RSA

Théorème chinois et de Fermat.

Rappel du protocole RSA.
A (= Alice) et B (= Bob),
B va établir une clé publique et privée :

- Il choisit deux premiers p et q ($\log_2 p \approx \log_2 q \approx 1024$),
- Calcule $n = pq$.
- Il choisit e t.q.
  $\text{pgcd}(e, p-1) = \text{pgcd}(e, q-1) = 1$.
  Choix typique :
  $e = 17, 257, 65537$.
  N.B. ces choix sont premiers,
  $e = 2^4+1, 2^8+1, 2^{16}+1$.
- Sa clé publique est (n, e).
- Avec p et q, il calcule
  $$d = \begin{cases} \bar{e}^1 \bmod (p-1) \\ \bar{e}^1 \bmod (q-1) \end{cases}$$
  $$= \bar{e}^1 \bmod \text{ppcm}(p-1, q-1).$$
- Sa clé privée est (n, d).

Pour envoyer un message $m \in \mathbb{Z}/n\mathbb{Z}$, A va :

- calculer $m^e \bmod n = c$.
- Elle envoie c à B.
- B calcule $m = c^d \bmod n$.

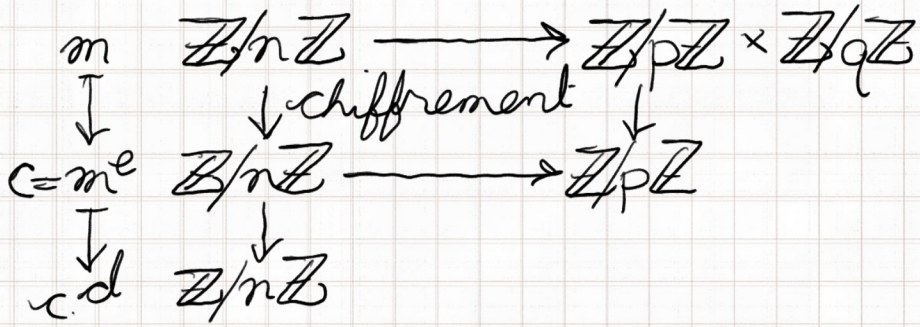
(On affirme que $c^d \equiv m$.)

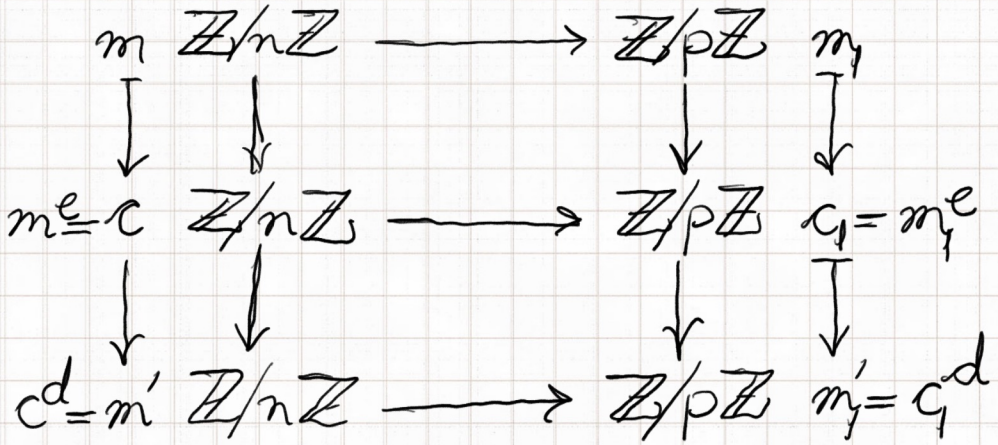
Tout le monde connaît (n, e) et c. [données publiques]

Problème RSA (computationnel)
Etant donné n, e et c, trouver m tel que $m^e \equiv c \bmod n$.

Une 'solution' de ce problème est un algorithme. On pose la question de la meilleur complexité d'une solution.

Preuve que $m = c^d \bmod n$.



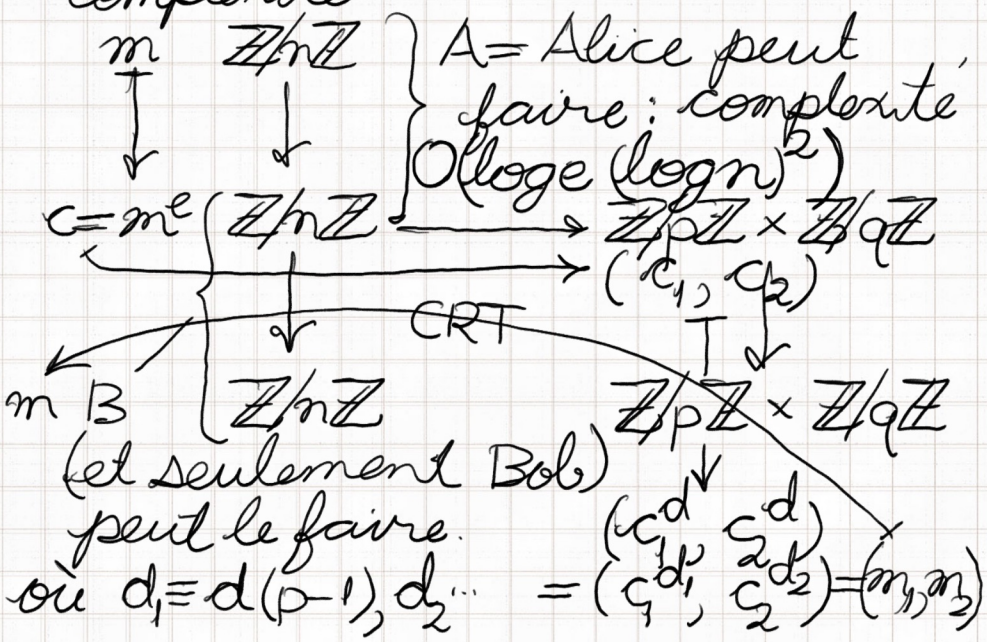


Il suffit de m.q. $m_1' = m_1 \bmod p$.
Par symétrie, on aura :
$m_2 = m \bmod q = m_2' = m_2^{ed} \bmod q$.
Pour montrer
$m_1 \equiv m_1' \equiv m^{ed} \bmod p$,
on rappelle le théorème de Fermat : $a^{p-1} \equiv 1 \bmod p$ si $a \not\equiv 0 \bmod p$. Mais
$ed \equiv 1 \bmod (p-1)$. [par construction de d.]
Alors $ed = 1 + k(p-1)$, et donc
$m_1' = m_1^{ed} = m_1 m_1^{k(p-1)} \bmod p$
$= m_1 \bmod p$.

On a montrer que
$m_1' \equiv m^{ed} \equiv m \bmod p$, et par
$m_2' \equiv m^{ed} \equiv m \bmod q$ sym
par le théorème chinois :

$$\mathbb{Z}/n\mathbb{Z} \xrightarrow{\sim} \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z}$$
$$m \longmapsto (m_1, m_2)$$

et par le fait cette bijection est un homomorphisme,
$c^d \equiv m^{ed} \equiv m \bmod n$ ⊛
[car $(m_1', m_2') = (m_1^{ed}, m_2^{ed})$
$= (m_1, m_2)$.]
On a demontré ~~que~~ ⊛.
Complexité :



En résumé (complexité)

- $c \longmapsto (c_1, c_2)$ réduction modulaire
- $(c_1, c_2) \longmapsto (c_1^{d_1}, c_2^{d_2}) = (m_1, m_2)$

Complexité :

$c_M(\log_2 d_1)\, M(\log_2 p) + (\log_2 d_2)\, M(\log_2 q))$

où

$M(r)$ = complexité de multiplication de deux entiers de r bits.

$c_M(2 \log_2 p\, M(\log_2 p))$

où $\log_2 p \approx \log_2 q$.

- $(m_1, m_2) \longmapsto m = m_1 + up(m_2 - m_1)$

où

$up + vq = 1$. Complexité $M(\log n)$

L'avantage entre $c^d \bmod n$

$c_M(\log_2 d)\, M(\log n) = c_M \log n\, M(\log n)$

vs.

$c_M \frac{(2 \log p)}{\log n}\, M\left(\frac{\log_2 n}{2}\right)$

L'avantage
$M\left(\frac{\log n}{2}\right)$ au lieu de $M(\log_2 n)$.
Si multiplication est quadratique (qui est le cas pour mult. naïve), la différence est un facteur de 4.
Attention Une multiplication en $\mathbb{Z}/n\mathbb{Z}$ est
- mult des entiers de taille $\log_2 n$ bits
- reduction modulaire mod n de ab de taille $2\log_2 n$ bits.

Rappel : Complexité de red. mod. m mod n :
$$O(\log n \log\left(\frac{m}{n}\right))$$
$$= O((\log n)^2).$$

Toujours quadratique.
On peut faire mieux pour la partie multiplication.

Algorithme de Karatsuba

Multiplication naïve des entiers a et b de r bits ($r = \lfloor \log_2 n \rfloor + 1$) est quadratique : $O(r^2)$.

N.B. Taille d'entrée : $2r$ pour les deux entiers a & b.

On écrit M et A pour la complexité de mult. et addition. Pour préciser le nombre de bits, on écrit $M(r)$ et $A(r)$.

On sait que $A = A(r)$ est linéaire : $A(r) = c_A r$.

On suppose que

$$M = M(r) = c_M r^\omega.$$

où $1 < \omega \leq 2$.

Ici $\omega = 2$ est possible avec mult. naïve.

L'idée de Karatsuba est qu'on peut multiplier deux polynômes linéaires $f(x) = a_0 + a_1 x$ et $b_0 + b_1 x = g(x)$ avec $3M + o(M)$ (complexité).
L'algorithme naïf donne une complexité de $4M$:

$$a_0 b_0,\ a_0 b_1,\ a_1 b_0,\ a_1 b_1$$

pour déterminer le résultat

$$a_0 b_0 + (a_0 b_1 + a_1 b_0) x + a_1 b_1 x^2.$$

Donc $4M + 1A$.
On procède autrement...
On calcule $f(1) = a_0 + a_1$ et

$$g(1) = b_0 + b_1$$

avec $2A$, et ensuite

$$f(0)g(0) = a_0 b_0$$
$$f(1)g(1) = a_0 b_0 + a_0 b_1 + a_1 b_0 + a_1 b_1$$
$$f(\infty)g(\infty) = a_1 b_1$$

avec $3M$.

N.B. $f(\infty) = a_1$ et $g(\infty) = b_1$ est une notation formelle.

Ensuite on calcule le terme manquant :

$$a_1b_1 + a_1b_0 = f(1)g(1) - f(0)g(0) - f(\infty)g(\infty)$$

avec 2A (deux soustractions).

Conclusion : L'algorithme permet la multiplication de polynômes linéaires avec complexité 3M + 4A.

Karatsuba pours entiers

On note que $\mathbb{Z}[x] \longrightarrow \mathbb{Z}$ est un $x \longmapsto a$ homomorphisme, envoyant $f(x) \longmapsto f(a)$.

Rappel

On suppose que $M(r) = c_M r^\omega$ et $A(r) = c_A r$.

Soit a et b des entiers de $\leq r$ bits, et posons $s = \lceil r/2 \rceil$, et on écrit

$a = a_0 + a_1 2^s$ et

$b = b_0 + b_1 2^s$, où a_i, b_i sont de s bits.

En appliquant la mult de Karatsuba à

$(a_0+a_1x)(b_0+b_1x), \in \mathbb{Z}[x]$

on obtient

$$a_0b_0+(a_0b_1+a_1b_0)x+a_1b_1x^2 \in \mathbb{Z}[x]$$

$$a_0b_0+(a_0b_1+a_1b_0)2^s+a_1b_1 2^{2s} \in \mathbb{Z}$$

avec complexité

$$3M(s)+4A(s)+2A(2s)$$
$$= 3M(s)+O(s)$$
$$\sim 3M(s).$$

Rappel que $s \sim r/2$ et donc on a, par recurrence

$$M(r) \sim 3M(s) = 3c_M(r/2)^\omega$$
$$= c_M r^\omega$$

En prenant le logarithme

$$\omega \log r = \omega \log r - \omega \log 2 + \log 3.$$

Donc $\omega = \log_2 3.$

$$\approx 1,58\ldots < 2 = \log_2 4$$

En RSA, c'est toujours mieux

$$c \longmapsto (c_1, c_2) \longmapsto (m_1, m_2) \longmapsto m.$$

On avait un avantage de $M(r/2)$ au lieu de $M(r)$, où $r = \log_2 n$.
Pour $M(r) = c_M r^\omega$, avec
$\omega = 2$ c'est un facteur de 4, et
$\omega = \log_2 3$, c'est 3 :

$$\frac{c_M r^{\log_2 3}}{c_M (r/2)^{\log_2 3}} = 2^{\log_2 3} = 3.$$

Ceci concerne la partie multiplication, or la partie réduction modulaire reste quadratique (facteur de 4).

N.B.
Il existe un algorithme de multiplication FFT (fast Fourier transform — version discrète) avec complexité $O((\log n)^{1+\varepsilon})$.