

Cryptologie & Sécurité

Master 1 Informatique

Université Claude Bernard Lyon 1

Fabien LAGUILLAUMIE

`fabien.laguillaumie@ens-lyon.fr`

`http://perso.ens-lyon.fr/fabien.laguillaumie`



Cryptologie & Sécurité

Master 1 Informatique

Université Claude Bernard Lyon 1

Fabien LAGUILLAUMIE

`fabien.laguillaumie@ens-lyon.fr`

`http://perso.ens-lyon.fr/fabien.laguillaumie`



Introduction

Ordres de grandeur

Groupe, anneau, corps

Arithmétique

$\mathbb{Z}/n\mathbb{Z}$

Théorème des restes chinois

RSA

Arithmétique algorithmique

Factorisation

Méthodes de complexité exponentielle

Trial division

Méthode de Fermat

Méthode ρ de Pollard

Méthodes de complexité sous-exponentielle

Méthode de Dixon

Primalité

Générateurs pseudo-aléatoires

Générateurs pseudo-aléatoires congruentiels

Introduction



Cryptologie :

▸ Cryptographie :

- conception de systèmes cryptographiques
- étude (preuve) de leur sécurité
- implantation efficace et amélioration des performances ←

▸ Cryptanalyse :

- mise en défaut des systèmes cryptographiques
- attaque des problèmes algorithmiques sous-jacents ←
- observation des “canaux auxiliaires”

Algorithmique de base



Un **algorithme** est une procédure prenant en entrée un ensemble de valeurs et qui donne en sortie un ensemble de valeurs après une suite finie d'instructions élémentaires.

La **complexité** d'un algorithme est mesurée par

- ▶ le nombre d'opérations qu'il réalise
- ▶ la quantité de mémoire dont il a besoin

en fonction de la *taille* de son entrée.

Croissance des fonctions



Trois “grands” types de fonction :

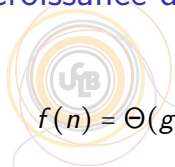
- ▶ $c \log(n)$ pour $c \in \mathbb{R}$ croissance lente
- ▶ cn^a pour $c \in \mathbb{R}$ et $a \in \mathbb{N}$ croissance rapide
- ▶ $ce^n = c \exp(n)$ pour $c \in \mathbb{R}$ croissance extrêmement rapide

Croissance des fonctions



n	10	100	1000
$\log(n)$	1	2	3
n^2	100	10000	10^6
n^{15}	10^{15}	10^{30}	10^{45}
$\exp(n)$	22026	2.7×10^{43}	1.97×10^{434}

Croissance des fonctions



$$f(n) = \Theta(g(n)) \iff \exists c_1, c_2, n_0 \in \mathbb{R} : 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ pour } n \geq n_0$$

$$f(n) = O(g(n)) \iff \exists c, n_0 \in \mathbb{R} : 0 \leq f(n) \leq c g(n) \text{ pour } n \geq n_0$$

$$f(n) = \Omega(g(n)) \iff \exists c, n_0 \in \mathbb{R} : 0 \leq c g(n) \leq f(n) \text{ pour } n \geq n_0$$

$$f(n) = o(g(n)) \iff \forall c \in \mathbb{R} \exists n_0 \in \mathbb{N} : 0 \leq f(n) < c g(n) \text{ pour } n \geq n_0$$
$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$

Croissance des fonctions



$$f(n) = O(g(n)) \iff \exists c, n_0 \in \mathbb{R} : 0 \leq f(n) \leq cg(n) \text{ pour } n \geq n_0$$

Exemples :

- ▶ $4n^2 + n + 1 = O(n^2)$
- ▶ $10 \log(n) + 5 \log^3(n) + 7n + 3n^2 + 6n^3 = O(n^3)$

Introduction

Quelques ordres de grandeur

- ▶ sécurité : $\geq 2^{80}$
- ▶ nombre d'atomes dans l'univers : $10^{80} \sim 2^{265}$
- ▶ taille d'un module RSA : 1024 bits $\sim 2^{1024} \sim 10^{310}$
- ▶ taille d'une clé AES : 256 bits
- ▶ Core 2 Quad (Penryn) - 3,2 GHz : $2 \times 24200 \text{ MIPS}^1$
 $1\,000\,000 \sim 2^{20}$
 2^{35} opérations en 1 seconde
Recherche exhaustive sur 2^{80} : $2^{80}/2^{35} = 2^{45}$ secondes
 $\leadsto 1114925$ années

[http://fr.wikipedia.org/wiki/Ordre_de_grandeur_\(nombres\)](http://fr.wikipedia.org/wiki/Ordre_de_grandeur_(nombres))

1. le nombre de millions d'instructions complétées par le microprocesseur en une seconde

Introduction

Quelques grandeurs

B. Schneier. Cryptographie appliquée.

Probabilité de mourir foudroyé (par jour)	1 chance sur 9 milliards (2^{33})
Probabilité de gagner le gros lot à la loterie américaine	1 chance sur 4 000 000 (2^{22})
Probabilité de gagner le gros lot à la loterie américaine et de mourir le même jour	1 chance sur 2^{61}
Probabilité d'être tué dans un accident automobile (aux États-Unis sur toute une vie)	1 chance sur 88 (2^7)
Âge de la Terre	10^9 années (2^{30})
Âge de l'Univers	10^{10} années (2^{34})
Nombre d'atomes constituant l'Univers	10^{77} (2^{265})

Un nombre de 1024 bit :

5858564308428828017644637396873011442410741924446401526444489392722401
2630691789482633773954538722685686046254635246206232275735158054157931
1060622915052999089708810050238601209069543816495749771173336617312655
2467966227874466635888109429506335487371428797711478405925439695590447
7668982192815149575434860493



Groupe Anneau Corps (finis)

Groupe



Un **groupe** (\mathbb{G}, \cdot) est un ensemble muni d'une opération \cdot satisfaisant :

1. associativité : $\forall a, b, c \in \mathbb{G} : (a \cdot b) \cdot c = a \cdot (b \cdot c)$
2. élément neutre : $\exists 1 \in \mathbb{G}, \forall a \in \mathbb{G} : a \cdot 1 = 1 \cdot a = a$
3. inverse : $\forall a \in \mathbb{G}, \exists a^{-1} \in \mathbb{G} : a \cdot a^{-1} = a^{-1} \cdot a = 1$

Anneau



Un **anneau** $(\mathbb{R}, +, \cdot)$ est un ensemble muni de deux opérations $+$ et \cdot tel que :

1. $(\mathbb{R}, +)$ est un groupe commutatif
2. (\mathbb{R}, \cdot) est un ensemble pour lequel \cdot est associative



Un **corps** $(\mathbb{F}, +, \cdot)$ est un ensemble muni de deux opérations $+$ et \cdot tel que :

1. $(\mathbb{F}, +)$ est un groupe commutatif d'élément neutre 0
2. $(\mathbb{F} \setminus \{0\}, \cdot)$ est un groupe commutatif d'élément neutre 1
3. $(\mathbb{F}, +, \cdot)$ satisfait une loi distributive :
$$\forall a, b, c \in \mathbb{F}, \quad a \cdot (b + c) = a \cdot b + a \cdot c$$



Arithmétique

Arithmétique élémentaire

$$\mathbb{N} = \{0, 1, 2, 3, \dots\}$$

$$\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$$

► $a, b \in \mathbb{Z}$:

$$a \mid b \iff \exists c \in \mathbb{Z} : b = ac$$

► Quelques propriétés :

1. $a \mid a$

2. si $a \mid b$ et $b \mid c$ alors $a \mid c$

3. si $a \mid b$ et $a \mid c$ alors $a \mid (bx + cy) \quad \forall x, y \in \mathbb{Z}$

4. si $a \mid b$ et $b \mid a$ alors $a = \pm b$

► Division euclidienne : $a, b \in \mathbb{Z}, b \geq 1$

$$\exists!(q, r) \in \mathbb{N}^2 : a = bq + r \text{ avec } 0 \leq r < b$$

Arithmétique élémentaire

Définition (pgcd)

Le **pgcd** de a et b est le plus grand entier qui divise à la fois a et b .

- ▶ $d = \text{pgcd}(a, b)$:
 $d \mid a$, $d \mid b$ et si $c \mid a$ et $c \mid b$ alors $c \mid d$.

Définition

a et b sont **premiers entre eux** si $\text{pgcd}(a, b) = 1$

Définition (Nombre premier)

Un entier $p \geq 2$ est **premier** si ses seuls diviseurs positifs sont 1 et p .

- ▶ $p \mid ab \implies p \mid a$ ou $p \mid b$
- ▶ il y a une infinité de nombres premiers
- ▶ y en a-t-il beaucoup ?
$$\pi(x) = \#\{p \text{ premier} \leq x\} \sim_{x \rightarrow \infty} x / \ln(x)$$



Théorème (Théorème fondamental de l'arithmétique)

Tout entier $n \geq 2$ admet une factorisation unique (à l'ordre près des facteurs) en produit de puissances de nombres premiers :

$$n = p_1^{e_1} p_2^{e_2} \dots p_\ell^{e_\ell}$$

avec les p_i distincts et $e_i > 0$ entiers.

(Preuve non-constructive...)

$\mathbb{Z}/n\mathbb{Z}$

$$\mathbb{Z}/n\mathbb{Z} = \{0, 1, 2, \dots, n-1\}$$

avec $+$ et \times " mod n "

$\mathbb{Z}/4\mathbb{Z}$:

\times	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

$\mathbb{Z}/5\mathbb{Z}$:

\times	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

- ▶ $(\mathbb{Z}/n\mathbb{Z}, +)$ est un groupe
- ▶ $(\mathbb{Z}/n\mathbb{Z}, +, \times)$ est un anneau
- ▶ $(\mathbb{Z}/p\mathbb{Z}, +, \times)$ est un *corps* si et seulement si p est premier
- ▶ $(\mathbb{Z}/n\mathbb{Z})^* = \{x \in \mathbb{Z}/n\mathbb{Z} : \exists x^{-1} \text{ tel que } xx^{-1} \equiv 1 \pmod{n}\}$

En particulier : $(\mathbb{Z}/p\mathbb{Z})^* = \mathbb{Z}/p\mathbb{Z} \setminus \{0\}$.

$(\mathbb{Z}/n\mathbb{Z})^*$ est l'ensemble des éléments de $\mathbb{Z}/n\mathbb{Z}$ *premiers* à n
(Bézout)

- ▶ $((\mathbb{Z}/n\mathbb{Z})^*, \times)$ est un groupe

$\mathbb{Z}/n\mathbb{Z}$

L'indicatrice d'Euler

Soit $n \geq 1$.

- ▶ $\varphi(n)$ représente le nombre d'entiers entre 1 et n qui sont premiers à n .

$$\varphi(n) = \#\{x \in [[1, n]] : \text{pgcd}(x, n) = 1\}$$

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

Propriétés :

- ▶ p premier $\implies \varphi(p) = p - 1$
- ▶ $\varphi(p^r) = (p - 1)p^{r-1}$
- ▶ φ est multiplicative : si $\text{pgcd}(m, n) = 1$

$$\varphi(mn) = \varphi(m)\varphi(n)$$

- ▶ $n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$

$$\varphi(n) = n \left(1 - \frac{1}{p_1}\right) \left(1 - \frac{1}{p_2}\right) \dots \left(1 - \frac{1}{p_k}\right)$$

Définition

Soit (\mathbb{G}, \times) un groupe d'ordre n ($= \#\mathbb{G}$). L'ordre d'un élément $g \in \mathbb{G}$ est le plus petit entier r tel que $g^r = 1$.

- ▶ $\mathbb{Z}/21\mathbb{Z} = \{0, 1, 2, \dots, 20\}$
- ▶ $\varphi(21) = \varphi(3 \times 7) = \varphi(3) \times \varphi(7) = 2 \times 6 = 12$
- ▶

$$(\mathbb{Z}/21\mathbb{Z})^* = \{1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20\}$$

$(\mathbb{Z}/21\mathbb{Z})^*$	{	1	2	4	5	8	10	11	13	16	17	19	20	}
ordre			6	3	6	2	6	6	2	3	6	6	2	

Définition

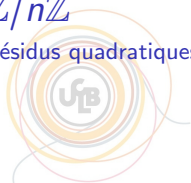
- ▶ Un entier a est appelé *résidu quadratique modulo n* si $\text{pgcd}(a, n) = 1$ et $a = x^2 \pmod{n}$ pour un certain entier x .
- ▶ x est une *racine carrée* de a modulo n .

$\mathbb{Z}/21\mathbb{Z}$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
carré	1	4	9	16	4	15	7	1	18	16	16	18	1	7	15	4	16	9	4	1

Les carrés de $\mathbb{Z}/21\mathbb{Z} = \{1, 4, 7, 9, 15, 16, 18\}$

$\mathbb{Z}/13\mathbb{Z}$	1	2	3	4	5	6	7	8	9	10	11	12
carré	1	4	9	3	12	10	10	12	3	9	4	1

Les carrés de $\mathbb{Z}/13\mathbb{Z} = \{1, 3, 4, 9, 10, 12\}$



Théorème

Soit $p > 2$ un nombre premier, le nombre de résidus quadratiques modulo p est $(p-1)/2$.

De plus, si x est une racine carrée de a modulo n , $-x$ l'est aussi.

Enfin, pour tout entier $a \neq 0 \pmod{p}$, $a^{(p-1)/2} = \pm 1 \pmod{p}$ et

$$a \text{ résidu quadratique} \iff a^{(p-1)/2} = 1 \pmod{p}.$$

$\mathbb{Z}/n\mathbb{Z}$

Logarithme discret

Si p est un nombre premier, \mathbb{Z}_p^* est un groupe cyclique d'ordre $p-1$: il existe un générateur $\gamma \in \mathbb{Z}_p^*$ tel que tout $\alpha \in \mathbb{Z}_p^*$ peut s'écrire

$$\alpha = \gamma^x \text{ pour un } 0 \leq x < p-1.$$

Définition

x s'appelle le *logarithme discret de α en base γ* : $x = d\log_\gamma(\alpha)$.

► $\mathbb{Z}/13\mathbb{Z}$

$$(\mathbb{Z}/13\mathbb{Z})^* = \mathbb{Z}/13\mathbb{Z} \setminus \{0\}$$

$$\varphi(13) = 12$$

$(\mathbb{Z}/13\mathbb{Z})^*$	=	1	2	3	4	5	6	7	8	9	10	11	12
ordre			12	3	6	4	12	12	4	3	6	12	2

► $\langle 6 \rangle = \{1, 6, 10, 8, 9, 2, 12, 7, 3, 5, 4, 11\}$

$$10 = 6^2 \pmod{13}$$

Soit m_1, \dots, m_r des entiers premiers entre eux 2 à 2, et a_1, \dots, a_r des entiers.

$$\begin{cases} x = a_1 \pmod{m_1} \\ x = a_2 \pmod{m_2} \\ \vdots \\ x = a_r \pmod{m_r} \end{cases}$$

Le théorème des restes chinois affirme qu'il existe une unique solution modulo $\prod_{i=1}^r m_i =: M$, donc que l'application π suivante est une bijection.

$$\begin{aligned} \pi : \mathbb{Z}/M\mathbb{Z} &\longrightarrow \mathbb{Z}/m_1\mathbb{Z} \times \dots \times \mathbb{Z}/m_r\mathbb{Z} \\ x &\longmapsto (x \bmod m_1, \dots, x \bmod m_r) \end{aligned}$$

► $\mathbb{Z}/15\mathbb{Z} \longrightarrow \mathbb{Z}/3\mathbb{Z} \times \mathbb{Z}/5\mathbb{Z}$

►

$\pi(0) = (0, 0)$	$\pi(5) = (2, 0)$	$\pi(10) = (1, 0)$
$\pi(1) = (1, 1)$	$\pi(6) = (0, 1)$	$\pi(11) = (2, 1)$
$\pi(2) = (2, 2)$	$\pi(7) = (1, 2)$	$\pi(12) = (0, 2)$
$\pi(3) = (0, 3)$	$\pi(8) = (2, 3)$	$\pi(13) = (1, 3)$
$\pi(4) = (1, 4)$	$\pi(9) = (0, 4)$	$\pi(14) = (2, 4)$

► En déduire la solution du système

$$\begin{cases} x &= 2 \pmod{3} \\ x &= 3 \pmod{5} \end{cases}$$



$$\begin{aligned}\pi^{-1} : \mathbb{Z}/m_1\mathbb{Z} \times \cdots \times \mathbb{Z}/m_r\mathbb{Z} &\longrightarrow \mathbb{Z}/M\mathbb{Z} \\ (a_1, \dots, a_r) &\longmapsto \sum_{i=1}^r a_i M_i y_i \pmod{M}\end{aligned}$$

avec

$$\begin{cases} M_i = M/m_i \\ y_i = M_i^{-1} \pmod{m_i} \end{cases}$$

- ▶ $\text{CRT}((a_1, \dots, a_r), (m_1, \dots, m_r), M)$
result = 0
for i from 1 to r do{
 Mi=M/m[i]
 yi= ModInv(Mi,m[i])
 result = result + a[i]*Mi*yi mod M
}
return result

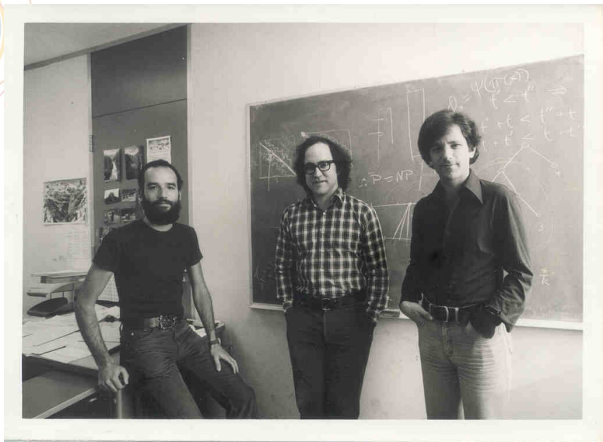
Difficile vs. facile



	facile	difficile
Calculer $N = p \times q$	✓	
Factoriser $N = p \times q$		✓
Résoudre $X^2 = a \pmod{p}$	✓	
Résoudre $X^2 = c \pmod{p \times q}$		✓
Résoudre $P(X) = 0 \pmod{p}$	✓	
Résoudre $P(X) = 0 \pmod{N}$		✓
Calculer $\text{dlog}(\cdot) \pmod{p}$		✓
Résoudre $X^n = a$ dans \mathbb{Z}	✓	
Trouver un générateur de \mathbb{Z}_p^*		✓

P est un polynôme

Le chiffrement RSA



A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. R. Rivest, A. Shamir, L. Adleman. Communications of the ACM, Vol. 21 (2), pp. 120–126 (1978)

Le chiffrement RSA



A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. R. Rivest, A. Shamir, L. Adleman. Communications of the ACM, Vol. 21 (2), pp. 120–126 (1978)



► Génération des clés :

Soit $k \in \mathbb{N}$ le *paramètre de sécurité*

- Construire 2 nombres premiers p et q tels que $2^{\lfloor k/2 \rfloor - 1} \leq p, q \leq 2^{\lfloor k/2 \rfloor} - 1$
- $N = p \times q$
- Choisir $e \in (\mathbb{Z}/\varphi(N)\mathbb{Z})^*$ et calculer d tel que

$$ed \equiv 1 \pmod{\varphi(N)}.$$

Primalité

Multiplication

Euclide étendu

clé publique	(N, e)
clé privée	(d, p, q)

► Chiffrement :

Un message m est un élément de $\mathbb{Z}/N_B\mathbb{Z}$.

- Alice obtient (N_B, e_B) .
- $c = m^{e_B} \pmod{N_B}$.

Exponentiation Modulaire

► Déchiffrement :

- Bob utilise sa clé secrète (d_B, p_B, q_B) .
- $c^{d_B} \pmod{N_B} = m$.

Exponentiation Modulaire

Certificate:

Data:

Version: 1 (0x0)

Serial Number: 7829 (0x1e95)

Signature Algorithm: md5WithRSAEncryption

Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
OU=Certification Services Division,
CN=Thawte Server CA/emailAddress=server-certs@thawte.com

Validity

Not Before: Jul 9 16:04:02 1998 GMT

Not After : Jul 9 16:04:02 1999 GMT

Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
OU=FreeSoft, CN=www.freesoft.org/emailAddress=baccala@freesoft.org

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public Key: (1024 bit)

Modulus (1024 bit):

00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
e8:35:1c:9e:27:52:7e:41:8f

Exponent: 65537 (0x10001)

Pourquoi ça marche ?

Théorème (Lagrange)

Si (\mathbb{G}, \cdot) est un groupe fini de cardinal n , alors $a^n = 1, \forall a \in \mathbb{G}$

Corollaire (Fermat)

p un entier premier, $a \in \mathbb{Z}$

$$\text{pgcd}(a, p) = 1 \implies a^{p-1} = 1 \pmod{p}$$

Corollaire (Euler)

$a, N \in \mathbb{Z}$

$$\text{pgcd}(a, N) = 1 \implies a^{\varphi(N)} = 1 \pmod{N}$$



► Pour déchiffrer $c \in \mathbb{Z}/N\mathbb{Z}$

$$m \equiv c^d \pmod{N}$$

En effet :

$$\begin{aligned} c^d \pmod{N} &\equiv m^{ed} \pmod{N} \\ &\equiv m^{1+k\varphi(N)} \pmod{N} \\ &\equiv m \times (m^{\varphi(N)})^k \pmod{N} \\ &\equiv m \end{aligned}$$

$$ed \equiv 1 \pmod{\varphi(N)} \iff \exists k \in \mathbb{Z} : ed = 1 + k\varphi(N)$$



Arithmétique algorithmique

Arithmétique algorithmique

Les opérations de bases

► $\text{Add}_2(a, b)$

$$(a = \sum_{i=0}^n a_i 2^i, b = \sum_{i=0}^n b_i 2^i)$$

$R[0] = 0$

for i from 0 to n

$c[i] = a[i] + b[i] + R[i]$

 if $c[i] \geq 2$

$c[i] = c[i] - 2$

$R[i+1] = 1$

$c[n+1] = R[n+1]$

return $c = \text{sum}(i=0, n+1, c[i] 2^i)$

Complexité : $O(n)$

Arithmétique algorithmique

Les opérations de bases



► $\text{Mul}_2(a, b)$

$$(a = \sum_{i=0}^n a_i 2^i, b = \sum_{i=0}^n b_i 2^i)$$

```
for i from 0 to n
    d[i] = a[i] * 2^i * b

return c = sum(i=0,n+1,d[i])
```

Complexité : $O(n^2)$

Arithmétique algorithmique

Les opérations de bases

Diviser pour régner

1. **Diviser** : Diviser le problème original en sous-problèmes plus faciles (plus petits).
2. **Régner** : Résoudre les petits sous-problèmes.
3. **Récolter** : combiner les solutions des petits sous-problèmes pour obtenir la solution au problème initial.

Arithmétique algorithmique

Les opérations de bases

Tentative n° 1 :

$$\begin{array}{l} a = \begin{array}{|c|c|} \hline a_L & a_R \\ \hline \end{array} = 2^{n/2} a_L + a_R \\ b = \begin{array}{|c|c|} \hline b_L & b_R \\ \hline \end{array} = 2^{n/2} b_L + b_R \end{array}$$

$$\begin{aligned} a \times b &= (2^{n/2} a_L + a_R) \times (2^{n/2} b_L + b_R) \\ &= 2^n a_L \times b_L + 2^{n/2} (a_L \times b_R + a_R \times b_L) + a_R \times b_R \end{aligned}$$

- ▶ nombre de produits d'entier de $n/2$ bits (\times) : 4
- ▶ nombre de shifts (de n ou $n/2$ bits) : 2
- ▶ nombre d'additions (d'entiers d'au plus $2n$ bits) : 3

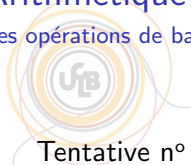
$T(n)$: temps de calcul de cet algorithme sur des entrées de n bits

$$T(n) = \begin{cases} \Theta(1) & \text{quand } n = 1 \\ 4T(n/2) + O(n) \end{cases} \rightsquigarrow T(n) = O(n^2)$$

(*master theorem* [Introduction to Algorithms - Cormen, Leiserson, Rivest, Stein])

Arithmétique algorithmique

Les opérations de bases



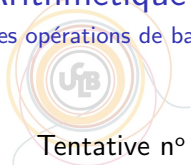
Tentative n° 2 : algorithme de Karatsuba ('60)

$$\begin{array}{l} a = \begin{array}{|c|c|} \hline a_L & a_R \\ \hline \end{array} = 2^{n/2} a_L + a_R \\ b = \begin{array}{|c|c|} \hline b_L & b_R \\ \hline \end{array} = 2^{n/2} b_L + b_R \end{array}$$

$$\begin{aligned} a \times b &= (2^{n/2} a_L + a_R) \times (2^{n/2} b_L + b_R) \\ &= 2^n a_L \times b_L + 2^{n/2} (a_L \times b_R + a_R \times b_L) + a_R \times b_R \\ &= 2^n a_L \times b_L + 2^{n/2} ((a_L + a_R) \times (b_R + b_L) - a_L \times b_L - a_R \times b_R) \\ &\quad + a_R \times b_R \end{aligned}$$

Arithmétique algorithmique

Les opérations de bases



Tentative n° 2 : algorithme de Karatsuba ('60)

- ▶ nombre de produits d'entier de $n/2$ bits (×) : 3 !
- ▶ nombre de shifts (de n ou $n/2$ bits) : 2
- ▶ nombre d'additions (d'entiers d'au plus $2n$ bits) : 3

$T(n)$: temps de calcul de cet algorithme sur des entrées de n bits

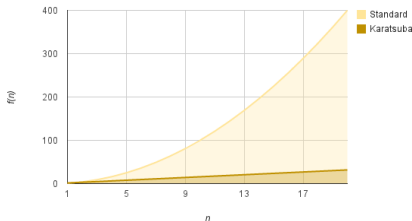
$$T(n) = \begin{cases} \Theta(1) & \text{quand } n = 1 \\ 3T(n/2) + O(n) \end{cases} \quad \leadsto \quad T(n) = O(n^{\log_2(3)}) = O(n^{1.59})$$

Arithmétique algorithmique

Les opérations de bases

Complexité des algorithmes de multiplications des entiers de n bits.

Algorithmes	Année	Complexité
classical		$O(n^2)$
Karatsuba	1960	$O(n^{1.59})$
Toom-Cook (Karatsuba généralisé)	1963-66	$O(n^{2\sqrt{2\log_2(n)}\log(n)})$
Schönhage-Strassen (FFT)	1971	$O(n\log(n)\log\log(n))$
Fürer (FFT)	2005	$n\log(n)2^{O(\log^*(n))}$



Arithmétique algorithmique

Les opérations de bases

► $\text{pgcd}(a, b)$

$$a \geq b > 0$$

On définit deux suites d'entiers $r_0, r_1, \dots, r_{\ell+1}$ et $q_1, q_2, \dots, q_{\ell}$

$$\begin{cases} r_0 &= a \\ r_1 &= b \end{cases}$$

$$r_0 = r_1 q_1 + r_2 \quad (0 < r_2 < r_1)$$

$$\vdots$$

$$r_{i-1} = r_i q_i + r_{i+1} \quad (0 < r_{i+1} < r_i)$$

$$\vdots$$

$$r_{\ell-2} = r_{\ell-1} q_{\ell-1} + r_{\ell} \quad (0 < r_{\ell} < r_{\ell-1})$$

$$r_{\ell-1} = r_{\ell} q_{\ell} \quad (r_{\ell+1} = 0)$$

On a $r_{\ell} = \text{pgcd}(a, b)$.

Arithmétique algorithmique

Les opérations de bases



- ▶ $\text{pgcd}(a, b)$

$$a \geq b > 0$$

```
while b != 0
    t = b
    b = a mod b
    a = t
return a
```

- ▶ Complexité $O(n^3)$?
- ▶ $O(n^2)$!

Arithmétique algorithmique

Les opérations de bases

Le nombre d'itérations de l'algorithme d'Euclide ℓ vérifie

$$\ell \leq \log b / \log \phi + 1 \quad (1)$$

où $\phi := (1 + \sqrt{5})/2 \approx 1,62$ est le nombre d'or.

1. ϕ vérifie l'équation $\phi^2 - \phi - 1 = 0$
2. Pour $i = 2, \dots, \ell - 1$,

$$r_{\ell-i} \geq r_{\ell-(i-1)} + r_{\ell-(i-2)}$$

3. Pour $i = 0, 1, \dots, \ell - 1$

$$r_{\ell-i} \geq \phi^i.$$

4. En posant $i = \ell - 1$ dans la relation précédente, on a bien (1)

Arithmétique algorithmique

Les opérations de bases

- ▶ La complexité de la division euclidienne de x par y est $O(l(y) \times l(q_{x,y}))$ où $q_{x,y}$ est le quotient de x par y
- ▶ La complexité de l'algorithme d'Euclide est donc au plus

$$\begin{aligned} & \sum_{i=1}^{\ell} (l(r_i) \times l(q_i)) \\ & \leq l(b) \sum_{i=1}^{\ell} l(q_i) \leq l(b) \sum_{i=1}^{\ell} (l(r_{i-1}) - l(r_i) + 1) \\ & \leq l(b)(l(r_0) - l(r_{\lambda}) + \lambda) \\ & \leq l(b)(l(a) + \log(b)/\log(\phi) + 1) = O(l(a)l(b)) \end{aligned}$$

et finalement

$$O(n^2)$$

Arithmétique algorithmique

Exponentiation modulaire

Un chiffrement RSA :

585856430842882801764463739687301144241074192444640
152644448939272240126306917894826337739545387226856
860462546352462062322757351580541579311060622915052
999089708810050238601209069543816495749771173336617
312655246796622787446663588810942950633548737142879
77114784059254396955904477668982192815149575434860493

48502682322937300170198050478318035172717126359264732
72917053460266783851110202318927412166518870818690786
36504254158873283040454382716615907535509571781887982
06108470451323948434443017954806012253949372682725356
49038632761226740908163973734546139018665542001670724
237928577415320139404040237390966904797167

mod

179769313486231590772930519078902473361797697894230657
273430081157732675805500963132708477322407536021120113
879871393357658789768814416622492849081450667704519456
266638519269269623133293482336582955895481722648601861
246662929361448136770863662699658175987684543951819223
123953830177176642598667396016540539057

Arithmétique algorithmique

Exponentiation modulaire



► $2^{16} = 2 \times 2 \times 2 \times \cdots \times 2 = 65536$

multiplications :15

► $2^2 = 4$

$$4^2 = 16$$

$$16^2 = 256$$

$$256^2 = 65536$$

multiplications : 4

Arithmétique algorithmique

Exponentiation modulaire

$$m^{11} \pmod{N} ?$$

- ▶ Méthode naïve :

$$m \times m \times m \times m \times m \times m \times m \times m \times m \times m \times m$$

multiplications : 10

- ▶ Méthode “square and multiply” :

$$11 = 2 \times 5 + 1 \times (2 \times 2 + 1) + 1$$

$$11_2 = 1011$$

$$\begin{aligned} m^{11} &= m^{(2 \times (2 \times 2 + 1) + 1)} \\ &= (m^2)^{(2 \times 2 + 1)} \times m \\ &= (m^2)^{2 \times 2} \times m^2 \times m \\ &= (((m^2)^2)^2) \times m^2 \times m \end{aligned}$$

multiplications : 6

Arithmétique algorithmique

Exponentiation modulaire



- ▶ $\text{ExpModN}(m, e, N)$

```
x=m
for i from 1 to e-1
    x = x*m mod N
return x
```

Complexité : $O(e \times \log(N)^2)$

Arithmétique algorithmique

Exponentiation modulaire



$$\begin{aligned}e &= e_0 + e_1 2 + e_2 2^2 + e_3 2^3 \dots + e_{t-1} 2^{t-1} + 2^t \\&= e_0 + 2(e_1 + e_2 2 + e_3 2^2 \dots + e_{t-1} 2^{t-2} + 2^{t-1}) \\&= e_0 + 2(e_1 + 2(e_2 + e_3 2 + \dots e_{t-1} 2^{t-3} + 2^{t-2})) \\&\vdots \\&= e_0 + 2(e_1 + 2(e_2 + 2(e_3 + \dots 2(e_{t-1} + 2))))\end{aligned}$$

$$\begin{aligned}m^e &= m^{e_0 + 2(e_1 + 2(e_2 + 2(e_3 + \dots 2(e_{t-1} + 2))))} \\&= m^{e_0} \left(m^{e_1} \left(m^{e_2} \left(\dots (m^{e_{t-1}} (m)^2)^2 \dots \right)^2 \right)^2 \right)^2\end{aligned}$$

Arithmétique algorithmique

Exponentiation modulaire



► ExpBinMod(m, e, N)

```
b=m
for i from t-1 to 0
    b = b2 mod N
    if (e[i] == 1) then
        b=b*m mod N
return b
```

Complexité : $O(\log(e) \times \log(N)^2)$

Arithmétique algorithmique

Aléa



Comment tirer uniformément des entiers dans un intervalle ?

Arithmétique algorithmique

Distribution uniforme dans $[0, B - 1]$

- ▶ Facile : générer uniformément dans $[0, 2^k - 1]$

- ▶ Générer uniformément dans $[0, B - 1]$? $2^{k-1} \leq B < 2^k$

Repeter

$s \leftarrow U([0, 2^k - 1])$

$n \leftarrow \text{Integer}(s)$

Jusqu'à $n < B$

Retourner n

$s \leftarrow U([0, 2^{k+t-1}])$

$n \leftarrow \text{Integer}(s) \bmod B$

Retourner n

Temps de calcul : $O(k)$

Distance statistique $\sim \frac{1}{2^t}$

Introduction

Comment attaquer RSA ?



Quel peut être le but de l'attaquant ?

- Retrouver la clé secrète
- Retrouver un message à partir de son chiffré

Cassage total

Sens unique

De quoi dispose un attaquant ?

- La clé publique
- La clé publique et un chiffré

Introduction

Comment attaquer RSA ?

Remarque :

- Casser le sens unique \leq casser totalement le système

Cassage total

Étant donné (N, e) , retrouver (d, p, q)

Factorisation

Sens unique

Étant donné c et (N, e) , retrouver m tel que $c = m^e \pmod{N}$

Problème RSA



Introduction

Factorisation



$$\begin{aligned} \blacktriangleright f: \mathcal{P}_k \times \mathcal{P}_k &\longrightarrow \{N = p \times q, \text{ avec } p, q \in \mathcal{P}_k\} \\ (p, q) &\longmapsto p \times q \end{aligned}$$

f est une fonction à sens unique

$$\begin{aligned} \blacktriangleright g: \mathbb{Z}/N\mathbb{Z} &\longrightarrow \mathbb{Z}/N\mathbb{Z} \\ x &\longmapsto x^e \pmod{N} \end{aligned}$$

g est une fonction à sens unique à *trappe*

On va maintenant s'intéresser à f .



Factorisation

Factorisation



Théorème (Théorème fondamental de l'arithmétique)

Tout entier $n \geq 2$ admet une factorisation unique (à l'ordre près des facteurs) en produit de puissances de nombres premiers :

$$n = p_1^{e_1} p_2^{e_2} \dots p_\ell^{e_\ell}$$

avec les p_i distincts et $e_i > 0$ entiers.

(Preuve non-constructive...)

Quelques records



RSA-200 = 27997833911221327870829467638722601621
07044678695542853756000992932612840010
76093456710529553608560618223519109513
65788637105954482006576775098580557613
57909873495014417886317894629518723786
9221823983
= 353246193440277012127260497819846436867
1197400197625023649303468776121253679
423200058547956528088349
× 792586995447833303334708584148005968773
7975857364219960734330341455767872818
152135381409304740185467

- ▶ factorisé avec GNFS (fin le 9 mai 2005)
- ▶ Next challenge : RSA-210
- ▶ record ECM : 67 chiffres décimaux
- ▶ <http://www.loria.fr/~zimmerma/records/factor.html>
(Google query : "Integer Factoring Records")

Quelques records

Email du 7 janvier 2010 :

We are pleased to announce the factorization of RSA768, the following 768-bit, 232-digit number from RSA's challenge list :

```
12301866845301177551304949583849627207728535695953347921973224521517264005
07263657518745202199786469389956474942774063845925192557326303453731548268
50791702612214291346167042921431160222124047927473779408066535141959745985
6902143413.
```

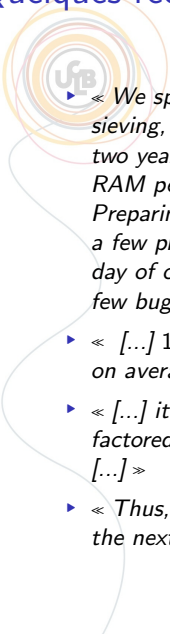
The factorization, found using the Number Field Sieve (NFS)the **Number Field Sieve** (NFS), is :

```
3347807169895689878604416984821269081770479498371376856891
2431388982883793878002287614711652531743087737814467999489
x
3674604366679959042824463379962795263227915816434308764267
6032283815739666511279233373417143396810270092798736308917
```

EPFL (Suisse), NTT (Japon), Univ. Bonn (Allemagne), INRIA (France), Microsoft (USA),

CWI (Pays-Bas)

Quelques records

- 
- ▶ « We spent half a year on 80 processors on polynomial selection. [...] the sieving, which was done on many hundreds of machines and took almost two years. On a single core 2.2 GHz AMD Opteron processor with 2 GB RAM per core, sieving would have taken about fifteen hundred years. [...] Preparing the sieving data for the matrix step took a couple of weeks on a few processors, the final step after the matrix step took less than half a day of computing, but took about four days of intensive labor because a few bugs had to be fixed.
 - ▶ « [...] $192796550 \times 192795550$ -matrix of total weight 27797115920 (thus, on average 144 non-zeros per row)[...] »
 - ▶ « [...] it is not unreasonable to expect that 1024-bit RSA moduli can be factored well within the next decade by an academic effort such as ours [...] »
 - ▶ « Thus, it would be prudent to phase out usage of 1024-bit RSA within the next three to four years »

Méthodes de complexité exponentielle

Trial Division

Soit N l'entier à factoriser.

▶ TrialDivision(N)

```
i:=1
rem:=1
while rem != 0 do {
    i:= nextprime(i+1)
    rem := N mod i
}
return i
```

Complexité :

$$\mathcal{O}(\sqrt{N}/\log(N))$$

Méthodes de complexité exponentielle

Méthode de Fermat

► Exercice : factoriser 8051

► Si $N = pq$ est impair, $N = ((p + q)/2)^2 - ((p - q)/2)^2$

► Fermat(N)

```
for a from ceil(sqrt(N)) to N do {  
    b2 = a*a - N  
    if issquare(b) then return a-sqrt(b2)  
}  
return ‘‘N est premier’’
```

Complexité : (p et q proche de \sqrt{N})

$$\mathcal{O}\left(\frac{(p - q)^2}{\sqrt{N}}\right)$$

Méthodes de complexité exponentielle

Méthode ρ de Pollard

- ▶ Considérons une fonction $f : \mathbb{Z}/N\mathbb{Z} \longrightarrow \mathbb{Z}/N\mathbb{Z}$ “aléatoire”.
- ▶ Soit s un élément quelconque de $\mathbb{Z}/N\mathbb{Z}$
- ▶ Construisons la suite

$$\begin{array}{ccccccc} s, & f(s), & f(f(s)), & f(f(f(s))), & f^{(4)}(s), & \dots \\ a_0 & a_1 & a_2 & a_3 & a_4 & \dots \end{array}$$

Cette suite est périodique

et donc : $\exists j, k \in \mathbb{N}, j < k$ tel que

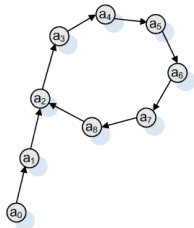
$$f^{(j)}(s) \equiv f^{(k)}(s)$$

soit

$$a_j = a_k \pmod{N}$$

Méthodes de complexité exponentielle

Méthode ρ de Pollard



- ▶ Exemple : $f(x) = x^2 + 1 \pmod{20}$
3, 10, 1, 2, 5, 6, 20, 17, 10, 1, 2, ...
- ▶ Exemple : $f(x) = x^2 + 1 \pmod{521}$
32, 504, 290, 220, 469, 100, 102, 506, 226, 19, 362, 274, 53, 205, 346, 408, 266, 422, 424, 32, 504, ...
- ▶ Remarque : entre les deux $32 \rightsquigarrow 18$ éléments
 $\sqrt{521} \simeq 22,8$.

Apparté : le paradoxe des anniversaires



- ▶ Question : Combien doit-il y avoir de personnes dans une pièce pour qu'il y ait au moins 1 chance sur 2 que deux personnes soient nées le même jour ?

- ▶ Application :

$$\ell_q + \ell_c \simeq \sqrt{p}.$$

- ▶ Application à la factorisation : méthode ρ de Pollard

Méthodes de complexité exponentielle

Méthode ρ de Pollard

Soit N l'entier à factoriser et $p \mid N$.

On considère les fonctions

- ▶ Soit $f : \mathbb{Z}/p\mathbb{Z} \longrightarrow \mathbb{Z}/p\mathbb{Z}$
 $x \longmapsto x^2 + 1 \pmod{p}$
- ▶ Soit $F : \mathbb{Z}/N\mathbb{Z} \longrightarrow \mathbb{Z}/N\mathbb{Z}$
 $x \longmapsto x^2 + 1 \pmod{N}$

La suite

$$\left(f^{(i)}(s) \right)_{i \geq 0}$$

va cycler au bout de \sqrt{p} itérations²

2. d'après le paradoxe des anniversaires

Méthodes de complexité exponentielle

Méthode ρ de Pollard

En d'autres termes :

$\exists j, k \in \mathbb{N}, j < k$ tel que

$$f^{(j)}(s) = f^{(k)}(s)$$

soit

$$a_j = a_k \pmod{p}$$

soit encore

$$p \mid a_j - a_k.$$

Problème : on ne connaît pas p ...

Méthodes de complexité exponentielle

Méthode ρ de Pollard



Mais...

comme $p \mid N$,

$$f(x) = F(x) \pmod{p}$$

$$N = 11 \times 17 = 187, \quad p = 11$$

$$\begin{aligned} f(10) &= 10^2 + 1 \pmod{11} \equiv 2 \pmod{11} \\ F(10) &= 10^2 + 1 \pmod{187} \equiv 101 \pmod{187} \end{aligned}$$

$$\begin{aligned} F(10) \pmod{11} &\equiv 101 \pmod{11} \\ &\equiv 2 \pmod{11} \end{aligned}$$

Méthodes de complexité exponentielle

Méthode ρ de Pollard

$\exists j, k \in \mathbb{N}, j < k$ tel que

$$f^{(j)}(s) = f^{(k)}(s)$$

donc

$\exists j, k \in \mathbb{N}, j < k$ tel que

$$F^{(j)}(s) \equiv F^{(k)}(s) \pmod{p}$$

ce qui signifie que

$$p \mid F^{(j)}(s) - F^{(k)}(s)$$

On peut alors extraire p :

$$p = \text{pgcd}(F^{(j)}(s) - F^{(k)}(s), N)$$

(sauf si ?)

Méthodes de complexité exponentielle

Méthode ρ de Pollard



- ▶ le cycle va se produire au bout de $\mathcal{O}(\sqrt{p})$
- ▶ comment le détecter ?
 - naïf : beaucoup d'espace mémoire, beaucoup de comparaisons
- ▶ détection de cycle de Floyd
 - idée : \leadsto 2 suites, l'une allant deux fois plus vite que l'autre

Méthodes de complexité exponentielle

Méthode ρ de Pollard

Détection de cycle de Floyd :

Posons $\ell = k - j$, alors, pour tout $m \geq j$

$$F^{(m)}(s) \equiv F^{(m+\ell)}(s) \equiv F^{(m+2\ell)}(s) \equiv \dots \pmod{p}$$

On considère alors le premier multiple n de ℓ qui excède j :

$$F^{(n)}(s) \equiv F^{(2n)}(s) \pmod{p}$$

et

$$n \leq k \simeq \sqrt{p}$$

Méthodes de complexité exponentielle

Méthode ρ de Pollard

► PollardRho(N)

```
s = random(N-1)
U = s
V = s
g = 1
while g == 1 do {
    U = U^2 + 1 [N]
    V = V^2 + 1 [N]
    V = V^2 + 1 [N]
    g = pgcd(U-V,N)
}
return g
```

Complexité :

$\mathcal{O}(\sqrt{p})$ donc $\mathcal{O}(N^{1/4})$

Méthodes de complexité exponentielle

Méthode ρ de Pollard

Exemple : $N = 1537$

$U = V = 11$

► $U \equiv 122 \pmod{1537}$

$V \equiv 1052 \pmod{1537}$

$\text{pgcd}(-930, 1537) = 1$

► $U \equiv 1052 \pmod{1537}$

$V \equiv 1152 \pmod{1537}$

$\text{pgcd}(-100, 1537) = 1$

► $U \equiv 65 \pmod{1537}$

$V \equiv 862 \pmod{1537}$

$\text{pgcd}(-797, 1537) = 1$

► $U \equiv 1152 \pmod{1537}$

$V \equiv 862 \pmod{1537}$

$\text{pgcd}(290, 1537) = 29$

$$U_1$$

$$V_1 = U_2$$

$$U_2$$

$$V_2 = U_4$$

$$U_3$$

$$V_3 = U_6$$

$$U_4$$

$$V_4 = U_8$$

Méthodes de complexité exponentielle

Conclusion



- ▶ Ces méthodes ont toutes une complexité exponentielle
- ▶ Une version améliorée de ρ a permis à R. Brent et J. Pollard de factoriser

$$F8 = 2^{2^8} + 1 = 2^{256} + 1$$

- ▶ Aucune chance de factoriser un entier de 2048 bits



Méthodes de complexité sous-exponentielle

Méthodes de complexité sous-exponentielle

Un principe universel

Pour factoriser N , on recherche deux entiers x et y tels que

$$x^2 \equiv y^2 \pmod{N} \quad \text{et} \quad x \not\equiv \pm y \pmod{N}$$

Alors

$$(x - y)(x + y) \equiv 0 \pmod{N}$$

i.e. $x - y$ et $x + y$ sont des diviseurs de 0 dans $\mathbb{Z}/N\mathbb{Z}$

Exemple :

$$505^2 \equiv 16^2 \pmod{84923}$$

$$(505 - 16)(505 + 16) \equiv 0 \pmod{84923}$$

$$\text{pgcd}(505 - 16, 84923) = 163$$

$$\text{pgcd}(505 + 16, 84923) = 521$$

Méthodes de complexité sous-exponentielle

La méthode de Dixon

► $N = 15770708441$

► On considère $\mathcal{B} = \{2, 3, 5, 7, 11, 13\}$

$$B = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13$$

► Tirages uniformes mod N :

$$x = 7038304916 \rightsquigarrow x^2 \pmod{N} = 7862350068 \rightsquigarrow \gcd(7862350068, B) = 36$$



$$x = 9983710959 \rightsquigarrow x^2 \pmod{N} = 10448531956 \rightsquigarrow \gcd(10448531956, B) = 4$$



$$x = 12894763099 \rightsquigarrow x^2 \pmod{N} = 3417563323 \rightsquigarrow \gcd(3417563323, B) = 1$$



$$x = 12239785937 \rightsquigarrow x^2 \pmod{N} = 12834523175 \rightsquigarrow \gcd(12834523175, B) = 25$$



$$x = 14885085472 \rightsquigarrow x^2 \pmod{N} = 9621807146 \rightsquigarrow \gcd(9621807146, B) = 14$$



$$x = 11978089837 \rightsquigarrow x^2 \pmod{N} = 2749230463 \rightsquigarrow \gcd(2749230463, B) = 7$$



$$x = 8340934156 \rightsquigarrow x^2 \pmod{N} = 21 \rightsquigarrow \gcd(21, B) = 21$$



$$x = 9161878375 \rightsquigarrow x^2 \pmod{N} = 6747817174 \rightsquigarrow \gcd(6747817174, B) = 14$$



$$x = 12044942944 \rightsquigarrow x^2 \pmod{N} = 78 \rightsquigarrow \gcd(78, B) = 78$$



$$x = 9322349340 \rightsquigarrow x^2 \pmod{N} = 6646772716 \rightsquigarrow \gcd(6646772716, B) = 4$$



⋮

$$x = 2773700011 \rightsquigarrow x^2 \pmod{N} = 182 \rightsquigarrow \gcd(182, B) = 182$$



Méthodes de complexité sous-exponentielle

La méthode de Dixon

► $N = 15770708441$

► On considère $\mathcal{B} = \{2, 3, 5, 7, 11, 13\}$

► Soit
$$\begin{cases} x_1 &= 8340934156 \\ x_2 &= 12044942944 \\ x_3 &= 2773700011 \end{cases}$$

► Alors
$$\begin{cases} x_1^2 &\equiv 21 \pmod{N} \\ x_2^2 &\equiv 78 \pmod{N} \\ x_3^2 &\equiv 182 \pmod{N} \end{cases}$$

►
$$\begin{cases} x_1^2 &\equiv 3 \times 7 \pmod{N} \\ x_2^2 &\equiv 2 \times 3 \times 13 \pmod{N} \\ x_3^2 &\equiv 2 \times 7 \times 13 \pmod{N} \end{cases}$$

d'où

$$(x_1 x_2 x_3)^2 \equiv (2 \times 3 \times 7 \times 13)^2 \pmod{N}$$



Méthodes de complexité sous-exponentielle

La méthode de Dixon

- ▶ et en réduisant $\bmod N$:

$$9503435785^2 \equiv 546^2 \pmod{N}$$

- ▶ on obtient un facteur non-trivial :

$$\text{pgcd}(9503435785 - 546, 15770708441) = 115759$$

Beaucoup de ☹️, peu de 😊

Méthodes de complexité sous-exponentielle

La méthode de Dixon

Encore un exemple : $N = 143$

$$\mathcal{B} = \{2, 3, 5, 7, 11, 13, 17, 19\}$$

x_i	$x_i^2 \pmod{N}$	factorisation
114	126	☹
137	36	☹
115	69	☹
17	3	3
132	121	☹
18	38	$2 \cdot 19$
90	92	☹
19	75	$3 \cdot 5^2$
37	82	☹
20	114	$2 \cdot 3 \cdot 19$

Considérons la matrice *binaire* :

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Méthodes de complexité sous-exponentielle

La méthode de Dixon

Une solution du système linéaire :

$$(x \ y \ z \ t) \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} = (0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0) \pmod{2}$$

$$(x \ y \ z \ t) = (1 \ 0 \ 1 \ 0)$$

$\leadsto 17, 19$

$$\begin{aligned} (17 \cdot 19)^2 &\equiv (3 \cdot 5)^2 \pmod{143} \\ 37^2 &\equiv 15^2 \pmod{143} \end{aligned}$$

$$\text{pgcd}(37 - 15, 143) = \text{pgcd}(22, 143) = 11$$

Méthodes de complexité sous-exponentielle

La méthode de Dixon

- ▶ $\text{Dixon}(N, B)$
 - ▶ // Constitution de la base de factorisation

```
p=2
FactorBase = [p]
while p <= B do{
    p = nextprime(p)
    FactorBase = [FactorBase,p]
}
```

$\mathcal{B} = \{p_1, p_2, \dots, p_b\}$ avec $p_1 = 2$, $p_2 = 3$, ... et $p_b \leq B$

Méthodes de complexité sous-exponentielle

La méthode de Dixon

► Dixon(N, B)

► // Etape 1 : les relations

```
M = []
X = []
i = 0
while i <= b+1 do{
    x = random(N)
    a = x^2 mod N
    (b,e) = TrialDivisionModified(a)
    if b do{
        AjouterMatrice(M,e)
        X=[X,x]
    }
}
```


Méthodes de complexité sous-exponentielle

La méthode de Dixon

où

- ▶ **TrialDivisionModified(a)**

renvoie un bit b et un vecteur $e = (e_1, e_2, \dots, e_b)$ tels que

- ▶ b vaut 1 si a se factorise sur \mathcal{B}

- ▶ $a = p_1^{e_1} p_2^{e_2} \dots p_b^{e_b}$

- ▶ **AjouterMatrice(M,e)**

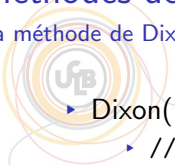
ajoute la ligne formée de

$$(e_1 \pmod{2}, e_2 \pmod{2}, \dots, e_b \pmod{2})$$

à la matrice M .

Méthodes de complexité sous-exponentielle

La méthode de Dixon



```
► Dixon(N, B)
  ► // Etape 2 : algèbre linéaire
    Soluce = LinSolve(M,0)
    Somme = vector(b,0)
    For i from 1 to b+1 do {
      if Soluce[i] == 1 do {
        Somme = Somme + M[i]
      }
    }
```

$$\mathcal{I} = \{i \in [[1, b]] : \text{Solu}ce[i] = 1\}$$

$$\begin{aligned} \text{Somme} &= \sum_{i \in \mathcal{I}} \left(e_1^{(i)}, e_2^{(i)}, \dots, e_b^{(i)} \right) = (E_1, E_2, \dots, E_b) \\ &\equiv (0, 0, \dots, 0) \pmod{2} \end{aligned}$$

$$\text{où } E_j = \sum_{i \in \mathcal{I}} e_j^{(i)}$$

Méthodes de complexité sous-exponentielle

La méthode de Dixon

► Dixon(N, B)

► // pgcd final

x=1

y=1

// calcul de x

For i from 1 to b+1 do {

 if Soluce[i] == 1 do {

 x=x*X[i] mod N

 }

// calcul de y

For i from 1 to b do {

 y=y*FactorBase[i]^(Solute[i]/2) mod N

}

return pgcd(x+y,N),pgcd(x-y,N)

$$x = \prod_{i \in \mathcal{I}} x_i \quad \text{et} \quad y = \prod_{i=1}^b p_i^{E_i/2} \quad \text{avec} \quad x^2 \equiv y^2 \pmod{N}$$

Méthodes de complexité sous-exponentielle

La méthode de Dixon

Complexité : (un peu de théorie...)

Définition

Soit $t \in]0, 1[$ et $c \in \mathbb{R}$. On définit la fonction $L_{t,c} : \mathbb{R}^+ \longrightarrow \mathbb{R}^+$ par

$$L_{t,c}(N) = e^{c(\ln N)^t (\ln \ln N)^{1-t}}$$

Remarque :

- ▶ $t = 0 : L_{0,c}(N) = e^{c(\ln \ln N)} = (\ln N)^c$
- ▶ $t = 1 : L_{1,c}(N) = e^{c(\ln N)}$

polynomiale
exponentielle

Méthodes de complexité sous-exponentielle

La méthode de Dixon



Complexité : (un peu de théorie...)

Définition

Soit $B \in \mathbb{N}$. Un entier $N \in \mathbb{N}$ est dit *B -friable* si tout diviseur p de N vérifie $p \leq B$.

Exemples :

- ▶ $1470 = 2 \cdot 3 \cdot 5 \cdot 7^2$ est 7-friable
- ▶ 1471 est ... premier
- ▶ $1473 = 3 \cdot 491$ est 491-friable

Méthodes de complexité sous-exponentielle

La méthode de Dixon



Pour simplifier, $L(N) = L_{1/2,1}(N) = e^{\sqrt{\ln N \ln \ln N}}$.

- ▶ On pose $B = L(N)^a$ pour un certain $a \in \mathbb{R}$.
- ▶ Combien de nombres premiers plus petits que B ?

On note $\pi(x) = \#\{n \leq x : n \text{ est premier}\}$

On a

$$\pi(x) \sim x / \ln x$$

\leadsto taille de \mathcal{B} : $\pi(L(N)^a) \sim L(N)^a$

Méthodes de complexité sous-exponentielle

La méthode de Dixon

- ▶ On note $\psi(x, y) = \#\{n \leq x, n \text{ est } y\text{-friable}\}$

On a

$$\psi(x, L(x)^a)/x \sim L(x)^{-1/2a}$$

Donc la probabilité qu'un entier tiré au hasard dans $[[1, N - 1]]$ soit B -friable est $L(N)^{-1/2a}$.

- ▶ Ainsi, pour que le système ait une unique solution, on doit obtenir $L(N)^a$ relations
- ▶ et donc faire de l'ordre de

$$L(N)^a \times L(N)^{1/2a} = L(N)^{a+1/2a}$$

essais

Méthodes de complexité sous-exponentielle

La méthode de Dixon



- ▶ Coût d'extraction des facteurs : $L(N)^a$
- ▶ Donc coût de la 1ère phase : $L(N)^{2a+1/2a}$
- ▶ Coût de la résolution d'un système linéaire (donc de la 2ème phase) : $L(N)^{3a}$
- ▶ Coût total :

$$L(N)^{\max(3a, 2a+1/2a)} = L(N)^2$$

Méthodes de complexité sous-exponentielle

- ▶ Crible quadratique (QS)

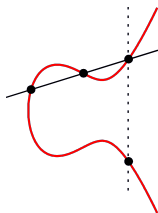
$$L(1/2, 1)(N) = e^{\sqrt{\ln N \ln \ln N}}$$

- ▶ Cribles dans des corps de nombres : NFS (GNFS/SNFS)

$$L(1/3, 64/9^{1/3}) = e^{(64/9)^{1/3} (\ln N)^{1/3} (\ln \ln N)^{2/3}}$$

- ▶ Elliptic Curve Method (ECM)

$$L(1/2, \sqrt{2})(p) = e^{\sqrt{2} \sqrt{\ln p \ln \ln p}}$$





Primalité

Primalité

Théorème (Fermat)

p un entier *premier*, $a \in \mathbb{Z}$

$$\text{pgcd}(a, p) = 1 \implies a^{p-1} = 1 \pmod{p}$$

- ▶ Test de composition avec $a = 2$:

$$x = 2^{N-1} \pmod{N}$$

$$N \text{ premier} \implies x = 1$$

$$x \neq 1 \implies N \text{ non premier}$$

Condition *nécessaire* de primalité.

$$N = 341 \rightsquigarrow 2^{340} = 1 \pmod{341}$$

$$341 = 13 \times 11$$

Remarque : on peut remplacer 2 par a tel que $\text{pgcd}(a, N) = 1$

Primalité

Il existe des nombres entiers composés N , appelés *nombres de Carmichael* qui vérifient :

$$\forall a \in \mathbb{N}, \text{pgcd}(a, N) = 1 : a^{N-1} = 1 \pmod{N}$$

$$561 = 3 \times 11 \times 17$$

$$1105 = 5 \times 13 \times 17$$

$$1729 = 7 \times 13 \times 19$$

$$2465 = 5 \times 17 \times 29$$

$$2821 = 7 \times 13 \times 31$$

$$6601 = 7 \times 23 \times 41$$

$$8911 = 7 \times 19 \times 67$$

Définition

Soit a un entier. Un entier N **composé** est dit pseudo-premier en base a s'il vérifie

$$a^{N-1} = 1 \pmod{N} \quad (2)$$

► Test de Fermat(N, t)

```
for i from 1 to t {  
    a = random(N)  
    if TestCond1(N,a) == false return(composé)  
}  
return(probablement premier)
```

Primalité

Soit N impair : $N - 1 = 2^s u$ avec $s \in \mathbb{N}$ et u impair.

$$a^{N-1} - 1 = (a^u - 1)(a^u + 1)(a^{2u} + 1) \dots (a^{2^{s-1}u} + 1)$$

Si N est premier (et a non multiple de N) :

$$N \mid a^{N-1} - 1$$

donc il divise l'un des facteurs.

$$a^u \equiv 1 \pmod{N} \quad \text{ou} \quad \exists \quad 0 \leq j \leq s : a^{2^j u} \equiv -1 \pmod{N} \quad (3)$$

Réciproque fausse...

$$N = 2047 = 23 \times 89 \quad N - 1 = 2 \times 1023 \quad \text{et} \quad 2^{(N-1)/2} \equiv 1 \pmod{N}$$

Primalité

Définition

Un nombre impair *composé* N qui passe le test 3 est appelé *pseudo-premier fort en base a* .

- Génération de nombre premier(k, d)

```
b = false
Repeat {
    N = random_b(k)
    for i from 1 to d {
        a = random(N)
        b = b & TestCond2(N,a)
    } until b == true
return N
```



$p_{k,d}$: probabilité que cet algorithme retourne un entier composé :

$\forall d \geq 1, k \geq 2,$

$$p_{k,d} \leq \frac{1}{4^d}$$

PRIMES is in P



Générateurs pseudo-aléatoires

Générateurs pseudo-aléatoires congruentiels

- ▶ Classe de générateurs définie par

$$\mathcal{G}(m, a, c, X_0) : X_{n+1} \equiv aX_n + c \pmod{m}$$

- ▶ Exemple :

- ▶ $X_{n+1} \equiv 25X_n + 16 \pmod{256}$

- ▶ $X_0 = 10 \rightsquigarrow 10, 10, 10, 10, 10, 10, \dots$

- ▶ $X_0 = 12 \rightsquigarrow 12, 60, 236, 28, 204, 252, 172, 220, 140, 188, 108, 156, 76, 124, 44 \dots$

▶ Théorème

Le générateur $\mathcal{G}(m, a, c, X_0)$ a pour période m si et seulement si

1. $\text{pgcd}(c, m) = 1$,
2. $b = a - 1$ est multiple de tout nombre premier qui divise m
3. b est multiple de 4 si m est multiple de 4

Générateurs pseudo-aléatoires congruentiels

- ▶ Classe de générateurs définie par

$$\mathcal{G}(m, a, c, X_0) : X_{n+1} \equiv aX_n + c \pmod{m}$$

- ▶ Exemple :

- ▶ Le générateur d'UNIX (dont le comité ANSI C a heureusement recommandé de ne conserver que les 16 bits de poids fort) :

$$X_{n+1} \equiv 1103515245X_n + 12345 \pmod{2^{31}}$$

- ▶ Le générateur de Blum, Blum, Shup :

$$N = pq, \quad p, q \equiv 2 \pmod{3}$$

$$X_{n+1} \equiv X_n^2 \pmod{N}$$

$$Z_{n+1} \equiv X_{n+1} \pmod{2}$$