

Algorithmique de la Théorie des nombres

Jean-Guillaume Dumas
2004

UNIVERSITÉ JOSEPH FOURIER
CENTRE DE RECHERCHE MATHÉMATIQUES

Laboratoire de
Modélisation et Calcul



Plan

- Introduction : calculs avec des entiers
Ordres de grandeur, complexité, vitesses pratiques
- Éléments d'arithmétique pratique
 - Prétexte : Codage RSA
 - 1. Théorèmes d'Euler, Fermat, Chinois, RSA
 - 2. Algorithmes d'Euclide étendu, puissance par carrés
 - 3. Tests de Primalité
 - 4. Factorisation d'entiers
- Construction des corps finis
 - Prétexte : Échange de clés Diffie-Hellman
 - 1. Logarithme discret
 - 2. Racines primitives et petits corps finis
 - 3. Construction des corps finis
 - 4. Cassage sous exponentiel

Calculs avec des entiers

Ordres de grandeur, complexité, vitesses pratiques

Ordres de grandeur

- Taille d'un entier n : $t_n = \lceil \log_2(n) \rceil$ bits $\rightarrow n = O(\exp(t_n))$
 - 128 bits = 39 chiffres, 512 bits = 155 chiffres, 1024 bits = 309 chiffres
- Vitesse des ordinateurs
 - 1 GHz = 10^9 secondes
 - Lumière du projecteur à la salle : 3m à 300000 km/s = 10^8 s
 - Ce portable fait donc 10 additions le temps que la lumière arrive du projecteur !
- Age de l'univers : 15 milliards * 365 * 24 * 3600 $\approx 5.10^{17}$ s
- Nombre d'électrons dans l'univers : $10^{61} * 10^{11} * 10^{12} \approx 10^{84}$
- Complexité d'un algorithme pour des nombres de 128 bits
 - $t_n = 128$ opérations, $t_n^2 = 16384$ ops, $t_n^3 = 2.10^6$ ops, $t_n^4 = 3.10^8$ ops
 - $n = 10^{89}$ opérations \rightarrow 2 millions de fois l'âge de l'univers sur un million de PC à 1GHz

Calcul sur les entiers

- 32 (ou 64) bits : codent entiers de 0 à $2^{32}-1$
 - addition (=1 cycle)
 - soustraction (=1 cycle)
 - multiplication (=1 cycle)
 - division (=10 cycles)
- Pour 1 GHz \approx 1 milliard d'additions/multiplications à la seconde

Entiers en précision arbitraire

- Ex: C/C++ Gnu Multiprecision Package
 - Liste de mots non signés de 32 bits, plus le signe.
 - Addition/soustraction d'entiers bornés par n : $O(t_n)$ opérations
 - Multiplication :
 - $O(t_n^2)$ opérations (méthode classique)
 - Karatsuba $O(t_n^{1.585})$
 - Toom-Cook $O(t_n^{1.465})$
 - Schönhage & Strassen $O(t_n \log(t_n) \log \log(t_n))$.
 - \rightarrow En pratique : classique jusqu'à 32 mots (=300 chiffres), Karatsuba jusqu'à 256 mots, première FFT à partir de 10000 mots.
 - Division : cf. multiplication, + divisions sur 32 bits

Arithmétique classique

- MAops = million d'opérations arithmétiques / seconde
→ Sur un Pentium IV, 2.4GHz, max ≈ 2400 MAops
- Mesures : comptent accès mémoire, copie, affectation, opération, ...
- Sur 32 bits machine (long int):
 - Addition, Soustraction, Multiplication : 365 MAops
 - Négation : 915 MAops
 - AXPYIN (r += a × b) : 711 MAops
 - Division : 81 MAops
- Avec GMP,

	32 bits	300 bits	3000 bits
- Addition, Soustraction :	2.2	1.7	0.64
- Multiplication :	2.2	0.6	0.017
- Négation :	4.4	2.6	0.85
- AXPYIN (r += a × b) :	4.5	1.1	0.035
- Division :	2.2	0.7	0.03

Entiers modulaires

- Notation : $a \equiv b [m]$, (a est congru à b modulo m)
si (a-b) est divisible par m. ex: $11 \equiv 5 [6]$
- $\mathbb{Z}/n\mathbb{Z} = \{0, 1, \dots, n-1\}$
- Opérations modulaires classiques :
 - Addition/Soustraction : addition d'entiers + décalage de m
 - $r = a + b$
 - Si (r > m) Alors $r = r - m$
 - Multiplication : multiplication d'entiers + division d'entiers
 - $r = a \times b$
 - $r = r \% m$ // Calcul du reste de la division de r par m
 - Division : pgcd + division d'entiers quand valide

Algorithme d'Euclide étendu

- Bézout : a et b premiers, $\exists (u,v), a u + b v = 1 = \text{pgcd}(a,b)$
- Algorithme d'Euclide pour le pgcd :
 - $a = b q_1 + r_1; b = r_1 q_2 + r_2; r_1 = r_2 q_3 + r_3 \dots$
 - Algorithme d'Euclide étendu, calcule aussi u et v
 1. $1 a + 0 b = a$
 2. $0 a + 1 b = b$
 → Appliquer le pgcd sur 1. et 2.
- Complexité classique : $O(t_n^2)$, ... FFT en $O(t_n \log(t_n)^2)$
- Application : calcul de la division dans un corps premier :
 - $a u \equiv 1 [p]$, i.e. u est l'inverse de a modulo b !
 - Corollaire : $\mathbb{Z}/p\mathbb{Z}$ est un corps pour p premier (tout non nul plus petit que p est inversible, puisque premier avec p)

Arithmétique MODULAIRE classique

- MAops = million d'opérations arithmétiques / seconde
→ Sur un Pentium IV, 2.4GHz, max ≈ 2400 MAops
- Mesures : comptent accès mémoire, copie, affectation, opération, ...
- Sur 16 bits machine (long int):
 - Addition, Soustraction : 300 MAops
 - Multiplication : 68 MAops
 - Négation : 475 MAops
 - AXPYIN (r += a × b) : 135 MAops
 - Division (Bézout) : 1.6 MAops
- Avec GMP,

	32 bits	300 bits	3000 bits
- Addition, Soustraction :	1.8	1.5	0.64
- Multiplication :	1.7	0.3	0.007
- Négation :	1.7	1.4	0.64
- AXPYIN :	3	0.55	0.013
- Division :	0.46	0.03	0.001

Fondements ensemblistes

- Groupe $(G, *)$: * binaire, interne, associative, \exists neutre, tout élément possède un inverse.
 - Un groupe est abélien si * est commutative
 - Un groupe est cyclique si il possède un générateur :
 - $\exists g \in G, \forall a \in G, \exists i \in \mathbb{Z}, a = g^i$
- Anneau $(A, +, \times)$: $(A, +)$ groupe abélien, \times associative, distributive sur +, \exists neutre pour \times (A est unitaire).
 - A^* est l'ensemble des inversibles par \times
 - Ex: $(\mathbb{Z}/n\mathbb{Z}, +, \times)$ et les inversibles sont les premiers avec n
- Corps $(A, +, \times)$: anneau et $A^* = A \setminus \{0\}$
 - $\mathbb{R}, \mathbb{C}, \mathbb{Q}$: infinis
 - Entiers modulo un nombre premier p $(\mathbb{Z}/p\mathbb{Z})$: finis de cardinal p

Système de chiffrement RSA

Théorie algorithmique des nombres

Système de chiffrement RSA

- Théorie**
 - Théorèmes d'Euler et de Fermat
 - Théorème chinois
 - Théorème Rivest-Shamir-Adleman
- Construction des clés**
 - Algorithme d'Euclide étendu
 - Élévation à la puissance
 - Tests de primalité
- Efficacité pratique**
- Cassage du code** : Factorisation

Petit théorème de Fermat

- Indicateur d'Euler : $\phi(n)$ nombre de premiers avec n
 → c'est le cardinal de $\mathbb{Z}/n\mathbb{Z}^*$
- Algorithme :
 - p premier, $\phi(p) = p - 1$, $\phi(p^k) = (p-1)p^{k-1}$
 - m, n premiers entre eux $\phi(mn) = \phi(m)\phi(n)$
- Théorème d'Euler : $a^{\phi(n)} \equiv 1 [n]$, pour a inversible modulo n
 - $G_a = \{y, y=ay, x \in \mathbb{Z}/n\mathbb{Z}^*\}$
 - $G_a = \mathbb{Z}/n\mathbb{Z}^*$, car a, x et y inversibles
 - $\prod_{x \in \mathbb{Z}/n\mathbb{Z}^*} x = \prod_{y \in G_a} y = \prod_{x \in \mathbb{Z}/n\mathbb{Z}^*} ax$
 - $\rightarrow a^{|\mathbb{Z}/n\mathbb{Z}^*|} = 1$
- Théorème de Fermat : $a^p \equiv a [p]$ pour p premier

Théorème Chinois

- Soient m_1, \dots, m_n positifs et premiers entre eux
- M leur produit
- Alors, pour tous résidus a_1, \dots, a_n il existe un unique $x \leq M$ tel que les restes de la division de x par m_i coïncident avec les a_i :

$$\forall a_1, \dots, a_k, \exists! x \leq M, x \equiv a_i [m_i]$$
- Construction (par Lagrange) :
 - Posons $M_i = M/m_i$,
 - D'après l'AEE, $\exists y_i$ tel que $y_i M_i \equiv 1 [m_i]$
 - Alors $x \equiv \sum a_i y_i M_i [M]$ convient !
- \exists Construction itérée par Newton (différence multipliées)

Théorème Rivest-Shamir-Adleman

- Conséquence des Théorèmes Chinois et d'Euler :
 - p et q premiers entre eux
 - $n = pq$, on a : $a^{k\phi(n)+1} \equiv a [n]$
- Dans le cas a *inversible* : Euler donne $a^{\phi(n)} \equiv 1 [n]$
- a *non-inversible* modulo n :
 - Si $a \equiv 0 [p]$, alors $a^{\phi(n)} \equiv 0 \equiv a [p]$
 - Si non a inversible mod p et $a^{p-1} \equiv 1 [p]$ donc $a^{(p-1)k+1} \equiv a [p]$
 - De même $a^{(q-1)k+1} \equiv a [q]$
 - Par théorème chinois, on alors $a^{\phi(n)+1} \equiv a [pq]$
- Intérêt : avec e et $d = k\phi(n) + 1$, e, n sont *publics*, d est *secret*
 - Codage de $a \rightarrow x \equiv a^e [n]$
 - Décodage de $x \rightarrow x^d \equiv a^{ed} \equiv a [n]$

Système de chiffrement RSA

- Théorie**
 - Théorèmes d'Euler et de Fermat
 - Théorème chinois
 - Théorème Rivest-Shamir-Adleman
- Construction des clés**
 - Algorithme d'Euclide étendu
 - Élévation à la puissance
 - Tests de primalité
- Efficacité pratique**
- Cassage du code** : Factorisation

Implémentation de RSA :

- Construire p et $q \rightarrow$ 1°) Test de primalité
- $(p-1)(q-1) = \phi(n)$
- Construire e et d :
 - On choisit e (pas trop petit) premier avec $\phi(n)$
 - \rightarrow 2°) Algorithme d'Euclide étendu donne d et $ktq + e + k\phi(n) = 1$
- Coder et décoder \rightarrow 3°) élévation à la puissance
- Casser le code
 i.e. : Trouver d , ne connaissant que e et $n \rightarrow$ calculer $\phi(n) = (p-1)(q-1)$
 Donc *factoriser* $n \rightarrow$ 4°) Factorisation d'entiers ?

Élévation à la puissance

- Ex: a^{11} , 10 multiplications ?
- $a^{11} = a^{10} a = (a^5)^2 a = ((a^2)^2 a)^2 a$: 5 multiplications !
- Algorithme *élévation récursive au carré*, pour $a^k [n]$:
 - Si $k = 0$ retourner 1
 - $d = \text{élévation récursive au carré}(a, \lfloor k/2 \rfloor)$
 - $d = d * d \% n$
 - Si k est impair retourner $d * a \% n$
 - Sinon retourner d
 (cf. Atelier ...)
- © Complexité : $O(t_n)$ multiplications

Tests de primalité

- Crible d'Eratosthène déterministe
 - Marquer tous les multiples jusqu'à \sqrt{n}
 - $O(n)$ additions, exponentiel, inutilisable au delà de 10 chiffres
- Miller/Rabin probabiliste polynomial (cf. atelier)
 - Tirage aléatoire d'un témoin
 - Si n est premier, le témoin indiquera *premier*
 - Si n est non premier, 3/4 des témoins indiqueront *nonpremier*
 - MAIS, 1/4 des témoins donneront une fausse indication
 - Répété k fois, probabilité de réponse *premier* incorrecte = 2^{-2k}
- Déterministe polynomial
 - Miller/Rabin avec suffisamment de témoins, sous ERH
 - Agrawal, Kayal et Saxena, sans ERH

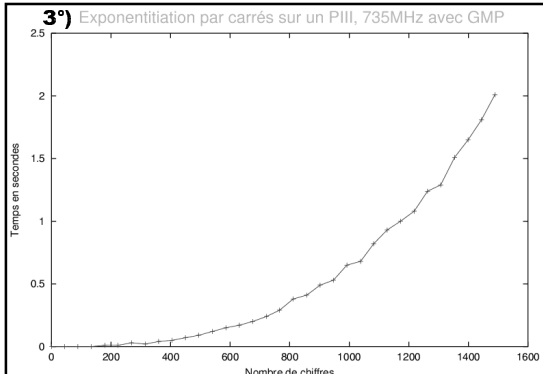
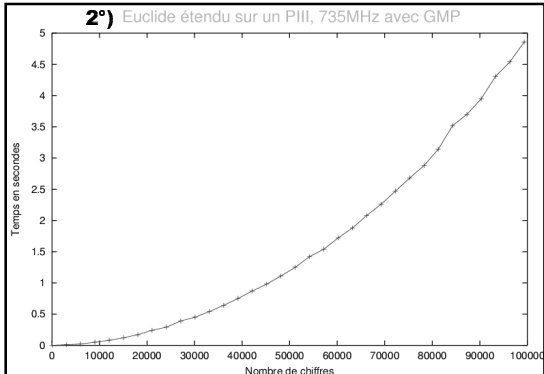
Miller-Rabin

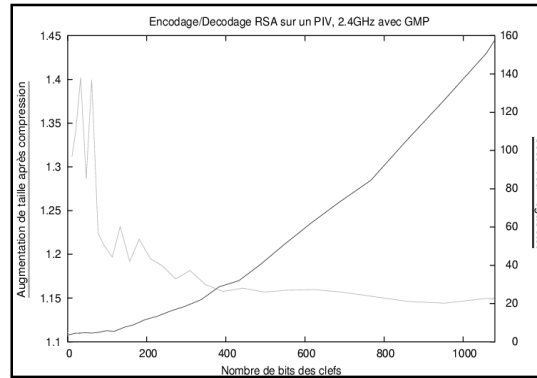
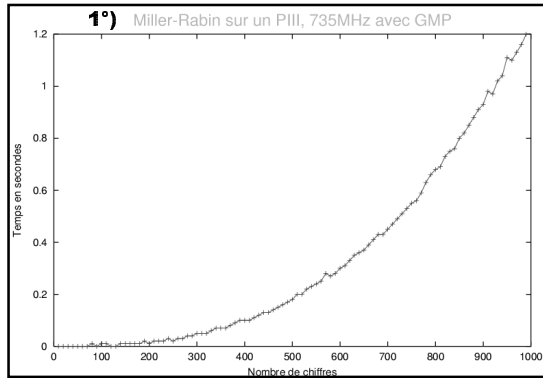
- Si n est impair, $n = t 2^s$, avec t impair
- Pour a quelconque on a alors :

$$a^{(n-1)} - 1 = (a^t)^{2^s} - 1 = (a^t - 1)(a^t + 1)(a^{2t} + 1) \dots (a^{(2^{s-1})t} + 1)$$
- Or, si n est premier alors Fermat donne $a^{n-1} \equiv 1[n]$, donc
 - Soit $a^t \equiv 1[n]$.
 - Soit $a^{2^i t} \equiv -1[n]$ avec $0 \leq i < s$.
- ⇒ Test de Miller-Rabin : tirer a au hasard et vérifier Fermat

Système de chiffrement RSA

1. Théorie
 - a) Théorèmes d'Euler et de Fermat
 - b) Théorème chinois
 - c) Théorème Rivest-Shamir-Adleman
2. Construction des clés
 - a) Algorithme d'Euclide étendu
 - b) Élévation à la puissance
 - c) Tests de primalité
3. Efficacité pratique
4. Cassage du code : Factorisation





Système de chiffrement RSA

1. Théorie
 - a) Théorèmes d'Euler et de Fermat
 - b) Théorème chinois
 - c) Théorème Rivest-Shamir-Adleman
2. Construction des clefs
 - a) Algorithme d'Euclide étendu
 - b) Élévation à la puissance
 - c) Tests de primalité
3. Efficacité pratique
4. Casser le code RSA ?

Casser le code RSA

- Trouver d, ne connaissant que e et n
→ calculer $\phi(n)=(p-1)(q-1)$
- Donc *décomposer n en ses facteurs premiers p et q*

Factorisation d'entiers

- Pollard rho : quelques secondes jusqu'à 25 chiffres
- Variantes si n-1, n+1 sont faciles à factoriser
- Courbes Elliptiques : quelques secondes jusqu'à 40 chiffres, record 57 chiffres au 21/06/2003
- Number Field Sieve : records de 244 chiffres (SNFS) et RSA160 (530 bits, GNFS), 03/2003 avec 104 machines sur 2 mois

Algorithme rho de Pollard

- Factorisation par pgcd (cf. AteMier)
 - On considère une suite $u_{k+1} \equiv f(u_k) [n]$ (souvent $f(x) = x^2+c$)
 - Soit p le plus petit facteur de n. Il existe i et j tels que $u_i \equiv u_j [p]$.
 - Donc $(u_i - u_j)$ est un multiple de p. En général non multiple de n.
 - Il suffit donc de faire $\text{pgcd}(u_i - u_j, n)$ pour trouver un facteur
 - Essayer tous les cas i et j : $O(p^2)$ opérations, $O(p \cdot n)$ espaces mémoire
- Algorithme Rho : recherche de cycle [Rho]
 - × L'idée c'est de ne pas garder tous les u_i précédents mais seulement $u_0, u_1, u_2, u_4, u_8, u_{16}, \dots$.
 - Car si $u_i \equiv u_j$ alors $u_{2^k i} \equiv u_{2^k j} \forall k$.
 - Garder successivement seulement les indices 2^k
 - ⊗ Au pire le nombre d'appels est doublé
 - ⊗ Complexité exponentielle $O(\sqrt{p}) = O(n^{1/2})$ avec $O(t_n)$ espaces mémoire



Cribles

- Meilleurs algorithmes actuels pour factoriser de grands entiers : cribles (quadratiques, de corps de nombres NFS, ...)
- Idée : Si $x^2 \equiv y^2 \pmod n$, avec $|x| \neq |y|$
alors $(x-y)(x+y) \equiv 0 \pmod n$
donc $\text{gcd}(x-y, n)$ est un facteur non-trivial de n.

- $87^2 = 7429 + 140$ et $140 = 2^2 \cdot 5 \cdot 7$
 - $88^2 = 7429 + 315$ et $315 = 3^2 \cdot 5 \cdot 7$

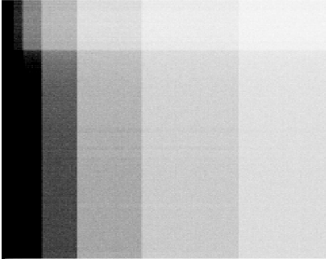
- $(87 \cdot 88)^2 \equiv (2^2 \cdot 3^2 \cdot 5^2 \cdot 7^2)^2 \pmod{7429}$
 • $x = 87 \cdot 88 = 227$
 • $y = 2^2 \cdot 3^2 \cdot 5 \cdot 7 = 210$
 • $7429 = 17 \cdot 347$

Crible quadratique

	Exposant de 2	Exposant de 3	Exposant de 5	Exposant de 7	...
83 ²	2	3	1	0	...
87 ²	2	0	1	1	...
88 ²	0	2	1	1	...
...

- (87*88)² est un carré ssi ligne 87² + ligne 88² est paire
- Trouver un vecteur binaire x tq x.M est pair
- Trouver un vecteur x tq x.M ≡ 0 modulo 2.

Matrice creuse de Crible



Noir ⇔ densité ≥ 0.01%

- Autre exemple : RSA-160 (cassé le 01/04/2003)
→ matrice presque carrée avec 5037191 colonnes

Échange de clefs

Construction des corps finis

Échange de clefs

1. Logarithme discret
 - Échanger une clef secrète
 - Logarithme Discret dans un corps fini
2. Corps premiers
 - implémentation classique
 - Implémentation avec inverse numérique
 - Racines primitives
3. Extensions
 - Anneau des polynômes sur un corps abélien
 - Anneau quotient G[X]/P
 - Test d'irréductibilité
 - Générateurs
 - Polynômes primitifs
 - Construction des grands corps finis
4. Casser le Logarithme Discret
 - Baby step Giant step, Méthode rho
 - Cribles

Logarithme discret

- Connaissant n, a et x, calculer b ≡ a^x [n]
 - Exponentiation rapide → OK, complexité O(t_n) multiplications
 - Exemple : 5¹¹ [7] ≡ ((5²)² 5)² 5 ≡ ((4)² 5)² 5 ≡ (2.5)² 5 ≡ 2.5 ≡ 3
- Connaissant n, a et b, calculer x ?
 - Exemple : 6^x ≡ 8 [11] ?
 - Beaucoup plus difficile
 - Énumération de tous les exposants : O(n) multiplications
 - Baby step / Giant step [Shanks], [Pollard] : O(√n)
 - ...
- x est le logarithme (discret) de b

6⁷ ≡ 8 [11]

Théorème du logarithme discret

- Si g est une racine primitive de Z/mZ
- ∀ x, y ∈ N

$$g^x ≡ g^y [m] \text{ ssi } x ≡ y [\phi(m)]$$
 - ⇨ si g^x ≡ g^y alors g^{x-y} ≡ 1, et donc x-y doit être un multiple de φ(m), car si g^t ≡ 1, alors t = qφ + r et donc g^r ≡ 1 avec r < φ(m)
 - ⇨ g^{φ(m)}} ≡ 1, donc si x=y+kφ, alors g^x ≡ g^y

Pour échanger une clef secrète

- Envoi d'une partie de la clef
 - Alice choisit pseudo-aléatoirement x et envoie $a \equiv g^x [n]$
 - Bob choisit pseudo-aléatoirement y et envoie $b \equiv g^y [n]$
 - Construction de la clef finale
 - Alice calcule $K \equiv b^x [n]$
 - Bob calcule $K \equiv a^y [n]$
- ⊙ À aucun moment la clef finale n'a transité
- Pour calculer K à partir de a et b , il faut résoudre un logarithme discret
- ⊙ Inefficace contre un adversaire actif (man-in-the-middle)

Des corps finis plus grands ?

- Le calcul du logarithme discret dépend du nombre d'éléments dans le corps
- Travailler avec des nombres premiers plus grands
 - ⊙ Même les calculs d'exponentiation deviennent lents
 - Travailler dans une extension
 - avec caractéristique accélérant les calculs $\rightarrow GF(2^k)$
 - ❖ Attention, $GF(2^k) \neq Z/2^kZ$

Échange de clefs

- Logarithme discret
 - Échanger une clef secrète
 - Logarithme Discret dans un corps fini
- Corps premiers
 - implémentation classique
 - Implémentation avec inverse numérique
 - Racines primitives
- Extensions
 - Anneau des polynômes sur un corps abélien
 - Anneau quotient $GF[x]/P$
 - Test d'irréductibilité
 - Générateurs
 - Polynômes primitifs
 - Construction des grands corps finis
- Casser le Logarithme Discret
 - Baby step Giant step, Méthode rho
 - Cribles

Quelques propriétés des corps finis

- Q et Z/pZ : corps premiers
 - $GF(q)$: corps fini ou corps de Galois à q éléments
- $K = \{k \in N^*, k \cdot 1_K = 0\}$; Caractéristique du corps :
- 0 si K est vide (ex: R, C, Q)
 - Plus petit élément de K (ex: p pour Z/pZ)
 - Caractéristique : 0 ou un nombre premier !
- Cardinal d'un corps fini = puissance de sa caractéristique
 - L'ensemble des inversibles d'un corps fini $GF(p^k)^*$ est cyclique de cardinal $p^k - 1$
 - L'ordre d'un élément est la pp puissance tq $x^n = 1$, il divise $p^k - 1$

Revenons à l'arithmétique modulaire sur des mots machine

implémentation classique

- Utilise division système pour le calcul de la multiplication
- AXPY:** $r = (a \times x + y)$; $r = (r < p ? r : r \% p)$;
- Addition, soustraction sont rapides
 - Multiplication est lente
- Requis : $(p^2 + p) < \text{taille des mots}$

Implémentation avec inverse numérique

- Stocke une représentation numérique de l'inverse de p
- AXPY:** $r = (a \times x + y)$; $r -= \text{floor}(r \times 1/p) \times p$
- Multiplication parfois plus rapide (pas de division, mais floor est assez lent également)
- Requiert aussi : $(p^2 + p) < \text{taille des mots}$

Utiliser un générateur

- Le groupe des inversibles est cyclique de cardinalité $q-1$:
 - Il existe un générateur, g
 - Chaque inversible est une puissance de g
 Ex. Mod 7 : $2^1=2, 2^2=4, 2^3=1$, et $3^1=3, 3^2=2, 3^3=6, 3^4=4, 3^5=5, 3^6=1$
- Un générateur des inversibles est une racine primitive
 - Théorème:
 - Si $m=2,4,p^k, 2p^k$, il y a $\phi(\phi(m))$ racines primitives dans $\mathbb{Z}/m\mathbb{Z}$
 - Sinon il n'y a pas de racine primitive
 - Si m est premier, $\mathbb{Z}/m\mathbb{Z}$ est un corps et il y a $\phi(m-1)$ racines primitives (il y a même $\phi(d)$ éléments d'ordre $d, \forall d | m-1$)

Comment trouver une racine primitive

- Il faut d'abord pouvoir tester si un nombre est générateur
 - Exhaustif, on calcule toutes les puissances jusqu'à trouver 1
 - L'ordre de a divise $\phi(m)$, donc Si $\forall p$ premier et divisant $\phi(m)$

$$a^{\frac{\phi(m)}{p}} \not\equiv 1[m]$$
 Alors a est une racine primitive.
 - Il faut tout de même factoriser m pour calculer $\phi(m)$
 - Factorisation partielle \rightarrow racine probablement primitive
- Ensuite, il faut tester des candidats
 - Dans 80% des cas, il semble qu'il existe une racine primitive ≤ 6
 - Tirages aléatoires : en moyenne $m^{-1/\phi(m-1)} < K \ln(\ln(m))$ essais
 - Moins de 0.1s sur 333MHz, pour $m < 2^{64}$

Petits corps premiers : générateurs 1/2

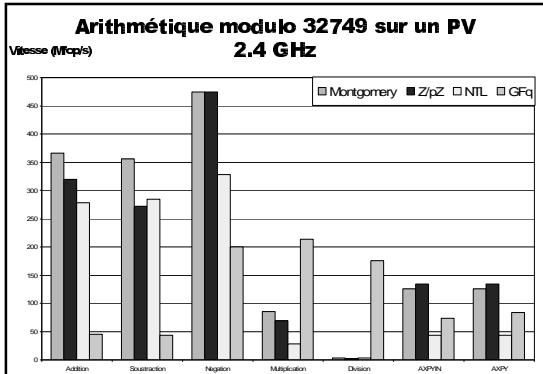
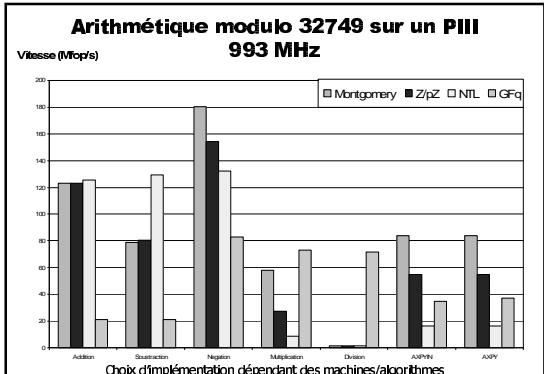
- Pré calcul de 3 tables
 - Moins de 2s de calcul pour des corps avec moins de 2^{24} éléments
 - Correspondance entre x et i : $t_1[x] = i, t_1 x = g^i$
 - Correspondance entre i et x : $t_2[i] = x, t_2 x = g^i$
 - Table des « Plus un » : $t_3[i] = j, t_3 g^{i+1} = g^j$
- [Conway] : faire les opérations sur les indices !
 - $a \times x : (g^j \times g^i) \% m = g^{j+i \pm(m-1)}$
 - 0 et 1 ont des valeurs particulières, par exemple 0 et $m-1$
- [Imamura], [Hubert], [Dovillet] \rightarrow réduisent la taille des tables

Petits corps premiers : générateurs 2/2

$a = g^i; x = g^j; y = g^k;$
 $a \times x + y = g^i \times g^j + g^k = g^k(1 + g^{i+j-k})$

AXPY: $r = i+j-k;$
 $r = t_3[r] + k;$
 et tests pour zéro, indices modulo $m-1$...

- Absolument aucune multiplication ni division système
- Chaque opération est une combinaison de tests, additions et accès aux tables.



Échange de clefs

1. Logarithme discret
 - Échanger une clef secrète
 - Logarithme Discret dans un corps fini
2. Corps premiers
 - Implémentation classique
 - Implémentation avec inverse numérique
 - Racines primitives
3. Extensions
 - Anneau des polynômes sur un corps abélien
 - Anneau quotient $GF[X]/P$
 - Test d'irréductibilité
 - Générateurs
 - Polynômes primitifs
 - Construction des grands corps finis
4. Casser le Logarithme Discret
 - Baby step Giant step, Méthode rho
 - Cribles

Casser le Logarithme Discret

- Recherche exhaustive : $O(n)$ multiplications
- [Shanks] Baby Step Giant Step $O(n^{1/2})$ multiplications, espace aussi en $O(\sqrt{n})$
- [Pollard] méthode Rho $O(n^{1/2})$ multiplications, $O(t_r)$ mémoire
- [Pohlig-Hellman] Il faut factoriser $n = |G| = \prod p_i^{e_i}$
 - Alors $O(\sum e_i (t_{r_i} + \sqrt{p_i}))$
- Les calculs d'index : la meilleure complexité actuelle
 - [Coppersmith] pour $GF(2^c)^*$: $L_2^*[1/3, c < 1.587] = O(n^{1/3})$
 - NFS pour Z/pZ^* : $L_p[1/3, 1.923] = O(n^{1/3})$
 - Non applicable dans tous les groupes (comme les courbes elliptiques sur un corps fini)

Baby step Giant step

- $h=g^x$ dans un groupe G de cardinal n. Soit $m = \lceil \sqrt{n} \rceil$
- x pourra s'écrire $x = i m + j$ donc $g^x = g^{im} g^j$
- Ou encore $h (g^{-m})^i = g^j$

1. On calcule et on stocke les m premiers g^j	$O(m)$ mult
2. On trie les m premiers g^j	$O(m \log(m))$ comp
3. Calculer g^m par carrés récursifs	$O(\log(m))$ mult
4. Inverser, $y = g^{-m}$	$O(\log(m)^2)$ mult
5. Pour i de 1 à m	$O(m \log(m))$ comp

si h y'est un des g^j renvoyer $x = i m + j$

- Complexité : $O(\sqrt{n})$ opérations arithmétiques, $O(\sqrt{n} t_r)$ tests
- ⊗ $O(\sqrt{n})$ espaces mémoire

Méthode rho !

- Il faut trois sous ensembles S_1, S_2, S_3 de $G, n = |G|$
 Ex: dans $Z/pZ, S_1 = \{ u = 1 [3] \}, S_2 = \{ u = 0 [3] \}, S_3 = \{ u = 2 [3] \}$

$$u_{k+1} = f(u_k) = \begin{cases} h u_k & \text{si } u_k \in S_1 \\ u_k^2 & \text{si } u_k \in S_2 \\ g u_k & \text{si } u_k \in S_3 \end{cases}$$

- Donc $u_i = g^{a_i} h^{b_i}$
- Or, il existe i tel que $u_i = u_j$ mais la plupart du temps $b_i \neq b_j$!
- Dans ce cas $g^{a_i} h^{b_i} = g^{a_j} h^{b_j}$, ou encore $h^{b_i - b_j} = g^{a_j - a_i}$, or $h = g^d$, donc les indices doivent vérifier $(b_i - b_j) x \equiv a_j - a_i [n]$
- Ce qui nous donne $x \equiv (a_j - a_i)(b_i - b_j)^{-1} [n]$.
- Complexité : $O(\sqrt{n})$ en général si f est bien aléatoire, $O(t_r)$ mémoire

Cribles : calcul d'index

- Si $\prod_{i=1}^m x_i \equiv \prod_{j=1}^n y_j$ pour certains éléments de $GF(q)^*$
- Alors $\sum_{i=1}^m \log_g x_i \equiv \sum_{j=1}^n \log_g y_j [q-1]$


⊗ Ainsi on obtient plusieurs équations linéaires en les \log_g

⊗ Si les x_i et y_j ne sont pas trop nombreux et si certains $\log_g(z)$ sont connus (par exemple pour $z=g$!), le système linéaire peut être résolu

- ⇒ Pour trouver des z simples :
- ⇒ prendre a au hasard et tester si $g^a [p] \equiv \prod p_i$

Algorithmique de la Théorie des nombres

Jean-Guillaume Dumas



UNIVERSITÉ JOSEPH FOURIER
SCIENTIFIQUE TECHNIQUE MÉTIERS

Laboratoire de
Modélisation et Calcul

