DEA Informatique, Mathématiques et Applications
Filière algorithmique, complexité et cryptographie

Mémoire de DEA

Factoriser des entiers par la méthode des courbes elliptiques

Reynald LERCIER

Laboratoire d'Informatique
Ecole Polytechnique
Route de Saclay
91128 Palaiseau Cedex

Juin 1993

Remerciements

Je tiens à remercier toutes les personnes qui m'ont aidé à réaliser ce travail. Notamment :

- François Morain. De notre première rencontre où il m'a intéressé à un domaine qui m'était nouveau jusqu'à aujourd'hui, il est toujours resté disponible, su m'impliquer dans des activités de recherche et m'a soutenu dans les moments difficiles. Je lui dois de plus un sujet de DEA particulièrement formateur qui, grâce à ses remarques liées aussi bien à la programmation qu'à des aspects plus théoriques, m'a permis d'appréhender des difficultés que je ne soupçonnais pas auparavant.
- L'Ingénieur Principal de l'Armement P. Dejean et l'Ingénieur de l'Armement V. Auger. Ils m'ont acceuilli au sein du Groupe Etude Chiffre du CELAR d'octobre à décembre 1992.
- Le centre informatique de l'École Nationale Supérieure des Techniques Avancées. J'ai pu avec son accord utiliser en continu ses stations de travail pour factoriser des entiers.
- Le LIX pour m'avoir acceuilli et son secrétariat pour sa gentillesse.

Table des matières

Chapitre 1

Introduction

Les systèmes de chiffrement à clefs publiques et plus particulièrement le système "RSA" en 1976 [8] sont à l'origine d'un regain d'intérêt pour des problèmes comme la factorisation d'entiers, la primalité ou encore le logarithme discret. De nouveaux algorithmes prenant en compte la faisabilité des calculs sur les machines actuelles ont ainsi vu le jour ces dix dernières années.

En ce qui concerne les algorithmes de factorisation d'entiers, trois algorithmes majeurs ont vu le jour entre 1980 et 1990 : Le crible quadratique, la méthode des courbes elliptiques et le crible algébrique.

Les algorithmes de crible quadratique [9, 10] et de crible algébrique [6] recherchent pour factoriser un entier N de nombreuses congruences $x \equiv y \bmod N$ où x et y sont des carrés ou des produits de nombres premiers issus d'une "base de factorisation". Quand suffisamment de telles congruences sont obtenues, on peut alors trouver deux entiers X et Y tels que $X^2 \equiv Y^2 \bmod N$ sans que $X \equiv \pm Y \bmod N$. On a ensuite un facteur de N par $\gcd(X+Y, N)$ ou $\gcd(X-Y, N)$. La façon de découvrir les congruences modulo N diffère suivant les deux méthodes mais elles ont toutes les deux une complexité dépendant de la taille de N. Le crible quadratique a ainsi pu factoriser un nombre de 120 chiffres alors que le crible algébrique qui ne s'applique qu'aux nombres de la forme $b^n \pm 1$ avec $b \leq 12$ a permis de factoriser le nombre de Fermat $2^{523} - 1$ de 158 chiffres.

La méthode des courbes elliptiques découverte par Hendrik W. Lenstra, Jr en 1985 [11] a une complexité sous exponentielle en la taille du plus petit facteur p de N. Elle consiste à rechercher l'ordre d'un élément du groupe défini par une courbe elliptique sur $\mathbf{GF}(p)$ grâce à une arithmétique modulo N. Les plus grands facteurs découverts à ce jour par cette méthode ont une quarantaine de chiffres.

quoi	par	chiffres	quand
$11^{118}+1$	Arjen K. Lenstra Marc S. Manasse	38	1991
F_{467}	Robert D. Silverman	38	1992
F_{667}	Peter L. Montgomery	38	1992
$p(1127)$	Arjen K. Lenstra	40	1992
$10^{201}-1$	Dave Rusin	42	1992
$p(19997)$	Franz-Dieter Berger Thomas Denny	43	1993

Tableau 1 : Principaux facteurs découverts par l'algorithme de Lenstra.

Ce travail est la suite d'un précédent stage réalisé d'avril à juillet 1992 à l'École Polytechnique. Il a eu lieu d'octobre à décembre 1992 au Centre d'Électronique de L'Armement[1] et d'avril à juin 1993 au Laboratoire d'Informatique de l'École Polytechnique[2].

Après avoir rappelé les propriétés essentielles des courbes elliptiques au chapitre 1, nous décrirons l'algorithme avec ses différents développements au chapitre 2. Il y apparaîtra que pour pouvoir mettre en œuvre une seconde phase efficace de la méthode, une algorithmique polynomiale asymptotiquement rapide est nécessaire, ce sera l'objet du chapitre 3. A l'aide cette algorithmique polynomiale, nous serons en mesure de détailler dans le chapitre 4 les choix algorithmiques faits pour l'implantation MECM de l'algorithme. Enfin le chapitre 5 fera mention des factorisations obtenues.

[1]CELAR, Route de Laillé, 35170 BRUZ.
[2]LIX, École Polytechnique, Route de Saclay, 91128 Palaiseau.

Chapitre 2

Les courbes elliptiques sur $\mathbf{Z}/p\mathbf{Z}$

Les résultats suivants, énoncés pour la plupart dans le livre de J.H. Silverman [19], sont valides pour des entiers p strictement plus grand que 3.

2.1 Premières définitions

On appelle $\mathbf{P}^2(\mathbf{Z}/p\mathbf{Z})$ l'ensemble des classes $(x\ :y\ :z)$ de la relation d'équivalence :

$$\forall (x,y,z) \in (\mathbf{Z}/p\mathbf{Z})^3, \forall (x',y',z') \in (\mathbf{Z}/p\mathbf{Z})^3,$$
$$(x,y,z) \equiv (x',y',z') \Leftrightarrow \exists c \in (\mathbf{Z}/p\mathbf{Z})^*, x' = cx, y' = cy, z' = cz. \tag{2.1}$$

C'est sur $\mathbf{P}^2(\mathbf{Z}/p\mathbf{Z})$ que sont définies les courbes elliptiques.

Définition 1 *Une courbe elliptique est une paire $(a,b) \in (\mathbf{Z}/p\mathbf{Z})^2$ où a et b sont les coefficients de l'équation :*

$$y^2 = x^3 + ax + b \quad avec \quad \Delta = -16(4a^3 + 27b^2) \neq 0 \bmod p. \tag{2.2}$$

L'ensemble des points qui lui sont associés est alors :

$$\mathbf{E}_{a,b}(\mathbf{Z}/p\mathbf{Z}) = \{(x\ :y\ :z) \in \mathbf{P}^2(\mathbf{Z}/p\mathbf{Z}), y^2z = x^3 + axz^2 + bz^3\}. \tag{2.3}$$

(2.2) est appelée paramétrisation de Weierstrass.

Un seul point correspond à $z = 0$, il s'agit de $O = (0\ :1\ :0)$ et chaque classe différente de O a un unique représentant $(x\ :y\ :1)$. On considérera donc par la suite que $\mathbf{E}_{a,b}(\mathbf{Z}/p\mathbf{Z})$ est la réunion de O (vu comme point à l'infini) avec l'ensemble

$$\mathbf{E}_{a,b} = \{(x,y) \in \mathbf{Z}/p\mathbf{Z}^2, y^2 = x^3 + ax + b\}. \tag{2.4}$$

2.2 Propriétés pour p premier

Tout l'intérêt d'une courbe elliptique pour la factorisation d'entiers, c'est de pouvoir munir son ensemble de points (2.3) d'une loi de groupe.

Définition 2 $(\mathbf{E}_{a,b}(\mathbf{Z}/p\mathbf{Z}), +)$ *est un groupe d'élément neutre O si on munit cet ensemble de la loi d'addition $M_3(x_3,y_3) = M_1(x_1,y_1) + M_2(x_2,y_2)$ suivante :*

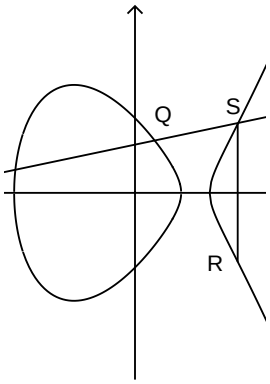


FIG. 2.1 – Une courbe elliptique sur $\mathbf{R}$.

– *Si* $M_1 = O, M_1 + M_2 = M_2$.
– *Si* $M_2 = O, M_1 + M_2 = M_1$.
– *Si* $x_1 = x_2$ *et* $y_1 = -y_2$, $M_1 + M_2 = O$.
– – *Si* $M_1 \neq M_2$, *on pose* $\lambda = (y_1 - y_2)/(x_1 - x_2)$.

– *Si* $M_1 = M_2$, *on pose* $\lambda = (3x_1^2 + a)/(2y_1)$.
On obtient alors M_3 *avec*

$$\begin{cases} x_3 = \lambda^2 - x_1 - x_2 \\ y_3 = \lambda(x_1 - x_3) - y_1 = \lambda(x_2 - x_3) - y_2. \end{cases}$$

La quantité kP *correspond de plus à* $\overbrace{P + P + \ldots + P}^{k\ fois}$ *pour tout* $k \in \mathbf{N}^*$ *et à* O *pour* $k = 0$.

Sur $\mathbf{R}$, cela revient à associer à deux points M_1 et M_2 le symétrique M_3 par rapport à l'axe des abscisses du point intersection P de la droite D passant par M_1 et M_2 avec la courbe. Si jamais les deux points sont égaux, il suffit alors de prendre pour D la tangente à la courbe en M_1.

Théorème 1 *Le nombre de courbes elliptiques sur* $\mathbf{Z}/p\mathbf{Z}$ *vaut* $p(p-1)$.

PREUVE : Le nombre de courbes elliptiques sur $\mathbf{Z}/p\mathbf{Z}$ est le nombre de couples $(a, b) \in (\mathbf{Z}/p\mathbf{Z})^2$ tels que $4a^3 + 27b^2 \neq 0 \bmod p$. Le nombre de toutes les paires vaut p^2, et $4a^3 + 27b^2 \equiv 0 \bmod p$ si et seulement si $a \equiv -3c^2$, $b \equiv 2c^3$ pour $c \in \mathbf{Z}/p\mathbf{Z}$. Chaque c étant de plus déterminé de manière unique par $c \equiv -3b/2a \bmod p$ $(a \neq 0)$, le nombre des couples (a, b) tels que $4a^3 + 27b^2 \neq 0 \bmod p$ vaut p et celui des courbes elliptiques $p^2 - p$.

Certaines courbes ont cependant le même comportement du point de vue de la loi de groupe. On considère alors des classes de courbes isomorphes définissant une même structure de groupe.

Définition 3 *Un isomorphisme* $u : \mathbf{E}_{a,b} \to \mathbf{E}_{a',b'}$ *est défini par la donnée d'un élément* u *de* $\mathbf{Z}/p\mathbf{Z}$ *vérifiant* $a' = u^4 a$, $b' = u^6 b$.

Il induit l'isomorphisme $(x : y : z) \to (u^2 x : u^3 y : z)$ *et* $\mathbf{E}_{a,b}$ *est alors dite isomorphe à* $\mathbf{E}_{a',b'}$.

Plus particulièrement, un automorphisme de $\mathbf{E}_{a,b}$ *est un isomorphisme de* $\mathbf{E}_{a,b}$ *dans* $\mathbf{E}_{a,b}$ *et l'ensemble des automorphismes d'une courbe* $\mathbf{E}_{a,b}$ *est noté* $\mathrm{Aut}(\mathbf{E}_{a,b})$.

Compter le nombre de groupes distincts définis par des courbes elliptiques sur $\mathbf{Z}/p\mathbf{Z}$ nécessite donc une connaissance précise de $\mathrm{Aut}(\mathbf{E}_{a,b})$.

Théorème 2 *On a les 3 propriétés :*

1. *Si* $a = 0$ *et* $p \equiv 1 \bmod 6$, $\mathrm{Aut}(\mathbf{E}_{a,b})$ *est un groupe cyclique d'ordre 6.*
2. *Si* $b = 0$ *et* $p \equiv 1 \bmod 4$, $\mathrm{Aut}(\mathbf{E}_{a,b})$ *est un groupe cyclique d'ordre 4.*
3. *Sinon* $\mathrm{Aut}(\mathbf{E}_{a,b})$ *est réduit à* $\{I, -I\}$.

Ce qui permet d'obtenir le résultat suivant :

Théorème 3 *Le nombre* $\#Cl(E)_{\cong \mathbf{Z}/p\mathbf{Z}}$ *de groupes distincts sur* $\mathbf{Z}/p\mathbf{Z}$ *vaut respectivement* $2p+6$, $2p+2$, $2p+4$, $2p$ *pour* $p \equiv 1, 5, 7, 11 \bmod 12$.

PREUVE : Le nombre des courbes elliptiques isomorphes à une courbe donnée vaut $\#(\mathbf{Z}/p\mathbf{Z})^*/\#\mathrm{Aut}(E) = (p-1)/\#\mathrm{Aut}(E)$ d'après la définition 3. Le calcul de $\#\mathrm{Aut}(E)$ vient alors du théorème 2 :

- Si $p \equiv 11 \bmod 12$, $\forall E$ courbe elliptique, $\#\mathrm{Aut}(E) = 2$ d'après le théorème 2.3 et le nombre de classes vaut $\frac{p^2-p}{(p-1)/2} = 2p$.
- Si $p \equiv 7 \bmod 12$, on a 6 classes de courbes elliptiques telles que $\#\mathrm{Aut}(E) = 6$ correspondant d'après le théorème 2.1 aux 6 éléments de $(\mathbf{Z}/p\mathbf{Z})^*$ d'ordre 6 et le nombre total de classes vaut $6 + \frac{(p^2-p)-(p-1)}{(p-1)/2} = 2p+4$.
- Si $p \equiv 5 \bmod 12$, on a 4 classes de courbes elliptiques telles que $\#\mathrm{Aut}(E) = 4$ correspondant d'après le théorème 2.2 aux 4 éléments de $(\mathbf{Z}/p\mathbf{Z})^*$ d'ordre 4 et le nombre total de classes vaut $4 + \frac{(p^2-p)-(p-1)}{(p-1)/2} = 2p+2$.
- Si $p \equiv 1 \bmod 12$, on a 4 classes de courbes elliptiques telles que $\#\mathrm{Aut}(E) = 4$ correspondant d'après le théorème 2.2 aux 4 éléments de $(\mathbf{Z}/p\mathbf{Z})^*$ d'ordre 4 et 6 classes de courbes elliptiques telles que $\#\mathrm{Aut}(E) = 6$ correspondant d'après le théorème 2.1 aux 6 éléments de $(\mathbf{Z}/p\mathbf{Z})^*$ d'ordre 6. Le nombre total de classes vaut donc $4 + 6 + \frac{(p^2-p)-2(p-1)}{(p-1)/2} = 2p+6$.

On peut alors citer le résultat[1] fondamental suivant :

Théorème 4 *L'ordre d'une courbe elliptique* $\mathbf{E}_{a,b}$ *sur* $\mathbf{Z}/p\mathbf{Z}$ *vaut :*

$$p + 1 - t \quad avec \quad |t| \leq 2\sqrt{p}.$$

Et réciproquement, pour tout $m \in [(\sqrt{p}-1)^2,\ (\sqrt{p}+1)^2]$, *il existe une courbe elliptique* $\mathbf{E}_{a,b}$ *d'ordre* m.

Ainsi, pour un p donné, il existe un grand nombre de courbes elliptiques d'ordres distincts.

[1] Démontré pour la première fois par Hasse.

2.3 Généralisation à N non premier

Dans le cas où N est composé (non premier), on peut encore définir une loi de groupe pour une courbe $\mathbf{E}_{a,b}$ sur $\mathbf{Z}/N\mathbf{Z}$. Mais vu la complexité de la théorie des courbes elliptiques définies sur un anneau, on préfère généralement définir à moindre coût une **pseudo addition** avec les mêmes formules que la définition 2.

Cette pseudo addition n'est cependant possible que si le nombre w à inverser lors du calcul de λ pendant la pseudo-addition de deux points $M_1(x_1, y_1)$ et $M_2(x_2, y_2)$ est premier avec N.

Si ce n'est pas le cas, pour un des facteurs premiers p de N, cela équivaut lors de l'addition des deux points $M_1^p(x_1 \bmod p, y_1 \bmod p)$ et $M_2^p(x_2 \bmod p, y_2 \bmod p)$ sur la courbe $\mathbf{E}_{a,b}$ considérée sur $\mathbf{Z}/p\mathbf{Z}$ à

$$M_1^p + M_2^p = O^p.$$

Ainsi, dès qu'une addition sur une courbe modulo N est impossible, les deux points à sommer sont opposés modulo p et on peut avoir un diviseur de N en calculant $p = \gcd(w, N)$. C'est ce résultat qui est à la base de l'algorithme de factorisation de Lenstra.

Chapitre 3

La méthode de factorisation des courbes elliptiques

3.1 Un précurseur, la méthode $p-1$

Cette méthode est due à J.M. Pollard [18] et repose sur le petit théorème de Fermat dont voici l'énoncé :

Théorème 5 *Si p est un nombre premier, $\forall a \in (\mathbf{Z}/p\mathbf{Z})^*$ $a^{p-1} \equiv 1 \bmod p$.*

D'après le théorème précédent, tout facteur premier p d'un entier N divise $\gcd(a^{p-1} - 1, N)$. Si on trouve un entier r multiple de $p-1$ sans être pour autant un multiple de $\varphi(N)$ (φ étant la fonction d'Euler), $\gcd(a^r - 1, N)$ sera égal à un facteur non trivial de N.

```
procedure Pollard(a, b, N)
# NextPrime(p) retourne le nombre premier
# immédiatement supérieur à p.
p := NextPrime(1)
g := 1
while p < b and g = 1 do
   k := p
   while k.p < b do k := k.p end
   a := a^k mod N
   g := gcd(a - 1, N)
   p := NextPrime(p)
end
return(g)
end Pollard
```

Algorithme 1 : La méthode $p-1$.

L'algorithme $p-1$ repose sur ce constat en choisissant pour r le produit de "petits" nombres premiers.

Pour à titre d'exemple factoriser 143 avec $a = 2$ et $b = 5$, on calcule d'abord $a = 2^4 \bmod 143 = 16$ et $\gcd(a-1, 143) = 1$. A l'étape suivante, $a = 16^3 \bmod 143 = 92$ et on a le facteur 13 de 143 par $\gcd(a-1, 143)$.

La méthode a été ici un succès car $13 - 1 = 2^2 3$ n'a que 2 ou 3 comme facteur premier. Dans le cas général, elle réussit si au moins un facteur p de N est tel que tous les diviseurs de $p-1$ soient petits ($p-1$ est dit **friable**).

Avec une version optimisée, un facteur de 34 chiffres du 575^{eme} nombre de Fibonnacci a été découvert par P.L. Montgomery [15]. Il est cependant facile de mettre la méthode en échec pour des nombres N ayant tous leurs facteurs premiers p avec $p-1$ non friable.

3.2 L'algorithme de Lenstra

La méthode des courbes elliptiques fut découverte par H. W. Lenstra, Jr [12]. Elle repose sur les mêmes principes que la méthode $p-1$ à l'exception près qu'au lieu de travailler sur l'anneau $\mathbf{Z}/N\mathbf{Z}$ où N est l'entier à factoriser, on travaille sur une courbe elliptique $\mathbf{E}_{a,b}$ choisie au hasard sur $\mathbf{Z}/N\mathbf{Z}$.

Plus précisément, on prend un point P sur une courbe $\mathbf{E}_{a,b}$ aléatoire et une borne $B_1 \in \mathbf{N}^*$ fixée par l'utilisateur. On cherche alors l'ordre de ce point sur la courbe $\mathbf{E}_{a,b}$ modulo un des facteurs premiers p de N. On calcule pour cela kP où k est aussi le produit des "petits" nombres premiers inférieurs à B_1. Si pour un des facteurs, l'ordre de P est friable, k sera alors un multiple de cet ordre et une des inversions pendant le calcul de kP donnera un facteur de N.

Étant donné qu'une pseudo-addition modulo N est impossible dès que l'on cherche à additionner deux opposés modulo un facteur premier p de N, échouer dans le calcul de kP signifie que $kP = O \bmod p$, ce qui équivaut à k multiple de l'ordre s du sous-groupe engendré par P sur $\mathbf{E}_{a,b}$ modulo p. Plus s est friable, plus la méthode est efficace.

procedure ECM($P, \mathbf{E}_{a,b}, B_1, N$)
p := NextPrime(1)
g := 1
while $p < b_1$ **and** $g = 1$ **do**
  k := p
  while $k.p < B_1$ **do** k := $k.p$ **end**
  # EcModMult($k, P, \mathbf{E}_{a,b}, g, N$) calcule kP mod N sur $\mathbf{E}_{a,b}$.
  # Si une inversion est impossible, g est un facteur de N.
  P := EcModMult($k, P, \mathbf{E}_{a,b}, g, N$)
  p := NextPrime(p)
end
return(g)
end ECM

Algorithme 2 : L'algorithme de Lenstra.

Pour factoriser 143, considérons $B_1 = 4$, la courbe $\mathbf{E}_{1,1} : y^2 = x^3 + x + 1$ modulo 143, $P = (0 : 1 : 1)$ et appliquons l'algorithme ainsi :

- Première étape : $R = 2P = (36 : 124 : 1)$
- Seconde étape : $R = 3.2P = 2.2P + 2P = (127 : 71 : 1) + (36 : 124 : 1)$. L'addition de ces deux points étant alors impossible pendant le calcul de $\frac{124-71}{36-127}$, on obtient un facteur de 143 en calculant $\gcd(127 - 36, 143) = 13$.

L'avantage par rapport à la méthode $p - 1$, c'est qu'à chaque courbe $\mathbf{E}_{a,b}$ correspond des ordres s différents. Ainsi, en cas d'échec lors de la factorisation de N, il suffit de changer de courbe et de recommencer en espérant avoir plus de chance. Il est de plus possible d'utiliser plusieurs courbes et points à la fois (Et il y en a un grand nombre d'après le théorème 4). On espère alors que l'un des ordres s est plus friable que les autres.

Le facteur de 43 chiffres du nombre de partition $p(19997)$ vient ainsi d'être récemment découvert par Franz-Dieter Berger.

3.3 Complexité de l'algorithme

On reprend ici l'analyse de la complexité de l'algorithme faite par R. P. Brent [5]. Courbe et point initial y sont considérés choisis aléatoirement et on suppose connu les résultats suivants [7] :

Définition 4 *Soit $M > 0$ et $n_1 \geq n_2 \geq \ldots$ les facteurs premiers d'un entier M, on définit :*

$$\rho(\alpha) = \lim_{M \to +\infty} \text{prob}(n_1 < M^{1/\alpha}).$$

On peut alors montrer que ρ vérifie l'équation :

$$\alpha\rho'(\alpha) + \rho(\alpha - 1) = 0, \tag{3.1}$$

ou encore le résultat asymptotique :

$$\log \rho(\alpha) = -\alpha(\log \alpha + \log\log \alpha - 1) + o(\alpha) \quad \text{quand} \quad \alpha \to \infty, \tag{3.2}$$

qui mène à :

$$\rho(\alpha - 1)/\rho(\alpha) = \alpha(\log \alpha + O(\log\log \alpha)) \quad \text{quand} \quad \alpha \to \infty. \tag{3.3}$$

Dans l'algorithme de Lenstra, le coût essentiel est dû aux multiplications et aux inversions modulo N. On néglige le coût des additions modulo N et des multiplications/divisions sur de petits entiers. On prendra donc le coût d'une multiplication comme unité de base (u.b) et K u.b, celui d'une inversion.

Les méthodes usuelles du calcul de kP pour k entier ont pour coût $c \log k$ u.b avec $c \in \mathbf{R}^+$ (cf paragraphe 5.1.2). Sachant que le nombre d'entiers premiers inférieurs à x est asymptotiquement $\pi(x) = \frac{x}{\log x}$, un essai complet testant tous les nombres premiers inférieurs à B_1 nécessite donc cB_1 u.b.

Supposons que les résultats de la définition 4 s'applique aussi pour M non infini (hypothèse raisonnable), la probabilité qu'un essai réussisse à trouver un facteur p vaut $\rho(\alpha)$ avec $\alpha = \frac{\log p}{\log B_1}$ et le travail nécessaire est alors

$$W(\alpha) \simeq cp^{1/\alpha}/\rho(\alpha). \tag{3.4}$$

On cherche à minimiser W en différenciant (3.4) et en annulant le résultat. On obtient alors $\log p = -\alpha^2\rho'(\alpha)/\rho(\alpha)$, ou encore d'après (3.1),

$$\log p = \alpha\rho(\alpha-1)/\rho(\alpha). \tag{3.5}$$

En supposant p connu, cherchons α tel que (3.5) soit satisfait. On a d'après (3.3)

$$\log p = \alpha^2(\log\alpha + O(\log\log\alpha)), \tag{3.6}$$

d'où

$$\alpha \simeq (\frac{2\log p}{\log\log p})^{1/2}. \tag{3.7}$$

Comme d'autre part

$$\log W(\alpha) \simeq \log c + \frac{\rho(\alpha-1)}{\rho(\alpha)} - \log\rho(\alpha) \simeq \log c + 2\alpha\log\alpha, \tag{3.8}$$

on obtient finalement

$$\boxed{W(p) = ce^{(1+o(1))\sqrt{2\log(p)\log\log(p)}}}$$

Deux remarques :

- Cette complexité signifie que le temps nécessaire à l'algorithme pour trouver le facteur p de N est égal au temps mis pour réaliser $W(p)$ multiplications modulo N . Elle dépend de la taille de p et non pas de N comme celle du crible quadratique [9, 10], par exemple.
- $W(p)$ étant sous-exponentiel, On comprend ici l'importance d'optimiser au maximum les opérations sur courbes elliptiques afin d'améliorer la constante multiplicative c.

3.4 Courbes elliptiques avec un sous groupe de torsion d'ordre 16 sur Q

Si on pouvait avoir des courbes d'ordres suffisamment friables, la méthode serait fortement accélérée. C'est pourquoi H. Suyama [20] ou encore A. O. L. Atkin et F. Morain [3] ont proposé des courbes contenant des sous groupes de torsion d'ordres connus.

Par un théorème de Mazur, on sait que les seuls groupes de torsion sur $\mathbf{Q}$ sont isomorphes à :

$$\begin{cases} \mathbf{Z}/m\mathbf{Z} & m = 1, 2, \ldots, 10, 12 \\ \mathbf{Z}/2\mathbf{Z} \times \mathbf{Z}/2m\mathbf{Z} & m = 1, 2, 3, 4. \end{cases}$$

Les 2 articles précédents donnent des courbes contenant des sous-groupes de torsion isomorphes à la plupart de ceux déjà cités. Je ne rappellerai ici que des résultats sans démonstrations pour des courbes avec un sous groupe de torsion d'ordre 16.

Toutes les courbes intéressantes sont en fait des courbes de Kubert :

Définition 5 *Une courbe de Kubert est donnée par :*

$$\mathcal{E}(b,c) \ : Y^2 + (1-c)XY - bY = X^3 - bX^2$$

$$\textit{avec } (b,c) \in \mathbf{Q} \ \textit{ tels que } \ 16b^5 - (8c^2+20c-1)b^4 - c(1-c)^3b^3 \neq 0.$$

Elle a alors pour point d'ordre fini maximal le point P(0,0).

On a en outre :

Théorème 6 *: Pour b et c donné par :*

$$d = \frac{2\alpha(4\alpha+1)}{8\alpha^2-1}, c = \frac{(2d-1)(d-1)}{d}, b = (2d-1)(d-1) \ \textit{ avec } \ \alpha \in N,$$

$\mathcal{E}(b,c)$ *possède un sous groupe de torsion d'ordre 16. Si de plus* $\alpha = \left(\frac{t+25}{s-9}+1\right)^{-1}$ *où (s,t) est un point de*

$$T^2 = S^3 - 8S - 32 \tag{3.9}$$

la courbe $\mathcal{E}(b,c)$ *a un point rationnel d'abscisse (1-2d)/4.*

Quand on sait que le point $P(12,40)$ est un point d'ordre infini de (3.9), chaque multiple de P permet quasiment d'obtenir une courbe $\mathcal{E}(b,c)$ différente.

Il suffit alors transformer ces courbes

– soit en notation de Weierstrass :

Théorème 7 *L'équation en notation de Weierstrass provient de la normalisation de la courbe :*

$$y^2 = x^3 + \frac{((c-1)^2-4b)}{4}x^2 + \frac{b(c-1)}{2}x + \frac{b^2}{4} \tag{3.10}$$

obtenue après le changement de variable :

$$\begin{cases} X &= x \\ Y &= y + (b-(1-c)x)/2. \end{cases}$$

Le point d'abscisse $(1-2d)/4$ *y est alors rationnel.*

– soit en notation de Montgomery (courbes de la forme $By^2 = x^3 + Ax^2 + x$ avec $(A,B) \in \mathbf{Q}^2$, $B \neq 0$ et $A \neq \pm 2$) :

Théorème 8 *On obtient l'équation en notation de Montgomery :*

$$y^2 = x^3 + \left(\left[\frac{(2d-1)^2}{2d(d-1)}\right]^2 - 2\right)x^2 + x \tag{3.11}$$

en utilisant le changement de variable :

$$\begin{cases} X &= (d-1)^2x + d(d-1) \\ Y &= (d-1)^3y - (1-c)X/2 + b/2. \end{cases}$$

Le point d'abscisse $(d-1)(-2d^2+7d-1)/4$ *y est alors rationnel.*

3.5 Ajouter une seconde phase

Peter L. Montgomery [14] et Richard P. Brent [5] se sont très vite rendus compte que l'on pouvait, tout comme dans la méthode $p-1$, stopper la première phase plus tôt. On obtient ainsi un point $R = kP \neq O \bmod p$ qui va servir de base à une seconde phase.

En effet si $k = \prod_{p_i=2}^{p_i<B_1} p_i^{e_i}$, on peut raisonnablement penser que l'ordre de P divise kq ou q est un entier premier plus grand que B_1. Les secondes phases connues à ce jour reposent sur cette hypothèse et se concentrent sur la découverte de q.

Deux secondes phases sont décrites ici :

- La seconde phase "Classique".
- La seconde phase "Paradoxe des anniversaires".

3.5.1 La seconde phase "Classique"

Cette seconde phase consiste tout simplement à calculer sR pour tous les nombres premiers s compris entre B_1 et B_2 où $B_2 \gg B_1$. On peut cependant éviter de calculer sR par multiplication comme dans la première phase.

Le calcul se fait par récurrence en remarquant que si s_j et s_{j+1} sont deux entiers premiers qui se suivent, on a $s_{j+1}R = s_jR + (s_{j+1} - s_j)R$. Étant donné que la différence $s_{j+1} - s_j$ est toujours petite (elle est par exemple majorée par $2L = 144$ pour les nombres premiers inférieurs à 5.10^6), on peut tester très vite un grand nombre de points sR en préstockant kR pour tous les entiers k pairs compris entre 2 et $2L$.

procedure ClassicalStep2($R, \mathbf{E}_{a,b}, B_1, B_2, N$)
$g := 1$
EcModAdd($P, Q, \mathbf{E}_{a,b}, g, N$) calcule $P + Q \bmod N$ sur $\mathbf{E}_{a,b}$.
Si l'inversion est impossible, g est un facteur de N.
$P_1 :=$ EcModAdd($R, R, \mathbf{E}_{a,b}, g, N$)
for $k := 2, 3, \ldots, 100$ **while** $g = 1$ **do**
  $P_k :=$ EcModAdd($P_{k-1}, P_1, \mathbf{E}_{a,b}, g, N$)
end
if $g = 1$ **then**
  $p :=$ NextPrime(B_1)
  $R :=$ EcModMult($p, R, \mathbf{E}_{a,b}, g, N$)
end
$s :=$ NextPrime(p)
while $s < B_2$ **and** $g = 1$ **do**
  $R :=$ EcModAdd($R, P_{(s-p)/2}, \mathbf{E}_{a,b}, g, N$)
  $p := s$
  $s :=$ NextPrime(p)
end
return(g)
end ClassicalStep2

Algorithme 3 : La seconde phase classique.

Cette seconde phase nécessite donc
– $O(\log(B_2))$ opérations pour précalculer les points P_i.
– $O(\log(B_1))$ opérations pour calculer le point NextPrime$(B_1)R$.
– $O(\frac{B_2}{\log B_2} - \frac{B_1}{\log B_1})$ opérations pour tester tous les premiers entre B_1 et B_2.
On a ainsi un coût asymptotique de $O(B_2/\log B_2)$.

C'est cette seconde phase qui fut historiquement d'abord utilisée avec l'algorithme. Le plus grand facteur découvert avec celle-ci est un entier de 40 chiffres de $p(11279)$ par Arjen K. Lenstra sur une MasPar avec 2^{14} processeurs en allouant 11 processeurs par courbe.

3.5.2 La seconde phase "Paradoxe des anniversaires"

A la fin de la première phase, le point R obtenu engendre un sous groupe cyclique $< R >$ de la courbe $\mathbf{E}_{a,b}$ supposé avoir modulo p (p ; plus petit facteur premier de l'entier N à factoriser) un ordre q premier assez grand ($\simeq 10^9$). Plutôt que de tester $O(q/\log q)$ points s_iR comme dans la seconde phase classique, l'idée est ici de rechercher si parmi $r = O(\sqrt{q})$ points aléatoires $R_0, R_1, \ldots, R_{r-1}$ de $< R >$ calculés modulo N, deux points seraient égaux modulo p.

Cette idée était déjà celle sous-jacente à la méthode de factorisation rho-pollard [7]. Dans cet algorithme on calcule à partir de $a \in (\mathbf{Z}/N\mathbf{Z})^* \setminus \{1\}$ la suite définie par

$$x_0 = a \quad \text{et} \quad \forall i \in \mathbf{N}^*,\ x_{i+1} = x_i^2 + 1 \bmod N.$$

Cette suite considérée sur $\mathbf{Z}/p\mathbf{Z}$ étant pseudo-aléatoire, on recherche l'indice i tel, que $x_{2i} \equiv x_i \bmod p$. On peut alors montrer que i est alors de l'ordre de grandeur $\sqrt{p}$.

Dans le cas des courbes elliptiques, un tel schéma n'est pas réalisable tout simplement parce que seules l'addition de points et la multiplication par des entiers sont définies. La multiplication de deux points n'y a pas de sens. En outre on ne connaît pas de fonction déterministe pseudo-aléatoire de $< R >$ dans $< R >$.

Un moindre mal peut alors être la génération proposée par R. P. Brent [5] :

$$R_0 = R \text{ et } \forall j \in \{1, 2, 3, \ldots, r-1\}\ R_{j+1} = \begin{cases} 2R_j & \text{avec probabilité } \frac{1}{2}. \\ 2R_j + R & \text{avec probabilité } \frac{1}{2}. \end{cases}$$

P. L. Montgomery propose pour sa part cette autre génération de points R_i [14] :

$$R_0 = R \text{ et } \forall j \in \{1, 2, 3, \ldots, r-1\},\ R_j = P(m_j)R$$

où $\{m_j\}_{j=1}^{r-1}$ est un sous ensemble de $\mathbf{N}$ et $P(X)$ est un polynôme à coefficients entiers tel que $P(X) \pm P(Y)$ ait beaucoup de petits diviseurs. Un tel polynôme peut être par exemple $P(X) = X^k$ avec k très friable car $(P(X) + P(Y))(P(X) - P(Y)) = X^{2k} - Y^{2k}$ a autant de facteurs polynomiaux irréductibles que $2k$ possède de diviseurs entiers.

Dans ces deux suites de points, rechercher s'il existe un indice i tel que $R_{2i} = R_i \bmod p$ est inutile car ici l'existence de deux indices i et j tels que $R_i = R_j$ n'implique plus $R_{i+1} = R_{j+1}$. On doit donc comparer deux à deux les points générés au prix de $r(r-1)/2$ opérations. En comparant seulement les abscisses de ces points on

augmente alors sensiblement la probabilité de succès car non seulement on recherche deux indices i et j tels que $R_i = R_j$ mais aussi tels que $R_i = -R_j$ (l'inverse d'un point $P(x, y)$ sur $\mathbf{E}_{a,b}$ est le point $-P(x, -y)$).

La difficulté est alors de trouver si deux valeurs parmi les abscisses $x_0, x_1, \ldots, x_{r-1}$ modulo N sont égales modulo p. Les techniques usuelles de hachages ou de tris sont ici caduques et on doit se résoudre à calculer la quantité

$$d = \prod_{0 \leq i < j < r} (x_i - x_j) \bmod N \tag{3.12}$$

puis $\gcd(d, N)$ en espérant retrouver p.

Avec cette méthode, on teste non seulement pendant leurs générations si les r points R_i sont nuls modulo p mais aussi s'il en est de même pour les $r(r-1)/2$ autres points $R_i \pm R_j$. Au total, plus de multiples de R sont calculés ici que pendant la seconde phase classique même si chaque point n'est pas forcément le produit de R par un nombre premier.

BirthdayStep2() est une implantation naïve de cet algorithme.

```
procedure BirthdayStep2(R, E_{a,b}, r, N)
g := 1
Q := R
# Absc(Q) retourne l'abscisse du point Q.
x_0 := Absc(Q)
for k := 1, 2, 3, ..., r − 1 while g = 1 do
  Q := EcModAdd(Q, Q, E_{a,b}, g, N)
  # Random() retourne aléatoirement 0 ou 1.
  if Random() = 1 and g = 1 then
    Q := EcModAdd(Q, R, E_{a,b}, g, N)
  end
  x_k := Absc(Q)
end
if g = 1 then
  d := 1
  for k := 0, 1, ..., r − 2 do
    for l := k, k + 1, ..., r − 1 do
      d := d(x_k − x_l) mod N
    end
  end
  g := gcd(d, N)
end
return(g)
end BirthdayStep2
```

Algorithme 4 : Le paradoxe des anniversaires.

Dans une variante de cet algorithme, on considère deux familles de points $\{R_i\}_{i=0}^{r_1-1}$ et $\{S_j\}_{j=0}^{r_2-1}$ et on recherche s'il existe deux indices i et j tels que

$$R_i = \pm S_j \quad (0 \leq i < r_1,\ 0 \leq j < r_2).$$

Pour ce faire on calcule une quantité analogue à (3.12) :

$$\gcd\left(\prod_{0\leq i<r_1}\prod_{0\leq j<r_2}(x_i^r - x_j^s), N\right) \tag{3.13}$$

où x_i^r est l'abscisse du point R_i et x_j^s est l'abscisse du point S_j.

Dans la version principale, la probabilité de succès est alors la même que celle de deux personnes parmi r aient la même date anniversaire sur une planète ayant q jours par an.

En général, pour $r \ll q$, cette probabilité vaut

$$Pr = 1 - \prod_{j=1}^{r-1}(1 - \frac{j}{q}) \simeq 1 - e^{\frac{-r^2}{2q}}.$$

Lorsque l'on ne considère que les abscisses, la probabilité de succès devient alors

$$Pr \simeq 1 - e^{\frac{-r^2}{q}}$$

et celle-ci est supérieure à $\frac{1}{2}$ pour

$$\boxed{r > \sqrt{\ln(2)q}}$$

Ainsi, la réalisation naïve d'une telle seconde phase coûte :

1. $O(\sqrt{q})$ opérations pour générer les points R_i.
2. $O(q)$ opérations pour comparer les $\sqrt{q}$ abscisses obtenues.

Il est cependant possible grâce à une arithmétique polynomiale asymptotiquement rapide de ramener ce deuxième coût à $O(\sqrt{q}(\log q)^2)$ opérations.

C'est avec cette seconde phase que le facteur de 43 chiffres fut découvert.

Chapitre 4

Algorithmique polynomiale

L'implantation de la seconde phase "paradoxe des anniversaires" nécessite une arithmétique polynomiale efficace. Ce chapitre résume quelques-un des algorithmes les plus rapides sur $\mathbf{Z}/N\mathbf{Z}$ pour

- multiplier deux polynômes,
- diviser deux polynômes,
- évaluer un polynôme en de nombreux points.

Comme précédemment, l'unité de base de complexité (u.b) est la multiplication modulo N (Additions et soustractions modulaires sont négligées).

4.1 La multiplication polynomiale

Nous allons citer 3 méthodes de complexités asymptotiquement différentes pour multiplier deux polynômes $F(X) = \sum_{i=0}^{n_f} f_i X^i$ et $G(X) = \sum_{i=0}^{n_g} g_i X^i$.

4.1.1 La multiplication usuelle

Elle consiste simplement à calculer le produit par la formule

$$F(X)G(X) \bmod N = \sum_{i=0}^{n_f+n_g} \sum_{k+l=i} f_k g_l X^i \bmod N.$$

Chaque coefficient du résultat s'obtient alors directement à partir des coefficients des opérandes comme le montre l'algorithme 5.

procedure PolyUsualMult($F(X), G(X)$)
for $i := 0, 1, \ldots, n_f + n_g$ **do** $h_i := 0$ **end**
for $i := 0, 1, \ldots, n_f$ **do**
  for $j := 0, 1, \ldots, n_g$ **do**
    $h_{i+j} := h_{i+j} + f_i g_j$
  end
end
return$(H(X) = \sum_{i=0}^{n_f+n_g} h_i X^i)$
end PolyUsualMult

Algorithme 5 : Multiplication polynomiale usuelle.

La méthode a pour coût

$$\boxed{(n_f+1)(n_g+1)\ \text{u.b}}.$$

On l'utilise essentiellement lorsque n_f et n_g sont d'ordres de grandeur différents.

4.1.2 L'algorithme de Karatsuba

On peut faire mieux avec l'algorithme "divide and conquer" de Karatsuba [7]. On suppose pour cela que les degrés de $F(X)$ et $G(X)$ sont égaux ; $n_f = n_g = n$. Écrivons tout d'abord :

$$\begin{aligned} F(X) &= F_-(X) + X^\mu F_+(X), \\ G(X) &= G_-(X) + X^\mu G_+(X) \end{aligned}$$

avec les degrés de $F_-(X)$ et $G_-(X)$ au plus égaux à μ où $\mu = \lfloor (n+1)/2 \rfloor$ et les degrés de $F_+(X)$ et $G_+(X)$ au plus égaux à ν où $\nu = \mu + 1$ si n est pair et $\nu = \mu$ si n est impair.

Le produit recherché s'exprime alors en fonction
- de $H_-(X) = F_-(X)G_-(X)$,
- de $H_+(X) = F_+(X)G_+(X)$,
- de $H_\pm(X) = (F_-(X) + F_+(X))(G_-(X) + G_+(X))$.

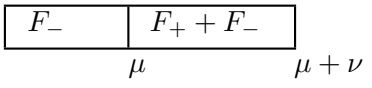
Il s'écrit :

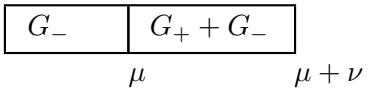
$$(FG)(X) \bmod N = H_-(X) + X^\mu (H_\pm - H_- - H_+)(X) + X^{2\mu} H_+(X) \bmod N.$$

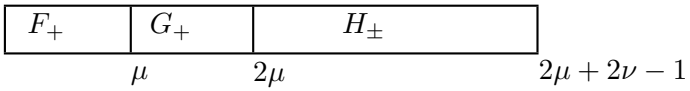
La routine PolyKaratsubaMult() est une implantation de cet algorithme. Elle se sert de $G(X)$ (qui est à part le coefficient g_n détruit à l'issue) comme buffer intermédiaire mais cette complication permet en contre partie de n'utiliser aucun autre polynôme auxilliaire.

Elle procède ainsi pour calculer $H(X) = F(X)G(X)$:

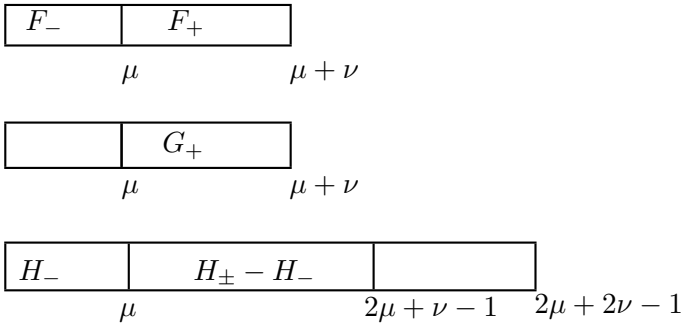
1. Les μ premiers coefficients de F_+ et G_+ sont copiés dans les $2\mu^{eme}$ premiers coefficients de H. F_- et G_- sont alors additionnés à leurs homologues F_+ et G_+ à la place de ces derniers et on effectue récursivement le produit $H_\pm$ dans les coefficients restants de H.



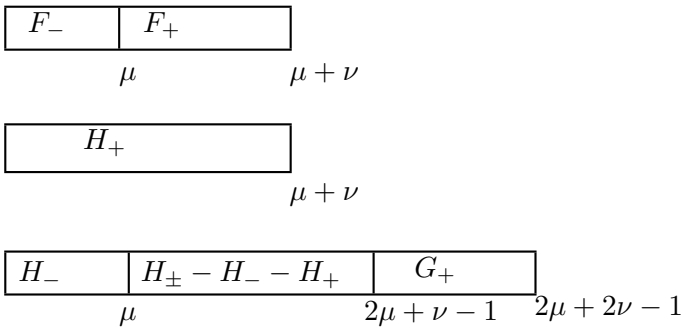




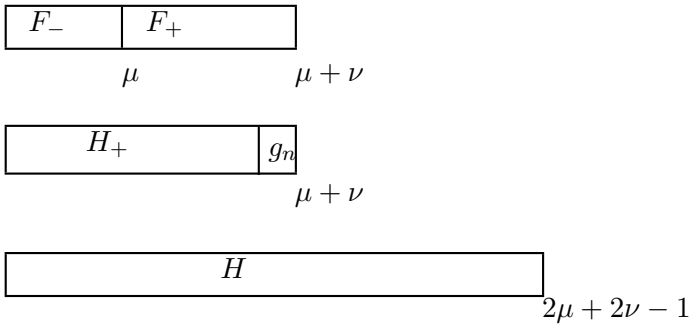
2. F et G sont alors entièrement restaurés à l'aide des sauvegardes de F_+ et G_+ dans H (il est à noter que si $\nu = \mu+1$, la restauration de seulement μ coefficients est suffisante car le ν^{eme} reste inchangé pendant la multiplication). L'évaluation du produit H_- est alors effectuée dans les $2\mu-1$ premiers coefficients de H et le résultat est soustrait à $H_\pm$. Les $\mu+\nu-1$ coefficients du polynôme obtenu sont alors "décalés" à la μ^{eme} position dans H en additionnant les $\mu-1$ premiers aux derniers coefficients de H_-.



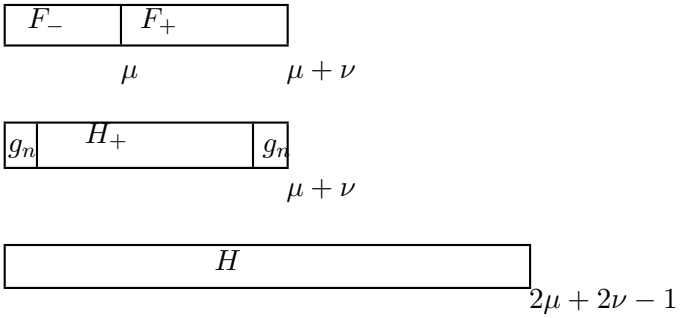
3. Les coefficients de G_+ sont alors transférés dans les ν coefficients libres de H et le produit H_+ est alors réalisé dans G. On soustrait alors à $H_\pm - H_-$ le polynôme H_+ obtenu et on additionne les $\nu-1$ premiers coefficients de H_+ aux derniers coefficients de $H_\pm - H_- - H_+$.



4. Si $\nu = \mu$, le produit H_+ occupe $2\mu-1$ coefficients et le dernier coefficient de G est resté inchangé. Il suffit donc de transférer les ν derniers coefficients de H_+ pour obtenir le produit H recherché.



5. Sinon, $\nu = \mu+1$ et le dernier coefficient de G a été "écrasé" par H_+. Cependant, ce dernier coefficient g_n est aussi le dernier coefficient de H, et il suffit alors de le sauvegarder dans g_0, de transférer les ν derniers coefficients de H_+ dans les derniers coefficients de H et ensuite de copier g_0 à la n^{eme} position de G.



On peut vérifier *a posteriori* que $F(X)$ est bien inchangé, que $G(X)$ est détruit sauf son dernier coefficient et surtout que $H(X) = F(X)G(X)$.

Cette façon de faire est sensiblement plus compliquée qu'une implantation standard mettant en œuvre un buffer de la taille du produit $F(X)G(X)$. Elle permet cependant de manipuler des polynômes de degrés élevés avec une augmentation du temps de calcul quasi nulle.

L'algorithme 6 est une écriture plus formelle de PolyKaratsubaMult().

procedure PolyKaratsubaMult($R(X), F(X), G(X)$)
# degree(F) retourne le degré de F.
if degree(F) $= 0$ **then**
  $r_0 := f_0.g_0$
else if degree(F) $= 1$ **then**
  $r_0 := f_0 + f_1 \; ; r_2 := g_0 + g_1$
  $r_1 := r_0.r_2$
  $r_0 := f_0.g_0 \; ; r_2 := f_1.g_1$
  $r_1 := r_1 - r_0 \; ; r_1 := r_1 - r_2$
else
  $\mu := \lfloor \frac{\text{degree}(F)+1}{2} \rfloor \; ; \nu := \text{degree}(F) + 1 - \mu$
  $\sum_{i=0}^{\mu-1} r_i X^i := \sum_{i=0}^{\mu-1} f_{\mu+i} X^i$
  $\sum_{i=0}^{\mu-1} r_{\mu+i} X^i := \sum_{i=0}^{\mu-1} g_{\mu+i} X^i$
  $\sum_{i=0}^{\mu-1} f_{\mu+i} X^i := \sum_{i=0}^{\mu-1} f_{\mu+i} X^i + \sum_{i=0}^{\mu-1} f_i X^i$
  $\sum_{i=0}^{\mu-1} g_{\mu+i} X^i := \sum_{i=0}^{\mu-1} g_{\mu+i} X^i + \sum_{i=0}^{\mu-1} g_i X^i$;
  PolyKaratsubaMult($\sum_{i=0}^{2\nu-1} r_{2\mu+i} X^i , \sum_{i=0}^{\nu-1} f_{\mu+i} X^i, \sum_{i=0}^{\nu-1} g_{\mu+i} X^i$)
  $\sum_{i=0}^{\mu-1} f_{\mu+i} X^i := \sum_{i=0}^{\mu-1} r_i X^i$
  $\sum_{i=0}^{\mu-1} g_{\mu+i} X^i := \sum_{i=0}^{\mu-1} r_{\mu+i} X^i$
  PolyKaratsubaMult($\sum_{i=0}^{2\mu-1} r_i X^i, \sum_{i=0}^{\mu-1} f_i X^i, \sum_{i=0}^{\mu-1} g_i X^i$)
  $\sum_{i=0}^{2\mu-2} r_{2\mu+i} X^i := \sum_{i=0}^{2\mu-2} r_{2\mu+i} X^i - \sum_{i=0}^{2\mu-2} r_i X^i$
  $\sum_{i=0}^{\mu-2} r_{\mu+i} X^i := \sum_{i=0}^{\mu-2} r_{\mu+i} X^i + \sum_{i=0}^{\mu-2} r_{2\mu+i} X^i$
  $\sum_{i=0}^{\nu-1} r_{2\mu-1+i} X^i := \sum_{i=0}^{\nu-1} r_{3\mu-1+i} X^i$
  $\sum_{i=0}^{\nu-1} r_{2\mu+\nu-1+i} X^i := \sum_{i=0}^{\nu-1} g_{\mu+i} X^i$
  PolyKaratsubaMult($\sum_{i=0}^{2\nu-1} g_i X^i, \sum_{i=0}^{\nu-1} f_{\mu+i} X^i, \sum_{i=0}^{\nu-1} r_{2\mu+\nu-1+i} X^i$)
  $\sum_{i=0}^{\nu+\mu-2} r_{\mu+i} X^i := \sum_{i=0}^{\nu+\mu-2} r_{\mu+i} X^i - \sum_{i=0}^{\nu+\mu-2} g_i X^i$
  $\sum_{i=0}^{\nu-2} r_{2\mu+i} X^i := \sum_{i=0}^{\nu-2} r_{2\mu+i} X^i + \sum_{i=0}^{\nu-2} g_i X^i$
  if $\nu > \mu$ **do**
    $\sum_{i=0}^{\nu-\mu-1} g_i X^i := \sum_{i=0}^{\nu-\mu-1} r_{3\mu+\nu-1+} X^i$
    $\sum_{i=0}^{\nu-1} r_{2\mu+\nu-1+i} X^i := \sum_{i=0}^{\nu-1} g_{\nu-1+i} X^i$
    $\sum_{i=0}^{\nu-\mu-1} g_{2\mu+i} X^i := \sum_{i=0}^{\nu-\mu-1} g_i X^i$
  else
    $\sum_{i=0}^{\nu-1} r_{2\mu+\nu-1+i} X^i := \sum_{i=0}^{\nu-1} g_{\nu-1+i} X^i$
  end
end
end PolyKaratsubaMult

Algorithme 6 : Multiplication polynomiale de Karatsuba.

Le coût de multiplication $M(n)$ de 2 polynômes de degré n vérifie :

$$\begin{cases} M(n) = M(\mu) + 2M(\mu+1) & \text{si } n = 2\mu, \\ M(n) = 3M(\mu) & \text{si } n = 2\mu + 1. \end{cases}$$

Ainsi lorsque $n = 2^k - 1$, on a $M(2^k - 1) = 3^k$ et donc asymptotiquement le coût vaut

$$\boxed{n^{\log_2 3} \text{ u.b}}.$$

4.1.3 La multiplication polynomiale par FFT

La multiplication polynomiale par FFT réalise habituellement le produit de deux polynômes $F(X)$ et $G(X)$ de degré n définis sur un corps en $O(n\log(n))$ u.b [7]. La formulation initiale de cet algorithme est inapplicable sur $\mathbf{Z}/N\mathbf{Z}$. Une solution est la réalisation de la multiplication modulo suffisamment de "petits" nombres premiers pour lesquels on utilise l'algorithme usuel. Les coefficients modulo N sont ensuite obtenus par le théorème chinois [15].

La transformée de Fourier dans GF(p)

On rappelle ici quelques définitions et résultats du livre de H. J. NussBaumer [17] relatifs à la transformée de Fourier dans $\mathbf{GF}(p)$.

ω étant un élément de $\mathbf{GF}(p)$ et n un entier non nul, on définit la transformée de Fourier sur $\mathbf{GF}(p)^n$ ainsi

Définition 6 *La transformée de Fourier $\mathcal{F}$ est définie par*

$$\begin{array}{ccc} \mathbf{GF}(p)^n & \rightarrow & \mathbf{GF}(p)^n \\ [a_0, a_1, \ldots, a_{n-1}] & \rightarrow & \mathcal{F}(a) = [\hat{a}_0, \hat{a}_1, \ldots, \hat{a}_{n-1}] \end{array}$$

où

$$\hat{a}_i = \sum_{j=0}^{n-1} \omega^{ij} a_j.$$

Pour pouvoir définir la transformée de Fourrier inverse, il est nécessaire que ω soit une racine n-ième principale de l'unité.

Définition 7 *Un élément $\omega \neq 0, 1$ de $\mathbf{GF}(p)$ est une racine n-ième principale de l'unité si*

$$\begin{array}{l} \omega^n = 1, \\ \forall i \in \{1, 2, \ldots, n-1\} \ \sum_{j=0}^{n-1} \omega^{ij} = 0. \end{array} \tag{4.1}$$

Lorsque $n = p - 1$, on dit alors que ω est une racine primitive de $\mathbf{GF}(p)$.

Pour de tels ω, on a alors

Théorème 9 *Si ω est une racine n-ième principale de l'unité dans $\mathbf{GF}(p)$, la transformée de Fourier inverse de $\mathcal{F}$ vérifie*

$$\begin{array}{ccc} \mathbf{GF}(p)^n & \rightarrow & \mathbf{GF}(p)^n \\ [\hat{a}_0, \hat{a}_1, \ldots, \hat{a}_{n-1}] & \rightarrow & \mathcal{F}^{-1}(a) = [a_0, a_1, \ldots, a_{n-1}] \end{array}$$

où

$$a_i = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{-ij} \hat{a}_j.$$

Dans $\mathbf{GF}(p)$, il n'existe cependant pas de racine n-ième principale de l'unité pour tout n.

Théorème 10 *Il existe des racines n-ièmes principales de l'unité dans $\mathbf{GF}(p)$ si et seulement si n est un diviseur de $p - 1$. ζ étant une racine primitive de $\mathbf{GF}(p)$, $\omega = \zeta^{\frac{p-1}{n}}$ est alors une telle racine.*

La détermination d'une racine primitive de $\mathbf{GF}(p)$ est donc primordiale pour pouvoir effectuer la transformée de Fourier. La recherche de telles racines est grandement facilitée par le théorème 11 :

Théorème 11 *ζ est une racine primitive de* $\mathbf{GF}(p)$ *si et seulement si pour tous les diviseurs premiers q de $p-1$, $\zeta^{\frac{p-1}{q}} \neq 1 \bmod p$.*

Une fois déterminés les facteurs q de $p-1$, il suffit donc d'élever aux puissances $\frac{p-1}{q}$ les entiers 2,3,...jusqu'à ce qu'un de ces entiers vérifie le théorème 11.

Lorsqu'elle est définie pour $n = 2^k$, l'intérêt de la transformée de Fourier est qu'il existe un algorithme appelé FFT (Fast Fourier Transform) qui permet de la calculer à l'aide $\frac{n}{2} \log n$ multiplications.

Si on note $a(X)$ le polynôme $\sum_{i=0}^{n-1} a_i X^i$, on peut voir $\mathcal{F}(a)$ comme l'évaluation du polynôme $a(X)$ en les n puissances de ω :

$$[a(1), a(\omega), \ldots, a(\omega^{n-1})].$$

Notons maintenant $c_0, c_1, \ldots, c_{n-1}$, une permutation des puissances de ω que nous définirons plus loin et définissons les quantités q_{lm} pour $0 \leq m \leq k$ et l un entier multiple de 2^m dans l'intervalle $0 \leq l \leq 2^k - 1$ par

$$q_{lm} = \prod_{j=l}^{l+2^m-1} (X - c_j).$$

On a alors

$$q_{lm} = q_{l,m-1} q_{l+2^{m-1},m-1},$$

et surtout les formules récurrentes :

$$\begin{cases} a(X) \bmod q_{l,m-1} & = & (a(X) \bmod q_{lm}) \bmod q_{l,m-1}, \\ a(X) \bmod q_{l+2^{m-1},m-1} & = & (a(X) \bmod q_{lm}) \bmod q_{l+2^{m-1},m-1}. \end{cases} \tag{4.2}$$

Ainsi, en calculant $r_{0k} = a(X) \bmod q_{0k}$, on peut avoir $r_{0,k-1} = a(X) \bmod q_{0,k-1}$ et $r_{2^{k-1},k-1} = a(X) \bmod q_{2^{k-1},k-1}$ par $r_{0k} \bmod q_{0,k-1}$ et $r_{0k} \bmod q_{2^{k-1},k-1}$.

Il suffit de réitérer le procédé pour obtenir finalement

$$r_{l0} = a(c_l) \text{ pour } 0 \leq l < 2^k.$$

Cette façon de faire les divisions grâce à (4.2) est déjà un gain mais on peut faire mieux en choisissant une permutation des puissances de ω qui garantisse l'obtention de polynômes q_{lm} de la forme $X^{2^m} - \omega^s$ pour un s obtenu par le théorème 12.

Théorème 12 *Soit $n = 2^k$ et ω une racine n-ième principale de l'unité. Si pour $0 \leq j < 2^k$, $[d_0 d_1 \ldots d_{k-1}]$ est la représentation binaire de j et si rev(j) est l'entier dont la représentation binaire est $[d_{k-1} d_{k-2} \ldots d_0]$ alors pour $c_j = \omega^{rev(j)}$ on a*

$$q_{lm} = \prod_{j=l}^{l+2^m-1} (X - c_j) = X^{2^m} - \omega^{rev(l/2^m)}.$$

La division d'un polynôme $b(X)$ de degré $2t-1$ par $X^t - c$ est alors très facilement calculable :

Théorème 13 *Si* $b(X) = \sum_{j=0}^{2t-1} b_j X^j$ *et* c *est une constante, alors le reste de* $b(X)$ *par* $X^t - c$ *est*

$$r(X) = \sum_{j=0}^{t-1} (a_j + c a_{j+t}) X^j.$$

L'algorithme FFT découle de ces résultats.

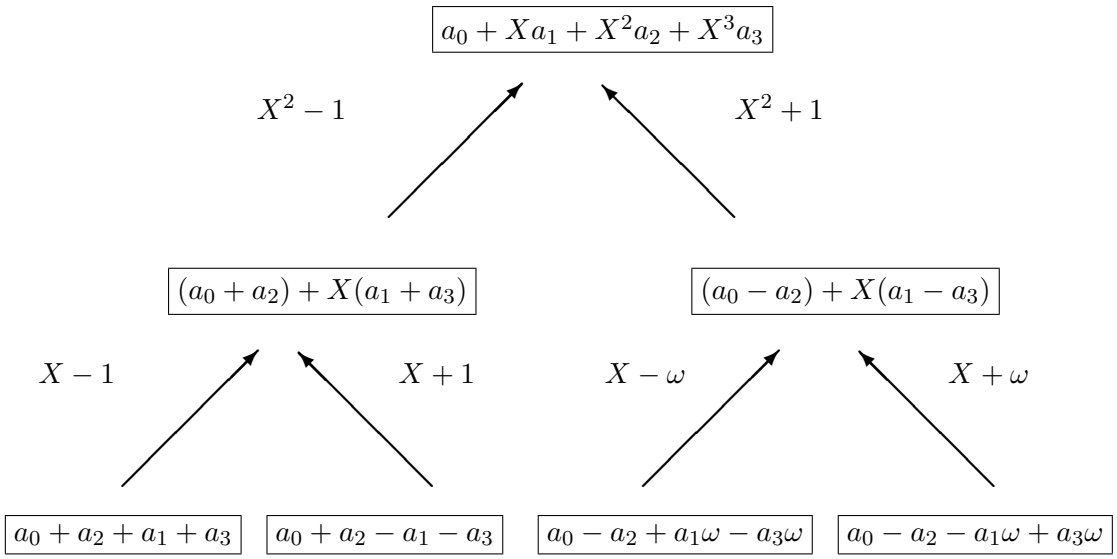


Schéma 1 : Application de la FFT à un vecteur de taille 4.

Lorsque l'on doit implanter la FFT, on travaille directement avec les coefficients. On réalise alors les mêmes opérations organisées autrement. Le schéma 2 reprend ainsi les calculs du schéma 1. La valeur au bout d'une flèche est le coefficient par lequel est multiplié l'élément à son origine.

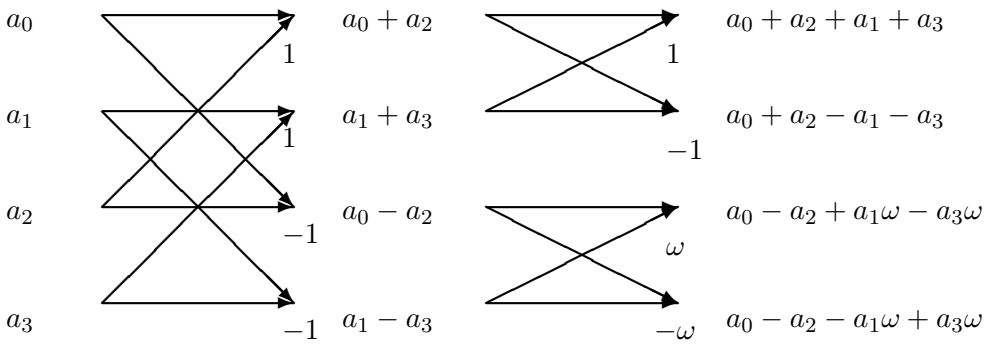


Schéma 2 : Une autre façon de réaliser la FFT.

La routine FFT montre comment faire de tels calculs dans le cas général.

```
procedure FFT(a, k, p, ω)
for i := 0, ..., 2^k − 1 do
  r[i] := a[i]
end
f := 1 ; l := 2^{k−1}
for i := 0, 1, ..., k − 1 do
  for m := 0, ..., f − 1 do
    # rev(m) retourne l'entier de représentation
    # binaire l'inverse de celle de m.
    d := rev(m)
    for n := 0, ..., l − 1 do
      t := w^d.r[m.2^{k−i} + n + l] mod p
      r[m.2^{k−i} + n + l] := r[m.2^{k−i} + n] − t mod p
      r[m.2^{k−i} + n] := r[m.2^{k−i} + n] + t mod p
  end
f := 2f
l := l/2
end
for i := 0, ..., 2^k − 1 do
  b[i] := r[rev(i)]
end
return(b)
end FFT
```

Algorithme 7 : Transformée de Fourier rapide sur **GF**(p).

Dans cet algorithme, on effectue k fois lf multiplications et $2k$ fois lf additions ou soustractions. Or, à chaque étape, $lf = 2^{k-1}$. La FFT réalise donc $k2^{k-1}$ multiplications et deux fois plus de soustractions/additions lorsqu'elle est appliquée à un vecteur de taille 2^k.

La FFT inverse peut être obtenue par le même algorithme en utilisant ω^{-1} à la place de ω et en multipliant par n^{-1} le vecteur obtenu par FFT(). De plus, lorsque l'on se sert de la FFT pour effectuer des convolutions, il n'est pas nécessaire d'inverser le vecteur obtenu. Une FFT inverse adaptée permet alors de retrouver le bon résultat.

Multiplication par FFT sur GF(p)

La multiplication modulo $X^n - 1$ de 2 polynômes $F(X)$ et $G(X)$ de degré n (n est une puissance de 2) par FFT sur **GF**(p) comporte 3 étapes :

1. Transformation des polynômes par FFT.
2. Multiplication terme à terme des transformés.
3. Transformation inverse du produit.

Le produit de $F(X)$ et $G(X)$ peut en fait se ramener à ce que l'on appelle une convolution circulaire.

Définition 8 *Si $f = [f_0, f_1, \ldots, f_{n-1}]$ et $g = [g_0, g_1, \ldots, g_{n-1}]$ sont deux vecteurs de taille n de $\mathbf{GF}(p)$, alors leur convolution circulaire $f \otimes g$ est le vecteur $h = [h_0, h_1, \ldots, h_{n-1}]$ de longueur n avec*

$$h_k = \sum_{i+j \equiv k \bmod n} f_i g_j \quad \forall k \in \{0, 1, \ldots, n-1\}.$$

La propriété essentielle de la transformée de Fourier, c'est qu'elle permet de faire des convolutions circulaires très facilement.

Théorème 14 *Si $f = [f_0, f_1, \ldots, f_{n-1}]$ et $g = [g_0, g_1, \ldots, g_{n-1}]$ sont deux vecteurs de taille n de $\mathbf{GF}(p)$ de transformées de Fourier respectives $\hat{f} = [\hat{f_0}, \hat{f_1}, \ldots, \hat{f_{n-1}}]$ et $\hat{g} = [\hat{g_0}, \hat{g_1}, \ldots, \hat{g_{n-1}}]$, alors la transformée de Fourier de $h = f \otimes g$ est le produit de vecteur terme à terme*

$$\hat{h} = [\hat{f_0}.\hat{g_0},\ \hat{f_1}.\hat{g_1},\ \ldots,\ \hat{f_{n-1}}.\hat{g_{n-1}}].$$

Or, si

$$F(X) = \sum_{i=0}^{n-1} f_i X^i, G(X) = \sum_{i=0}^{n-1} g_i X^i, H(X) = \sum_{i=0}^{n-1} h_i X^i,$$

alors il est immédiat que

$$h = f \otimes g \Longleftrightarrow H(X) \equiv F(X)G(X) \bmod (X^n - 1).$$

Les convolutions circulaires de longueur n où n est une puissance de 2 sont donc de véritables multiplications polynomiales modulo $X^n - 1$.

Dans le cas général, lorsque $F(X)$ et $G(X)$ sont de degrés quelconques, on les assimile à des polynômes de degré n,la plus petite puissance de 2 immédiatement supérieure à leurs degrés, et de coefficients supérieurs nuls.

Lorsque l'on désire un produit $F(X)G(X)$ exact, il est nécessaire que degree(F)+degree(G) soit strictement inférieur à n. On peut de plus remarquer que si degree(F)+degree(G)=n, il suffit pour obtenir un résultat correct après avoir appliqué la méthode, de calculer coefficient constant et de plus haut degré du produit directement.

Le coût des FFT et de la FFT inverse est de $\frac{3}{2} n \log n$ multiplications modulo p, celui du produit terme à terme est de n multiplications modulo p.

Multiplication polynomiale sur $\mathbf{Z}/N\mathbf{Z}$

Lorsque N est composite, on ne sait pas en règle générale trouver d'élément ω de $\mathbf{Z}/N\mathbf{Z}$ vérifiant les relations (4.1) sans connaître la factorisation de N.

P. L. Montgomery et Silverman [15] réalisent des convolutions modulo N en exécutant plusieurs de ces dernières modulo de petits nombres premiers et en utilisant le théorème du reste chinois pour obtenir un résultat modulo N.

Supposons que l'on veuille multiplier deux polynômes $F(X)$ et $G(X)$ modulo N et $X^n - 1$ avec les degrés de $F(X)$ et $G(X)$ strictement inférieurs à n où n est une puissance de 2. On choisit alors K nombres premiers distincts tels que $p_i \equiv 1 \bmod n$ avec

$$P = \prod_{i=1}^{K} p_i > \frac{nN^2}{1-\epsilon}. \tag{4.3}$$

et où ϵ dépend de la précision de l'arithmétique flottante. Sachant que chaque coefficient du produit sera plus petit que nN^2 dans $\mathbf{Z}$, les coefficients modulo P seront en fait les coefficients sur $\mathbf{Z}$ du produit que l'on peut ensuite réduire modulo N pour avoir le résultat attendu.

En réduisant les coefficients de $F(X)$ et $G(X)$ modulo chaque p_i, on réalise les K produits polynomiaux modulo ces petits nombres premiers par FFT :

$$F(X)G(X) \equiv \sum_{j=0}^{n-1} h_{ij}X^j \bmod p_i, X^n - 1. \tag{4.4}$$

Les entiers p_i sont choisis suffisamment petits pour tirer parti de l'arithmétique simple précision des ordinateurs. Afin de pouvoir réaliser les multiplications modulaires il serait donc nécessaire qu'un tel nombre premier p soit inférieur à 2^{15}. Ils ne doivent néanmoins pas être trop nombreux pour ne pas augmenter inconsidérément le coût final. Une solution intermédiaire est de tirer parti de la forme particulière de $p = e2^f + 1$ avec $(e, f) \in \mathbf{N}$ pour pouvoir utiliser des nombres premiers $p < 2^{30}$.

Je m'explique ; supposons que l'on veuille multiplier 2 entiers x et y ($0 \leq x, y < p$). On calcule d'abord

$$\begin{cases} u_e &= xy \bmod e, \\ u_f &= xy \bmod 2^f \text{ en tronquant à } f \text{ bits le produit obtenu modulo } 2^{32}. \end{cases}$$

On obtient alors le produit $xy \bmod e2^f$ grâce au théorème chinois par

$$\begin{aligned} y &= (u_f - u_e)(1/e \bmod 2^f) \bmod 2^f, \\ u &= u_e + ye. \end{aligned}$$

On calcule $q = \lfloor (xy)/p \rfloor$ (division effectuée en arithmétique flottante). On a ainsi $xy = q(e.2^f) + u$. Reste alors à avoir $r = xy \bmod p$ en remarquant que :

– Si $u \geq q$, alors $xy = q(e.2^f + 1) + u - q$ et $r = u - q$.

– Si $u < q$, alors $xy = (q-1)(e.2^f + 1) + p - (q - u)$ et $r = p + u - q$.

Cette façon de faire ne nécessite que 4 multiplications, 3 divisions entières et une division flottante, soit un coût moindre qu'une multiplication en double précision réduite ensuite modulo p. La programmation de cette astuce est de plus aisée.

D'autre part, la détermination d'une racine primitive dans $\mathbf{GF}(p_i)$ et ensuite d'une racine n^{eme} principale de l'unité nécessaire au calcul de (4.4) se fait facilement étant donné la petite taille des p_i.

Par contre, vu la grande taille de $P \simeq N^2$, l'exécution du théorème chinois modulo P est beaucoup trop coûteuse et on préfère effectuer les opérations modulo N.

Comme $h_j \equiv h_{ij} \bmod p_i$, le théorème chinois montre que

$$h_j \equiv \sum_{i=1}^{K} \frac{P}{p_i} y_{ij} \bmod P,$$

avec

$$y_{ij} \equiv \left(\frac{P}{p_i}\right)^{-1} h_{ij} \bmod p_i.$$

Ce qui peut encore s'écrire

$$h_j = \sum_{i=1}^{K} \frac{P}{p_i} y_{ij} - k_j P \text{ avec } k_j = \left\lfloor \sum_{i=1}^{K} \frac{y_{ij}}{p_i} \right\rfloor . \tag{4.5}$$

L'évaluation de k_j peut se faire en arithmétique flottante moyennant quelques précautions. Si on choisit ϵ de telle façon que l'erreur due aux arrondies dans (4.5) ne dépasse pas $\frac{\epsilon}{2}$, on a d'après (4.3) :

$$\begin{array}{rcl} \sum_{i=1}^{K} \frac{y_{ij}}{p_i} + \frac{\epsilon}{2} & = & \sum_{i=1}^{K} \frac{(P/p_i)y_{ij}}{P} + \frac{\epsilon}{2}, \\ & = & \frac{(h_j + k_j P)}{P} + \frac{\epsilon}{2}, \\ & \in & [k_j + \frac{\epsilon}{2}, k_j + \frac{n(N-1)^2}{P} + \frac{\epsilon}{2}], \\ & \in & [k_j + \frac{\epsilon}{2}, k_j + (1-\epsilon) + \frac{\epsilon}{2}], \\ & \in & [k_j + \frac{\epsilon}{2}, k_j + 1 - \frac{\epsilon}{2}]. \end{array}$$

Cette quantité diffère alors d'au plus $\frac{\epsilon}{2}$ du plus proche entier.

Une formule plus informatique pour h_j est aussi

$$h_j \equiv \sum_{i=1}^{K} \left(\frac{P}{p_i} \bmod N \right) y_{ij} + ((-P) \bmod N) \left\lfloor \sum_{i=1}^{K} \frac{y_{ij}}{p_i} + \frac{\epsilon}{2} \right\rfloor . \tag{4.6}$$

où les coefficients $(P/p_i)^{-1} \bmod p_i, -P \bmod N$ et $P/p_i \mod N$ peuvent être précalculés et stockés par avance.

Tous ces éléments sont réunis dans l'algorithme 8.

procedure PolyFFTMult($F(X), G(X), p, \omega, K$)
$k := \lfloor \log_2(\text{degree}(F) + 1) \rfloor + 2$
for $i := 1, 2, \ldots, K$ **do**
  for $j := 0, 1, \ldots, \text{degree}(F)$ **do** $a_{ij} := f_j \bmod p$ **end**
  for $j := \text{degree}(F) + 1, \ldots, 2^k - 1$ **do** $a_{ij} = 0$ **end**
  $a_i := \text{FFT}(a_i, k, p_i, \omega_i)$
  for $j := 0, 1, \ldots, \text{degree}(F)$ **do** $b_{ij} := g_j \bmod p$ **end**
  for $j := \text{degree}(F) + 1, \ldots, 2^k - 1$ **do** $b_{ij} = 0$ **end**
  $b_i := \text{FFT}(b_i, k, p_i, \omega_i)$
  for $j := 0, 1, \ldots, 2^k - 1$ **do** $t_{ij} := a_{ij}.b_{ij}$ **end**
  $a_i := \text{FFT}(a_i, k, p_i, 1/\omega_i \bmod p_i)$
  for $j := 0, 1, \ldots, 2^k - 2$ **do** $t_{ij} := t_{ij}/2^k$ **end**
end
ChineseTheorem$(t, p, 2^k - 1, K)$ calcule les coefficients
modulo $P = \prod_{i=1}^{K} p_i$ du produit $F(X)G(X)$ à l'aide
des coefficients t_{ij} modulo p_i par le théorème chinois.
$h := \text{ChineseTheorem}(t, p, 2^k - 1, K)$
if $\text{degree}(F) = 2^{k-1}$ **then**
  $h_0 := f_0.g_0$
  $h_{2^k} := f_{2^{k-1}}.g_{2^{k-1}}$
end
return(H)
end PolyFFTMult

Algorithme 8 : Multiplication polynomiale par FFT sur $\mathbf{Z}/N\mathbf{Z}$.

Le coût final de la méthode est :
- $O(nK \log N)$ opérations pour réduire les $2n$ coefficients de $F(X)$ et $G(X)$ modulo les K premiers p_i.
- $O(Kn \log n)$ opérations pour calculer les K convolutions dans chaque $\mathbf{GF}(p_i)$.
- $O(nK \log n)$ opérations pour le théorème chinois.

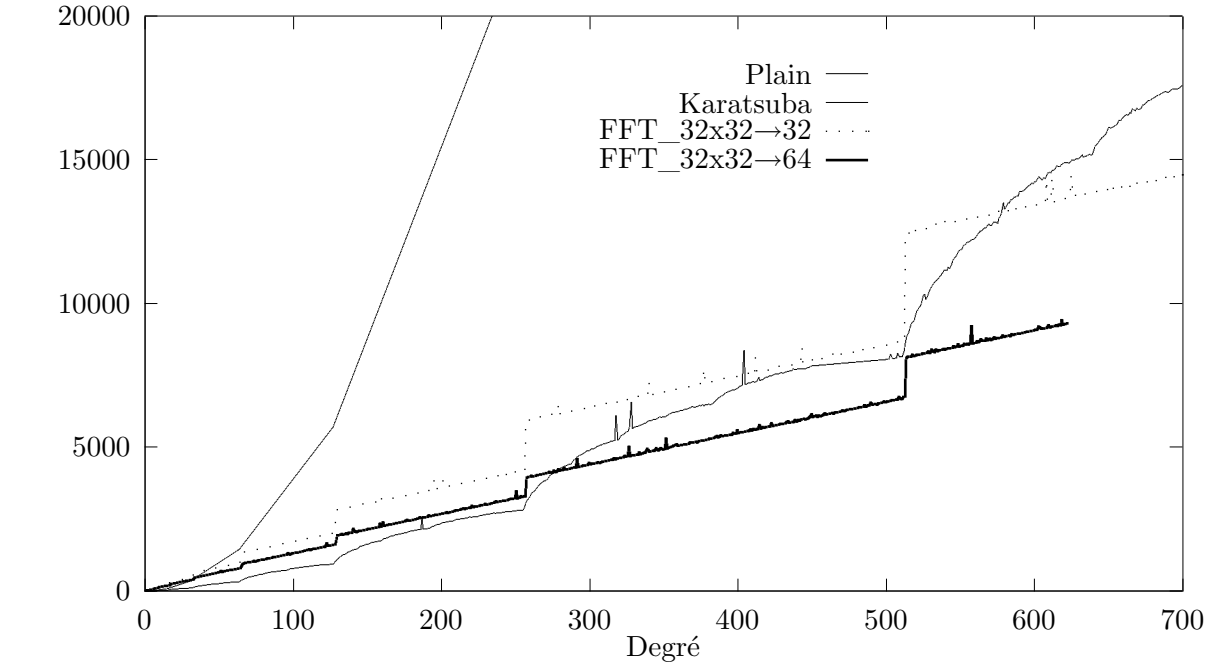
Soit un coût de $O(nK \log(nN))$ opérations. Comme de plus $K = O(\log(nN^2))$, on a finalement pour complexité asymptotique

$$\boxed{O(n(\log n + \log N^2)^2 \text{ u.b})}.$$

4.1.4 Résultats expérimentaux

Il est intéressant ici de comparer ces trois méthodes de multiplication. En effet, si les complexités permettent de se rendre compte qu'asymptotiquement, la multiplication par FFT est plus efficace que celle de Karatsuba et que cette dernière est elle-même plus efficace que la naïve, la comparaison pour de petits degrés est par contre moins aisée.

C'est dans cette optique que nous avons mesuré le temps de multiplication de 2 polynômes dont on a fait varier les degrés sur une DEC Station ALPHA. Tous les calculs ont été fait modulo un nombre premier de 200 chiffres pour les 3 méthodes précédemment décrites. Cette machine permet de plus d'obtenir un résultat en simple précision de 64 bits quand on multiplie deux entiers de 32bits, on a donc adjoint aux 3 méthodes précédentes une multiplication polynomiale par FFT utilisant pour son arithmétique modulo les K premiers p_i cette multiplication entière.



Graphe 1 : Multiplication polynomiale de de deux polynômes modulo un nombre premier de 200 chiffres.

Il ressort de cette étude :

- Asymptotiquement, les ordres de grandeur des complexités sont bien respectés avec une croissance de FFT_32x32→32 et FFT_32x32→64 quasi linéaire alors que celle de Plain et Karatsuba est quadratique.
- Karatsuba est toujours meilleur que Plain.
- FFT_32x32→64 est toujours meilleur que FFT_32x32→32.
- Par contre Karatsuba est plus efficace que FFT_32x32→32 pour des degrés inférieurs à 580 et est plus efficace que FFT_32x32→64 pour des degrés inférieurs à 280.

Dans l'implantation, on utilise Karatsuba ou la FFT suivant le degré du polynôme à multiplier. Étant donné que dans la seconde phase du paradoxe des anniversaires, certains produits mettent en jeu des polynômes de degrés supérieurs à 1024, le gain apporté par la FFT est déterminant même si pour les petits degrés il est toujours nécessaire d'utiliser Karatsuba.

4.2 La division polynomiale

Deux algorithmes de division polynomiale sur $\mathbf{Z}/N\mathbf{Z}$ vont être décrits pour obtenir à partir d'un polynôme $G(X) = \sum_{i=0}^{n_g} g_i X^i$ et un polynôme $F(X) = \sum_{i=0}^{n_f} f_i X^i$ ($n_f \leq n_g \leq 2n_f$) le quotient $Q(X) = \left\lfloor \frac{G(X)}{F(X)} \right\rfloor$ et le reste $R(X) = G(X) \bmod F(X)$.

Il s'agit de la division usuelle de complexité $O(n_f n_g)$ qui va servir d'étalon à la méthode de dite de Newton. Cette dernière est asymptotiquement plus rapide car elle est de même complexité que la multiplication par FFT. Nous ne rappellerons ici que les algorithmes et résultats essentiels et le lecteur est invité à se réferrer pour avoir les justifications adéquates à [7] pour la division naïve et à [2] pour la méthode de Newton.

4.2.1 La division usuelle

Elle met en œuvre la division euclidienne de 2 polynômes sur $\mathbf{Z}/N\mathbf{Z}$ comme décrit dans l'algorithme 9. Remarquons qu'il est possible de découvrir un diviseur de N lors de l'inversion de f_n.

```
procedure PolyUsualDivide(Q(X), R(X), G(X), F(X), N)
n_f := degree(F)
n_g := degree(G)
g := gcd(f_{n_f}, N)
if g ≠ 1 then
  return(g)
else
  h := 1/f_{n_f} mod N
end
R(X) := G(X)
for k := n_g − n_f, n_g − n_f − 1, ..., 0 do
  q_k := g_{n_f+k}.h mod N
  R(X) := R(X) − q_k X^k F(X) mod N
end
return(g)
end PolyUsualDivide
```

Algorithme 9 : Division polynomiale usuelle.

Le nombre d'opérations nécessaires est alors essentiellement proportionnel à

$$\boxed{(ng - n_f + 1)n_f \text{ u.b}}.$$

Cette méthode est surtout utilisée quand n_g est proche de n_f.

4.2.2 La méthode de Newton

On y distingue 3 étapes pour obtenir $R(X) = G(X) \bmod F(X)$:

1. Calcul du polynôme réciproque de $F(X)$,

$$\mathcal{RECIP}(F(X)) = \left\lfloor \frac{X^{2n_f}}{F(X)} \right\rfloor .$$

.

2. Obtention de $Q(X) = \left\lfloor \frac{G(X)}{F(X)} \right\rfloor$ à partir de $\mathcal{RECIP}(F(X))$ avec une multiplication polynomiale supplémentaire.
3. Obtention de $R(X) = G(X) \bmod F(X)$ à partir de $Q(X)$ avec une multiplication polynomiale supplémentaire.

Polynôme réciproque

Lorsque n_f est une puissance de 2, l'algorithme 10 calcule le polynôme réciproque de $F(X)$, $\mathcal{RECIP}(F(X))$, en $O(M(n))$ opérations où $M(n)$ est le coût de la multiplication de deux polynômes de degré n.

procedure PolyReciprocal($R(X), F(X), N$)
$n_f := \text{degree}(F)$
$g := \gcd(f_{n_f}, N)$
if $g \neq 1$ **then return**(g) **else**
  $h := 1/f_{n_f} \bmod N$
end
$R_1(X) := h$
$e_1 := -f_{n-1}.h \bmod N$
for $k := 2, 2, 8, \ldots, n_f$ **do**
  $\sum_{j=0}^{2k-3} h_j X^j := R_{k/2}(X)^2 \sum_{j=0}^{k-1} f_{n-j} X^{k-1-j} \bmod N$
  $R_k(X) := 2R_{k/2}(X)X^{k/2} - \sum_{j=0}^{k-1} h_{j+k-2} X^j \bmod N$
  if $k = 2$ **then**
    $e_2 := e_1^2 - f_{n-2}.h \bmod N$
  else
    $e_k := e_{k/2}^2 - h_{k-3} f_n - f_{n-k}.h \bmod N$
  end
end
$R(X) := XR_n(X) + e_n.h \bmod N$
return(g)
end PolyReciprocal

Algorithme 10 : Polynôme réciproque de F quand n_f est une puissance de 2.

On vérifie facilement que $R_k(X)$ est un polynôme de degré $k-1$. Il est par contre plus difficile de vérifier que l'algorithme est correct en montrant que

$$\text{degree}\left((XR_k(X) + e_k/f_{n_f})F(X) - X^{n_f+k}\right) \leq n_f - 1 \quad \forall k = 1, 2, 3 \ldots, n_f$$

Une preuve donnée par [14] consiste à poser $\rho_k(X) = XR_k(X) + e_k/f_{n_f}$ et de voir que l'algorithme est équivalent à

$$\begin{aligned} \rho_1(X) &= \frac{X}{f_{n_f}} - \frac{f_{n_f-1}}{f_{n_f}^2} \\ \rho_k(X) &= 2X^{k/2}\rho_{k/2}(X) - \left\lfloor \frac{R_{k/2}(X)^2 \left\lfloor F(X)/X^{n_f-k} \right\rfloor}{X^k} \right\rfloor \text{ pour } k > 1. \end{aligned}$$

Lorsque n_f n'est pas une puissance de 2, il suffit alors de multiplier $F(X)$ par le monôme X^k permettant d'obtenir un produit de degré une puissance de 2. A partir de ce polynôme, on utilise l'algorithme 10 afin d'en calculer le polynôme réciproque $R(X)$ et on a $\mathcal{RECIP}(F) = \left\lfloor \frac{R(X)}{X^k} \right\rfloor$. Dans ce cas, le calcul de e_i est inutile.

Quotient et reste

Pour obtenir le quotient de $G(X)$ par $F(X)$ à l'aide de $\mathcal{RECIP}(F(X)$, il suffit de remarquer que :

$$Q(X) = \left\lfloor \frac{G(X)}{F(X)} \right\rfloor = \left\lfloor \frac{\lfloor G(X)/X^{n_f} \rfloor \, \mathcal{RECIP}(F(X))}{X^{n_f}} \right\rfloor.$$

En effet

$$\begin{aligned} X^{n_f}(G(X) - F(X)Q(X)) &= X^{n_f}\left(G(X) - X^{n_f}\left\lfloor \tfrac{G(X)}{X^{n_f}} \right\rfloor\right) \\ &\quad - \left\lfloor \tfrac{G(X)}{X^{n_f}} \right\rfloor \left(F(X)\mathcal{RECIP}(F(X)) - X^{2n_f}\right) \\ &\quad -F(X)\left(Q(X)X^{n_f} - \left\lfloor \tfrac{G(X)}{X^{n_f}} \right\rfloor \mathcal{RECIP}(F(X))\right). \end{aligned}$$

Comme la seconde partie de cette inégalité ne met en jeu que des produits de polynômes de degré au plus n_f et $n_f - 1$, la partie gauche est donc bien de degré au plus $2n_f - 1$.

Le reste $R(X) = G(X) \bmod F(X)$ s'obtient alors tout simplement par

$$R(X) = G(X) - Q(X)F(X).$$

On a donc encore un coût asymptotique de $O(M(n_f))$ opérations, c'est à dire

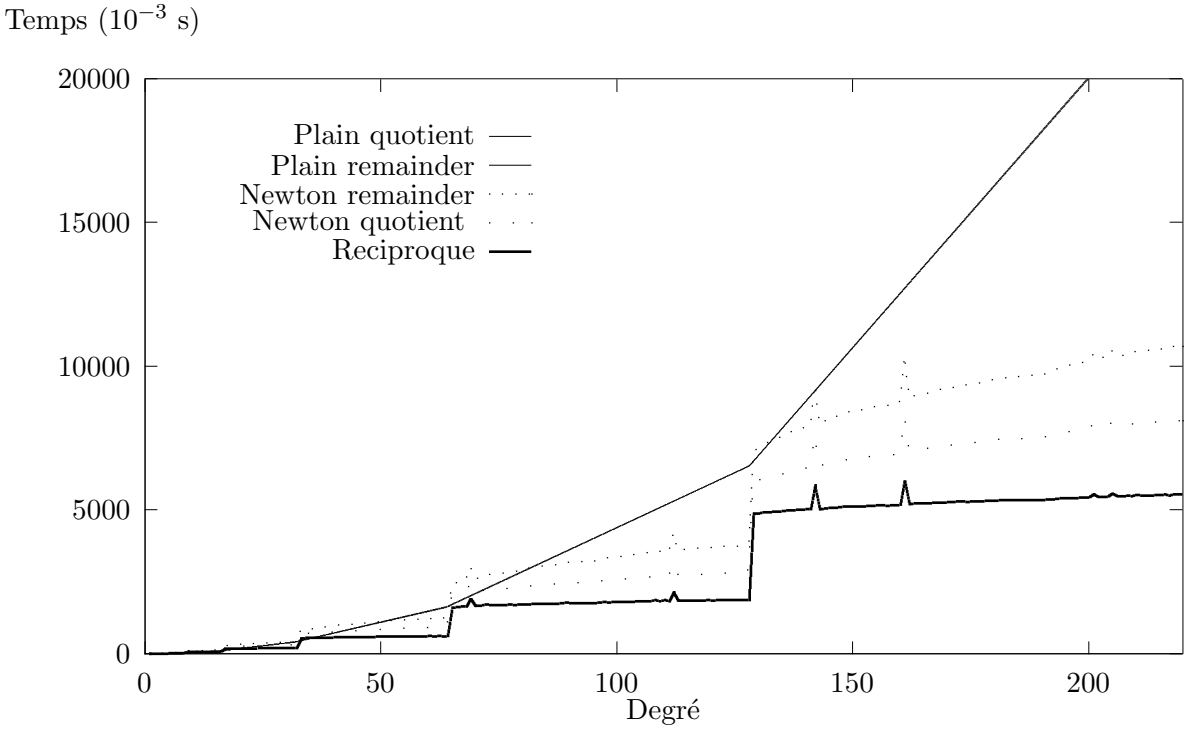
$$\boxed{O(n_f(\log n_f)^2) \text{ u.b}}.$$

lorsque l'on utilise la multiplication par FFT.

4.2.3 Résultats expérimentaux

Pour la seconde phase des paradoxes des anniversaires, on a uniquement besoin de la division de polynômes de degré $2n$ par des polynômes de degré n. Le temps d'exécution de cette division a donc été mesuré sur une DEC Station ALPHA. Tous les calculs ont été faits modulo un nombre premier de 200 chiffres pour la division usuelle et celle de Newton.

On a de plus mesuré le temps de calcul de $\mathcal{RECIP}$ pour chaque degré afin de mesurer la part de ce précalcul dans la méthode de Newton. La multiplication polynomiale utilisée est celle de Karatsuba ou celle par FFT suivant les degrés.



Graphe 2 : Division polynomiale d'un polynôme de degré $2n$ par un polynôme de degré n modulo un nombre premier de 200 chiffres.

.

On en déduit :

- Asymptotiquement, la méthode de Newton est meilleure que la division naïve.
- La méthode de Newton devient efficace très tôt ; dès le quotient d'un polynôme de degré 150 par un polynôme de degré 75 et dès le reste d'un polynôme de degré 180 modulo un polynôme de degré 90.
- La part de $\mathcal{RECIP}$ est prépondérant dans la méthode de Newton ; en moyenne 70% du temps pour le quotient et 50% pour le reste.

4.3 Evaluation d'un polynôme en de nombreux points

Le problème est ici d'évaluer un polynôme $G(X) = \sum_{i=0}^{n_g} g_i X^i$ en n_f points $x_0, x_1, \ldots, x_{n_f-1}$.

La méthode usuelle pour réaliser cela est d'évaluer G en chacun des points x_i par l'algorithme d'Horner. On a alors une complexité de l'ordre de $n_f n_g$ u.b. Il est néanmoins possibles de faire mieux en réutilisant les idées sous-jacentes à la FFT lorsque $n_f = 2^k$, $k \in \mathbf{N}$ [2].

La première étape est la construction récursive des polynômes :

$$q_{lm} = \prod_{j=l}^{l+2^m-1} (X - x_j),$$

avec $0 \leq m \leq k$ et l un multiple de 2^m de l'intervalle $0 \leq l \leq 2^k - 1$. On utilise pour

cela la formule de récurrence :

$$q_{lm} = q_{l,m-1} q_{l+2^{m-1},m-1}.$$

Le schéma 3 est un exemple d'une telle construction pour 8 valeurs.

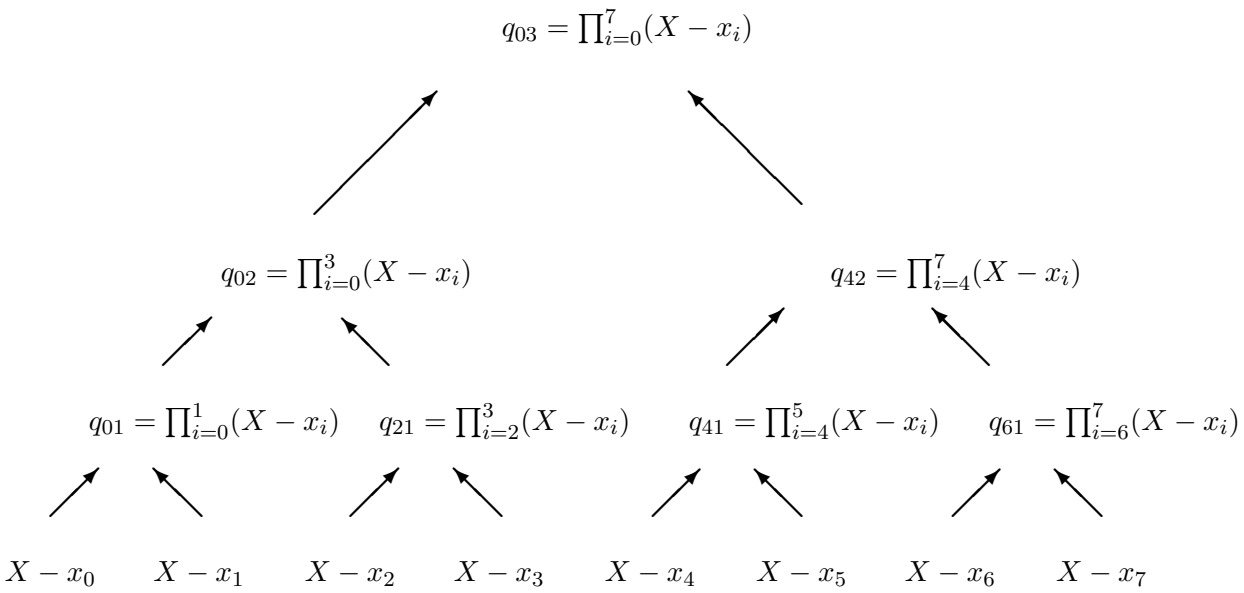


Schéma 3 : Construction d'un polynôme à partir de ses 8 racines.

L'algorithme 11 en montre la mise en œuvre dans le cas général.

```
procedure PolyFromRoots(x, k)
for l := 0, 1, ..., 2^k − 1 do q_{k0} = X − x_l end
for m := 1, 2, ..., k do
  for l := 0, 2^m, 2.2^m, ..., (2^{k−m} − 1).2^m do
    q_{lm} := q_{l,m−1} q_{l+2^{m−1},m−1}
  end
end
return(q)
end PolyFromRoots
```

Algorithme 11 : Construction d'un polynôme à partir de ses racines.

Ensuite, exactement comme pour la FFT, on calcule récursivement le reste de G par les polynômes q_{lm} grâce aux formules

$$\begin{cases} G(X) \bmod q_{l,m-1} & = & (G(X) \bmod q_{lm}) \bmod q_{l,m-1}, \\ G(X) \bmod q_{l+2^{m-1},m-1} & = & (G(X) \bmod q_{lm}) \bmod q_{l+2^{m-1},m-1}. \end{cases}$$

Ces formules peuvent aussi s'écrire en posant $r_{0k} = G(X)$:

$$\begin{cases} r_{l,m-1} & = & r_{lm} \bmod q_{l,m-1}, \\ r_{l+2^{m-1},m-1} & = & r_{lm} \bmod q_{l+2^{m-1},m-1}, \end{cases}$$

et on a alors $\forall l \in \{0, 1, \ldots, n_f - 1\}, r_{l0} = G(X) \bmod X - x_l = G(x_l)$.

Lorsque l'on utilise la méthode de Newton pour effectuer toutes ces divisions, on peut encore améliorer l'algorithme en remarquant que

$$\begin{cases} \mathcal{RECIP}(q_{l,m-1}) & = & \left\lfloor \frac{q_{l+2^{m-1},m-1} \left\lfloor \mathcal{RECIP}(q_{lm})/X^{2^m} \right\rfloor}{X^{2^m}} \right\rfloor, \\ \mathcal{RECIP}(q_{l+2^{m-1},m-1}) & = & \left\lfloor \frac{q_{l,m-1} \left\lfloor \mathcal{RECIP}(q_{lm})/X^{2^m} \right\rfloor}{X^{2^m}} \right\rfloor. \end{cases} \tag{4.7}$$

Pour l'évaluation de G en 4 points x_i, on a alors exactement l'arbre des calculs décrits par le schéma 4. Sur ce schéma, la flèche $\hookrightarrow$ indique que le polynôme réciproque ρ est calculé avec les formules (4.7) à et la flèche $\rightsquigarrow$ indique de la même façon une obtention de reste par la méthode de Newton.

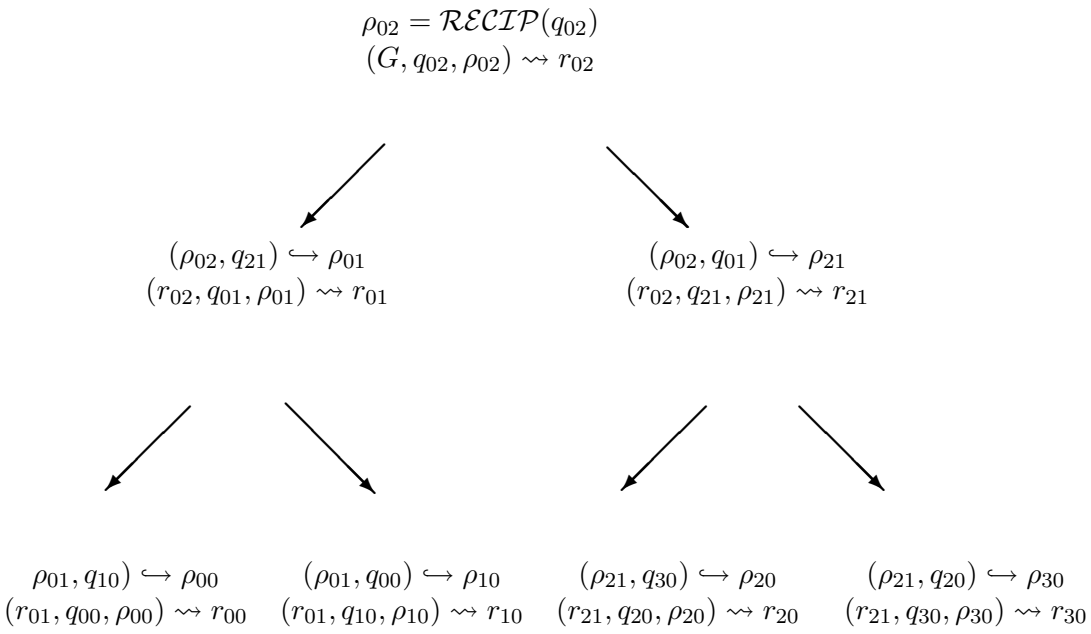


Schéma 4 : Évaluation d'un polynôme G en 4 valeurs x_i.

Plus généralement, l'algorithme 12 utilise cette méthode pour évaluer un polynôme $G(X)$ à partir du polynôme réciproque $\rho = \mathcal{RECIP}(q_{0k})$.

procedure PolyEvalFromRecip$(y, G(X), \rho(X), q, N)$
$k := \log_2(\text{degree}(\rho))$; $\rho_{0k} := \rho(X)$
PolyDivideFromRecip$(r(X), G(X), q(X), \rho(X), N)$ où $\rho(X) = \mathcal{RECIP}(q(X))$
calcule $r(X) := G(X)$ mod $q(X)$ par la méthode de Newton.
$g :=$ PolyDivideFromRecip$(r_{0k}, G(X), q_{0k}, \rho(X), N)$
for $m := k-1, k-2, \ldots, 0$ **while** $g = 1$ **do**
  for $l := 0, 2.2^m, \ldots, (2^{k-m}-2).2^m$ **while** $g = 1$ **do**
    # PolyRecipFromRecip$(\rho(X), q(X))$ retourne le polynôme réciproque
    # obtenu à partir de $q(X)$ et $\rho(X)$ avec les formules (4.7).
    $\rho_{lm} :=$ PolyRecipFromRecip$(\rho_{l,m+1}, q_{l+2^m,m})$
    $\rho_{l+2^m,m} :=$ PolyRecipFromRecip$(\rho_{l,m+1}, q_{lm})$
    $g :=$ PolyDivideFromRecip$(r_{lm}, r_{l,m+1}, q_{lm}, \rho_{lm}, N)$
    if $g \neq 1$ **then**
      $g :=$ PolyDivideFromRecip$(r_{l+2^m,m}, r_{l,m+1}, q_{l+2^m,m}, \rho_{l+2^m,m}, N)$
    end
  end
end
if $g \neq 1$ **then**
  for $l := 0, 1, \ldots, 2^k - 1$ **do** $y_l := r_{l0}$ **end**
end
return(g)
end PolyEvalFromRecip

Algorithme 12 : Evaluation de $G(X)$ à partir du polynôme réciproque $\rho(X)$.

Chaque étape de cette récursion inverse nécessite 6 produits de polynômes qui sont de degré 2^m ou de degré 2^m-1. On a donc un coût de $2^{k-m+1}6M(2^m)$ opérations lorsque l'on passe de m à $m+1$, c'est à dire $3(2^k/2^m)M(d) \leq 6M(2^{k-1})$ opérations. En sommant ces coûts sur les k niveaux, on obtient alors une complexité de l'ordre de $O(M(n_f/2))$.

Comme on a un coût supplémentaire de l'ordre de $O(M(n_f))$ lorsque l'on calcule le polynône q_{ok} à partir des racines x_i et un autre du même ordre pour le calcul de son polynôme réciproque, cette évaluation polynomiale a une complexité asymptotique

$$\boxed{O(n_f(\log(n_f)^)2 \text{ u.b}}.$$

en utilisant la multiplication par FFT.

On peut faire ici deux remarques

- Il est nécessaire dans cet algorithme de stocker tous les polynômes q_{lm} calculés par PolyFromRoots() pour pouvoir les réutiliser dans PolyEvalFromRecip(). On a donc besoin d'une mémoire en $O(n_f \log n_f)$. Il est néanmoins possible de les stocker sur disque conservant ainsi un stockage raisonnable en mémoire vive en $O(n_f)$.
- Si on utilise la multiplication par FFT, beaucoup des opérandes intervenant dans PolyFromRoots() sont réutilisées dans PolyFromRecip(). Pour chacune de ces opérandes il est alors nécessaire de recalculer leurs transformées de Fourier. Un gain substantiel peut être obtenu en stockant les transformées de

Fourier des polynômes q_{lm} modulo les petits premiers plutôt que les coefficients modulo N.

La seconde phase "paradoxe des anniversaires" étant en fait une mise en œuvre de cet algorithme, la comparaison expérimentale de celui-ci avec l'évaluation naïve est faite au chapitre 5.

Chapitre 5

Implantation de l'algorithme

5.1 Implantation de la première phase

Deux points essentiels sont à optimiser dans une implantation efficace de l'algorithme de Lenstra ; l'addition de deux points sur une courbe et la multiplication d'un point par un entier k.

L'addition de deux points peut-être programmée telle qu'elle est décrite dans la définition 2 du chapitre 1. Cependant, vu le coût important des inversions, deux optimisations majeures sont possibles :

- La première est proposée par P. L. Montgomery [13]. Elle consiste à utiliser des courbes de la forme $By^2 = x^3 + Ax^2 + x$ pour $(A, B) \in (\mathbf{Z}/N\mathbf{Z})^2, B \neq 0$ et $A \neq \pm 2$. Les additions y sont alors moins coûteuses car l'inversion y est remplacée par des multiplications.
- La seconde consiste à utiliser l courbes et à effectuer l additions en même temps, remplaçant par là-même l inversions (dont le coût est prohibitif) par une seule et quelques multiplications.

Quant à la multiplication, la méthode binaire pour la réaliser est ici d'autant plus efficace qu'il est possible de compacter les nombres premiers 2 par 2 afin d'en diminuer le poids [11]. Face à cette astuce, des méthodes de coût théoriquement moindre comme la multiplication par blocs [7] ou par chaînes d'additions [16] perdent leur attrait et ne seront pas décrites.

5.1.1 Choix de l'addition sur courbes elliptiques

L'addition naïve

Pour des courbes données en paramétrisation de Weierstrass, les formules de définition se programment sans difficulté su $\mathbf{Z}/N\mathbf{Z}$ en tenant compte de la découverte d'éventuels facteurs de N lors de l'inversion. La procédure EcModAdd() est un exemple d'une telle pseudo-addition.

procedure EcModAdd($P, Q, \mathbf{E}_{a,b}, g, N$)
$g := 1$
if $P = O$ **then return**(Q) **end**
if $Q = O$ **then return**(P) **end**
if $P = Q$ **then**
  # Ord(P) retourne l'ordonnée du point P.
  if Ord(P) $= 0$ **then return**(O) **end**
  $g := \gcd(2\text{Ord}(P) \bmod N, N)$
  if $g = 1$ **then**
    $\lambda := \frac{3\text{Absc}(P)^2+1}{2\text{Ord}(P)} \bmod N$
  end
else
  if $P = -Q$ **then return**(O) **end**
  $g := \gcd(\text{Absc}(Q) - \text{Absc}(P) \bmod N, N)$
  if $g = 1$ **then**
    $\lambda := \frac{\text{Ord}(Q)-\text{Ord}(P)}{\text{Absc}(Q)-\text{Absc}(P)} \bmod N$
  end
end
if $g = 1$ **then**
  $x := \lambda^2 - \text{Absc}(P) - \text{Absc}(Q) \bmod N$
  $y := \lambda(\text{Absc}(P) - x) - \text{Ord}(P) \bmod N$
  # Point(x, y) retourne le point d'abscisse x
  # et d'ordonnée y
  return(Point(x, y))
else
  return(O)
end
end EcModAdd

Algorithme 13 : Addition de deux points sur courbe elliptique.

Pour sommer 2 points identiques, 4 multiplications, 6 additions et une inversion sont nécessaires alors que pour une addition simple, il faut 3 multiplications, 5 additions et une inversion.

La paramétrisation de Montgomery

Pour éviter l'inversion de l'addition naïve, on peut chercher à garder la trace des numérateurs et dénominateurs des coordonnées d'un point P. C'est ce que l'on appelle les coordonnées homogènes de P.

Ainsi, au lieu de travailler avec $P = (x, y)$ sur la courbe $y^2 = x^3 + ax + b$, on utilise $P(X, Y, Z)$ sur la courbe $Y^2Z = X^3 + aYZ^2 + bZ^3$ où x=X/Z et y=Y/Z. Les expressions de X, Y et Z sont cependant si coûteuses lors d'une addition ($\simeq 10$ multiplications pour chaque coordonnée) qu'elles ont poussé P. L. Montgomery [13] à considérer des courbes particulières et une paramétrisation adaptée.

Il utilise des courbes de la forme

$$By^2 = x^3 + Ax^2 + x \quad \text{pour} \quad (A, B) \in (\mathbf{Z}/p\mathbf{Z})^2, B \neq 0 \text{ et } A \neq \pm 2. \tag{5.1}$$

Considérant $P_1(x_1, y_1)$ et $P_2(x_2, y_2)$ avec $x_1 \neq x_2$ et $x_1 x_2 \neq 0$, l'abscisse de $P_3(x_3, y_3) = P_1 + P_2$ vérifie :

$$x_3 = B\left(\frac{y_1 - y_2}{x_1 - x_2}\right)^2 - A - x_1 - x_2,$$

qui peut s'écrire encore

$$x_3(x_1 - x_2)^2 = B\frac{(x_2 y_1 - x_1 y_2)^2}{x_1 x_2}. \tag{5.2}$$

En utilisant (5.2), on obtient de même pour $P_4(x_4, y_4) = P_1 - P_2$

$$x_4(x_1 - x_2)^2 = B\frac{(x_2 y_1 + x_1 y_2)^2}{x_1 x_2}. \tag{5.3}$$

Ces deux équations conduisent alors à une expression pour $P_1 \neq P_2$ encore valable quand $x_1 x_2 = 0$:

$$x_4 x_3 (x_1 - x_2)^2 = (x_1 x_2 - 1)^2. \tag{5.4}$$

De même, on aurait pour $P_1 = P_2$:

$$4x_1 x_3 (x_1^2 + Ax_1 + 1) = (x_1^2 - 1)^2. \tag{5.5}$$

Ces formules permettent alors de calculer l'abscisse de la somme de deux points connaissant les abscisses de ces points ainsi que celle de leur différence.

Pour la méthode de Lenstra, on doit calculer mP avec $m \in \mathbf{N}$. Si on note (X_m, Y_m, Z_m) et (X_n, Y_n, Z_n), les coordonnées homogènes de mP et nP, on obtient alors à partir de (5.4) et (5.5) connaissant $(m-n)P$ de coordonnées $(X_{m-n}, Y_{m-n}, Z_{m-n})$ les formules suivantes pour $mP \neq nP$:

$$\begin{aligned} X_{m+n} &= Z_{m-n}[(X_m - Z_m)(X_n + Z_n) + (X_m + Z_m)(X_n - Z_n)], \\ Z_{m+n} &= X_{m-n}[(X_m - Z_m)(X_n + Z_n) - (X_m + Z_m)(X_n - Z_n)], \end{aligned}$$

et pour $mP = nP$:

$$\begin{aligned} 4X_n Z_n &= (X_n + Z_n)^2 (X_n - Z_n)^2, \\ X_{2n} &= (X_n + Z_n)^2 - (X_n - Z_n)^2, \\ Z_{2n} &= 4X_n Z_n((X_n - Z_n)^2 + \tfrac{A+2}{4} 4X_n Z_n). \end{aligned}$$

Cette façon de faire nécessite pour une somme simple 6 multiplications et 4 additions et pour une somme double 5 multiplications et 4 additions si on connaît la différence des 2 points. Il faut cependant remarquer que les courbes de la forme (5.1) ne représentent pas toutes les courbes de $\mathbf{Z}/p\mathbf{Z}$. En effet, à partir d'une courbe en notation de Montgomery

$$Y^2 = BX^3 + AX^2 + BX,$$

le changement de variable linéaire

$$\begin{aligned} x &= BX + A/3, \\ y &= BY, \end{aligned}$$

permet d'obtenir la courbe

$$y^2 = x^3 + ax + b.$$

On a alors

$$\begin{aligned} a &= -3(A/3)^2 + B^2, \\ b &= 2(A/3)^3 - (A/3)B^2. \end{aligned}$$

Mais la transformation inverse n'est possible que si une solution $A/3$ de

$$(A/3)^3 + b(A/3) + a = 0,$$

est telle que la quantité

$$B^2 = b + 3(A/3)^2$$

soit un résidu quadratique.

Additionner $2l$ points sur l courbes en même temps

Au lieu de calculer en coordonnées affines $P + Q$ sur une courbe E, on calcule en même temps $P_i + Q_i$ sur l courbes E_i $(0 \leq i < l)$. Si on calculait successivement ces l additions, on aurait alors l entiers w_i à inverser et un coût prohibitif.

L'optimisation est basée sur la remarque suivante ; si on a deux entiers x et y à inverser, on a leurs inverses respectifs par $1/x = y/(xy)$ et par $1/y = x/(xy)$. Pour une unique inversion $1/(xy)$, on obtient donc deux inverses $1/x$ et $1/y$.

Cette remarque est généralisée pour l inversions w_j^{-1} dans l'algorithme ManyInversions() :

procedure ManyInversions(w, l, g, N)
$v_0 := w_0$
for $k := 1, 2, \ldots, l-1$ **do**
  $v_k := v_{k-1} w_k \bmod N$
end
$g := \gcd(v_{l-1}, N)$
if $g = 1$ **then**
  $u := v_{l-1}^{-1} \bmod N$
  for $k := l-1, l-2, \ldots, 1$ **do**
    $t_k := v_{k-1} u \bmod N$
    $u := w_k u \bmod N$
  end
  $t_0 := u$
end
return(t)
end ManyInversions

Algorithme 14 : l inversions modulaires avec une seule.

On remplace ainsi l inversions par une inversion et $3(l-1)$ multiplications. Le prix d'une somme double sur une courbe vaut alors

$$7 + \frac{K-3}{l}$$

et celui d'une somme simple

$$6 + \frac{K-3}{l}$$

où K est le coût d'une inversion.

5.1.2 La multiplication de points par des entiers

La méthode binaire

Pour calculer kP où k est un entier et P un point, une méthode naïve consisterait à calculer $2P, 3P, \ldots, kP$ avec $k-1$ additions.

Il est possible de réduire ce nombre en remarquant que si k s'écrit en base 2

$$k = b_0 + 2(b_1 + 2(\ ..(b_{\lfloor \log_2 k \rfloor - 1} + 2b_{\lfloor \log_2 k \rfloor})..)),$$

alors

$$kP = b_0P + 2(b_1P + 2(\ ..(b_{\lfloor \log_2 k \rfloor - 1}P + 2b_{\lfloor \log_2 k \rfloor}P)..)).$$

Ainsi, à titre d'exemple, $11P$ s'écrit

$$11P = P + 2(P + 2(2P))$$

en utilisant 5 additions au lieu de 10.

La procédure EcModMult() est une implantation de la remarque précédente.

```
procedure EcModMult(k, P, E_{a,b}, g, N)
g := 1
e := ⌊log_2 k⌋
R := P
for l := e − 1, e − 2, ..., 0 while g = 1 do
   R := EcModAdd(R, R, E_{a,b}, g, N)
   if ⌊k/2^e⌋ mod 2 = 1 and g = 1 then
      R := EcModAdd(R, P, E_{a,b}, g, N)
   end
end
return(R)
end EcModMult
```

Algorithme 15 : Méthode binaire de calcul de kP.

Dans le cadre de la paramétrisation de Montgomery, il est nécessaire de connaître la différence de 2 points avant d'en faire la somme. Cela n'est pas génant si dans la méthode binaire, au lieu d'obtenir $2mP$ ou $(2m+1)P$ à partir de mP, on obtient le couple $(2mP, (2m+1)P)$ ou $(mP, (m+1)P)$ à partir de $(mP, (m+1)P)$. Cependant, chaque étape nécessite alors 11 multiplications pour un coup total de

$$11 \log_2 k \quad \text{multiplications.}$$

En revanche, lorsque l'on effectue plusieurs inversions ensembles en notation de Weierstrass, on effectue à chaque étape une somme double et parfois une somme

simple. Statistiquement, un nombre a environ autant de 0 que de 1 dans sa décomposition binaire, on a donc besoin pour calculer kP, de $\lfloor \log_2 k \rfloor$ additions doubles et de $\frac{1}{2}\lfloor \log_2 k \rfloor$ additions simples en moyenne. Ce qui conduit à un coup total de

$$\left(10 + \frac{3(K-3)}{2l}\right) \log_2 k \quad \text{multiplications.}$$

Minimiser le poids de k

Dans la méthode de Lenstra, on part d'un point P sur une courbe $\mathbf{E}_{a,b}$, et on fait ensuite en permanence des produits de la forme $R = {p_i}^{e_i} R$ avec p_i premier.

Une idée de A. K. Lenstra [11] consiste à réaliser des produits de la forme

$$R = {p_i}^{e_i} {p_j}^{e_j} R \tag{5.6}$$

et de chercher à minimiser le poids (nombre de 1 dans la décomposition binaire) des ${p_i}^{e_i} {p_j}^{e_j}$ afin de diminuer le nombre des sommes simples à réaliser dans l'algorithme de multiplication binaire.

Par exemple, si

$$p_i = 2133541 \quad (10000010000111000100101 \text{ en base } 2)$$

et

$$p_j = 2134861 \quad (10000010010011010011101 \text{ en base } 2)$$

alors

$$p_i p_j = 4554813472801$$

de représentation binaire 100001001001000000000001010000000000000100001 n'est que de de poids 8.

5.1.3 Résultats expérimentaux

Les constantes c du tableau 2 correspondant au coût de la multiplication pour chacune des méthodes d'addition sont déterminantes afin de déterminer quelle addition est la la plus efficace pour l'algorithme de Lenstra.

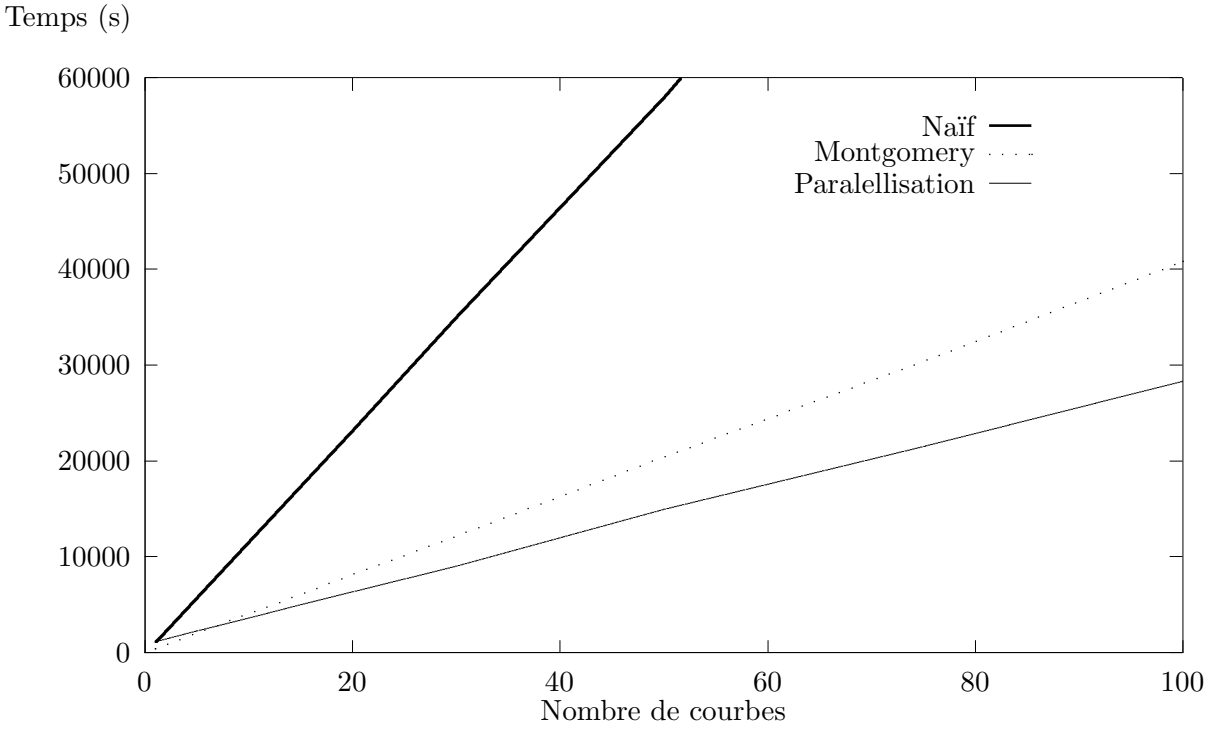
Version	c
Naïve	$\frac{11+3K}{2\ln 2}$
Montgomery	$\frac{11}{\ln 2}$
Paralellisée	$\frac{20+3(K-3)/l}{2\ln 2}$

Tableau 2 : Constantes de complexité pour différentes versions.

De ce tableau, on déduit que la version paralellisée est meilleure que celle de Montgomery dès que

$$\frac{K-3}{l} < \frac{2}{3}.$$

Ces trois versions ont été programmées. Les temps de non-factorisation d'un nombre premier sont alors un bon indicateur de l'efficacité relative de ces dernières. Le graphe de la figure 3 a été réalisé en exécutant la méthode de Lenstra avec une seconde phase classique ($B_1 = 5.10^4, B_2 = 10^5$) sans compactage sur un nombre premier de 100 chiffres.



Graphe 3 : Temps de non-factorisation d'un premier de 100 chiffres.

On retrouve alors :

- La version naïve et paralellisée mettent exactement le même temps pour $l = 1$ (1160 secondes), les temps de multiplication étant alors identiques.
- La version naïve est toujours plus coûteuse que les deux autres versions.
- La version de Montgomery devient moins efficace que la version paralellisée pour $l \simeq 6,59$.
- Les courbes de temps des deux premières versions sont exactement linéaires en fonction de l alors que celle de la version paralellisée ne l'est qu'asymptotiquement comme le laisse prévoir le terme $\frac{K-3}{l}$ de c.

En ce qui concerne la multiplication, un fichier de produits optimisés de deux nombres premiers a été généré par F. Morain pour tous les nombres premiers inférieurs à 2.10^6. Le nombre d'opérations dans la méthode binaire est alors donné dans le tableau 3

Nombre d'opérations	Premiers usuels	Premiers compactés	Gain
Sommes doubles	7040811	4498828	1.56
Sommes simples	3620758	652799	5.54
Total	10661569	5151627	2.06

Tableau 3 : Nombre d'additions nécessaires à la méthode binaire.

Par cette astuce, deux fois moins d'opérations sont donc nécessaires pour réaliser l'algorithme de Lenstra.

Il est souhaitable d'utiliser plusieurs courbes dans l'algorithme, c'est donc l'addition paralellisée qui a été retenue et ce d'autant plus que

- d'une part, le gain dû au compactage des premiers ne serait que de 1.56 avec la paramétrisation de Montgomery,
- et d'autre part celle-ci ne permet pas de représenter toutes les courbes elliptiques de $\mathbf{Z}/p\mathbf{Z}$.

5.2 Implantation des secondes phases

L'implantation des 2 secondes phases est de difficulté inégale. Le coût de la seconde phase classique est pour l'essentiel dû aux additions sur les courbes elliptiques et on bénéficie donc des optimisations faites pour la première phase.

Pour le paradoxe des anniversaires, il n'en est rien. La génération des points ne représente qu'une faible partie du coût total et la véritable difficulté est l'évaluation des quantités (3.12), (3.13) ; $\prod_{0\leq i,j<r}(x_i - y_j)$. Une évaluation naïve nécessite $\frac{r(r-1)}{2}$ multiplications mais grâce à une arithmétique polynomiale rapide, il est possible de descendre en dessous de ce seuil, du moins asymptotiquement.

5.2.1 La seconde phase "classique"

Tout comme la première phase, on effectue cette seconde phase sur plusieurs courbes en même temps en suivant le schéma de l'algorithme (3) afin d'optimiser les additions de points. Une courbe E ayant un point R (issu de la première phase) de plus petit q tel que $qR = O \bmod p$ sera alors responsable de la factorisation.

Cette seconde phase n'est cependant pas utilisable directement avec la paramétrisation de Montgomery car il faudrait connaître $(s_{j+1} - 2s_j)R$ lors de l'addition de s_jR et $(s_{j+1} - s_j)R$. On est donc obligé de repasser en coordonnées affines. Ainsi, après une première phase avec la courbe

$$y^2 = x^3 + Ax^2 + x \quad \text{pour} \quad A \neq \pm 2 \tag{5.7}$$

et l'abscisse x de kP, on effectue une seconde phase avec le point $(x' = x, y' = 1)$ sur la courbe

$$B'y^2 = x^3 + Ax^2 + x \quad \text{avec} \quad B' = x'^3 + Ax'^2 + x'. \tag{5.8}$$

que l'on transforme aisément pour retrouver la forme de la définition 2.2.

Le point de la courbe (5.8) garde alors les mêmes propriétés que celui de la courbe (5.7) pour 2 raisons :

– Les 2 courbes sont isomorphes car B' est en fait un carré (sa racine carrée est égale à y)
– Tous les calculs faits dans la première phase sont les mêmes que si il s'était s'agit de ceux de la courbe de la seconde phase puisqu'ils utilisent la même valeur de A.

5.2.2 La seconde phase "paradoxe des anniversaires"

La génération des points pseudo-aléatoires $\{R_i\}_{i=0}^{r_1-1}$ d'abscisses x_i (et $\{S_j\}_{j=0}^{r_2-1}$ d'abscisses y_j pour la variante) étant faite par les formules (3.5.2) en considérant plusieurs courbes à la fois ne nécessite que bien peu de temps par rapport à l'évaluation de la quantité

$$d = \prod_{0 \le i < j < r_1} (x_i - x_j)$$

($d = \prod_{0 \le i < r_1} \prod_{0 \le j < r_2} (x_i - x_j)$ pour la variante).

Un calcul de d comme l'algorithme 4 le réalise nécessite $\frac{r_1(r_1-1)}{2}$ multiplications modulaires, soit une complexité asymptotique en $O(r_1^2)$.

Si cependant $r_1 = 2^k$, P. L. Montgomery [14] a montré comment il était possible de mettre en place une arithmétique polynomiale asymptotiquement rapide pour effectuer ces mêmes calculs en $O(r_1(\log r_1)^2)$.

A cet effet ;

1. On calcule tout d'abord le polynôme $F(X)$ ayant x_i comme racines :

$$F(X) = \prod_{i=0}^{r_1-1} X - x_i. \tag{5.9}$$

   Avec une multiplication polynomiale usuelle, on peut calculer F en $O(r_1^2)$ opérations alors qu'avec celle basée sur une transformée de Fourier rapide (FFT) multipliant 2 polynômes de degré n asymptotiquement en $O(n(\log n)^2)$, ce calcul est alors réalisable en $O(r_1(\log r_1)^2)$ opérations.

2. A partir de F, on obtient

$$d^2 = \prod_{i=0}^{r_1-1} F'(x_i). \tag{5.10}$$

   Avec une évaluation polynomiale usuelle, le calcul de d^2 nécessite toujours $O(r_1^2)$ opérations (avec d'ailleurs un coût final supérieur à celui d'une évaluation naïve de d ! !). Avec l'évaluation décrite au chapitre 4, il est aussi possible de calculer les r_1 évaluations $F'(x_i)$ en $O((r_1(\log r_1)^2)$ opérations.

L'algorithme 16 illustre ces deux étapes à l'aide des routines du chapitre 4.

procedure BirthdayVandermondeStep2($R, \mathbf{E}_{a,b}, k, N$)
$g := 1 ; r := 2^k$
$Q := R ; x_0 :=$ Absc(Q)
for $i := 1, 2, 3, \ldots, r-1$ **while** $g = 1$ **do**
  $Q :=$ EcModAdd($Q, Q, \mathbf{E}_{a,b}, g, N$)
  if Random() $= 1$ **and** $g = 1$ **then**
    $Q :=$ EcModAdd($Q, R, \mathbf{E}_{a,b}, g, N$)
  end
  $x_i :=$ Absc(Q)
end
if $g \neq 1$ **then return**(g) **end**
$q :=$ PolyFromRoots(x, k)
$g :=$ PolyReciprocal($\rho(X), q_{0k}(X), N$)
$If g \neq 1$ **then return**(g) **end**
$g :=$ PolyEvalFromRecip($y, F'(X), \rho(X), q, N$)
$If g \neq 1$ **then return**(g) **end**
$d := 1$
for $i := 1, 2, 3, \ldots, r-1$ **do** $d := d.y_i \bmod N$ **end**
$g := \gcd(d, N)$
return(g)
end BirthdayVandermondeStep2

Algorithme 16 : Le paradoxe des anniversaires en $O(r(\log r)^2)$ u.b.

Les mêmes techniques sont aussi tout à fait applicable pour calculer (3.13) dans la variante de cette seconde phase en remplaçant F' par $G(X) = \prod_{j=0}^{r_2-1} X - y_j$. Il y est alors possible d'y choisir $r_2 \gg r_1$ sans augmenter le stockage. En effet, si au lieu de donner comme argument $G(X)$ à la routine PolyEvalFromrecip(), on donne $G(X) \bmod F(X)$, le résultat reste valide. On peut alors calculer $G(X) \bmod F(X)$ par l'intermédiaire de polynômes de degrés r_1 comme le montre la routine 17 et ensuite effectuer les mêmes opérations que dans l'algorithme 16.

procedure PolyFromRootsModF($s, r_2, F(X)$)
$r_1 := degree(F)$
$G(X) := 1$
for $j := 1, 2, \ldots, r_2/r_1$ **do**
  $H(X) := \prod_{i=(j-1)r_1}^{jr_1}(X - s_i) \bmod N$
  $G(X) := G(X)H(X) \bmod N \bmod F(X)$
end
end PolyFromRootsModF

Algorithme 17 : Construction d'un polynôme à partir de ses r_2 racines modulo un polynôme de degré r_1.

L'inconvénient de cette méthode est qu'elle nécessite un stockage en $O(r_1 \log r_1)$. Une alternative préférée par P. L. Montgomery [14] est, au lieu de calculer (5.10),

de réaliser

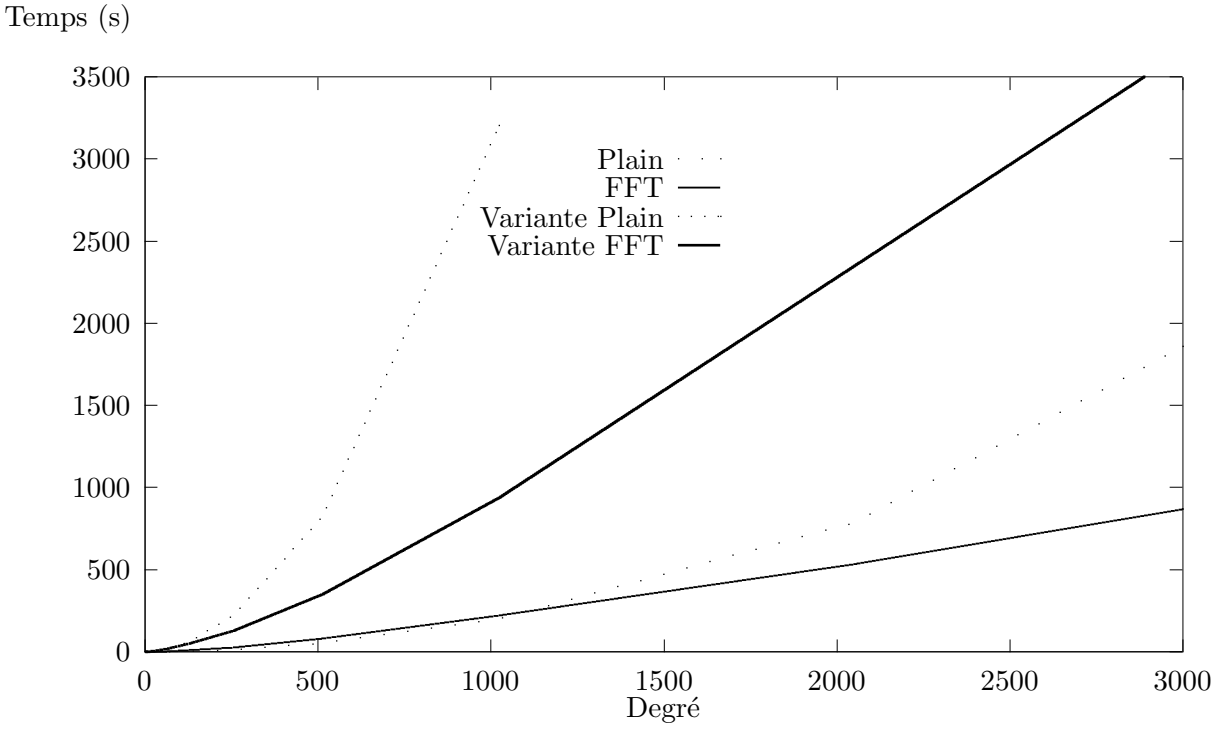
$$\gcd(F'(X), F(X)) \bmod N \tag{5.11}$$

qui n'a besoin que d'un stockage en $O(r_1)$. On a alors aussi une complexité asymptotique en $O((r_1(\log r_1)^2)$ pour ce pgcd rapide, mais des performances en pratique bien moins bonnes. Il est de plus tout à fait possible de conserver un stockage linéaire en mémoire vive en stockant les données intermédiaires de l'évaluation polynomiale sur disque.

5.2.3 Résultats expérimentaux

Les 2 variantes de la seconde phase paradoxe des anniversaires avec évaluation naïve et asymptotiquement rapides ont été implantées. On a ensuite mesuré le temps en secondes de non factorisation d'un nombre premier de 200 chiffres sur une DEC Station ALPHA.

Pour la version principale de l'algorithme on a fait varier le logarithme en base 2 du nombre de points r_1 comparés. Pour la variante, on a fait de même, en choisissant $r_2 = 8r_1$.



Graphe 4 : Versions naïves et optmisées des deux variantes de la seconde phase "paradoxe des anniversaires".

On peut remarquer :

- Asymptotiquement, les versions optimisées sont meilleures que les naïves ! Pour $r_1 = 2^{13}$ points, on a un gain de 4 pour la version principale et un gain de 5 pour la variante.
- Il faut attendre $r_1 = 1024$ pour avoir la version optimisée meilleure que la naïve pour la version principale.
- Par contre, dés que $r_1 = 64$, la version optimisée de la variante est plus efficace que son homologue naïf.

Chapitre 6

Résultats

RSA Data Security a mis en place le RSA Factoring Challenge en mars 91 [1]. Cet organisme proposa alors 2 listes de nombres à factoriser.

La première est composée de clefs potentielles pour le système de chiffrement à clefs publiques RSA [8] de 100, 110..., jusqu'à 500 chiffres. En avril 92, les deux premiers nombres de cette liste ont été factorisés par le crible quadratique (100 chiffres par Arjen Lenstra et Marc Manasse, 110 chiffres par Arjen Lenstra). En Juin 93, Thomas Denny, Bruce Dodson, Arjen Lenstra, Walter Lioen, Mark Manasse et Herman te Riele avec des machines de Saarbruecken University, Lehigh University, Bellcore, CWI Amsterdam, et DEC Systems Research Center ont aussi factorisé le nombre de 120 chiffres par la même méthode. Les trois quarts des calculs ont été faits sur stations de travail et le reste sur la MasPar de Bellcore.

L'autre liste est composée des nombres de partition $p(n)$ de plus de 100 chiffres. Le nombre de partition d'un entier n est le nombre de sommes (sans regarder l'ordre) égales à n. Ainsi, $p(5) = 7$ car :

$$5 = 1+1+1+1+1 = 1+1+1+2 = 1+1+3 = 1+2+2 = 1+4 = 2+3$$

Une table des $p(n)$ peut être calculée en utilisant la fonction génératrice

$$\sum_{n=0}^{\infty} p(n)X^n = \prod_{n=1}^{\infty} \frac{1}{(1-X^n)} = \frac{1}{\sum_{-\infty}^{\infty}(-1)^n X^{(3n^2+n)/2}}.$$

Afin de réduire la taille de la table, seuls les nombres $p(n)$ avec n premier ont été conservés. Initialement, la table commençait avec $p(8681)$. Mais durant la première semaine environ 30% de la liste originale des 1182 entrées était déjà factorisées. Actuellement, celle-ci comprend des nombres de 111 à 161 chiffres.

Le programme **MECM**[1] met en œuvre les algorithmes décrits dans les chapitres précédents. Il a été écrit en C à l'aide du package **CESAR**[2] développé par François Morain et la librairie **LIPS**[3] développée à l'Universität des Saarlandes par Ralf Roth et Thomas Setz.

[1] Multiple Elliptic Curves Method
[2] Courbes Elliptiques Et ARithmetique
[3] LIbrary for Parallel Systems

CESAR comprend entre autre la librairie BigNum de calcul en multiprécision [4] et la librairie BigMod de calcul modulaire qui, grâce à un noyau assembleur spécifique à chaque architecture, sont particulièrement performantes. En outre, certaines routines développées pour MECM seront bientôt incluses dans CESAR ; arithmétique avec plusieurs courbes elliptiques, certaines opérations polynomiales, routines de factorisations, etc.

LIPS est une librairie qui permet de gérer des communications par "sockets" UNIX entre stations de travail du réseau INTERNET. Elle permet de plus de configurer des applications afin qu'elles tiennent compte des autres utilisateurs (Mise en sommeil les jours de semaines, lorsqu'un autre utilisateur travaille sur la station, ...). Cette librairie, prévue initialement uniquement pour machines SUN est en cours de "portage" par l'auteur pour machines HP, DEC et IBM.

Après 9 mois d'utilisation sur notamment le réseau des 20 stations de travail HP de l'École Nationale Supérieure des Techniques Avancées, MECM a permis de trouver 8 factorisations complètes de cette liste.

quoi	chiffres	quand	phase	itérations
$p(15737)$	22	22.08.92	2^{nde}	
$p(22079)$	23	12.10.92	2^{nde}	562579
$p(22157)$	26	10.11.92	2^{nde}	434407
$p(20173)$	26	29.11.92	2^{nde}	1372331
$p(18587)$	29	20.03.93	2^{nde}	2677133
$p(17791)$	29	28.04.93	2^{nde}	3056329
$p(12799)$	32	24.03.93	2^{nde}	689437
$p(14627)$	37	09.03.93	2^{nde}	2803103

Tableau 4 : Factorisations du challenge RSA trouvées par MECM.

En coopération avec F. Morain qui a développé un crible quadratique pouvant être mis en œuvre sur réseau de stations de travail communicant par mail, 5 autres entiers ayant des cofacteurs non premiers mais de taille raisonnable après l'utilisation de MECM ont pu être complètement factorisés.

quoi	taille entier	taille facteur	taille cofacteur	quand
$p(16981)$	141	18	90	14.12.92
$p(21059)$	157	19	89	26.12.92
$p(16369)$	138	24	78	09.01.93
$p(19079)$	149	27	95	20.01.93
$p(18911)$	149	15	95	01.03.93

Tableau 5 : Factorisations du challenge RSA finies avec le Crible Quadratique.

D'autres facteurs d'une trentaine de chiffres ont aussi pu être trouvé grâce à MECM ; la découverte du facteur 43 25491 26887 38803 96648 58367 en janvier 93 de $6.2^{203} - 1$ qui est un nombre de 122 chiffres du projet Cunningham (liste des nombres $b^n \pm 1$ avec $b \leq 12$) en est un exemple parmi d'autres.

Pour obtenir ces résultats, on a prèféré plutôt que de "s'acharner" sur chaque nombre en distribuant les courbes qui lui sont relatives aux 20 stations, donner un nombre différent à chaque machine.

Avec les paramètres couramment utilisés ;

- recherche des diviseurs triviaux inférieurs à $B_0 = 5.10^6$,
- utilisation de 25 courbes en même temps pour première et seconde phase classique,
- limite pour la première phase : $B_1 = 2.10^6$,
- limite pour la seconde phase classique : $B_2 = 10^7$,

il faut alors environ une journée de temps utilisateur sur une HP 9000/710 pour chaque nombre. Étant donné la quantité d'entiers à factoriser, ces choix semblent un compromis acceptable entre les 2 autres stratégies qui consistent à :

1. utiliser peu de courbes (typiquement une ou deux en paramétrisation de Montgomery) avec des bornes B_1, B_2 importantes.
2. utiliser un grand nombre de courbes (typiquement plusieurs centaines en paramétrisation de Weierstrass) avec des bornes B_1, B_2 petites.

La seconde phase "paradoxe des anniversaires", n'étant quant à elle au point que depuis peu, n'a pas encore de facteurs à son actif mais elle devrait néanmoins permettre de trouver encore quelques facteurs de la liste RSA.

Chapitre 7

Conclusion

Si la première phase de l'algorithme de Lenstra avec la seconde phase classique est maintenant bien rodée (la découverte de facteurs d'une trentaine de chiffres en atteste), la seconde phase du "paradoxe des anniversaires" a été encore peu utilisée. Une utilisation intensive devrait néanmoins permettre cet été de mieux cerner quelle est l'efficacité de cette dernière en pratique avec à la clef la découverte de nouvelles factorisations.

Les routines développées ici pour obtenir une algorithmique efficace ne sont pas toutes spécifiques à la factorisation. Ainsi, l'arithmétique liée aux courbes elliptiques et aux polynômes est d'un usage général et pourra certainement être réemployée dans d'autres domaines ; calcul du nombre de points sur une courbe elliptique de $\mathbf{GF}(p)$, logarithme discret sur courbes elliptiques ...

Bibliographie

[1] RSA Challenge Administrator. Information about rsa factoring challenge. Send electronic mail to challenge-info@rsa.com, March 1991.

[2] John E. Hopcroft Alfred V. Aho and Jeffrey D.Ullman. *The design and analysis of computer algorithms*. Reading. Addison–Wesley, 1974.

[3] A. O. L. Atkin and F. Morain. Finding suitable curves for the elliptic curve method of factorization. To appear in *Mathematics of Computation*. Also available as INRIA Research Report no 1547, March 1991.

[4] J. Vuillemin B. Serpette and J.C. Herve. Bignum : A portable and efficient package for arbitrary-precision arithmetic. Technical report, Digital Equipment Corporation and INRIA, 1989. Can be get at doc-server@prl.dec.com.

[5] R. P. Brent. Some integer factorization algorithms using elliptic curves. In *Proc. 9th Australian Computer Science Conference*, February 1986.

[6] J. Buhler, H. W. Lenstra, Jr., and Carl Pomerance. Factoring integers with the number field sieve. Preprint (version 19920507), 1992.

[7] D. E. Knuth. *The Art of Computer Programming : Seminumerical Algorithms*. Addison-Wesley, 1981.

[8] A. Shamir L. M. Adleman, R. L. Rivest. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM*, 21(2) :120–126, 1978.

[9] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard. The number field sieve. Preprint, November 1989.

[10] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard. The number field sieve. In *Proceedings 22nd Annual ACM Symposium on Theory of Computing (STOC)*, 1990.

[11] Arjen K. Lenstra. Massively parallel computing and factoring. LNCS 583, LATIN'92 proceedings, 1987.

[12] Hendrik W. Lenstra, Jr. Factoring integers with elliptic curves. *Annals of Math.*, 126 :649–673, 1987.

[13] P. L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Math. Comp.*, 48(177) :243–264, January 1987.

[14] P. L. Montgomery. An fft extension of the elliptic curve method of factorisation. Master's thesis, University of California - Los Angeles, March 1992.

[15] P. L. Montgomery and B. Silverman. An FFT extension to the $P-1$ factoring algorithm. *Math. Comp.*, 54(190) :839–854, April 1990.

[16] F. Morain and J. Olivos. Speeding up the computations on an elliptic curve using addition-substractions chains. *R.A.I.R.O. Theoritical Informatics and Applications 24*, 6 :531–543, 1990.

[17] H. J. Nussbaumer. *Fast Fourier transform and convolution algorithms*, volume 2 of *Springer Series in Information Sciences*. Springer-Verlag, 2 edition, 1982.

[18] J. M. Pollard. Theorems on factorization and primality testing. *Proc. Cambr. Philos. Soc.*, 76 :521–528, 1974.

[19] J. H. Silverman. *The arithmetic of elliptic curves*, volume 106 of *Graduate Texts in Mathematics*. Springer, 1986.

[20] Hiromi Suyama. Informal preliminary report (8). 25 Oct 1985.