, si $r < s-1$, alors $s-(r+1) > s-(s-1+1) = 0$ et h est pair, donc

$$\left(\frac{a}{n}\right) \equiv 1 \equiv a^{\frac{n-1}{2}} \pmod{n}$$

Si $r = s-1$, alors h est impair et

$$\left(\frac{a}{n}\right) \equiv -1 \equiv a^{\frac{n-1}{2}} \pmod{n}$$

□

Le théorème suivant, dû à Rabin, garantit que si n est composé, la proportion de "menteurs" pour le test de Rabin-Miller est toujours inférieur à $\frac{1}{4}$. Par conséquent, si k nombres a tirés au hasard passent le test de Rabin-Miller, alors la probabilité que n soit effectivement premier est

$$1 - \frac{1}{4^k}$$

ce qui est meilleur que le test de Solovay-Strassen. Par ailleurs, comme pour le test de Solovay-Strassen, le test de Rabin-Miller devient déterministe sous l'hypothèse de Riemann.

Nous allons maintenant énoncer et démontrer le théorème de Rabin. Nous avons besoin de quelques définitions et d'un lemme.

Définition 3.25. Soit $m = 2^s M$ pair avec M impair, G un groupe abélien et $\alpha \in G$ est un élément d'ordre 2. Nous allons considérer les $s+1$ équations suivantes dans G:

$$\begin{aligned}
x^M &= 1, && (E_0(G,\alpha))\\
x^M &= \alpha, && (E_1(G,\alpha))\\
x^{2M} &= \alpha, && (E_2(G,\alpha))\\
&\vdots \quad \vdots && \vdots\\
x^{2^{j-1}M} &= \alpha, && (E_j(G,\alpha))\\
&\vdots \quad \vdots && \vdots\\
x^{2^{s-1}M} &= \alpha, && (E_s(G,\alpha))
\end{aligned}$$

Lemme 3.26. *Soit G un groupe cyclique d'ordre pair $\phi = 2^u v$ avec v impair et $v \geq 1$ et soit $m = 2^s M$ pair avec M impair. Soit $r = \min(u,s)$ et $w = \operatorname{pgcd}(M,v)$. Alors:*

(1) *L'équation $E_0(G,\alpha)$ a w solutions*
(2) *Pour $1 \leq j \leq r$, l'équation $E_j(G,\alpha)$ a $2^{j-1}w$ solutions.*
(3) *Pour $j > r$, l'équation $E_j(G,\alpha)$ n'a pas de solution.*

Démonstration.

(1) En vertu du lemme démontré plus haut, l'équation $E_0(G,\alpha)$ admet des solutions si et seulement si $1^{\frac{\phi}{w}} = 1$. Toujours en vertu du lemme, $E_0(G,\alpha)$ admet $\operatorname{pgcd}(\phi, M) = w$ solutions.

(2)-(3) Supposons que $j \geq 1$. Soit

$$k = \operatorname{pgcd}(\phi, 2^{j-1}M)$$

Pour que l'équation $E_j(G,\alpha)$ admette des solutions, il faut et il suffit, en vertu du lemme démontré plus haut, que $\alpha^{\frac{\phi}{k}} = 1$, c'est-à-dire que $\frac{\phi}{k}$ soit pair puisque par hypothèse, $\alpha \in G$ est d'ordre 2. Or

$$k = \text{pgcd}(2^u v, 2^{j-1}M) = 2^{\min(u,j-1)}w$$

et

$$\frac{\phi}{k} = \frac{2^u v}{2^{\min(u,j-1)}\omega}$$

Par conséquent, $\frac{\phi}{k}$ est pair si et seulement si $\min(u, j-1) < u$, c'est-à-dire si et seulement si $j - 1 < u$. Comme par ailleurs $j \leq s$, nous trouvons que $E_j(G,\alpha)$ admet des solutions si et seulement si $j \leq r$. Dans ce cas, en vertu du lemme, l'équation $E_j(G,\alpha)$ admet $\text{pgcd}(\phi, 2^{j-1}M) = k = 2^{j-1}w$ solutions.

□

THÉORÈME 3.27 (Rabin, 1976). *Soit n impair composé. Si $n \neq 9$, alors*

$$\frac{\#S}{\#G_0} \leq \frac{1}{4}$$

où, par exemple, $\#S$ désigne le nombre d'éléments de S.

DÉMONSTRATION. Soit $n = p_1^{n_1} \cdots p_N^{n_N}$ la décomposition de n en facteurs premiers. Par ailleurs, nous notons $n - 1 = 2^s M$ avec M impair.

En vertu du lemme chinois, on a un isomorphisme de groupe

$$\mathbb{Z}_n^* \simeq \prod_{i=1}^{N} \mathbb{Z}_{p_i^{n_i}}^*$$

Nous avons vu que $\mathbb{Z}_{p_i^{n_i}}^*$ est cyclique d'ordre ϕ_i (nous savons que $\phi_i = \varphi(p_i^{n_i}) = p_i^{n_i-1}(p_i - 1)$). Notons $\phi_i = 2^{u_i} v_i$ avec v_i impair, $w_i = \text{pgcd}(M, v_i)$ et $v_i' = \frac{v_i}{w_i}$. Nous introduisons également

$$V = \prod_{i=1}^{N} v_i, \qquad V' = \prod_{i=1}^{N} v_i' \ .$$

Il suit que

$$\varphi(n) = \prod_{i=1}^{N} \phi_i = 2^U V \text{ où } U = \sum_{i=1}^{N} u_i \ .$$

Le nombre de solutions de l'équation $E_j(\mathbb{Z}_n^*, -1)$ est donné par le produit du nombre de solutions de l'équation $E_j(\mathbb{Z}_{p_i^{n_i}}^*, -1)$ pour $1 \leq i \leq N$. En vertu du lemme qui précède, l'équation $E_j(\mathbb{Z}_{p_i^{n_i}}^*, -1)$ admet des solutions si et seulement si $j \leq \min(u_i, s)$. Par conséquent, l'équation $E_j(\mathbb{Z}_n^*, -1)$ admet des solutions si et seulement si

$$j \leq r := \min\left(\min\{u_i \;\middle|\; 1 \leq i \leq N\}, s\right)$$

Supposons donc que $j \leq r$.

Pour $j = 0$, l'équation $E_0(\mathbb{Z}^*_{p_i^{n_i}}, 1)$ admet w_i solutions. Il suit que l'équation $E_0(\mathbb{Z}^*_n, 1)$ admet

$$A_0 = \prod_{i=1}^{N} w_i = \prod_{i=1}^{N} \frac{v_i}{v'_i} = \frac{\prod_{i=1}^{N} v_i}{\prod_{i=1}^{r} v'_i} = \frac{V}{V'}$$

solutions.

Pour $1 \leq j \leq r$, l'équation $E_j(\mathbb{Z}^*_{p_i^{n_i}}, -1)$ admet $2^{j-1}w_i$ solutions. Il suit que l'équation $E_j(\mathbb{Z}^*_n, -1)$ admet

$$A_j = \prod_{i=1}^{N} 2^{j-1} w_i = 2^{N(j-1)} \prod_{i=1}^{N} w_i = 2^{N(j-1)} \frac{V}{V'}$$

solutions.

Le nombre total des solutions vaut:

$$A = \sum_{j=0}^{r} A_j = \frac{V}{V'} \left(1 + 1 + 2^N + 2^{2N} + \cdots + 2^{N(r-1)}\right) = \frac{V}{V'} \left(1 + \frac{1 - 2^{Nr}}{1 - 2^N}\right)$$

et il suit que

$$\frac{\#S}{\#G_0} = \frac{A}{\varphi(n)}$$

Nous distinguons maintenant deux cas:

($N = 1$) Si $N = 1$, notons $n = p^a$, $n - 1 = p^a - 1 = 2^s M$ et $p - 1 = 2^u \Omega$ avec Ω et M impairs. Remarquons que

$$n - 1 = p^a - 1 = (1 + p + p^2 + \cdots p^{a-1})(p - 1)$$

Par conséquent, $p - 1$ divise $n - 1$. Il suit que $r = u = \min(\min\{u\}, s)$. Par ailleurs, notons $\varphi(n) = 2^u v$ avec v impair. Alors,

$$2^u v = \varphi(n) = p^{a-1}(p - 1) = p^{a-1} 2^u \Omega \;\Rightarrow\; v = p^{a-1}\Omega$$

Remarquons que

$$\Omega \mid p - 1 \;\Rightarrow\; \Omega \mid n - 1 \;\Rightarrow\; \Omega \mid M \;\Rightarrow\; \Omega \leq \operatorname{pgcd}(M, v)$$

Remarquons également que pour tout $1 \leq b \leq a$, p^b est relativement premier à $n - 1$. En effet, $\operatorname{pgcd}(p^b, n - 1) = p^c$ avec $0 \leq c \leq b$ et p^c divise $n - 1$ et n, donc p^c divise 1 et $c = 0$. Il suit que p^b est aussi relativement premier à M (rappelons que n est impair), ce qui montre que

$$\operatorname{pgcd}(M, v) = \Omega$$

Il suit que

$$\frac{V}{V'} = \Omega \;\Rightarrow\; A = \Omega \left(1 + 1 + 2 + 2^2 + \cdots + 2^{u-1}\right) = \Omega \left(1 + \frac{1 - 2^u}{1 - 2}\right) = \Omega 2^u = p - 1$$

$$\Rightarrow\; \frac{\#S}{\#G_0} = \frac{A}{\varphi(n)} = \frac{p - 1}{p^{a-1}(p - 1)} = \frac{1}{p^{a-1}} \overset{n \neq 9 \text{ et impair}}{\Rightarrow} \frac{\#S}{\#G_0} \leq \frac{1}{5} \leq \frac{1}{4}$$

($N > 1$) Supposons que $N > 1$. Alors

$$A = \frac{V}{V'}\left(1 + \frac{1-2^{Nr}}{1-2^N}\right) = \frac{V}{V'}\left(1 + \frac{2^{Nr}-1}{2^N-1}\right) \leq \frac{V}{V'}\left(1 + \frac{2^{Nr}}{2^N-1}\right) \leq \frac{V}{V'}\left(1 + \frac{2^{Nr}}{2^N}\right)$$
$$= \frac{V}{V'}\left(1 + 2^{N(r-1)}\right) \leq \frac{V}{V'} 2^{N(r-1)+1}$$

Comme $\varphi(n) = 2^U V$, il suit que

$$\frac{\#S}{\#G_0} = \frac{A}{\varphi(n)} \leq \frac{\frac{V}{V'}2^{N(r-1)+1}}{2^U V} = \frac{2^{N(r-1)+1-U}}{V'}$$

Comme pour tout $1 \leq i \leq N$, $u_i \geq r$, il suit que $U = \sum_{i=1}^N u_i \geq Nr$. Par conséquent,

$$\frac{\varphi(n)}{A} = \left(\frac{\#S}{\#G_0}\right)^{-1} \geq 2^{U-N(r-1)-1} V' = 2^{U-Nr} 2^{N-1} V' \geq 2^{N-1} V'$$

Remarquons que

$$\left[\frac{\varphi(n)}{A} \not\geq 4\right] \Rightarrow [N = 2,\ V' = 1 \text{ et } U = 2r]$$

Par ailleurs que M divise $n-1$. Par conséquent M est premier à chaque p_i. Rappelons que

$$\phi_i = 2^{u_i} v_i = p_i^{n_i-1}(p_i - 1) \text{ et } w_i = \operatorname{pgcd}(M, v_i) \text{ et } v_i' = \frac{v_i}{w_i} \text{ et } p_i^{n_i-1}(p_i-1) = 2^{u_i} v_i' w_i$$

Si $n_i > 1$, comme M est premier à chaque p_i, il suit que w_i est premier à chaque p_i et donc $p_i^{n_i-1} \mid v_i'$. Par conséquent, V' est divisible par tous les p_i qui apparaissent plus qu'une fois dans la décomposition en facteurs premiers de n. Par conséquent, $V' = 1$ seulement si $n = p_1 p_2$.

En résumé, si $\frac{\varphi(n)}{A} \not\geq 4$, alors

$$n = p_1 p_2,\ v_1' = v_2' = 1,\ n-1 = 2^s M,\ p_1 - 1 = 2^r v_1 \text{ et } p_2 - 1 = 2^r v_2 \text{ avec } v_1 \mid M \text{ et } v_2 \mid M \text{ et } r \leq s$$

Cela implique que $p_1 - 1$ et $p_2 - 1$ divisent $n-1$, ce qui est impossible: on a en effet

$$n - 1 = p_1 p_2 - 1 = (p_1 - 1)(p_2 - 1) + (p_1 - 1) + (p_2 - 1)$$

et il en résulterait que $p_1 - 1$ divise $p_2 - 1$ et réciproquement.

□

4. Production de clefs RSA

Pour construire des grands nombres premiers utilisables pour des clefs RSA, nous allons tirer au hasard un grand nombre entier impair N_0 et chercher le prochain nombre premier p par l'algorithme:

(1) Tester si N_0 est premier.
(2) Si N_0 est premier, alors $p = N_0$, sinon ajouter 2 à N_0 et recommencer au point (1).

En vertu du théorème des nombres premiers, il y a environ $\frac{H}{\ln(N_0)}$ nombres premiers dans l'intervalle $[N_0, N_0 + H]$ ($\frac{N_0+H}{\ln(N_0+H)} - \frac{N_0}{\ln(N_0)} \approx \frac{N_0+H}{\ln(N_0)} - \frac{N_0}{\ln(N_0)}$); on peut donc s'attendre à trouver un nombre premier au bout d'un nombre d'essais proportionnel à $\ln(N_0)$.

Le programme *python* donné ci-dessous permet de trouver des grands nombres premiers grâce à la fonction prochainpremier qui utilise l'algorithme de Rabin-Miller. Elle est comparée à la méthode ntheory.generate.nextprime du module sympy: les résultats sont comparés ainsi que les temps d'exécution. On obtient, par exemple

```
nextprime:  0.54 ''  prochainpremier:  0.81 '' Diff.:  0
nextprime:  0.2 ''  prochainpremier:  0.18 '' Diff.:  0
nextprime:  1.13 ''  prochainpremier:  1.88 '' Diff.:  0
nextprime:  1.89 ''  prochainpremier:  3.26 '' Diff.:  0
nextprime:  0.41 ''  prochainpremier:  0.53 '' Diff.:  0
nextprime:  0.51 ''  prochainpremier:  0.72 '' Diff.:  0
nextprime:  3.01 ''  prochainpremier:  5.63 '' Diff.:  0
nextprime:  0.86 ''  prochainpremier:  1.34 '' Diff.:  0
nextprime:  4.2 ''  prochainpremier:  7.64 '' Diff.:  0
nextprime:  0.95 ''  prochainpremier:  1.52 '' Diff.:  0
```

```
1  import random
2  import sympy
3  import time
4  def decomp(n):
5      s=0
6      M=n
7      while M%2==0:
8          M=M/2
9          s=s+1
10     print "n=2^"+str(s)+"*"+str(M)
11     return M,s
12 def testrabinmiller(n,M,s,a):
13     if n%2==0:
14         return False
15     else:
16         t1=pow(a,n-1,n)
17         if t1!=1:
18             return False
19         else:
20             t2=pow(a,M,n)
21             if t2!=1 and t2!=n-1:
22                 r=1
23                 t2=pow(a,2**r*M,n)
24                 while t2!=n-1 and r<s-1:
25                     r=r+1
26                     t2=pow(a,2**r*M,n)
27                 if t2!=n-1:
28                     return False
29                 else:
30                     return True
31             else:
32                 return True
33 def testrb(n):
34     M,s=decomp(n)
35     max=10
36     for k in xrange(0,max,1):
37         a=random.randint(2,n-1)
38         t=testrabinmiller(n,M,s,a)
39         if not t:
40             break
41     return t
42 def prochainpremier(n):
43     if n%2==0:
44         n=n+1
45     while not testrb(n):
```

```
46          n=n+2
47      return n
48  for k in xrange(0,10,1):
49      n0=random.randint(1,10**300)
50      a=time.clock()
51      n=sympy.ntheory.generate.nextprime(n0)
52      b=time.clock()
53      print "nextprime: ",b-a," ''",
54      a=time.clock()
55      m=prochainpremier(n0)
56      b=time.clock()
57      print " prochainpremier: ",b-a," ''",
58      print "Diff.: ",n-m
```

APPENDICE A

Structures algébriques

1. Groupes

1. Définitions et notations.

DÉFINITION 1.1. Un groupe $(G, *)$ est un ensemble G muni d'une application

$$\begin{array}{rccc} * : & G \times G & \longrightarrow & G \\ & (x, y) & \mapsto & x * y \end{array}$$

telle que

(1) l'opération $*$ est associative, *i.e.* $x * (y * z) = (x * y) * z$ pour tous x, y, $z \in G$,
(2) il existe un élément neutre, *i.e.* il existe $e \in G$ tel que $e * x = x * e = x$ pour tout $x \in G$,
(3) chaque élément admet un inverse, *i.e.* pour tout $x \in G$, il existe un élément $x' \in G$ tel que $x * x' = x' * x = e$.

Le groupe G est dit abélien, ou commutatif, si l'opération $*$ est commutative, *i.e.* $x * y = y * x$ pour tous x, $y \in G$.

REMARQUE 1.2. Remarquons que l'élément neutre est forcément unique. En effet, si e et e' sont des éléments neutres, alors

$$e = e * e' = e'$$

Il en va de même pour l'inverse: si x' et x'' sont des inverses de $x \in G$, alors

$$x' = e * x' = (x'' * x) * x' = x'' * (x * x') = x'' * e = x''$$

On note généralement l'inverse de x par x^{-1} et $x*y$ simplement par xy (notation multiplicative). Si le groupe est abélien, l'opération est généralement notée par $+$ et le neutre par 0. De plus, l'inverse d'un élément $x \in G$ se note $-x$.

DÉFINITION 1.3. Soient G et H des groupes et $f : G \to H$ une application. Alors, f est un **morphisme de groupes** si

$$f(xy) = f(x)f(y), \ \forall \ x, \ y \in G$$

Un **isomorphisme** est un morphisme bijectif. Un morphisme de G dans lui-même est appelé un **endomorphisme** et un endomorphisme bijectif est appelé un automorphisme. On définit

$$\mathrm{Ker}(f) = \Big\{ x \in G \ \Big| \ f(x) = e' \Big\} \text{ et } \mathrm{Im}(f) = \Big\{ f(x) \ \Big| \ x \in G \Big\}$$

REMARQUE 1.4. Soit $f : G \to H$ un morphisme de groupes. Notons e le neutre de G et e' le neutre de H.

(1) $f(e) = f(ee) = f(e)f(e) \ \Rightarrow \ e' = f(e)^{-1}f(e) = f(e)$ donc

$$f(e) = e'$$

(2) $f(x^{-1})f(x) = f(x^{-1}x) = f(e) = e'$ et $f(x)f(x^{-1}) = f(xx^{-1}) = f(e) = e'$, par conséquent,
$$f(x^{-1}) = f(x)^{-1}$$

(3) Si f est un isomorphisme, alors f^{-1} est aussi un morphisme:
$$f^{-1}(ab) = f^{-1}(f(x)f(y)) = f^{-1}(f(xy)) = xy = f^{-1}(a)f^{-1}(b)$$

(4) $\mathrm{Ker}(f)$ est un sous-groupe normal de G et $\mathrm{Im}(f)$ est un sous-groupe de H. f est injectif si et seulement si $\mathrm{Ker}(f) = \{e\}$ et f est surjectif si et seulement si $\mathrm{Im}(f) = H$.

2. Sous-groupes.

Définition 1.5. Soit $(G, *)$ un groupe. Un sous-ensemble $H \subseteq G$ est un **sous-groupe** de G si

(1) $x * y \in H$, $\forall\ x, y \in H$,
(2) $(H, *)$ est un groupe.

Remarque 1.6. Soit G un groupe et H un sous-groupe de G. Si e est l'élément neutre de G et e' l'élément neutre de H, alors
$$e * e' = e' = e' * e' \ \ \therefore\ e = e'$$

Définition 1.7. Soit G un groupe et H un sous-groupe. On définit la relation binaire
$$x \sim y \Leftrightarrow xy^{-1} \in H$$

Lemme 1.8. *Soit G un groupe et H un sous-groupe. La relation $\sim$ est une relation d'équivalence.*

Démonstration. La relation $\sim$ est réflexive car $xx^{-1} = e \in H$; elle est symétrique car comme $(xy^{-1})^{-1} = yx^{-1}$, il suit
$$x \sim y \ \Leftrightarrow\ xy^{-1} \in H \ \Leftrightarrow\ (xy^{-1})^{-1} \in H \ \Leftrightarrow\ yx^{-1} \in H \ \Leftrightarrow\ y \sim x$$
Finalement, elle est transitive:
$$x \sim y \text{ et } y \sim z \ \Rightarrow\ xy^{-1} \in H \text{ et } yz^{-1} \in H \ \Rightarrow\ xz^{-1} = xez^{-1} = xy^{-1}yz^{-1} \in H$$

□

Définition 1.9. Soit G un groupe et H un sous-groupe. On note G/H l'ensemble des classes d'équivalence pour la relation $\sim$. Ces classes sont appelées les **classes à droite modulo** H. Les **classes à gauche modulo** H sont définies par la relation $x \sim' y \ \Leftrightarrow\ x^{-1}y \in H$. L'ensemble des classes à gauche modulo H se note $H\backslash G$.

Remarque 1.10. Soit G un groupe et H un sous-groupe. Notons $A : G \to G$ la bijection définie par $A(x) = x^{-1}$. Si y est dans la classe à droite de x modulo H, alors $xy^{-1} \in H$. Or, $xy^{-1} = A(x)^{-1}A(y)$. De plus, $A(x)$ est dans la classe à droite et la classe à gauche de x modulo H. Par conséquent, $A(y)$ est dans la classe à gauche de x modulo H. On en déduit que A induit une bijection de G/H dans $H\backslash G$.

Définition 1.11. Soit G un groupe et H un sous-groupe. Si G/H est fini (alors $H\backslash G$ l'est aussi) le nombre d'éléments de G/H est appelé **l'indice de H dans G** et se note $[G : H]$. Le nombre d'éléments d'un groupe fini G est appelé **l'ordre** de G et se note $[G]$.

THÉORÈME 1.12 (Lagrange). *Soit G un groupe fini et H un sous groupe de G. Alors*

$$[G] = [G : H][H]$$

DÉMONSTRATION. Soit $X \in G/H$ et $x \in X$. Définissons une application $f : H \to G$ par $f(y) = yx$. Alors f est une bijection de H dans X. En effet, si $f(y) = f(z)$, alors $yx = zx$ et donc $y = z$. De plus, si $z \in X$, alors $(xz^{-1})^{-1} = zx^{-1} \in H$ et $f(zx^{-1}) = z$. Par conséquent,

$$\#X = [H],\ \forall\ X \in G/H$$

Finalement, comme en tant qu'ensemble,

$$G = \coprod_{X \in G/H} X$$

il suit que

$$[G] = \#(G/H)[H] = [G : H][H]$$

□

3. Sous-groupe normal.

DÉFINITION 1.13. Soit G un groupe et H un sous-groupe. La relation d'équivalence $\sim$ est **compatible** avec la loi de G si pour tous x, y, a, $b \in G$, on a

$$[x \sim y \text{ et } a \sim b] \ \Rightarrow\ xa \sim yb$$

DÉFINITION 1.14. Soit G un groupe et H un sous-groupe. On dit que H est **normal** si

$$xyx^{-1} \in H\ \forall\ x \in G \text{ et } \forall\ y \in H$$

REMARQUE 1.15. Si G est un groupe abélien, alors tout sous-groupe de G est normal.

LEMME 1.16. *Soit G un groupe et H un sous-groupe. Alors la relation d'équivalence $\sim$ est compatible avec la loi de G si et seulement si H est normal.*

DÉMONSTRATION. Si $\sim$ est compatible avec la loi de G, alors, pour tout $x \in G$ et tout $y \in H$, on a

$$e = xx^{-1} \in H \text{ et } ey^{-1} \in H \ \Rightarrow\ x \sim x \text{ et } e \sim y^{-1} \ \Rightarrow\ xe \sim xy^{-1}$$

$$\Rightarrow\ x\left(xy^{-1}\right)^{-1} = xyx^{-1} \in H$$

ce qui montre que H est normal.

Réciproquement, si H est normal, alors pour tous x, y, a, $b \in G$, on a

$$[x \sim a \text{ et } y \sim b] \ \Rightarrow\ \left[xa^{-1} \in H \text{ et } yb^{-1} \in H\right] \ \Rightarrow\ xyb^{-1}x^{-1} \in H \ \Rightarrow\ xyb^{-1}a^{-1}ax^{-1} \in H$$

$$\overset{xa^{-1} \in H}{\Rightarrow}\ xyb^{-1}a^{-1} \in H \ \Rightarrow\ xy \sim ab$$

ce qui prouve que la relation $\sim$ est compatible avec la loi de G. □

DÉFINITION 1.17. Soit G un groupe et H un sous-groupe normal. On appelle **groupe quotient** de G par H, et on note G/H, l'ensemble des classes d'équivalence pour la relation $\sim$ muni de la loi de G: $[x][y] = [xy]$.

Exemple 1.18. Pour un nombre entier n, $n\mathbb{Z}$ est un sous-groupe normal de $\mathbb{Z}$. Le groupe quotient $\mathbb{Z}/n\mathbb{Z}$ se note parfois simplement $\mathbb{Z}_n$. On note ses éléments par leur représentant $< n$. Ainsi,

$$\mathbb{Z}_n = \{0, 1, 2, \cdots, n-1\}$$

Proposition 1.19. *Soit $n \in \mathbb{N}$. Pour chaque entier d divisant n, il existe un unique sous-groupe de $\mathbb{Z}_n$ d'ordre d: c'est le groupe cyclique engendré par $\frac{n}{d}$ dans $\mathbb{Z}_n$.*

Démonstration. Supposons que $n = dr$. Alors, $r \in \mathbb{Z}_n$ est d'ordre d car $dr \equiv 0 \pmod{n}$. De plus, si $cr \equiv 0 \pmod{n}$, alors n divise cr, c'est-à-dire $cr = kn = kdr$, donc d divise c.

Soit H un sous-groupe de $\mathbb{Z}_n$ d'ordre d. Soit $s : \mathbb{Z} \to \mathbb{Z}_n$ la surjection canonique. Alors, $s^{-1}(H)$ est un sous-groupe de $\mathbb{Z}$, par conséquent, il existe un nombre m tel que $s^{-1}(H) = m\mathbb{Z}$. Or $m\mathbb{Z}$ est engendré par m, ainsi H est engendré par $t = s(m)$. On sait que $dt \equiv 0 \pmod{n}$, donc $kdr = kn = dt$, ce qui montre que r divise t. Par conséquent, H est contenu dans le sous-groupe engendré par r et donc égal à ce sous-groupe. □

Remarque 1.20.

(1) Remarquons que le nombre de générateurs de $\mathbb{Z}_n$ est égal au nombre d'éléments du groupe $\mathbb{Z}_n^*$ des éléments inversibles pour la multiplication de $\mathbb{Z}_n$ qui est égal au nombre de nombres relativement premiers à n inférieurs à n. En effet, si x est un générateur de $\mathbb{Z}_n$ ou si $x \in \mathbb{Z}_n^*$, alors il existe k tel que $kx = 1 + ln$, donc $1 = xk - ln$ ce qui montre que $\text{pgcd}(x, n)$ divise 1. Réciproquement, si x est relativement premier à n, alors, en vertu du théorème de Bézout, il existe deux nombres s et t tels que $1 = sx + tn$, ce qui montre que x est un générateur de $\mathbb{Z}_n$ et $x \in \mathbb{Z}_n^*$. **La fonction indicatrice d'Euler** $\varphi(n)$ est définie par

$$\varphi(n) = \#\mathbb{Z}_n^*$$

(2) On a la formule

$$\varphi(n) = \sum_{d|n} \varphi(d)$$

En effet, $\mathbb{Z}_n$ est l'union des ensembles d'éléments d'ordre d pour d divisant n. Le nombre de ces éléments est le nombre de générateurs de l'unique sous-groupe d'ordre d. Comme il est isomorphe à $\mathbb{Z}_d$, le nombre d'éléments d'ordre d est $\varphi(d)$.

4. Groupes cycliques.

Remarque 1.21. Soit G un groupe et $\{H_i\}_{i\in\Omega}$ une famille de sous-groupes de G. Alors $\bigcap_{i\in\Omega} H_i$ est aussi un sous-groupe de G.

Définition 1.22. Soit G un groupe et $A \subseteq G$ un sous-ensemble de G. On définit le **sous-groupe engendré par** A comme l'intersection de tous les sous-groupes H de G contenant A et on le note $\langle A \rangle$:

$$\langle A \rangle = \bigcap \left\{ H \;\middle|\; A \subseteq H \text{ et } H \text{ est un sous-groupe de } G \right\}$$

L'ordre d'un élément $x \in G$ est l'ordre du sous-groupe $\langle \{x\} \rangle$.

Définition 1.23. Un groupe G est **cyclique** s'il existe $x \in G$ tel que $\langle \{x\} \rangle = G$.

Proposition 1.24. *Tout groupe cyclique est isomorphe soit à $\mathbb{Z}$ soit à $\mathbb{Z}_n$ où n est l'ordre du groupe si celui-ci est fini.*

DÉMONSTRATION. Soit $G = \langle\{x\}\rangle$ un groupe cyclique. On définit $f : \mathbb{Z} \to G$ par $f(k) = x^k$ (et $f(0) = e$). Alors, f est un morphisme de groupe surjectif. Si G est fini (disons $[G] = n$) alors $\ker(f) = n\mathbb{Z}$ $f : \mathbb{Z}/n\mathbb{Z} \to G$ est un isomorphisme. Sinon, $f : \mathbb{Z} \to G$ est un isomorphisme. □

2. Anneaux

1. Définitions et notations.

DÉFINITION 2.1. Un anneau $(A, +, *)$ est un ensemble muni de deux applications $+$ et $*$ de $A \times A$ dans A telles que

(1) $(A, +)$ est un groupe abélien,
(2) $*$ est associative,
(3) $*$ admet un élément neutre (noté 1),
(4) $*$ est distributive à gauche et à droite par rapport à $+$, *i.e.*

$$x * (y + z) = x * y + x * z, \qquad (y + z) * x = y * x + z * x \qquad \forall\ x,\ y,\ z \in A$$

REMARQUE 2.2. L'élément neutre de $*$ est forcément unique et se note généralement par 1. Un anneau commutatif est un anneau dans lequel l'opération $*$ est commutative.

DÉFINITION 2.3. Soient $(A, +, *)$ et $(A', +', *')$ des anneaux. Un morphisme d'anneaux est une application $f : A \to A'$ telle que

(1) $f(x + y) = f(x) +' f(y)$ pour tous x, $y \in A$,
(2) $f(x * y) = f(x) *' f(y)$ pour tous x, $y \in A$,
(3) $f(1) = 1'$ où 1 désigne le neutre de $*$ et 1' le neutre de $*'$.

REMARQUE 2.4. $\mathbb{Z}$ est un anneau et pour tout anneau A, l'application $f : \mathbb{Z} \to A$ définie par $f(m) = m \cdot 1$ est un morphisme d'anneau dont le noyau est de la forme $q\mathbb{Z}$ pour un entier $q \geq 0$.

DÉFINITION 2.5. Soit A un anneau. La **caractéristique** de A est l'entier $q \geq 0$ tel que $q\mathbb{Z}$ soit le noyau du morphisme $m \mapsto m \cdot 1$ de $\mathbb{Z}$ dans A.

REMARQUE 2.6. Si la caractéristique est nulle, alors A contient $\mathbb{Z}$ comme sous-anneau. Si elle est non nulle, disons q, alors q est le plus petit entier > 0 tel que $q \cdot 1 = 0$.

2. Idéaux.

DÉFINITION 2.7. Soit $(A, +, *)$ un anneau commutatif (*i.e.* $*$ est commutative) et $I \subseteq A$ un sous-ensemble de A. Alors, I est un **idéal** si $(I, +)$ est un sous-groupe de $(A, +)$, et si, pour tout $x \in I$ et tout $\lambda \in A$, on a: $\lambda x \in I$.

REMARQUE 2.8. Remarquons immédiatement que:

(1) $I \subseteq A$ est un idéal si et seulement si

$$[x \in I \text{ et } y \in I] \ \Rightarrow\ [x + y \in I]$$

$$[x \in I \text{ et } \lambda \in A] \ \Rightarrow\ [\lambda x \in I]$$

(2) Soit $I \subseteq A$ un idéal de A. Alors,

$$I = A \ \Leftrightarrow\ 1 \in I$$

où 1 désigne le neutre de $*$.

(3) $\{0\}$ et A sont des idéaux de A. Tout idéal de A contient 0.

(4) Si $(I_i)_{i\in\Omega}$ est une famille d'idéaux de A, alors $\bigcap_{i\in\Omega} I_i$ est un idéal de A.

Définition 2.9. Soit A un anneau commutatif et $S \subseteq A$ un sous-ensemble de A. L'intersection de la famille des idéaux de A contenant S est appelée l'**idéal engendré par** S; S est appelé un **système de générateurs** de cet idéal.

Remarque 2.10. L'idéal engendré par le sous-ensemble $S \subseteq A$ est l'ensemble de tous les éléments de la forme

$$\lambda_1 s_1 + \cdots + \lambda_n s_n \text{ où } \lambda_i \in A \text{ et } s_i \in S$$

Définition 2.11. Si $S \subseteq A$ est fini ($S = \{s_1, s_2, \cdots, s_n\}$), on note l'idéal engendré par S simplement par $(s_1, \cdots, s_n)$. Un idéal $I = (s)$ engendré par un élément s est appelé un **idéal principal** et s son **générateur**.

Définition 2.12. Soit A un anneau commutatif. Un élément non nul $x \in A$ est un **diviseur de zéro** s'il existe un élément non nul $y \in A$ tel que $x * y = 0$ (où 0 est le neutre pour $+$). Un anneau est **intègre** s'il n'admet pas de diviseur de zéro.

Définition 2.13. Un anneau intègre est **principal** si tous ses idéaux sont principaux.

3. Divisibilité

Définition 3.1. Un nombre entier m **divise** un nombre entier n s'il existe un nombre entier k tel que $n = km$. On dit aussi que m est un diviseur de n ou que n est divisible par m et on note $m \mid n$.

Proposition 3.2. *L'anneau $\mathbb{Z}$ est principal.*

Démonstration. Soit $I \subseteq \mathbb{Z}$ un idéal de $\mathbb{Z}$. Soit n le plus petit élément strictement positif de I:

$$n = \min\{x \in I \;\Big|\; x > 0\} \ .$$

Alors $(n) \subseteq I$.

Soit $x \in I$. Alors, par l'algorithme de la division, nous savons qu'il existe deux nombres entiers k et r tels que $x = kn + r$ avec $0 \leq r < n$. Comme I est un idéal, nous trouvons que $r = x - kn \in I$ et $0 \leq r < n$. Or, par hypothèse, n est le plus petit élément strictement positif de I. Par conséquent, $r = 0$ et $x = kn$, ce qui prouve que $I \subseteq (n)$ et, par la première partie, que $I = (n)$. □

4. Corps

1. Définitions et notations.

Définition 4.1. Un corps $(C, +, *)$ est un anneau non trivial tel que

(1) l'opération est $*$ est commutative,
(2) $(C\backslash\{0\}, *)$, où 0 désigne le neutre de l'opération $+$, est un groupe abélien.

Remarque 4.2. Un anneau non trivial qui satisfait seulement à l'axiome (2) mais pas forcément à l'axiome (1) est appelé un anneau intègre (*division ring* en anglais). Un corps est donc un anneau intègre commutatif (*field* en anglais). Pour un corps $\mathbb{K}$, on note le groupe $(\mathbb{K} - \{0\}, *)$ par $\mathbb{K}^*$.

DÉFINITION 4.3. Soient C et C' des corps. Un morphisme de corps est une application $f : C \to C'$ qui est un morphisme d'anneaux. Un isomorphisme de corps est un morphisme de corps bijectif.

REMARQUE 4.4. Soit $f : C \to C'$ un morphisme de corps. Les opérations dans C et C' sont désignées par les mêmes symboles. Soit $0 \neq x \in C$. Alors $f(x^{-1}) = f(x)^{-1}$. En effet,

$$f(x) * f(x^{-1}) = f(x * x^{-1}) = f(1) = 1 \;\Rightarrow\; f(x^{-1}) = f(x)^{-1}$$

Si f est un isomorphisme, alors f^{-1} est un morphisme (notons $f(x) = a$ et $f(y) = b$):

$$f(x+y) = f(x) + f(y) = a + b \;\Rightarrow\; f^{-1}(a+b) = x + y = f^{-1}(a) + f^{-1}(b)$$

$$f(x*y) = f(x) * f(y) = a * b \;\Rightarrow\; f^{-1}(a*b) = x * y = f^{-1}(a) * f^{-1}(b)$$

$$f(1) = 1' \;\Rightarrow\; f^{-1}(1') = 1$$

EXEMPLE 4.5. $\mathbb{Z}_p$ est un corps si et seulement si p est premier. En effet, soit $1 \leq x \in \mathbb{Z}_p$. Si p est premier, par le théorème de Bézout, il existe deux nombres s et t tels que

$$1 = \text{pgcd}(x, p) = sx + tp \;\Rightarrow\; sx \equiv 1 \pmod{p}$$

ce qui montre que x admet un inverse.

Réciproquement, si $\mathbb{Z}_p$ est un corps, alors pour tout $1 \leq x \in \mathbb{Z}_p$, il existe $1 \leq s \in \mathbb{Z}_p$ tel que

$$sx \equiv 1 \pmod{p} \;\Rightarrow\; 1 = sx + tp$$

Cette dernière égalité implique que $\text{pgcd}(x, p)$ divise 1. Par conséquent, $\text{pgcd}(x, p) = 1$ pour tout $1 \leq x \leq p-1$, ce qui montre que p est premier.

PROPOSITION 4.6. *Soit $\mathbb{K}$ un corps et $p \in \mathbb{K}[x]$ un polynôme non nul. Alors p admet au plus $\deg(p)$ racines dans $\mathbb{K}$.*

DÉMONSTRATION. Procédons par récurrence sur $n = \deg(p)$. Si $n = 0$, alors $p(x) = c \neq 0$ et p admet aucune racine dans $\mathbb{K}$. Supposons que la proposition soit vraie pour tous les polynômes de degré $\leq n-1$ et soit p un polynôme de degré n à coefficients dans $\mathbb{K}$. Soit λ une racine de p: $p(\lambda) = 0$. Par l'algorithme de la division polynomiale, on trouve

$$p(x) = (x - \lambda)q(x) + r \text{ où } q,\, r \in \mathbb{K}[x],\ \deg(q) \leq n-1 \text{ et } \deg(r) = 0\ .$$

Comme $p(\lambda) = 0$, il suit que $r = 0$. En d'autres termes:

$$\begin{aligned} p(x) &= p(x) - p(\lambda) \\ &= a_n(x^n - \lambda^n) + \cdots + a_1(x - \lambda) + a_0(1 - 1) \\ &= (x - \lambda)\left[a_n(x^{n-1} + \lambda x^{n-2} + \lambda^2 x^{n-2} + \cdots + \lambda^{n-1}) + \cdots + a_2(x + \lambda) + a_1\right] \\ &= (x - \lambda)q(x) \end{aligned}$$

où $q \in \mathbb{K}[x]$ et $\deg(q) \leq n-1$. Si η est une autre racine de p, alors

$$0 = p(\eta) = \underbrace{(\eta - \lambda)}_{\neq 0} q(\eta)$$

et comme $\mathbb{K}$ n'admet pas de diviseurs de zéro, il suit que η est une racine de q. Or, par hypothèse de récurrence, q admet au plus $n-1$ racines dans $\mathbb{K}$, par conséquent, p admet au plus n racines dans $\mathbb{K}$. □

Exemple 4.7. Dans $\mathbb{Z}_5$, on a: $1^2 \equiv 1,\ 2^2 \equiv 4,\ 3^2 \equiv 4,\ 4^2 \equiv 1 \pmod 5$. L'équation $x^2 = 1$ admet effectivement deux solutions: 1 et $4 \equiv -1 \pmod 5$. En revanche, $3^2 \equiv 9^2 \equiv 15^2 \equiv 9 \pmod{18}$ ce qui montre que l'équation $x^2 = 1$ admet 3 solutions dans $\mathbb{Z}_{18}$.

Lemme 4.8. *Soit $\mathbb{K}$ un corps et G un sous-groupe fini de $\mathbb{K}^*$. Alors, G est cyclique. En particulier, $\mathbb{Z}_p^*$ est cyclique.*

Démonstration. Soit $n = \#G$ le nombre d'éléments de G et $\psi(d)$ le nombre d'éléments d'ordre d dans G. Alors, en vertu du théorème de Lagrange, il vient

$$n = \sum_{d|n} \psi(d)$$

Soit d qui divise n. Alors, soit $\psi(d) = 0$, soit il existe un élément $x \in G$ tel que $H = \langle\{x\}\rangle$ est d'ordre d. Tous les éléments de H sont solution de l'équation $X^d = 1$. Mais comme $\mathbb{K}$ est un corps, cette équation possède au plus d solutions. Par conséquent, tous les éléments d'ordre d sont dans H. Or, comme H est isomorphe à $\mathbb{Z}_d$, $\psi(d) = \#\{m < d \text{ premiers à d}\} = \varphi(d)$ la fonction indicatrice d'Euler.

Par conséquent, pour tout $d \mid n$, soit $\psi(d) = 0$ soit $\psi(d) = \varphi(d)$. Or, comme

$$n = \sum_{d|n} \psi(d) \text{ et } n = \sum_{d|n} \varphi(d)$$

il suit que $\psi(d) = \varphi(d)$ pour tout $d \mid n$. En particulier, $\psi(n) = \varphi(n) \geq 1$, ce qui montre que G est cyclique. □

Lemme 4.9. *Soit p un premier impair et a un entier premier à p. Soit $\mathbb{Z}_{p^n}^*$ le groupe des éléments inversibles pour la multiplication de $\mathbb{Z}_{p^n}$. La classe de $1 + ap$ dans $\mathbb{Z}_{p^n}^*$ est d'ordre p^{n-1}*

Démonstration. Commençons par montrer par récurrence la congruence

$$(1 + ap)^{p^k} \equiv 1 + ap^{k+1} \pmod{p^{k+2}}$$

Pour $k = 0$, la congruence est triviale. Pour $k = 1$ on a (C_k^n sont les coefficients binomiaux)

$$(1 + ap)^p = 1 + C_1^p ap + C_2^p a^2 p^2 + l \cdot p^3 \ \Rightarrow\ (1 + ap)^p \equiv 1 + ap^2 + \frac{p(p-1)}{2} a^2 p^2 \pmod{p^3}$$
$$\Rightarrow\ (1 + ap)^p \equiv 1 + ap^2 \pmod{p^3}$$

car comme p est impair, $\frac{p(p-1)}{2} p^2 \equiv 0 \pmod{p^3}$.

Soit $k \geq 2$. Supposons que la congruence soit vraie pour tout nombre $n \leq k-1$. Alors,

$$(1 + ap)^{p^{k-1}} = 1 + ap^k + l \cdot p^{k+1} = 1 + p^k(a + l \cdot p)$$

$$\Rightarrow\ (1 + ap)^{p^k} = \left(1 + p^k(a + l \cdot p)\right)^p = 1 + pp^k\,(a + l \cdot p) + \frac{p(p-1)}{2} p^{2k}\,(a + l \cdot p)^2 + \cdots$$
$$= 1 + pp^k\,(a + l \cdot p) + p^{2k+1} \cdot m = 1 + ap^{k+1} + p^{k+2} \cdot s$$

car comme $k \geq 1$, il suit que $2k+1 \geq k+2$.

En particulier,

$$(1+ap)^{p^{n-1}} \equiv 1 \pmod{p^n}$$

mais

$$(1+ap)^{p^{n-2}} \equiv 1 + p^{n-1} \not\equiv 1 \pmod{p^n}$$

ce qui montre que $p+1$ est d'ordre p^{n-1} dans $\mathbb{Z}_{p^n}^*$. □

PROPOSITION 4.10. *Soit p un premier impair et $n \geq 1$, Alors, $\mathbb{Z}_{p^n}^*$, le groupe des éléments inversibles de $\mathbb{Z}_{p^n}$, est cyclique.*

DÉMONSTRATION. La proposition est déjà démontrée si $n = 1$ (voir plus haut). Supposons que $n \geq 2$. Soit x un générateur de $\mathbb{Z}_p^*$. En d'autres termes,

$$x^{p-1} = 1 + ap$$

(1) Si $p \nmid a$, alors x^{p-1} est d'ordre p^{n-1} dans $\mathbb{Z}_{p^n}^*$ en vertu du lemme qui précède. Par conséquent, x est d'ordre $(p-1)p^{n-1} = \varphi(p^n)$ dans $\mathbb{Z}_{p^n}^*$, ce qui montre que x est un générateur de $\mathbb{Z}_{p^n}^*$.

(2) Si $p \mid a$, on définit $h = x + p$. Alors, $h^{p-1} \equiv 1 \pmod{p}$ et

$$h^{p-1} = x^{p-1} + (p-1)x^{p-2}p + \cdots = 1 + ap - px^{p-2} + p^2x^{p-2} + \cdots \Rightarrow h^{p-1} \equiv 1 + \underbrace{(-x^{p-2})}_{a'} p \pmod{p^2}$$

car $p \mid a$. Comme $p \nmid a' = -x^{p-2}$, il suit par le point (1) que la classe de h dans $\mathbb{Z}_{p^n}^*$ est un générateur de $\mathbb{Z}_{p^n}^*$.

□