Introduction au domaine de recherche

Cyril Bouvier

3 novembre 2011

Table des matières

Chapitre 1

Introduction au domaine de recherche

Factorisation : algorithmes et implémentations.

Introduction

La factorisation d'entier est un problème fondamental en algorithmique arithmétique : la sécurité de l'algorithme RSA [22] utilisé dans les cartes bancaires actuelles est basée sur la difficulté de factoriser un entier produit de deux nombres premiers de même taille. Mesurer la difficulté de ce problème avec les meilleurs algorithmes connus et les architectures actuelles permet donc d'estimer la sécurité obtenue en fonction des tailles de clés utilisées.

Dans une première partie, on definira des notions souvent utilisées en théorie algorithmiques des nombres, ensuite dans une seconde partie on présentera quelques algorithmes de factorisation et l'on soulèvera les problèmes d'implémentations qui y sont liés.

Dans toute la suite, N désignera un entier que l'on cherche à factoriser.

1.1 Définitions

1.1.1 Complexité

Pour pouvoir comparer deux algorithmes indépendamment de leurs implémentations, on utilise la notion de *complexité*. La complexité d'un algorithme est définie comme le nombre d'opérations élémentaires nécessaires pour résoudre le problème posé. Le nombre exactes d'opérations élémentaires est rarement calculable alors ce que l'on étudie est l'évolution de la complexité en fonction de

la taille des données d'entrée de l'algortihme.

Pour les algorithmes de factorisation, on exprimera la complexité en fonction de N (l'entier que l'on veut factoriser) ou du plus petit facteur premier de N (que l'on note souvent p).

Par exemple, l'algorithme de factorisation naïf (appelé *trial division*) qui consiste à diviser par tous les premiers inférieurs à la racine de N à une complexité en $O(p)$ (et donc $O(\sqrt{N})$ dans le pire cas), si l'on compte les divisions comme des opérations élementaires.

1.1.2 Notation L

Pour simplifier les notations, on utilise souvent la fonction suivante pour exprimer la complexité des algorithmes de factorisation.

Définition 1.1.1. On définit, pour toute constante c positive et pour tout $\alpha \in [0, 1]$ la fonction (en la variable x) suivante :

$$L_x(\alpha, c) = e^{((c+o(1))(\ln x)^{\alpha}(\ln \ln x)^{1-\alpha})}$$

Remarque 1.1.2. Remarquons d'abord les deux cas particulier :
- $\alpha = 0 : L_x(0, c) = e^{((c+o(1)) \ln \ln x)} = (\ln x)^{(c+o(1))}$.
- $\alpha = 1 : L_x(1, c) = e^{((c+o(1)) \ln x)} = x^{(c+o(1))}$.

La valeur de α permet de classer les algorithmes ayant un complexité en $L_N(\alpha, c)$. Dans le cas de $\alpha = 0$ on parle de complexité polynomiale, pour $\alpha = 1$ on parle de complexité exponentielle et pour $0 < \alpha < 1$ on parle de complexité sous-exponentielle.

Dans la suite on utilisera cette notation pour exprimer la complexité des algorithmes en fonction de N ou p. Par exemple, l'algorithme de factorisation naïf à une complexité exponentielle ($L_N(1/2, 1) = O(\sqrt{N})$) tandis que tous les algorithmes qui seront présentés dans la partie suivante ont un complexité sous-exponentielle. Il n'existe à ce jour aucun algorithme de factorisation ayant une complexité polynomiale.

1.1.3 Friabilité

La notion de *friabilité* est une notion importante en factorisation, elle mesure, en un certain sens, la difficulté d'un nombre à être factorisé.

Définition 1.1.3. Soit $x, y \in \mathbb{N}$. L'entier x est dit y-friable (ou y-lisse) si pour tout premier p divisant x alors p est inférieur à y. L'entier x est dit y-superfriable si pour tout premier p et pour tout entier e tels que p^e divise x alors p^e est inférieur à y.

Exemple 1.1.4. L'entier $424242 = 2 \cdot 3^2 \cdot 7^2 \cdot 13 \cdot 37$ est 37-friable mais pas 37-superfriable.

Le théorème de Canfield, Erdős et Pomerance [6] donne une estimation asymptoptique du nombre d'entier friable.

Théorème 1.1.5. *On pose*

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

En notant $u = \ln x / \ln y$,

$$\psi(x,y) = xu^{-u(1+o(1))},$$

uniformément pour $x \to \infty$ *si* $(\ln x)^\epsilon < u < (\ln x)^{1-\epsilon}$ *pour un* ϵ *fixé dans* $]0,1[$.

Un corollaire utile pour l'analyse de complexité des algorithmes de factorisation est qu'un entier aléatoire de taille $L_x(\alpha, c)$ est $L_x(\beta, c')$-friable avec probabilité environ $L_x(\alpha - \beta, -\frac{c}{c'}(\alpha - \beta))$.

1.2 Algorithmes de factorisation

On va, dans cette partie, décrire trois algorithmes de factorisations et soulever les problèmes qui peuvent apparaître lors de leurs implémentations.

1.2.1 L'algorithme $p-1$

Le premier algorithme que l'on étudiera est l'algorithme $p-1$, proposé par Pollard en 1974 [19].

Description de l'algorithme :

- Choisir tout d'abord un entier B. Cet entier sert à contrôler la quantité de calcul que l'on est prêt à faire. On pose

$$s = \prod_{\substack{\pi \leq B \\ \pi \text{ premier}}} \pi^{\lfloor \log(B)/\log(\pi) \rfloor}.$$

- Choisir ensuite un entier a au hasard dans $[1, N-1]$. On suppose que a est premier avec N (si ce n'est pas le cas le pgcd de a avec N donne un facteur non trivial de N et on peut donc s'arrêter là).
- On espère trouver un facteur en calculant : $\text{pgcd}((a^s \mod N) - 1, N)$.

L'algorithme retournera un facteur de N différent de 1 s'il existe un premier p divisant N tel que $p-1$ est B-superfriable. En effet, soit p un facteur de N. Comme a est premier avec N, $(a \mod p)$ est non nul, donc $(a \mod p)$ est d'ordre $p-1$ dans $\mathbb{Z}/p\mathbb{Z}^*$. Donc si $p-1$ est B-superfriable, ce qui signifie que s est un multiple de $p-1$, alors $a^s \equiv 1 \pmod{p}$. C'est-à-dire que p divisera $(a^s \mod N) - 1$. L'algorithme retourne donc le produit de tous les premiers p divisant N tels que $p-1$ est B-superfriable.

La complexité de l'algorithme en fonction de B est en $O(B \log B (\log N)^2)$. Le choix de la borne B est le résultat d'un compromis : plus B est grand, plus on a de chance de trouver un facteur mais plus les calculs sont longs. Le corollaire du théorème 1.1.5 permet de quantifier ce compromis. Pour trouver un facteur p de N, il faut choisir $B = L_p(1/2, 1/\sqrt{2})$, pour une probabilité de réussite en

$L_p(1/2, -\frac{\sqrt{2}}{2})$ et une complexité en $L_p(1/2, 1/\sqrt{2})$. On choisit donc la borne B en fonction de la taille des facteurs que l'on souhaite trouver. La complexité de l'algorithme dépend donc de la taille du plus petit facteur de N.

Remarques 1.2.1.

- Les nombres N et s pouvant être très grands (plusieurs centaines de chiffres pour N, plusieurs millions pour s) il est nécessaire d'implémenter efficacemment le calcul de $a^s \mod N$. Ce problème est connu sous le nom d'*exponentiation modulaire* [13, chap. 4] [4, chap. 2.6].
- La valeur de B ne pouvant pas être infiniment augmentée car le temps de calcul deviendrait trop long (en pratique pas plus de $B = 10^{10}$), l'algorithme peut se révéler inefficace. Soit N est tel que l'algorithme va trouver des facteurs, soit, pour la valeur de B que l'on est prêt à essayer (*i.e.* le temps que l'on est prêt à dépenser), l'algorithme ne trouvera pas de facteurs.
- Pour augmenter la probabilité de réussite de l'algorithme, on effectue souvent une étape de calcul supplémentaire, appelée *étape 2*. Pour une description de l'étape 2 voir [14, Chap. 2]. On choisit pour cela un entier B_2 supérieur à B. Cet ajout permet de récupérer tous les facteurs premiers p tels que leur plus grand facteur permier π est compris entre B et B_2 et tel que p/π est B-superfriable. L'étape 2 est souvent négligeable en terme de temps d'exécution mais peut exiger énormément de mémoire. Elle ne change pas la complexité de l'algorithme mais augmente la probabilité de trouver un facteur de N.
- En 1982, H. C. Williams proposa un algorithme, appelé $p+1$, qui permet de trouver un facteur p de N si $p+1$ est B-superfriable [25]. Cela a ensuite été généralisé en 1989 [1] à tous les polynômes cyclotomiques en p ($p-1$, $p+1$, p^2+p+1, ...).

1.2.2 ECM, la factorisation *via* les courbes elliptiques

L'algorithme ECM (*Elliptic Curve Method*), proposé par H. Lenstra en 1987 [11], utilise les propriétés des courbes elliptiques sur les corps finis pour factoriser des entiers. L'algorithme ECM repose sur les mêmes idées que l'algorithme $p-1$ où l'on remplace les groupe $(\mathbb{Z}/N\mathbb{Z})^*$ et $(\mathbb{Z}/p\mathbb{Z})^*$ par des groupes de courbes elliptiques sur les corps finis.

Une courbe elliptique est la donnée d'une courbe algébrique projective lisse de genre 1 et d'un point O sur cette courbe. En caractéristique différente de 2 et 3, toute courbe elliptique peut se voir comme l'ensemble des couple (x, y) vérifiant une équation de la forme $y^2 = x^3 + ax^2 + b^3$, appelée forme de Weierstrass, auquelle on ajoute un point a l'infini O. On peut munir toute courbe elliptique d'une loi de groupe commutative d'élément neutre O. Cette loi de groupe s'exprime de façon rationelle (donc calculable) en fonction des coordonnées des points sur la courbe. Cette loi de groupe a aussi une interprétation géométrique très simple (appelée méthode des tangentes et des sécantes), comme illustré dans la figure 1.1.

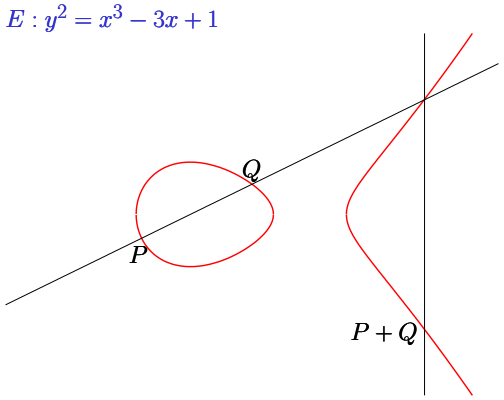

FIGURE 1.1 – Interprétation géométrique de la loi de groupe sur une courbe elliptique.

Description de l'algorithme :

- Choisir une courbe elliptique E sur $\mathbb{Q}$ d'élément neutre O_E. Soit P un point de E qui n'est pas un point de torsion. On suppose que la courbe a bonne réduction modulo tous les premiers p divisant N (cela se vérifie à l'aide de calculs de pgcd), sinon on en choisit une autre. Cela signifie que vu modulo p, la courbe est encore une courbe elliptique.
- Choisir un entier B. On pose, comme lors de l'algorithme $p-1$,

$$s = \prod_{\substack{\pi \leq B \\ \pi \text{ premier}}} \pi^{\lfloor \log(B)/\log(\pi) \rfloor}.$$

- Calculer le point $(x, y) := [s]P$ ($= P + \cdots + P$ répété s fois) en effectuant tous les calculs intermédiaires modulo N. Cela revient à ne plus voir la courbe sur $\mathbb{Q}$ mais sur $\mathbb{Z}/N\mathbb{Z}$. La seule opération qui n'existe pas toujours dans $\mathbb{Z}/N\mathbb{Z}$ est l'inverse. Mais un élément non inversible de $\mathbb{Z}/N\mathbb{Z}$ est un élément qui n'est pas premier avec N, il fournit donc un facteur de N différent de 1.
- Si le calcul termine ou si l'on tombe sur le point à l'infini de la courbe modulo N, on recommence le calcul avec une autre courbe.

Exemple 1.2.2. ECM sur un petit exemple.

On prend, pour cet exemple, $N = 4453$. On considère la courbe donnée par $y^2 = x^3 + 3x$, avec comme point initial $P = (1, 2)$. On choisit $B = 3$, on a donc $s = 6$. En effectuant les calculs modulo N, on obtient $P = (1, 2)$, $3P = (1003, 610)$ et une erreur lors du calcul de $6P$. En effet sur $\mathbb{Q}$, $6P = (\frac{51825601}{16208676}, -\frac{424317173273}{65256129576})$. Comme $\text{pgcd}(16208676, 4453) = 61$, le calcul modulo N ne peut aboutir mais on a obtenu un facteur de N.

On trouve donc $N = 4453 = 61 \cdot 73$.

L'algorithme retournera tous les facteurs p tels que l'ordre du groupe formé par la courbe modulo p est B-superfriable.

Comme pour $p-1$, il y a un compromis pour le choix de B entre le temps de calcul et la probabilité de réussite. Le temps d'exécution d'ECM pour trouver un facteur p d'un nombre N est $L_p(1/2, \sqrt{2})$ (on obtient ce résultat en combinant le théorème d'Hasse-Weil sur les cardinaux des courbes elliptiques sur les corps finis et le corollaire du théorème 1.1.5)

L'algorithme ECM est plus efficace que l'algorithme $p-1$ car on peut essayer plusieurs courbes avec des cardinaux différents (pour la méthode $p-1$ le cardinal du groupe est toujours $p-1$). Il est même plus avantageux de faire le calcul sur plusieurs courbes avec un petit B que d'augmenter la valeur de B sur la même courbe.

Il existe plusieurs similitudes avec $p-1$ (et même $p+1$). On peut en effet décrire tous ces algorithmes avec des groupes abstraits vérifiant certaines propriétés. Ils partagent donc certaines améliorations (l'étape 2, ...).

Remarques 1.2.3.

- Il existe plusieurs forme sous laquelle une même courbe elliptique peut s'écrire (Weierstrass, Montgomery [16], Edwards [9, 2] , ...). Il faut bien choisir la forme sous laquelle on représente la courbe car elle influe sur la rapidité de la loi de groupe.
- Comment générer des courbes où l'on connaisse un point de la courbe ? En effet trouver un point sur une courbe elliptique aléatoire revient à trouver une racine carrée modulo N ce qui est équivalent à factoriser N... Il existe des familles de courbes (donnée avec un point) paramétriser par les nombres rationnelles (paramétrisation de Montgomery [16], de Suyama [24], ...).
- Comment calculer efficacement la multiplication scalaire sur la courbe (c'est-à-dire $[s]P$) ? Le problème est similaire au problème d'exponentiation modulaire rencontré dans $p-1$ mais la loi de groupe plus compliquée oblige à une étude plus importante pour favoriser certaines opérations plus faciles (comme un carré par rapport à une multiplication). Il existe différentes méthodes (Montgomery Ladder [16], PRAC [17], NAF [10, p. 98], ...) plus ou moins adaptées aux différentes formes de courbes. Si une forme de courbes a une loi de groupe qui permet d'effectuer des doublements ou des triplements efficacemment, il faut utiliser une méthode qui en tire profit.

Il existe un grand nombre d'implémentations de cette algorithme sur plusieurs architecture : processeurs classiques [8], cartes graphiques (plus de détail dans le rapport de stage de Master 2 en annexe **??**), Playstation 3 [3]... Chaque nouvelle architecture soulève de nouveaux problèmes d'implémentation

1.2.3 Congruences de carrés

La congruences de carrés est plus une méthode générale qu'un algorithme particulier. Elle a donné lieu à plusieurs algorithme, dont un seul sera présenté dans cette partie : l'algorithme de Dixon.

L'idée principale remonte à Fermat : il suffit d'écrire N comme une différence

de deux carrées. Par exemple $8051 = 8100 - 49 = 90^2 - 7^2 = 83 \times 97$. De plus l'identité $ab = (\frac{1}{2}(a+b))^2 - (\frac{1}{2}(a-b))^2$ nous montre que cette méthode fonctionne pour tous les nombres impairs. Elle fonctionne bien si $N = ab$ est tel que a et b sont proche de la racine de N. Sinon la recherche exhaustive de la différence de carré devient trop longue.

Dans les années 1920, Kraitchik remarqua qu'il suffisait de trouver deux entiers u et v tels que $u^2 - v^2 \equiv 0 \pmod N$, en espérant que l'on n'ait pas $u \equiv \pm v \pmod N$. Pour un N impair divisible par au moins deux premiers, on a $u \not\equiv \pm v \pmod N$ dans plus de la moitié des cas. Un facteur non trivial peut alors être trouvé en calculant $\text{pgcd}(u - v, N)$.

Kraitchik remarqua aussi que si t^2 et s^2 ne sont pas des carrés modulo N, on peut les multiplier et espérer que leur produit soit un carré modulo N. C'est ce qu'on appelle la *combinaison de congruence*.

Toutes ces idées ont été formalisées par Dixon dans l'algorithme qu'il a proposé en 1981 [7].

Description de l'algorithme :

- Choisir un entier B. On note $\mathcal{P} := \{p \text{ premier}, p < B\}$ et k le nombre de premiers inférieurs à B (c'est-à-dire le cardinal de $\mathcal{P}$).
- Pour x pris aléatoirement dans $[\sqrt{N}, N-1]$, calculer $y = x^2 \mod N$. Si y est B-friable, on garde le couple (x, y), appelé *relation*.
- On appelle $\mathcal{R}$ l'ensemble des relations et m son cardinal. On répète l'étape précedente jusqu'à avoir $m > k$.
- Pour $1 \leq i \leq m$ on note (x_i, y_i) la i-ième relation et on définit $v_{i,p}$ par $x_i^2 \mod N = y_i = \prod_{p \in \mathcal{P}} p^{v_{i,p}}$.
- Construire la matrice $M = (v_{i,p} \mod 2)_{p \in \mathcal{P}, 1 \leq i \leq m}$.
- Trouver un vecteur non nul $(e_1, \ldots, e_m) \in \mathbb{Z}/2\mathbb{Z}^m$ du noyau de M
- On obtient la congruence de carrés voulu en calculant $\prod_{1 \leq i \leq m} x_i^{2e_i}$

La dernière étape fournit bien une congruence de carrés car :

$$\prod_{1 \leq i \leq m} x_i^{2e_i} \equiv \prod_{1 \leq i \leq m} \prod_{p \in \mathcal{P}} p^{v_{i,p} e_i} \equiv \prod_{p \in \mathcal{P}} p^{\sum_{1 \leq i \leq m} v_{i,p} e_i} \mod N.$$

Et par définition du vecteur $(e_1, \ldots, e_m)$, $\sum_{1 \leq i \leq m} v_{i,p} e_i \equiv 0 \mod 2$. On a donc bien une congruence de carrés ($u^2 \equiv v^2 \mod N$ avec $u = \prod_{1 \leq i \leq m} x_i^{e_i}$ et $v = \prod_{p \in \mathcal{P}} p^{\frac{1}{2} \sum_{1 \leq i \leq m} v_{i,p} e_i}$) qui, avec une probabilité supérieure à $1/2$ (si N est impair et possède au moins deux facteurs premiers), donnera un facteur non trivial de N.

De même que pour les autres algorithmes, le choix de B est très important. Si l'on choisit B trop grand, le nombre de relations nécessaires est trop important et si l'on choisit B trop petit on passe trop de temps pour trouver une relation. Le meilleur compromis donne un complexité en $O(L_N(1/2, 2))$. On remarque que, contrairement aux algorithmes décrit précédement, la complexité dépend de N et non du plus petit facteur premier p. En effet, les algorithmes reposant sur la congruence de carrés sont plus efficaces pour factoriser des entiers de type $N = pq$ avec p et q premiers que pour trouver des petits facteurs de grands entiers.

Il existent différentes méthodes pour touver une congruence de carrés qui ont donné lieu à différents algorithmes :

- CFRAC [15, 18] utilisant les fractions continues ;
- QS [20] (*Quadratic Sieve*) et MPQS [23] (*Multiple Polynomial Quadratic Sieve*) introduisant au passage une nouvelle méthode pour trouver plus rapidement des relations : le crible ;
- SNFS (*Special Number Field Sieve*) puis GNFS (*General Number Field Sieve*) [5] utilisant de la théorie algébrique des nombres et reposant sur la factorisation d'idéaux dans des anneaux d'entiers.

La fin de l'algorithme (recherche d'un vecteur dans le noyau d'une matrice et calcul du pgcd) est identique pour tous ces algorithmes.

Pour une description historique et mathématique de ces algorithmes on pourra consulter [21].

GNFS a une complexité en $L_N(1/3, \sqrt[3]{\frac{64}{9}})$, c'est actuellement la meilleure complexité pour un algorithme de factorisation, et c'est donc l'algorithme le plus adapté pour factoriser des très grands entiers (plus de 100 chiffres).

L'implémentation des ces algorithmes a soulevé de nouveaux problèmes :

- Ces algorithmes demandent énormément de calculs, on peut donc se demander quelles sont les étapes que l'on peut paralléliser (*i.e.* effectuer en même temps sur plusieurs threads/cœurs/ordinateurs).
- Lors de la dernière étape de l'algorithme, la matrice peut être très grande (plusieurs millions de ligne et colonnes) mais possède des propriétés particulères (elle est creuse). Il est donc nécessaire d'avoir des algorithmes efficaces pour ce genre de matrices contre lesquelles le pivot de Gauss est innefficace.

1.2.4 Quel algorithme utiliser ?

Que faire face à entier que l'on veut factoriser et dont on ne connaît pas a priori la taille des facteurs. Tout d'abord il ne faut pas négliger la méthode naïve pour éliminer les petits facteurs. Un entier aléatoire a 92% de chance de posséder un facteur inférieur à 1000. Et pour des petits facteurs (moins de 10 chiffres), la méthode naïve est la plus rapide. Ensuite on peut essayer les algorithmes $p-1$ et $p+1$ mais c'est surtout ECM qui sert à trouver les facteurs de taille moyenne. Enfin, suivant le taille de ce qu'il reste à factoriser, on utilise MPQS (moins de 100 chiffres) ou NFS (plus de 100 chiffres) ou on abandonne...

Les tableaux ci-dessous résument les records de factorisation pour les algorithmes les plus répandus. Les records ont été séparés en deux tableaux, le premier pour les algorithmes dont la complexité dépend du plus petit facteur premier p de N (qui servent plutôt à éliminer les petits et moyens facteurs de grands entiers) et le second pour les algorithmes dont la complexité dépend de N (qui servent plutôt à factoriser des entiers sous la forme pq avec p et q premiers).

L'algorithme SNFS est mis à part car il ne fonctionne que pour des entiers de la forme $r^e \pm s$ où r et s sont très petits. Il est en particulier très adapté pour

Algorithme	Record (en nombre de chiffres)	Deuxième	Troisième
$p-1$	66	59	58
$p+1$	60	55	53
ECM	73	73	70

TABLE 1.1 – Record pour les algorithmes dont la complexité dépend de p.

Algorithme	Taille N	Taille des facteurs
MPQS	135	66 et 69
GNFS	232	116 et 116
SNFS	307	80 et 227

TABLE 1.2 – Record pour les algorithmes dont la complexité dépend de N.

factoriser les nombres de Mersenne $(2^k - 1)$ et de Fermat $(2^{2^n} + 1)$. Pour des nombres généraux, le record de factorisation est actuellement tenu par GNFS pour la factorisation de RSA-768 [12] (entre 1500 et 2000 années de calcul distribuées sur plusieurs centaines de machines). Plus de détails peuvent être trouvés en annexe **??** (rapport de stage de première année).

1.3 Conclusion

La factorisation est aussi bien un problème théorique (recherche de nouveaux algorithmes, étude précise de complexité, ...) que pratique (implémentation efficace des algorithmes).

Chaque nouvel algorithme élargit un peu plus le domaine de recherche de la factorisation. Il est ainsi utile d'étudier l'arithmétique algorithmique pour l'implémentation d'une arithmétique modulaire rapide, d'étudier les courbes elliptiques pour obtenir une loi de groupe plus rapide, d'étudier la théorie algébrique des nombres pour résoudre les problèmes que l'implémentation de GNFS pose, d'étudier les différentes architectures pour trouver la plus adaptée à un algorithme donné, ...

Bibliographie

[1] Eric Bach and Jeffrey Shallit. Factoring with cyclotomic polynomials. *Mathematics of Computation*, 52(185) :pp. 201–219, 1989.

[2] Daniel J. Bernstein, Tien-Ren Chen, Chen-Mou Cheng, Tanja Lange, and Bo-Yin Yang. ECM on Graphics Cards. Cryptology ePrint Archive, Report 2008/480, 2008. `http://eprint.iacr.org/`.

[3] Joppe W. Bos, Thorsten Kleinjung, Arjen K. Lenstra, and Peter L. Montgomery. Efficient SIMD arithmetic modulo a Mersenne number. Cryptology ePrint Archive, Report 2010/338, 2010.

[4] Richard Brent and Paul Zimmermann. *Modern Computer Arithmetic*, volume 18 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, 2010.

[5] Joe P. Buhler, Jr. Hendrik W. Lenstra, and Carl Pomerance. Factoring integers with the number field sieve. In Arjen K. Lenstra and Jr. Hendrik W. Lenstra, editors, *The development of the number field sieve*, volume 1554 of *Lecture Notes in Mathematics*, pages 50–94, Berlin, 1993. Springer-Verlag.

[6] E. Rodney Canfield, Paul Erdős, and Carl Pomerance. On a problem of Oppenheim concerning "factorisatio numerorum". *Journal of Number Theory*, 17 :1–28, 1983.

[7] John D. Dixon. Asymptotically fast factorization of integers. *Mathematics of Computation*, 36(153) :pp. 255–260, 1981.

[8] Bruce Dodson and Paul Zimmermann. 20 years of ECM. In F. Hess, S. Pauli, and M. Pohst, editors, *7th Algorithmic Number Theory Symposium (ANTS VII)*, volume 4076 of *Lecture Notes in Computer Science*, pages 525–542, Berlin/Germany Allemagne, 2006. Springer Verlag.

[9] Harold M. Edwards. A normal form for elliptic curves. *Bulletin of the American Mathematical Society*, 44 :393–422, 2007.

[10] Darrel Hankerson, Alfred Menezes, and Scott Vanstone. *Guide to elliptic curve cryptography*. Springer, New York, 2004.

[11] H. W. Lenstra, Jr. Factoring integers with elliptic curves. *The Annals of Mathematics*, 126(3) :pp. 649–673, 1987.

[12] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen Lenstra, Emmanuel Thomé, Joppe Bos, Pierrick Gaudry, Alexander Kruppa, Peter Montgomery, Dag Arne Osvik, Herman te Riele, Andrey Timofeev, and Paul Zimmermann. Factorization of a 768-bit rsa modulus. Cryptology ePrint Archive, Report 2010/006, 2010. `http://eprint.iacr.org/`.

[13] Donald E. Knuth. *The art of computer programming, volume 2 : seminumerical algorithms*. Addison-Wesley, Reading, 1st (1st printing) edition, 1969.

[14] Alexander Kruppa. *Améliorations de la multiplication et de la factorisation d'entier*. These, Université Henri Poincaré - Nancy I, 2010.

[15] D. H. Lehmer and R. E. Powers. On factoring large numbers. *Bulletin of the American Mathematical Society*, 37(10) :pp. 770–776, 1931.

[16] Peter L. Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Mathematics of Computation*, 48(177) :pp. 243–264, 1987.

[17] Peter L. Montgomery. Evaluating recurrences of form $X_{m+n} = f(X_m, X_n, X_{m-n})$ via Lucas chains, 1992.

[18] Michael A. Morrison and John Brillhart. A method of factoring and the factorization of f_7. *Mathematics of Computation*, 29(129) :pp. 183–205, 1975.

[19] J. Pollard. Theorems of factorization and primality testing. *Proc. Cambridge Philos. Soc.*, 76 :521–528, 1974.

[20] Carl Pomerance. The quadratic sieve factoring algorithm. In Thomas Beth, Norbert Cot, and Ingemar Ingemarsson, editors, *Advances in cryptology : EUROCRYPT '84*, volume 209 of *Lecture Notes in Computer Science*, pages 169–182, Berlin, 1985. Springer-Verlag.

[21] Carl Pomerance. A tale of two sieves. *Notices of the American Mathematical Society*, 43 :1473–1485, 1996.

[22] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2) :120–126, February 1978.

[23] R. D. Silverman. The multiple polynomial quadratic sieve. *Mathematics of Computation*, 48(177) :329–339, 1987.

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

[25] H. C. Williams. A $p+1$ method of factoring. *Mathematics of Computation*, 39(159) :pp. 225–234, 1982.