UNIVERSITÉ DE LORRAINE

École doctorate IAEM
DFD Informatique
Laboratoire LORIA

Théorie algorithmique des nombres et applications à la cryptanalyse de primitives cryptographiques

Emmanuel Thomé

Mémoire présenté et soutenu publiquement le 13 décembre 2012
pour l'obtention de l'

Habilitation à diriger des recherches
(spécialité informatique)

de l'Université de Lorraine

Composition du jury :

Rapporteurs :	Alfred Menezes, professeur à l'Université de Waterloo Bruno Salvy, directeur de recherche à l'INRIA Karim Belabas, professeur à l'Université de Bordeaux
Examinateurs :	Arjen K. Lenstra, professeur à l'École polytechnique fédérale de Lausanne Jean-Yves Marion, professeur à l'université de Lorraine Paul Zimmermann, directeur de recherche à l'INRIA Jean-Michel Muller, directeur de recherche au CNRS

Table des matières

Chapitre 1

Introduction

1.1 Structuration de ce document

Le présent document est divisé en différentes parties. Ce chapitre d'introduction indique les motivations très générales de nos travaux, et explique notamment notre approche mêlant à la fois la considération d'objets mathématiques et la mise en œuvre algorithmique des idées proposées.

L'essentiel du document est composé par les parties I à IV, relatives aux différents axes de nos travaux. Pour chacun de ces axes, la partie correspondante propose un ou deux chapitres de présentation des enjeux de la thématique, suivi de plusieurs développements donnant les idées directrices essentielles des travaux que nous avons menés sur ce thème précis. Chacun de ces développements fait l'objet d'un chapitre séparé, ces chapitres pouvant recouvrir une ou plusieurs publications, ou un travail d'implantation.

Ce document se conclut par un chapitre de perspectives (partie V, contenant l'unique chapitre 17).

Ce mémoire n'est pas court, et ne se cantonne pas à un résumé d'articles. Nous avons souhaité y introduire quelques éléments qui ne sont pas présents dans les articles correspondant aux travaux que nous abordons. Les chapitres introductifs de chaque partie (chapitres 2, 3, 7, 11, et 14) ont vocation à remplir la mission de synthèse incluse dans ce document, de présentation du contexte précis de nos travaux, et d'aiguillage vers nos principales contributions. Notre choix de composition de ce manuscrit nous a amené à développer ces derniers dans des chapitres spécifiques, par groupe de travaux, le contenu de ces derniers occupant souvent des pages plus techniques (chapitres 4, 5, 6, 8, 9, 10, 12, 13, 15, et 16).

Chacune des parties I à IV de ce mémoire fait notamment apparaître les plus marquants de nos travaux. Ainsi la partie I de ce mémoire, consacrée au crible algébrique, mentionne entre autres les articles [T8, T12] sur les problèmes RSA ou DH avec oracle, ou encore le record de factorisation RSA-768 [T13]. La partie II de ce mémoire traite du sujet du logarithme discret sur les (jacobiennes de) courbes, et développe notamment les articles [T7, T9, T15], étudiant respectivement l'impact des *large primes* sur la complexité du calcul de logarithme discret, l'impact de ces travaux sur la sécurité des courbes de genre 3, et enfin l'existence d'un algorithme de complexité heuristique dite $L[1/3]$ pour le logarithme discret sur une certaine famille de courbes. La partie III évoque notre contribution déterminante au succès et à l'originalité du record de factorisation RSA-768 [T13], au travers des spécificités de la résolution du système linéaire intervenant dans ce calcul. L'usage de grilles de calcul pour cette étape a par exemple été abordée dans l'article [T14]. Toujours sur ce thème de l'algèbre linéaire, une partie de notre activité scientifique a pris la forme du composant logiciel d'algèbre linéaire dans le programme CADO-NFS [90], qui constitue une implémentation de référence en la matière. Des éléments de cette implémentation sont discutés dans la partie III de ce mémoire. En dernier lieu, la partie IV de ce mémoire mentionne nos principaux travaux sur le thème arithmétique [T10, T5].

Le poids relatif plus important de la partie I de ce mémoire est en accord avec le positionnement

de l'algorithme du crible algébrique dans nos travaux. Le crible algébrique est à la fois un objet d'étude, que nous abordons au travers des chapitres de la partie I liés à nos travaux s'en rapprochant, mais aussi une source d'inspiration pour des algorithmes dans d'autres contextes. Ainsi, notre regard sur le crible algébrique a été le germe qui a mené aux travaux que nous abordons dans la partie II sur le problème du logarithme discret dans les jacobiennes de courbes. Le crible algébrique est aussi un cadre d'application, pour nos travaux d'algèbre linéaire couverts par la partie III. Enfin, les travaux évoqués dans la partie IV sont une prolongation naturelle de notre souci d'efficacité dans la mise en œuvre algorithmique, présent dans toutes les parties de notre travail.

1.2 Motivations cryptologiques

La plupart de nos travaux trouvent une motivation, souvent leur motivation principale, dans le contexte de la cryptologie à clé publique, dite asymétrique. Dans ce contexte, c'est en s'appuyant sur l'existence de problèmes mathématiques « à sens unique », difficiles dans un sens mais pas dans l'autre, qu'il est possible de mettre en place des protocoles où les tâches des participants diffèrent. Pour introduire nos travaux, nous partons de deux exemples importants de problèmes pour lesquels une telle asymétrie est patente, à savoir le problème de la factorisation et le problème du logarithme discret.

$$\text{Factorisation :} \quad N \in \mathbb{Z} \longmapsto \{(p_i, e_i)\} \text{ tels que } N = \prod_i p_i^{e_i};$$

$$\text{Logarithme discret :} \quad \mathsf{G} = \langle g \rangle \text{ groupe, } h \in \mathsf{G} \longmapsto x \in \mathbb{Z} \text{ tel que } h = g^x.$$

Nous offrons un bref exposé de la façon dont des protocoles cryptographiques naissent de ces deux contextes, et des problématiques sous-jacentes *dans la perspective de ce mémoire uniquement*. Notons que d'autres problèmes difficiles permettent de mettre sur pied des protocoles cryptographiques (théorie des codes, théorie des réseaux euclidiens, ...), mais ne sont pas abordés dans ce mémoire.

1.2.1 Primitives cryptographiques et protocoles

La mission de la cryptographie est d'assurer certaines garanties (confidentialité, authenticité par exemple) dans des contextes bien établis : on suppose par exemple qu'on est confronté à un attaquant disposant de certaines capacités bien définies, et on montre qu'il ne peut mettre en échec les garanties qu'on souhaite offrir.

La cryptographie accomplit sa mission à l'aide de protocoles. Un protocole cryptographique est un agencement assez vite complexe de *briques de base*, appelées primitives cryptographiques. Ces primitives sont le centre de ce mémoire, tandis que les protocoles qui les utilisent sortent (le plus souvent) du périmètre du mémoire. Les protocoles sont généralement bâtis sur certaines hypothèses de résistance des primitives. Ce mémoire est davantage concerné par l'étude théorique et pratique de ces hypothèses faites sur les primitives que sur les protocoles à proprement parler.

Il arrive que le terme de protocole soit employé dans ce mémoire, le plus souvent afin d'illustrer comment la primitive cryptographique qui nous intéresse est utilisée dans un contexte à peine plus élaboré. Notre usage de ce terme, au vu de son utilisation dans des contextes scientifiques voisins, peut être jugé comme impropre, au sens où notre intérêt reste réellement focalisé sur la primitive cryptographique, pas sur la façon dont un cryptosystème répondant à des contraintes que nous n'étudions pas peut être bâti. Par exemple, nous mentionnons un intérêt pour le *padding* dans la

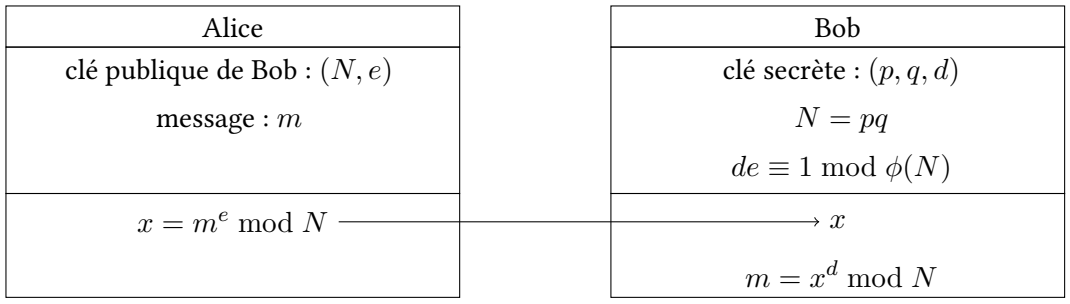


FIG. 1.1 : Le cryptosystème RSA (chiffrement)

signature RSA, pour nos travaux développés au chapitre 5. Cet intérêt nous fournit essentiellement un cas d'étude pour l'application de nos techniques, mais ne porte pas notre champ d'investigation vers la meilleure façon de définir un tel *padding* afin de faire de RSA une brique de base solide pour l'établissement de protocoles, travail mené par exemple par [197].

1.2.2 Le contexte de la factorisation

Dans le cas de la factorisation, on se convainc vite qu'alors que la multiplication d'entiers est une opération polynomiale, et même quasi-linéaire, en la taille des entrées, il est beaucoup plus ardu de factoriser : l'étude des algorithmes de factorisation d'entiers peut être l'objet de nombreux cours, et l'algorithme le plus avancé pour cette tâche, le crible algébrique, embarque à lui seul des pans entiers de la théorie algorithmique des nombres ; ce dernier algorithme occupe une place importante de ce mémoire, puisque la partie I et ses développements lui sont consacrés. Bâtir à partir de cette asymétrie entre multiplication et factorisation un cryptosystème, où la tâche « facile » est celle des participants « honnêtes », jouant le jeu du protocole, tandis que la tâche « difficile » incomberait à l'espion, n'est pas quelque chose d'évident a priori. Aujourd'hui, le cryptosystème RSA [178] (voir figure 1.1) qui offre une réponse partielle à cette question, est évidemment tellement connu que la subtilité du passage d'un problème difficile à un cryptosystème est parfois un peu facilement oubliée.

Lorsqu'on souhaite instancier le protocole RSA dans un cas concret, il convient de bien établir quelles sont les tâches, dans le jeu du protocole, qui incombent à Alice et Bob, et quelles sont les tâches qui incomberaient à un espion potentiel s'il voulait « casser » le cryptosystème :

- Bob doit fabriquer sa clé N, produit de deux nombres premiers.
- Alice et Bob doivent manipuler des entiers modulo N.
- Alice et Bob doivent effectuer des exponentiations modulaires modulo N.
- Un espion éventuel tente de résoudre l'équation $m^e = x \bmod N$, connaissant e, x, et N (*problème RSA*).

Problème RSA versus factorisation

L'examen de la difficulté de la tâche de l'espion, et de son lien avec le problème de la factorisation, révèle un obstacle. On ne connaît pas à ce jour de réduction polynomiale du problème RSA

au problème de la factorisation. C'est l'objet de l'un des problèmes auxquels nos travaux offrent un (petit) élément de réponse.

Problème 1.2. *Étant donnés deux entiers N et $e \leq N$, peut-on trouver une réduction du problème RSA (résoudre $m^e = x \bmod N$) au problème de la factorisation de N, la complexité étant entendue pour N croissant ? A contrario, existe-t-il un algorithme permettant de résoudre $m^e = x \bmod N$ plus vite qu'en factorisant N ?*

Le problème 1.2 est abordé au chapitre 5, ainsi que dans [T8] : dans le contexte particulier où l'accès à un certain oracle est possible, on répond positivement à la seconde question ci-dessus. Toutefois, notre étude ne change pas fondamentalement le constat suivant : sauf cas spécifiques (accès à un oracle par exemple), pour résoudre le problème RSA, le plus réaliste est de tenter de factoriser N.

NFS et les limites de la factorisation

Les paramètres du cryptosystème RSA doivent être choisis de sorte à ce que la tâche de l'espion soit insurmontable, au moins en pratique. Il est important d'étudier quelles sont les limites de ce qui est possible en terme d'attaque.

Pour tenter de factoriser des nombres qui sont difficiles à factoriser (le cas le plus représentatif étant celui de nombres $N = pq$, produits de deux nombres premiers p et q de même taille), il convient d'employer l'algorithme le plus efficace, à savoir le *crible algébrique* (NFS) [136], décrit plus complètement au chapitre 2. La complexité de cet algorithme dépend du nombre N à factoriser[1]. Sous certaines hypothèses heuristiques, cette complexité s'exprime à l'aide de la fonction sous-exponentielle :

$$L_N[1/3, (64/9)^{\frac{1}{3}}]^{1+o(1)} = \exp\left((1+o(1))(64/9)^{\frac{1}{3}}(\log N)^{\frac{1}{3}}(\log\log N)^{\frac{2}{3}}\right).$$

La description (et la pratique) de NFS laissent apparaître que le temps de calcul qu'il requiert *en pratique* pour factoriser un nombre d'une taille donnée est particulièrement difficile à extrapoler au vu de résultats expérimentaux sur des nombres de taille plus petite. Aussi, la complexité inhérente à cet algorithme et la multitude d'améliorations locales qui sont susceptibles de modifier son temps d'exécution nécessitent une évaluation principalement *par l'expérience* du temps et des ressources de calcul requises pour réaliser des factorisations d'une taille donnée. C'est l'objet d'un des problèmes que nous souhaitons mettre en exergue.

Problème 1.3. *À l'instant t, quelles ressources de calcul sont nécessaires pour factoriser un nombre d'une taille donnée ?*

Comme son énoncé l'indique, ce travail d'évaluation de l'état de l'art a vocation à être repris en permanence. Notre contribution à cette étude est la factorisation record du nombre RSA-768 [T13], dont quelques détails sont donnés au chapitre 6.3. Plus spécifiquement, chaque pierre qu'on peut apporter à l'édifice de l'optimisation du crible algébrique est une contribution au problème 1.3 ; c'est donc naturellement que nous présentons l'optimisation du crible algébrique comme un objectif en soi.

Problème 1.4. *Rechercher des améliorations pratiques dans les différentes étapes du crible algébrique.*

[1] Ceci s'entend en comparaison avec des algorithmes comme ECM [139], dont la complexité dépend principalement des facteurs p et q, la dépendance en $\log N$ étant polynomiale. Dans le cas qui nous intéresse, les facteurs p et q sont de taille maximale, et l'emploi de l'algorithme ECM n'offre pas d'avantage.

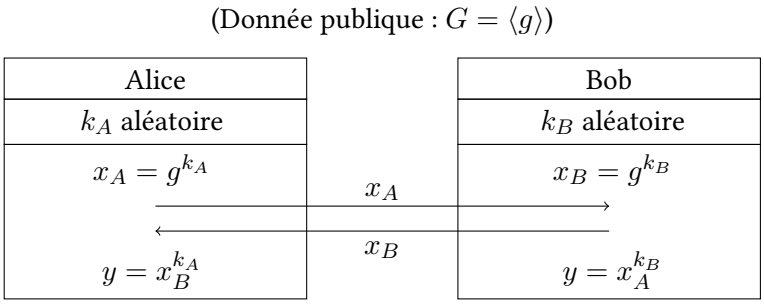


FIG. 1.5 : Échange de clés de Diffie-Hellman

Nos travaux dans le contexte de l'algèbre linéaire creuse sur $\mathbb{F}_2$, décrits dans la partie III, outre leur intérêt propre, s'inscrivent clairement dans cette perspective ; nous verrons aussi que l'étude au chapitre 16 de l'arithmétique des polynômes sur $\mathbb{F}_2$ est l'une des approches pour optimiser l'étape centrale de l'algorithme de Wiedemann par blocs [52], utilisé dans ces calculs.

1.2.3 Le contexte du logarithme discret

En comparaison avec le cas de la factorisation, le problème du logarithme discret donne de manière un peu plus directe une primitive cryptographique, à savoir l'échange de clés ; le protocole de Diffie-Hellman [60] (voir figure 1.5) a d'ailleurs été publié avant RSA, fournissant le premier exemple de cryptographie à clé publique. Le calcul de g^k dans un groupe cyclique s'effectue en $O(\log k)$ multiplications dans le groupe, tandis que lorsque le groupe n'est muni d'aucune autre structure particulière, retrouver k à partir de $x = g^k$ est difficile. Plus explicitement, bien qu'un groupe cyclique $G = \langle g \rangle$ soit accompagné d'un isomorphisme effectif :

$$\phi : \begin{cases} \mathbb{Z}/(\#G)\mathbb{Z} & \to & G, \\ k & \mapsto & g^k, \end{cases}$$

le calcul de ϕ^{-1} n'est pas a priori aussi aisé que celui de ϕ.

Le protocole de Diffie-Hellman propose un échange de clés, et on peut bien sûr souhaiter d'autres primitives (chiffrement, signature). La littérature abonde de tels protocoles, tels le protocole de chiffrement d'ElGamal ou l'algorithme de signature DSA. On renvoie aux textes de référence pour un traitement adéquat [150].

De manière similaire à ce qui a été vu dans le contexte de la factorisation, il convient de lister quelles sont les tâches d'Alice et Bob, et d'un espion éventuel.

- Alice et Bob doivent manipuler des éléments du groupe G.
- Alice et Bob doivent effectuer des exponentiations dans le groupe G.
- Un espion éventuel, connaissant g, ainsi que $x_A = g^{k_A}$ et $x_B = g^{k_B}$, tente de retrouver $y = g^{k_A k_B}$ (*problème DH*).

FIG. 1.9 : Deux boîtes noires pour un groupe générique

DH versus DL

Comme cela était déjà le cas pour RSA, on rencontre ici une situation où la tâche de l'espion (problème DH) n'est pas exactement le problème qu'on a énoncé initialement comme problème potentiellement difficile (problème DL). Toutefois, à la différence du cas RSA, il existe des résultats partiels permettant d'établir un lien entre DH et DL. Pour une large classe de groupes (le critère dépendant du cardinal du groupe), savoir résoudre DH donne un algorithme de résolution de DL [149, 163].

Notre travail [T12] mentionné en 5.4 fournit un analogue à la situation rencontrée pour le problème RSA : en présence d'un certain oracle, une instance faible du problème DH (DH statique) se résout plus facilement que DL. Ceci étant, comme précédemment, ce résultat ne remet pas en cause le fait que dans la pratique (par exemple en l'absence d'oracle), la résolution de DH passe en l'état actuel des connaissances par la résolution de DL.

DL dans un groupe générique

Les énoncés précédemment donnés pour les tâches d'Alice, Bob, et l'espion appellent une réponse immédiate, sous la forme d'une demande de précision : tout dépend du groupe G choisi.

Il est possible d'étudier le problème DL dans un cadre théorique avec un groupe « boîte noire » (figure 1.9). Un tel groupe n'est connu que via les deux algorithmes de la loi de groupe. Dans un tel cas, les trois résultats suivants, qui exploitent exclusivement ce contexte « boîte noire », sont de première importance.

PROPOSITION 1.6 (réduction de Pohlig-Hellman). *Soit G un groupe de cardinal n. Soit $n = \prod_{i=1}^{k} p_i^{e_i}$ la factorisation de n. Alors la résolution du problème DL dans G est équivalente à la résolution du problème DL dans l'ensemble des k sous-groupes de G de cardinaux $p_1, \ldots, p_k$.* (preuve : théorème chinois et lemme de Hensel).

PROPOSITION 1.7 (algorithme de Shanks « pas de bébé, pas de géant »). *Soit G un groupe de cardinal n. Il existe un algorithme déterministe dont la complexité est $O(\sqrt{n})$ appels à la loi de groupe, qui résout le problème DL dans G.*

PROPOSITION 1.8 (Nechaev-Shoup). *Soit G un groupe* générique *de cardinal n, et p un facteur premier de n. Alors un algorithme résolvant DL dans G requiert* au minimum *$\Omega(\sqrt{p})$ appels à la loi de groupe dans G.*

Les trois résultats précédents nous indiquent que *pour un groupe générique*, l'algorithme de Shanks est optimal, en ce sens qu'à une constante multiplicative près, combiné à la réduction de Pohlig-Hellman, il permet d'atteindre la borne inférieure donnée par le résultat de Nechaev et Shoup.

Ceci est un résultat théorique important, mais il convient toutefois d'en mesurer correctement la portée. Dans un cadre pratique, un groupe générique n'existe pas : dès lors qu'on *désigne* un groupe, il n'est pas générique. Aussi, le problème de la résolution de DL dans un groupe donné reste entier.

Desiderata pour l'utilisation d'un groupe en cryptographie

Au vu des calculs que doivent mener Alice, Bob, et l'éventuel espion dans l'échange de clés de Diffie-Hellman, on peut établir une liste de desidarata pour le choix d'un « bon » groupe pour l'utilisation en cryptographie. L'objectif est d'offrir à Alice et Bob la meilleure garantie contre l'attaque par l'espion, pour le minimum de ressources à utiliser de leur part.

Plus exactement, il est souhaitable d'obtenir une *famille* de groupes, puisqu'on souhaite être en mesure de régler un compromis entre sécurité et coût de manipulation. Ainsi, par « un groupe » G, on entend le résultat d'un algorithme de choix d'un groupe, choisi parmi une famille aussi riche que possible de groupes offrant divers niveaux de sécurité. On note par exemple une telle famille $\mathcal{G}$, et on voit G comme une instance particulière de $\mathcal{G}$. Plusieurs standards adoptent cette approche de propositions de familles de groupes. Il convient de remarquer que dans cette perspective, bien que le point de vue asymptotique des choses puisse se montrer éclairant à bien des égards, la préoccupation de la cryptographie telle qu'elle est mise en pratique est bel et bien celle de la sécurité et du coût de manipulation d'instances *concrètes* de groupes. Aussi, tout en considérant « un groupe » G comme une instance de $\mathcal{G}$, on conserve une attention accrue sur les groupes obtenus pour les paramètres les plus pertinents en pratique, et une attention moindre au cas (par exemple) de groupes bâtis dans l'objectif d'assurer une « sécurité » de 2^{10^6}. Aussi, bien que nous employions ponctuellement des terminologies asymptotiques dans ce qui suit, notre cible privilégiée reste une poignée d'instances concrètes.

Bien entendu, pour bâtir un cryptosystème autour du logarithme discret dans le groupe G, le plus grand sous-groupe cyclique d'ordre premier $H = \langle h \rangle$ de G est de première importance, en vertu de la réduction de Pohlig-Hellman. Il est souhaitable d'être en mesure de déterminer ce sous-groupe, donc en particulier de calculer le cardinal de G. On souhaite en outre disposer d'un moyen de représentation aussi compacte que possible des éléments de H (par exemple proche du minimum $\log_2 \#H$ bits). Les opérations de groupe sur les éléments de H, notamment l'exponentiation, doivent être efficaces, au maximum de complexité polynomiale en leur taille, à l'évidence.

Enfin, le calcul de logarithmes discrets dans H doit être aussi difficile que possible. Une hypothèse très forte serait d'exiger qu'il existe une borne inférieure pour ce calcul qui soit exponentielle en $\log_2 \#H$. On ne connaît pas à ce jour de classe de groupes ayant une telle propriété. Toutefois, la cryptologie des courbes algébriques fournit des exemples de groupes dans lesquels une telle borne correspond à l'état de l'art des algorithmes les plus efficaces connus.

On aboutit donc à quatre contraintes, résumées dans la liste suivante :

- calcul de la cardinalité possible ;
- représentation compacte ;
- loi de groupe efficace ;
- logarithme discret difficile.

Deux exemples Quelques choix sont naturels, mais ne permettent pas nécessairement de satisfaire pleinement les critères proposés plus haut. Par exemple, on peut s'intéresser à une famille constituée de groupes multiplicatifs de corps finis bien choisis. Supposons qu'on dispose par exemple d'une famille de nombres premiers $p_1, p_2, \ldots$, tels que pour chaque p_i, le plus grand facteur premier q_i de $p_i - 1$ est tel que[2] $\log q_i \geq \frac{1}{1+\epsilon} \log p_i$ pour une constante $\epsilon > 0$. La représentation des éléments du

[2]Il serait possible de viser $q_i \geq (p_i - 1)/\kappa$ pour $\kappa \geq 2$, mais nous ne souhaitons pas rentrer ici dans des considérations relatives à l'existence théorique d'une telle suite suffisamment riche de p_i pour des petites valeurs de κ.

sous-groupe H de cardinal q_i dans $\mathbb{F}_{p_i}^*$ nécessite $\log_2 p_i$ bits, soit $(1+\epsilon)\log_2 \#H$ bits. Les opérations dans H sont des opérations modulo p_i, donc une exponentiation modulaire comme le calcul de h^a pour $h \in H$ et $a \in \mathbb{Z}/q_i\mathbb{Z}$ coûte $O((\log p_i)^3)$ en supposant une arithmétique naïve.

Le bât blesse lorsqu'il s'agit de mesurer la difficulté du problème du logarithme discret dans un tel groupe H. En effet, comme nous le verrons au chapitre 3, le calcul de logarithmes discrets dans $\mathbb{F}_{p_i}^*$ (et donc dans H) a pour complexité heuristique :

$$L_{p_i}[1/3, (64/9)^{\frac{1}{3}}]^{1+o(1)} = \exp\left((1+o(1))(64/9)^{\frac{1}{3}}(\log p_i)^{\frac{1}{3}}(\log\log p_i)^{\frac{2}{3}}\right).$$

Cette expression est *sous-exponentielle* en $\log p_i$, soit largement en deçà de l'« idéal » (du point de vue du cryptographe).

L'utilisation des courbes algébriques fournit des exemples un peu meilleurs (si tant est qu'on accepte d'éviter les points de vue polémiques). Le premier exemple de la cryptologie utilisant des courbes algébriques, proposé par Miller et Koblitz [152, 130] est donné par le choix, pour une variété de nombres premiers p_i, de courbes elliptiques E_i non singulières, et définies sur $\mathbb{F}_{p_i}$. On choisit en outre des courbes elliptiques dont le cardinal est divisible par un facteur premier q_i vérifiant $\log q_i \geq \frac{1}{1+\epsilon}\log p_i$ pour une constante $\epsilon > 0$. On travaille alors dans le sous-groupe H de cardinal q_i du groupe $E_i(\mathbb{F}_{p_i})$ (dans le cas que nous étudions, q_i divise une seule fois p_i puisque $q_i^2 \gg p_i$. Le sous-groupe H est donc défini de manière univoque, et correspond au groupe des points de q_i-torsion de $E_i(\mathbb{F}_{p_i})$).

Le calcul du cardinal de $E_i(\mathbb{F}_{p_i})$ est un problème non trivial, mais de complexité polynomiale. Pour les tailles qui nous intéressent, ce problème est relativement aisé. La représentation des points de $E_i(\mathbb{F}_{p_i})$ occupe $2\log_2 p_i$ bits si on représente les coordonnées (x, y) complètement (ce qui n'est pas toujours nécessaire). Le calcul d'une « exponentiation modulaire », qui dans le groupe additif $E_i(\mathbb{F}_{p_i})$ revient à calculer un multiple $[a]Q$ d'un point Q, a pour complexité $O((\log q_i)^3)$. L'élément qui différencie vraiment le cas des courbes elliptiques du cas des corps finis est la complexité du calcul du logarithme discret. En effet, pour calculer un logarithme discret dans le sous-groupe H, aucun algorithme de complexité meilleure que $O(\sqrt{\#H})$ n'est connu à ce jour (sauf cas particuliers de certaines classes de courbes, faciles à éviter). Le logarithme discret dans les courbes elliptiques est donc, à taille égale, considérablement plus difficile que dans les corps finis. Si l'objectif est d'atteindre un niveau de sécurité donné, les ressources nécessaires à la mise en œuvre d'un cryptosystème utilisant les courbes elliptiques sont bien moindres que pour une entreprise similaire s'appuyant sur les corps finis.

Objectifs Les deux exemples que nous venons de donner ont pour seul objet d'illustrer le propos, et pas de prendre en compte les multiples ramifications des problèmes sous-jacents. Bien que le logarithme discret soit plus résistant dans les courbes elliptiques, l'étude du problème du logarithme discret dans les corps finis reste une problématique très actuelle, notamment en lien avec l'étude des couplages (qui ne sont pas abordés dans ce mémoire). Par ailleurs, même si l'on admet que l'utilisation des courbes algébriques offre un avantage décisif, en offrant plus de sécurité pour un moindre coût, il n'en reste pas moins primordial d'étudier les différentes facettes du compromis sécurité-coût de mise en œuvre pour les cryptosystèmes utilisant des courbes algébriques. Ces problèmes comportent un pan « cryptographie », et un pan « cryptanalyse », formulés par les deux propositions qui suivent.

Problème 1.10. *Étant donnée une instance G d'une famille de groupes $\mathcal{G}$ qu'on souhaite utiliser pour des applications cryptographiques, proposer des améliorations théoriques et pratiques pour le calcul du cardinal de G, et pour les opérations de groupe dans G.*

Problème 1.11. *Étant donnée une instance G d'une famille de groupes $\mathcal{G}$ qu'on souhaite utiliser pour des applications cryptographiques, déterminer (et améliorer) le meilleur algorithme de résolution du problème du logarithme discret dans G, et déterminer sa portée pratique.*

Nous verrons notamment en 7.3 que ce mémoire, même lorsqu'il se concentre sur des groupes issus de courbes algébriques, traite de problèmes parfois distincts en fonction de la définition exacte de la famille de groupes qu'on choisit d'étudier. Nos travaux relatifs au problème du logarithme discret sur les courbes, abordant donc l'aspect « cryptanalyse », apparaissent dans la partie II, tandis que l'aspect « cryptographique », relatif à l'objectif fixé par le problème 1.10, est abordé notamment au chapitre 15.

1.3 Caractéristiques de l'approche employée

L'approche que nous employons pour étudier les problèmes cités ci-dessus implique la considération d'objets mathématiques, et se dirige très vite vers une mise en pratique. Notre approche aborde les différents « étages » du processus menant du concept mathématique et des fondements des problèmes étudiés, au programme informatique performant.

Sur l'aspect de performance, l'implantation est presque systématiquement présente dans nos travaux. Sa vocation peut se cantonner à la validation des algorithmes, mais la situation la plus commune dans notre travail est celle où l'implantation est l'objet d'un soin significatif, pouvant éventuellement aller jusqu'au niveau de la programmation en assembleur pour les parties critiques. À titre d'exemple, les développements du chapitre 13 de ce mémoire témoignent de la réalité du souci d'implémentation dans nos travaux. Il est très important de noter qu'il n'y a pas de parcours à sens unique de l'algorithme vers l'implantation. La quête d'une implantation efficace nous amène ponctuellement à effectuer un retour vers les algorithmes eux-mêmes, et à en modifier des points essentiels afin de pouvoir obtenir de meilleurs résultats. L'aspect théorique de réflexion sur les algorithmes, et l'aspect pratique de la quête de l'efficacité sont donc en symbiose dans nos travaux.

Partie I

La grande famille du crible algébrique

Chapitre 2

Description du crible algébrique

2.1 Cadre de travail

Le crible algébrique, en anglais Number Field Sieve, et abrégé le plus souvent NFS, a été formulé en tant qu'algorithme par l'aboutissement du travail de nombreuses personnes. L'ouvrage de référence [135], notamment les articles [173, 136, 35] qui y apparaissent, permettent d'obtenir une idée de la problématique telle qu'elle se présentait au début des années 1990, où on peut davantage parler de la mise en place de l'algorithme du crible algébrique plutôt que de sa découverte, tant le chemin entre les prémisses de l'algorithme et la description d'un algorithme général est non trivial.

Le crible algébrique connaît plusieurs précurseurs. En tant qu'algorithme de factorisation, il semble naturel de le placer dans la lignée de son prédécesseur direct en tant qu'algorithme de factorisation le plus compétitif, à savoir le crible quadratique (QS) [175]. Pourtant, ce lien d'héritage est ténu. Si la machinerie générale reste similaire à celle du crible quadratique ou de l'algorithme CFRAC [161], il est faux de croire que NFS est une généralisation du crible quadratique. Il est plus pertinent de rapprocher le crible algébrique de l'idée « originelle » de Pollard consistant à factoriser avec des entiers cubiques [173, 176]. Par ailleurs, l'étude du lien entre le crible algébrique et l'algorithme de Coppersmith pour le calcul de logarithmes discrets dans $\mathbb{F}_{2^n}$, premier algorithme de complexité heuristique $L[1/3]$, est particulièrement enrichissante [187].

Une description du crible algébrique est rendue nécessaire par l'importance de cet algorithme dans nombre de nos travaux, développés notamment au sein des chapitres ultérieurs de cette partie. Notre investissement sur le crible algébrique prend aussi la forme du travail sur l'implantation de `cado-nfs`, que nous mentionnons parmi les différentes implémentations du crible algébrique à la fin du présent chapitre.

2.1.1 Choix des polynômes et structures rencontrées

Le crible algébrique établit des *relations multiplicatives* dans des structures qu'il est possible de relier à $\mathbb{Z}/N\mathbb{Z}$. Les structures en jeu sont, si l'on souhaite se placer dans un cadre très général, deux *corps de nombres*. Toutefois, en phase avec le cas le plus utilisé en pratique, et afin de simplifier les notations, nous choisissons d'exposer NFS dans une version « traditionnelle » avec un « côté rationnel », où apparaissent des relations multiplicatives dans $\mathbb{Z}$, et un « côté algébrique », où apparaissent des relations multiplicatives dans un corps de nombres.

Le moyen utilisé par le crible algébrique pour factoriser un entier N est l'obtention d'une égalité entre deux carrés modulo N. Une telle égalité, pour un module composé N, révèle un facteur de N avec bonne probabilité (précisément avec une probabilité $1 - 2^{1-k}$, où k désigne le nombre de facteurs premiers *distincts* de N – en particulier, cette méthode ne s'applique pas aux puissances de nombres premiers). Pour atteindre cet objectif, on crée deux carrés, dans deux structures algébriques distinctes, toutes deux pourvues d'un morphisme d'anneaux vers $\mathbb{Z}/N\mathbb{Z}$.

Les structures algébriques étudiées sont construites comme suit. Soient deux polynômes irréductibles distincts à coefficients entiers f et g, vérifiant la condition $\mathrm{Res}(f, g) \equiv 0 \bmod N$. Cette

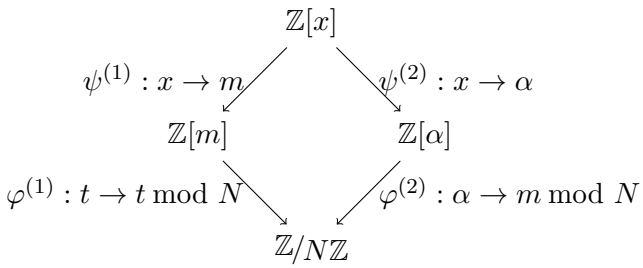


FIG. 2.1 : Schéma de NFS

condition impose que les polynômes f et g disposent d'une racine commune modulo N. Lorsqu'on se place dans le cas classique où g est un polynôme linéaire $g = g_0 + g_1 x$, il est aisé d'écrire que cette racine commune est $m = -\frac{g_0}{g_1} \bmod N$. Notons $\mathbb{Q}(\alpha)$ le corps de nombres défini par le polynôme f, où α vérifie $f(\alpha) = 0$. On note d le degré du polynôme f. Ses coefficients sont notés $f = f_0 + f_1 x + \cdots + f_d x^d$.

La racine de f dans $\mathbb{Q}(\alpha)$ est α. Pour une meilleure facilité d'écriture, on désignera par m le rationnel $-\frac{g_0}{g_1}$, racine de g dans $\mathbb{Q}$. La racine commune de f et g, quant à elle, est notée $m \bmod N$, ou simplement m lorsque le contexte le prescrit, les deux notations étant compatibles quoi qu'il advienne.

On associe aux polynômes f et g les deux anneaux $\mathbb{Z}[\alpha]$ et $\mathbb{Z}[m]$. En toute généralité, on ne suppose pas les polynômes f et g unitaires, donc en particulier α n'est pas un entier algébrique, pas plus que $m = -\frac{g_0}{g_1}$: les anneaux considérés ne sont pas des $\mathbb{Z}$-modules de type fini, mais ceci n'entrave pas le fonctionnement de l'algorithme. Les anneaux $\mathbb{Z}[\alpha]$ et $\mathbb{Z}[m]$ interviennent dans le diagramme 2.1. Ce diagramme commute, en vertu de l'existence de la racine commune m de f et g modulo N. Les flèches du diagramme 2.1 indiquent des morphismes d'anneaux.

2.1.2 Relations

Dans NFS, on considère des fonctions de la forme $\phi = a - bx$, dans le but d'en collecter un nombre suffisant qui satisfont un critère de *friabilité* des deux côtés. On détaille maintenant ce critère. L'image de $\phi = a - bx$ dans $\mathbb{Z}[m]$ est $a - bm = \frac{1}{g_1}(ag_1 + bg_0)$, qui s'écrit aussi de façon plus synthétique comme $\frac{1}{g_1}\operatorname{Res}(\phi, g)$. Nous introduisons la définition suivante.

DÉFINITION 2.2. *Soit $g \in \mathbb{Z}[x]$ de degré 1. On appelle* base de facteurs rationnelle *associée à g et à la borne B l'ensemble :*

$$\mathcal{F}(g, B) \overset{\text{déf}}{=} \{p \in \mathbb{N}^*,\ p \text{ premier}, p \leq B\} \cup \{p \in \mathbb{N}^*,\ p \text{ premier}, p \mid g_1\}.$$

La terminologie « rationnelle » est dans cette définition liée au degré du polynôme g. Le contexte étant généralement clair, on fixe une borne B, et on note la base de facteurs rationnelle $\mathcal{F}(g, B)$ simplement par $\mathcal{F}_r$.

Une telle base de facteurs rationnelle ainsi définie, on définit une fonction ϕ comme friable du côté rationnel lorsque l'entier $\operatorname{Res}(\phi, g)$ se factorise complètement sur la base de facteurs $\mathcal{F}_r$. On peut aussi adopter le point de vue suivant. Le corps de fractions de l'anneau $\mathbb{Z}[m]$ est $\mathbb{Q}$, et son anneau des entiers est $\mathbb{Z}$. La norme de l'image de $\phi = a - bx$ dans $\mathbb{Z}[m]$ est $\frac{1}{g_1}\operatorname{Res}(\phi, g)$, mais la grandeur intéressante est celle qui a une signification dans $\mathbb{Z}$, à savoir $\operatorname{Res}(\phi, g)$. L'adjonction à $\mathcal{F}_r$

des facteurs premiers du terme de tête g_1 permet de confondre la friabilité de l'entier $\mathrm{Res}(\phi, g)$ et celle du rationnel $\frac{1}{g_1}\mathrm{Res}(\phi, g)$. Cette dernière remarque est la justification de l'inclusion du second terme dans la définition 2.2.

L'extension de ce concept au côté algébrique impose d'introduire la notion de factorisation en idéaux. En effet, outre le fait que α n'est pas un entier algébrique, il se trouve que l'anneau des entiers de $\mathbb{Q}(\alpha)$ n'est pas, en général, un anneau factoriel. Il convient donc de s'intéresser à la factorisation en idéaux premiers. Soit $K = \mathbb{Q}(\alpha)$, et $\mathcal{O}_K$ son anneau des entiers. On considère non pas l'élément $a - b\alpha$ image de ϕ dans $\mathbb{Z}[\alpha]$, mais plutôt l'idéal principal $I = (a - b\alpha)$. Comme α n'est pas un entier algébrique, cet idéal est en général un idéal fractionnaire. L'idéal de $\mathcal{O}_K$ défini par

$$J = \langle 1, \alpha \rangle^{-1} = (f_d, f_d\alpha + f_{d-1}, \ldots, f_d\alpha^{d-1} + \cdots + f_1)$$

a pour norme $|f_d|$, et le dénominateur de la factorisation de I divise J. En particulier, on a $JI \subset \mathcal{O}_K$. (Ce dernier point découle de la définition de l'inverse $\langle 1, \alpha \rangle^{-1}$, puisque $I \subset \mathbb{Z} + \alpha\mathbb{Z}$, et $J(\mathbb{Z} + \alpha\mathbb{Z}) \subset \mathcal{O}_K$.) Cet idéal est utile à la définition suivante, qui étend la définition 2.2 au cas d'un polynôme de définition non linéaire.

Définition 2.3. *Soit $f \in \mathbb{Z}[x]$ de degré d, noté $f = f_d\alpha^d + \cdots + f_0$. Soit $K = \mathbb{Q}(\alpha)$ le corps de nombres défini par f, et $\mathcal{O}_K$ son anneau des entiers. On appelle* base de facteurs algébrique *associée à f et à la borne B l'ensemble :*

$$\begin{array}{rcl} \mathcal{F}_a(f, B) & \stackrel{\text{déf}}{=} & \{\mathfrak{p} \subset \mathcal{O}_K,\ \mathfrak{p} \textit{ idéal premier},\ [\mathcal{O}_K/\mathfrak{p} : \mathbb{Z}/\mathbb{Z} \cap \mathfrak{p}] = 1,\ \mathrm{Norm}(\mathfrak{p}) \leq B\} \\ & \cup & \{\mathfrak{p} \subset \mathcal{O}_K,\ \mathfrak{p} \textit{ idéal premier},\ \mathfrak{p} \mid \langle 1, \alpha \rangle^{-1}\}. \end{array}$$

Comme pour le cas rationnel, le contexte est généralement univoque. La borne B utilisée pour la base de facteurs rationnelle est réutilisée pour la base de facteurs algébrique, et on note cette dernière $\mathcal{F}_a = \mathcal{F}(f, B)$. On note qu'il est possible d'employer deux bornes différentes pour les bases de facteurs rationnelle et algébrique. Pour la simplicité de l'exposition, nous choisissons de nous restreindre à une borne unique.

Puisque nous avons construit l'idéal J de sorte à assurer $JI \subset \mathcal{O}_K$, nous considérons alors la fonction ϕ comme friable du côté algébrique lorsque l'idéal JI se factorise entièrement sur la base de facteurs $\mathcal{F}_a$. Comme pour le cas rationnel, l'adjonction à $\mathcal{F}_a$ des facteurs de l'idéal J permet à cette notion d'être confondue avec la friabilité de l'idéal fractionnaire I. C'est donc la même raison technique qui nous a amené à écrire la définition 2.3 sous forme d'une union de deux termes.

Idéaux intervenant dans la factorisation Les idéaux premiers susceptibles d'intervenir dans la factorisation de l'idéal JI sont tout d'abord contraints par la norme. On a :

$$\mathfrak{p} \mid JI \Rightarrow \mathrm{Norm}(\mathfrak{p}) \mid \mathrm{Norm}(JI) = f_d \mathrm{Norm}(a - b\alpha).$$

En outre, $f_d \mathrm{Norm}(a - b\alpha)$ est l'entier

$$\mathrm{Res}(\phi, f) = f_0 b^d + \cdots + f_d a^d.$$

D'autre part, parmi les idéaux premiers qui satisfont cette condition, on ne peut obtenir n'importe quel idéal. En effet, pour un idéal premier $\mathfrak{p}$ divisant JI, on distingue deux cas. Soit $\mathfrak{p} \mid J$, ce qui contraint à un nombre fini d'idéaux. Ces facteurs sont couramment appelés facteurs projectifs (seconde partie, additionnelle, de la définition 2.3). Dans les autres cas, l'idéal $\mathfrak{p}$ peut être décrit de manière synthétique sous la forme $\mathfrak{p} = \langle p, \alpha - r \rangle$ pour un entier r satisfaisant $f(r) \equiv 0 \bmod p$. Cette forme est particulièrement utile pour les calculs. Ceci rejoint par ailleurs la contrainte imposée dans la définition 2.3 aux idéaux ne divisant pas J. En effet, dans la définition 2.3, l'indice $[\mathcal{O}_K/\mathfrak{p} : \mathbb{Z}/\mathbb{Z} \cap \mathfrak{p}]$ est le degré d'inertie de l'idéal premier $\mathfrak{p}$.

Terminologie Il est usuel dans le contexte du crible algébrique de désigner par le terme « norme » les entiers $\mathrm{Res}(\phi, f)$ et $\mathrm{Res}(\phi, g)$ rencontrés ci-dessus, quand bien même ceux-ci ne correspondent qu'à des multiples des normes de $\phi(\alpha)$ et $\phi(m)$, respectivement. Toutefois, le plus souvent la caractéristique recherchée est la friabilité. Au regard de cette caractéristique, il est équivalent de considérer $\mathrm{Res}(\phi, f)$ ou $\mathrm{Norm}_{K/\mathbb{Q}}\,\phi(\alpha)$. D'autre part, pour les considérations relatives à la *taille* de ces normes, ce sont bien les valeurs absolues des entiers $\mathrm{Res}(\phi, f)$ qui sont pertinentes, puisque ce sont les grandeurs qui interviennent dans les calculs.

L'anneau des entiers $\mathcal{O}_K$ D'une manière générale, de nombreux énoncés dans la présentation faite ici du crible algébrique font intervenir $\mathcal{O}_K$. Par exemple, lorsque pour un développement algorithmique on souhaite exploiter une relation de la forme

$$S(\alpha)\mathcal{O}_K = \prod_{\mathfrak{p}\in\mathcal{F}_a} \mathfrak{p}^{\lambda\cdot e_\mathfrak{p}},$$

il semble être nécessaire d'avoir calculé au préalable l'anneau des entiers $\mathcal{O}_K$. Ceci est hélas une tâche difficile (voir par exemple [46, chap. 6]), puisqu'elle requiert la détermination des facteurs carrés du discriminant du polynôme f. Généralement, et à tout le moins dans le contexte de la factorisation d'entiers, ceci requiert la factorisation de l'entier $\mathrm{disc}(f)$. Pour le cas de NFS, ce discriminant peut se trouver être significativement plus grand que l'entier N lui-même.

Toutefois, cette difficulté peut être évitée. Pour cela, on se contente du calcul « par approximations successives » d'un sous-ordre de $\mathcal{O}_K$. Partant d'un sous-ordre quelconque de $\mathcal{O}_K$, on construit, pour chaque nombre premier p rencontré dans les factorisations (en particulier, pour chaque nombre premier inférieur à B), un sur-ordre localement maximal en p, via l'algorithme Round-2 [46, chap. 6]. On construit ainsi un ordre $\mathcal{O}$ dont le conducteur est premier à tous les nombres premiers recontrés. Les factorisations en idéaux ainsi obtenues, ne faisant intervenir aucun idéal dont la norme divise le conducteur de $\mathcal{O}$, sont également valides dans $\mathcal{O}_K$: la capacité à écrire des factorisations en idéaux valides dans $\mathcal{O}_K$ ne nécessite pas le *calcul* de $\mathcal{O}_K$. Il est donc correct de s'intéresser à la factorisation des idéaux considérés en tant qu'idéaux de $\mathcal{O}$.

Pour la simplicité de l'exposition, nous supposons dans l'ensemble de la présentation que nous travaillons avec l'ordre maximal, quand bien même cette aisance de formulation cache une certaine technicité en pratique. Il convient d'avoir cette subtilité à l'esprit lorsqu'on aborde les problèmes de racine carrée dans NFS, comme nous le faisons par exemple au chapitre 4, ou encore sur un thème voisin au chapitre 5.

2.1.3 Égalités entre carrés et obstructions

Le crible algébrique vise à identifier de nombreuses expressions $\phi = a - bx$ telles qu'à la fois $\mathrm{Res}(\phi, g)$ et $\mathrm{Res}(\phi, f)$ sont friables. On manipule donc une collection de *paires* (a_i, b_i), chaque paire déterminant une relation, qu'on peut par exemple écrire :

$$|a_i - b_i m| = p_1^{e_1}\cdots p_k^{e_k},\ \text{où } \{p_i\} = \mathcal{F}_r,$$
$$(a_i - b_i\alpha)\mathcal{O}_K = \mathfrak{p}_1^{e'_1}\cdots \mathfrak{p}_k^{e'_k},\ \text{où } \{\mathfrak{p}_i\} = \mathcal{F}_a.$$

Ces deux expressions sont liées par le fait qu'à la fois $a - bm$ et $a - b\alpha$ ont pour image $(a - bm) \bmod N$ dans le diagramme 2.1. On note que notre choix d'inclure les facteurs de g_1 et de J dans les définitions 2.2 et 2.3 des bases de facteurs nous permet d'obtenir l'écriture ci-dessus. Les idéaux ou nombres premiers correspondant à ces facteurs sont potentiellement affectés de valuations négatives.

Pour obtenir une égalité entre carrés modulo N, l'objectif poursuivi est de forcer toutes les valuations e_j et e'_j dans les relations ci-dessus à être *paires*. Ceci est obtenu par la résolution d'un système linéaire défini sur $\mathbb{F}_2$. Supposons ce premier objectif atteint. On note $T(x) = \prod(a_i - b_i x)$ le produit des relations ainsi constituées. On obtient ainsi que le rationnel $T(m)$ peut s'écrire sous la forme

$$T(m) = \pm\square. \tag{2.4}$$

Si l'on peut obtenir $T(m) = R^2$, pour un nombre rationnel calculable R, alors un premier carré apparaît en exploitant les morphismes décrits par le diagramme 2.1 : on a $T(m) \equiv (R \bmod N)^2 \bmod N$. Il apparaît que pour atteindre ce but, il est nécessaire d'ajouter des contraintes pour lever la difficulté relative au signe $\pm$ dans l'équation 2.4, difficulté que nous mettons de côté un instant.

L'identité obtenue côté algébrique est plus difficile à exploiter. En effet, on obtient que l'idéal $T(\alpha)\mathcal{O}_K$ est un carré dans le groupe des idéaux fractionnaires de $\mathcal{O}_K$. Ceci est insuffisant pour garantir que $T(\alpha)$ est un carré dans $\mathbb{Z}[\alpha]$, pour de multiples raisons, évoquées notamment dans [35] :

- $T(\alpha)\mathcal{O}_K$, idéal principal, est le carré d'un idéal que nous choisissions de noter I, mais cet idéal I n'a pas de raison d'être lui-même principal. L'idéal I est un élément de 2-torsion du groupe des classes $\mathrm{Cl}(\mathcal{O}_K)$. La structure de $\mathrm{Cl}(\mathcal{O}_K)$ détermine donc cette obstruction.

- Même s'il existe $\zeta \in K$ tel que $T(\alpha)\mathcal{O}_K = (\zeta\mathcal{O}_K)^2$, ceci ne garantit tout au plus qu'une égalité de la forme $T(\alpha) = \mu\zeta^2$, où μ est une unité, qui n'est pas nécessairement le *carré* d'une unité. Ainsi, une seconde obstruction est induite par le quotient $\mathcal{O}_K^*/(\mathcal{O}_K^*)^2$.

- Même si $T(\alpha) = \zeta^2$, il n'est pas certain que $\zeta \in \mathbb{Z}[1, \alpha, \ldots, \alpha^{d-1}]$. Cette contrainte est plus aisément interprétée dans le cas $f_d = 1$ lorsque α est un entier algébrique. L'entier algébrique $T(\alpha) = \zeta^2$ appartient au sous-ordre $\mathbb{Z}[\alpha]$ de $\mathcal{O}_K$. Il n'est pas certain que ζ appartient au même sous-ordre. Cette difficulté est la plus simple à contourner, puisqu'en vertu de la propriété $f'(\alpha)\mathcal{O}_K \subset \mathbb{Z}[\alpha]$, il est suffisant de considérer $f'(\alpha)^2 T(\alpha)^2 = (f'(\alpha)\zeta)^2$.

Aux difficultés que nous venons d'énoncer s'en ajoute une autre. En pratique, les relations considérées sont rarement complètes. Certains idéaux ne sont pas considérés, car l'outillage algorithmique pour les manipuler est (modérément) plus important. Même dans cette situation, l'ensemble des difficultés peuvent être résolues aisément par l'emploi de caractères.

2.1.4 Caractères

Notons comme précédemment $T(x) = \prod(a_i - b_i x)$. On a trivialement $\mathrm{sgn}(T(m)) = \prod \mathrm{sgn}(a_i - b_i m)$. Ainsi, le signe est un caractère multiplicatif. Pour le contraindre, il est possible de considérer plusieurs produits $T_1, \ldots T_k$, données par k solutions indépendantes du système linéaire considéré, et d'en déduire des combinaisons (en moyenne $k-1$ indépendantes) possédant la propriété $\mathrm{sgn}(T(m)) = +1$.

La parité des valuations en les nombres premiers, et les idéaux, de la base de facteurs, sont aussi des caractères utilisables de la même façon. Toutefois, leur utilisation est de peu d'intérêt, car leur valeur est nulle pour des produits $T(\alpha)$ possédant *par construction* la propriété d'avoir des valuations paires.

Nous souhaitons utiliser d'autres caractères, permettant un meilleur discernement entre les possibles carrés. Prenons le temps de détailler les structures qui sont en jeu ici, en reprenant la description faite dans [35].

Considérons le sous-groupe multiplicatif V de K^* constitué des nombres algébriques ξ tels que l'idéal $\xi\mathcal{O}_K$ se factorise en un produits d'idéaux dont les seuls apparaissant avec une valuation possiblement impaire sont les idéaux de $\mathcal{F}_a$. Les valeurs $T(\alpha)$ sont par construction des éléments de V, et naturellement tout carré d'un élément de K^* appartient à V. Le quotient $V/(K^*)^2$ est un $\mathbb{F}_2$-espace vectoriel. Le sous-groupe W de V correspondant aux nombres algébriques pour lesquels $\nu_{\mathfrak{p}}(\xi\mathcal{O}_K) \equiv 0 \bmod 2$ pour tous les idéaux de la base de facteurs $\mathcal{F}_a$ contient en particulier les produits $T(\alpha)$. Le quotient $W/(K^*)^2$ est un sous-espace vectoriel de $V/(K^*)^2$. La quantification de sa dimension est précisément le travail de quantification des obstructions évoquées en 2.1.3. C'est un espace vectoriel de *petite* dimension. Cette dimension est essentiellement donnée par la 2-torsion du groupe de classes $\mathrm{Cl}(\mathcal{O}_K)$, et par le rang du groupe des unités $\mathcal{O}_K^*$ [35].

Par caractère, on entend une application linéaire de $W/(K^*)^2$ vers $\mathbb{F}_2$, c'est-à-dire un élément du dual $(W/(K^*)^2)^*$. Comme vu plus haut, il est évident que sur cette structure, le caractère « parité de la valuation » pour $\mathfrak{p} \in \mathcal{F}_a$ est trivial et bien sûr inintéressant. Une famille de caractères qu'il s'avère pertinent d'examiner a été proposée par Adleman [2], et est donnée ainsi : soit $q > B$ un nombre premier modulo lequel f possède une racine simple r. Notons $s(\alpha)$ un élément arbitraire de K^*. On considère $\chi_{q,r}(s(\alpha)) = \left(\frac{s(r)}{q}\right)$. Ces caractères sont bien définis (puisque $q > B$), et l'hypothèse heuristique sous-tendant leur utilisation est qu'en choisissant quelques-uns de ces caractères, la totalité du dual $(W/(K^*)^2)^*$ est couverte. Ainsi, un élément de K^* tel que les caractères ainsi considérés sont tous nuls est de façon presque certaine un carré : les obstructions évoquées plus haut ont été levées.

Différentes généralisations des caractères que nous venons de présenter sont possibles. Par exemple, pour détecter les puissances e-èmes, il est possible de s'intéresser aux résidus modulo des nombres premiers vérifiant $q \equiv 1 \bmod e$. Cette généralisation s'avèrera utile au chapitre 5, particulièrement en 5.3 et 5.4. Il est aussi possible de détecter des puissances plus grandes, en employant des caractères dits ℓ-adiques, définis par Schirokauer [181] et brièvement décrits en 3.1.3.

2.1.5 Calcul de racine carrée

L'identification des obstructions, et la considération des caractères, permet d'aboutir à une combinaison $T(x)$ telle qu'il existe à la fois un polynôme $R(x)$ tel que $T(m) = R(m)^2$, et aussi un autre polynôme tel que $T(\alpha) = A(\alpha)^2$. Ceci permet donc d'aboutir à $R(m)^2 \equiv A(m)^2 \bmod N$. Encore faut-il être en mesure de *calculer* les polynômes $R(m)$ et $A(m)$.

Le problème du calcul de $R(m)$ est vite énoncé. Cela revient à calculer la racine carrée d'un entier dans $\mathbb{Z}$. L'une de façons de traiter le problème peut donc être de recourir à des algorithmes asymptotiquement rapides [33].

Calculer $A(x)$ est plus ardu. Nous revenons à ce point au chapitre 4.

2.2 Analyse de NFS

2.2.1 Préambule à l'analyse : arithmétique avec la fonction L

L'analyse de NFS utilise la fonction suivante notée L, introduite par Schroeppel et Pomerance. Cette fonction est couramment appelée *fonction sous-exponentielle*, puisqu'elle interpole entre une croissance polynomiale et une croissance exponentielle. On note :

$$L_x[a, u] = \exp\left(u(\log x)^a(\log\log x)^{1-a}\right).$$

Les manipulations autour de la fonction L font appel à quelques règles de calcul que nous détaillons brièvement.

Proposition 2.5. *On a :*

$$L_x[a,u] \times L_x[b,v] = \begin{cases} L_x[a, u+o(1)] & \text{si } a > b, \\ L_x[b, v+o(1)] & \text{si } b > a, \\ L_x[a, u+v] & \text{si } a = b. \end{cases}$$

$$\text{(pour } u,v>0\text{)}\ L_x[a,u] + L_x[b,v] = \begin{cases} L_x[a, u+o(1)] & \text{si } a > b, \\ L_x[b, v+o(1)] & \text{si } b > a, \\ O(L_x[a, \max(u,v)]) & \text{si } a = b. \end{cases}$$

$$L_{L_x[b,v]}[a,u] = L_x[ab, uv^a b^{1-a} + o(1)].$$

$$L_x[b,v]^{\log_{\log x} L_x[a,u]} = L_x[ab, u+v].$$

La raison essentielle qui motive l'emploi de la fonction L est l'expression agréable du théorème de Canfield-Erdős-Pomerance :

Proposition 2.6. *La proportion d'entiers $< L_x[a,u]$ qui sont $L_x[b,v]$-friables est*

$$L_x[a-b, -\frac{u}{v}(a-b) + o(1)].$$

2.2.2 Importance de la sélection polynomiale

L'analyse de NFS est difficilement dissociable de la sélection polynomiale. En effet, la façon de choisir les polynômes définissant les structures algébriques du diagramme 2.1 influe de manière importante sur la complexité. Nous discutons les deux cas essentiels.

GNFS, base-m En toute généralité, et en particulier lorsque N ne dispose pas d'une écriture algébrique compacte, on parle de l'algorithme *General Number Field Sieve* (GNFS). Il est possible de trouver un couple de polynômes f, g par la méthode élémentaire appelée « base-m » [35]. Cette méthode procède comme suit :

- Choisir un degré d (l'optimum asymptotique est donné par l'analyse) ;
- Calculer un entier $m \approx N^{\frac{1}{d+1}}$;
- Écrire le développement $N = f_d m^d + \cdots + f_0$ en base m ;
- Prendre $g = x - m$, et $f = f_d x^d + \cdots + f_0$.

Cette méthode conduit à $\mathrm{Res}(f,g) = N$.

Il existe de nombreuses améliorations de cette approche, que nous ne détaillons pas dans le cadre de ce mémoire. En particulier, l'algorithme de Kleinjung [124], et sa variante plus récente [126] permettent d'obtenir des polynômes bien meilleurs qu'avec cette aproche naïve. Il est important, pour obtenir de bons polynômes pour GNFS, de définir des critères de qualité. Ces critères incluent classiquement des considérations de *taille*, où l'on apprécie un polynôme f prenant de petites valeurs $f(a,b)$ sur la zone à considérer, et d'autre part les considérations dites de *propriétés des racines* qui favorisent les polynômes possédant de nombreuses racines modulo les petits nombres premiers. Ce dernier critère, conjointement avec les algorithmes proposés par Kleinjung, peut être grandement amélioré grâce à la technique dite du *root sieve* [162, 13].

SNFS Un cas particulier est celui où la forme de l'entier N donne lieu à un choix « naturel » pour f et g. C'est le cas par exemple pour $2F_7 = 2^{129} + 2$: le choix $f = x^3 + 2$, $g = x - 2^{43}$ n'est pas absurde. Plus généralement, un nombre N apparaissant comme expression algébrique, ou comme cofacteur d'une telle expression, donne lieu à un choix « naturel » de polynômes, ce qui ne préjuge en rien de l'intérêt d'un tel choix : il se peut que les polynômes ainsi obtenus, bien que dérivés directement de l'écriture de N, ne soient pas optimaux. Lorsque NFS est utilisé avec des polynômes ainsi dérivés de l'écriture du nombre à factoriser N, on parle de *Special Number Field Sieve* (SNFS). Pour des nombres donnant lieu à des très bons polynômes selon cette procédure, SNFS est une méthode efficace. Le choix du « meilleur » polynôme à utiliser en fonction de l'écriture de N dans un tel cas est discuté dans [199].

2.2.3 Analyse dans le cas GNFS

Nous faisons l'analyse complète pour le cas GNFS, puis indiquons dans un second temps les modifications qui s'appliquent au cas SNFS. L'analyse est menée pour N tendant vers $+\infty$, et se limite au premier ordre. Nombre de notations sont introduites dans les paragraphes qui suivent, et leur portée est limitée à l'analyse.

Notons d'abord les liens entre les différents paramètres de l'algorithme et le temps de calcul. Il est légitime pour l'analyse de considérer le temps qui est nécessaire pour traiter un ensemble de paires (a, b), pouvant être décrit de manière simple (par exemple un parallélotope), et produisant exactement assez de relations. Si l'ensemble des paires (a, b) considérés est trop restreint, alors le nombre de relations obtenues est insuffisant. Si cet espace est trop grand, alors deux phénomènes sont possibles. Il se peut que le nombre de relations produites dépasse le nombre nécessaire, ou éventuellement que la croissance des bornes sur (a, b) entraîne une croissance des normes telle que trop peu de relations sont obtenues *in fine*. Ceci justifie le fait de poser comme hypothèse de l'analyse le fait que l'espace des paires (a, b) à considérer est dimensionné correctement. Le temps nécessaire par paire (a, b) examinée, ainsi que le temps nécessaire pour construire la relation à partir d'une paire (a, b) satisfaisant la condition de double friabilité requise, n'apparaissent pas dans l'analyse, car ils ne relèvent pas du terme dominant. En particulier, les techniques usuelles de crible, remontant au crible quadratique [175] permettent d'effectuer cette tâche de sorte à éliminer leur impact sur le terme dominant dans (l'exposant de) l'expression de la complexité de l'algorithme. Aussi, nous considérons le temps de traitement de l'espace des paires (a, b) comme équivalant à un temps constant par paire.

La taille des bases de facteurs est aussi un paramètre crucial. La borne de friabilité contrôle à la fois la probabilité de friabilité (une borne plus grande donne un plus grand nombre de relations friables), mais aussi le nombre de relations qui doivent être obtenues afin de rencontrer un système linéaire possédant une solution non triviale. Par là même, les dimensions de ce système linéaire sont conditionnées par la taille des bases de facteurs. La résolution d'un tel système occupe au moins un temps quadratique [217, 133].

L'analyse est menée avec la supposition que la résolution du système linéaire et la recherche de relations occupent un temps comparable.

Nous supposons que les deux bases de facteurs considérées sont déterminées par une même borne $B = L_N[b', \beta]$, et que les fonctions $\phi = a - bx$ considérées satisfont les encadrements $|a| < S$ et $0 < b < S$, où la borne vaut $S = L_N[s, \sigma]$. L'espace des paires (a, b) considérées est donc de cardinal $S^2 = L_N[s, 2\sigma]$, et les bases de facteurs sont de cardinal $L_N[b', \beta + o(1)]$. Le degré, enfin, est choisi sous la forme $d = \log_{\log N} L_N[\Delta, \delta]$. Sous les hypothèses qu'on vient d'évoquer, on a $S^2 \approx B^2$, d'où le choix $b' = s$, et $\sigma = \beta$.

Les quantités $\mathrm{Res}(f,\phi)$ et $\mathrm{Res}(g,\phi)$ sont bornées ainsi :

$$|\mathrm{Res}(f,\phi)| \leq (1+\deg f)\cdot \|f\|_\infty \cdot S^{\deg f},$$
$$|\mathrm{Res}(g,\phi)| \leq (1+\deg g)\cdot \|g\|_\infty \cdot S^{\deg g}.$$

La méthode dite « base-m » donne alors les bornes suivantes :

$$|\mathrm{Res}(f,\phi)| \leq L_N[1-\Delta, 1/\delta]L_N[s+\Delta, \sigma\delta],$$
$$|\mathrm{Res}(g,\phi)| \leq L_N[1-\Delta, 1/\delta]L_N[s, \sigma].$$

Le degré d est choisi de sorte à minimiser ces normes. Aussi, on fixe $1-\Delta = s+\Delta$, au regard de la première des équations ci-dessus. Il convient d'évaluer la probabilité π qu'une paire (a,b) soit doublement friable. Pour ce faire, on recourt à l'heuristique suivante.

Heuristique 2.7. *Pour un ensemble de fonctions $\{\phi\}$ fixé, les valeurs $|\mathrm{Res}(f,\phi)|$ et $|\mathrm{Res}(g,\phi)|$ sont $L_N[b',\beta]$-friables avec une probabilité $\pi = \pi_{random}^{1+o(1)}$, où π_{random} désigne la probabilité de friabilité, calculable par le théorème de Canfield-Erdős-Pomerance 2.6, d'entiers aléatoires satisfaisant les mêmes contraintes de taille que celles auxquelles sont soumises les quantités $|\mathrm{Res}(f,\phi)|$ et $|\mathrm{Res}(g,\phi)|$.*

Notons que l'hypothèse heuristique 2.7 joue un rôle capital dans l'analyse de l'ensemble des algorithmes de complexité $L[1/3]$ apparaissant dans ce mémoire. Cette hypothèse est largement corroborée par les résultats expérimentaux.

Sous l'hypothèse heuristique 2.7, il est possible d'appliquer la proposition 2.6 pour poursuivre l'analyse. On néglige en outre le terme $L_N[s,\sigma]$ dans l'expression de $|\mathrm{Res}(g,\phi)|$, puisqu'on a $L_N[s,\sigma] = o(L_N[s+\Delta,\sigma\delta])$. La probabilité π est alors donnée par

$$\pi = L_N[\Delta, -\Delta\frac{1}{\beta}(\frac{2}{\delta}+\beta\delta)+o(1)],$$

où l'on a appliqué les simplifications $s+\Delta-b' = \Delta$ et $\sigma = \beta$, en vertu de la condition $S \approx B$. Nous avons encore la liberté de fixer la constante δ. Celle-ci est choisie de manière à maximiser la probabilité ci-dessus, ce qui conduit à $\delta = \sqrt{2/\beta}$, et l'expression suivante de la probabilité :

$$\pi = L_N[\Delta, -\Delta\frac{1}{\beta}(2\sqrt{2}\sqrt{\beta})+o(1)],$$

La supposition que l'espace des paires (a,b) est dimensionné de manière appropriée revient à poser l'équation $S^2\pi = L[b',\beta+o(1)]$, qui se récrit $L[b',2\beta]\pi = L[b',\beta+o(1)]$. Ceci conduit à $\Delta = b'$, et $\Delta\frac{1}{\beta}(2\sqrt{2}\sqrt{\beta}) = \beta$.

Ces contraintes permettent d'aboutir à l'équation de la complexité de GNFS. Des trois équations $s = b'$, $1-\Delta = s+\Delta$, et $\Delta = b'$, on déduit tout d'abord $s = b' = \Delta = \frac{1}{3}$, puis ensuite $\beta = (8/9)^{1/3}$. Nous sommes donc en présence d'un algorithme dont la complexité heuristique est de la forme $L_N[1/3, (64/9)^{1/3}+o(1)]$. Plus précisément, les valeurs induites par cette analyse asymptotique sont données par la table 2.8.

2.2.4 Le cas SNFS

Nous avons évoqué informellement SNFS comme étant une version du crible algébrique pour laquelle un couple de polynômes « naturel » se présentait. C'est à la lumière de l'analyse de GNFS

Bornes de friabilité	$B = L_N[1/3, (8/9)^{1/3} + o(1)]$
Bornes sur (a, b)	$S = L_N[1/3, (8/9)^{1/3} + o(1)]$
Degré de f	$\deg f = \log_{\log N} L_N[1/3, 3^{1/3} + o(1)] \sim 3^{1/3} \left(\frac{\log N}{\log\log N}\right)^{1/3}$
Normes algébriques	$\lvert\mathrm{Res}(f, \phi)\rvert \leq 3, 3/\sqrt{2} + o(1)]$
Normes rationnelles	$\lvert\mathrm{Res}(f, \phi)\rvert \leq 3, \sqrt{2} + o(1)]$
Probabilité de friabilité	$\pi = L_n[1/3, -(8/9)^{1/3} + o(1)]$ (†)
Temps de recherche des relations, temps de l'algèbre linéaire, complexité totale	$L_N[1/3, (64/9)^{1/3} + o(1)]$ (†).

(†) : Complexité heuristique

Tab. 2.8 : Quantités induites par l'analyse asymptotique de GNFS.

qu'il est possible de spécifier le mieux l'algorithme SNFS. En effet, on entend par SNFS l'algorithme qui peut être appliqué à une classe de nombres tels qu'il existe un couple de polynômes f, g qui satisfont, pour une certaine constante δ :

$$\begin{aligned} \mathrm{Res}(f, g) &\equiv 0 \bmod N, \\ \|f\|_\infty &= L_N[1/3, o(1)], \\ \deg f &= \log_{\log N} L_N[1/3, \delta]. \end{aligned}$$

Informellement, on s'intéresse aux couples de polynômes ayant peu ou prou le bon degré, et pour lesquels les coefficients de f sont petits. Sous ces hypothèses, on vérifie que les normes rencontrées ont pour expression :

$$\begin{aligned} \lvert\mathrm{Res}(f, \phi)\rvert &\leq L_N[2/3, \sigma\delta], \\ \lvert\mathrm{Res}(g, \phi)\rvert &\leq L_N[2/3, 1/\delta] L_N[s, \sigma]. \end{aligned}$$

Le cas le plus favorable est donc celui pour lequel la constante δ satisfait $\delta = \sqrt{\sigma}$. Dans ce cas, en examinant à nouveau les probabilités de friabilité comme on l'a fait plus haut pour GNFS, on obtient $\beta = (4/9)^{1/3}$, d'où une complexité en $L_N[1/3, (32/9)^{1/3} + o(1)]$, nettement inférieure (cette complexité s'appuie toujours sur l'heuristique 2.7).

2.3 Implémentations et records

La description du crible algébrique que nous venons de donner a laissé de côté les aspects, primordiaux, qui touchent à la mise en pratique : l'art de sélectionner les polynômes, ou l'art de cribler pour obtenir des relations, sont bien plus que de simples considérations d'optimisation. Sur quantité de points précis, les travaux sur NFS depuis deux décennies ont conduit à organiser les calculs de façon efficace, pour faire du crible algébrique un algorithme pratique. Exposer l'ensemble de cet état de l'art est une tentation alléchante, mais conduirait à un alourdissement conséquent de ce mémoire[3].

Plusieurs implémentations de NFS existent, et elles rassemblent, collectivement, les algorithmes les plus adéquats pour les différentes étapes du crible algébrique. Parmi les implantations aujourd'hui visibles et pas trop anciennes, on peut citer les suivantes.

[3]La mise sur pied d'un ouvrage avec un tel objectif est néanmoins en projet dans le cadre de l'ANR Catrel que nous coordonnons (2013-2016).

Date	bits	N	Auteurs
1999	512	RSA-155	The Cabal
2002	524	$2^{953}+1$	Bahr, Franke, Kleinjung
2003	532	RSA-160	Franke et al.
2003	545	$2^{1826}+1$	Aoki et al.
2005	582	$11^{281}+1$	Aoki et al.
2005	640	RSA-640	Franke et al.
2005	663	RSA-200	Franke et al.
2009	768	RSA-768	EPFL + Caramel + CWI + NTT + Bonn

FIG. 2.9 : Records de factorisation depuis RSA-155.

– Le code du CWI. Le CWI (Amsterdam) a été la force d'initiative majeure des travaux de factorisation durant la décennie 1990-2000. Un code complet de crible algébrique a été associé à ce travail. Ce code a cependant peu évolué depuis le milieu des années 2000, et n'est plus actif aujourd'hui. Il n'a jamais été publiquement diffusé, mais est désormais en passe de l'être, à la suite du départ en retraite de H. te Riele. Le code du CWI contient notamment une implantation en Pari/GP de l'algorithme de calcul de racine carrée de Montgomery (que nous décrivons au chapitre 4), ainsi qu'une version de l'algorithme de Lanczos par blocs en parallèle (brièvement évoqué au chapitre 11). Le code du CWI a par ailleurs été le réceptacle originel des travaux de S. Cavallar sur le filtrage de relations pour NFS. En comparaison avec les autres codes existants, le code du CWI est technologiquement inférieur concernant la sélection polynomiale et le crible.

– Le code de Franke et Kleinjung. En 2002, J. Franke et T. Kleinjung ont démarré une implantation du crible algébrique. Ils ont introduit plusieurs nouveaux algorithmes, notamment pour la sélection polynomiale et le crible. Leur code n'est pas diffusé sous une forme précise, bien que des composants précis circulent. Ce code est au niveau de l'état de l'art pour l'ensemble des étapes de NFS.

– Le programme GGNFS [155] est une initiative individuelle, démarrée en 2004 dans l'objectif de fournir une implantation librement accessible du crible algébrique. L'ambition de GGNFS est modeste, mais ce code librement accessible sous licence GNU GPL est l'un des moyens d'obtenir une copie des codes de crible et de sélection polynomiale de Franke et Kleinjung, qui y ont été inclus. GGNFS contient aussi une implémentation de l'algorithme de Montgomery pour la racine carrée. GGNFS n'est aujourd'hui plus maintenu.

– Le programme `msieve` [169] est aussi une initiative individuelle, qui démarra par une implémentation du crible quadratique. `msieve` inclut désormais une implémentation de NFS. L'ensemble du programme `msieve` est distribué sous licence « *public domain* ». L'implantation utilisée pour le crible est reconnue comme étant sous-optimale, et peut être remplacée par le programme de Franke et Kleinjung pour de meilleures performances. `msieve` inclut des travaux récents sur la sélection polynomiale, en exploitant notamment la présence de cartes graphiques pour accélérer le calcul. Parmi les atouts de `msieve`, on remarque aussi l'étape de filtrage, qui obtient de bons résultats. `msieve` inclut une implantation originale de l'algorithme de Lanczos par blocs.

– Le programme `cado-nfs` [90], est le produit d'une initiative débutée en 2007 dans le cadre du projet ANR Cado, coordonné par l'équipe Cacao de l'INRIA Lorraine, dont nous étions membre. `cado-nfs` est une implémentation complètement originale du crible algébrique, sous licence GNU LGPL. En particulier, `cado-nfs` contient la seule implémentation de l'algorithme du *lattice sieving* qui soit concurrentielle avec l'implémentation de Franke et Kleinjung, pour des performances aujourd'hui comparables. `cado-nfs` inclut d'autre part une implémentation de l'algorithme de Wiedemann par blocs (algorithme décrit au chapitre 12). Certains de nos travaux sont aussi disponibles sous forme logicielle au cein de `cado-nfs`. C'est notamment le cas de l'algorithme de calcul de racine carrée que nous décrivons au chapitre 4, ou encore de l'algorithme de *root sieve* décrit dans [13].

La figure 2.9 recense les derniers records de factorisation par GNFS depuis que la barre des 512 bits a été franchie. Les records depuis 2005 ont tous été obtenus en utilisant le code de Franke et Kleinjung. Le record de factorisation RSA-768, pour lequel notre participation a été déterminante, est décrit dans le chapitre 6.

Parallèlement aux records les plus visibles, les autres codes mentionnés plus haut, notamment `msieve` et `cado-nfs`, ont aussi été utilisés pour mesurer leurs limites. Ainsi, en 2012, RSA-704, nombre de 212 chiffres décimaux, a été factorisé en utilisant `cado-nfs` exclusivement, et le numérateur du nombre de Bernoulli B200, dont un cofacteur de 204 chiffres restait sans facteur premier connu, a été factorisé en utilisant `msieve`, le programme de Franke et Kleinjung pour le crible, et `cado-nfs` pour l'algèbre linéaire.

Chapitre 3

NFS pour le logarithme discret

3.1 Contexte

Une variante importante de NFS est son avatar pour le problème du logarithme discret dans les corps finis. Cette variante a été proposée initialement par Gordon [95], et étudiée notamment par Schirokauer [187, 181, 182, 185, 184, 186], ainsi que par Joux et Lercier [116], en collaboration ponctuelle avec Smart et Vercauteren [122, 141]. L'état de l'art expérimental sur le sujet a été alimenté au cours de la décennie passée par de nombreux résultats, principalement par Joux et Lercier [110, 111, 112, 113, 114, 117, 141, 118, 119, 120], mais aussi par d'autres travaux [209, 125, 102].

Ce chapitre a une vocation introductive pour certains des travaux présentés dans ce mémoire. En effet, bien qu'aucune de nos réalisations récentes n'ait pris la forme, à l'instar de la factorisation de RSA-768, d'un record de calcul de logarithmes discrets dans le groupe multiplicatif d'un corps fini, la présentation théorique de l'usage de NFS pour le calcul de logarithmes discrets dans les corps finis est essentielle pour nos travaux à plus d'un titre. En premier lieu, ce problème en lui-même apparaît au chapitre 1 parmi nos motivations : notre participation passée à de tels records [209], ou bien notre investissement récent dans le projet ANR Catrel (2013-2016) nous engageant dans cette voie en sont des témoins. D'autre part, et plus précisément en lien avec les travaux mentionnés dans ce mémoire, c'est bel et bien la connaissance de cette méthode de calcul de logarithmes discrets qui a rendu possibles les travaux décrits au chapitre 5, ou encore le travail décrit au chapitre 10.

3.1.1 Variantes de l'algorithme

Le cadre général du crible algébrique pour le logarithme discret permet aussi d'englober l'algorithme de Coppersmith [50], ou des variantes de celui-ci [193], ainsi que l'algorithme FFS du crible du corps de fonctions [3, 6, 115] (voir section 3.4), ou encore la variante NFS-HD [122]. Le crible algébrique pour le logarithme discret recouvre donc un assez vaste panel d'algorithmes. Ceux-ci se différencient par le corps fini auquel ils sont adaptés. Pour $q = p^n$, les éléments du corps $\mathbb{F}_q = \mathbb{F}_{p^n}$ sont de taille $n \log_2 p$ bits. La part relative de n et $\log p$ dans ces $n \log_2 p$ bits conditionne l'algorithme de référence à utiliser pour résoudre le problème du logarithme discret. Plusieurs lignes de séparation existent. Pour présenter les choses simplement, on peut avoir à l'esprit une discrimination en trois zones correspondant aux cas « p petit » pour $\log n \geq 2 \log\log p$, « p grand » pour $\log\log p \geq 2 \log n$, et enfin au cas « p moyen » entre ces deux frontières. Plus exactement, les conditions définissant les zones sont en fait $p = L_q[\ell, \cdot]$ avec $\ell < 1/3$ pour p petit, et $p = L_q[\ell, \cdot]$ avec $\ell > 2/3$ pour p grand. Dans ces différents cas, les algorithmes considérés sont les suivants.

- Pour p petit, l'algorithme FFS peut être utilisé [3, 6, 115]. Dans le cas particulier où $p = 2$, alors l'algorithme de Coppersmith [50] s'applique, et est efficace pour certaines valeurs de n. Ces algorithmes ont une complexité heuristique qui s'écrit sous la forme $L_q[1/3, (32/9)^{1/3}]$.

- Pour p grand, un algorithme très proche du crible algébrique s'applique. Ceci est vrai notamment dans le cas $n = 1$ (corps premiers) [95, 116], mais aussi dans le cas où n est modérément

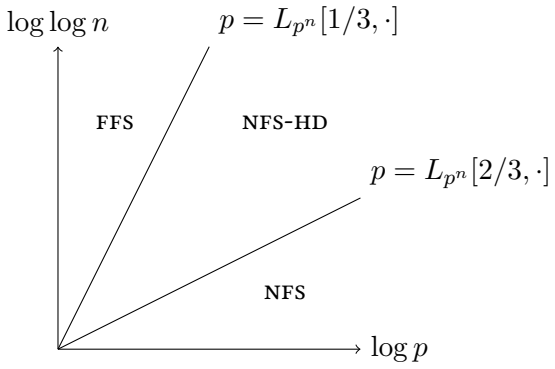


FIG. 3.1 : Domaines de validité pour différents algorithmes de logarithme discret sur les corps finis.

plus grand, sujet à la borne requise [185, 122]. La complexité heuristique dans cette zone est $L_q[1/3, (64/9)^{1/3}]$, sauf à considérer des nombres premiers « spéciaux » (par exemple dans le sens de l'algorithme SNFS), auquel cas une complexité semblable à celle de SNFS peut être obtenue [194].

– Le cas « médian » a longtemps paru être plus difficile, puisque jusqu'en 2006 seul un algorithme de complexité $L_q[1/2, \cdot]$ permettait de calculer des logarithmes discrets en temps sous-exponentiel dans des corps finis évoluant dans cette zone [4]. Cet algorithme est toutefois en deçà, en termes de performances, de l'adaptation du crible algébrique [122], dont la complexité heuristique est en $L_q[1/3, \cdot]$, la constante atteignant dans le cas le plus difficile la borne de $(128/9)^{1/3}$.

L'ensemble des variantes de NFS et FFS pour calculer des logarithmes discrets partagent un socle commun (pour une bonne part similaire à NFS pour la factorisation). Le choix des polynômes de définition, notamment, conditionne de manière importante la complexité. Pour le cas particulier du calcul de logarithmes discrets, des spécificités essentielles se dégagent, à savoir le nécessaire ajout de caractères idoines [181], la notion de *logarithmes virtuels* (3.3), ainsi qu'une procédure de *descente*, destinée au calcul de logarithmes arbitraires à partir du précalcul de logarithmes correspondant aux éléments de la base de facteurs. Nous décrivons dans ce chapitre les modifications qui sont apportées au schéma général du crible algébrique dans l'objectif de traiter le calcul de logarithmes discrets. L'exposition privilégie à titre d'exemple le cas des corps premiers, et nous donnons un bref aperçu du cas FFS. Cette restriction de vue est suffisante pour notre propos, qui vise particulièrement à mettre en évidence les spécificités du calcul de logarithmes discrets par ces méthodes.

3.1.2 Restriction du problème

Nous fixons un corps fini $\mathbb{F}_q$, où pour le confort de l'exposition nous supposons initalement que $q = p^n$ est un nombre premier (ainsi, $n = 1$). Notre objectif est le calcul de logarithmes discrets non pas vraiment dans $\mathbb{F}_q^*$, mais plus précisément dans le plus grand sous-groupe d'ordre premier. En effet, en vertu de la réduction de Pohlig-Hellman, la vraie difficulté réside dans ce sous-groupe, l'information dans les sous-groupes de taille petite pouvant être obtenue à peu de frais. Nous notons ℓ le cardinal de ce sous-groupe. Pour le contexte des applications en cryptanalyse, ℓ est donc un grand nombre premier. On fait la supposition que ℓ divise exactement $q - 1$. En introduisant la notation

usuelle ν_ℓ pour la ℓ-valuation d'un entier, cette supposition se reformule $\nu_\ell(q-1) = 1$. D'une manière générale, pour le contexte qui prévaut dans notre perspective, toute supposition du type « ℓ est premier avec… » est valide sauf infinie malchance. Le groupe $\mathbb{F}_q^*$ étant cyclique, le sous-groupe dans lequel nous souhaitons calculer des logarithmes discrets est $G_\ell = \mathbb{F}_q^*/\mathbb{F}_q^{*\ell}$, qui s'identifie à $\mathbb{F}_q^*[\ell]$ puisqu'on fait l'hypothèse $\nu_\ell(q-1) = 1$. Pour référence, les hypothèses que nous faisons sont résumées ainsi.

Conditions 3.2. *Pour le calcul de logarithmes discrets par NFS dans un sous-groupe de cardinal ℓ de $\mathbb{F}_q$, on suppose que :*

- *ℓ est un nombre premier, et $\nu_\ell(q-1) = 1$.*
- *Le corps de nombres K utilisé pour le côté algébrique dans NFS est tel que ℓ est premier au nombre de classes h de l'ordre maximal.*
- *Le noyau de la restriction des caractères ℓ-adiques définis en 3.1.3 aux unités $\mathcal{O}_K^*$ est précisément le groupe des puissances ℓ-èmes des unités.*

3.1.3 Définition des caractères ℓ-adiques

Les caractères définis en 2.1.4, ou en tout cas leur généralisation à des exposants différents de 2, ne conviennent pas à notre cas d'étude, où ℓ est un grand nombre premier, puisque leur calcul revient à un calcul de logarithme discret dans un groupe de cardinal ℓ d'un corps fini, qui est précisément notre objectif de départ. Pour être en mesure de déterminer si un entier algébrique d'un corps de nombres K est une puissance ℓ-ème, pour un nombre premier ℓ donné, Schirokauer propose d'étudier l'anneau $A_\ell = \mathcal{O}_K/\ell\mathcal{O}_K$. Si le nombre premier ℓ est non ramifié dans $\mathcal{O}_K$, et se factorise sous la forme $\ell\mathcal{O}_K = \mathfrak{l}_1 \cdots \mathfrak{l}_k$, nous notons m le p.p.c.m. des degrés d'inertie $[\mathfrak{l}_i : \ell]$. Alors un entier algébrique x n'appartenant *pas* à l'idéal $\ell\mathcal{O}_K$ vérifie $x^{\ell^m-1} \in 1 + \ell\mathcal{O}_K$. Nous introduisons l'application :

$$\Lambda_\ell : \left\{ \begin{array}{ccccc} \mathcal{O}_K - \ell\mathcal{O}_K & \longrightarrow & \ell\mathcal{O}_K/\ell^2\mathcal{O}_K & \longrightarrow & \mathbb{F}_\ell^{[K:\mathbb{Q}]}, \\ x & \longmapsto & x^{\ell^m-1} - 1 & \longmapsto & \frac{1}{\ell}\left(x^{\ell^m-1} - 1\right), \end{array} \right.$$

où la correspondance avec $\mathbb{F}_\ell^{[K:\mathbb{Q}]}$ est simplement donnée par l'écriture sur une base d'entiers idoine. Il est aisé de constater que Λ_ℓ est un morphisme du groupe multiplicatif $\mathcal{O}_K - \ell\mathcal{O}_K$ dans le $\mathbb{F}_\ell$-espace vectoriel $\mathbb{F}_\ell^{[K:\mathbb{Q}]}$, et d'autre part on remarque que les puissances ℓ-èmes appartiennent à son noyau. Circonscrire exactement ce noyau, en revanche, est potentiellement plus difficile. Schirokauer [181] montre que sous des conditions raisonnables, l'hypothèse évoquée plus haut limitant ce noyau aux puissances ℓ-èmes est vérifiée.

3.2 Adaptation de NFS pour le logarithme discret

Il est aisé de formuler un algorithme « simple », mais hélas peu efficace, pour calcul des logarithmes discrets dans $\mathbb{F}_q$ en temps $L_q[1/2, \cdot]$. Cet algorithme est l'algorithme d'Adleman [1]. Il repose sur l'idée importante de la combinaison de congruences. L'algorithme d'Adleman ne permet pas, toutefois, de se départir du cas où les grandeurs qu'on souhaite voir friables sont exponentielles en $\log q$. Dès lors, la complexité ne descend pas en dessous de $L_q[1/2, \cdot]$. Le crible algébrique offre une machinerie permettant de fabriquer des objets de taille sous-exponentielle, dont on espère la friabilité.

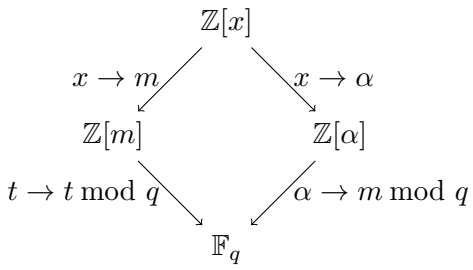


FIG. 3.3 : Schéma de NFS pour le logarithme discret

Nous supposons, comme dans le cas classique de NFS, que deux polynômes $f(x) = f_d x^d + \cdots + f_0$ et $g(x) = g_1 x + g_0$ ont été choisis[1], et possèdent une racine commune m dans $\mathbb{F}_q$. Une base de facteurs rationnelle $\mathcal{F}_r$ et une base de facteurs algébrique $\mathcal{F}_a$ sont définies selon les définitions 2.2 et 2.3, relativement à une borne B choisie en fonction de q. La figure 3.3 indique les structures mathématiques qui interviennent dans cette construction.

NFS s'appuie sur la découverte de nombreuses fonctions $\phi = a - bx$ pour lesquelles les entiers $\mathrm{Res}(\phi, f)$ et $\mathrm{Res}(\phi, g)$ sont friables, ce qui permet de décomposer $\phi(m)$ et $\phi(\alpha)\mathcal{O}_K$ sur les bases de facteurs $\mathcal{F}_r$ et $\mathcal{F}_a$. Ce point fondamental est inchangé pour le contexte du logarithme discret. Nous faisons donc la supposition qu'à l'issue d'une phase de *collecte de relations*, on dispose de relations de la forme :

$$(a_i - b_i m) = \prod_{p \in \mathcal{F}_r} p^{\nu_p(i)}, \quad (a_i - b_i \alpha)\mathcal{O}_K = \prod_{\mathfrak{p} \in \mathcal{F}_a} \mathfrak{p}^{\nu_\mathfrak{p}(i)}.$$

3.2.1 Dépendances entre relations

En imitation de ce qui est fait pour le contexte de la factorisation, il est naturel à partir ce point de chercher à obtenir, par la résolution d'un système linéaire, une combinaison multiplicative $Q(x)$ des $\phi_i = a_i - x b_i$ telle que $Q(\alpha)\mathcal{O}_K$ possède uniquement des valuations multiples de ℓ en les éléments de la base de facteurs $\mathfrak{p} \in \mathcal{F}_a$, et de même pour $Q(m)$ et les $p \in \mathcal{F}_r$. Toutefois, pour obtenir à partir de telles combinaisons des relations valables dans G_ℓ, on rencontre exactement les mêmes obstructions que le crible algébrique classique pour la factorisation. Ces obstructions peuvent être surmontées de manière exactement similaire[2], et on peut de la sorte établir des égalités entre deux puissances ℓ-èmes dans G_ℓ. Est-ce utile ? Oui, mais ce n'est pas la solution universelle au problème du calcul de logarithmes discrets. La voie ainsi ouverte fait appel aux étapes suivantes.

- Obtenir de nombreuses relations, sous la forme de paires (a, b) telles que $a_i - b_i m$ et $(a_i - b_i \alpha)\mathcal{O}_K$ sont friables.
- À supposer que c et d sont tels que $\log_c d$ doit être calculé, obtenir par un procédé de descente (voir section 3.5) une relation permettant de « relier » c et d à la base de facteurs. Plus spécifiquement, comme nous le verrons en 3.5, cela revient à déterminer une fraction rationnelle Q_c (respectivement Q_d) satisfaisant deux conditions. D'abord $Q_c(m) \cdot \hat{c} = C_r$, où C_r est

[1] Comme souvent, il est aussi possible d'installer le cadre de travail dans le cas de deux corps de nombres. Une telle construction est notamment proposée par [116].

[2] Les caractères utilisés pour ℓ grand sont de nature différente [181], mais contournent la même obstruction. Ce point est brièvement mentionné en 3.1.3.

friable, et en outre $Q_c(\alpha) = C_a$, où l'idéal C_a est friable, et $\hat{c}$ se réduit en c dans $\mathbb{F}_q^*$. La fraction rationnelle Q_d est soumise aux mêmes contraintes, avec les notations correspondantes D_r, D_a.

– On constitue alors un système linéaire à coefficients dans $\mathbb{Z}/\ell\mathbb{Z}$, où apparaissent les relations friables collectées, ainsi que les deux « seconds membres » donnés par les factorisations de C_r, C_a, ainsi que D_r, D_a. La résolution de ce système permet d'obtenir une fraction rationnelle Q telle que $Q(m)(\hat{c}^\gamma \hat{d}^\delta) \in \mathbb{Q}^{*\ell}$, et $Q(\alpha)\mathcal{O}_K$ est la puissance ℓ-ème d'un idéal. Les coefficients γ et δ correspondent ici aux coefficients affectant les lignes relatives à Q_c et Q_d dans la dépendance obtenue entre les équations du système linéaire. Nous ne détaillons pas le problème de la garantie de non-nullité de ces coefficients, et renvoyons à [69] qui propose quelques éléments d'analyse de ce problème, dans un contexte quelque peu différent toutefois.

– En considérant en outre des caractères (soit en exploitant plusieurs solutions au problème précédent, soit en les incluant dans la résolution du système linéaire), il est possible de combiner plusieurs des relations obtenues à l'étape précédente pour obtenir une expression pour laquelle on a $Q(\alpha) \in (\mathcal{O}_K^*)^\ell$. Ainsi, on obtient une relation multiplicative entre c et d dans G_ℓ, ce qui suffit à déterminer le logarithme $\log_c d$.

Il est important de mesurer le caractère inconfortable de cette approche. Un système linéaire doit être résolu pour *chaque* logarithme discret devant être calculé. C'est une obstruction pratique importante. Pour éviter cette difficulté, on introduit les logarithmes virtuels.

3.3 Logarithmes virtuels

Une approche préférable consiste à résoudre le système linéaire formé par les relations collectées, non pas pour obtenir une dépendance entre les relations, mais dans l'objectif d'obtenir des « logarithmes virtuels ». Informellement, pour un nombre premier $p \in \mathcal{F}_r$, on souhaite que la résolution du système linéaire permette de calculer le logarithme de p dans $G_\ell = \mathbb{F}_q^*/\mathbb{F}_q^{*\ell}$. Cette interprétation est correcte lorsqu'on se limite au côté rationnel. Pour le côté algébrique, le problème est sensiblement différent. En effet, les idéaux n'étant pas principaux, il y a une ambiguïté pour déterminer de manière univoque un élément de G_ℓ correspondant à un idéal $\mathfrak{p} \in \mathcal{F}_a$.

Dans le cas quelque peu idéaliste où groupe de classes et groupe d'unités sont connus, le choix arbitraire d'un des générateurs de $\mathfrak{p}^h$ (difficiles à calculer) permet de réinterpréter les relations collectées comme des relations dans G_ℓ. Dès lors, il est possible de pointer du doigt les éléments de G_ℓ dont on cherche les logarithmes. Ceci est hélas peu réaliste lorsque le polynôme définissant le corps de nombres considéré dépasse le cadre d'exemples-jouets. Le calcul du groupe de classes et des générateurs du groupe des unités devient alors prohibitif, et rend cette approche peu raisonnable.

Le concept de « logarithme virtuel » introduit par [116] et formalisé par [184] permet de contourner cette difficulté. Nous donnons ici une présentation de ce concept. Un point particulièrement remarquable, en comparaison avec l'ébauche proposée plus haut, est que les logarithmes virtuels nous amènent à considérer le noyau *à droite* de la matrice de relations constituée, à la différence de la situation précédente où une dépendance entre les relations était recherchée, impliquant dès lors le noyau *à gauche*.

3.3.1 Notations utilisées

Pour une meilleure concision des énoncés, nous notons π_r et π_a les morphismes d'anneaux liant respectivement les anneaux $\mathbb{Z}[m]$ et $\mathbb{Z}[\alpha]$ au corps fini $\mathbb{F}_q$ (voir diagramme 3.3). Les corps de fractions de ces anneaux sont $\mathbb{Q}$ et K, et leurs anneaux des entiers sont $\mathbb{Z}$ et $\mathcal{O}_K$. Nous nous intéressons à la détermination de logarithmes discrets modulo ℓ, c'est-à-dire à la détermination d'un caractère non trivial de G_ℓ (ou, de manière équivalente, de $\mathbb{F}_q^*$) dans $\mathbb{F}_\ell$. Les applications π_r et π_a sont commodément transformées en des applications multiplicatives en considérant en premier lieu la restriction aux éléments dont l'image n'est pas nulle, puis en étendant cette définition aux sous-groupes correspondants de $\mathbb{Q}^*$ et K^*. Le risque d'ambiguïté étant minime, la notation est maintenue identique.

Les bases de facteurs rationnelle et algébrique sont $\mathcal{F}_r$ et $\mathcal{F}_a$. Comme dans [181], on désigne par $\mathcal{F}_a$-unité un élément $u \in K^*$ tel que l'idéal $u\mathcal{O}_K$ a une valuation nulle en tout idéal premier de $\mathcal{O}_K$ qui n'appartient *pas* à $\mathcal{F}_a$. L'ensemble des $\mathcal{F}_a$-unités est un sous-groupe multiplicatif de K^*. On définit de même des $\mathcal{F}_r$-unités, formant un sous-groupe multiplicatif de $\mathbb{Q}^*$. On note que le choix des bases de facteurs prescrit en particulier que les idéaux (q) et $\langle q, \alpha - m\rangle$ n'appartiennent pas à $\mathcal{F}_r$ et $\mathcal{F}_a$. Dès lors, les applications multiplicatives π_r et π_a telles que construites plus haut sont bien définies sur les $\mathcal{F}_r$-unités et $\mathcal{F}_a$-unités. Plus « visuellement », la construction des $\mathcal{F}_a$-unités et de π_a s'appuie sur les propriétés suivantes (la notation $A \lhd B$ désigne ici un sous-groupe) :

$$\begin{aligned} \{x \in \mathbb{Z}[\alpha],\ \pi_a(x) \neq 0\} &\subset \mathbb{Z}[\alpha] \\ \{\mathcal{F}_a\text{-unités}\} \lhd \langle x \in \mathbb{Z}[\alpha],\ \pi_a(x) \neq 0\rangle &\lhd K^* \\ \pi_a :\ \langle x \in \mathbb{Z}[\alpha],\ \pi_a(x) \neq 0\rangle &\to \mathbb{F}_q^* \end{aligned}$$

3.3.2 Caractères de $\mathbb{F}_q^*$ vers $\mathbb{F}_\ell$

Les caractères de $\mathbb{F}_q^*$ vers $\mathbb{F}_\ell$ forment un $\mathbb{F}_\ell$-espace vectoriel de dimension 1, puisque $\mathbb{F}_q^*$ est cyclique. Pour déterminer un tel caractère *non trivial* χ, on se fixe comme objectif la détermination de deux applications $\xi_r = \chi \circ \pi_r$ et $\xi_a = \chi \circ \pi_a$, via l'expression de conditions nécessaires. Pour déterminer ξ_a (et de même pour ξ_r), on s'intéresse aux valeurs prises sur les $\mathcal{F}_a$-unités. Nous remarquons que le sous-groupe constitué par les *puissances* ℓ-èmes de celles-ci est nécessairement dans le noyau de ξ_a. Aussi, il est suffisant de s'intéresser au quotient suivant, noté Σ_a, comme groupe de départ (on définit de même Σ_r) :

Notation 3.4.

$$\Sigma_a = \{x \in \{\mathcal{F}_a\text{-}\textit{unités}\}\}/\{x^\ell,\ x \in \{\mathcal{F}_a\text{-}\textit{unités}\}\}.$$

Notons que l'image par π_a de Σ_a dans $G_\ell = \mathbb{F}_q^*/\mathbb{F}_q^{*\ell}$ est bien définie.

L'objectif que nous nous fixons est la détermination d'un vecteur de logarithmes virtuels, comme suit.

Caractérisation d'un élément de Σ_r et Σ_a Dans un premier lieu, nous déterminons les informations dont nous disposons sur Σ_r et Σ_a. Un élément de Σ_r est en particulier un rationnel z non nul, à une puissance ℓ-ème près. La connaissance de $\nu_p(z) \bmod \ell$, pour chaque $p \in \mathcal{F}_r$, suffit à déterminer z au signe près, c'est-à-dire aux unités près. Comme ℓ est impair, cette distinction est sans objet. Un élément de Σ_r est donc entièrement caractérisé par le vecteur des p-valuations modulo ℓ pour $p \in \mathcal{F}_r$.

Les $\mathfrak{p}$-valuations modulo ℓ de l'idéal engendré par un élément de Σ_a permettent similairement de déterminer cet élément à une unité près, soit à un élément de $\mathcal{O}_K^*/\mathcal{O}_K^{*\ell}$ près. Le petit degré de K

ne permettant bien sûr pas que K^* contienne une racine ℓ-ème de l'unité, ce groupe est isomorphe à $\mathbb{F}_\ell^{r_K}$, où r_K est le rang de la partie sans torsion du groupe des unités $\mathcal{O}_K^*$. Les caractères ℓ-adiques (parfois appelés « *Schirokauer maps* ») proposés par [181], que nous évoquons en 3.1.3, permettent de lever cette ambiguïté, sous des conditions ou heuristiques qui sont discutées dans [181]. Ainsi, un élément z de Σ_a est uniquement déterminé par le vecteur des valuations $\nu_p(z) \bmod \ell$ et les caractères ℓ-adiques.

Logarithmes virtuels comme forme linéaires Nous pouvons donc identifier Σ_r et Σ_a à des sous-espaces des espaces vectoriels $\mathbb{F}_\ell^{\#\mathcal{F}_r}$ et $\mathbb{F}_\ell^{\#\mathcal{F}_a+r_K}$. Les applications ξ_r et ξ_a recherchées sont des éléments des duaux de ces deux espaces :

$$\begin{aligned}\xi_r &\in L(\Sigma_r, \mathbb{F}_\ell),\\ \xi_a &\in L(\Sigma_a, \mathbb{F}_\ell).\end{aligned}$$

Considérons maintenant l'espace produit $\Sigma = \Sigma_r \times \Sigma_a$, et l'application $\xi = \chi \circ \pi$, où π est définie comme suit :

$$\pi : \left\{ \begin{array}{rcl} \Sigma = \Sigma_r \times \Sigma_a & \longrightarrow & G_\ell, \\ (z_r, z_a) & \longmapsto & \pi((z_r, z_a)) = \pi_r(z_r)/\pi_a(z_a). \end{array} \right.$$

Une conséquence immédiate de cette notation est $\xi((z_r, z_a)) = \xi_r(z_r) - \xi_a(z_a)$. Nous pouvons maintenant définir la notion de logarithme virtuel.

Définition 3.5 (Logarithmes virtuels). *On appelle* vecteur de logarithmes virtuels *la donnée d'un $(\#\mathcal{F}_r + \#\mathcal{F}_a + r_K)$-uplet d'éléments de $\mathbb{F}_\ell$ déterminant un élément $\xi \in L(\Sigma, \mathbb{F}_\ell)$.*

Un vecteur de logarithmes virtuels peut aussi être vu par la donnée conjointe de coordonnées pour ξ_r et ξ_a. Il nous importe de déterminer un vecteur de logarithmes virtuels correspondant à une forme linéaire ξ qui s'écrive effectivement sous la forme $\xi = \chi \circ \pi$. Pour cela, nous établissons d'abord des conditions nécessaires.

Conditions nécessaires découlant de $\xi = \chi \circ \pi$ Une *relation*, telle que collectée par le processus de crible, est la donnée d'une $\mathcal{F}_r$-unité et d'une $\mathcal{F}_a$-unité, soit en particulier d'un élément $(z_r, z_a) \in \Sigma$, qui peut être caractérisé de manière unique par le vecteur des p-valuations (rationnelles), des $\mathfrak{p}$-valuations (algébriques), et des caractères. Le couple (z_r, z_a) possède par construction la propriété $\pi_r(z_r) = \pi_a(z_a)$ (dans G_ℓ, soit à une puissance ℓ-ème près), ce qui implique en particulier (par $\xi = \chi \circ \pi$) :

$$\begin{aligned}(z_r, z_a) &\in \operatorname{Ker} \pi,\\ (z_r, z_a) &\in \operatorname{Ker} \xi.\end{aligned}$$

Une relation donne donc une condition nécessaire satisfaite par l'application ξ, confinant celle-ci dans un hyperplan de $L(\Sigma, \mathbb{F}_\ell)$. De la même façon, un ensemble de relations engendre un certain sous-espace de Σ, qu'on dénomme « espace engendré par les relations ». Cet espace constitue un ensemble de conditions nécessaires, qui confinent ξ dans un sous-espace de $L(\Sigma, \mathbb{F}_\ell)$.

Conditions suffisantes Pour garantir que ξ s'écrive sous la forme $\xi = \chi \circ \pi$, il suffit qu'on ait $\operatorname{Ker} \pi \subset \operatorname{Ker} \xi$. Un vecteur de logarithmes virtuels construit (par le calcul) de sorte à rester compatible avec toutes les relations, c'est-à-dire un élément du noyau de la matrice formée par les relations,

ENTRÉE : Un corps fini $\mathbb{F}_q$, et un diviseur ℓ de $q-1$ (voir conditions 3.2 page 27).
SORTIE : Un vecteur de « logarithmes virtuels » définissant une fonction ξ telle que présentée en 3.3.

1. Construire deux polynômes f et g de sorte à obtenir un diagramme semblable à la figure 3.3 : les deux polynômes doivent avoir une racine commune modulo q. Il importe bien entendu de respecter les contraintes de tailles indiquées par l'analyse asymptotique de NFS.
2. Construire les bases de facteurs $\mathcal{F}_r$ et $\mathcal{F}_a$.
3. Collecter de nombreuses relations, sous la forme de paires (a, b), satisfaisant les conditions de friabilité requises.
4. Pour chaque relation, calculer la factorisation complète du nombre rationnel $a - bm$, et de l'idéal $(a - b\alpha)\mathcal{O}_K$. Adjoindre le vecteur de caractères ℓ-adiques $\Lambda(a - b\alpha) \in \mathbb{F}_\ell^d$ (voir section 3.1.3).
5. Résoudre le système linéaire homogène à $\#\mathcal{F}_r + \#\mathcal{F}_a + r_K$ inconnues ainsi défini par les relations, à coefficients dans $\mathbb{F}_\ell$.

ALGORITHME 3.6 : NFS-DL, partie 1 : calcul des logarithmes virtuels.

correspond à un ξ qui vérifie cette propriété dès lors que l'ensemble des relations considérées engendre $\mathrm{Ker}\,\pi$ dans sa totalité[3].

Il est possible à ce stade de formuler une première partie d'un algorithme de calcul de logarithmes discrets dans G_ℓ, dans la mesure où nous pouvons donner un sens à l'objet construit. Les premières étapes sont données par le pseudo-algorithme 3.6 Les coordonnées du vecteur solution produit par l'algorithme 3.6 sont les « logarithmes virtuels ». La correspondance de ces logarithmes virtuels avec les logarithmes d'éléments calculables de G_ℓ peut être établie aisément pour le côté rationnel. Pour le côté algébrique, calculer cette correspondance (non complètement univoque) requiert un effort de calcul important, puisque le groupe des classes et le groupe des unités de $\mathcal{O}_K$ doivent être calculés. Le point essentiel est que ces logarithmes virtuels sont calculables par la simple résolution du système linéaire modulo ℓ rencontré, et que leur seule utilité est de définir la fonction ξ. Il convient donc d'exploiter cette fonction ξ pour calculer des logarithmes discrets dans G_ℓ, ce que nous détaillons dans les paragraphes suivants.

3.3.3 Définition d'un logarithme à partir de ξ

La procédure de descente que nous abordons en 3.5 est utilisée pour calculer le logarithme discret modulo ℓ d'un élément cible $t \in \mathbb{F}_q^*$ (l'élément t est donc important seulement aux puissances ℓ-èmes près, on peut le voir comme un élément de G_ℓ). Plus spécifiquement, la procédure de descente construit un relèvement $\hat{t}$ et une fraction rationnelle $Q(x)$ correspondant exactement aux objets considérés dans la proposition suivante. Nous établissons donc ici en quelque sorte le « cahier des charges » de l'étape de descente, qui est étudiée plus loin.

PROPOSITION 3.7. *Soit ξ construite par l'algorithme 3.6, à partir d'un ensemble de relations suffisamment riche pour garantir $\mathrm{Ker}\,\pi \subset \mathrm{Ker}\,\xi$. Soit $t \in G_\ell$, et $\hat{t} \in \mathbb{Q}$ tel que $\pi_r(\hat{t}) \equiv t$ dans G_ℓ. Soit $Q(x)$ une fraction rationnelle vérifiant :*

$$Q(m)\hat{t} \in \Sigma_r, \quad Q(\alpha) \in \Sigma_a.$$

[3]Cf la discussion à ce sujet en 3.3.4.

Notons $z^* = (Q(m)\hat{t}, Q(\alpha))$. L'élément de $\mathbb{F}_\ell$ donné par $\chi(t) \overset{\text{déf}}{=} \xi(z^*)$ est bien défini.

DÉMONSTRATION. On a par construction $z^* \in \Sigma$, et $\pi(z^*) = t$. On exploite alors simplement le fait que $\operatorname{Ker}\pi \subset \operatorname{Ker}\xi$ par hypothèse. ■

On remarque, comme codicille à la proposition 3.7, que la situation $Q(m) \in \Sigma_r$, $Q(\alpha)\hat{t} \in \Sigma_a$ s'exploite de façon semblable.

La proposition précédente est suffisante pour calculer des logarithmes discrets dans G_ℓ. En effet, pour calculer le logarithme discret de d relativement à la base c, on doit calculer $\frac{\log d}{\log c} = \frac{\chi(d)}{\chi(c)}$. D'une manière générale, pour calculer k logarithmes discrets, il suffit de calculer $k+1$ valeurs de χ, la première servant à l'« étalonnage ». Ce faisant, nous exploitons de manière cruciale le fait que le caractère χ construit est non trivial.

3.3.4 Vecteurs sporadiques et vérification des logarithmes virtuels

Un écueil pratique important dans le calcul de logarithmes discrets est l'hypothèse requise par la proposition 3.7 : « un ensemble de relations suffisamment riche pour garantir $\operatorname{Ker}\pi \subset \operatorname{Ker}\xi$ ». En pratique, une telle hypothèse est irréaliste, pour plusieurs raisons.

En premier lieu, les relations collectées n'impliquent jamais la *totalité* des idéaux des bases de facteurs. Dès lors, ces relations ne peuvent assurément pas engendrer la totalité de $\operatorname{Ker}\pi$. Poursuivre l'objectif de collecter suffisamment de relations pour garantir cette propriété est irréaliste. Toutefois, quitte à remplacer dans les développements qui précèdent les bases de facteurs $\mathcal{F}_r$ et $\mathcal{F}_a$ par leurs sous-ensembles d'idéaux *rencontrés*, il est possible de contourner cette difficulté.

Un second écueil, plus préoccupant, est lié à la présence de *vecteurs sporadiques* parmi les solutions du système linéaire. La cause de ce problème tient encore à la sous-détermination de $\operatorname{Ker}\pi$ par l'ensemble des relations collectées. Considérons par exemple trois nombres premiers ou idéaux arbitraires, p, $\mathfrak{p}$, et $\mathfrak{p}'$, et plaçons-nous dans le cas où ces derniers n'apparaissent que dans les deux relations suivantes :

$$\begin{aligned} a_1 - b_1 m &= \ldots \times p, \quad a_1 - b_1 \alpha = \ldots \times \mathfrak{p}, \\ a_2 - b_2 m &= \ldots \times p, \quad a_2 - b_2 \alpha = \ldots \times \mathfrak{p}'. \end{aligned}$$

Dans un tel cas, considérons l'élément de $L(\Sigma, \mathbb{F}_\ell)$ défini par $p \to 1$, $\mathfrak{p} \to -1$, $\mathfrak{p}' \to -1$. Son noyau contient l'espace engendré par les relations collectées. Hélas ce vecteur est un artefact du calcul : l'espace engendré par les relations est simplement trop petit. Notons brièvement que la situation engendrant un tel artefact est une situation connue des procédures de *filtrage* rencontrées dans l'algorithme NFS. En déployant les techniques adéquates (voir [177] ou encore [39]), un exemple semblable à celui proposé peut être évité, mais la difficulté fondamentale demeure : il est important de pouvoir *vérifier* autant que faire se peut les logarithmes virtuels calculés.

Comment vérifier un logarithme discret ? Il s'agit d'une tâche relativement aisée, des lors qu'on sait à quelle grandeur le logarithme correspond. Pour vérifier que $\log_g x = k$, il suffit de tester l'identité $g^k = x$. Pour réaliser la même vérification dans G_ℓ, il suffit de vérifier l'égalité aux puissances près, c'est-à-dire $(g^k/x)^{(q-1)/\ell} = 1$. De la sorte, on peut vérifier les logarithmes du côté rationnel. Toutefois, ce point de vue est peu utile pour vérifier les logarithmes virtuels côté algébrique. Comme on l'a déjà mentionné, la correspondance entre ces logarithmes virtuels et des éléments de $\mathbb{F}_q^*$ est difficile à obtenir de manière effective (voir [184]).

ENTRÉE : Le vecteur construit par l'algorithme 3.6, définissant l'application ξ.
L'ensemble des structures algébriques utilisées par l'algorithme 3.6
SORTIE : Un ensemble C_{ok} indiquant les coordonnées a priori correctes.
Un ensemble C_{bad} indiquant les coordonnées a priori erronées.

1. $\mathsf{C}_{\text{ok}} \leftarrow \emptyset$, $\mathsf{C}_{\text{bad}} \leftarrow \emptyset$.
2. Pour chaque $p \in \mathcal{F}_r$, puis $\mathfrak{p} \in \mathcal{F}_a$:

3a. Dans le cas où $p \in \mathcal{F}_r$, soit $z = (p, 1)$.

3b. Dans le cas où $\mathfrak{p} \in \mathcal{F}_a$, soit $Q(x) \in \mathbb{Z}[x]$ telle que $Q(\alpha)$ est une $\mathcal{F}_a$-unité, et $\nu_{\mathfrak{p}}(Q(\alpha)) \neq 0$.

Soit alors $z = (1, Q(\alpha)^{-1}) \in \Sigma$.

4. Calculer $\chi(z)$.
5. Construire de même z' pour un autre élément de $\mathcal{F}_r$ ou $\mathcal{F}_a$.
6. Si $\pi(z)^{\chi(z')} = \pi(z')^{\chi(z)}$, alors $\mathsf{C}_{\text{ok}} \leftarrow \mathsf{C}_{\text{ok}} \cup \{\text{support des factorisations de } z \text{ et } z'\}$.
7. Si $\pi(z)^{\chi(z')} \neq \pi(z')^{\chi(z)}$, alors $\mathsf{C}_{\text{bad}} \leftarrow \mathsf{C}_{\text{bad}} \cup \{\text{support des factorisations de } z \text{ et } z'\}$.
8. $\mathsf{C}_{\text{ok}} \leftarrow \mathsf{C}_{\text{ok}} - \mathsf{C}_{\text{bad}}$.

ALGORITHME 3.8 : Vérification des logarithmes virtuels produits par l'algorithme 3.6.

Le bon point de vue consiste à revenir à la définition de l'application ξ. Cette application calcule l'image d'éléments de Σ. Pour vérifier des logarithmes, l'approche la plus pratique consiste à fabriquer un élément *creux* de Σ, ayant la caractéristique de faire intervenir un logarithme virtuel donné. On vérifie la cohérence des logarithmes obtenus pour deux éléments z et z' de Σ en s'assurant qu'on a $\pi(z)^{\xi(z')} = \pi(z')^{\xi(z)}$. Cette cohérence donne une indication de la correction des coordonnées correspondantes.

Cette approche peut se formuler pour le cas simple du côté rationnel. Soit $p \in \mathcal{F}_r$ un nombre premier. On souhaite vérifier que la coordonnée correspondante du vecteur solution définissant ξ est correcte. Pour cela, considérons $z = (p, 1) \in \Sigma$. On a $\pi(z) = p$. Il faut vérifier que $\chi(p)$ est bien le logarithme discret de p. Pour étalonner cette vérification, il faut l'effectuer par exemple sur une paire de nombres premiers. On vérifie ainsi la cohérence.

On souhaite faire une vérification similaire pour un logarithme virtuel côté algébrique. Considérons un idéal $\mathfrak{p} = \langle p, \alpha - r \rangle \in \mathcal{F}_a$. Soit une fraction rationnelle $Q(x)$ telle que :

- La valuation $\nu_{\mathfrak{p}}(Q(\alpha))$ n'est pas nulle.
- $Q(\alpha)$ est une $\mathcal{F}_a$-unité.

Les techniques nécessaires à la construction d'une telle fraction rationnelle Q apparaissent dans le « lattice sieve » [174], ainsi qu'en 3.5.3. On considère alors $z = (1, Q(\alpha)^{-1}) \in \Sigma$. On a $\pi(z) = Q(m) \mod p$, et on peut calculer $\chi(\pi(z)) = \xi(z)$. En comparant les valeurs obtenues pour deux fractions rationnelles Q et Q' obtenues de cette façon, on peut juger de la correction des coordonnées associées aux idéaux apparaissant dans les factorisations de $Q(\alpha)$ et $Q'(\alpha)$.

L'algorithme 3.8 construit deux ensembles C_{ok} et C_{bad} qui permettent d'obtenir une information utile sur les coordonnées probablement incorrectes du vecteur calculé par l'algorithme 3.6. Cet algorithme est essentiellement heuristique, et n'a pas d'autre prétention que celle de traiter le cas de vecteurs sporadiques de petit poids. Il est important de voir qu'en termes de complexité, la détermination de la fraction rationnelle $Q(x)$, bien que non triviale, est peu coûteuse, car la friabilité n'est requise que d'un seul côté (on ne demande *pas* que $Q(m)$ soit une $\mathcal{F}_r$-unité). L'analyse montre que l'ensemble de cette vérification peut se faire en un temps négligeable par rapport au reste du calcul.

3.4 L'algorithme FFS

Comme mentionné en 3.1.1, il existe plusieurs variantes de l'algorithme NFS pour le calcul de logarithmes discrets. Nous ne donnons pas de détail précis sur ces autres variantes, en particulier nous n'abordons pas la variante NFS-HD proposée par [122].

Nous donnons quelques rapides éléments cependant sur l'algorithme du *Function Field Sieve* (FFS). L'algorithme FFS se situe à l'autre extrémité du spectre par rapport à NFS, puisque FFS est adapté au calcul de logarithmes discrets dans $\mathbb{F}_{p^n}$ avec p petit, plus précisément avec p et n soumis à la condition $p = L_q[\ell, \cdot]$ avec $\ell < 1/3$ et $q = p^n$. La description de FFS est faite dans [3, 6], et d'un point de vue plus pratique dans [115]. Nous notons pour les paragraphes qui suivent $k = \mathbb{F}_p$.

3.4.1 Gain automatique pour la sélection polynomiale

FFS fonctionne de manière très similaire à NFS. Un degré de liberté considérable, toutefois, est offert par le fait que tous les corps finis de cardinal p^n sont isomorphes. Aussi, il est loisible de choisir un polynôme de définition adapté à une contrainte qu'on souhaite satisfaire. C'est exactement ainsi que fonctionne la sélection polynomiale. Le cadre de travail de FFS est celui de NFS, avec l'anneau $\mathbb{Z}$ remplacé par $k[t]$. La tâche de sélection polynomiale requiert de déterminer deux polynômes f et g à coefficients dans $k[t]$. Ces polynômes sont écrits indifféremment $f(t, x)$ pour souligner f comme étant un polynôme bivarié en t et x, ou $f(x)$ pour souligner l'appartenance de f à $k[t][x]$. Les polynômes f et g construits doivent disposer d'une racine commune dans $\mathbb{F}_{p^n}$. Pour ce faire, en fonction de la taille de $q = p^n$ et des estimations données par l'analyse de complexité, on choisit f et g de degré convenable, avec des coefficients $f_i, g_i \in k[t]$ tels que le résultant $\mathrm{Res}_x(f, g)$ soit de degré au moins égal à n. Ce processus de choix est répété jusqu'à ce qu'apparaisse un polynôme irréductible de degré n dans la factorisation de $\mathrm{Res}_x(f, g)$. Comme cela est détaillé dans [115], ceci permet en particulier de construire un polynôme f dont les coefficients f_i sont de petit degré. La conséquence *in fine* de cette propriété est que l'algorithme FFS a une complexité heuristique semblable à celle de l'algorithme SNFS, à savoir en $L_q[1/3, (32/9)^{1/3}]$.

3.4.2 Objets mathématiques considérés

Notons $\mathcal{C}$ la courbe définie sur k d'équation $f(t, x) = 0$. L'analogue pour FFS du corps de nombres $\mathbb{Q}(\alpha)$ qui intervient dans NFS est le corps de fractions de $k[\mathcal{C}] = k(t)[x]/f(x)$, c'est-à-dire le *corps de fonctions* $k(\mathcal{C})$ de la courbe $\mathcal{C}$ (notons que $k[\mathcal{C}]$ est un objet fondamentalement affine). La transcription en termes de corps de fonctions des concepts utilisés pour le côté algébrique dans NFS est essentiellement une récriture (voir notamment l'ouvrage [144]), plusieurs « dictionnaires » peuvent être utilisés, suivant qu'on souhaite mettre en avant l'aspect géométrique (avec la courbe $\mathcal{C}$), son corps de fonction $k(\mathcal{C})$, ou encore insister sur l'analogie avec NFS en considérant avant tout $k(\mathcal{C})$ comme une extension de $k(t)$. La factorisation d'un idéal, concept mis en avant pour NFS, est (par exemple) remplacé par la décomposition d'un diviseur.

Le point de vue affine sur le corps de fonctions $k(\mathcal{C})$ (corps de fraction de $k(t)[x]/f(x)$) privilégie x comme étant l'élément adjoint à $k(t)$ pour former $k(\mathcal{C})$. C'est à travers ce point de vue que les considérations les plus directement liées à NFS sont interprétées. Notons A la clôture intégrale de $k[t]$ dans $k(\mathcal{C})$, soit l'analogue de l'anneau des entiers qui intervient dans le contexte de NFS. L'anneau $k[\mathcal{C}]$ est intégralement clos (donc $A = k[\mathcal{C}]$) dans $k(\mathcal{C})$ si $\mathcal{C}$ n'a pas de singularité affine, et l'analogue pour FFS du calcul de l'anneau des entiers est la détermination des singularités affines de $\mathcal{C}$ (détermination, au demeurant, qui peut être réalisée sans se heurter au coûteux obstacle de la

factorisation d'entiers). D'autre part, les objets qui sont écartés par ce point de vue affine sont les points à l'infini de la courbe $\mathcal{C}$. Ceux-ci définissent exactement les unités de A, qui sont donc plus faciles à déterminer que dans le cas NFS. Ces deux points illustrent comment le cas FFS, quel que soit le polynôme f choisi, est beaucoup plus simple que le cas NFS, au moins en ce qui concerne ce qui apparaît dans NFS comme des obstructions. Le point essentiel de cette différence est que la factorisation des *polynômes* sur les corps finis se résout en temps polynomial.

Pour autant, la flexibilité offerte par FFS dans le choix des polynômes f et g offre aussi le loisir d'éviter complètement ces « difficultés » potentielles. On peut contraindre la courbe $\mathcal{C}$ à n'avoir aucune singularité affine, et il est possible de choisir f comme définissant une courbe $\mathcal{C}$ dite « C_{ab} », de sorte à forcer une unique place à l'infini [146]. Ceci implique que le groupe des unités de A est trivial (réduit à k^*). Ces restrictions sont confortables pour l'exposition, mais comme on l'a dit, non nécessaires au moins d'un point de vue théorique. On peut concevoir qu'il est dommage de restreindre l'étude à ces cas « faciles ». En effet, une telle restriction limite la marge de manœuvre dans le choix du polynôme f. Il est loin d'être évident que les caractéristiques d'un « bon » polynôme au regard des performances qu'il permet pour le crible soient orthogonales aux restrictions qu'on impose. En évitant de poser de telles restrictions artificielles (particulièrement concernant les singularités affines), on peut concevoir que la recherche du meilleur polynôme f possible soit à la fois plus simple (espace de recherche défini avec moins de conditions), et donne un meilleur résultat.

3.4.3 Bases de facteurs

Dans le cas facile où $k[\mathcal{C}] = A$ est intégralement clos (pas de singularité sur la courbe affine), et où il n'existe qu'une seule place à l'infini, nous décrivons les analogues des bases de facteurs. L'analogue pour FFS de la borne de friabilité B dans NFS est une borne en degré, notée β. La base de facteurs rationnelle est constituée, à l'instar de la définition 2.2, des polynômes de degré inférieur ou égal à β, et des facteurs irréductibles du terme de tête (en x) de $g(t, x)$. Ceci revient à une base de facteurs dont le cardinal est approximativement q^β/β (il est aussi possible de fixer le nombre B de polynômes voulus, sans imposer une limite claire en termes de degré). La base de facteurs algébrique est constituée de places du corps de fonction, soumises à des conditions semblables à celles de la définition 2.3. Soit P un point de $\mathcal{C}(\bar{k})$. On note $k(P)$ le corps minimal de définition de coordonnées de P. Notons en outre le morphisme $\mathbf{x}$ de $\mathcal{C}$ vers $\mathbb{P}^1$ qui à $(x : y : z)$ associe $(x : z)$. Ce morphisme est de degré $\deg_x f$. Les places considérées dans la base de facteurs algébrique sont les classes d'équivalence sous l'action de $\mathrm{Gal}(\bar{k}/k)$ des points $P \in \mathcal{C}(\bar{k})$ tels que d'une part, $k(\mathbf{x}(P))$ est de degré au plus β, et d'autre part, $[k(P) : k(\mathbf{x}(P))] = 1$ (cette dernière condition correspond au degré d'inertie intervenant dans la définition 2.3). Pour constituer la base de facteurs dans le cas où des singularités existent, ou en présence de plusieurs places à l'infini, il convient d'ajouter à cette base de facteurs un nombre fini de places, ce que nous ne détaillons pas.

3.4.4 Schéma de fonctionnement

Pour donner une formulation un peu plus explicite du principe de fonctionnement de FFS, supposons donné un couple de polynômes $f, g \in k[t, x]$ tels que $\mathrm{Res}_x(f, g)$ est multiple d'un polynôme irréductible de degré n, ce degré étant le degré sur $k = \mathbb{F}_p$ du corps fini $\mathbb{F}_q$ dans lequel on souhaite calculer des logarithmes discrets. Exactement comme dans l'algorithme NFS, on considère dans FFS des *paires* (a, b), où a et b sont ici des *polynômes* de $k[t]$, soumis à une contrainte de degré comme dans NFS. On note $\phi(t, x) = a(x) - tb(x)$. Suivant l'analogue du diagramme 3.3, la friabilité rationnelle revient à la friabilité du polynôme $\mathrm{Res}_x(\phi, g)$, tandis que la friabilité du côté algébrique

correspond à la friabilité du polynôme $\mathrm{Res}_x(\phi, f)$. D'autre part, d'un point de vue pratique, les techniques de *lattice sieving* [174] employées pour NFS pour la recherche de relations s'appliquent aussi à FFS, au moins dans les grandes lignes [115]. Les optimisations très attachées au contexte entier comme [77, T13] se transcrivent quant à elles moins immédiatement.

Considérons à titre d'illustration une combinaison multiplicative de relations, telle que toutes les valuations sont multiples de ℓ. Comme il est accessible dans le cas FFS d'obtenir une décomposition complète des idéaux, une telle combinaison est *presque* une puissance ℓ-ème dans $k(\mathcal{C})$, à un élément de k^* près. La seule obstruction importante que le cadre que nous présentons n'a pas évacuée à ce stade est celle du groupe de classes (ici, la jacobienne), exactement similaire au cas de NFS décrit en 2.1.3. Si la ℓ-torsion de ce groupe de classes est non triviale, alors le recours à des caractères est nécessaire pour obtenir une puissance ℓ-ème. En pratique, comme indiqué en 3.1.2, ℓ est un grand nombre premier, et on peut faire la supposition que ℓ est premier au groupe de classes en question.

Bien que l'illustration que nous venons de donner mette en avant principalement l'approche de combinaison multiplicatives de relations, vue brièvement en 3.2.1, il est naturellement possible de développer les outils proposés en 3.3 pour développer un concept de logarithmes virtuels dans le cas FFS.

3.4.5 Cas particulier : FFS avec deux côtés rationnels

Une catégorie d'instances particulières de FFS peut être produite en considérant des couples de polynômes (f, g) particuliers, de la forme

$$f(x, y) = y - u(x), \quad g(x, y) = x - v(y).$$

Supposons, dans un tel cas, que $\mathrm{Res}_x(f, g)$ a parmi ses facteurs irréductibles un polynôme $Q(y)$ de degré n. Alors le résultant $\mathrm{Res}_y(f, g)$ a un facteur irréductible $P(x)$ de degré n. Quotienter par f, puis g, ou dans l'autre sens, sont deux façons d'atteindre deux copies isomorphes du corps fini $\mathbb{F}_{p^n}$. Un choix arbitraire de représentation de $\mathbb{F}_{p^n}$ étant fait, nous pouvons choisir deux racines ρ et σ de P et Q, vérifiant en outre $\rho = v(\sigma)$, et $\sigma = u(\rho)$. Nous pouvons dans ce contexte obtenir le diagramme de la figure 3.9.

Dans un tel contexte, il n'y a plus de *courbe* visible (ou, dit autrement, la seule courbe qui intervient est la droite projective $\mathbb{P}^1$). Les bases de facteurs sont, de chaque côté, formées par les polynômes irréductibles en-deçà d'un certain degré :

$$\mathcal{F}_f = \{\phi(x) \in \mathbb{F}_p[x],\ \phi \text{ irréductible},\ \deg \phi \leq B\},$$
$$\mathcal{F}_g = \{\psi(y) \in \mathbb{F}_p[y],\ \psi \text{ irréductible},\ \deg \psi \leq B\}.$$

Une paire (a, b) est friable dès lors que les deux polynômes $a(x) - u(x)b(x)$ et $a(v(y)) - yb(v(y))$ le sont. Dans un tel cas, on a :

$$a(x) - u(x)b(x) = \mathrm{Res}_y(a(x) - b(x)y, f) = \prod_{\phi \in \mathcal{F}_f} \phi^{k_\phi},$$
$$a(v(y)) - yb(v(y)) = \mathrm{Res}_x(a(x) - b(x)y, g) = \prod_{\psi \in \mathcal{F}_g} \psi^{\ell_\psi}.$$

Il est assez pratique, dans ce contexte précis, de disposer d'une application explicite bien définie des bases de facteurs vers le corps cible $\mathbb{F}_{p^n}$. Nous utilisons les applications i_f et i_g qui apparaissent

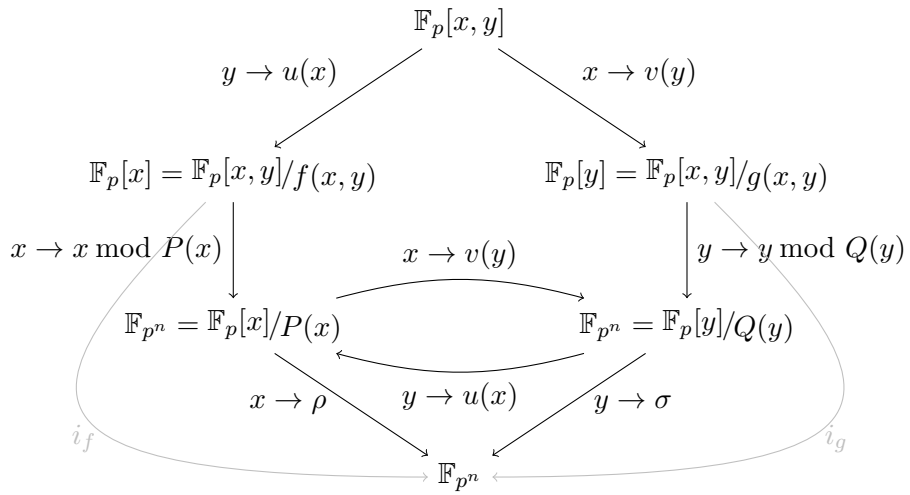


FIG. 3.9 : Schéma de FFS avec deux côtés rationnels

sur la figure 3.9, respectivement définies par $i_f(x) = \rho$ et $i_g(y) = \sigma$. Ceci permet d'écrire la relation suivante, issue de la double friabilité de la paire (a, b) considérée, et de la commutativité du diagramme 3.9 :

$$\prod_{\phi \in \mathcal{F}_f} \phi(\rho)^{k_\phi} = \prod_{\psi(\sigma) \in \mathcal{F}_g} \psi^{\ell_\psi}.$$

Il peut s'avérer préférable dans une utilisation de cet exemple, de ne pas se restreindre à des paires (a, b), c'est-à-dire à des fonctions $a(x) - yb(x)$ qui sont linéaires en y. Le déséquilibre entre les degrés de $a(x) - u(x)b(x)$ et $a(v(y)) - yb(v(y))$ est un handicap potentiel. En revanche, rien n'interdit du point de vue théorique de considérer des fonctions de plus haut degré en y. L'implantation d'un tel schéma, toutefois, ne pourrait réutiliser directement du code existant, généralement optimisé pour le cas où les fonctions considérées sont linéaires en l'une de leurs variables.

3.4.6 Cas particulier : l'algorithme de Coppersmith

L'algorithme de Coppersmith [50] fut le précurseur des algorithmes de la « famille » du crible algébrique. En reprenant comme illustration l'exemple de l'article [50], il est possible d'exposer son fonctionnement avec relativement peu de bagage. Le contexte emprunte directement à celui qu'on a posé pour l'algorithme FFS (bien qu'évidemment nous commettons là un anachronisme évident). La *courbe*, dans le contexte de l'algorithme de Coppersmith, et en particulier dans le cas particulier destiné au calcul de logarithmes discrets dans $\mathbb{F}_{2^{127}} \cong \mathbb{F}_2[t]/t^{127} + t + 1$, est donnée par l'équation $f = x^4 - (t^2 + t)$. Le « côté rationnel », quant à lui, est déterminé par le polynôme $g = x - t^{32}$. Une *paire* (a, b), constituée de deux polynômes de $\mathbb{F}_2[t]$, donne des normes rationnelles et algébriques qui s'écrivent :

$$D(t) = \mathrm{Res}_x(a - bx, f) = a^4 + (t^2 + t)b^4,$$
$$C(t) = \mathrm{Res}_x(a - bx, g) = a + t^{32}b.$$

Ces grandeurs vérifient $C(t)^4 = D(t) \bmod (t^{127} + t + 1)$. Dès lors, lorsque toutes deux sont friables, une relation est établie dans $\mathbb{F}_{2^{127}}$. La collecte de relations se poursuit comme à l'accoutumée.

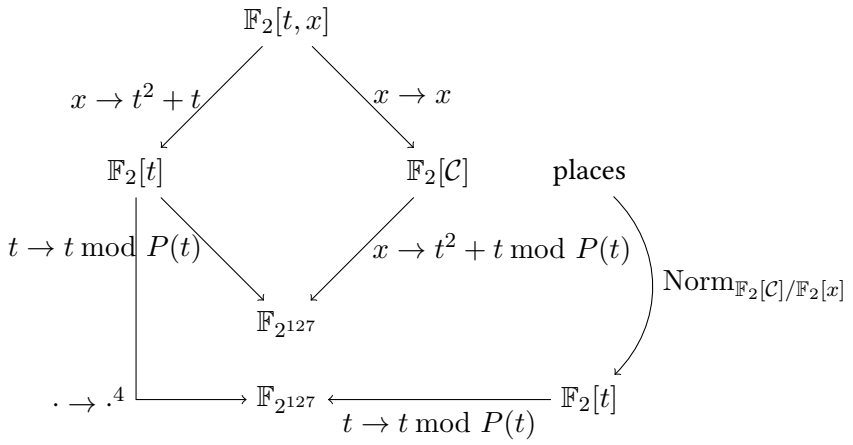


FIG. 3.10 : L'algorithme de Coppersmith pour $\mathbb{F}_{2^{127}}$, vu comme une instance de FFS

Pour faire rentrer l'algorithme de Coppersmith dans le cadre de l'algorithme FFS, la particularité tient à la forme de la courbe considérée. En effet, l'équation $f = x^4 - (t^2 + t)$ définit une courbe de genre 0, dont le corps de fonction est une extension purement inséparable (radicielle) de $\mathbb{F}_2(t)$. C'est cette dernière caractéristique qui « fait marcher » l'algorithme de Coppersmith.

Il est possible de transcrire le diagramme 3.3 en page 28 pour l'algorithme FFS en général, et pour l'algorithme de Coppersmith en particulier. Dans le diagramme 3.10, nous notons $\mathcal{C}$ la courbe d'équation $f = x^4 - (t^2 + t)$, et $P(t)$ le polynôme irréductible $t^{127} + t + 1$.

À l'instar de ce qui se passe pour FFS en général, une fonction de la forme $\phi = a(t) - xb(t)$ se décompose de deux façons différentes selon le « côté » du diagramme qui est observé. Côté rationnel, la décomposition est directement donnée par la factorisation du polynôme $\mathrm{Res}_x(\phi, g)$. Côté algébrique, le polynôme $\mathrm{Res}_x(\phi, f)$ donne une indication de la décomposition, mais celle-ci a effectivement lieu sur l'ensemble des places du corps de fonctions de la courbe $\mathcal{C}$. Ainsi, on a une écriture de la forme :

$$\operatorname{div} \phi = n_1\pi_1 + \cdots + n_s\pi_s - (\sum n_i)\infty.$$

où les π_i désignent les éléments de la base de facteurs, constituée de places affines sur la courbe. Dans l'exemple choisi, il y a une unique place rationnelle à l'infini sur $\mathcal{C}$ que nous notons ici ∞.

La particularité de la situation de l'algorithme de Coppersmith est que la courbe $\mathcal{C}$ n'est autre qu'un revêtement purement inséparable de $\mathbb{P}^1$. Aussi, pour chaque place affine π définie par deux fonctions $u(t)$ et $x + v(t)$, où $u(t)$ est un polynôme irréductible, le diviseur $4(\pi - (\deg \pi)\infty)$ est le diviseur principal $\operatorname{div} u(t)$. Ceci permet de définir une relation particulière à partir de la fonction ϕ^4. Adoptons les notations $\{p_i\}$ pour les polynômes de la base de facteurs rationnelle, et comme plus haut $\{\pi_i\}$ pour les places de la base de facteurs algébriques, avec pour chacune la contrainte de notation $4(\pi_i - (\deg \pi_i)\infty) = \operatorname{div} p_i$. On peut supposer les factorisations suivantes, pour une paire (a, b) doublement friable :

$$\begin{aligned}
&\text{côté rationnel :} && \mathrm{Res}_x(\phi, g) = \prod_i p_i^{m_i},\\
&\text{côté algébrique :} && \operatorname{div} \phi = (\sum_i n_i\pi_i) - (\sum n_i)\infty,\\
&\text{d'où} && \operatorname{div} \phi^4 = \sum_i \operatorname{div} p_i^{n_i}.
\end{aligned}$$

On en déduit, dans $\mathbb{F}_{2^{127}}$ (à une unité près, sans objet sur $\mathbb{F}_2$) :

$$\prod_i p_i^{4m_i} \equiv \prod_i p_i^{n_i}.$$

Un bénéfice évident qui provient de cette situation est la possibilité de confondre les bases de facteurs. En termes pratique, cela induit une réduction par un facteur 2 de la taille du système linéaire à résoudre.

Il est possible d'étendre ce schéma, y compris au cas où le polynôme g définit une courbe (non nécessairement de genre 0), et où le polynôme f définit un revêtement purement inséparable de cette dernière. Des essais préliminaires de constructions de ce type semblent toutefois indiquer qu'il est difficile d'obtenir une construction intéressante pour un calcul de logarithmes discret de cette façon.

3.5 Procédures de descente

Nous indiquons maintenant le fonctionnement des processus de *descente*. En reprenant la terminologie établie par la proposition 3.7, nous nous intéressons au problème suivant, que nous formulons dans le contexte NFS.

Problème 3.11. *On conserve les notations précédemment introduites. Soit $t \in G_\ell$. Déterminer $\hat{t} \in \mathbb{Q}$ tel que $\pi_r(\hat{t}) \equiv t$ dans G_ℓ, et $Q(x)$ une fraction rationnelle vérifiant :*

$$Q(m)\hat{t} \in \Sigma_r, \quad Q(\alpha) \in \Sigma_a.$$

La résolution de ce problème 3.11 nous fournit précisément les hypothèses de la proposition 3.7, et permet donc de calculer le logarithme discret d'un élément cible $t \in G_\ell$.

Une approche qui ne fonctionne pas, étant donné un générateur g, est de ramener le problème à celui du calcul du logarithme de $tg^r \bmod q$, pour r aléatoire, et g un élément dont le logarithme discret est connu (par exemple le générateur choisi). La réduction liant ces deux problèmes est évidente. Cette approche et ses variations sont adaptées aux algorithmes de complexité $L(1/2)$. On retrouve notamment cette approche dans le cas de l'algorithme d'Adleman [1], ou dans une version améliorée utilisant l'algorithme d'Euclide étendu [27]. La raison fondamentale interdisant son succès est que $tg^r \bmod q$ est de taille $q = L_q[1, 1]$, ou au mieux $\sqrt{q} = L_q[1, 1/2]$. En appliquant la proposition 2.6, il apparaît que la probabilité de friabilité d'un objet de cette taille, relativement à une base de facteurs en $L_q[1/3, \cdot]$ est en $L_q[2/3, \cdot]$, ce qui revient à dire que la complexité de l'algorithme est asymptotiquement prohibitive[4]. D'autre part, on remarque que cette approche qui ne fonctionne pas ne tire aucunement parti des *deux* bases de facteurs fournies par le contexte NFS.

3.5.1 Principe général

Historiquement, le premier procédé de descente adapté à un algorithme de complexité heuristique $L[1/3, \cdot]$ est dû à Coppersmith [50]. Le contexte de l'algorithme de Coppersmith est quelque peu particulier, ce qui rend la descente plus facile à énoncer. Toutefois, en généralisant un peu le point de vue, on constate que l'objet construit est exactement destiné à résoudre le problème 3.11.

[4] L'aspect *asymptotique* qui prévaut dans l'analyse des descentes peut parfois occulter des considérations pratiques. Un terme asymptotiquement invisible dans les estimations de taille peut avoir un impact considérable, positivement ou négativement, sur les probabilités de friabilité. Aussi est-il concevable, pour des petites tailles, d'utiliser une approche apparemment non valable asymptotiquement.

Pour l'essentiel, la formulation que nous adoptons suppose que nous nous plaçons dans le cas NFS et pas FFS, les modifications pour passer d'un cas à l'autre étant discutées lorsque cela s'avère pertinent. Une descente part d'un objectif (noté t dans le problème 3.11), de taille $L_q[1, \cdot]$. Elle comporte deux étapes essentielles, et distinctes.

- L'*amorce* de la descente consiste à déterminer un élément $\hat{t}$ qui agit comme un relèvement de t. Les notations que nous choisissons pour notre présentation installent ce relèvement comme élément de $\mathbb{Q}$, ce qui revient à dire que nous réalisons *l'amorce du côté rationnel*. En vérité, il est aussi possible de faire « fonctionner » le processus en réalisant l'amorce du côté algébrique (cependant la complexité peut en pâtir). La contrainte à laquelle est soumise l'amorce de la descente est une contrainte de friabilité de $\hat{t}$, relativement à une borne de friabilité que nous notons B_0, considérablement plus large que la borne de friabilité utilisée dans les étapes précédentes de l'algorithme. Cette borne est typiquement de l'ordre de $L_q[2/3, \cdot]$.

- Les *pas* de la descente amènent, chacun, à construire la fraction rationnelle qui intervient dans la proposition 3.7 (qui établit le « cahier des charges ») de la descente. Pas après pas, la fraction rationnelle $Q(x)$ est multipliée avec un nouveau terme. Les grandeurs $Q(m)\hat{t}$ et $Q(\alpha)$ sont soumises, lors de ces différents pas, à des contraintes successives de réduction de la borne de friabilité. Une suite de bornes est introduite, on passe ainsi d'une borne en $L_q[1/3 + \lambda, \cdot]$ à une borne en $L_q[1/3+\lambda/2, \cdot]$. La descente se conclut lorsque tous les facteurs qui apparaissent appartiennent aux bases de facteurs $\mathcal{F}_r$ et $\mathcal{F}_a$.

Pour énoncer les contraintes de friabilité intervenant dans ces étapes (amorce et pas), nous introduisons des structures similaires aux groupes Σ_r et Σ_a vus précédemment. Par une notation comme $\Sigma_r^{(0)}$, on entendra une structure construite de manière exactement similaire à Σ_r, mais relativement à une borne B_0 plus large que la borne initiale sur les nombres premiers rencontrés dans $\mathcal{F}_r$. Plus formellement, en reprenant les notations des définitions 2.2 et 2.3, on définit :

Notation 3.12.

$$\begin{aligned} B_0 &\geqq B, \\ \Sigma_r^{(0)} &= \{x \in \{\mathcal{F}_r(g, B_0)\text{-}\textit{unités}\}\}/\{x^\ell,\ x \in \{\mathcal{F}_r(g, B_0)\text{-}\textit{unités}\}\}, \\ \Sigma_a^{(0)} &= \{x \in \{\mathcal{F}_a(f, B_0)\text{-}\textit{unités}\}\}/\{x^\ell,\ x \in \{\mathcal{F}_a(f, B_0)\text{-}\textit{unités}\}\}. \end{aligned}$$

On définit de même les structures $\Sigma_r^{(i)}$ et $\Sigma_a^{(i)}$ relativement à une suite décroissante de bornes de friabilité $(B_i)_{i\geq 0}$.

Le point clé des procédures de descente, particulièrement des pas, est la possibilité de construire des paires (a, b) telles que $a - bm$, ou $a - b\alpha$, possède un facteur prescrit. Cette problématique est semblable à celle du « *special-q* » introduit par Davis et Holridge [57], aussi le terme de « *special-q descent* » est-il rencontré. Une étape de cette réduction est schématisée par l'algorithme 3.13. Il est important de vérifier, dans l'algorithme 3.13, que si on prend les notations suivantes, issues de la proposition 3.7 :

$$z^* = (Q(m)\hat{t}, Q(\alpha)), \quad \tilde{z}^* = (\tilde{Q}(m)\hat{t}, \tilde{Q}(\alpha)),$$

alors on a $\pi(z^*) = \pi(\tilde{z}^*) = t$. En particulier, le processus de descente repose de manière cruciale sur le fait d'avoir *deux* côtés à disposition, puisque la contribution du facteur $a - bx$ ajouté à la fraction rationnelle Q peut alors être *compensée*.

Entrée : Une fraction rationnelle $Q(x)$, et $\hat{t} \in \mathbb{Z}[m]$, tels que $\tilde{Q}(m)\hat{t} \in \Sigma_r^{(i)}$, $\tilde{Q}(\alpha) \in \Sigma_a^{(i)}$,
où $\Sigma_r^{(i)}$ et $\Sigma_a^{(i)}$ sont définis similairement à Σ_r et Σ_a, avec une borne B_i plus large sur les normes des idéaux qui interviennent.
Un idéal (ou nombre premier) $\mathfrak{p}$ de norme maximale apparaissant dans les factorisations de $Q(m)\hat{t}$ ou $Q(\alpha)$.
Une borne cible B_{i+1}.

Sortie : Une fraction rationnelle $\tilde{Q}(x)$ telle que : $\tilde{Q}(m)\hat{t} \in \Sigma_r^{(i+1)}$, $\tilde{Q}(\alpha) \in \Sigma_a^{(i+1)}$,
où $\Sigma_r^{(i+1)}$ et $\Sigma_a^{(i+1)}$ sont définis relativement à la borne $B_{i+1} < B_i$ (voir notation 3.12).

Exemple dans le cas où $\mathfrak{p}^\nu$ apparaît dans la factorisation de $Q(\alpha)$:

1. Déterminer $\mathcal{L}_\mathfrak{p}$ l'ensemble des paires (a, b) telles que $\mathfrak{p} \mid (a - b\alpha)$.
2. Soit $(a, b) \in \mathcal{L}_\mathfrak{p}$. Si $\tilde{Q}(x) = Q(x)(a - bx)^{-\nu}$ satisfait les conditions, retourner $\tilde{Q}$.

Algorithme 3.13 : Une étape de descente par spécial-q (principe).

Nous donnons quelques détails rapides sur diverses façons d'aborder le problème de l'amorce de la descente d'une part, et les pas élémentaires de celle-ci d'autre part. Ces éléments sont indissociables de l'analyse de complexité.

Le travail de description de la procédure de descente que nous menons ici est l'un des éléments importants des travaux décrits aux chapitres 5 et 10.

3.5.2 Amorce de la descente

Le premier pas d'une descente, étant donné un élément cible t, est un peu similaire à l'approche inopérante énoncée plus haut. On exploite l'équivalence entre le calcul des logarithmes discrets de t et de $tg^r \bmod q$. Nous omettons dans la suite le paramètre de randomisation r. Il est en effet acquis qu'à ce stade, pour l'amorce de la descente, il est possible de randomiser l'entrée, celle-ci n'étant soumise qu'à la borne $t < q = L_q[1, 1]$. En d'autres termes, t peut être vu comme une variable aléatoire. On cherche à obtenir une valeur de t qui soit friable.

Pour reprendre la terminologie de la proposition 3.7, l'étape d'amorce permet de trouver un nombre rationnel noté $\hat{t}$, vérifiant $\pi_r(\hat{t}) = t$. Nous requérons que ce nombre rationnel $\hat{t}$ soit friable relativement à la borne B_0, ce que nous écrivons $\hat{t} \in \Sigma_r^{(0)}$, en faisant appel aux notations $\Sigma_r^{(0)}$ et $\Sigma_a^{(0)}$ introduites précédemment page 41.

Le temps maximal alloué à la descente est de l'ordre de $L_q[1/3, \cdot]$. Aussi, nous devons déterminer pour quelle borne de friabilité B_0 il est possible en temps $L_q[1/3, \cdot]$ d'obtenir une valeur de t qui soit B_0-friable. D'après la proposition 2.6, une telle borne doit s'exprimer sous la forme $B_0 = L_q[2/3, c_0]$, pour une probabilité de friabilité de t en $u^{u(1+o(1))} = L_q[1/3, -\frac{1}{3c_0} + o(1)]$ où $u = \frac{\log q}{\log B_0}$.

Une amélioration simple de cette amorce, proposée en [27], consiste à utiliser l'algorithme d'Euclide étendu, qui pour un coût polynomial permet de récrire $t \equiv \frac{t_n}{t_d} \bmod q$, avec $t_n \approx t_d \approx \sqrt{q}$. La probabilité de friabilité est alors en $(u/2)^{u(1+o(1))} = L_q[1/3, -\frac{1}{3c_0} + o(1)]$. L'impact sur le terme dominant de la complexité n'apparaît pas, mais l'amélioration pratique est importante, puisqu'un facteur 2^u est épargné dans la complexité (la complexité qui apparaît à cette étape de l'algorithme repose sur l'heuristique selon laquelle t_n et t_d se comportent comme des entiers aléatoires bornés par $\sqrt{q}$).

Déterminer si un nombre de taille $L_q[1, 1]$ est $L_q[2/3, c_0]$ ne se fait pas sans effort. La factorisation est un problème difficile, ce que l'étude de NFS se saurait contredire ! Pour accomplir cette tâche,

on recourt à l'algorithme ECM [139, 220]. Celui-ci requiert un temps $L_p[1/2, \sqrt{2}]$ pour trouver un facteur p du nombre fourni en entrée. La combinaison des complexités, notamment en exploitant la proposition 2.5, permet d'aboutir à un choix optimal $c_0 = 3^{-1/3}$, soit une complexité de l'amorce en $L_q[1/3, 3^{1/3}]$. On trouve par ailleurs une amélioration de cette complexité via une stratégie d'*early abort* dans [14].

Dans le contexte de l'algorithme FFS ou de l'algorithme de Coppersmith, où des polynômes remplacent les entiers, le coût de l'amorce de la descente est radicalement moindre puisque la factorisation de polynômes est de complexité polynomiale. Seul le terme de probabilité compte dans l'évaluation de la complexité, ce qui modifie la donne. Ce point est analysé dans [T4].

Approche de crible

Une amélioration pratique proposée dans [116] consiste à cribler pour identifier des rationnels $\hat{t}$ pour lesquels certains facteurs sont présents d'emblée, réduisant de ce fait la partie restant à « friabiliser ». Notons (A_0, B_0) et (A_1, B_1) deux vecteurs courts du réseau dont deux vecteurs de base sont les lignes de

$$\begin{pmatrix} q & 0 \\ t & 1 \end{pmatrix}.$$

On a $A_0 - tB_0 \equiv A_1 - tB_1 \equiv 0 \bmod q$. Pour deux entiers k_0 et k_1, on a de même :

$$\frac{k_0 A_0(m) + k_1 A_1(m)}{k_0 B_0(m) + k_1 B_1(m)} \equiv t \bmod q.$$

Afin de ne se concentrer que sur les entiers donnant une probabilité de succès maximale, Joux et Lercier proposent dans [116] de cribler sur les entiers (k_0, k_1), dans un espace prescrit, pour ne garder que les candidats ayant de nombreux petits facteurs. En termes de complexité asymptotique, cette approche n'a pas été analysée, mais il est peu vraisemblable qu'un effet visible soit obtenu. En effet, cribler pendant un temps $L[1/3, \cdot]$, donc relativement à une base de facteurs de taille $L[1/3, \cdot]$, et dans un espace de taille $L[1/3, \cdot]$, a peu de chances d'isoler parmi un ensemble de nombres candidats de taille $L[1, \cdot]$, une partie friable de taille dépassant $L[2/3, \cdot]$, et laissant donc un cofacteur de taille $L[1, \cdot]$. En outre les propriétés de friabilité des cofacteurs rencontrés dans ce processus ne sont pas évidentes à analyser. Cette approche de crible est donc assurément valide en pratique, mais probablement pas d'une efficacité visible en termes de complexité asymptotique.

3.5.3 Pas de la descente

Le produit de l'amorce de la descente est un rationnel $\hat{t}$ exactement conforme aux notations et conditions du problème 3.11, à la nuance près que la fraction rationnelle n'apparaît pas encore, et que la condition de friabilité n'est satisfaite que pour la borne lâche B_0. On peut ainsi écrire, à l'issue de l'amorce :

$$\hat{t} \in \Sigma_r^{(0)}, \quad 1 \in \Sigma_a^{(0)}.$$

L'objectif des pas de la descente est de fabriquer la fraction rationnelle $Q(x)$ permettant de satisfaire pleinement les conditions énoncées par la proposition 3.7. Le principe de ces pas est indiqué par l'algorithme 3.13 page 42. Nous détaillons brièvement l'une des façons d'aborder ce problème, et renvoyons à la section suivante 3.5.4 sur le positionnement de cette approche parmi les autres approches existantes. L'approche que nous présentons est celle que nous avons développée dans [T15] (voir aussi chapitre 10).

En adoptant les notations de l'algorithme 3.13, il existe un nombre fini d'idéaux premiers ou de nombres premiers intervenant dans les factorisations de $Q(m)\hat{t}$ et $Q(\alpha)$, et dont la norme est comprise entre B_{t+1} et B_t. Quitte à répéter les étapes qui suivent plusieurs fois, on peut supposer qu'un seul idéal est ainsi à considérer. Dans le but d'abréger l'exposition, nous faisons la supposition que cet idéal apparaît du côté algébrique, et qu'il s'écrit $\mathfrak{p} = \langle p, \alpha - r\rangle$. Nous supposons en outre que $p = \mathrm{Norm}(\mathfrak{p}) = B_t$. L'approche est identique pour le côté rationnel.

L'ensemble des paires (a, b) qui constitue l'ensemble $\mathcal{L}_\mathfrak{p}$ apparaissant dans l'algorithme 3.13 est semblable à celui apparaissant dans la technique du *lattice sieving* [174], à savoir les points du réseau engendré par la matrice

$$\begin{pmatrix} p & 0 \\ r & 1 \end{pmatrix}.$$

L'algorithme 3.13 n'est toutefois qu'un *principe*, et une approche plus exacte nécessite de recourir à un ensemble $\mathcal{L}_\mathfrak{p}$ légèrement différent. En effet, on définit $\mathcal{L}_\mathfrak{p}$ comme étant l'ensemble des idéaux principaux de $\mathcal{O}_K$ engendrés par un élément de la forme $u(\alpha) = u_0 + u_1\alpha + \cdots + u_{k-1}\alpha^{k-1}$, pour un degré $k-1$ à déterminer. Les coordonnées de tels générateurs sont les vecteurs du réseau suivant.

$$\begin{pmatrix} p & & & \\ -r & 1 & & \\ & \ddots & \ddots & \\ & & -r & 1 \end{pmatrix}.$$

Ce réseau a pour norme p. Pour $u(x)$ de degré $k-1$, le réseau constitué a pour dimension k, et les vecteurs courts de ce réseau produits par l'algorithme LLL ont des coefficients de taille approximativement

$$\|u\| \approx p^{\frac{1}{k}}.$$

Considérons une base produite par LLL, qu'on note $(u^{(0)}, \ldots, u^{(k-1)})$. Les facteurs en $2^{O(k)}$ dans la taille des vecteurs produits n'importent pas dans notre analyse et peuvent être négligés. Aussi supposons-nous que l'estimation donnée plus haut est valide pour chacun des vecteurs. Nous considérons des combinaisons linéaires à coefficients entiers des différents vecteurs de base $u^{(j)}$:

$$u(x) = a_0 u^{(0)}(x) + \cdots + a_{k-1}u^{(k-1)}(x).$$

L'ensemble $\mathcal{L}_\mathfrak{p}$ que nous prenons en compte est celui constitué par les combinaisons linéaires à coefficients bornés par une borne A :

$$\mathcal{L}_\mathfrak{p} = \{u(x) = a_0 u^{(0)}(x) + \cdots + a_{k-1}u^{(k-1)}(x),\ |a_i| \leq A\}.$$

Nous recherchons, comme mentionné dans l'algorithme 3.13, un $u(x) \in \mathcal{L}_\mathfrak{p}$ qui soit B_{t+1}-friable.

Afin de mesurer la possibilité de succès d'une telle entreprise, une étude des comportements asymptotiques des différentes bornes est nécessaire. Toutefois, nous souhaitons avant cela remarquer un point de détail. La factorisation d'un idéal engendré par un élément $u(\alpha)$, de degré k, peut faire intervenir des idéaux premiers dont le degré d'inertie n'est pas égal à 1. Aussi, la discussion en page 15 indiquant que la restriction des bases de facteurs aux idéaux premiers de degré d'inertie égal à 1 n'est *pas* contraignante pour la factorisation de $(a - b\alpha)\mathcal{O}_K$, n'a pas cours ici. Il se trouve, néanmoins, que le nombre de ces idéaux est considérablement inférieur. En effet, s'il y a asymptotiquement environ $\pi(x)$ idéaux de degré d'inertie 1 dont la norme est inférieure à une borne x, le nombre d'idéaux de

degré d'inertie 2 ou plus est de l'ordre de $\pi(\sqrt{x})$. La non-considération des factorisations faisant intervenir de tels idéaux n'est donc pas un problème en pratique.

Supposons qu'on a $p = B_t = L_q[1/3+s, \cdot]$ avec $s \leq 1/3$. Il nous importe de pouvoir déterminer quelle borne cible B_{t+1} constitue un objectif réaliste pouvant être atteint pour un effort $L_q[1/3, \cdot]$. Aussi fixons-nous pour A une expression telle que $A^k = L_q[1/3, \cdot]$ (ceci représente le nombre total de combinaisons considérées). Les ordres de grandeur pour $\mathrm{Res}(u, f)$ et $\mathrm{Res}(u, g)$ sont :

$$\begin{aligned}\mathrm{Res}(u, f) &\approx \|u\|^{\deg f}\|f\|^{\deg u} = (Ap^{1/k})^d q^{(k-1)/d},\\ \mathrm{Res}(u, g) &\approx \|u\|^{\deg g}\|g\|^{\deg u} = (Ap^{1/k}) q^{(k-1)/d}.\end{aligned}$$

La norme algébrique, dans l'expression qui précède, est plus problématique que la norme rationnelle. Notre attention se focalise donc sur la norme algébrique. Si l'on se contente de prendre en compte le terme principal de la complexité, alors $A^k p$ est un terme en $L_q[1/3 + s, \cdot]$. Supposons qu'on se limite au cas $k = 2$. Alors la norme algébrique, selon l'expression ci-dessus, est en $L_q[2/3 + s, \cdot]$. En temps $L_q[1/3, \cdot]$, il semble alors ardu de satisfaire une condition de friabilité pour une borne autre que $L_q[1/3+s, \cdot]$, qui est la borne d'origine. Pour cette raison, nous considérons plutôt k de la forme $k = \log_{\log q} L_q[s/2, \cdot]$. L'expression de la norme algébrique est alors (en vertu de la propriété 2.5) :

$$\begin{aligned}\mathrm{Res}(u, f) &\approx (Ap^k)^{d/k}(q^{1/d})^{k-1},\\ &= L_q[1/3 + s + 1/3 - s/2, \cdot] \times L_q[1 - 1/3 + s/2],\\ &= L_q[2/3 + s/2, \cdot].\end{aligned}$$

Il apparaît alors que parmi l'espace de recherche de taille A^k qu'on a choisi, identifier un candidat qui soit $L_q[1/3+s/2, \cdot]$-friable est possible. Dans cette estimation, nous avons employé l'heuristique usuelle 2.7.

Nous ne donnons pas ici le détail des termes de second ordre, cette analyse est faite dans notre article [T15] en collaboration avec Enge et Gaudry (bien que dans un contexte différent, les éléments d'analyse sont identiques). Parmi les termes qui n'apparaissent même pas au second ordre, on compte le nombre d'étapes de la descente. En effet, ce nombre d'étapes est en $L_q[1/3, o(1)]$ et n'impacte donc pas la complexité. Cette discussion est détaillée en particulier dans [50, 49, T15]. La complexité totale du processus de descente, dans le cas de NFS, est ainsi dominée par l'étape d'amorce, soit en $L_q[1/3, 3^{1/3}]$ (complexité heuristique).

3.5.4 Autres approches pour la descente

D'autres processus de descente ont été proposés dans la littérature. La proposition initiale dans cette thématique est l'algorithme de Coppersmith [50]. Ce dernier met en place la structure de la descente telle que nous la présentons, mais bénéficie d'un avantage particulier. Dans le cadre de l'algorithme de Coppersmith, il est possible de travailler dans une tour[5] de corps de fonctions compatibles les uns avec les autres, puisque les courbes considérées sont des revêtements purement inséparables les unes des autres, comme entrevu en 3.4.6.

Les tentatives d'adaptation directe de la descente telle que proposée par Coppersmith se sont heurtées au problème suivant. Pour considérer l'ensemble des paires (a, b) satisfaisant une condition de divisibilité modulo un idéal premier $\mathfrak{p}$ (côté rationnel ou algébrique) de norme $L_q[2/3, \cdot]$, les valeurs (a, b) sont immanquablement de norme $L_q[2/3, \cdot]$. Dès lors, le degré du corps de nombres choisi

[5] En pratique, le degré de ces corps de fonctions est au plus le degré k considéré dans les paramètres de l'algorithme Coppersmith. Ceci revient à dire que l'introduction de cette tour est une considération assez virtuelle.

initialement paraît trop gros, puisqu'il porte cette norme à $L[1, \cdot]$ (ces considérations se retranscrivent dans le contexte FFS/Coppersmith en remplaçant la norme par l'expression $p^{\text{degré}}$). Comme nous l'avons vu, ce problème est mentionné dans la présentation que nous faisons plus haut, et il est possible de le contourner en considérant des fonctions de degré plus grand que 1.

Historiquement, l'approche que nous évoquons n'a pas été proposée. L'algorithme de Coppersmith profite de sa situation particulière pour utiliser des courbes de degré plus petit que le degré initial (mais toujours des puissances de 2). Les premiers articles proposant d'utiliser le crible algébrique pour calculer des logarithmes discret furent [95, 181, 182]. Dans ces deux articles, la difficulté de la descente est globalement contournée, à grands frais, puisqu'à l'issue de l'amorce de la descente un nouveau corps de nombres est considéré pour chacun des termes apparaissant dans la factorisation.

Une première forme de descente a été proposée par Adleman et Huang dans [6] dans le contexte FFS, en reprenant autant que faire se peut les propriétés de la descente de l'algorithme de Coppersmith. Il en résulte une forte contrainte sur la caractéristique p, puisque l'algorithme proposé impose $p^6 \leq n$. Toujours dans le contexte FFS, mais sans cette contrainte, Schirokauer [183] propose une descente où apparaît une succession de bornes de la forme $L[1/3 + 1/(3 \cdot 2^j), \cdot]$. Cependant, un nouveau corps de fonction est considéré pour chacune de ces bornes, et une étape d'algèbre linéaire propre a chaque pas s'avère nécessaire.

À partir de 2002 environ, il semble que la possibilité d'adapter une descente est acceptée de plusieurs auteurs, mais peu présente dans les articles. Une raison expliquant cela est le fait que des approches plus directes, à ce stade, ne rencontrent pas d'obstruction majeure. Par exemple dans [116], le calcul des logarithmes individuels évite l'étape de *descente*, et emploie l'approche de crible mentionnée plus haut. Bien que praticable et efficace, cette approche ne résout pas la question de complexité, puisqu'une tentative d'analyse aboutit à une complexité en $L[1/2, \cdot]$. Semaev [194] propose dans un cas particulier une descente qui fonctionne, et met en évidence une décroissance exponentielle des bornes de friabilité. Implicitement, ce fonctionnement est sous-jacent à la descente qui est évoquée dans [115]. Cette ligne d'analyse a ensuite été reprise par [49].

En résumé, l'approche de *descente* fonctionne, et il y a diverses façon de la formuler. La formulation que nous avons proposée, avec des fonctions de degré dépassant 1, a l'avantage de fournir une décroissance des bornes de friabilité plus rapide si l'on s'en tient à l'asymptotique. Toutefois, dans la pratique, il se peut que la différence soit particulièrement ténue. Il est tout à fait viable, d'un point de vue pratique, de se cantonner à une approche de crible telle que proposée plus haut, au moins pour une amorce « étendue », pour ensuite utiliser des étapes de descente avec des fonctions de degré 1.

Chapitre 4

Racine carrée dans NFS

Les travaux présentés dans cette section relèvent d'une partie de l'article [T8], et d'un algorithme évoqué dans [T13], et implémenté dans le logiciel `cado-nfs` [90]. Ce chapitre a été traduit sous la forme de l'article invité [T18].

Nous nous intéressons dans cette section au problème du calcul de la racine carrée algébrique dans NFS (pour la factorisation). Le problème est donné comme suit. Pour simplifier les notations, on suppose dans toute cette section que $f(x)$ est un polynôme unitaire, ce qui implique en particulier que α est un entier algébrique. Soit $\mathcal{S}$ un ensemble de paires (a, b), construit de sorte à garantir que $S(\alpha) = \prod_{(a,b)\in\mathcal{S}}(a - b\alpha)$ est un carré dans $\mathbb{Z}[\alpha]$. Un tel ensemble est précisément le résultat des étapes d'algèbre linéaire et de calcul des caractères dans NFS. L'objectif est de déterminer un polynôme à coefficients entiers $T(x)$ tel que $T(\alpha)^2 = f'(\alpha)^2 S(\alpha)$. L'introduction de $f'(\alpha)^2$ vise uniquement à contourner l'une des obstructions théoriques listées en 2.1.3. Pour alléger les notations, on omet le facteur $f'(\alpha)^2$ dans la suite de ce chapitre.

Le corps de nombres défini par le polynôme f est noté K, et son anneau des entiers $\mathcal{O}_K$. On note en outre $d = [K : \mathbb{Q}]$. Deux points sont importants pour certains des algorithmes développés ici. Tout d'abord, avec chacune des paires $(a, b) \in \mathcal{S}$, nous disposons de l'information sur la factorisation de l'idéal $(a - b\alpha)\mathcal{O}_K$ (ou au minimum, de l'information sur la factorisation de sa norme). D'autre part, il est important de noter que le calcul de $T(\alpha)$ n'est pas en soi intéressant pour NFS. Dans le contexte NFS, la grandeur qui est voulue est $T(m) \bmod N$. Dans certains cas, on verra qu'il est possible de calculer cette quantité sans calculer explicitement $T(\alpha)$, ce qui induit un gain substantiel. Le fait de passer par le calcul de $T(\alpha)$ comme un point obligé est plutôt à comprendre comme un inconvénient dans les algorithmes ci-dessous, car cela interdit un tel gain.

Les complexités des différents algorithmes présentés sont données en fonction de la taille en bits de l'entrée. Nous supposons que l'entrée $\mathcal{S}$ occupe n bits. Ainsi chaque coefficient de $S(\alpha)$ occupe également à peu près n bits, la taille totale de $S(\alpha)$ étant dn bits. Ceci induit des coefficients pour $T(\alpha)$ de l'ordre de $n/2$ bits chacun. Le corps de nombres considéré, dans ce chapitre, est considéré constant. Aussi la dépendance est les paramètres de celui-ci sera quasi systématiquement omise, exception faite de certains coûts arithmétiques relatifs au degré d du corps de nombres. Comme il est d'usage, la notation $\mathsf{M}(k)$ désigne dans ce chapitre le coût de la multiplication de deux entiers de k bits.

4.1 Approches possibles

Plusieurs approches existent. Nous les donnons ici. Toutefois, comme nous allons le remarquer, toutes ne sont pas applicables dans tous les cas.

4.1.1 L'approche par relèvement

Une approche dite p-adique, ou par relèvement, peut s'appliquer lorsqu'il existe un nombre premier *inerte*. Cela n'est possible que lorsque le groupe de Galois de f possède un élément d'ordre d.

D'une manière générique, le groupe de Galois de f dans le cas GNFS est égal à $\mathfrak{S}_d$. On est donc certain de pouvoir rencontrer un nombre premier inerte, avec probabilité $\frac{1}{d!}$. Ce n'est toutefois pas le cas en général pour SNFS. En effet, il est possible de rencontrer par exemple un polynôme f de degré 4 dont le groupe de Galois est $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$. Ce cas est suffisamment peu exceptionnel pour être significatif pour SNFS, de sorte qu'on ne se limitera pas à l'approche que nous détaillons ici.

Sous cette hypothèse, soit p un nombre premier inerte, i.e. tel que $p\mathcal{O}_K$ est un idéal premier. L'extension $K_p = \mathbb{Q}_p[x]/f(x)$ est donc une extension non ramifiée de $\mathbb{Q}_p$. Notons α_p une racine de f dans K_p. On dispose d'un morphisme d'anneaux injectif de $\mathbb{Z}[\alpha]$ dans $\mathbb{Z}_p[\alpha_p]$. L'objectif de l'algorithme est d'utiliser cette injection pour reconnaître la racine carrée recherchée.

La première étape de l'algorithme consiste à déterminer une racine carrée à petite précision de $S(\alpha_p)$. Notons $\mathbb{F}_p(\beta) = \mathbb{F}_{p^d}$ le corps résiduel de K_p, la projection $K_p \to \mathbb{F}_{p^d}$ étant donnée par $\alpha_p \mapsto \beta$. On souhaite ainsi déterminer une racine carrée de $S(\beta) \in \mathbb{F}_{p^d}$. Comme p est inerte, et les (a, b) premiers entre eux, nous avons $S(\beta) \neq 0$. Notons $T(x)$ un polynôme à coefficients entiers satisfaisant $T(\beta)^2 = S(\beta)$. On a par construction :

$$T(\alpha_p)^2 \in S(\alpha_p) + p\mathcal{O}_K.$$

On peut alors obtenir $T(x)$ par raffinements successifs en utilisant un relèvement de Newton. On peut trouver plus de détails par exemple dans [30, 33]. En particulier, il est algorithmiquement avantageux de calculer en premier lieu l'inverse de la racine carrée, l'itération du relèvement de Newton s'en trouvant simplifiée.

Borne sur les coefficients du résultat Il est possible de poursuivre le relèvement de $T(x)$ jusqu'à « constater » la présence de coefficients entiers. Cela n'est toutefois pas satisfaisant, dans la mesure où une borne effective sur les coefficients du résultat est accessible.

Pour un corps de nombres K, on note $\sigma_1, \ldots, \sigma_r$ les plongements réels, et $\sigma_{r+1}, \ldots, \sigma_{r+s}$ les plongements complexes non conjugués. On note en outre $\sigma_{r+s+k} = \overline{\sigma_{r+k}}$ pour $k = 1, \ldots, s$. Soit $U = \sum_{i=0}^{d-1} u_i \alpha^i \in K$. Les coefficients u_i de l'expression de U sont liés aux valeurs des plongements, puisque ces derniers sont une expression polynomiale en les u_i. On a en effet :

$$(\sigma_1(U), \ldots, \sigma_d(U)) = (u_0, \ldots, u_{d-1}) \times V\,(\sigma_1(\alpha), \ldots, \sigma_d(\alpha))\,.$$

La matrice intervenant dans l'expression ci-dessus est une matrice de Vandermonde. Il est possible de renverser cette expression pour obtenir une borne sur les $|u_i|$ à partir d'une borne sur les $|\sigma_i(U)|$. En effet, il suffit pour cela de précalculer l'inverse de la matrice de Vandermonde à une précision modeste, de sorte à pouvoir dériver une borne pas trop grossière.

En pratique, un tel mécanisme est utilisé pour borner les coefficients d'un élément $U \in \mathbb{Z}[\alpha]$ à partir des logarithmes des plongements, les $\log|\sigma_i(U)|$ (les logarithmes permettant d'éviter des dépassements d'exposants). C'est précisément ce qui peut être fait dans le contexte du calcul de la racine carrée dans NFS. On connaît $S(\alpha)$, et on recherche $T(\alpha) \in \mathbb{Z}[\alpha]$ satisfaisant $T(\alpha)^2 = S(\alpha)$. Dès lors, on a $\log|\sigma_i(T(\alpha))| = \frac{1}{2}\log|\sigma_i(S(\alpha))|$. En outre, $S(\alpha)$ est connu sous la forme d'un produit. Il est donc pratique de calculer les $\log|\sigma_i(S(\alpha))|$ selon l'expression suivante :

$$\log|\sigma_i(S(\alpha))| = \sum_{(a,b)\in\mathcal{S}} \log|\sigma_i(a - b\alpha)|.$$

Le calcul des bornes sur les coefficients t_i de $T(\alpha) = \sum_i t_i \alpha^i$, et donc la détermination du nombre d'étapes de relèvement de Newton, s'effectue donc comme suit. En premier lieu, on calcule les racines

complexes de f, puis les $\log|\sigma_i(S(\alpha))|$ comme on vient de l'indiquer. On calcule ensuite l'inverse, sans effort excessif sur la précision, de la matrice $V(\sigma_1(\alpha),\ldots,\sigma_d(\alpha))$ (dont le déterminant est $\sqrt{\mathrm{disc}(f)}$). Ce calcul doit être mené avec un arrondi vers $+\infty$. Avec ces données, on déduit des bornes sur les $|t_i|$. Supposons par exemple qu'on obtienne une borne M. Alors on se limite au calcul du relèvement de Newton jusqu'à la précision $k = \lceil \log_p M \rceil$. Ceci implique en particulier qu'il est possible, dans les calculs faisant intervenir $S(\alpha)$, de réduire les coefficients de S modulo p^k.

Complexité Le calcul de racine carrée par la méthode de Newton est de complexité quasi-linéaire [30, 33]. Ceci se vérifie bien sûr aussi dans le cas qui nous intéresse. La complexité du calcul est de la forme $O(d^2\mathsf{M}(n))$, où on a noté d^2 le coût pour multiplier deux polynômes de degré au plus $d-1$ (sujet bien sûr à améliorations triviales, dans la mesure de leur pertinence pour une valeur de d qui n'est après tout qu'un degré dans NFS). On remarque qu'asymptotiquement, cette complexité est dominée par celle du précalcul consistant à calculer $S(\alpha)$ lui-même, qui nécessite $O(d^2\mathsf{M}(n)\log n)$ en utilisant un arbre de sous-produits.

4.1.2 L'algorithme de Couveignes

Une approche permettant de s'abstraire des besoins en mémoire liés au calcul du relèvement de Newton est l'algorithme de Couveignes [54], qui utilise le théorème des restes chinois. Cette méthode s'applique uniquement dans le cas où, comme précédemment, il existe un nombre premier inerte, et où en outre le degré du corps de nombres est impair.

Le principe de départ est le suivant. On souhaite calculer $T(\alpha)$ par restes chinois, i.e. calculer $T(x) \bmod p_i$ pour de nombreux nombres premiers p_i, et reconstruire le résultat. On suppose pour ce faire que le produit $P = \prod_i p_i$ est plus grand que la borne M sur les coefficients de T. Supposons alors connue la valeur $T_i(x) = T(x) \bmod p_i$, et notons q_i le plus petit entier positif vérifiant $q_i \bmod p_j = \delta_{i,j}$ (en d'autres termes, $q_i = s\cdot(s^{-1} \bmod p_i)$, où $s = \prod_{j\neq i} p_j$). On a dans ce cas[1]

$$T(x) = \sum_i q_i T_i(x) \bmod P. \tag{4.1}$$

L'obstacle rencontré par cette approche a priori simple est lié au choix des racines. La supposition est que les nombres premiers p_i sont nombreux. Or le calcul de $S(x) \bmod p_i$ ne permet de déterminer $T_i(x)$ qu'à un signe près. La grandeur calculée modulo p_i est ainsi $T'_i(x) = \pm T_i(x)$. Le nombre de recombinaisons possibles dans l'expression $T(x) = \sum_i \pm q_i T'_i(x)$ est grand.

Pour lever l'indétermination, l'algorithme de Couveignes exploite le fait que le degré du corps de nombre est supposé impair. Sous cette hypothèse, on a

$$\mathrm{Norm}_{K/\mathbb{Q}}(-\zeta) = -\mathrm{Norm}_{K/\mathbb{Q}}(\zeta)$$

pour $\zeta \in K$. Il se trouve que dans le cas particulier du calcul de racine carrée pour NFS, il est possible de calculer assez aisément $|\mathrm{Norm}_{K/\mathbb{Q}}(T(\alpha))|$. En effet, on a :

$$S(\alpha)\mathcal{O}_K = \prod_{\mathfrak{p}\in\mathcal{F}_a} \mathfrak{p}^{2e_\mathfrak{p}} \text{ avec } e_i \in \mathbb{Z},$$
$$|\mathrm{Norm}_{K/\mathbb{Q}}(T(\alpha))| = \prod_{\mathfrak{p}\in\mathcal{F}_a} \mathrm{Norm}(\mathfrak{p})^{e_\mathfrak{p}},$$

[1] On remarque que puisque $M < P$, on a bien ici une égalité et pas seulement une relation de congruence, pourvu qu'on définisse l'opérateur binaire $\bmod$ comme étant centré en 0. Nous n'épilogons pas sur ce détail.

$$\mathrm{Norm}_{K/\mathbb{Q}}(T(\alpha)) = \pm \prod_{\mathfrak{p}\in\mathcal{F}_a} \mathrm{Norm}(\mathfrak{p})^{e_\mathfrak{p}}.$$

On fait *un* choix arbitraire de racine carrée en choisissant le signe dans le second membre de la dernière expression ci-dessus. Notons ν la norme ainsi calculée. On a alors $\mathrm{Norm}_{K/\mathbb{Q}}(T(\alpha)) = \nu$, et $\mathrm{Norm}_{K/\mathbb{Q}}(-T(\alpha)) = -\nu$. Ceci implique qu'étant donné une valeur calculée $T_i'(\alpha)$, on peut distinguer si $T_i'(\alpha)$ est $T_i(\alpha)$ ou $-T_i(\alpha)$ en comparant $\mathrm{Norm}_{\mathbb{F}_{p_i^d}/\mathbb{F}_{p_i}}(T_i'(\alpha))$ avec $\nu \bmod p_i$.

L'indétermination étant levée, il est possible d'utiliser l'expression 4.1. Notons toutefois que dans la perspective du crible algébrique, seule la quantité $T(m) \bmod N$ est utile. Aussi, il n'est pas nécessaire de reconstituer $T(x)$. Pour obtenir $T(m) \bmod N$, il est suffisant de procéder en deux étapes. D'abord, déterminer le polynômes à coefficients entiers $r(x)$ tel que l'égalité 4.1 s'écrit plus précisément

$$T(x) = \sum_i q_i T_i(x) - Pr(x).$$

Les coefficients de $r(x)$ sont de petits entiers, aussi est-il aisé de calculer ce polynôme en arithmétique à petite précision. Dans un second temps, une fois connu $r(x)$, on déduit :

$$T(m) \bmod N \equiv \sum_i (q_i \bmod N)(T_i(m) \bmod N) - (P \bmod N)(r(m) \bmod N).$$

Complexité Supposons que des nombres premiers de taille fixe sont considérés : on pose[2] $\log p_i \leq \lambda$. Le nombre de nombres premiers qui doivent être considérés pour pouvoir réaliser la reconstruction est $O(n/\lambda)$. Pour chaque nombre premier, l'ensemble $\mathcal{S}$ est lu afin de calculer la norme $\nu \bmod p_i$. Cette étape détermine la complexité : en effet, son coût total est $O(n^2)$. En outre, puisqu'il s'agit d'une opération d'entrée-sortie, il est peu valide d'invoquer la possibilité de parallélisation (l'entrée se lit généralement d'une seule source).

Un compromis temps-mémoire peut être envisagé pour cet algorithme. On peut lire l'ensemble $\mathcal{S}$ en $\sqrt{n}$ blocs de taille $\sqrt{n}$, un sous-produit de taille $\sqrt{n}$ étant calculé à chaque lecture, puis réduit modulo chacun des p_i. Ainsi, la complexité est réduite par exemple à $O(n^{3/2})$, pour une mémoire en $O(n^{1/2})$.

4.1.3 L'algorithme de Montgomery

L'algorithme de Montgomery [157, 159] pour la racine carrée est l'algorithme le plus spécialisé au cas particulier de la racine carrée dans NFS. Il diffère radicalement des deux algorithmes précédemment exposés, ainsi que de celui décrit plus loin en 4.2. Il est en revanche particulièrement important dans le travail développé en 5.2 et 5.3, et est pour cette raison détaillé ici.

Une version de l'algorithme de Montgomery est aussi décrite dans [164].

L'algorithme de Montgomery fonctionne en toute généralité, et exploite de manière cruciale le fait que le nombre algébrique $S(\alpha)$ dont on recherche la racine carrée est le générateur d'un idéal friable, dont la factorisation complète est accessible. Parmi les différents algorithmes pour la racine carrée, cet algorithme est, en outre, celui ayant la meilleure « dépendance en 2 ». En effet, il peut être utilisé pour le calcul de racine λ-ème dans un corps de nombres pour un exposant λ autre que 2, pour un surcoût polynomial en $\log \lambda$ (toujours sous l'hypothèse, forte, que la factorisation complète de l'idéal $S(\alpha)\mathcal{O}_K$ est connue). Cette propriété est utilisée en 5.3. Pour cette raison, nous nous plaçons ici dans le contexte un peu plus général du calcul de racines algébriques λ-èmes.

[2] Le fait qu'une conséquence de cette contrainte nous limite à un nombre fini de nombres premiers est totalement hors de propos pour l'analyse.

Comme mentionné en 2.1.2, les calculs ne sont pas menés avec l'ordre maximal $\mathcal{O}_K$, mais avec une approximation de celui-ci, qui peut être calculée avec l'algorithme Round-2. Les énoncés qui suivent sont formulés en faisant apparaître l'ordre maximal $\mathcal{O}_K$, mais restent valides avec cette approximation $\mathcal{O}$.

Réduction itérative Le principe de fonctionnement de l'algorithme est le suivant. On maintient une expression de la forme :

$$S(\alpha)(\gamma_0^{\epsilon_0} \dots \gamma_{k-1}^{\epsilon_{k-1}})^{-\lambda}\mathcal{O}_K = \prod_{\mathfrak{p}\in\mathcal{F}_a} \mathfrak{p}^{\lambda\cdot e_{\mathfrak{p}}^{(k)}} \tag{4.2}$$

où k indique l'étape de calcul (initialement $k = 0$), et γ_i est un nombre algébrique calculé à l'étape i. La notation ϵ_i désigne un signe (choix entre numérateur et dénominateur). Les exposants $e_{\mathfrak{p}}^{(k)}$ peuvent devenir négatifs au cours de l'algorithme. L'objectif *in fine* de ces étapes de réduction est de réduire le second membre à un nombre algébrique de petits coefficients. Non seulement on souhaite obtenir une factorisation en idéaux triviale ou ne faisant intervenir qu'un nombre restreint d'idéaux, mais on souhaite aussi œuvrer pour obtenir une contribution minimale du groupe des unités.

Notons $S_k(\alpha) = S(\alpha)(\gamma_0^{\epsilon_0} \dots \gamma_{k-1}^{\epsilon_{k-1}})^{-\lambda}$ (ainsi, on a $S_0(\alpha) = S(\alpha)$). À l'étape k, on choisit un sous-ensemble des idéaux apparaissant dans le second membre de l'équation 4.2. Cette factorisation se décompose naturellement en un numérateur (exposants positifs) et un dénominateur (exposants négatifs). Le sous-ensemble choisi est de la forme $I_k = \mathfrak{p}_1 \dots \mathfrak{p}_{n_k}$, divisant soit le numérateur, soit le dénominateur (les idéaux $\mathfrak{p}_i$ ne sont pas nécessairement tous distincts, il est licite de choisir un idéal $\mathfrak{p}$ plusieurs fois). L'objectif est de réduire la contribution des idéaux $\mathfrak{p}_1, \dots, \mathfrak{p}_{n_k}$ dans la factorisation de $S_k(\alpha)\mathcal{O}_K$.

Une base LLL-réduite de l'idéal I_k est calculée. De la sorte, on obtient des entiers algébriques appartenant à I_k. Un tel entier algébrique v vérifie $\mathrm{Norm}_{K/\mathbb{Q}}(I_k) \mid \mathrm{Norm}_{K/\mathbb{Q}}(v)$, et même un encadrement

$$m(v) \overset{\text{déf}}{=} \left|\frac{\mathrm{Norm}_{K/\mathbb{Q}}(v)}{\mathrm{Norm}_{K/\mathbb{Q}}(I_k)}\right| \leq C_K,$$

où la constante C_K est calculable (elle ne dépend que de K). Supposons donné un tel élément v, et plaçons-nous par exemple dans le cas où I_k est un facteur du numérateur de $S_k(\alpha)\mathcal{O}_K$ (le cas du dénominateur est exactement similaire). On a alors :

$$\mathrm{Norm}(S_k(\alpha)v^{-\lambda}) = \mathrm{Norm}(S_k(\alpha))\,\mathrm{Norm}(I_k)^{-\lambda}m(v)^{-\lambda}.$$

Il apparaît que la norme du numérateur, dans la factorisation de $S_k(\alpha)v^{-\lambda}\mathcal{O}_K$, est réduite de $\mathrm{Norm}(I_k)^{\lambda}$ par rapport à $S_k(\alpha)\mathcal{O}_K$, tandis que la norme du dénominateur croît de $m(v)^{\lambda} \leq C_k^{\lambda}$. Dès lors que I_k est choisi de norme significativement plus grande que la constante C_K, on obtient ainsi une réduction stricte de la norme de l'expression.

La réduction décrite doit être « guidée » afin de ne pas aboutir à une unité $S_k(\alpha)$ qui soit difficile à manipuler. Pour ce faire, il est suffisant à chaque étape de choisir un élément $v \in I_k$ tel que les plongements logarithmiques $\log|\sigma_i(S_k(\alpha)v^{-\lambda})|$ sont les moins déséquilibrés possibles par rapport à leur valeur moyenne. Quelques détails supplémentaires sont donnés dans [159].

Complexité Du moment que l'on prend soin de ne pas spécifier la dépendance de la complexité en les paramètres du corps de nombres K, l'évaluation de celle-ci est aisée. En effet, le processus est linéaire en la taille du résultat (à aucun moment le produit $S(\alpha)$ n'est calculé. Les plongements

logarithmiques sont accumulés pour un coût linéaire en la taille de l'entrée). Remarquons que l'algorithme n'est pas significativement sensible à la valeur de l'exposant λ. Seule la précision requise pour les plongements logarithmiques croît linéairement en $\log \lambda$.

4.2 Une approche combinant relèvement et restes chinois

Nous décrivons ici une approche intermédiaire entre l'approche par relèvement (voir section 4.1.1, page 47) et l'approche de Couveignes par restes chinois (voir section 4.1.2, page 49). L'approche que nous proposons ici a l'avantage de fonctionner en toute généralité, sans supposer une propriété particulière sur le corps de nombres considéré. En particulier, on ne suppose pas qu'il existe un nombre premier inerte.

Ce travail peut être rapproché de travaux récents de Enge et Sutherland, et de Sutherland, sur le calcul de polynômes de classes pour la construction de courbes elliptiques à multiplication complexe par la méthode des restes chinois [71, 203, 204]. Dans ces travaux, comme dans la méthode décrite ici, un accent particulier est mis sur la formulation dite « explicite » du théorème des restes chinois.

Notons $d = [K : \mathbb{Q}]$. On rappelle que pour simplifier les notations, on se place dans le cas où le corps K est défini par un polynôme f unitaire. Comme cela a déjà été décrit pour l'approche par relèvement, et utilisé implicitement pour l'algorithme de Couveignes également, la première étape de l'algorithme de calcul de racine carrée est de calculer une borne sur les coefficients du résultat. Notons une telle borne M. Nous choisissons un ensemble de nombres premiers $\mathcal{P} = \{p_i\}$ totalement décomposés[3], de cardinal $\ell = t \times r$ (les valeurs précises sont discutées plus loin). Dans l'objectif d'une organisation du calcul qui sera détaillée plus loin, l'ensemble $\mathcal{P}$ est partitionné en t sous-ensembles $\mathcal{P}_1, \ldots, \mathcal{P}_t$, chacun de cardinal r. On note $P = \prod_{p \in \mathcal{P}} p$, et $\lambda = \left\lceil \frac{\log M/\epsilon}{\log P} \right\rceil$, où $\epsilon \leq 1$ est choisi arbitrairement, et discuté plus loin. On note $B = \frac{\lambda}{\ell} \log_2 P$ la taille en bits des valeurs p^λ pour $p \in \mathcal{P}$. En accord avec nos hypothèses sur la taille des objets considérés, la taille attendue pour les coefficients de $T(\alpha)$ est environ $n/2$ bits, où n désigne comme précédemment la taille en bits de la donnée d'entrée $\mathcal{S}$. Par conséquent, nous estimons $\log_2 M \approx \ell B \approx n/2$.

4.2.1 Reconstruction par restes chinois

La racine carrée est calculée par restes chinois à partir de calculs modulo p_i^λ, pour chaque p_i. L'une des difficultés est naturellement liée à la résolution de l'indétermination parmi les choix de racines. Une procédure par restes chinois nécessite le calcul de n racines carrées dans chacun des corps finis $\mathbb{F}_{p_i}$, soit un total égal à ℓd.

Pour chaque nombre premier p_i, on note $(r_{i,j})_{j=1\ldots d}$ les racines de f modulo p_i. Par hypothèse, les p_i sont totalement décomposés, donc pour i donné les $r_{i,j}$ sont d éléments distincts de $\mathbb{F}_{p_i}$. Cette hypothèse permet en outre de calculer pour chacun des $r_{i,j}$ un relèvement $\tilde{r}_{i,j}$ dans $\mathbb{Z}/p_i^\lambda\mathbb{Z}$. On a donc $f(\tilde{r}_{i,j}) \equiv 0 \bmod p_i^\lambda$.

Pour chaque p_i, et chaque $r_{i,j}$, on calcule un relèvement p_i-adique de $\sqrt{S(\tilde{r}_{i,j})}$, à précision λ. Notons $T'_{i,j}$ ce relèvement. Par construction, on a

$$T'_{i,j} = s_{i,j} T(\tilde{r}_{i,j}) \quad \text{où } s_{i,j} = \pm 1.$$

Nous souhaitons déterminer $T(x)$ à partir des $T(\tilde{r}_{i,j})$ (ces derniers, pour l'instant, ne sont connus qu'au signe $s_{i,j}$ près). Une façon de voir le processus de reconstruction en jeu ici est de considérer

[3] La densité des nombres premiers totalement décomposés est $\frac{1}{\#\,\mathrm{Gal}(f)}$.

les idéaux $I_{i,j} = \langle p_i, \alpha - r_{i,j}\rangle^\lambda = \langle p_i^\lambda, \alpha - \tilde{r}_{i,j}\rangle$. En effet, $T_{i,j}$ n'est autre que l'image de $T(x)$ par la projection $\mathcal{O}_K \to \mathcal{O}_K/I_{i,j}$. Pour relever cette projection, on considère les entiers Q_i et les polynômes $H_{i,j}(x) \in \mathbb{Z}[x]$ définis ainsi :

$$Q_i \stackrel{\text{déf}}{=} \left(\frac{P}{p_i}\right)^\lambda = \prod_{\substack{p\in\mathcal{P},\\ p\neq p_i}} p^\lambda,$$

$$H_{i,j} \stackrel{\text{déf}}{=} \frac{f(x)}{(x-\tilde{r}_{i,j})} = \prod_{j'\neq j} (x - \tilde{r}_{i,j'}).$$

Ces polynômes sont construits de sorte à vérifier :

Proposition 4.3. *Soit $T_{i,j} = T(\tilde{r}_{i,j}) \bmod p_i^\lambda$. On a :*

$$T(x) = \left(\sum_{i,j} Q_i H_{i,j}(x) T_{i,j} \frac{1}{Q_i f'(\tilde{r}_{i,j})}\right) \bmod P^\lambda.$$

L'inverse dans l'énoncé qui précède s'entend dans $\mathbb{Z}_{p_i}$, plus précisément modulo p_i^λ. Il suffit pour obtenir cette identité de constater que $Q_i \bmod p_{i'}^\lambda$ est nul pour $i' \neq i$, et que dans le cas $i = i'$, $H_{i,j}(\tilde{r}_{i',j'}) \bmod p_{i'}^\lambda$ vaut exactement $f'(\tilde{r}_{i,j})$ pour $j' = j$, et zéro sinon. Ensuite, la borne $M \leq P^\lambda$ sur les coefficients de T permet de conclure.

4.2.2 Détermination des signes

La proposition 4.3 permet de reconstruire $T(x)$ lorsque les $T_{i,j}$ sont connus. Par le calcul de racines modulo les p_i, nous n'avons accès qu'aux $T'_{i,j} = s_{i,j} T_{i,j}$. Notons toutefois qu'il est possible d'écrire l'identité suivante en examinant le coefficient de degré $d-1$ de $T(x)$ (n'importe quel coefficient peut être considéré ; l'expression pour le degré $d-1$ est un peu moins lourde) :

$$[x^{d-1}]T(x) = \sum_{i,j} Q_i T_{i,j} \frac{1}{Q_i f'(\tilde{r}_{i,j})} \bmod P^\lambda,$$

$$\frac{1}{P^\lambda}[x^{d-1}]T(x) = \sum_{i,j} \frac{1}{p_i^\lambda}\left(T_{i,j}\frac{1}{Q_i f'(\tilde{r}_{i,j})} \bmod p_i^\lambda\right) \bmod 1.$$

Soient maintenant $x_{i,j}$ et $y_{i,j}$ les nombre réels

$$x_{i,j} = \frac{1}{p_i^\lambda}\left(T_{i,j}\frac{1}{Q_i f'(\tilde{r}_{i,j})} \bmod p_i^\lambda\right) \in [0,1[,$$

$$y_{i,j} = \frac{1}{p_i^\lambda}\left(T'_{i,j}\frac{1}{Q_i f'(\tilde{r}_{i,j})} \bmod p_i^\lambda\right) \in [0,1[,$$

$$\equiv \pm x_{i,j} \bmod 1.$$

Nous pouvons montrer que la somme des $x_{i,j}$ est exceptionnellement proche d'un nombre entier. En effet, le coefficient $[x^{d-1}]T(x)$ est borné par M. Dès lors, on vérifie que l'expression $\left|\frac{1}{P^\lambda}[x^{d-1}]T(x)\right|$ est bornée par $MP^{-\lambda} \leq \epsilon$, où ϵ est le paramètre arbitraire fixé plus haut. On a ainsi :

$$\sum_{i,j} x_{i,j} \in [-\epsilon, \epsilon] + \mathbb{Z},$$

$$\sum_{i,j} s_{i,j} y_{i,j} \in [-\epsilon, \epsilon] + \mathbb{Z}.$$

Il est donc possible de lever l'indétermination de signe par la résolution d'un problème de sac à dos, en trouvant la « bonne » combinaison de signes $(s_{i,j})_{i,j}$. Ce problème est difficile, et nous y revenons plus loin. Notons pour l'instant que la résolution d'un tel problème reste praticable pour des petites dimensions.

Une fois calculés les $s_{i,j}$, on peut déduire l'entier κ_{d-1} proche de $\sum_{i,j} x_{i,j}$. On obtient de même $T_{i,j} = s_{i,j} T'_{i,j}$. Il en découle :

$$\frac{1}{P^\lambda}[x^{d-1}]T(x) = \sum_{i,j} \frac{1}{p_i^\lambda} \left(s_{i,j} T'_{i,j} \frac{1}{Q_i f'(\tilde{r}_{i,j})} \bmod p_i^\lambda \right) - \kappa_{d-1}.$$

Cette approche peut ensuite se généraliser pour dériver une information pour chacun des coefficients de $T(x)$ (toujours à partir des mêmes $s_{i,j}$, qui n'ont donc pas besoin d'être calculés de nouveau). Ceci nécessite de prendre en compte les polynômes d'interpolation $H_{i,j}(x)$. On définit :

$$\begin{aligned} c_{i,j,k} &= [x^k] \left(T'_{i,j} \frac{H_{i,j}(x)}{Q_i f'(\tilde{r}_{i,j})} \bmod p_i^\lambda \right), \\ c^*_{i,j,k} &= s_{i,j} c_{i,j,k}. \end{aligned}$$

Ces coefficients généralisent les notations $x_{i,j}$ et $y_{i,j}$ utilisées plus haut, puisqu'on a :

$$x_{i,j} = \frac{c^*_{i,j,d-1}}{p_i^\lambda}, \quad \text{et} \quad y_{i,j} = \frac{c_{i,j,d-1}}{p_i^\lambda}.$$

Avec ces notations, on peut généraliser l'expression donnée plus haut, et enfin récrire l'expression de la proposition 4.3.

$$\begin{aligned} \frac{1}{P^\lambda}[x^k]T(x) &= \sum_{i,j} \frac{1}{p_i^\lambda}[x^k] s_{i,j} c_{i,j,k} - \kappa_k, \\ T(x) &= \sum_{i,j,k} x^k \left(Q_i s_{i,j} c_{i,j,k} - \kappa_k P^\lambda \right). \end{aligned}$$

Nous remarquons enfin que l'expression que nous venons d'obtenir se transcrit en une expression donnant $T(m) \bmod N$ sans nécessité de calculer $T(\alpha)$. On est donc intéressé par le calcul des quantités $c_{i,j,k} m^k \bmod N$, qui sont des données beaucoup plus petites que p_i^λ.

4.2.3 Stratégies pour un calcul rapide

L'approche développée jusqu'ici nécessite quelques précisions. En effet, le découpage en de nombreux calculs indépendants, s'il est mal fait, peut s'avérer contre-productif. Le premier point d'importance est le calcul des valeurs $S(\tilde{r}_{i,j})$. Rappelons que ce sont $d\ell$ entiers de B bits. Afin de les calculer efficacement, il est possible de procéder comme suit. Tout d'abord, on calcule $S(\alpha)$ au moyen d'un arbre de sous-produits (voir par exemple [84, § 10.1]). On calcule ensuite la réduction de chacun des coefficients modulo l'ensemble des ℓ puissances de nombres premiers p_i^λ. Cette réduction multi-modulaire, à nouveau, peut être réalisée au moyen d'un arbre de sous-produits. Pour finit, le calcul des valeurs voulues relève à nouveau d'une multi-évaluation en les $\tilde{r}_{i,j}$, quoique la largeur de cette dernière (d points d'évaluation) est limitée.

Algorithme crtalgsqrt$(N, m, f, \mathcal{S})$

ENTRÉES : f unitaire définissant un corps de nombres $K = \mathbb{Q}(\alpha)$,
N entier,
m une racine de f modulo N.
$\mathcal{S}$ ensemble de paires (a, b) tel que $S(\alpha) = \prod(a - b\alpha) \in \mathbb{Z}[\alpha]^2$.
PARAMÈTRES : $\ell = r \times t$, nombre de nombres premiers à considérer.
SORTIE : $T(m) \bmod N$, où $T(\alpha)^2 = S(\alpha)$.

1. Déterminer t ensembles $\mathcal{P}_1, \ldots, \mathcal{P}_t$ de r nombres premiers totalement décomposés dans K. Partitionner $\mathcal{S}$ en t ensembles disjoints $\mathcal{S}_1, \ldots, \mathcal{S}_t$.
2. Pour $k = 1, \ldots, t$, lire $\mathcal{S}_k$. En déduire M, et λ. Calculer $S_k(x) = \prod_{(a,b)\in\mathcal{S}_k}(a - bx) \bmod f$.
3. Pour $i = 1, \ldots, \ell$, calculer les quantités p_i^λ, Q_i, $\frac{1}{Q_i} \bmod p_i^\lambda$, ainsi que pour $j = 1, \ldots, d$, la racine $r_{i,j} \bmod p_i$ de f, le relèvement $\tilde{r}_{i,j}$, et enfin $\frac{H_{i,j}(x)}{f'(\tilde{r}_{i,j})} \bmod p_i^\lambda$.
4. Pour $\tau = 1, \ldots, t$:
   4.1. calculer $(S_\sigma(x) \bmod \mathcal{P}_\tau \stackrel{\text{déf}}{=} \{S_\sigma(x) \bmod p,\ p \in \mathcal{P}_\tau\})_{\sigma=1\ldots t}$.
   4.2. pour chaque $p_i \in \mathcal{P}_\tau$, et pour $j = 1, \ldots, d$, calculer :
      4.2.1 les évaluations $S_\sigma(\tilde{r}_{i,j}) \bmod p_i^\lambda$ pour $\sigma = 1, \ldots, t$,
      4.2.2 les produits $S(\tilde{r}_{i,j})$,
      4.2.3 les racines $T'_{i,j} = \sqrt{S(\tilde{r}_{i,j})} \bmod p_i^\lambda$.
      4.2.4 les coefficients $c_{i,j,k} = [x^k]\left(T'_{i,j}\frac{H_{i,j}(x)}{Q_i f'(\tilde{r}_{i,j})} \bmod p_i^\lambda\right)$, ainsi que $c_{i,j,k}/p_i^\lambda \in \mathbb{R}$, et $c_{i,j,k}m^k \bmod N$.
5. Déterminer les signes $s_{i,j}$ et l'entier κ_{d-1} tels que $\left|\sum_{i,j} s_{i,j}(c_{i,j,d-1}/p_i^\lambda) - \kappa_{d-1}\right| \leq \epsilon$. En déduire les entiers $\kappa_0, \ldots, \kappa_{d-2}$.
6. retourner $\sum_{i,j,k} Q_i s_{i,j} c_{i,j,k} m^k - \kappa_k P^\lambda \bmod N$.

ALGORITHME 4.4 : Racine carrée pour NFS par relèvement et restes chinois.

Pour une entrée $\mathcal{S}$ de n bits, rappelons que nous avons $\ell B \approx n/2$. Dès lors, les deux premières étapes que nous avons indiquées ont pour complexité $O(d^2\mathsf{M}(n)\log n)$ et $O(d\mathsf{M}(n)\log n)$ respectivement, tandis que la dernière a pour complexité $O(\ell d^2\mathsf{M}(B))$ (où on prend en compte le fait que d est petit).

Il est possible de paralléliser, dans une mesure limitée, l'algorithme que nous avons décrit. Nous pouvons réduire d'un facteur t à la fois la complexité en temps et en mémoire, en utilisant en parallèle t^2 machines. Ceci s'obtient en divisant les ensemble $\mathcal{S}$ et $\mathcal{P}$ en t parties de tailles égales, que nous notons respectivement $\mathcal{S}_1, \ldots, \mathcal{S}_t$ et $\mathcal{P}_1, \ldots, \mathcal{P}_t$.

Nous associons à chacun des ensembles $\mathcal{S}_\sigma$ le polynôme

$$S_\sigma(x) = \prod_{(a,b)\in\mathcal{S}_\sigma}(a - bx) \bmod f.$$

Pour chaque couple d'indices (σ, τ), nous définissons le r-uplet suivant :

$$S_\sigma(x) \bmod \mathcal{P}_\tau \stackrel{\text{déf}}{=} \{S_\sigma(x) \bmod p,\ p \in \mathcal{P}_\tau\}.$$

Chacun des t^2 processeurs que nous utilisons, indexé par un couple (σ, τ), calcule l'une des quantités

Étape	Temps	Mémoire
1	$O(\ell)$	$O(\ell)$
2	$O(d^2\mathsf{M}(n))$	$O(dn)$
3	$O(\ell d^2\mathsf{M}(B))$	$O(dn)$
4.1	$O(d\mathsf{M}(n)\log n)$	$O(dn\log n)$
4.2.1	$O(\ell d^2\mathsf{M}(B))$	$O(dn)$
4.2.2	$O(\ell d\mathsf{M}(B))$	$O(dn)$
4.2.3, 4.2.4	$O(\ell d\mathsf{M}(B))$	$O(dn)$
5	$O(2^{d\ell/2})$	$O(2^{d\ell/2})$
6	$O(\ell)$	$O(\ell)$

TAB. 4.5 : Complexité en temps et en mémoire des différentes étapes de l'algorithme 4.4

ci-dessus. Il est aisé de vérifier que la complexité en temps et en mémoire sur chaque processeur est $(1/t)$-ème de la complexité totale donnée plus haut.

Une fois calculées les quantités $S_\sigma(\tilde{r}_{i,j})$, on en déduit $S(\tilde{r}_{i,j})$ par un produit sur tous les indices σ.

Les étapes que nous venons de décrire constituent les étapes centrales de l'algorithme 4.4, plus précisément ses étapes numérotées 1, 4.2.1, et 4.2.2. Les autres étapes reprennent la description faite plus haut, et ne sont pas développés dans les détails. La présentation de l'algorithme 4.4 privilégie le point de vue parallèle sur t^2 nœuds. Toutefois, il est possible d'en déduire la version séquentielle en substituant $t = 1$.

4.2.4 Complexité

La table 4.5 rassemble les complexités des différentes étapes de l'algorithme. Concernant la parallélisation, les étapes 3, 4.2.3, et 4.2.4 passent à l'échelle avec un gain linéaire jusque t nœuds. Nous avons montré plus haut comment les étapes 4.1 à 4.2.2 peuvent bénéficier d'une amélioration d'un facteur t en utilisant t^2 machines, si l'on néglige les coûts de communication.

L'étape 5 revient à la résolution d'un problème de sac à dos, qui est un problème difficile. Le problème à résoudre est un problème à $d\ell$ variables. Il est immédiat de le résoudre en complexité $2^{\frac{1}{2}d\ell}$, et la meilleure approche connue est en $2^{0.313d\ell}$ [107].

Au total, l'algorithme proposé offre une complexité en temps séquentielle totale de l'ordre de $\widetilde{O}(d^2\mathsf{M}(n)\log n)$, ce qui est compétitif avec l'approche par relèvement proposée plus haut (en incluant dans cette dernière le coût du calcul de $S(\alpha)$). En outre, un parallélisme limité est possible.

L'algorithme 4.4 a été implanté dans `cado-nfs` [90]. Pour une application pratique, le choix des paramètres r et t est limité en premier lieu par la complexité de l'étape 5, puisqu'au delà d'une certaine dimension, la résolution du problème de sac à dos est impraticable. Un cas d'utilisation, pour le calcul de la racine carrée dans RSA-768, a les paramètres suivants : $d = 6$, $t = 3$, $r = 2$, pour une entrée de taille totale 21 Go. Sur 3 machines Intel Xeon E5520, (2 processeurs, 4 coeurs par processeur), le calcul a nécessité 6 heures.

À titre de comparaison, l'algorithme de Montgomery a été utilisé pour mener ce même calcul pour RSA-768. Le temps nécessaire a été de 4 heures sur le même nombre de cœurs, ce qui permet d'arguer que l'algorithme présenté est raisonnablement compétitif.

4.2.5 Variante

Une variante peut être employée afin de pouvoir utiliser un plus grand nombre de nombres premiers (par exemple 10 000). Pour ce faire, il importe d'éviter l'étape de reconstruction par sac à dos. Une idée relevée dans les travaux de Enge et Sutherland [71] permet de contourner cette difficulté, dans le cas où on dispose de nombres premiers inertes. Nous notons $\mathcal{P}$ l'ensemble des nombres premiers choisis pour la reconstruction. Pour simplifier, nous supposons que chacun n'est utilisé qu'à l'exposant $\lambda = 1$, quoiqu'une variation sur ce thème soit possible et sans incidence. Pour $p \in \mathcal{P}$, nous notons $T_p(x) = T(x) \bmod p$. Reprenant une notation similaire déjà employée, on note $T'_p(x)$ la racine carrée calculée modulo p, et enfin on note s_p le signe tel que $T'_p(x) = s_p T(x) \bmod p$.

En menant les calculs modulo p, il est possible de déterminer $T'_p(x) = s_p T(x) \bmod p$. Intéressons-nous à un coefficient en particulier, par exemple celui de degré $d-1$. On note ce coefficient $\tau'_p = s_p \tau \bmod p$. Comme on a seulement $\tau \equiv \pm\tau'_p \bmod p$, on ne peut distinguer $\tau \bmod p$ de $-\tau \bmod p$. En revanche, l'*ensemble* $\{\tau \bmod p, -\tau \bmod p\}$ est lui bien déterminé par le calcul. Il est donc possible de calculer sans ambiguïté $\tau^2 \bmod p = (\tau'_p)^2 \bmod p$. En menant ce calcul modulo de très nombreux nombres premiers p, il est donc possible de déterminer l'entier τ^2. Pour ce faire, on doit considérer un ensemble de nombre premiers $\mathcal{P}$ tel que $\prod_{p\in\mathcal{P}} p \geq M^2$, où M est la borne sur les coefficients de $T(x)$. Dès lors, il est possible de déterminer τ en calculant une racine carrée entière, dont la complexité est plus basse que la racine carrée algébrique. Ce calcul permet de déduire les s_i, et donc de terminer le calcul.

Le surcoût associé à cette approche est la plus grande précision requise pour le relèvement. Il convient de noter que dans le cas qui nous intéresse, la borne M n'est pas *a priori* excessive concernant l'un des coefficients en particulier. Dans [71], le contexte d'application amène à une situation où naturellement, une borne plus stricte existe pour un coefficient particulier. Poursuivre le relèvement jusqu'au carré de ladite borne est alors peu couteux en comparaison.

Notons enfin qu'il est envisageable d'appliquer cette approche au cas où il n'existe pas de nombres premiers inertes. Dans un tel cas, soit $H \subset \mathfrak{S}_d$ un ensemble de représentants du quotient

$$H \leftrightarrow \mathrm{Gal}(L/\mathbb{Q})/\mathrm{Gal}(L/K),$$

où L est la clôture galoisienne de K. Soit $\sigma \in H$ ayant une décomposition en cycles disjoints faisant intervenir le minimum de cycles. Notons ce minimum γ. On s'intéresse aux nombres premiers dont le motif de décomposition dans K correspond à la décomposition en cycles de σ. D'après le théorème de densité de Čebotarev, la densité de ceux-ci est la densité dudit motif dans H, donc en particulier positive. Dès lors, pour chacun des nombres premiers p à considérer, le coefficient τ choisi ci-dessus s'exprime sous la forme $\pm\tau'^{(1)}_p \pm \ldots \pm \tau'^{(\gamma)}_p$. Cet ensemble a pour cardinal 2^γ. En évaluant les polynômes symétriques élémentaires en les 2^γ solutions possibles modulo chaque p, il est possible d'obtenir τ comme racine d'un polynôme à coefficients entiers. Le surcoût sur la précision du relèvement est alors multiplicatif en 2^γ. Notons que la valeur minimale de γ est $d/2$ dans le cas des extensions de Swinnerton-Dyer. On retrouve là sans surprise le cas difficile de la factorisation de polynômes sur les corps de nombres.

Chapitre 5

Le crible algébrique à l'unilatérale

Nous évoquons dans ce chapitre les travaux [T8] et [T12]. Guidé par des contextes d'applications en cryptanalyse, nous mettons en évidence comment il est possible de mettre en place le cadre du crible algébrique de manière adaptée, où l'un des « côtés » est remplacé par un oracle. L'existence de l'oracle fait partie du scénario invoqué par le contexte de cryptanalyse.

5.1 Problèmes avec oracle

Le cryptosystème RSA est la source à la fois d'un système de chiffrement et d'un système de signature. Une version simpliste consisterait à présenter le schéma de signature ainsi. Soient p et q deux nombres premiers, et $N = pq$. Soit e un exposant public (premier avec $\varphi(N)$), et d l'exposant privé défini par $de \equiv 1 \bmod \varphi(N)$. Une signature du message m, représenté sous la forme d'un entier modulo N, pourrait être $s = m^d \bmod N$. En effet, cette signature est vérifiable avec l'exposant public e, et le calcul de s à partir de m n'est a priori accessible qu'à celui qui connaît l'exposant secret d.

Naturellement, une telle présentation est trop limitée, car ce schéma se prête à des falsifications indésirables de signature. Par exemple, il est possible de créer des couples (message,signature) valides au sens de ce schéma. Pour ce faire, l'attaquant choisit d'abord la signature s, et en définit ensuite $m = s^e \bmod N$. Ce message m, bien que n'étant pas nécessairement pertinent sémantiquement parlant, est correctement signé par s. Ceci est un exemple de « falsification existentielle », qu'un attaquant obtient après bien peu d'effort.

La parade naturelle pour éliminer ce défaut consiste à éliminer la propriété de multiplicativité de la signature. Diverses techniques existent, l'une d'entre elles, de valeur essentiellement historique, emploie un *padding*. Le message, dans un tel contexte, est limité à k bits, avec k une fraction de $\log_2 N$. La signature de m est alors $(P + m)^d \bmod N$, où P est une constante de *padding*. De la sorte, on met en défaut la manipulation simple permettant la falsification existentielle vue plus haut. D'autre schémas existent, au nombre desquels le schéma OAEP [21, 197, 79], normalisé sous la forme du standard PKCS 1.5. Le schéma simple de *padding* doit être bien compris comme un sujet *d'étude* en premier lieu.

Peut-on obtenir des falsifications pour des signatures RSA qui utilisent ce schéma de *padding* ? Dans le contexte de l'établissement de propriétés cryptographiques [94, 19, 171], il est pertinent de savoir ce qu'un attaquant potentiel est en mesure d'obtenir dans un scénario d'attaque à *messages choisis* : étant donné un oracle réalisant la fonction de signature sur des messages choisis par l'attaquant, ce dernier est-il par exemple en mesure de créer une fausse signature d'un *challenge* (qui lui est donné après les requêtes) ? Peut-il, notamment, atteindre ce but significativement plus efficacement qu'en factorisant l'entier N ?

Nous répondons par l'affirmative à cette dernière question. En effet, tandis que la factorisation de N par le crible algébrique a une complexité heuristique $L_N[1/3, (64/9)^{1/3}]$, nous montrons qu'il est possible de falsifier *sélectivement* des signatures adoptant le schéma de *padding* précédemment évo-

$\mathcal{A}$		$\mathcal{C}$
$(m_i \in \mathbb{Z}/N\mathbb{Z})_{1\leq i\leq \ell}$	$\xrightarrow{(m_i)_i}$	
	$\xleftarrow{s_i}$	$s_i \leftarrow m_i^d \bmod N$
	$\xleftarrow{\hat{m}}$	$\hat{m} \in_R \mathbb{Z}/N\mathbb{Z}$
$\hat{s}$	$\xrightarrow{\hat{s}}$	$\hat{s}^e \stackrel{?}{=} \hat{m} \bmod N$

FIG. 5.1 : Problème « DTRSA »

$\mathcal{A}$		$\mathcal{C}$
		$b \in_R [\![1, p-1]\!]$
$(y_i \in \mathbb{Z}/N\mathbb{Z})_{1\leq i\leq \ell}$	$\xrightarrow{(y_i)_i}$	
	$\xleftarrow{z_i}$	$z_i \leftarrow y_i^b \bmod p$
	$\xleftarrow{\hat{y}}$	$\hat{y} \in_R \mathbb{F}_p^*$
$\hat{z}$	$\xrightarrow{\hat{z}}$	$\hat{z} \stackrel{?}{=} y^b \bmod p$

FIG. 5.2 : Problème « DTDHP »

qué avec une complexité heuristique $L_N[1/3, (32/9)^{1/3}]$, et en obtenant $L_N[1/3, (4/9)^{1/3}]$ réponses de la part de l'oracle. Par falsification sélective on entend la possibilité d'obtenir une signature d'un message m, choisi par l'attaquant avant les requêtes à l'oracle. Nous parvenons à ce but en exhibant des relations multiplicatives entre les messages, de la forme $(P+m_1)\times\ldots\times(P+m_k) \equiv 1 \bmod N$. Ce travail est décrit en 5.2.

Une application particulière de ce travail est liée non plus au cas particulier des signatures avec *padding* fixe, mais au problème RSA lui-même. Le problème RSA est celui du calcul de racines e-èmes modulo N. Nous montrons que ce problème peut être rendu plus aisé par l'appel à un oracle. Le scénario que nous envisageons (qu'on retrouve dans [78]) est un scénario semblable à une attaque à chiffrés choisis sur un système de chiffrement : après des appels à l'oracle, l'attaquant reçoit un *challenge*, dont il doit calculer la racine e-ème. Le résultat obtenu dans [T8] est que la complexité pour atteindre cet objectif est inférieure à celle de la factorisation de N. Notons qu'en cela, nous résolvons un problème calculatoire plus subtil à spécifier que le problème RSA lui-même. Nous pouvons rapprocher le problème que nous résolvons du problème nommé « *one-more-RSA* » par [20], mais ces deux problèmes sont différents, comme remarqué par [131]. Le problème que nous résolvons dans [T8], semblable au problème énoncé dans [78] a été baptisé « *delayed-target-RSA* » (DTRSA) dans [131]. La figure 5.1 représente la formalisation de ce problème, où $\mathcal{A}$ est l'attaquant, et $\mathcal{C}$ l'acteur honnête du protocole. Les grandeurs N, d, e correspondent à un contexte RSA usuel (d est connu par $\mathcal{C}$). Nous étudions ce problème en 5.3.

Dans le contexte du logarithme discret, les problèmes proposés par [131] comportent aussi le « *delayed-target Diffie-Hellman problem* » (DTDHP), pour lequel nos travaux offrent de manière semblable une solution significativement plus efficace que la résolution du problème du logarithme discret [T12]. Comme pour le problème DTRSA, une formalisation du problème DTDHP est proposée par la figure 5.2. Dans cette figure, $\mathcal{A}$ est l'attaquant, et $\mathcal{C}$ l'acteur honnête du protocole. Le contexte est celui du groupe multiplicatif $\mathbb{F}_p^*$ de $\mathbb{F}_p$, et l'acteur $\mathcal{C}$ agit comme oracle « Diffie-Hellman statique » (l'entier b est fixe). Ce problème, objet de l'article [T12], est étudié en 5.4.

5.2 Relations multiplicatives

La première étape de nos travaux dans la direction que nous avons évoquée est la fabrication de relations multiplicatives entre signatures RSA à *padding* affine fixe. L'objectif est donc de produire des ensembles de messages $(m_i)_{1\leq i\leq k}$ valides (c'est-à-dire, principalement, où les m_i obéissent à une borne prédéfinie) et d'exposants x_i tels que

$$(P+m_1)^{x_1}\times\ldots\times(P+m_k)^{x_k} \equiv Y^e \bmod N.$$

Une telle relation permet d'obtenir une falsification sélective de signature : l'attaquant peut demander à l'oracle la signature de $(k-1)$ messages de son choix parmi $m_1,\ldots,m_k$, et en déduire la signature

du dernier. En effet, si on note $s_i = (P + m_i)^d$ la signature de m_i selon le schéma de *padding* affine fixe choisi, la relation proposée implique :

$$s_1^{x_1} \times \ldots \times s_k^{x_k} \equiv Y \bmod N.$$

5.2.1 Description de l'approche

Utilisation de combinaisons de congruences Obtenir de telles relations multiplicatives peut se faire grâce à une approche de combinaison de congruences. Si on choisit des m_i tels que $P + m_i$ est friable sur une base de facteurs $\mathcal{F}$ de cardinal B, alors l'obtention de $B + 1$ telles valeurs m_i permet de garantir l'existence d'une solution, plus exactement une solution au système linéaire défini dans $\mathbb{Z}/e\mathbb{Z}$ par les équations correspondantes. Formellement, nous notons

$$(P + m_i) = \prod_{p \in \mathcal{F}} p^{\lambda_{i,p}},$$
$$\prod_{i=1}^{k} (P + m_i)^{x_k} = \prod_{p \in \mathcal{F}} p^{\sum_{i=1}^{k} x_i \lambda_{i,p}}.$$

Si on note M la matrice $M = (\lambda_{i,p})_{1 \leq i \leq k, p \in \mathcal{F}}$, alors il apparaît que si $\underline{x} = (x_1, \ldots, x_k)$ est tel que $\underline{x}M = 0 \bmod e$, alors on a bien une relation multiplicative de la forme souhaitée.

Une telle approche est hélas inefficace. En effet, les entiers $P + m_i$ sont par construction trop gros pour qu'il soit possible de les rendre friables. La complexité d'une approche de ce type est vouée à ne pas faire mieux que $L_N[1/2]$, soit un résultat moins efficace que le crible algébrique (qui permet de factoriser N). Une approche possible, en revanche, consiste à exploiter un morphisme entre une structure bien choisie et $\mathbb{Z}/N\mathbb{Z}$, tel que $(P + m_i) \in \mathbb{Z}/N\mathbb{Z}$ puisse apparaître naturellement comme l'image par ce morphisme d'un élément potentiellement « friable »

Une telle situation est fournie par le schéma du crible algébrique (figure 2.1). Il suffit pour cela de construire un polynôme dont P est une racine modulo N, ce que nous étudions plus loin.

Considérons alors $K = \mathbb{Q}(\alpha)$ le corps de nombres défini par un tel polynôme f (si f n'est pas irréductible, soit on peut utiliser l'un de ses facteurs stricts qui a une racine modulo N, soit on exhibe grâce à f une factorisation de N). Le morphisme d'anneaux de K dans $\mathbb{Z}/N\mathbb{Z}$ est donné par $\alpha \mapsto P$. L'élément $(P + m_i)$ de $\mathbb{Z}/N\mathbb{Z}$ est donc obtenu comme l'image de $\alpha + m_i$ par ce morphisme. Nous souhaitons donc nous intéresser aux éléments de la forme $\alpha + a \in K$. Notons que ces éléments ont un degré de liberté seulement, contre deux pour les éléments de la forme $(a - b\alpha)$ qui apparaissent dans NFS.

Nous définissions une *base de facteurs algébrique* $\mathcal{F}_a$ correspondant aux idéaux premiers de $\mathcal{O}_K$ dont la norme est inférieure à une certaine borne. Nous nous intéressons aux idéaux de la forme $(a + \alpha)\mathcal{O}_K$ qui sont *friables* sur cette base de facteurs $\mathcal{F}_a$. En d'autres termes, nous nommons *relations* les identités obtenues en considérant des valeurs m_i telles que cette propriété de friabilité est satisfaite. Nous écrivons ces identités pour une, ainsi que pour plusieurs relations, combinées par des facteurs multiplicatifs notés x_i :

$$(\alpha + m_i)\mathcal{O}_K = \prod_{\mathfrak{p} \in \mathcal{F}_a} \mathfrak{p}^{\lambda_{i,\mathfrak{p}}},$$
$$\prod_{i=1}^{k} (\alpha + m_i)^{x_k} \mathcal{O}_K = \prod_{\mathfrak{p} \in \mathcal{F}_a} \mathfrak{p}^{\sum_{i=1}^{k} x_i \lambda_{i,\mathfrak{p}}}.$$

Soit M la matrice $M = (\lambda_{i,\mathfrak{p}})_{1\leq i\leq k,\mathfrak{p}\in\mathcal{F}}$. Choisissons pour $\underline{x} = (x_1, \ldots, x_k)$ un vecteur solution de $\underline{x}M = 0 \bmod e$. L'idéal engendré par $(\alpha+m_i)^{x_i}$ est alors la puissance e-ème d'un idéal de $\mathcal{O}_K$, au vu des identités qui précèdent. Toutefois, exactement pour les mêmes raisons que celles mentionnées en 2.1.3, cela ne garantit pas que $(\alpha + m_i)^{x_i}$ est une puissance e-ème dans K. Les mêmes solutions que celles évoquées en 2.1.3 s'appliquent. Par l'emploi de caractères, il est possible de contraindre $(\alpha + m_i)^{x_i}$ à être effectivement une puissance e-ème. On a alors :

$$\begin{aligned} (\alpha + m_i)^{x_i} &= A(\alpha)^e, \\ \text{d'où} \quad (P + m_i)^{x_i} &\equiv A(P)^e \bmod N. \end{aligned}$$

Nous avons donc décrit une ligne de travail pour obtenir des relations multiplicatives, et donc des falsifications sélectives. Nous donnons maintenant quelques éléments de précision sur les différentes étapes de ce processus.

Choix d'un polynôme Une approche pour construire un polynôme ayant P comme racine modulo N est de considérer le réseau constitué par les lignes de la matrice $(d + 1) \times (d + 1)$ suivante :

$$M_{N,P} = \begin{pmatrix} N & & & \\ -P & 1 & & \\ & \ddots & \ddots & \\ & & -P & 1 \end{pmatrix} \in \mathcal{M}_{(d+1)\times(d+1)}(\mathbb{Z}).$$

Un vecteur court de ce réseau, obtenu par l'algorithme LLL, a des coefficients de l'ordre de $N^{\frac{1}{d+1}}$, en négligeant les facteurs en d. Un polynôme f ayant ces coefficients a par construction P comme racine modulo N.

Polynômes tordus La norme d'un élément de la forme $\alpha + a$ est donnée, comme on l'a vu pour NFS plus haut, par l'expression $\frac{1}{f_d}\operatorname{Res}(f(x), x + a)$. Dès lors, pour $|a| \approx S$, on a

$$\operatorname{Norm}_{K/\mathbb{Q}}(\alpha + a) \approx \frac{1}{f_d} N^{\frac{1}{d+1}} S^d.$$

On note que la présence d'un unique degré de liberté a pour conséquence que les différents termes dans l'expression $\operatorname{Res}(f(x), x + a)$ qui donne la norme sont déséquilibrés. C'est le terme de tête en $f_d S^d$ qui domine, les autres termes étant significativement moins importants. Dans un tel cas, il est pertinent de recourir à une construction de polynômes tordus (*skewed*). Cette pratique est usuelle dans la sélection polynomiale pour GNFS, puisqu'elle permet de choisir dans un espace de polynômes plus vaste. Pour forcer un ratio constant ρ entre les coefficients du polynôme obtenu, on considère la matrice $(d + 1) \times (d + 1)$ donnée par $\delta_\rho = \operatorname{diag}(1, \rho, \ldots, \rho^d)$, et on s'intéresse à un vecteur court $v = (v_0, \ldots, v_d)$ du réseau défini par les lignes de $M\delta_\rho$. Les coefficients de ce vecteur vérifient

$$\begin{aligned} v_i &\equiv 0 \bmod \rho^i, \\ v_i &\approx N^{1/(d+1)} \rho^{d/2}. \end{aligned}$$

Notons que pour éviter que la contrainte de divisibilité perturbe le comportement « géométrique » qui égale les coefficients, il faut imposer $\rho^d \ll N^{1/(d+1)}\rho^{d/2}$, soit $\rho \ll N^{\frac{2}{d(d+1)}}$. Dans un tel cas, si f

est le polynôme ayant pour coefficients les $v_i\rho^{-i}$ (soit les coefficients de $v\delta_\rho^{-1}$), alors on a $f_i/f_{i+1} \approx \rho$, et $f_i \approx N^{1/(d+1)}\rho^{d/2-i}$.

Pour l'application au calcul de relations multiplicatives entre les $P+m$, nous verrons au cours de l'analyse dans quelle mesure cette possibilité de créer des polynômes tordus permet un gain visible.

Obtenir des relations Une relation, dans le cadre proposé, est une valeur a_i telle que $(\alpha + a_i)$ est un idéal friable sur la base de facteurs $\mathcal{F}_a$. On considère l'intervalle $[\![-A, +A]\!]$ des valeurs a_i possibles[1]. Il est tout à fait possible de *cribler* pour localiser les valeurs a_i donnant des valeurs friables de $\mathrm{Norm}_{K/\mathbb{Q}}(\alpha+a_i)$. En revanche, l'algorithme efficace du *lattice sieving* [174] n'est pas utilisable, car ce dernier repose de manière absolument fondamentale sur le fait que les valeurs friables à détecter sont dans un espace à deux dimensions.

Le point-clé de la recherche de relations, et de l'analyse, est qu'ici la condition n'est plus, comme dans NFS, la friabilité des *deux* côtés, mais d'un seul. En effet, il importe peu que l'entier $P + a$ soit friable ou non : cela n'intervient pas. La seule condition importante est la friabilité de l'idéal $(\alpha+a)$. Requérir la friabilité d'un seul côté a un impact considérable sur l'amélioration de la complexité.

Exploiter les relations obtenues Lorsque suffisamment de relations ont été collectées, la matrice M correspondante, définie plus haut, a un noyau (à gauche) non trivial. Il est donc possible de déterminer une combinaison linéaire des lignes qui s'annule modulo e. Ceci peut être combiné avec l'emploi de caractères tels que décrits précédemment en 2.1.4 et 3.1.3. De la sorte, on peut forcer $(\alpha + m_i)^{x_i}$ à être une puissance e-ème dans le corps de nombres K.

Avant que ceci permette de conclure, il faut pouvoir *calculer* ladite racine e-ème. Dans ce contexte précis, l'algorithme de Montgomery [157, 159], que nous avons détaillé en 4.1.3, permet de trouver la racine. La complexité de l'algorithme de Montgomery dépend de la taille du résultat, et pas de la taille du produit développé (lequel croit exponentiellement en $\log e$). Ceci est un atout considérable pour permettre le calcul de la racine carrée algébrique.

5.2.2 Analyse

Nous effectuons l'analyse de manière semblable à l'analyse de NFS. Notons $B = L_N[b, \beta]$ la borne définissant la base de facteurs, et $S = L_N[s, \sigma]$ la valeur maximale considérée pour le paramètre a qui définit les idéaux $(\alpha + a)\mathcal{O}_K$ qu'on tente de « friabiliser ». Le degré d du polynôme construit est noté $\log_{\log N} L_N[\Delta, \delta]$. Sous ces hypothèses, on a $S \approx B^2$ (l'espace de recherche guide le temps de recherche de relations car le temps de factorisation est amorti grace au crible et peut être négligé). Ceci impose $s = b$ et $\sigma = 2\beta$. Le polynôme f est choisi tordu, avec un rapport entre les coefficients ρ qui est égal à S. On obtient alors la borne suivante pour $\mathrm{Res}(f, a + x)$:

$$\begin{aligned}\mathrm{Res}(f, a + x) &\approx N^{1/(d+1)}\rho^{-d/2}S^d,\\ &\approx N^{1/(d+1)}S^{d/2} \quad \text{si } S < N^{\frac{2}{d(d+1)}},\end{aligned}$$

d'où découle en particulier que l'expression de la norme est de la forme $L_N[\max(1 - \Delta, s + \Delta), \cdot]$. Notons qu'on a

$$N^{\frac{2}{d(d+1)}} = L_N[1 - 2\Delta, \frac{2}{\delta^2} + o(1)],$$

[1]La notation $P + m$ suppose qu'on choisit $m > 0$. Toutefois, décaler P de sorte à centrer l'intervalle des valeurs admises pour m est trivial.

de sorte que la contrainte sur S impose $s = b \leq 1 - 2\Delta$. La norme, et la probabilité de friabilité, ont alors pour expression :

$$\begin{aligned}\mathrm{Res}(f, a + x) &= L_N[s + \Delta, \frac{1}{\delta} + \frac{\sigma\delta}{2}] = L_N[b + \Delta, \frac{1}{\delta} + \beta\delta + o(1)],\\ \pi &= L_N[\Delta, -\Delta\frac{1}{\beta}(\frac{1}{\delta} + \beta\delta) + o(1)].\end{aligned}$$

Dans l'expression qui précède, la probabilité de friabilité a été estimée au moyen de la proposition 2.6, toujours sous l'heuristique usuelle 2.7. Afin de minimiser la norme, on pose $\delta = \beta^{-1/2}$. Pour obtenir $S\pi \approx B$, on pose $\Delta = s = b$, d'où $\Delta = s = b = \frac{1}{3}$. Toujours en exploitant $S\pi \approx B$, on obtient alors l'équation :

$$\begin{aligned}2\beta - \frac{2}{3\beta}\sqrt{\beta} &= \beta,\\ \beta^{3/2} &= \frac{2}{3}.\end{aligned}$$

Ceci donne la complexité totale de l'algorithme sous la forme de l'expression suivante.

Proposition 5.3. *Soit N un entier, et $\mathcal{O}$ un oracle répondant à la requête $s \leftarrow (P + m)^d \bmod N$. On note :*

$$\begin{aligned}T &= L_N[1/3, (32/9)^{1/3} + o(1)],\\ N_{\mathcal{O}} &= L_N[1/3, (4/9)^{1/3} + o(1)].\end{aligned}$$

Alors il est possible en un temps de calcul heuristiquement égal à T d'identifier un ensemble de valeurs m_i tels qu'une relation de dépendance multiplicative entre les m_i existe. Le cardinal de cet ensemble est $N_{\mathcal{O}} + 1$, de sorte qu'avec un nombre de requêtes à l'oracle égal à $N_{\mathcal{O}}$, il est possible d'exhiber un couple $(P + m, (P + m)^d)$ valide ou $(P + m)$ ne duplique aucune des requêtes effectuées.

5.2.3 Résultats expérimentaux

Afin d'illustrer l'approche, un test a été mené sur un entier RSA de 512 bits, à savoir l'entier du défi RSA-155, qui fut factorisé en 1999 [40]. On a choisi une valeur arbitraire pour le *padding* P, ainsi qu'une valeur qu'on a jugée significative pour l'exposant public e. Bien entendu le calcul a été mené sans exploiter la factorisation connue de l'entier N.

$$\begin{aligned}N = \quad & 10\,941\,738\,641\,570\,527\,421\,809\,707\,322\,040\,357\,612\,003\,732\,945\\ & 449\,205\,990\,913\,842\,131\,476\,349\,984\,288\,934\,784\,717\,997\,257\,891\,267\,332\\ & 497\,625\,752\,899\,781\,833\,797\,076\,537\,244\,027\,146\,743\,531\,593\,354\,333\,897,\\ P = \quad & 10^{154},\\ e = \quad & 65\,537.\end{aligned}$$

On a choisi une borne de friabilité $B = 2^{22}$, soit environ 300 000 idéaux premiers dans la base de facteurs. Pour un espace de recherche de relations borné par $S = 2^{50}$, un polynôme adéquat est donné par :

$$f(x) = f_4x^4 + f_3x^3 + f_2x^2 + f_1x^1 + f_0,$$

$$
\begin{aligned}
f_4 &= 8,\\
f_3 &= 5\,451\,802\,006\,688\,119,\\
f_2 &= -7\,344\,893\,341\,388\,732\,622\,814\,165\,470\,437,\\
f_1 &= 833\,050\,630\,351\,576\,525\,584\,507\,524\,542\,841\,090\,670\,386\,803,\\
f_0 &= -80\,690\,902\,433\,251\,999\,116\,158\,516\,330\,020\,702\,292\,190\,401\,223\,994\,350\,445\,959.
\end{aligned}
$$

Aucun effort particulier n'a été mis sur le choix de ce polynôme (un millier de tests). Dans le corps de nombres défini par f, exactement $295\,842$ idéaux premiers de degré 1 sont de norme inférieure à B.

Une procédure de crible par « special-q » a été employée [57]. En considérant les $20\,000$ idéaux de plus grande norme parmi ces derniers comme des « special-q », on s'est ramené à rechercher les $x \in [2^{-28}, 2^{28}]$ tels que l'idéal engendré par $(r-\alpha)+qx$ est friable. De la sorte, on restreint notre vue à un sous-ensemble de $[-2^{50}, 2^{50}]$. Ces idéaux sont par construction divisibles par $\langle q, \alpha - r\rangle$. Par un processus de crible, on a isolé les valeurs de x telles que la norme cumulée des facteurs identifiés de l'idéal $(r-\alpha)+qx$ dépasse 2^{145} (la norme de $(r-\alpha)+qx$ étant quant à elle de l'ordre de 2^{200}). De la sorte, environ $380\,000$ relations ont été obtenues, en un temps de l'ordre d'une quarantaine d'heures de calcul sur un cœur Intel Core2 6700 à 2.667 GHz.

Après filtrage, le système linéaire à résoudre a pour dimension $283\,355 \times 283\,355$. Une vingtaine d'heures de calcul ont été utilisées pour l'algèbre linéaire (matériel identique), en utilisant l'algorithme de Wiedemann par blocs.

La donnée expérimentale concernant le calcul de racine e-ème est probablement plus spécifique à notre calcul. Pour calculer une racine $65\,537$-ème dans le corps de nombres, nous avons utilisé un programme écrit en Magma.

L'élément de départ est une expression sous forme de produit de la valeur dont on doit calculer la racine e-ème. Cette valeur, initialement, a un dénominateur et un numérateur de normes respectives $\simeq e^{7.6\times 10^5}$. D'autre part, les plongements logarithmiques de cette valeur initiale apparaissent comme peu déséquilibrés, puisqu'on peut les écrire

$$(\lambda,\lambda,\lambda,\lambda) + \underbrace{(45, 45, -155, 65)}_{\delta} \quad \text{avec} \quad \lambda = \frac{1}{d}\log \mathrm{Norm}(\pi) \simeq 6710.$$

Dans l'expression ci-dessus λ est un terme de normalisation. Le déséquilibre dans ces valeurs doit être jugé comme maigre, puisqu'une unité qui aurait comme plongements logarithmiques le vecteur δ ci-dessus correspondrait à un entier algébrique de coefficients occupant environ 20 chiffres décimaux. Les quatre premières étapes de la réduction itérative ont suffi à recentrer la contribution logarithmique (égaler les différents termes des plongements logarithmiques avec leur moyenne). Au total, $2\,000$ étapes de réduction ont été nécessaire pour réduire complètement la valeur initiale et calculer la racine e-ème voulue, pour un temps total de 5 minutes.

La dépendance multiplicative ainsi produite implique $N_{\mathcal{O}} \approx 242\,700$ valeurs $(P + m_i)$.

5.3 Calcul de racines e-èmes arbitraires

Une adaptation de la méthode présentée plus haut s'attaque au problème du calcul de racines e-èmes arbitraires. On se place toujours dans un contexte RSA, et comme présenté plus haut on suppose l'accès à un oracle. Ici, l'oracle effectue la tâche du calcul de racine e-ème : $y \leftarrow x^d \bmod N$. Il n'y a plus de *padding*.

La situation retrouvant la propriété de multiplicativité de RSA, il convient de bien fixer le scénario, certains scénarios possibles étant vides de sens. On fixe comme objectif le calcul d'une racine *arbitraire*, ce qui correspond dans le langage de la falsification de signature à une falsification universelle : l'attaquant n'a aucun contrôle sur la valeur dont il doit calculer la racine e-ème. Celle-ci lui est communiquée sous la forme d'un défi. D'autre part, il est sans intérêt de permettre à l'attaquant d'effectuer des requêtes à l'oracle une fois que celui-ci a reçu le défi à calculer (il est trivial de masquer le défi : la racine e-ème de xr^e permet à l'attaquant de retrouver x^d). En d'autres termes, on se place exactement dans le contexte du problème calculatoire « DTRSA » schématisé par la figure 5.1 en page 60, et directement voisin des notions présentées dans [78, 131].

Le point-clé du travail effectué est lié à l'utilisation d'une procédure de descente, semblable à la procédure du même nom utilisée dans le contexte du logarithme discret, et décrite plus en détail en 3.5. Nous indiquons en quoi cette procédure nous est utile, et les ajustements nécessaires pour qu'il soit possible de l'utiliser pour résoudre notre problème.

5.3.1 Principe général

Considérons un contexte semblable à celui du crible algébrique (on peut faire référence à la figure 2.1 page 14). Les notations adoptées sont celles déjà utilisées dans l'ensemble des chapitres 2 et 3 : K est un corps de nombres défini par un polynôme f, le « côté rationnel » est défini par un polynôme g, et f et g ont pour résultant un multiple de N. Les notations α et m désignent des racines de f et g, respectivement dans K et $\mathbb{Q}$. On note π_K et $\pi_{\mathbb{Q}}$ les morphismes d'anneaux de K et $\mathbb{Q}$ vers $\mathbb{Z}/N\mathbb{Z}$. Le positionnement du contexte que nous définissons, selon les lignes qui précèdent peut être perçu assez proche du thème de la factorisation. En réalité, exception faite de l'intervention de l'entier composé N, nous verrons que ce sont principalement l'aspect « logarithme discret » et le formalisme défini au chapitre 3 qui prévalent. Ainsi, nous supposons définies deux bases de facteurs $\mathcal{F}_r$ et $\mathcal{F}_a$. Nous avons à notre disposition la machinerie classique du crible algébrique pour créer des relations, et nous allons voir maintenant comment il est possible de l'utiliser.

Nous reprenons l'objectif de la procédure de descente (problème 3.11) décrite en 3.5.1. Celle-ci, étant donné un élément cible $t \in \mathbb{Z}/N\mathbb{Z}$, produit un rationnel $\hat{t}$ et une fraction rationnelle $Q(x)$ tels que :

$$Q(m)\hat{t} \in \Sigma_r, \quad Q(\alpha) \in \Sigma_a, \quad \pi_{\mathbb{Q}}(\hat{t}) \equiv t \bmod N$$

où les ensembles Σ_r et Σ_a désignent des grandeurs *friables* relativement aux bases de facteurs $\mathcal{F}_r$ et $\mathcal{F}_a$ (voir notation 3.4).

Une telle fraction rationnelle, construite à partir d'un élément cible $t \in \mathbb{Z}/N\mathbb{Z}$ dont on souhaite précisément calculer une racine e-ème, permet d'aboutir sous réserve qu'on oracle a pu fournir les réponses voulues au préalable. En effet, écrivons :

$$Q(m)\hat{t} = \prod_{p \in \mathcal{F}_r} p^{u_p}, \tag{5.4}$$

$$Q(\alpha) = \prod_{\mathfrak{p} \in \mathcal{F}_a} \mathfrak{p}^{v_{\mathfrak{p}}}. \tag{5.5}$$

Nous faisons maintenant la supposition que les requêtes à l'oracle, réalisées dans une étape préalable de l'attaque, incluent les racines e-èmes des nombres premiers p apparaissant dans la base de facteurs $\mathcal{F}_r$. De la sorte, il est possible à l'attaquant qui obtient ces informations, de calculer $\sqrt[e]{Q(m)\hat{t}}$ (ou

encore, plus formellement, de calculer $\sqrt[e]{\pi_{\mathbb{Q}}(Q(m)\hat{t})}$, puisque c'est cette grandeur qui est définie sur $\mathbb{Z}/N\mathbb{Z}$).

Avant d'indiquer comment il est possible de calculer la racine e-ème associée à la seconde identité indiquée plus haut, nous souhaitons insister sur un point absolument capital dans cette approche de descente. La procédure de descente, notamment les étapes élémentaires indiquées par l'algorithme 3.13 en page 42, reposent de façon cruciale sur le fait que *deux cotés*, classiquement « rationnel » et « algébrique », sont définis dans le contexte. C'est grâce à cette dualité que l'algorithme 3.13 peut « annuler » des idéaux d'un côté, pour les remplacer par des idéaux de l'*autre* côté (ce point a déjà été évoqué en 3.5.1). La mise en place (détaillée plus loin) de l'ensemble du processus pour le calcul de racines e-èmes impose donc de définir deux côtés.

Pour déterminer une racine e-ème de l'image dans $\mathbb{Z}/N\mathbb{Z}$ de $Q(\alpha)$, nous proposons d'utiliser une recherche de relations classique, mais ne faisant intervenir qu'une friabilité unilatérale comme cela a été fait dans la section précédente. Supposons en effet collecté (avant que soit proposé le *challenge* t) un ensemble de relations de la forme :

$$(a_i - b\alpha_i)\mathcal{O}_K = \prod_{\mathfrak{p}\in\mathcal{F}_a} \mathfrak{p}^{v_{\mathfrak{p}}}.$$

Lorsque ces relations sont en nombre suffisant, il est possible de les combiner pour obtenir un produit qui égale la factorisation en idéaux de $Q(\alpha)$ écrite plus haut, à la puissance e-ème d'un idéal près. À l'instar de ce qui a été développé en 2.1.3, il est possible d'étendre ce processus en utilisant des caractères, afin de créer une égalité entre éléments, à une racine e-ème près, s'écrivant :

$$Q(\alpha) \times \prod_i (a_i - b_i\alpha)^{-y_i} = R(\alpha)^e. \tag{5.6}$$

Dans cette expression, l'algorithme de Montgomery décrit en 4.1.3 permet de calculer le nombre algébrique $R(\alpha)$ en exploitant sa factorisation connue en idéaux. Ceci fait, si nous faisons l'hypothèse que l'ensemble des $\pi_K(a_i - b_i\alpha) = a_i - b_i m \bmod N$ constitue un second ensemble de requêtes à soumettre à l'oracle (rappelons que ces relations ont été collectées avant la connaissance de t), tous les éléments sont réunis pour écrire :

$$\begin{aligned}
\pi_K(Q(\alpha)) \times \prod_i (\pi_K(a_i - b_i\alpha))^{-y_i} &= \pi_K(R(\alpha))^e, \\
Q(m) \times \prod_i (a_i - b_i m)^{-y_i} &= R(m)^e \bmod N, \\
\sqrt[e]{Q(m)} \times \prod_i \sqrt[e]{a_i - b_i m}^{-y_i} &= R(m) \bmod N.
\end{aligned} \tag{5.7}$$

Cette dernière identité, combinée à l'information déjà obtenue donnant $\sqrt[e]{Q(m)\hat{t}}$ en exploitant le côté rationnel (équation 5.4), permet de conclure et de calculer $\sqrt[e]{t}$.

5.3.2 Étapes

Nous reprenons les étapes du calcul mené par l'attaquant. Parmi les éléments de contexte, nous supposons donné un entier N, et un exposant e.

1. Phase de sélection polynomiale.
   Étant donné le nombre N, déterminer deux polynômes f et g tels que $\mathrm{Res}(f, g) = N$, et

tels que f est un « bon » polynôme pour NFS : on requiert classiquement de f les mêmes propriétés que dans le contexte GNFS, à savoir des coefficients équilibrés, et de nombreuses racines modulo les petits nombres premiers.

2. Préparation des bases de facteurs.
On choisit une borne de friabilité B (détaillée lors de l'analyse). Les bases de facteurs, rationnelle et algébrique, sont constituées conformément aux définitions 2.2 et 2.3 en pages 14 et 15.

3. Collecte de relations.
On recherche un ensemble de paires $\mathcal{R} = \{(a_i, b_i)\}$ tel que les $a_i - b_i\alpha$ sont *friables*. Pour chacune de ces paires, on calcule quelques caractères d'ordre e (dans le cas où e est trop grand pour que les caractères évoqués en 2.1.4 puissent être utilisés, l'emploi de caractères ℓ-adiques de Schirokauer est possible (voir section 3.1.3 et [181]). Le nombre de relations que nous devons collecter ainsi est au minimum $\#\mathcal{F}_a$, plus le nombre de caractères calculés en sus.

4. Émission des requêtes.
L'attaquant requiert de la part de l'oracle le calcul des racines e-èmes suivantes :

$$\{\sqrt[e]{p},\ p \in \mathcal{F}_r\} \cup \{\sqrt[e]{a_i - b_i m},\ (a_i, b_i) \in \mathcal{R}\}.$$

5. Réception du *challenge*.
L'attaquant reçoit la valeur $t \bmod N$ dont il doit calculer la racine carrée.

6. Procédure de descente.
L'attaquant choisit un relèvement arbitraire $\hat{t} \in \mathbb{Q}$ de t, tel que $\pi_{\mathbb{Q}}(\hat{t}) = t$. En appliquant une procédure de descente, l'attaquant calcule une fraction rationnelle $Q(x)$ satisfaisant les équations 5.4 et 5.5 énoncées plus haut.

7. Algèbre linéaire.
L'attaquant résout le système linéaire inhomogène lui permettant d'obtenir une combinaison multiplicative des $a_i - b_i\alpha$ qui satisfasse l'équation 5.6.

8. Racine e-ème algébrique.
L'attaquant applique l'algorithme de Montgomery pour calculer la valeur $R(\alpha)$ apparaissant dans l'équation 5.6.

9. Calcul final.
En combinant les équations 5.4 et 5.7, l'attaquant obtient :

$$\sqrt[e]{t} = \prod_{p \in \mathcal{F}_r} \sqrt[e]{p}^{u_p} \times \prod_i \sqrt[e]{a_i - b_i m}^{-y_i} \times R(m)^{-1} \bmod N.$$

5.3.3 Analyse

Nous ne reprenons pas *in extenso* les éléments d'analyse de l'algorithme proposé, que l'on peut retrouver dans l'article [T8]. Nous nous limitons à la mention des éléments essentiels. En choisissant comme borne de friabilité

$$B = L_N[1/3, (4/9)^{1/3}],$$

on obtient pour les étapes 3 (collecte de relations) et 7 (algèbre linéaire) de l'énumération qui précède la complexité heuristique

$$t_{\text{rel. collec.}} + t_{\text{linalg}} = L_N[1/3, (32/9)^{1/3} + o(1)].$$

L'autre étape importante pour le calcul est l'étape de descente. Elle nécessite la considération d'expressions $a - bx$ *doublement friables*, exactement comme une descente classique telle qu'analysée en 3.5.3. On obtient ainsi une complexité heuristique bornée par $L_N[1/3, 3^{1/3} + o(1)]$ pour l'amorce de la descente. Les *dernières* étapes de la descente, analysées dans [T8], sont de complexité moindre, d'où

$$t_{\text{descent}} = L_N[1/3, 3^{1/3} + o(1)].$$

D'autre part, les autres étapes de l'algorithme, notamment l'étape de calcul de racine carrée algébrique, sont de complexité négligeable par rapport aux étapes évoquées.

On peut résumer le résultat d'analyse de complexité obtenu par la proposition suivante.

Proposition 5.8. *Soit N un entier, et $\mathcal{O}$ un oracle répondant à la requête $s \leftarrow m^d \bmod N$. On note :*

$$T = L_N[1/3, (32/9)^{1/3} + o(1)],$$
$$N_{\mathcal{O}} = L_N[1/3, (4/9)^{1/3} + o(1)].$$

Alors il est possible en un temps de calcul heuristiquement égal à T d'identifier un ensemble de requêtes m_i à soumettre à l'oracle, de cardinal $N_{\mathcal{O}}$, permettant ensuite, sans accès supplémentaire à l'oracle, de calculer une racine e-ème arbitraire modulo N en temps T.

5.3.4 Variantes

Il est possible de définir des variantes de l'approche précédemment évoquée. En premier lieu, une variante de complexité moins bonne, mais plus « pratique » repose sur l'idée de l'*extension* des bases de facteurs utilisées. Cette idée est détaillée et analysée dans [T8], et a été utilisée pour les résultats expérimentaux mentionnés ci-après.

Une autre variante atteint le même but en restreignant la capacité de l'oracle. Plus exactement, plutôt que de supposer la disponibilité d'un oracle *général* calculant des racines e-èmes, il est possible de se contenter d'un oracle répondant à des requêtes de la forme $\sqrt[e]{P + m}$. Dans un tel cas, l'étape 1 de l'algorithme présenté plus haut impose le choix non plus d'un côté rationnel et d'un côté algébrique, mais de deux côtés algébriques. Ceci n'a pas d'incidence sur la présentation générale de la procédure, mais a en revanche un impact sur la complexité de l'étape de descente, puisque les probabilités de friabilité qui sont rencontrées s'en trouvent modifiées. L'analyse, faite dans [T8], aboutit à une complexité heuristique de la forme $L_N[1/3, 6^{1/3}]$, qui est extrêmement semblable à la complexité du crible algébrique pour factoriser N, à savoir $L_N[1/3, (64/9)^{1/3}]$. L'impact de cette seconde variante est donc extrêmement modeste.

5.3.5 Résultats expérimentaux

L'algorithme proposé plus haut pour le calcul de racines e-èmes arbitraires a été mis en œuvre modulo l'entier du défi RSA-155, avec comme exposant $e = 65\,537$. Bien entendu, le calcul a été mener sans exploiter la factorisation connue de l'entier en question. L'algorithme le permettant, nous avons utilisé un algorithme de sélection polynomiale adapté de l'algorithme GNFS afin de réaliser

l'étape de sélection polynomiale initiale. Nous avons donc les éléments suivante (N et e étant les éléments du contexte) :

$$\begin{aligned}
N = {} & 10\,941\,738\,641\,570\,527\,421\,809\,707\,322\,040\,357\,612\,003\,732\,945 \\
& 449\,205\,990\,913\,842\,131\,476\,349\,984\,288\,934\,784\,717\,997\,257\,891\,267\,332 \\
& 497\,625\,752\,899\,781\,833\,797\,076\,537\,244\,027\,146\,743\,531\,593\,354\,333\,897, \\
e = {} & 65\,537, \\
f = {} & 28200000x^5 - 7229989539851x^3 - 24220733860168568962x^3 - \\
& 64017364896001753866621 32x^2 + \\
& 41178502704727500578312235348 80x + \\
& 74747458114557637077624434699092 9200, \\
g = {} & 14507315380338583x - \\
& 20785833648781819382424015028 7.
\end{aligned}$$

Conformément à l'algorithme, l'étape 3 de collecte de relations a été appliquée uniquement au côté algébrique (polynôme f). En deux heures de temps de calcul sur un processeur AMD Opteron 250 à 2.4 GHz, il a été possible d'obtenir un système surdéterminé reliant les 283 042 idéaux du corps de nombres de norme bornée par $B = 4 \times 10^6$. Suivant la variante d'*extension de base de facteurs* évoquée plus haut et détaillée dans [T8], nous avons complété ce calcul par 44 heures de calcul sur la même machine pour relier une bonne fraction (37%) des idéaux de norme bornée par $B' = 2^{32}$ aux idéaux précédents. Au total, cette étape de précalcul a permis d'isoler un ensemble de 4×10^8 requêtes à l'oracle offrant l'assurance de pouvoir achever le calcul.

L'implémentation de l'étape de descente a été menée en Magma, en utilisant comme routines auxiliaires le programme `lasieve4` (Franke-Kleinjung) et la bibliothèque gmp-ecm (Zimmermann *et al.*). Un total de 53 étapes de descente, pour un temps total d'une heure environ, a permis d'aboutir.

L'étape numéro 7, d'algèbre linéaire, a été menée pour un coût semblable à l'expérience similaire mentionnée en 5.2, soit une vingtaine d'heures de calcul sur des processeurs Intel Core2 6700 à 2.667 GHz.

5.4 Problème de Diffie-Hellman statique

Une extension du travail présenté précédemment, notamment en 5.3, s'applique au contexte du problème de Diffie-Hellman statique. Le scénario cryptographique, dans ce contexte, fait intervenir un acteur dont le rôle est de calculer x^s à partir de l'entrée x, les calculs étant menés dans un groupe cyclique, et s étant un exposant *constant*, connu uniquement de l'oracle. Ce scénario est présent dans divers protocoles cryptographiques, au nombre desquels on peut par exemple citer le protocole de signature de Chaum-van Antwerpen [42], le chiffrement d'ElGamal [67], ou le protocole d'authentification par mot de passe de Ford-Kaliski [75].

Nous pouvons appliquer les techniques développées dans les paragraphes précédents à ce contexte, lorsque le groupe choisi est le groupe multiplicatif d'un corps fini. L'acteur que nous venons d'évoquer est employé comme oracle réalisant une exponentiation dans le groupe, et l'objectif de l'attaquant est, après un certain nombre de requêtes à cet oracle, de simuler son comportement sur une valeur cible qui lui est inconnue au moment des requêtes.

Nous choisissons de ne pas répéter ici *in extenso* les travaux présentés dans [T12], et nous concentrons sur un cas significatif de cette approche.

5.4.1 Exemple dans le contexte FFS

Nous considérons le cas où le groupe utilisé est le groupe multiplicatif d'un corps de petite caractéristique, typiquement de caractéristique 2 : soit $K = \mathbb{F}_{2^n}$, et $G = \mathbb{F}_{2^n}^*$. L'adaptation naturelle à ce contexte du processus déjà détaillé amène à considérer l'algorithme FFS en lieu et place de l'algorithme NFS. Ainsi, on souhaite obtenir deux polynômes bivariés f et g dont le résultant (par rapport à l'une des variables) admet pour facteur un polynôme irréductible de degré $n = [K : \mathbb{F}_2]$.

Il est utile de se placer dans le cas spécial de l'algorithme FFS avec deux côtés rationnels, tels que présenté en 3.4.5. Nous rappelons que dans ce contexte, le couple de polynômes choisi a la forme particulière suivante :

$$f(x, y) = y - u(x), \quad g(x, y) = x - v(y).$$

Dans ce contexte, les deux structures algébriques qui interviennent sont toutes deux des *côtés rationnels*, isomorphes à l'anneau de polynômes $\mathbb{F}_2[t]$. Il existe, pour chacune, une image univoque des idéaux premiers (polynômes irréductibles) vers les éléments du corps cible, ici $K = \mathbb{F}_{2^n}$. Nous adoptons les notations déjà utilisées en 3.4.5 pour la notation des applications correspondantes :

$$i_f : \begin{cases} \mathbb{F}_2[x, y]/f(x, y) \cong \mathbb{F}_2[x] & \longrightarrow & K, \\ x & \longmapsto & \rho, \end{cases}$$
$$i_g : \begin{cases} \mathbb{F}_2[x, y]/g(x, y) \cong \mathbb{F}_2[y] & \longrightarrow & K, \\ y & \longmapsto & \sigma. \end{cases}$$

D'autre part ces images ρ et σ vérifient $\rho = v(\sigma)$ et $\sigma = u(\rho)$.

Les bases de facteurs, de chaque côté, sont constituées de l'ensemble des polynômes irréductibles en-deçà d'un certain degré. On définit donc :

$$\mathcal{F}_f = \{\phi(x) \in \mathbb{F}_2[x],\ \phi \text{ irréductible},\ \deg \phi \leq B\},$$
$$\mathcal{F}_g = \{\psi(y) \in \mathbb{F}_2[y],\ \psi \text{ irréductible},\ \deg \psi \leq B\}.$$

Pour les éléments de chacune de ces bases de facteurs, il est possible de calculer explicitement l'élément de K correspondant au moyen des applications i_f et i_g. Dès lors, l'ensemble des requêtes à soumettre à l'oracle peut être défini comme l'ensemble de ces valeurs :

$$\{\phi(\rho),\ \phi \in \mathcal{F}_f\} \cup \{\psi(\sigma),\ \psi \in \mathcal{F}_g\}.$$

Considérons maintenant la procédure de descente. En reprenant les analogues des équations 5.4 et 5.5, partant d'un relèvement arbitraire $\hat{t}$ de la valeur cible t, satisfaisant $\hat{t}(\rho, \sigma) = t$, on aboutit à :

$$Q(x, u(x))\hat{t} = \prod_{\phi \in \mathcal{F}_f} \phi(x)^{k_\phi},$$
$$Q(v(y), y) = \prod_{\psi \in \mathcal{F}_g} \psi(y)^{\ell_\psi}.$$

En récrivant ces grandeurs dans K, on obtient :

$$\hat{t}(\rho, \sigma) = t \prod_{\phi \in \mathcal{F}_f} \phi(\rho)^{k_\phi} \prod_{\psi \in \mathcal{F}_2} \psi(\sigma)^{-\ell_\psi}.$$

Ceci est donc suffisant pour conclure. Dans ce cas précis, considérer deux côtés rationnels permet d'éliminer les deux étapes les plus coûteuses du calcul, qui sont la collecte de relations, et l'algèbre linéaire.

Il résulte de ce cas particulier une complexité bien meilleure que dans les autres cas. Comme détaillé dans [T12], et en reprenant les éléments d'analyse déjà employés à diverses reprises, on aboutit à un nombre de requêtes à l'oracle en $L_{2^n}[1/3, (4/9)^{1/3} + o(1)]$, pour un temps de calcul (exclusivement le temps de la descente) heuristiquement égal à $L_{2^n}[1/3, (4/9)^{1/3} + o(1)]$.

5.4.2 Résultats expérimentaux

Nous reprenons le résultat expérimental indiqué dans [T12]. Cette procédure a été mise en place dans le corps fini $\mathbb{F}_{1025}$, avec le choix de polynômes suivant :

$$f(x) = y - (x^{171} + x^4 + x^3 + x^2 + 1), \quad g(y) = x - (y^6 + y + 1).$$

On suppose un nombre d'accès à l'oracle égal à 80×10^6 (polynômes irréductibles de degré au plus 29 de chaque côté). Chaque étape de la descente a nécessité environ cinq minutes de calcul sur un processeur Intel Core2 à 2.667 GHz. L'ensemble de la descente a nécessité environ 2000 étapes, pour un coût total de l'ordre d'une semaine.

Chapitre 6

RSA-768 : calculs sur une grille

Ce chapitre reprend quelques éléments factuels relatifs à la factorisation de RSA-768, que nous avons achevée fin 2009 grâce à un effort conjoint avec différents partenaires, notamment de l'ÉPFL de Lausanne, et de NTT au Japon. Ce record de factorisation a reçu une assez large publicité, et a fait l'objet de l'article [T13].

6.1 Éléments de contexte

RSA-768 désigne un nombre composé parmi une liste publiée par l'entreprise RSA Labs dans le cadre d'un défi de factorisation appelé *challenge RSA*. Les nombres de cette liste sont des nombres composés de la forme $N = pq$, où les facteurs premiers p et q (non publiés) sont de la même taille. Dans le cas particulier de RSA-768, on a un nombre de 768 bits (232 chiffres décimaux), les facteurs inconnus p et q occupant 384 bits (116 chiffres décimaux). Le nombre N pour RSA-768 est :

$$\begin{aligned} N = 1\,230\,186\,684\,530\,117\,755\,130\,494\,958\,384\,962\,720\,772\,853\,569\,595\,334\,792\,197\,322\,452\,151 \\ 726\,400\,507\,263\,657\,518\,745\,202\,199\,786\,469\,389\,956\,474\,942\,774\,063\,845\,925\,192\,557\,326 \\ 303\,453\,731\,548\,268\,507\,917\,026\,122\,142\,913\,461\,670\,429\,214\,311\,602\,221\,240\,479\,274\,737 \\ 794\,080\,665\,351\,419\,597\,459\,856\,902\,143\,413. \end{aligned}$$

Jusqu'en 2007, RSA a offert des primes récompensant les factorisations trouvées parmi les nombres de la liste du *challenge RSA*. Malgré l'arrêt de cette mécanique de récompense, cette liste est demeurée une liste de référence généralement acceptée indiquant des nombres connus comme étant difficiles à factoriser (la façon dont ils sont générés, avec p et q de la même taille, voue à l'échec une approche comme ECM, à même de bénéficier de l'éventuelle petite taille d'un facteur premier). Les nombres ayant été générés par l'entreprise RSA, l'auteur d'une factorisation peut raisonnablement être présumé ne pas avoir gagné la connaissance des facteurs premiers p et q autrement qu'en ayant effectivement résolu le problème de la factorisation de N, c'est-à-dire avoir attaqué le problème par la face nord. Pour cette raison, la liste du *challenge RSA* est le choix privilégié pour les exemples de factorisation par le crible algébrique.

Notre présentation du crible algébrique a montré combien les étapes de cet algorithme sont nombreuses, ce qui rend difficile une extrapolation pour déterminer le temps de calcul nécessaire à la factorisation d'un nombre de cet acabit. Pour déterminer la limite, en terme de factorisation par la méthode du crible algébrique, de ce qui est calculable, la pratique reste l'approche de choix.

Une factorisation comme RSA-768 amène à se pencher sur les politiques de choix des tailles de clé pour l'algorithme RSA, dans les contextes où il est utilisé. Ces contextes ont des caractéristiques distinctes. Dans le cadre des serveurs SSL (notamment http), les machines qui communiquent sont rarement bloquées par le poids des calculs cryptographiques. Aussi il y a peu de bonnes raisons de rencontrer des clés sous-dimensionnées. Ceci, hélas, ne veut pas dire qu'elles ne le sont pas. Début 2011, 75% des clés RSA des serveurs Web SSL rencontrés sur Internet sont des clés RSA-1024, contre

20% RSA-2048, et 2% RSA-512 (!). Même si la lenteur de la migration de RSA-1024 à des tailles de clés plus importantes peut être jugée préoccupante, l'établissement d'un nouveau record de factorisation à 768 bits ne cause pas de péril immédiat.

Un contexte plus délicat est celui des technologies embarquées. Une carte de crédit EMV effectue une opération de signature RSA, et le temps de cette opération dépend naturellement de la taille de clé. En outre, les ressources matérielles dont doit disposer la carte (mémoire, registres) sont conditionnées par la taille de clé. Cette dernière se traduit donc en coûts de déploiements. Il est donc attendu que dans un tel contexte, les clés soient dimensionnées au plus juste, et en accord avec la durée de validité des cartes émises. Nous avons été en mesure de vérifier sur des cartes bancaires EMV la taille des clés employées, sans hélas qu'il nous soit possible d'obtenir un panorama de l'évolution des tailles de clé rencontrées dans les différentes cartes. En 2012, l'immense majorité des clés publiques rencontrées sur les cartes bancaires que nous avons testées sont des clés de 960 bits. Quelques rares cartes ont des clés de 1024 bits. Nous avons rencontré sur une carte expirant fin 2009 une clé de 896 bits. Ces tailles de clé sont dimensionnées au plus juste, mais correctes. Avant qu'une clé de 960 bits puisse être factorisée (probablement pas avant 2015), ces cartes auront expiré. D'autre part, la mise en regard des moyens à déployer pour accomplir une telle factorisation et le risque que pourrait constituer une falsification de carte n'amène pas d'inquiétude immense. En tout état de cause, ce dimensionnement au plus juste ne peut être réalisé qu'à la condition expresse de disposer de données expérimentales sur l'état de l'art en termes de factorisation. C'est l'objectif de notre travail sur RSA-768.

6.2 Mode de travail

Pour mener à bien la mission de factorisation de RSA-768, plusieurs équipes se sont associées.

- L'initiative d'origine est partie de T. Kleinjung, en post-doc à l'université de Bonn en 2007. T. Kleinjung, en collaboration avec J. Franke, est le détenteur du record de factorisation d'entiers depuis 2002, et possède bien sûr une forte expertise en la matière, ainsi que de l'implémentation la plus efficace de toutes les étapes de l'algorithme.

- Au moment du début du projet, T. Kleinjung a effectué une mobilité vers Lausanne pour rejoindre le laboratoire Lacal de A. K. Lenstra. Ce dernier a toujours été un acteur important sur le thème de la factorisation depuis les années 1990.

- Le projet Cacao de l'INRIA, dont nous faisons partie, a apporté son expérience sur des projets similaires, sur l'algèbre linéaire, l'étape de racine carrée, ainsi que des ressources de calcul dont nous parlons un peu dans ce chapitre.

- K. Aoki de NTT, auteur de quelques factorisations d'importance, a contribué à l'effort de calcul.

- H. te Riele et P. Montgomery au CWI ont participé au projet, dans la suite de la forte implication du CWI dans les records de factorisation par le passé.

- D'autres contributeurs isolés ont participé au projet.

La « direction » du projet a été essentiellement assumée par l'équipe du Lacal. Toutefois la coordination des tâches est restée lâche. L'importance de ce point dépasse l'aspect organisationnel. En effet, il a toujours été totalement inenvisageable d'installer un mode de distribution des calculs automatisés, la diversité des contributeurs ne le permettant pas.

À la différence d'une approche qui aurait pu être envisagée, à savoir un projet massif impliquant de nombreux contributeurs (type BOINC [29], voire « à l'ancienne » par courrier électronique comme cela se faisait dans les années 1990 [137]), l'objectif choisi a été de se concentrer sur un nombre restreint de contributeurs expérimentés et autonomes. Ce choix permet de réduire la tâche de coordination.

6.3 Étapes de l'algorithme

Nous donnons quelques éléments sur les choix faits pour les différentes étapes de l'algorithme.

6.3.1 Séléction polynomiale

La sélection polynomiale pour RSA-768 a été effectuée assez largement en amont du début du projet (2005), en utilisant l'algorithme de Kleinjung [124]. Les 20 années-cœur[1] utilisées en 2005 pour mener cette sélection polynomiale ont été complétées par le même effort avec du matériel plus récent au printemps 2007, sans fournir de meilleur couple de polynômes. La paire choisie est :

$$\begin{aligned} f(x) =& 265482057982680x^6 \\ &+ 1276509360768321888x^5 \\ &- 5006815697800138351796828x^4 \\ &- 4647785447172785427177267745 0x^3 \\ &+ 6525437261935989397109667371894785x^2 \\ &- 1818577935208859435672601886243480 3054x \\ &- 2775652667915438819952161997138011033431 20 \\ g(x) =& 346610035504925018514458 29x \\ &- 1291187456580021223163547791574810881 \end{aligned}$$

Le polynôme f a été sélectionné pour ses bonnes propriétés pour le crible algébrique. Ses coefficients sont petits, puisqu'ils occupent au total moins de 660 bits (contre 768 bits pour un polynôme pris « au hasard »), et le nombre de racines de ce polynôme est élevé modulo p pour les petits nombres premiers p. Plus précisément, la valuation moyenne de $\mathrm{Norm}_{K/\mathbb{Q}}(a - b\alpha)$ en p est élevée, où $K = \mathbb{Q}(\alpha)$ désigne le corps de nombres défini par f. Cette valuation moyenne peut être rapprochée du nombre de zéros de f sur la droite projective $\mathbb{P}^1(\mathbb{F}_p)$ (prenant ainsi aussi en compte les racines « à l'infini », liées au terme de tête), mais se limiter à ce point de vue ne tiendrait pas suffisamment compte des phénomènes de ramification. Pour mesurer l'exceptionnalité du polynôme f, nous indiquons dans la table 6.1 le gain sur la valuation moyenne obtenue en considérant les entiers $\mathrm{Norm}_{K/\mathbb{Q}}(a - b\alpha)$ par rapport à des entiers aléatoires. Un raffinement de cette mesure amène à considérer la grandeur $\alpha(f)$, qui apparaît dans les travaux de sélection polynomiale comme étant un indicateur des « propriétés des racines ». Cette grandeur, introduite par [162] n'est pas étudiée dans ce mémoire.

[1] L'unité « année-cœur » que nous utilisons désigne une année de calcul sur un cœur d'un processeur « typique » au moment où le calcul a été mené. Dans la plupart des cas cités dans ce chapitre, un processeur 64 bits équipé d'un processeur à 2 GHz est une bonne mesure. On utilise parfois le terme « année-CPU » pour désigner exactement la même quantité.

p	$\mathrm{E}[\nu_p]$	p	$\mathrm{E}[\nu_p]$	p	$\mathrm{E}[\nu_p]$	p	$\mathrm{E}[\nu_p]$	p	$\mathrm{E}[\nu_p]$	p	$\mathrm{E}[\nu_p]$
2	+232%	23	+286%	59	-1%	97	+97%	137	-1%	179	-1%
3	+249%	29	+385%	61	+392%	101	+98%	139	+98%	181	-1%
5	+332%	31	+386%	67	+196%	103	+98%	149	+98%	191	+98%
7	+349%	37	+291%	71	-1%	107	-1%	151	+197%	193	+98%
11	+90%	41	-1%	73	+97%	109	+98%	157	+297%	197	-1%
13	+456%	43	+96%	79	-1%	113	+197%	163	+197%	199	-1%
17	+465%	47	+96%	83	+97%	127	+395%	167	+197%		
19	+379%	53	+391%	89	+295%	131	+98%	173	+297%		

TAB. 6.1 : Valuation moyenne en p pour $\mathrm{Norm}_{K/\mathbb{Q}}(a - b\alpha)$ (a et b premiers entre eux) avec le polynôme f de RSA-768, comparée à un entier aléatoire.

6.3.2 Crible

Pour l'étape de crible, seul l'algorithme du *lattice sieve* [174] a été utilisé. Les *special-q* ont été considérés du côté algébrique (ce choix est guidé par le fait que la norme algébrique est la plus importante de ce côté), pour q appartenant à la plage $[450 \times 10^6, 11.1 \times 10^9]$, soit un total d'environ 480×10^6 idéaux premiers $\mathfrak{q}$ représentés sous la forme $\mathfrak{q} = \langle q, \alpha - r \rangle$.

Le grain le plus fin de la tâche de crible est l'étude d'un *special-q*, c'est-à-dire d'un idéal premier. Pour des raisons pratiques, ces tâches sont regroupées en lots, mais elles demeurent indépendantes. Pour chaque *special-q*, on recherche les paires (a, b) satisfaisant la condition $\mathfrak{q} \mid (a - b\alpha)$, et telles qu'à la fois $(a - b\alpha)\mathfrak{q}^{-1}$ et $(a - bm)$ apparaissent friables au regard de la base de facteurs choisie, en autorisant éventuellement quelques *large primes*. Les paramètres choisis dépendent des machines utilisées.

- Sur les machines disposant de seulement 1 Go de RAM par cœur, nous avons criblé sur une base de facteur limitée à 450×10^6 du côté algébrique, et 100×10^6 du côté rationnel.
- Sur les machines disposant de 2 Go de RAM par cœur, les limites correspondantes ont été fixées à 1.1×10^9 et 200×10^6.

Dans les deux cas, la cofactorisation a été tentée pour des cofacteurs inférieurs à 2^{140} côté algébrique, et 2^{110} côté rationnel, autorisant des cofacteurs de taille jusqu'à 2^{40}.

Cette étape de crible a permis de collecter 64 milliards de relations, occupant 5 To de données.

Ressources utilisées, et mode d'utilisation

L'étape de crible est la plus massivement parallélisable de l'algorithme. Les partenaires du projet ont mis à contribution l'ensemble des ressources de calcul qu'il a été possible de mobiliser.

Lorsqu'il s'est agi de plate-formes de calcul partagées, ceci a nécessité un comportement responsable vis-à-vis des autres utilisateurs, notamment au moyen de tâches sous-prioritaires appelées *best-effort*.

Nous décrivons ici les mécanismes d'organisation du crible, que nous avons aussi relatés dans [T16]. Rien n'est extrêmement original dans cette approche, concentrée sur la simplicité pour atteindre l'objectif donné. Notre description est organisée du point de vue du partenaire CACAO, dont nous étions membre. Au sein de CACAO, quatre personnes se partagaient la supervision de huit sites de la plate-forme `Grid5000`. Chacun de ces sites héberge deux à cinq clusters homogènes, les caractéristiques

des différents clusters étant variables. Un cluster « typique » au moment du calcul RSA-768 comporte environ 20 nœuds avec environ quatre cœurs par nœud, et accède à un volume NFS partagé au niveau du site.

Plutôt que de distribuer les tâches *special-q* par *special-q*, ce qui se serait avéré particulièrement incommode, nous avons opté pour une allocation des plages de travail hiérarchisée, comme nous l'avons décrit dans [T16]. L'attribution des *special-q* dans la plage $[450 \times 10^6, 11.1 \times 10^9]$ qui a été traitée a été effectuée comme suit. Nous décrivons le processus du point de vue des programmes lancés par le partenaire Cacao, les autres partenaires adaptant une procédure potentiellement différente à partir de l'étage 2.

1. À la requête d'un partenaire, l'équipe de l'Épfl lui alloue une plage sous la forme d'un intervalle de largeur 100×10^6 au minimum, ou un multiple entier de cette largeur. Une telle plage correspond à une charge de travail importante, de l'ordre de 10 à 15 années-cœur, pour un volume de données à produire de l'ordre de 50 à 70 Go. Cette allocation se fait par courrier électronique, à raison d'une fois toutes les deux semaines environ.

2. À partir de cette étape, le choix d'organisation fait par les différents partenaires peut diverger, en fonctions des particularités de la ressource de calcul employée. Pour Cacao, des sous-intervalles de largeur 5×10^6 à 20×10^6 sont réservés (à la main, une fois tous les quelques jours au plus) par les différents participants individuels supervisant l'avancement des tâches sur les clusters employés. La réservation d'un tel sous-intervalle implique sa subdivision en plages, dimensionnées pour contenir un nombre constant de *special-q*, correspondant à une charge de travail d'une heure CPU environ. Ces plages sont listées sous la forme de fichiers de taille nulle ajoutés dans un répertoire `inqueue/`, où le nom de fichier encode la plage. Ce choix est privilégié pour garantir l'atomicité des opérations de déplacement et renommage de fichier dans un contexte où le système de fichier NFS est très présent.

3. Les tâches présentes dans la file de tâches de l'ordonnanceur du cluster utilisé sont par essence sous-prioritaires. Leur instant de démarrage est imprévisible, et elles sont susceptibles d'être tuées sans préavis. Il est capital de prendre en compte cette caractéristique. Au démarrage d'une tâche, réservant typiquement un petit nombre de nœuds, chaque cœur utilisé tente d'acquérir un fichier du répertoire `inqueue/` de manière atomique. La seule opération atomique garantie par le protocole NFS étant l'instruction de renommage `mv`, cet essai se résume à la tentative de déplacer un fichier de `inqueue/` vers un autre répertoire. En cas de succès, le programme de crible démarre, produisant sa sortie dans un répertoire `working/` sur le volume NFS. Si le renommage échoue, alors la tâche correspondante a déjà été acquise par un autre client. Il convient donc de tenter d'acquérir un nouveau fichier.

4. En cas d'achèvement sans erreur du programme de crible, le fichier de résultats est compressé, déplacé vers un autre répertoire, et le fichier décrivant la plage est déplacé vers un autre répertoire ou celle-ci est donc comprise comme achevée. Le cœur en question sollicite alors une nouvelle plage comme précédemment.

5. En cas d'interruption de la tâche (dans son ensemble), à la discrétion de l'ordonnanceur de tâches, plusieurs plages se retrouvent incomplètement traitées. Un automate écrit en `perl` est exécuté régulièrement pour nettoyer ces plages, récupérer et archiver le travail fait, et replacer dans `inqueue/` un descriptif de la sous-plage restant à traiter.

6. À l'achèvement d'un sous-intervalle (au sens de l'étape 2) sur un cluster, les résultats sont rapatriés vers les machines du projet Cacao, et assemblés en un petit nombre fichiers.

INRIA/Cacao	24 322 274 179	37.97%
ÉPFL	19 080 641 425	29.79%
NTT	9 614 969 474	15.01%
Bonn	5 212 423 746	8.14%
H. Stockinger	2 906 539 451	4.54%
CWI	2 202 786 036	3.44%
P. Leyland	443 023 506	0.69%
S. Contini	278 083 916	0.43%

Tab. 6.2 : Contributions des différents partenaires au crible pour RSA-768.

7. À l'achèvement d'un intervalle (au sens de l'étape 1), les fichiers correspondants sont transférés vers l'Épfl qui centralise les données.

L'article [T16] contient de plus amples détails sur les procédures employées pour l'organisation du crible.

L'accumulation des 64 milliards de relations a ainsi été l'aboutissement d'un très grand nombre de tâches, pour un coût calculatoire total estimé à environ 1500 années-cœur. La répartition des contributions pour le crible enter les différents partenaires est donnée par la table 6.2.

6.3.3 Filtrage

Les 5 To de relations collectées, impliquant des *large primes* jusqu'à une borne de 2^{40}, sont ensuite passés par l'étape de filtrage. L'objectif de l'étape de filtrage est multiple :

- Élimination des éventuels doublons[2].
- Réduction à un sous-ensemble des relations et des *large primes* permettant d'obtenir une dépendance linéaire.
- Production d'une matrice pour l'étape de résolution de système linéaire.

Cette étape a été entièrement réalisée à l'Épfl. La table 6.3 rapporte quelques éléments quantitatifs du filtrage. Notons que le filtrage a en fait été réalisé plusieurs fois, afin de tester la pertinence de divers choix de paramètres pour optimiser le temps de la résolution du système linéaire. Le grand nombre de relations collecté a offert cette flexibilité, de sorte à obtenir une matrice plus « facile » à traiter qu'originellement attendu. Les tests réalisés ont montré qu'il aurait en fait été suffisant de n'autoriser les *large primes* que jusqu'à une borne de l'ordre de 2^{34} pour obtenir une matrice (qui aurait toutefois été plus difficile à traiter). Limiter ainsi les *large primes* aurait permis un gain considérable sur la place occupée par les relations, et un gain significatif mais modeste sur le temps de crible (10 à 15%). Cette perte relative a été compensée par le gain sur l'étape d'algèbre linéaire.

6.3.4 Algèbre linéaire

L'étape d'algèbre linéaire a été menée de manière distribuée sur différents clusters, comme relaté dans l'article [T14]. Nous reprenons des éléments relatifs à la résolution de ce système linéaire en 13.3.

[2]Outre les aspects organisationnels inévitables, des raisons naturelles amènent l'algorithme à produire des relations en double. De telles relations sont inutiles.

Relations	64 334 489 730	
Doublons	17 629 469 788	(27.4%)
Relations uniques	46 705 019 942	
« Free relations »	57 223 462	
Relations sans singletons	2 458 287 361	
Idéaux sans singletons	1 697 618 199	
relation-sets après filtre (= lignes de la matrice)	192 796 550	
idéaux après filtre (= colonnes de la matrice)	192 795 550	

Tab. 6.3 : Données du filtrage pour RSA-768.

À l'issue de l'étape d'algèbre linéaire, les 512 solutions produites ont été croisées avec 128 caractères quadratiques (voir section 2.1.4), pour produire 460 « vraies » solutions. Le fait que 128 caractères n'aient réduit que de 52 la dimension de l'espace de solution est un bon indice que ces caractères engendrent correctement le dual $\left(W/(K^*)^2\right)^*$ défini en 2.1.4.

6.3.5 Racine carrée

Nous avons donné au chapitre 4 différentes méthodes pour mener à bien le calcul de racine carrée pour le crible algébrique. Les différentes méthodes ont été testées (sauf la méthode de Couveignes s'appliquant au degré impair). L'implantation par F. Bahr et T. Kleinjung de l'algorithme de Montgomery décrit en 4.1.3 a permis de calculer une solution en environ 4h de calcul sur 144 cœurs (12 nœuds bi-processeur AMD Opteron 2427, 16 Go de RAM par nœud).

Nous avons aussi testé l'approche dite « naïve » par relèvement, décrite en 4.1.1. Pour le calcul de la racine carrée rationnelle, 2h30 de calcul sur un cœur se sont avérées suffisantes avec cette méthode. Pour la racine carrée algébrique, la consommation en mémoire de cette méthode par relèvement nous a dissuadé de mesurer le temps pris (48 Go, la limite qui nous étaient accessible à l'époque, ne suffisant pas).

L'approche par restes chinois que nous avons développée et décrite en 4.2 a aussi été utilisée, à titre d'essai. Cette méthode a permis d'obtenir le résultat en six heures de calcul sur 144 cœurs (18 nœuds bi-processeur Intel Xeon E5520, 32 Go de RAM par nœud). Dans la mesure où nous reconnaissons que le paramétrage de notre implantation de cet algorithme a été peu optimisé, nous considérons cet expérience comme validant l'intérêt pratique de cette approche.

6.4 Étape suivante

Il est naturel de planifier quelle pourrait être la prochaine entreprise de calcul de cette ampleur. La prochaine « grande » cible est RSA-1024, puisque le déploiement de cette taille de clé est aujourd'hui encore considérable. Toutefois, un calcul de cette ampleur ne peut à ce jour être entrepris en employant une quantité de ressources comparables à ce que nous avons mis en œuvre pour RSA-768. Une factorisation intermédiaire interviendra. Il est vraisemblable que ce choix se porte sur le nombre RSA-896, à la fois car il apparaît après RSA-768 sur la liste du *challenge RSA*, et aussi car une telle taille semble constituer une étape raisonnable entre RSA-768 et RSA-1024. À notre connaissance, une telle entreprise n'est pas entamée, mais cette situation peut assez bien ne pas durer.

D'autres projets connexes utilisant le crible algébrique sont des cibles raisonnables. Le record de l'algorithme SNFS, actuellement à 1024 bits, peut clairement être battu. Un tel calcul au-delà de 1024 bits, même s'il s'agit dans le cadre de SNFS de nombres « spéciaux », permettrait sans doute une meilleure prise de conscience par les acteurs de l'Internet de l'obsolescence des clés de 1024 bits. Enfin, dans le contexte du logarithme discret, l'état de l'art des calculs avec les algorithmes NFS-DL ou FFS est à ce jour bien en-deçà de 1024 bits, et des calculs de « rattrapage » sont à attendre, de 900 à 1024 bits.

Partie II

Courbes

Chapitre 7

Courbes et logarithme discret

Les travaux que nous avons menés dans le contexte du logarithme discret sur les courbes ont fait l'objet de trois articles [T7, T9, T15]. Deux axes sont à l'origine de ces travaux. Le premier axe, porté par [T7, T9], a trait à l'étude de la complexité de la variante *à genre fixé* de l'algorithme de Adleman-DeMarrais-Huang [5], dans la suite du travail débuté par Gaudry dans l'article [86]. Dans un second temps l'étude fine des procédures de *descente* dans les algorithmes de calcul de logarithme discret a donné lieu à l'article [T15], premier algorithme obtenant une complexité sous-exponentielle $L[1/3]$ pour résoudre le logarithme discret dans une classe de courbes raisonnablement large[1]. Avant de proposer une synthèse de ces travaux et un rapide aperçu de leurs antécédents et extensions, ce chapitre présente le contexte général du problème du logarithme discret sur les courbes, ainsi que les définitions nécessaires.

7.1 Définitions

L'introduction à la thématique des courbes algébriques qui occupe les premiers paragraphes de cette partie n'a aucunement une vocation de présentation précise et complète, mais simplement aide à fixer quelques éléments de vocabulaire. La totalité des concepts abordés dans cette partie du mémoire sont introduits plus complètement dans les ouvrages de référence, par exemple [198].

Nos objets de départ sont les courbes algébriques sur un corps k, qui sont des variétés projectives absolument irréductibles de dimension 1. Une courbe $\mathcal{C}$ définie sur un corps k est donnée par une équation implicite telle :

$$\mathcal{C} : \quad F(x, y, z) = 0,$$

où les coefficients du polynôme homogène F sont dans k. Par commodité d'écriture, on écrit aussi l'équation d'une telle courbe communément sous une forme affine $F(x, y) = 0$, désignant l'expression homogène $z^{\deg F} F(x/z, y/z) = 0$. Le point de vue est ainsi déplacé de la courbe projective vers un modèle affine, toutefois ce sont bel et bien les propriétés du modèle projectif qui sont étudiées. Les points de $\mathcal{C}$ désignent les solutions dans $\mathbb{P}^2(\bar{k})$ de l'équation de définition sur une clôture algébrique $\bar{k}$ de k, tandis que les points désignés explicitement comme k-rationnels sont les solutions sur k. On peut aussi voir ces derniers comme les points de $\mathcal{C}$ stables sous l'action de groupe de Galois $\mathrm{Gal}(\bar{k}/k)$. On supposera toujours que les courbes considérées disposent d'un point k-rationnel. Les courbes qu'on considère ici sont toujours lisses (sans point singulier). Toutefois, celles-ci peuvent parfois être présentées sous formes de modèles singuliers, par exemple à l'infini. Le processus de résolution des singularités est alors sous-entendu, et écarté de la présentation.

[1] On entend par là qu'on exclut par exemple l'attaque MOV sur une famille de courbes elliptiques supersingulières.

7.1.1 Diviseurs, fonctions

Le groupe des diviseurs $\mathrm{Div}(\mathcal{C})$ de $\mathcal{C}$ est le groupe abélien libre engendré par les points de $\mathcal{C}$. Un diviseur est donc une somme formelle de points, s'écrivant

$$D = \sum_{P \in \mathcal{C}} n_P(P),$$

où les entiers relatifs n_P sont tous nuls à l'exception d'un nombre fini. Le degré est la somme des coefficients n_P.

Un diviseur est dit *effectif*, ce qu'on note $D \geq 0$, lorsque tous les coefficients n_P vérifient $n_P \geq 0$. Pour deux diviseurs, on note $D \geq E$ lorsque le diviseur $D - E$ est effectif. Le *support* d'un diviseur est l'ensemble (fini) des points de $\mathcal{C}$ pour lesquels on a $n_P \neq 0$. Un sous-groupe important de $\mathrm{Div}(\mathcal{C})$ est le groupe constitué par les diviseurs de degré zéro, noté $\mathrm{Div}^0(\mathcal{C})$.

Le *poids* d'un diviseur, lorsqu'on est en présence d'un diviseur effectif, est identique à son degré. Le poids d'un diviseur non effectif est le poids de sa partie effective.

Les *fonctions* sur la courbe $\mathcal{C}$ sont les quotients de polynômes homogènes de $\bar{k}[x, y, z]$ de même degré, modulo la relation d'équivalence définie par $F : f/g \sim f'/g'$ si et seulement si F divise $fg' - f'g$. Les fonctions sur $\mathcal{C}$ constituent le *corps de fonctions* de $\mathcal{C}$, noté $\bar{k}(\mathcal{C})$. Les fonctions sur $\mathcal{C}$ définies sur k sont définies de façon similaire.

Une fonction non constante sur $\mathcal{C}$ a naturellement des zéros et des pôles ; par exemple sur une courbe elliptique donnée par une équation affine $y^2 = x^3 + ax + b$, la fonction y (qui correspond, projectivement, à y/z), a un pôle en le point $(0 : 1 : 0) \in \mathbb{P}^2$, et des zéros aux points de coordonnées $(\alpha : 0 : 1)$, où α est une racine de x^3+ax+b. À un tel zéro ou pôle P, on associe une multiplicité [198, p. 9, 22] notée $\mathrm{ord}_P(f)$. Le diviseur d'une fonction $f \in \bar{k}(\mathcal{C})$ est

$$\mathrm{div}(f) = \sum_{P \in \mathcal{C}} \mathrm{ord}_P(f)(P).$$

Un tel diviseur, construit à partir d'une fonction de $\bar{k}(\mathcal{C})$, est appelé *diviseur principal*. L'ensemble des diviseurs principaux constitue un sous-groupe $\mathrm{Prin}(\mathcal{C})$ de $\mathrm{Div}^0(\mathcal{C})$, puisque le degré d'un diviseur principal est nul. Plus généralement, au sein de $\mathrm{Div}(\mathcal{C})$, on peut définir une relation d'équivalence entre diviseurs appelée *équivalence linéaire*, qu'on note $D \sim D'$ lorsque $D - D'$ est un diviseur principal (en particulier, l'équivalence linéaire conserve trivialement le degré).

7.1.2 Jacobienne

La jacobienne $\mathrm{Jac}_{\mathcal{C}}$ de la courbe $\mathcal{C}$ est le quotient $\mathrm{Div}^0(\mathcal{C})/\mathrm{Prin}(\mathcal{C})$, muni d'une structure de variété algébrique projective. Les diviseurs k-rationnels de $\mathrm{Jac}_{\mathcal{C}}$, qu'on note $\mathrm{Jac}_{\mathcal{C}}(k)$, sont les diviseurs stables sous l'action du groupe de Galois $\mathrm{Gal}(\bar{k}/k)$. Ces diviseurs k-rationnels peuvent être représentés comme sommes formelles de points, mais il convient de remarquer que les points intervenant dans cette somme ne sont pas tous nécessairement à coordonnées dans k, puisque seule la stabilité du diviseur est nécessaire.

La jacobienne d'une courbe a une structure de groupe algébrique, et est donc une variété abélienne. Les calculs de la cryptographie utilisant les courbes algébriques ont lieu précisément dans ce groupe (plus précisément, dans le groupe des diviseurs k-rationnels de la jacobienne, le corps k étant un corps fini dans les applications les plus courantes).

7.1.3 Genre d'une courbe

Un invariant important des courbes algébriques est leur genre. Il est défini au moyen du théorème de Riemann-Roch. Soit $D = \sum_{P \in \mathcal{C}} n_P(P) \in \mathrm{Div}(\mathcal{C})$ et $f \in k(\mathcal{C})$. On s'intéresse à la condition $D + \mathrm{div}(f) \geq 0$. Cette propriété impose une contrainte sur la fonction f en chacun des points P de $\mathcal{C}$ pour lesquels on a $n_P \neq 0$. Pour $n_P > 0$, il est nécessaire qu'un éventuel pôle de f en P ait une multiplicité au plus égale à n_P. Pour $n_P < 0$, il est nécessaire que f possède un zéro en P de multiplicité au moins égale à $-n_P$.

Soit $K_{\mathcal{C}}$ la classe de diviseurs de $\mathrm{Div}(\mathcal{C})$ (modulo équivalence linéaire) donnée par les diviseurs des formes différentielles [198, § II.4]. On appelle $K_{\mathcal{C}}$ *diviseur canonique*. Le $\bar{k}$-espace vectoriel de $\bar{k}(\mathcal{C})$ constitué des fonctions telles que $D + \mathrm{div}(f) \geq 0$ est appelé *espace de Riemann* associé à D, et noté $\mathcal{L}(D)$ (la définition impose d'inclure la fonction nulle dans $\mathcal{L}(D)$) :

$$\mathcal{L}(D) = \left\{ f \in \bar{k}(\mathcal{C}) - \{0\},\ D + \mathrm{div}(f) \geq 0 \right\} \cup \{0\}.$$

La dimension de $\mathcal{L}(D)$ est notée $\ell(D)$. L'espace $\mathcal{L}(0)$ est constitué par les fonctions constantes, donc $\ell(0) = 1$. Le théorème de Riemann-Roch donne l'existence d'un entier g unique reliant, pour tout $\mathrm{Div}(\mathcal{C})$ les dimensions de $\mathcal{L}(D)$ et $\mathcal{L}(K_{\mathcal{C}} - D)$, à savoir :

$$\ell(D) - \ell(K_{\mathcal{C}} - D) = \deg D - g + 1.$$

L'entier g est appelé le *genre* de $\mathcal{C}$. De cette identité découlent en particulier $\ell(K_{\mathcal{C}}) = g$ et $\deg K_{\mathcal{C}} = 2g-2$. Il est possible de présenter le genre de $\mathcal{C}$ d'autres façons. Le genre de $\mathcal{C}$ est aussi la dimension de l'espace des formes différentielles holomorphes sur $\mathcal{C}$. Une autre façon informelle (mais liée) consiste à voir g comme le « nombre de trous » de la courbe, lorsqu'elle est définie sur $\mathbb{C}$, vue comme surface de Riemann.

Un point de vue beaucoup plus arithmétique est donné par l'impact du genre sur le choix d'un représentant d'une classe donnée de $\mathrm{Jac}_{\mathcal{C}}(k)$, sous forme d'un diviseur. Supposons donné un point k-rationnel particulier, qu'on note ∞. Le théorème de Riemann-Roch permet d'écrire un représentant d'une classe quelconque de $\mathrm{Jac}_{\mathcal{C}}(k)$ comme un diviseur $D = E - (\deg E)(\infty)$, où E est un diviseur effectif de degré au plus égal à g. Une représentation $D = E - (\deg E)(\infty)$ avec $\deg E$ minimal est appelée représentation réduite de D.

Ainsi, une fois fixé le point ∞, la jacobienne d'une courbe de genre 1 est en bijection avec les points de la courbe. En genre 2, un diviseur est donné par le choix d'un ou deux points de $\mathcal{C}(k)$ (éventuellement identiques), ou bien d'une paire de points conjugués dans une extension de degré 2 de k.

7.1.4 Exemples

On illustre les définitions qui viennent d'être données par quelques exemples de courbes importantes par leur intérêt cryptographique et/ou vis-à-vis de ce mémoire. Tout d'abord le premier cas non trivial de courbe algébrique est constitué par les courbes elliptiques, qui sont des courbes de genre 1. Une courbe elliptique, sur un corps de caractéristique $p > 3$, peut être donnée sous forme dite de Weierstraß courte :

$$E : \quad y^2 = x^3 + ax + b.$$

Les courbes elliptiques sont des objets mathématiques fréquents, et aujourd'hui incontournables en cryptographie. Toutefois les travaux liés aux courbes qui sont présentés dans ce mémoire sont principalement tournés vers les courbes de genre supérieur, dont une famille particulière est donnée

par les courbes *hyperelliptiques*. Sur un corps de caractéristique p, une courbe hyperelliptique de genre g est donnée par une équation de la forme

$$y^2 + h(x)y = f(x),$$

où $f(x)$ est un polynôme de degré $2g+1$ ou $2g+2$, et $h(x)$ est un polynôme de degré au plus g. Génériquement, une telle courbe est de genre g. Une courbe hyperelliptique est caractérisée par l'existence d'une involution dite *involution hyperelliptique* $(x, y) \mapsto (x, -y - h(x))$. Toute courbe de genre 2 est hyperelliptique, mais ce n'est pas le cas en genre supérieur. Le cas $\deg f = 2g+1$ est appelé couramment *modèle imaginaire*, et le cas $\deg f = 2g+2$ est appelé couramment *modèle réel*. Ceci est une analogie avec le cas des corps quadratiques réels et imaginaires, le lien se trouvant dans le nombre de places à l'infini.

7.1.5 Représentation des diviseurs

Il est souvent commode de considérer la jacobienne de $\mathcal{C}$ comme un groupe de classes d'idéaux. Cela est particulièrement vrai dans le cas où $\mathcal{C}$ est une courbe hyperelliptique, et possède une unique place à l'infini notée ∞ ($\mathcal{C}$ est donc donnée sous la forme d'un modèle imaginaire). On considère un représentant d'une classe de $\mathrm{Jac}_{\mathcal{C}}(k)$ sous forme d'un diviseur réduit $D = E - (\deg E)\infty$. On choisit de manipuler ce diviseur réduit comme idéal de $k[x, y]/(F)$. Les points du support de E sont notés $P_i = (x_i, y_i)$. On peut représenter D comme $\langle u(x), y - v(x)\rangle$. Les racines dans $\bar{k}$ de $u(x)$ sont les x_i. En outre pour chaque x_i, on a $v(x_i) = y_i$, ou encore, u divise $F(x, v(x))$. Il apparaît que $u(x)$ est de degré au plus égal à g, et $v(x)$ de degré au plus égal à $g-1$. Comme D est stable sous l'action de Galois, les polynômes u et v sont à coefficients dans k. On note que la condition $u \mid F(x, v(x))$ correspond précisément au fait que $\langle u(x), y - v(x)\rangle$ désigne bien un idéal non trivial dans $k[x, y]/(F)$. Cette représentation est couramment appelée représentation de Mumford. Sa généralisation au cas non hyperelliptique partage plusieurs aspects avec la problématique de la représentation d'un idéal dans un corps de nombres : le cas hyperelliptique ici, comme le cas quadratique dans les corps de nombres, est un cas simple.

7.2 Cryptologie des courbes et logarithme discret

On rappelle que les courbes manipulées sont des courbes projectives lisses, pourvues d'un point rationnel.

L'utilisation des courbes algébriques en cryptologie commence par porter l'attention sur les courbes algébriques définies sur les corps finis. À l'évidence, lorsqu'on s'intéresse à une courbe $\mathcal{C}$ définie sur un corps fini $\mathbb{F}_q$, le nombre de points de $\mathcal{C}(\mathbb{F}_q)$ est fini. En outre, la construction de diviseurs réduits à l'aide du théorème de Riemann-Roch permet de déduire que $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ est aussi un groupe fini. Le cardinal de ce groupe évolue dans un intervalle intimement lié au genre de la courbe $\mathcal{C}$.

Théorème 7.1 (Hasse-Weil). *Soit $\mathcal{C}$ une courbe définie sur $\mathbb{F}_q$, de genre g. Le cardinal de $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ vérifie :*

$$(\sqrt{q} - 1)^{2g} \leq \#\,\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q) \leq (\sqrt{q} + 1)^{2g}.$$

L'objectif du cryptographe est de déterminer dans quelle mesure il est possible d'utiliser le groupe $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ pour mettre en place, par exemple, un échange de clés de Diffie-Hellman. Pour cela, il faut obtenir la réponse à plusieurs questions :

- Comment représente-t-on les éléments de $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$. Comment calcule-t-on efficacement avec ?
- Comment peut-on calculer $\#\,\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$, afin de s'assurer de l'existence d'un sous-groupe cyclique de cardinal premier de taille suffisante (aussi proche que possible de $\#\,\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$) ?
- Quels sont les algorithmes dont dispose un attaquant potentiel pour résoudre le problème du logarithme discret dans $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$?

7.2.1 Représentation des éléments et arithmétique

La représentation de Mumford, évoquée plus haut, est une manière satisfaisante de représenter les éléments de $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ (dans le cas du genre 1, parler de la représentation de Mumford est simplement une façon lourde de dire qu'on représente les points par une paire de coordonnées). Un élément de la jacobienne est ainsi représenté à l'aide d'au plus $2g$ coefficients dans $\mathbb{F}_q$. Travailler avec cette représentation, et notamment calculer la loi de groupe dans $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$, est accessible. Bien souvent toutefois, l'arithmétique la plus efficace nécessite de représenter les éléments sous une forme alternative. Il arrive aussi que cette quête d'efficacité s'accompagne d'un sacrifice de généralité, l'emploi des formules les plus efficaces nécessitant la restriction à une certaine classe de courbes. Dans le cas du genre 1 (courbes elliptiques), les différentes formes de formules pour la loi de groupe, en fonction des classes de courbes considérées, sont mentionnées dans [25]. Pour le genre 2, un travail de recensement semblable n'a pas été fait. Des formules explicites existent de longue date, et à ce jour les formules les plus efficaces sont celles obtenues en considérant la surface de Kummer associée à la courbe $\mathcal{C}$ [87] (voir aussi 15.4). Des travaux ont aussi été menés pour fournir une arithmétique assez efficace sur les courbes de genre 3, qu'on évoque en 8.7.1.

L'étude des formules explicites en genre 1 et 2 permet d'arriver à la conclusion suivante : calculer avec des courbes elliptiques, ou avec des jacobiennes de courbes hyperelliptiques de genre 2, peut se faire efficacement. Une illustration en est d'ailleurs donnée en 15.4 ou encore plus récemment dans [192, 143]. La complexité apparente des courbes de genre 2 par rapport aux courbes elliptiques est une fausse idée. L'écart d'efficacité entre genre 1 et genre 2 est en fait assez ténu.

7.2.2 Cardinalité

On a fait état plus haut de l'importance du calcul du cardinal d'un groupe dans lequel on souhaite mettre en œuvre un protocole cryptographique. Ici, on souhaite donc être en mesure de calculer le cardinal de la jacobienne d'une courbe définie sur un corps fini, dans le but de s'assurer de l'existence d'un grand facteur premier, de sorte à disposer dans $\mathrm{Jac}_{\mathcal{C}}(k)$ d'un sous-groupe cyclique aussi grand que possible. Cette assurance permet de se prémunir contre une éventuelle attaque via la réduction de Pohlig-Hellman.

Plusieurs réponses peuvent être données, en fonction du contexte. Pour les courbes de genre 1, les courbes elliptiques, le problème est essentiellement résolu. Sur un corps fini $\mathbb{F}_q$ où $q = p^n$ avec p premier, le calcul du cardinal (du groupe des points) d'une courbe elliptique est réalisable par un algorithme polynomial. Pour p petit, les algorithmes p-adiques [180, 76, 151] sont les plus compétitifs, tandis que plus généralement l'algorithme de Schoof-Elkies-Atkin (algorithme ℓ-adique) peut être utilisé. Dans les deux cas, l'état de l'art montre qu'il est possible (et même instantané pour p petit) de calculer le cardinal d'une courbe elliptique pour des tailles bien au-delà des courbes rencontrées en pratique dans le but d'assurer une sécurité donnée.

Dans le cas des courbes de genre 2, la situation est différente. En petite caractéristique, les algorithmes p-adiques conservent toute leur efficacité, et permettent de considérer le problème comme

résolu [140, 214]. Dans un cadre plus général, parvenir à rendre efficace un algorithme ℓ-adique, soit une extension de l'algorithme de Schoof, n'est pas aisé. Le record actuel est le calcul, au prix d'efforts important, du cardinal d'une jacobienne de courbe de 256 bits, dont le cardinal possède un grand facteur premier [91]. Il est important de remarquer que bien que ce calcul nécessite des moyens importants, il est suffisant de ne l'effectuer qu'une seule fois. Une courbe satisfaisant à la fois aux exigences de sécurité souhaitées, et possédant éventuellement des avantages d'implantation (équation de définition avec de petits coefficients, par exemple) peut être standardisée et utilisée ensuite universellement. Le protocole de Diffie-Hellman, ou les divers protocoles s'appuyant sur la difficulté du logarithme discret, ne l'interdisent pas. Il est donc acceptable d'avoir à effectuer un calcul significatif pour connaître la cardinalité, gage de sûreté.

Notons que le cas des jacobiennes de courbes de genre 3 et plus reçoit une attention décroissante depuis nos travaux présentés au chapitres 8 et 9.

7.2.3 Logarithme discret

Une autre vérification nécessaire en préalable à l'emploi d'un groupe pour la cryptographie est la difficulté du problème du logarithme discret dans ce groupe. Hélas, on ne connaît pas de preuve d'une borne *inférieure* (non triviale) de complexité pour un algorithme de calcul de logarithme discret dans $\mathrm{Jac}_{\mathcal{C}}(k)$, assurant la difficulté de ce problème et donc la solidité de l'hypothèse cryptographique donnant lieu à l'utilisation de $\mathrm{Jac}_{\mathcal{C}}(k)$ dans un objectif cryptographique. Tout au plus peut-on dire que la connaissance d'un tel algorithme donne une borne supérieure sur la complexité de l'algorithme optimal.

Les algorithmes permettant de calculer un logarithme discret dans un groupe quelconque sont de complexité $O(\sqrt{\#G})$. En genre 1 et 2, l'état de l'art ne fait pas mieux que ces algorithmes exponentiels (en $\log \#G$).

L'essentiel des travaux présentés dans cette partie sont directement motivés par le problème de la résolution du logarithme discret dans les jacobiennes de courbes. Nous allons d'abord fixer une distinction dans la problématique entre « genre grand » et « genre petit », qui sépare l'approche en deux branches distinctes.

7.3 Genre grand ou petit

Lorsqu'on s'intéresse à la complexité du calcul du cardinal de la jacobienne d'une courbe de genre g sur un corps fini $\mathbb{F}_q$, il est implicite qu'on s'intéresse à la complexité de cet algorithme en fonction des paramètres q et g. Ainsi, plus rigoureusement, considère-t-on une famille de courbes $\mathcal{C}_i$, de genre g_i, définies sur des corps finis $\mathbb{F}_{q_i}$. On souhaite déterminer une fonction de coût $C(q_i, g_i)$ telle que l'algorithme (probabiliste) considéré pour calculer un logarithme discret dans $\mathrm{Jac}_{\mathcal{C}_i}(\mathbb{F}_{q_i})$ est de complexité bornée par $C(q_i, g_i)$.

Un algorithme est typiquement valide sous certaines hypothèses quant à l'évolution des paramètres $\mathcal{C}_i$, g_i, q_i. On distingue deux problématiques distinctes :

– *Petit genre.* Un algorithme pour le petit genre résout le problème ci-dessus pour une famille de courbes dont le genre g_i est constant, fixé à une valeur g. Un algorithme pour le petit genre peut aussi être valide uniquement pour une certaine famille de courbes, par exemple uniquement pour les courbes qui sont (ou ne sont pas) hyperelliptiques.

  La complexité d'un algorithme en petit genre, c'est-à-dire à g constant, peut dépendre de g. Toutefois, puisque les notations dont notées sous la forme $O()$, il est usuel d'omettre les

constantes absorbées par l'expression $O()$. Ainsi, si on suppose qu'on dispose d'un algorithme de complexité exacte $C(q,g) = g!q^{2-2/g}$, il est licite, dans le cadre du petit genre, d'écrire cette complexité comme $O(q^{2-2/g})$.

– *Grand genre.* Un algorithme pour le grand genre est l'opposé du cas précédent. Ici, on s'intéresse à une famille de courbes dont le genre g_i croît vers $+\infty$. Dans les travaux que nous décrivons au chapitre 10, nous nous intéressons au cas où q_i est autorisé à croître simultanément, dans certaines limites. Un algorithme pour le grand genre peut n'être valide que pour une sous-famille de courbes.

7.4 État de l'art précédent

Un exemple d'algorithme se plaçant dans le contexte du genre grand est l'algorithme d'Adleman-DeMarrais-Huang [5], qui calcule des logarithmes discrets sur les jacobiennes de courbes hyperelliptiques[2] en complexité $L_{q^g}[1/2, c+o(1)]$, où L désigne la fonction sous-exponentielle usuelle, déjà rencontrée lors de l'étude de la grande famille du crible algébrique dans la partie I :

$$L_x[\alpha, c] = \exp\left(c(\log x)^\alpha (\log\log x)^{1-\alpha}\right).$$

À l'instar d'autres algorithmes de complexité sous-exponentielle, l'algorithme de Adleman-DeMarrais-Huang exploite une notion de taille, et de friabilité. L'objectif, pour D_1 et D_2 deux éléments d'un même sous-groupe cyclique d'une jacobienne, est de calculer le logarithme de D_1 en base D_2. Cet objectif est atteint au moyen de l'écriture d'une identité du type $\alpha D_1 + \beta D_2 = 0$ dans la jacobienne. Pour y parvenir, les points de la jacobienne représentés sous forme réduite par des diviseurs de poids $\sqrt{g}$ constituent la base de facteurs. De nombreuses combinaisons $D_{\alpha,\beta} = \alpha D_1 + \beta D_2$ sont tentées, et une *relation* est obtenue lorsque $D_{\alpha,\beta}$ peut s'écrire sous forme d'une somme de diviseurs de la base de facteurs. À l'instar d'une procédure déjà rencontrée, les nombreuses relations ainsi collectées sont combinées grâce à la résolution d'un système linéaire, amenant la relation $\alpha D_1 + \beta D_2 = 0$ voulue.

La notion de friabilité utilisée dans l'algorithme d'Adleman-DeMarrais-Huang fournit une complexité sous-exponentielle grâce à la croissance du genre, puisque la probabilité de friabilité de $D_{\alpha,\beta}$ est essentiellement la probabilité de friabilité d'un polynôme de degré g sur une base de facteurs de polynômes de degré au plus $\sqrt{g}$ (on exploite ici la représentation de Mumford).

Une spécialisation de l'algorithme de Adleman-DeMarrais-Huang a été proposée par Gaudry [86], afin d'étudier le problème en *petit* genre. L'objectif est ici de se focaliser sur une famille de jacobiennes de courbes hyperelliptiques de genre fixé, afin d'étudier son intérêt cryptographique. Ici, point de genre croissant pour déterminer une base de facteurs de taille croissante. Seul le cardinal du corps de base croît. Aussi, le poids des diviseurs choisis pour appartenir à la base de facteurs est nécessairement constant. Supposons ainsi qu'on choisit les diviseurs de poids 1 comme éléments de la base de facteurs. On a ainsi $O(q)$ éléments de la base de facteurs $\mathcal{B}$. La probabilité pour qu'une combinaison $D_{\alpha,\beta} = \alpha D_1 + \beta D_2$ puisse se décomposer sur $\mathcal{B}$ (en d'autres termes, être $\mathcal{B}$-friable) est une probabilité constante en g, heuristiquement $\frac{1}{g!}$, c'est-à-dire $O(1)$ puisque g est une constante. Dès lors, $O(q)$ tentatives permettent d'obtenir $O(q)$ relations. Il est ainsi possible, pour un coût $O(q^2)$ lié à la résolution d'un système linéaire, de résoudre le logarithme discret dans la jacobienne $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ de la courbe hyperelliptique $\mathcal{C}$ de genre g en temps $O(q^2)$. Ce résultat donne un algorithme

[2] Cet algorithme a par la suite été généralisé [69, 55].

meilleur que l'algorithme générique des « pas de bébé, pas de géant » dès que le genre g est supérieur ou égal à 4. Il peut, en genre 2, être reformulé pour obtenir une complexité $O(q)$ en exploitant la forme spéciale du système linéaire à résoudre. Une inconnue particulière laissée par ce système est l'éventualité d'améliorations spécifiques au genre 3 : en effet, la complexité obtenue par l'algorithme de [86] est de $O(q^2)$, qui n'est pas meilleure que la complexité $O(q^{3/2})$ de l'algorithme générique.

L'algorithme de [86] a été ensuite amélioré par Harley, partant du constat que le coût de $O(q)$ pour la recherche de relations, contre $O(q^2)$ pour la résolution du système linéaire, est un déséquilibre flagrant encourageant la recherche d'améliorations. Ceci peut être atteint de la façon suivante. Le choix pour la base de facteurs $\mathcal{B}$ d'utiliser les diviseurs de poids 1 est légèrement modifié. On considère toujours des diviseurs de poids 1, mais pas la *totalité* d'entre eux. Seule une partie de cardinal q^r, pour $r < 1$, est retenue. Ainsi, la probabilité de $\mathcal{B}$-friabilité du diviseur $D_{\alpha,\beta}$ devient $O(q^{(r-1)g})$: en effet, si $D_{\alpha,\beta}$ se décompose en une somme de g diviseurs de poids 1, chacun de ces g diviseurs doit être dans $\mathcal{B}$. Par voie de conséquence, le coût pour obtenir les $O(q^r)$ relations requises devient $O(q^{r+(1-r)g})$, et le coût de l'étape d'algèbre linéaire devient $O(q^{2r})$. Pour $r = g/(1+g)$, ces coûts s'équilibrent, ce qui permet d'atteindre une complexité, en général, de $O(q^{2-2/(g+1)})$. Pour le cas précis de $g = 3$, l'amélioration apportée est substantielle, puisqu'on passe ainsi d'une complexité $O(q^2)$ à une complexité $O(q^{3/2})$. Toutefois cette dernière complexité permet seulement de rattraper la complexité de l'algorithme générique.

L'évolution suivante de cet algorithme en petit genre est l'introduction de *large primes*, par Thériault [213]. Cette idée, que nous décrivons simultanément avec son extension des *double large primes* dans le chapitre 8, permet d'obtenir une complexité pour le calcul de logarithme discret en $O(q^{2-2/(g+1/2)})$, soit $O(q^{10/7}) = O(q^{1.43})$ en genre 3. Nos travaux, décrits dans les chapitres qui suivent, ont permis d'améliorer cette complexité.

Chapitre 8

Double large primes pour le logarithme discret

Nous résumons dans ce chapitre les travaux présentés dans l'article [T7], en collaboration avec P. Gaudry, N. Thériault, et C. Diem. Nous prenons le parti de ne pas reprendre *in extenso* les détails de [T7], mais uniquement les lignes directrices, ainsi que certains points auxiliaires n'apparaissant pas dans [T7].

8.1 Présentation

Nous nous intéressons au calcul de logarithme discret en genre petit, c'est-à-dire sur la jacobienne d'une courbe $\mathcal{C}$ de genre g fixé. Les travaux de [T7] peuvent être vus sous deux angles. Dans le cas où la courbe $\mathcal{C}$ est hyperelliptique et le groupe $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ cyclique, alors le résultat suivant peut être prouvé rigoureusement.

Théorème 8.1 ([T7]). *Soit $g \geq 3$ fixé, et $\mathcal{C}$ une courbe hyperelliptique de genre g sur $\mathbb{F}_q$, telle que le groupe $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ est un groupe cyclique. Alors le problème du logarithme discret dans $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ peut être résolu en temps*

$$\widetilde{O}(q^{2-2/g}).$$

Dans un cadre plus général (courbe $\mathcal{C}$ non nécessairement hyperelliptique, ou jacobienne non cyclique), le résultat reste valide heuristiquement. La présentation synthétique que nous faisons ici des travaux de [T7] omet sciemment certains points techniques, tout en tentant d'illustrer le contenu de l'article.

8.2 Choix d'une base de facteurs

La stratégie est bâtie à partir de l'état de l'art précédent, présenté en 7.4. La variante à genre fixé de l'algorithme de Adleman-DeMarrais-Huang, exposée dans [86], commence par fixer une *base de facteurs*. Le choix pertinent en genre fixé est une base de facteurs constituée de diviseurs de poids 1, c'est-à-dire, à une bijection près, de points rationnels sur $\mathcal{C}$. La borne de Serre indique que le cardinal de $\mathcal{C}(\mathbb{F}_q)$ est au plus $q + 2g\sqrt{q}$, soit $q + O(\sqrt{q})$. À l'instar de la variante due à Harley, on suppose ici qu'on s'intéresse seulement à une petite portion de ces $q + O(\sqrt{q})$ diviseurs. On note un réel $r < 1$, optimisé en fonction de l'analyse de l'algorithme, et on choisit une base de facteurs constituée d'une partie arbitraire[1] de q^r points de $\mathcal{C}(\mathbb{F}_q)$. Étant donné que $r < 1$, et $\#\mathcal{C}(\mathbb{F}_q) = q + O(\sqrt{q})$, construire

[1] Il est bien sûr possible d'optimiser modérément ce choix en ne conservant qu'un unique représentant de chaque classe modulo l'involution hyperelliptique. À utilité égale, la base de facteurs est réduite de moitié. Outre l'avantage pratique ainsi obtenu, cette optimisation est aussi un ingrédient de la présentation rigoureuse du théorème 8.1. Par souci de concision, nous choisissons de conserver des notations plus légères.

une telle base de facteurs est toujours possible pour q assez grand. Notons $\mathcal{B}$ la base de facteurs ainsi construite.

L'ensemble des diviseurs de poids 1 n'appartenant *pas* à $\mathcal{B}$ est noté $\mathcal{L}$. Son cardinal est $q - q^r + O(\sqrt{q})$, soit $\#\mathcal{L} \sim q$. Dans la variante de Harley, on détermine la probabilité qu'un diviseur soit $\mathcal{B}$-friable, c'est-à-dire se décompose comme une somme de diviseurs appartenant à $\mathcal{B}$. La proposition suivante permet de généraliser cette considération.

Proposition 8.2. *Soit $\mathcal{C}$ une courbe hyperelliptique de genre g sur $\mathbb{F}_q$, et $\mathcal{B} \cup \mathcal{L}$ une partition de l'image de $\mathcal{C}(\mathbb{F}_q)$ dans $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$, avec $\#\mathcal{B} = q^r$ ($0 < r < 1$). Soit $0 \leq k \leq g$ un entier. La probabilité π qu'un diviseur aléatoire uniforme de $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ se décompose en une somme de k diviseurs de $\mathcal{L}$ et au plus $g - k$ diviseurs de $\mathcal{B}$ est :*

$$\pi \sim \frac{1}{k!(g-k)!} q^{(r-1)(g-k)}.$$

La proposition 8.2 peut être prouvée rigoureusement sous les hypothèses du théorème 8.1. Dans un contexte plus général, les conclusions restent valides heuristiquement.

La procédure que nous envisageons pour calculer le logarithme discret est donné par l'algorithme 8.5. Toutefois, avant d'être en mesure d'atteindre sa formulation complète, un passage en revue des outils et concepts est nécessaire. Le problème que nous résolvons est le calcul du logarithme discret de D_1 en base D_2, où D_1 et D_2 sont deux diviseurs non nuls du même sous-groupe cyclique d'ordre premier de $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$. Notre algorithme est un descendant de l'adaptation par Gaudry [86] de l'algorithme de Adleman-DeMarrais-Huang [5]. Nous calculons un grand nombre de combinaisons $D_{\alpha,\beta} = \alpha D_1 + \beta D_2$, et nous examinons comment le diviseur $D_{\alpha,\beta}$ peut être décomposé afin d'écrire une *relation*. Ces relations sont *in fine* combinées par une étape d'algèbre linéaire, dans le but de produire une relation de dépendance linéaire $\alpha D_1 + \beta D_2 = 0$, de laquelle la solution recherchée peut se déduire.

Notre choix d'une base de facteurs ne contenant pas la totalité des points de $\mathcal{C}(\mathbb{F}_q)$ donne lieu à l'existence de diviseurs se décomposant complètement comme somme d'au plus g points de la courbe, mais hélas pas g points de la base de facteurs choisie. Dans la lignée de la terminologie consacrée par les travaux sur la factorisation d'entiers [161, 138], une relation[2] $D_{\alpha,\beta} = (P_1) + (P_2) + \cdots + (P_k)$ est appelée complète (ou encore *full*, ou FF), lorsque tous les sommants (P_i) sont des diviseurs de poids 1 appartenant à la base de facteurs $\mathcal{B}$. Une relation est appelée *partielle*, ou FP, lorsqu'exactement un des sommants appartient à $\mathcal{L}$. Une relation est appelée PP lorsqu'exactement deux sommants appartiennent à $\mathcal{L}$. Les sommants appartenant à $\mathcal{L}$ dans la décomposition de $D_{\alpha,\beta}$ sont appelés *large primes*.

8.3 Un seul *large prime*

Le travail de Thériault [213] consiste à examiner le cas d'au plus *un* large prime. Dès lors que deux relations ou plus sont rencontrées impliquant le même *large prime*, il est possible de les combiner pour construire de nouvelles relations sans *large primes*. Ce contexte a l'avantage certain de se prêter assez bien à l'analyse, soit via une variante du paradoxe des anniversaires, soit à l'aide de séries génératrices [160, T4]. Pour obtenir $\#\mathcal{B} \sim q^r$ relations ou davantage impliquant les éléments de la base de facteurs, on compose T combinaisons de la forme $D_{\alpha,\beta}$. Lorsque les large primes ne

[2] Là encore, une implémentation pratique exploite l'involution hyperelliptique et choisit un représentant unique par classe.

sont pas utilisés, le nombre de relations voulu est obtenu pour $T = \Theta(q^{r+(1-r)g})$. Lorsqu'on utilise des relations impliquant un *large prime*, on obtient à partir de T combinaisons, en vertu de la proposition 8.2 :

$$\begin{aligned} \text{nombre de relations FF :} \quad & n_{\text{FF}} = \Theta(Tq^{(r-1)g}), \\ \text{nombre de relations FP :} \quad & n_{\text{FP}} = \Theta(Tq^{(r-1)(g-1)}). \end{aligned}$$

Le nombre de combinaisons de relations FP permettant de déduire des relations sans large primes est alors $n_{\text{FP}}^2/(2\#\mathcal{L})$, d'où un nombre total de relations FF et de relations FP combinées qui est

$$\Theta(Tq^{(r-1)g}) + \Theta(T^2 q^{2(r-1)(g-1)-1}).$$

On néglige le nombre de relations FF produites. En exploitant le fait que l'algèbre linéaire qui suit a un coût $O(q^{2r})$, on aboutit à une valeur optimale $r = 1 - \frac{1}{g+1/2}$, donnant un temps de calcul total :

$$O\left(q^{2-\frac{2}{g+1/2}}\right)$$

opérations dans $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$. Ces dernières ayant un coût polynomial en $\log q$, on obtient une complexité totale pour le calcul de logarithmes discrets dans $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ selon cette méthode qui est en $\widetilde{O}\left(q^{2-\frac{2}{g+1/2}}\right)$.

8.4 Deux *large primes*

Le prolongement naturel de l'approche précédente est l'utilisation de deux *large primes*, dans la droite ligne de travaux classiques dans le contexte de la factorisation d'entiers.

Le principe, rapidement énoncé, est le suivant. Au fur et à mesure de la constitution de combinaisons $D_{\alpha,\beta}$, les relations FF, FP, et PP sont toutes conservées. La recombinaison de relations PP entre elles est possible pour obtenir une relations sans *large primes* lorsqu'on considère un ensemble de relations PP dont les sommets sont arrangés en *cycle*, dans un graphe qu'il convient de définir, appelé LP-graphe.

Définition 8.3. *Soit $\mathcal{S}$ un ensemble de relations FP ou PP, faisant intervenir des* large primes *qui appartiennent à un ensemble $\mathcal{L}$. Le* **LP-graphe** *associé à $\mathcal{S}$ est un graphe non orienté dont l'ensemble des sommets est l'union disjointe $\mathcal{L} \cup \{*\}$, et dont les arêtes sont donnée par la correspondance suivante :*

$$\begin{aligned} \textit{relation FP faisant intervenir un} \text{ large prime } P_1 : \quad & \textit{arête } P_1 \text{ — } *, \\ \textit{relation PP faisant intervenir deux} \text{ large primes } P_1, P_2 : \quad & \textit{arête } P_1 \text{ — } P_2. \end{aligned}$$

Le LP-graphe évolue au fur et à mesure de l'ajout d'arêtes, provenant de relations. Les *cycles* intervenant dans ce graphe permettent des recombinaisons de relations :

Proposition 8.4. *Soit P_1 — $\cdots$ — P_k — P_1 un cycle de longueur k dans le LP-graphe. Il est possible de recombiner les relations correspondant aux arêtes de ce cycle :*

- *en toute généralité, en $k-1$ relations faisant intervenir au plus un* large prime *;*
- *dans le cas où le sommet $*$ est l'un des P_i, on obtient $k-1$ relations combinées qui ne contiennent pas de* large prime.

Dans la proposition précédente, la discrimination entre deux cas possibles pour le nombre de *large primes* intervenant dans la recombinaison des relations tient à la nécessité de considérer les relations comme faisant intervenir les *large primes* avec des coefficients. Ces coefficients peuvent apparaître pour plusieurs raisons. Tout d'abord, pour un souci d'optimisation il est naturel d'utiliser un représentant unique de chaque classe d'équivalence sous l'involution hyperelliptique, ce qui peut induire des coefficients ± 1 dans les relations. D'autre part, recombiner des relations PP pour obtenir une relation FP produit une relation où le *large prime* est pondéré par un coefficient. Enfin, il est possible qu'un *large prime* apparaisse avec multiplicité dans une relation (exceptionnellement rarement toutefois). Si les k relations considérées font chacune intervenir les *large primes* P_i et P_{i+1} (où $k+1$ correspond à 1) affectés de coefficients m_i et m'_i, alors la nature de la relation recombinée qui peut être obtenue est directement caractérisé par le déterminant sur $\mathbb{Z}$ de la matrice :

$$\begin{pmatrix} m_1 & m'_1 & & \\ & \ddots & \ddots & \\ & & \ddots & m'_{k-1} \\ m'_k & & & m_k \end{pmatrix}$$

L'énumération d'une base des cycles apparaissant au cours de l'évolution d'un graphe est aisée, et de coût quasi-linéaire par exemple via l'algorithme classique *union-find* [7]. Nous décrivons l'évolution du LP-graphe au fur et à mesure de l'obtention de relations FF, FP ou PP. Initialement, un compteur indiquant le nombre de relations connues qui impliquent exclusivement des éléments de $\mathcal{B}$ débute avec la valeur 0, et le LP-graphe ne contient aucune arête. L'obtention d'une relation $\mathcal{R}$ donne lieu aux actions suivantes :

- Si $\mathcal{R}$ est une relation FF, le compteur C est incrémenté et le LP-graphe est inchangé.
- Si $\mathcal{R}$ est une relation FP ou PP, et que l'introduction de l'arête correspondante dans le LP-graphe (voir définition 8.3) n'introduit pas de cycle, alors cette arête est ajoutée au LP-graphe, et le compteur C est inchangé.
- Enfin, si $\mathcal{R}$ est une relation FP ou PP, et que l'introduction de l'arête correspondante dans le LP-graphe introduit un cycle Γ, alors l'arête n'est *pas* ajoutée au LP-graphe. On considère la relation $\mathcal{R}'$ obtenue par la réduction du cycle Γ. Comme évoqué plus haut, $\mathcal{R}'$ est une relation impliquant zéro ou un *large prime*.
  - Si $\mathcal{R}'$ n'implique aucun *large prime*, alors C est incrémenté, et le LP-graphe est inchangé.
  - Si $\mathcal{R}'$ implique un *large prime*, alors C est inchangé, le LP-graphe est inchangé, et $\mathcal{R}'$ est considéré à nouveau en tant que relation FP pour inclusion possible dans le LP-graphe par la même procédure.

Ce processus d'évolution étant défini, nous sommes en mesure de donner la formulation complète de notre algorithme de calcul de logarithmes discrets, présenté sous la forme de l'algorithme 8.5. L'analyse de l'algorithme 8.5 révèle deux difficultés.

- Il convient de déterminer le nombre de diviseurs $D_{\alpha,\beta}$ qui doivent être calculés afin que le compteur C atteigne $\#\mathcal{B}+1$.
- Il est important de déterminer le nombre moyen de diviseurs de $\mathcal{B}$ qui composent les relations recombinées à partir du LP-graphe.

ENTRÉE : $\mathcal{C}$ courbe hyperelliptique de genre g sur $\mathbb{F}_q$.
D_1 diviseur de $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$; D_2 diviseur de $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$, tel que $D_2 \in \langle D_1 \rangle$.
SORTIE : m tel que $D_2 = mD_1$.
NOTATION : Soit $r = 1 - \frac{1}{g}$.

1. Soit $\mathcal{B} = \{P_1, \ldots, P_N\}$ une partie de $\mathcal{C}(\mathbb{F}_q)$ de cardinal $N = q^r$. Soit $C = 0$, et G un graphe sur l'ensemble de sommets $\mathcal{L} \cup \{*\}$, où $\mathcal{L} = \mathcal{C}(\mathbb{F}_q) - \mathcal{B}$. Initialement, G ne contient pas d'arête.
2. Tant que $C \leq \#\mathcal{B}$:
   (a) Tirer aléatoirement deux entiers α_1 et α_2 dans $[0, \#\,\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)[$.
   (b) Soient $X \subset \mathcal{B}$ et $Y \subset \mathcal{L}$ tels que $\#Y \leq 2$ et :
   $$\#(X \cup Y) = k, \quad \alpha_1 D_1 + \alpha_2 D_2 = \sum_{P \in X \cup Y} (P) - k(\infty).$$
   S'il n'existe pas de telle décomposition, revenir à l'étape 2.
   (c) Si $Y = \{\}$, incrémenter C, conserver $\mathcal{R}$, et revenir en 2.
   (d) Si $Y = \{P\}$ ou $Y = \{P, Q\}$, soit e l'arête correspondante (voir définition 8.3). Si l'ajout de e à G ne crée pas de cycle, effectuer cet ajout, et revenir en 2.
   (e) Soit $\mathcal{R}'$ la relation obtenue en réduisant le cycle que créerait e dans G (voir proposition 8.4). Considérer cette nouvelle relation $\mathcal{R}'$ pour inclusion dans le graphe (i.e. revenir à l'étape 2(c)).
3. Noter chacune des relations collectées sous la forme :
   $$\alpha_1^{(i)} D_1 + \alpha_2^{(i)} D_2 = \sum_j n_j^{(i)} P_j - (\sum_j n_j^{(i)})(\infty).$$
4. Construire la matrice M de coefficients $(n_j^{(i)})$.
5. Déterminer v tel que $vM = 0$. Déduire $(\sum_i v_i \alpha_1^{(i)}) D_1 + (\sum_i v_i \alpha_2^{(i)}) D_2 = 0$, puis m.

ALGORITHME 8.5 : Logarithme discret sur $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ avec deux *large primes*.

Ces difficultés ont, dans les travaux précédents du contexte de la factorisation d'entiers, été évacuées de manière empirique. La validation de la méthode des *double large primes* passait alors uniquement par l'expérience. Il était constaté que cette méthode fonctionnait mieux que lorsqu'un unique *large prime* était utilisé, l'emploi de deux *large primes* (ou davantage) provoquant une « explosion combinatoire » ou une « transition de phase » qui hélas échappait à l'analyse rigoureuse [138, 61, 142].

L'apport essentiel de [T7] est de proposer un cadre, d'abord heuristique, puis rigoureux sous les hypothèses du théorème 8.1, pour analyser ce phénomène et le quantifier précisément[3].

8.5 Analyse heuristique

Une façon de présenter l'analyse de l'algorithme est la suivante. On considère une unité de temps comme la génération d'un nouveau diviseur $D_{\alpha,\beta}$. On considère, au temps t, la valeur $u(t) \in [0, 1]$

[3]Le fait que le théorème 8.1 fasse apparaître une amélioration significative de la complexité est lié à la nature exponentielle de l'algorithme employé. Ne rêvons pas, il n'y a pas de gain similaire à obtenir pour les algorithmes sous-exponentiels.

indiquant le nombre de sommets du LP-graphe appartenant à la composante connexe du sommet $*$, normalisé par rapport à sa valeur maximale $\#\mathcal{L}$. À chaque temps t, le diviseur $D_{\alpha,\beta}$ provoque un accroissement de la composante connexe de $*$ avec probabilité $b(1-u)+2cu(1-u)$, où on désigne par les constantes a, b et c les probabilités d'obtenir une relation FF, FP, ou PP. Ces constantes s'obtiennent par la proposition 8.2 en fixant $k=0,1,2$:

$$a=\frac{1}{g!}q^{(r-1)g},\quad b=\frac{1}{(g-1)!}q^{(r-1)(g-1)},\quad c=\frac{1}{2(g-2)!}q^{(r-1)(g-2)}.$$

On peut donc modéliser l'évolution du cardinal de la composante connexe de $*$ par l'équation différentielle :

$$u'(t)=\frac{1}{\#\mathcal{L}}(b(1-u)+2cu(1-u)).$$

Cette équation différentielle logistique s'intègre bien. La solution est $E(t)/(E(t)+1/T)$, avec $E(t)=\exp(t/T)-1$ et $T=\#\mathcal{L}/(2c+b)$. Aboutir à une telle fonction suffit à mettre en évidence le comportement « explosif » remarqué empiriquement par les travaux précédents, comme le montrent les figures 8.6 et 8.7. Le « temps critique » T à partir duquel essentiellement n'importe quelle relation PP permet d'obtenir une relation recombinée impliquant uniquement des diviseurs de $\mathcal{B}$, donne la complexité de la phase de recherche de relations.

Toutefois, légitimer totalement cette façon d'analyser le problème pose quelques difficultés. Le résultat du théorème 8.1 est obtenu avec un algorithme simplifié, d'une complexité *in fine* très comparable.

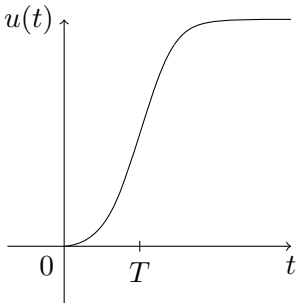


Fig. 8.6 : Forme générale de $u(t)$.

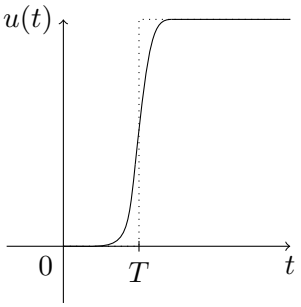


Fig. 8.7 : $u(t)$ lorsque $\#\mathcal{L}\gg 1$.

Nous présentons comment dans [T7], nous développons d'abord un modèle simplifié de l'algorithme qui se prête bien à l'analyse, et nous comparons ensuite les résultats expérimentaux de l'algorithme « complet » ci-dessus à la complexité de l'algorithme simplifié.

8.6 Algorithme simplifié

Le but de la simplification de l'algorithme est d'éliminer un maximum d'aspects non essentiels de l'algorithme, tout en préservant le cœur de son efficacité, à savoir la recombinaison des relations PP. De cette façon, on rend l'analyse plus praticable.

Le cœur de l'efficacité de la méthode des *double large primes* repose dans la croissance d'une composante connexe « géante » dans le LP-graphe. Lorsque cette composante connexe est suffisamment grande, toute relation PP permet d'obtenir avec bonne probabilité une relation recombinée sans *large prime*. Le phénomène de composante connexe géante est classique dans l'étude des graphes

aléatoires [72, 109]. Dans le cas d'un LP-graphe, cette composante connexe géante est la composante connexe du sommet $*$, puisque la présence des relations FP apporte une très forte affinité vers ce sommet spécial.

L'idée de l'algorithme simplifié est de construire *exclusivement* la composante connexe de $*$, ou plus exactement une partie significative de celle-ci, et de le faire de la manière la plus simple possible.

Les étapes de l'algorithme simplifié sont les suivantes. À aucun moment une composante connexe autre que celle de $*$ n'est créée. Par ailleurs, les (rares) relations FF sont toujours ignorées, et les relations FP n'interviennent que dans la première étape (qui n'utilise qu'une seule relation FP).

1. Laisser le graphe intact jusqu'à l'obtention d'*une* relation FP.

2. Faire croître le graphe *uniquement* avec des relations PP, jusqu'à une taille limite $N_{\max}$ décidée par l'analyse.

3. Figer définitivement le graphe. Exploiter *uniquement* les relations PP pour produire des relations recombinées impliquant des diviseurs de $\mathcal{B}$. Les relations PP ne donnant pas de relations recombinées utiles à ce stade sont ignorées.

L'analyse de l'algorithme simplifié réalisée dans [T7] étudie l'espérance du temps de calcul pour la collecte de relations en fonction de la borne $N_{\max}$ choisie, ainsi que la profondeur du LP-graphe sous ce même paramètre. La valeur optimale pour $N_{\max}$ est en $q^{1-\frac{1}{g}+\frac{1}{g^2}}$. L'analyse complète donne que la complexité de l'algorithme simplifié 8.9 page 102 rejoint celle obtenue pour l'algorithme heuristique 8.5, à un facteur multiplicatif $O(\log q)$ près. Ainsi, il reste vrai que le temps de calcul total est en $\widetilde{O}(q^{2-2/g})$. Toutefois, alors que l'analyse heuristique prédit $O(q^{2-2/g})$ opérations nécessaires dans la jacobienne, l'algorithme simplifé en prévoit davantage.

L'expérimentation permet de mesurer comment l'algorithme « complet » (par opposition à l'algorithme simplifié) se comporte en pratique. La table 8.10 donne quelques données expérimentales à cet effet en genre 3. Leur portée est uniquement empirique et ne saurait remplacer l'analyse. Pour différentes tailles du corps de base $\mathbb{F}_q$, et pour des courbes choisies à chaque fois au hasard, on s'intéresse d'une part à l'évolution du nombre de relations requises à la terminaison de l'algorithme. La valeur t indiquée dans la deuxième colonne indique le nombre de diviseurs $D_{\alpha,\beta}$ calculés. Ainsi, le temps de calcul pour la collecte de relations est le produit de t et d'un facteur polynomial en $\log q$ prenant en compte l'arithmétique dans $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ et les tests de friabilité. La comparaison de t avec $q^{4/3}$, donnée dans la troisième colonne, semble valider l'analyse heuristique qui estime à $O(q^{4/3})$ le nombre d'opérations dans la jacobienne nécessaires. D'autre part, et *a contrario*, le calcul de la longueur moyenne des cycles rencontrés, et sa comparaison avec $\log q$, donnée dans la dernière colonne, montre que celle-ci semble croître plus vite que $\log q$, sans qu'il paraisse raisonnable de s'aventurer à une estimation plus fine.

8.7 Mise en pratique

L'algorithme des *double large primes* a été implanté. Nous donnons ici quelques éléments relatifs à cette implantation. On rappelle que l'objectif de cet algorithme, *in fine*, est l'obtenir une relation de dépendance entre deux diviseurs D_1 et D_2 appartenant au même sous-groupe cyclique de $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$, où $\mathcal{C}$ est une courbe hyperelliptique de définie sur $\mathbb{F}_q$. Dans le cadre de l'application, on s'est concentré sur le cas où $\mathbb{F}_q$ est un corps premier, et $\mathcal{C}$ de genre 3.

q	t final	$t/q^{4/3}$	$\#\mathcal{B}$	$\frac{\text{lng. cyc.}}{\log q}$
$\approx 2^{15}$	815473	0.78	512	1.29
$\approx 2^{16}$	1811672	0.69	812	1.19
$\approx 2^{17}$	4705192	0.71	1290	1.41
$\approx 2^{18}$	11253002	0.67	2047	1.42
$\approx 2^{19}$	27776102	0.66	3250	1.44
$\approx 2^{20}$	66834647	0.63	5160	1.47
$\approx 2^{21}$	170327927	0.63	8191	1.59
$\approx 2^{22}$	417044579	0.62	13003	1.70
$\approx 2^{23}$	1036566361	0.61	20642	1.80
$\approx 2^{24}$	2576921045	0.60	32767	1.92
$\approx 2^{25}$	6430349490	0.59	52015	2.02
$\approx 2^{26}$	15899195912	0.58	82570	2.18
$\approx 2^{27}$	39993810485	0.58	131071	2.32

TAB. 8.10 : Valeur finale de t pour l'algorithme complet (données expérimentales).

Le *moyen* utilisé par l'algorithme est le calcul de nombreux diviseurs $D_{\alpha,\beta} = \alpha D_1 + \beta D_2$, et le test de friabilité pour leur forme réduite. L'essentiel du coût est donc le calcul de ces diviseurs $D_{\alpha,\beta}$, de leur forme réduite, et le test de friabilité associé.

Bien qu'une forme telle que $\alpha D_1 + \beta D_2$, où α et β sont choisis de manière aléatoire uniforme, garantisse des propriétés souhaitées pour l'analyse, il est naturel d'en faire sacrifice pour une implantation pratique. Suivant [172, 206], on choisit ainsi un mouvement brownien multi-dimensionnel, où $D_{\alpha_{i+1},\beta_{i+1}} = D_{\alpha_i,\beta_i} + R$, le diviseur R étant un choix aléatoire[4] parmi un certain nombre de diviseurs précalculés, tous de la forme $D_{\alpha,\beta}$. En fixant par exemple un nombre de dimensions égal à 20, on précalcule deux suites d'entiers aléatoires $(a_0, \ldots, a_{19})$ et $(b_0, \ldots, b_{19})$, et l'ensemble des diviseurs associés $\mathcal{R} = \{R_0, \ldots, R_{19}\}$, avec $R_k = a_k D_1 + b_k D_2$. Ensuite, partant d'un diviseur initial $D = 0$, chaque nouveau diviseur pseudo-aléatoire est construit comme le résultat de l'affectation $D \leftarrow D + R$, avec R tiré au hasard dans l'ensemble $\mathcal{R}$. Trivialement, les diviseurs ainsi fabriqués restent connus sous la forme de combinaisons de D_1 et D_2. Chaque nouveau diviseur devant être construit requiert donc *une* addition dans $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$, suivi d'un test de friabilité pour le polynôme $u(x)$ de la représentation de Mumford. Nous donnons quelques brefs éléments relatifs à ces opérations.

8.7.1 Arithmétique en genre 3

Une addition dans la jacobienne d'une courbe hyperelliptique de genre 3 peut être effectuée au moyen de formules explicites. Un travail d'optimisation de telles formules explicites a été réalisé dans [218]. On se place dans le cas d'une courbe $\mathcal{C}$ définie sur un corps fini $\mathbb{F}_q$ de caractéristique

[4] À la différence de la méthode ρ, où le parcours doit correspondre à un graphe fonctionnel, ici on ne requiert *pas* que $D_{\alpha_{i+1},\beta_{i+1}}$ soit déterminé par son prédécesseur D_{α_i,β_i}.

impaire par une équation :

$$\begin{aligned}\mathcal{C}: \quad & y^2 + h(x)y = f(x),\\ & h(x) = h_3x^3 + h_2x^2 + h_1x + h_0,\\ & f(x) = x^7 + f_6x^6 + f_5x + f_4 + f_3x^3 + f_2x^2 + f_1x + f_0.\end{aligned}$$

En outre, on se restreint au cas où les coefficients h_i sont tous égaux à 0 ou 1. L'article [218] mentionne des coûts relatifs à la situation $f_6 = 0$, restriction que nous ne faisons pas ici.

Les formules proposées dans [218, table IX] permettent d'effectuer une addition dans $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ pour un coût noté $I + 70M + 6S$ (une inversion, 70 multiplications dans $\mathbb{F}_q$, 6 carrés dans $\mathbb{F}_q$). On remarque que cet article laisse certains points incomplètement traités. Les étapes et notations donnés ici font référence à la table IX de [218]. Pour mémoire, nous mentionnons ici différentes améliorations possibles, inédites car d'un intérêt relativement maigre.

– Étape 5, calcul de $(t^2 + s_1t + s_0)(t^3 + at^2 + bt + c)$. Seules les multiplications sont prises en compte, donc l'étude équivaut à celle du calcul du produit $(s_1t + s_0)(at^2 + bt + c)$. Le coût proposé dans [218] est de $6M$. Les quatre coefficients du polynôme produit peuvent être obtenus pour un coût moindre par évaluation/interpolation. Notons $r = r_0 + r_1t + r_2t^2 + r_3t^3$ le produit, et les évaluations

$$r(0) = r_0,\ r(1) = (s_0 + s_1)(a + b + c),\ r(-1) = (s_0 - s_1)(a - b + c),\ r(\infty) = s_1c.$$

Les coefficients de r sont liés aux valeurs ci-dessus par la relation :

$$(r_0\ r_1\ r_2\ r_3) \cdot \begin{pmatrix} 1 & 1 & 1 & 0\\ 0 & 1 & -1 & 0\\ 0 & 1 & 1 & 0\\ 0 & 1 & -1 & 1 \end{pmatrix} = (r(0)\ r(1)\ r(-1)\ r(\infty)).$$

La matrice d'évaluation M intervenant ci-dessus est de déterminant 2. En l'inversant, on obtient :

$$r_0 = r(0),\ r_2 = \frac{r(1) + r(-1)}{2} - r(0),\ r_1 = r_2 + r(0) - r(-1) - r(\infty),\ r_3 = r(\infty).$$

Ce qui, au prix de l'opération de division par 2 qui n'est pas onéreuse, permet d'effectuer l'étape 5 pour un coût de $4M$ (ou bien $5M$ sans division par 2).

– Étape 6. Les trois quantités s_1z_4, $s_0z_4 + s_1z_3$, s_0z_3 sont calculées, et de même pour les trois quantités du'_3, $eu'_3 + du'_2$, et eu'_2. Dans les deux cas, [218] décompte un coût de $4M$, ce qui se réduit à $3M$ par Karatsuba. On effectue ainsi l'étape 6 pour un coût de $13M$ contre $15M$.

– Étape 1, calcul du résultant de u_1 et u_2 (deux polynômes unitaires de degré 3). À partir des coefficients a,b,c,d,e,f, les quantités suivantes sont calculées, puis utilisées dans le calcul :

$$t_1 - t_2 = ae - bd,\ t_3 - t_4 = bf - ce,\ t_5 - t_6 = af - cd.$$

Il est possible de calculer ces quantités comme suit[5] :

$$\begin{gathered}p = af,\ q = cd,\ r = (a + b)(e + f),\ s = (b + c)(d + e),\ t = b(d + e + f),\\ ae - bd = r - p - t,\ bf - ce = q - s + t,\ af - cd = p - q.\end{gathered}$$

On obtient alors un coût de $5M$ contre $6M$ naïvement.

[5]Cette solution m'a été proposée par Michel Quercia (avril 2004)

Ces différentes optimisations aboutissent ainsi une réduction de $5M$ du coût proposé, soit une addition en $I + 65M + 6S$.

Pour le doublement, un gain similaire peut être obtenu sur les étapes 1, 6, et 8 de [218, table X]. Le produit $t_1 u'_3 = 2s_1^2 - as_1$ de l'étape 8 peut être obtenu gratuitement en réutilisant les données précédemment calculées pour s_1^2 et as_1. On obtient ainsi un gain de $3M$.

Ce travail d'implantation de l'arithmétique sur une jacobienne de courbe hyperelliptique de genre 3 a fait partie des précurseurs de la bibliothèque MPFQ, développée ultérieurement, et discutée au chapitre 15 de ce mémoire.

8.7.2 Factorisation de polynômes de degré 3

Une autre étape pour la mise en œuvre de l'algorithme développé au chapitre 8 est le test de friabilité. Une fois calculé un diviseur $D = D_{\alpha,\beta} = \alpha D_1 + \beta D_2$, il convient de déterminer si sa forme réduite s'écrit comme somme de diviseurs de poids 1. Ceci se détermine par l'examen du polynôme $u(x)$ de la représentation de Mumford : on souhaite déterminer rapidement si $u(x)$ se décompose en facteurs linéaires ou non. Pour ce faire, nous adoptons la procédure suivante.

- Génériquement, $u(x)$ est un polynôme unitaire de degré 3. Dans le cas exceptionnel où $u(x)$ est de degré inférieur, appliquer une procédure naïve (cas non critique) ;

- Calculer le discriminant Δ de $u(x)$ (pour des raisons pratiques, on calcule plutôt 27Δ). Génériquement, $u(x)$ est sans facteur multiple, donc $\Delta \bmod p$ est non nul. Dans le cas exceptionnel où $\Delta \equiv 0 \bmod p$, appliquer une procédure naïve (cas non critique).

- Appliquer le critère de Swan [205]. Le nombre de racines rationnelles d'un polynôme de degré 3 est égal à 0 ou 3 si et seulement si son discriminant Δ est un carré. On calcule donc le symbole de Legendre $\left(\frac{\Delta}{p}\right)$. Si le résultat n'est pas un carré, on sait que $u(x)$ ne se décompose pas de manière adaptée. Ce faisant, on parvient ainsi à évincer à peu de frais la moitié des cas possibles (la moitié des permutations de $\mathfrak{S}_3$ ont pour type $(1, 2)$).

Dans les cas restants, on sait que $u(x)$ est sans facteurs multiples, et possède zéro ou trois facteurs linéaires. Le produit des facteurs linéaires peut se décomposer, pour tout $h \in \mathbb{F}_q$, comme le produit :

$$\gcd(x^q - x, u(x)) = \gcd((x-h) \cdot ((x-h)^{(q-1)/2} - 1) \cdot ((x-h)^{(q-1)/2} + 1), u(x)).$$

Achever la factorisation de $u(x)$ requiert donc un travail d'ordonnancement des calculs d'exponentiation modulaire et de pgcd. Nous proposons l'ordonnancement suivant.

- Soit $h \in \mathbb{F}_q$ aléatoire. Pour des raisons d'efficacité, la première itération choisit $h = 0$.

- Calculer $u_h(x) = u(x + h)$. Exceptionnellement rarement, on peut avoir $u_h(0) = 0$, auquel cas il est aisé d'obtenir les deux racines restantes de u.

- Calculer $z_h(x) = x^{(q-1)/2} \bmod u_h(x)$, puis $z(x) = z_h(x - h)$. On a $z(x) = (x-h)^{(q-1)/2} \bmod u(x)$. Si $z(x) = \pm 1$, alors on sait que $u(x)$ se décompose en facteurs linéaires, mais ce choix de h ne permet pas d'obtenir les facteurs. Recommencer avec une autre valeur h. Si $z(x)$ est constant mais différent de ± 1, alors les deux polynômes $z(x) \pm 1$ sont premiers à $u(x)$, donc $u(x)$ n'a pas de racine dans $\mathbb{F}_q$.

– Calculer $\gcd(z(x)-1, u(x))$. Si $z(x)$ est de degré 0, alors $u(x)$ n'a pas de racine dans $\mathbb{F}_q$. Dans tout autre cas, $u(x)$ se décompose en facteurs linéaires, et ces facteurs sont alors aisément obtenus. On calcule $\gcd(z(x)+1, u(x))$. Les deux pgcd ainsi calculés sont de degré 1 et 2. Il est possible de calculer explicitement leurs racines.

Expérimentalement, en prenant pour exemple $q = 2^{27}$, la procédure complète de friabilisation a un coût de l'ordre de trois fois supérieur à celui de l'addition dans la jacobienne. L'opération coûteuse est le calcul de $z_h(x)$.

ENTRÉE : $\mathcal{C}$ courbe hyperelliptique de genre g sur $\mathbb{F}_q$.
D_1 diviseur de $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$; D_2 diviseur de $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$, tel que $D_2 \in \langle D_1 \rangle$.
SORTIE : m tel que $D_2 = mD_1$.
NOTATION : Soit $r = 1 - \frac{1}{g}$.

1. Soit $\mathcal{B} = \{P_1, \ldots, P_N\}$ une partie de $\mathcal{C}(\mathbb{F}_q)$ de cardinal $N = q^r$. Soit G un graphe sur l'ensemble de sommets $\mathcal{L} \cup \{*\}$, où $\mathcal{L} = \mathcal{C}(\mathbb{F}_q) - \mathcal{B}$. Initialement, G ne contient pas d'arête.
2. Tant que G ne contient pas d'arête :
   (a) Tirer aléatoirement deux entiers α_1 et α_2 dans $[0, \#\,\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)[$.
   (b) Soient $X \subset \mathcal{B}$ et $Y = \{P\} \subset \mathcal{L}$ tels que
   $$\#(X \cup Y) = k,\ \alpha_1 D_1 + \alpha_2 D_2 = \sum_{P \in X \cup Y} (P) - k(\infty). \tag{8.8}$$
   S'il n'existe pas de telle décomposition, revenir à l'étape 2.
   (c) Noter $\mathcal{R}$ la relation. Soit $e = *P$ l'arête correspondante. Insérer e dans G.
3. Tant que G a moins de $N_{\max} = q^{1-\frac{1}{g}+\frac{1}{g^2}}$ sommets :
   (a) Tirer aléatoirement deux entiers α_1 et α_2 dans $[0, \#\,\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)[$.
   (b) Soient $X \subset \mathcal{B}$ et $Y = \{P, Q\} \subset \mathcal{C}(\mathbb{F}_q) - \mathcal{B}$ vérifiant (8.8).
   S'il n'existe pas de telle décomposition, revenir à l'étape 3. Noter $\mathcal{R}$ la relation.
   (c) Soit e l'arête PQ. Si e n'est présente dans G, insérer e dans G.
4. Soit T l'ensemble des sommets de la composante connexe de $*$ dans G. Soit $C = 0$.
5. Tant que $C \leq \#\mathcal{B}$:
   (a) Tirer aléatoirement deux entiers α_1 et α_2 dans $[0, \#\,\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)[$.
   (b) Soient $X \subset \mathcal{B}$ et $Y \subset T$ vérifiant (8.8).
   S'il n'existe pas de telle décomposition, revenir à l'étape 5.
   (c) Incrémenter C. Utiliser T pour récrire la relation sous la forme :
   $$\alpha_1^{(i)} D_1 + \alpha_2^{(i)} D_2 = \sum_j n_j^{(i)} P_j - (\sum_j n_j^{(i)})(\infty).$$
6. Construire la matrice M de coefficients $(n_j^{(i)})$.
7. Déterminer v tel que $vM = 0$. Déduire $(\sum_i v_i \alpha_1^{(i)})D_1 + (\sum_i v_i \alpha_2^{(i)})D_2 = 0$, puis m.

ALGORITHME 8.9 : Logarithme discret sur $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ avec deux *large primes* (algorithme « simplifié »).

Chapitre 9

Logarithme discret sur les quartiques planes

Nous résumons dans les paragraphes qui suivent les travaux présentés dans [T9]. Ce travail est une collaboration avec C. Diem, auteur de [58] qui a proposé l'algorithme de départ. Dans [T9], nous nous sommes concentré sur le cas particulier du genre 3, et avons mené une analyse à la fois pratique et théorique sur ce cas d'importance cryptographique (il n'est pas exagéré de dire que [T9], pour une large part, a sonné le glas de l'utilisation des courbes de genre 3 en cryptographie).

9.1 Présentation

La problématique de ce chapitre est semblable à celle du chapitre précédent. On s'intéresse au calcul de logarithme discret en genre petit, sur la jacobienne d'une courbe $\mathcal{C}$ de genre fixé. On vise ici à obtenir une amélioration dans la majorité des cas.

Comme au chapitre 8, l'objectif du travail est la formulation d'un algorithme de calcul d'index. À cette fin, une base de facteurs est choisie. Cette base de facteurs est constituée de diviseurs de poids 1, et son cardinal vaut q^r pour une valeur de r qui est déterminée par l'analyse.

La différence entre l'algorithme que nous présentons ici et l'algorithme présenté au chapitre 8 réside dans la façon de construire des relations. Il est utile dans ce contexte d'évoquer l'algorithme de Hafner et McCurley pour le calcul de structure de groupes de classes dans les corps de nombres [97]. Les relations sont obtenues dans [97] par la factorisation d'idéaux principaux. Une jacobienne ayant une structure de groupe de classes, il est légitime de tenter une approche similaire. On considère une fonction, et le diviseur principal correspondant $\mathrm{div}(f)$, qu'on cherche à décomposer. Les facteurs sur $\mathbb{F}_q$ du polynôme $r(x) = \mathrm{Res}_y(f, F)$ permettent d'exhiber une décomposition de $\mathrm{div}(f)$ comme somme de diviseurs $\mathbb{F}_q$-rationnels. Si $r(x) = r_1(x) \dots r_k(x)$, on a :

$$\begin{aligned}\mathrm{div}(f) &= n_{1,1}(P_{1,1}) + \dots + n_{1,s_1}(P_{1,s_1}) + \dots + n_{k,1}(P_{k,1}) + \dots + n_{k,s_k}(P_{k,s_k}),\\ &= D_1 + \dots + D_k.\end{aligned}$$

Dans l'écriture ci-dessus, les diviseurs D_j sont les diviseurs dont le support est l'intersection de $\mathrm{div}(f)$ et $\mathrm{div}(r_j)$. Ceci revient à écrire dans $k(\mathcal{C})$:

$$\langle f \rangle = \langle r_1, f \rangle \dots \langle r_k, f \rangle.$$

Considérons le cas particulier d'une fonction linéaire sur $\mathbb{P}^2(\bar{k})$ donnée sous une forme affine $f = ax + by + c$. Le polynôme $\mathrm{Res}_y(f, F)$ est un polynôme univarié de $k[x]$ et a pour degré le degré total du polynôme F. Le poids du diviseur $\mathrm{div}(f)$ est le nombre de zéros de f sur la courbe $\mathcal{C}$, donc le degré total de F. La factorisation de $\mathrm{Res}_y(f, F)$ donne ainsi lieu à une décomposition de $\mathrm{div}(f)$ en une somme de diviseurs $\mathbb{F}_q$-rationnels, dont les poids sont donnés par les degrés des facteurs de $\mathrm{Res}_y(f, F)$. À supposer qu'on a un comportement uniforme pour $\mathrm{Res}_y(f, F)$, on s'attend

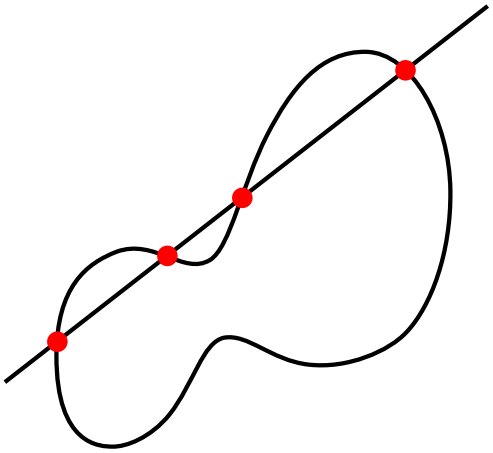


FIG. 9.1 : Intersection d'une quartique avec une droite.

ainsi asymptotiquement à avoir par exemple une probabilité de $1/(4!)$ d'avoir une décomposition $\mathrm{div}(f) = (P_1) + (P_2) + (P_3) + (P_4) - 4(\infty)$ où les points P_i sont $\mathbb{F}_q$-rationnels.

Pour donner un exemple qui sera par la suite central dans cette section, supposons que l'équation F définissant la courbe $\mathcal{C}$ soit de degré total 4. Le genre de la courbe $\mathcal{C}$ est alors génériquement égal à 3. L'intersection d'une droite $ax + by + c$ avec une telle quartique fait alors apparaître entre zéro et quatre points rationnels (comme illustré par la figure 9.1).

Supposons fixée une base de facteurs $\mathcal{B}$ constituée comme précédemment d'un ensemble de diviseurs $\mathbb{F}_q$-rationnels, de cardinal q^r avec $0 < r < 1$. Dans l'algorithme présenté au chapitre 8, obtenir une relation FF par exemple nécessite de recontrer g fois « par chance » un élément de $\mathcal{B}$, soit une probabilité en $\Theta(q^{(r-1)g})$. Supposons maintenant qu'on se place dans le cadre qu'on vient d'évoquer, avec F de degré total d. Choisissons deux diviseurs arbitraires de $\mathcal{B}$, correspondant chacun à un point de $\mathcal{C}(\mathbb{F}_q)$, et considérons l'unique fonction linéaire f passant par ces deux points. Les coordonnées des $d-2$ autres points d'intersection sur $\mathcal{C}(\bar{\mathbb{F}_q})$ sont solution d'une équation de degré $d-2$ sur $\mathbb{F}_q$. Heuristiquement, ces points sont tous $\mathbb{F}_q$-rationnels avec une probabilité $1/(d-2)!$. Plus important, ces points appartiennent tous à $\mathcal{B}$ avec une probabilité $\Theta(q^{(r-1)(d-2)})$. On obtient similairement la complexité d'obtenir des relations FP et PP. En reprenant l'analyse de [T7], on obtient alors un algorithme de calcul de logarithmes discrets de complexité $O(q^{2-2/(d-2)})$. Ce résultat est dû à Diem [58]. Une version synthétique de l'algorithme correspondant est donnée par l'algorithme 9.2.

9.2 Importance du degré du modèle

La complexité de l'algorithme proposée est donc sensible au degré du modèle plan choisi. Il souhaitable d'exprimer cette complexité par rapport à la taille du groupe qui nous intéresse, à savoir la jacobienne. Aussi, la question qui intervient est la détermination d'un modèle de degré minimal pour une courbe de genre g. Lorsqu'il est possible d'obtenir un modèle plan tel que $d \leq g+1$, alors la complexité ainsi obtenue est meilleure que la complexité $O(q^{2-2/g})$ de l'algorithme vu au chapitre 8.

Un algorithme polynomial probabiliste permettant de construire un tel modèle de degré $g+1$ pour une courbe *non hyperelliptique* de genre g a été proposé par Diem [58], résultant en un algorithme de calcul de logarithme discret sur les jacobiennes de courbes non hyperelliptiques dont la complexité est $O(q^{2-2/(g-1)})$.

ENTRÉE : $\mathcal{C}$ courbe de genre g sur $\mathbb{F}_q$, donnée par une équation F de degré d ;
D_1 et D_2 diviseurs de $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$, tels que $D_2 \in \langle D_1 \rangle$.
SORTIE : m tel que $D_2 = mD_1$.
NOTATION : Soit $r = 1 - \frac{1}{d-2}$.

1. Soit $\mathcal{B} = \{P_1, \ldots, P_N\}$ une partie de $\mathcal{C}(\mathbb{F}_q)$ de cardinal $N \approx 2q^r$.
2. Pour chaque paire $\{F_i, F_j\} \in \mathcal{B}$, déterminer l'équation linéaire L passant par F_i et F_j. Soit $D_{i,j}$ tel que $\mathrm{div}(L) = (F_i) + (F_j) + D_{i,j}$.
3. Construire un LP-graphe, alimenté par les relations obtenues par la procédure ci-dessus dans les cas où $D_{i,j}$ se décompose en deux diviseurs $\mathbb{F}_q$-rationnels de poids 1. Stopper lorsque le nombre de sommets du LP-graphe reliés à $*$ et à une distance de $*$ bornée par $(\log q)^2$, atteint la valeur plancher $q^{1-1/(g(d-2))}$. Soit T l'ensemble de ces sommets.
4. Obtenir au minimum $N+1$ relations entre D_1, D_2, et les éléments de $\mathcal{B}$ de la manière suivante.
   - Tirer aléatoirement α_1 et α_2 jusqu'à obtenir $\alpha_1 D_1 + \alpha_2 D_2$ décomposable sur $\mathcal{B} \cup T$. Une décomposition ainsi obtenue, en exploitant les relations du LP-graphe, se récrit en
   $$\alpha_{i,1} D_1 + \alpha_{i,2} D_2 = \sum_{1 \leq j \leq N} n_{i,j}(P_j).$$
5. Construire la matrice M de coefficients $(n_{i,j})$.
6. Déterminer v tel que $vM = 0$. Déduire $(\sum_i v_i \alpha_{i,1}) D_1 + (\sum_i v_i \alpha_{i,2}) D_2 = 0$, puis m.

ALGORITHME 9.2 : Logarithme discret sur $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ (petit degré).

9.3 Éléments d'analyse

L'algorithme 9.2 s'analyse selon les mêmes lignes que l'algorithme 8.9. Toutefois, quelques zones d'imprécision subsistent. Nous poursuivons l'objectif d'obtenir une analyse aussi rigoureuse que possible, sous les conditions suivantes :

- On étudie le cas d'une courbe de genre 3 donnée par une équation de degré 4.
- Le groupe $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ est supposé de cardinal premier (hypothèse simplificatrice).

On souhaite démontrer que l'algorithme aboutit en un temps moyen $\widetilde{O}(q)$. Pour cela, on veille en particulier à randomiser le choix de la base de facteurs $\mathcal{B}$ dans la première étape.

Nous ne donnons pas de détail sur les points de l'analyse qui ne révèlent aucune difficulté, et renvoyons à l'article [T9] ainsi qu'à [69] pour certains points de détail, notamment relatifs à la sensibilité de cet algorithme à la structure de $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$. Les points délicats de l'analyse sont les suivants.

- Dans l'étape 4, déterminer la probabilité que le diviseur $\alpha_1 D_1 + \alpha_2 D_2$ soit décomposable sur l'ensemble $\mathcal{B} \cup T$.
- Dans l'étape 3, déterminer la probabilité que la prise en considération de toutes les paires $\{F_i, F_j\}$ permette d'obtenir la taille requise pour T.

9.3.1 Probabilité de décomposition

Nous abordons d'abord le premier des points ci-dessus. Les entiers α_1 et α_2 étant choisis aléatoirement, il est clair que le diviseur $\alpha_1 D_1 + \alpha_2 D_2$ est un diviseur aléatoire de la jacobienne de $\mathcal{C}$

(on rappelle qu'on a fait l'hypothèse simplificatrice que $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ est un groupe de cardinal premier, donc *a fortiori* cyclique). Selon les hypothèses formulées par l'ébauche d'algorithme 9.2, le cardinal de l'ensemble T est au moins $q^{1-1/(g(d-2))}$ (et de même, *a fortiori*, pour $S = \mathcal{B} \cup T$). À l'aide de ces deux arguments, nous souhaitons connaître la proportion des diviseurs de la jacobienne de $\mathcal{C}$ que représentent les diviseurs dont la partie effective se décompose en somme d'éléments de $S = \mathcal{B} \cup T$.

Une approche heuristique est la suivante. Fixons un point rationnel arbitraire que nous notons $\infty \in \mathcal{C}(\mathbb{F}_q)$. Il y a $\Theta(q^g)$ classes de diviseurs dans la jacobienne de $\mathcal{C}$. Il est possible de composer $\binom{\#S+g}{g} = \Theta((\#S)^g)$ diviseurs effectifs de poids $k \leq g$ écrits sous forme de somme de points de S. À supposer l'injectivité de l'application qui à la somme $\sum_{i=0}^{k} P_i$ associe le diviseur $\sum_{i=0}^{k} P_i - k(\infty)$, on pourrait donc énoncer que le nombre des diviseurs décomposables comme on le souhaite est de la forme $\Omega((\#S)^g = g^{g-1/(d-2)})$, soit une probabilité de décomposition de l'ordre de $O(q^{-1/(d-2)})$, ce qui est satisfaisant pour les besoins de l'algorithme. Hélas, l'hypothèse d'injectivité est pour le moins hardie. On peut l'accepter comme une heuristique, mais dans la perspective d'une preuve plus rigoureuse, il est nécessaire de déterminer exactement la taille de l'image. C'est, dans le contexte $g = 3$ et $d = 4$, l'objet du lemme suivant. Pour cela, il convient de borner le nombre de *diviseurs spéciaux*. Ces derniers contiennent les points où l'application évoquée plus haut n'est pas injective. Nous prouvons dans [T9] le lemme suivant.

Lemme 9.3. *Soit $\mathcal{C}$ une courbe non hyperelliptique de genre 3 sur $\mathbb{F}_q$. Le nombre de diviseurs spéciaux est $\sim q^2$.*

Ce lemme permet d'appuyer le raisonnement heuristique que nous avons énoncé. Parmi les $\Omega(\#S)^g = \Omega(q^{5/2})$ diviseurs effectifs de degré 3 qui ont les propriétés désirées, au plus q^2 sont des diviseurs spéciaux. Dès lors, le nombre de diviseurs non spéciaux obtenus en combinant les éléments de S reste en $\Omega(q^{5/2})$. Puisqu'à chacun correspond une unique classe dans $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$, nous avons démontré qu'une proportion $\Omega(q^{-1/2})$ des classes de diviseurs de $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ est décomposable sur l'ensemble $S = \mathcal{B} \cup T$.

9.3.2 Taille atteinte par le LP-graphe

Il n'est pas a priori évident que la façon dont est construit le LP-graphe permet, en respectant la contrainte de profondeur maximale $(\log q)^2$, d'atteindre le nombre voulu de sommets $q^{1-1/(g(d-2))}$, soit dans le cas particulier que nous étudions, $q^{5/6}$.

Nous proposons l'approche suivante pour tenter de prouver ce résultat. Nous comparons le LP-graphe avec un graphe aléatoire selon un *modèle de Bernoulli*. Nous montrons qu'un tel graphe, soumis à des paramètres qui le rapprochent des caractéristiques essentielles du LP-graphe, possède la propriété voulue. Ensuite, nous comparons dans quelle mesure ce rapprochement est validé par les expérimentations pratiques.

Un graphe aléatoire selon un modèle de Bernoulli $\mathbb{G}(n, p)$ est défini sur un ensemble fixé de n sommets, où la probabilité qu'une arête joigne deux sommets est une constante p (indépendamment des autres paires de sommets). Pour ressembler au mieux au LP-graphe qui est l'objet de notre étude, il convient donc d'observer les propriétés d'un graphe aléatoire de Bernoulli $\mathbb{G}(n, p)$ de paramètres :

$$n = \#\mathcal{C}(\mathbb{F}_q) \approx q, \quad p = \frac{1}{2}\left(\frac{\#S}{\#\mathcal{C}(\mathbb{F}_q)}\right)^2 \approx \frac{2}{q}.$$

Nous rappelons que l'élément transformant notre LP-graphe en une variable aléatoire est la randomisation du choix de la base de facteurs à l'étape 1 de l'algorithme 9.2. Il convient toutefois d'apprécier

en quelle mesure cette comparaison avec un modèle de Bernoulli est inexacte, notamment au travers des points suivants :

- L'ensemble des sommets du LP-graphe est $\mathcal{L} \cup \{*\}$, qui n'est pas un ensemble fixé, mais variable (son cardinal, en revanche, est fixe).
- La probabilité de présence d'une arête entre deux sommets n'est pas indépendante des probabilité de présence des autres arêtes.
- Le sommet $*$ est différent des autres sommets, et son degré est plus important.

Un graphe de Bernoulli jouit de la propriété suivante.

PROPOSITION 9.4. *Soient c_1 et c_2 deux constantes. Considérons les propriété suivantes sur les graphes et sur les graphes pointés, notés respectivement G et (G, x).*

- $(\mathcal{Q}_{c_1,c_2})$ *Il existe un sous-graphe connexe de G ayant au moins $c_1 n$ sommets, et un diamètre au plus $c_2 \log n$.*
- $(\mathcal{Q}^x_{c_1,c_2})$ *Il existe un sous-graphe connexe de G ayant au moins $c_1 n$ sommets, un diamètre au plus $c_2 \log n$, et contenant le sommet x.*

Soit $c > 1$ une constante. Il existe des constantes positives c_1, c_2 telles que pour $p \geq \frac{c}{n}$, d'une part le graphe de Bernoulli $\mathbb{G}(n,p)$ satisfait $\mathcal{Q}_{c_1,c_2}$ avec probabilité tendant vers 1 (pour $n \to \infty$), et que d'autre part le graphe de Bernoulli pointé $(\mathbb{G}(n,p), x)$ satisfait $\mathcal{Q}^x_{c_1,c_2}$ avec probabilité $\Omega(1)$ (pour $n \to \infty$).

DÉMONSTRATION. La première proposition découle de la combinaison de deux résultats. D'une part, d'après [109, Theorem 5.4], il existe une constante c_1 pour laquelle avec probabilité tendant vers 1 (pour $n \to \infty$), le graphe $\mathbb{G}(n,p)$ a une composante connexe contenant au moins $c_1 n$ sommets. D'autre part, d'après [45], il existe une constante c_2 garantissant que pour $n \to \infty$, le graphe $\mathbb{G}(n,p)$ a un diamètre au plus $c_2 \log n$ avec probabilité tendant vers 1.

La seconde proposition découle de la première. Supposons le point x tiré uniformément aléatoirement parmi les sommets du graphe. Alors la probabilité que $\mathcal{Q}^x_{c_1,c_2}$ soit vérifiée est au moins c_1 fois la probabilité que $\mathcal{Q}_{c_1,c_2}$ soit vérifiée. Ceci suffit à montrer la proposition (énoncée pour un graphe pointé, donc avec x fixé), puisqu'à isomorphisme de graphe près, la propriété $\mathcal{Q}_{c_1,c_2}$ est invariante. ■

Les caractéristiques du LP-graphe nous incitent à comparer son comportement avec un graphe de Bernoulli $\mathbb{G}(n,p)$, où n et p sont donnés plus haut. On observe que leurs valeurs vérifient $p \sim \frac{2}{n}$, ce qui nous amène à suggérer la validité des conclusions de la proposition 9.4. Ces dernières nous amènent à supposer qu'avec une probabilité bornée inférieurement par une constante, un sous-graphe de cardinal $\Omega(q)$ existe, de profondeur $O(\log q)$. Il s'agit là d'une condition plus forte que ce qui est nécessaire pour l'analyse de complexité voulue, puisqu'il suffit de prouver l'existence d'un sous-ensemble de $\Omega(q^{5/6})$ sommets, à distance $O((\log q)^2)$, avec probabilité en $\Omega((\log q)^{-O(1)})$.

9.3.3 Comparaison expérimentale du LP-graphe et des graphes aléatoires

Afin d'apporter un soutien à l'heuristique que nous venons d'employer, nous pouvons *comparer* expérimentalement le comportement d'un LP-graphe, pour des courbes et des choix de bases de facteurs variés, avec le comportement d'un graphe aléatoire de Bernoulli. La procédure pour cette comparaison est la suivante. Pour diverses valeurs de q (nous avons choisi des puissances de 2),

q		$\mathsf{treedepth}(x, q^{5/6})$	$\mathsf{ccdepth}(x)$	c.c. géante (millions)	$\#\text{arêtes}/q$
2^{19}	LP-graphe	$14 \ldots \mathbf{16.8} \ldots 26$	$29 \ldots \mathbf{34.7} \ldots 47$	$0.41 \ldots \mathbf{0.42} \ldots 0.42$	$0.99 \ldots \mathbf{0.99} \ldots 1.00$
	G aléatoire	$14 \ldots \mathbf{16.8} \ldots 26$	$30 \ldots \mathbf{34.5} \ldots 42$	$0.41 \ldots \mathbf{0.42} \ldots 0.42$	$0.99 \ldots \mathbf{0.99} \ldots 1.00$
2^{20}	LP-graphe	$15 \ldots \mathbf{17.6} \ldots 28$	$32 \ldots \mathbf{36.4} \ldots 49$	$0.83 \ldots \mathbf{0.83} \ldots 0.84$	$0.99 \ldots \mathbf{1.00} \ldots 1.00$
	G aléatoire	$15 \ldots \mathbf{17.6} \ldots 28$	$32 \ldots \mathbf{36.5} \ldots 48$	$0.83 \ldots \mathbf{0.83} \ldots 0.83$	$0.99 \ldots \mathbf{1.00} \ldots 1.00$
2^{21}	LP-graphe	$15 \ldots \mathbf{18.4} \ldots 30$	$33 \ldots \mathbf{38.2} \ldots 48$	$1.67 \ldots \mathbf{1.67} \ldots 1.67$	$1.00 \ldots \mathbf{1.00} \ldots 1.00$
	G aléatoire	$16 \ldots \mathbf{18.4} \ldots 29$	$33 \ldots \mathbf{38.1} \ldots 48$	$1.66 \ldots \mathbf{1.67} \ldots 1.67$	$1.00 \ldots \mathbf{1.00} \ldots 1.00$
2^{22}	LP-graphe	$16 \ldots \mathbf{19.2} \ldots 28$	$35 \ldots \mathbf{39.8} \ldots 51$	$3.33 \ldots \mathbf{3.34} \ldots 3.34$	$1.00 \ldots \mathbf{1.00} \ldots 1.00$
	G aléatoire	$16 \ldots \mathbf{19.1} \ldots 28$	$35 \ldots \mathbf{39.7} \ldots 50$	$3.33 \ldots \mathbf{3.34} \ldots 3.34$	$1.00 \ldots \mathbf{1.00} \ldots 1.00$
2^{23}	LP-graphe	$17 \ldots \mathbf{20.0} \ldots 29$	$37 \ldots \mathbf{41.7} \ldots 53$	$6.67 \ldots \mathbf{6.68} \ldots 6.68$	$1.00 \ldots \mathbf{1.00} \ldots 1.00$
	G aléatoire	$17 \ldots \mathbf{20.0} \ldots 34$	$36 \ldots \mathbf{41.6} \ldots 55$	$6.67 \ldots \mathbf{6.67} \ldots 6.68$	$1.00 \ldots \mathbf{1.00} \ldots 1.00$
2^{24}	LP-graphe	$18 \ldots \mathbf{20.8} \ldots 31$	$39 \ldots \mathbf{43.3} \ldots 53$	$13.35 \ldots \mathbf{13.36} \ldots 13.37$	$1.00 \ldots \mathbf{1.00} \ldots 1.00$
	G aléatoire	$18 \ldots \mathbf{20.8} \ldots 29$	$38 \ldots \mathbf{43.3} \ldots 52$	$13.35 \ldots \mathbf{13.36} \ldots 13.36$	$1.00 \ldots \mathbf{1.00} \ldots 1.00$

TAB. 9.5 : Comparaison du LP-graphe et d'un graphe aléatoire

nous avons construit 160 LP-graphes, obtenu à partir de 10 choix aléatoires de bases de facteurs pour 16 courbes distinctes arbitraires, choisies aléatoirement. Ces LP-graphes ont été construits en omettant les relations FP (qui lient les *large primes* au sommet $*$). La raison de cette omission est que les relations FP ont asymptotiquement un poids négligeable, mais sont susceptibles de perturber les interprétations des observations expérimentales pour les petites tailles.

En comparaison, pour les mêmes valeurs de q, nous avons construit des graphes aléatoires de paramètres comparables.[1]

Pour chacun des graphes ainsi construits, nous avons mesuré différents paramètres que nous avons jugés pertinents. Pour un graphe G et un sommet x, nous définissons :

$$\begin{aligned} N_k(x) &= \{y \in G,\ d_G(x,y) \leq k\}, \\ \mathsf{treedepth}(x, S) &= \min\{k \,|\, \#N_k(x) \geq S\}, \\ \mathsf{ccdepth}(x) &= \max\{k \,|\, N_k(x) \supsetneq N_{k-1}(x)\}. \end{aligned}$$

On observe aisément que si x appartient à une composante connexe Γ, alors :

$$\mathsf{ccdepth}(x) \leq \text{diameter}(\Gamma) \leq 2\mathsf{ccdepth}(x).$$

Sur la base de ce constat, nous utilisons ccdepth comme une mesure indirecte du diamètre de la composante connexe géante.

Une autre mesure faite sur les différents graphes construits est $\mathsf{treedepth}(x, q^{5/6})$. En effet, nous savons prouver des résultats concernant cette grandeur pour les graphes aléatoires, mais pas pour les LP-graphes. Il n'en reste pas moins que cette grandeur a une importance capitale dans l'analyse de complexité de l'algorithme 9.2, ce qui justifie la mesure de l'accord expérimental entre le comportement observé pour les LP-graphes, et celui observé pour les graphes aléatoires.

La table 9.5 rassemble les données collectées dans cette comparaison. Pour chaque donnée, nous indiquons les valeurs extrémales, ainsi que la moyenne. Nous considérons que la table 9.5 apporte un argument de justification à notre assimilation du LP-graphe à un graphe aléatoire.

[1] Les détails du mode de tirage aléatoire de ces graphes et les discussions d'efficacité de cette procédure sont donnés dans [T9].

9.4 Calculs menés et portée cryptographique

L'algorithme présenté dans ce chapitre peut être vu comme une version améliorée de celui présenté au chapitre précédent. Aussi, l'impact cryptographique de ces deux approches voisines doit-il être jugé notamment à l'aune de l'évaluation des performances de cet algorithme amélioré. Nous avons implanté une version de l'algorithme 9.2. La différence entre l'algorithme tel qu'il a été présenté et tel qu'il a été implanté tient à une différence d'objectif. La présentation que nous avons faite avait en particulier pour but le fait de prouver un résultat raisonnablement rigoureux, s'appuyant essentiellement sur une hypothèse de similarité du LP-graphe avec un certain modèle de graphe aléatoire. Dans la perspective d'une mise en pratique, l'objectif est autre : il s'agit uniquement d'obtenir un résultat, assez rapidement, pour un problème donné. Aussi, si l'algorithme 9.2 peut être vu comme une adaptation de l'algorithme 8.9, l'algorithme que nous avons implémenté emprunte davantage à l'algorithme 8.5. Les différences entre nos choix d'implémentation et l'algorithme 9.2 sont les suivants :

- Le choix de la base de facteurs n'est pas fait au hasard. Nous choisissons plutôt l'ensemble des points de $\mathcal{C}(\mathbb{F}_q)$ dont l'abscisse x est représentée par un entier de l'intervalle $[0, B]$ pour une certaine borne B. La borne $B = \frac{4}{3}\sqrt{2}q$ s'est avérée suffisante.
- La collection de relations à fin de construction de la matrice n'attend pas que le graphe atteigne une certaine taille. Au contraire, à l'instar du comportement décrit dans l'algorithme 8.5, tout cycle rencontré au fur et à mesure de la croissance du graphe est potentiellement conservé pour ajouter une relation au système linéaire à résoudre.
- Le processus d'évolution du graphe passe par une étape de « filtrage », destinée à limiter l'attention à un sous-ensemble de l'ensemble des relations partielles collectées, possédant la propriété de contenir tous les cycles. Cette étape de « filtrage », bien que moins sophistiquée que le filtrage de NFS (peu évoqué dans ce mémoire, mais voir notamment [39]), en partage les étapes initiales.

L'exemple le plus important calculé dans [T9] (calcul réalisé à l'été 2006) est le calcul de logarithmes discrets dans la jacobienne de la courbe $\mathcal{C}$ d'équation :

$$Y^4 + Y^3 + Y^2 + X^2Y + X^3 + X + 1 = 0$$

définie sur le corps $\mathbb{F}_{2^{31}}$. La jacobienne de cette courbe sur $\mathbb{F}_q$ comporte environ 2^{93} éléments. La résolution du problème du logarithme discret dans ce groupe a occupé moins d'une semaine de temps à une machine de type AMD Opteron 250. Quelques détails supplémentaires sur ce travail de calcul sont donnés dans [T9].

La portée cryptographique de [T9] a été importante. En effet, comme nous l'avons déjà mentionné (p. 90), la thèse de Gaudry en 2000 avait laissé ouverte la question de savoir s'il existait des algorithmes significativement meilleurs que les algorithmes « génériques » pour résoudre le logarithme discret sur les courbes de genre 3, faisant de ces dernières des candidats potentiellement intéressants à exploiter pour des utilisations cryptographiques. Plusieurs travaux, dans les années qui ont suivi, ont proposé l'utilisation de diverses familles de courbes de genre 3 et au-delà, notamment les courbes $\mathcal{C}_{a,b}$, étudiées entre autres par [10, 11, 12, 15] (l'exemple utilisé plus haut appartient à cette famille de courbes), ou encore les courbes superelliptiques, en particulier les courbes de Picard, étudiées par [73, 17, 16].

L'article [T9] a montré que pour la plupart des courbes de genre 3, le calcul du logarithme discret dans la jacobienne n'était pas plus difficile que dans la jacobienne d'une courbe de genre 2 sur le même corps de base. L'expérimentation confirme cette situation. Ce résultat s'applique pleinement aux classes de courbes que nous venons de lister. L'éventuel avantage comparatif de ces classes de courbes s'est donc transformé en un inconvénient a priori. En effet, sauf à justifier de l'emploi de propriétés *spécifiques* aux courbes de genre 3, il semble inutile de considérer des courbes de genre 3 pour une utilisation cryptographique, lorsque des courbes de genre 2 font l'affaire.

Nous mentionnons enfin un aspect parfois cité comme une singularité du résultat donné par [58, T9]. Ce sont les *cas particuliers* des courbes de genre 3 *hyperelliptiques* qui résistent à l'attaque présentée, tandis que le cas générique est sujet à l'attaque. Cette situation tranche avec l'habitude, où le cas général est le plus souvent plus résistant que des cas particuliers. Au-delà du constat de singularité, cette situation a donné lieu à divers travaux tentant de « transporter » le problème du logarithme discret d'une instance « difficile », à savoir une jacobienne de courbe de genre 3 hyperelliptique, vers une instance « facile », à savoir une jacobienne de courbe de genre 3 *non* hyperelliptique, et donc sujette à l'attaque présentée. L'outil pour réaliser ce genre de transport est le calcul d'isogénies [200]. Le transport est effectif au moins dans une une fraction constante des cas.

Chapitre 10

Genre grand et petit degré

Ce chapitre reprend brièvement les résultats de [T15], en collaboration avec A. Enge et P. Gaudry, qui s'articulent notamment autour de présentations déjà détaillées en partie dans le contexte des algorithmes de descente pour le calcul de logarithme discret (voir section 3.5), ou encore les travaux des chapitres précédents (chapitres 8 et 9).

10.1 Rappel du contexte

Nous avons mentionné en 7.3 que la résolution du problème du logarithme discret sur les jacobiennes de courbes est un problème qui nécessite de fixer le point de vue. Implicitement, un algorithme s'applique à une famille de courbes $(\mathcal{C}_i)_i$, définies sur $\mathbb{F}_{q_i}$ et de genre g_i. Étant donné un algorithme, le point de vue est l'énoncé de restrictions sur cette famille de courbes. Nous avons étudié au chapitre 8 le contexte où g est fixé, et q croît. Au chapitre 9, nous avons ajouté la contrainte additionnelle que les courbes soient données par un modèle plan de degré au plus $g+1$ (ou, de manière équivalentes, ne soient pas hyperelliptiques).

Pour le cas de courbes où g n'est pas borné, les résultats sont différents. Nous avons mentionné précédemment, notamment en 7.4, l'algorithme d'Adleman-DeMarrais-Huang [5] qui introduit le *poids* d'un diviseur représentant un élément de la jacobienne d'une courbe comme un élément déterminant la *taille*. En qualifiant de « petits » les éléments d'une jacobienne J pouvant être représentés par des diviseurs de poids $\sqrt{g}$, l'algorithme d'Adleman-DeMarrais-Huang atteint une complexité en $L_{\#J}[1/2, O(1)]$ pour le calcul de logarithmes discrets. Plusieurs travaux ont fait suite à cet algorithme, afin d'en préciser quelques hypothèses, et d'étudier quelques améliorations et généralisations possibles [69, 55, 105].

L'algorithme d'Adleman-DeMarrais-Huang peut être vu comme un analogue de l'algorithme d'Adleman [1] pour le calcul de logarithmes discrets dans les corps finis. Cependant, alors que ce dernier est amélioré par des versions plus rapides en $L[1/3]$ (complexité heuristique), telles que les algorithmes FFS et NFS-DL, ou NFS-HD évoqués au chapitre 3, la question de l'existence de versions plus rapides, permettant d'atteindre une complexité heuristique $L[1/3]$ pour le calcul de logarithmes discrets dans les jacobiennes de courbes de genre grand, est longtemps restée en suspens.

L'article [T15] remplit cet objectif. Il est possible de spécifier une classe de courbes pour laquelle le logarithme discret est calculable en temps $L[1/3]$, sous les heuristiques usuelles, qu'on peut rapprocher de l'heuristique 2.7 pour l'analyse de NFS. Pour cela, nous supposons qu'il existe un modèle plan de degré au plus $g^{2/3}$. Il s'agit là d'une contrainte qu'on peut juger forte. Par exemple, les courbes hyperelliptiques ne remplissent pas cette condition.

L'algorithme que nous proposons partage de nombreux éléments d'ossature avec l'algorithme FFS, la courbe sur laquelle nous souhaitons calculer des logarithmes discrets jouant le rôle de la courbe définissant le « côté algébrique » de FFS. Avant d'en donner une description, nous définissons le modèle de friabilité qui intervient dans l'algorithme.

Nous proposons une revue des traits essentiels de l'algorithme présenté dans [T15], sans aborder

les détails.

10.2 Friabilité

Nous considérons dans ce chapitre des courbes de genre g, définies sur un corps fini $\mathbb{F}_q$ données par une équation plane absolument irréductible :

$$\mathcal{C} : F(X, Y) = 0.$$

Dans le contexte le plus général, [104] donne les éléments pour réaliser en temps polynomial les opérations arithmétiques sur de telles courbes. En particulier, une opération qui est importante pour notre contexte est la décomposition d'un diviseur effectif du corps de fonction de $\mathcal{C}$ comme somme de places. Cette opération peut être réalisée en temps polynomial, et se réduit à une factorisation de polynômes.

Afin d'éviter des complications inutiles, nous considérons que les jacobiennes dans lesquelles nous travaillons sont toujours cycliques, et nous omettons même toute considération relative aux possibles petits facteurs premiers divisant son cardinal. Une discussion sur le traitement de telles situations apparaît dans [69, T15].

Nous définissons une base de facteurs $\mathcal{F}$ sur de telles courbes. À quelques détails près indiqués plus bas, nous considérons pour cette base de facteurs l'ensemble des places de degré borné par μ. Un diviseur effectif de degré ν donné est dit $\mathcal{F}$-friable, ou μ-friable, s'il se décompose comme une somme de places de $\mathcal{F}$. Un point crucial de notre étude est la probabilité de friabilité. Le résultat de la proposition 2.6 page 19 se généralise à ce contexte (voir [166, 168], et pour un cas à la fois plus général et remplissant plus précisément les besoins de notre cadre d'étude, [105, Theorem 13]). Afin d'en obtenir une formulation qui convienne à notre usage, nous faisons appel à la fonction L déjà utilisée en 2.2.1. Sous des conditions qui nous installent dans le contexte « genre grand », à savoir essentiellement $g \geq (\log q)^\rho$ pour une certaine constante ρ, nous avons le résultat suivant.

Proposition 10.1 ([T15, Prop. 2]). *Soient α, β tels que $0 < \beta < \alpha \leq 1$, satisfaisant les bornes de* [T15, Prop. 2 et 3]*, et $c, d > 0$. Soit :*

$$\nu = \left\lfloor \log_q L_{q^g}[\alpha, c] \right\rfloor \quad \text{et} \quad \mu = \left\lfloor \log_q L_{q^g}[\beta, d] \right\rfloor.$$

Alors la probabilité qu'un diviseur effectif de degré ν soit μ-friable est au moins :

$$L_{q^g}\left[\alpha - \beta, -\frac{c}{d}(\alpha - \beta) + o(1)\right]$$

où le terme $o(1)$ tend vers 0 avec $q^g \to \infty$.

La constitution de la base de facteurs $\mathcal{F}$ est corrigée par deux détails.

- Les places de degré borné par μ incluent des places dont le degré d'inertie (relativement à l'extension de corps de fonctions $k(\mathcal{C})/k(X)$) est supérieur strictement à 1. Nous *excluons* ces places de la base de facteurs, pour des raisons pratiques. Ceci n'a pas d'incidence sur le résultat de friabilité, car ces places sont négligeables en nombre (au plus $\approx q^{\mu/f}$ ont un degré d'inertie f ou plus parmi les $\approx q^\mu$ places de degré au plus μ).

- Pour capturer les éventuelles singularités ou multiples places à l'infini sur les courbes considérées, il convient d'ajouter dans la base de facteurs un nombre fini de places. Ces ajouts ne modifient pas le résultat de friabilité.

Nous notons enfin qu'une expression comme $\log_q L_{q^g}[\alpha, c]$ est grossièrement voisine de cg^α. Plus précisément, si on définit : $\mathcal{M}_{q^g} = \log_q(g \log q)$, alors on a

$$\log_q L_{q^g}[\alpha, c] = cg^\alpha \mathcal{M}^{1-\alpha} = c(g/\mathcal{M})^\alpha \mathcal{M}.$$

La grandeur $\mathcal{M}$ agit donc comme un « facteur correctif ».

D'autre part, nous rencontrons dans les paragraphes qui suivent des produits de degrés. Une difficulté d'écriture tient au fait que les degrés rencontrés s'expriment souvent sous la forme $\log_q L_{q^g}[\alpha, c]$, et que le produit de deux telles expressions ne peut s'écrire confortablement sous la même forme. Aussi, l'arithmétique avec de telles expressions mêle-t-elle différents types d'écriture. Nous faisons par exemple usage de l'identité suivante :

$$\left(\log_q L_{q^g}[\alpha, c]\right) \times d(g/\mathcal{M})^\beta = \log_q L_{q^g}[\alpha\beta, cd].$$

10.3 Familles de courbes traitées, et recherche de relations

Notre algorithme aspire à résoudre sur problème du logarithme discret sur une famille de courbes déterminée par certaines conditions. L'énoncé suivant, qui reprend [T15, Theorem 7], réalise la première étape du calcul de logarithmes discrets, à savoir le précalcul d'une base de facteurs.

Théorème 10.2. *Soit $(\mathcal{C}_i)_{i\in\mathbb{N}}$ une famille de courbes planes de genre g_i définies sur $\mathbb{F}_{q_i}$, d'équations $H_i(X, Y)$. Soient n_i et d_i les degrés en Y et X de H_i, respectivement. On suppose l'existence de constantes $\kappa > 0$ et $\rho \geq 2$ telles que*[1] *:*

$$\begin{aligned} &\frac{n_i d_i}{g_i} \leq \kappa \\ &\frac{n_i}{(g_i/\mathcal{M}_i)^{1/3}} \to \infty, \frac{d_i}{(g_i/\mathcal{M}_i)^{1/3}} \to \infty \text{ avec } \mathcal{M}_i = \frac{\log(g_i \log q_i)}{\log q_i} \\ &g_i \geq (\log q_i)^\rho \end{aligned} \tag{10.3}$$

Soit $b = (8\kappa/9)^{\frac{1}{3}}$. Il existe un algorithme calculant une base de facteurs de cardinal $L_{q_i^{g_i}}[1/3, b]$, ainsi que les logarithmes discrets de ses éléments, heuristiquement en temps moyen

$$L_{q_i^{g_i}}[1/3, c + o(1)] \text{ avec } c = \sqrt[3]{\frac{64\kappa}{9}}.$$

Démonstration. La preuve de cet énoncé est donnée constructivement par l'énoncé des paramètres de l'algorithme. Nous en donnons les éléments essentiels, ainsi qu'une analyse.

La base de facteurs est choisie comme indiqué plus haut, relativement à une borne de friabilité (notée plus haut μ) choisie sous la forme $\log_q L_{q^g}[1/3, b]$ pour une certaine constante b déterminée par l'analyse. Pour obtenir des relations liant les diviseurs de $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$, nous considérons les diviseurs principaux associés à des fonctions $\phi(X, Y)$ qui satisfont les contraintes de degré suivantes, où les constantes ν et δ sont déterminées par l'analyse :

$$\deg_Y \phi \leq \nu \frac{n}{(g/\mathcal{M})^{1/3}}(1 + o(1)) \text{ et } \deg_X \phi \leq \delta \frac{\kappa g/n}{(g/\mathcal{M})^{1/3}}(1 + o(1)).$$

[1] La formulation $\frac{n_i}{(g_i/\mathcal{M}_i)^{1/3}} \to \infty$ revient à préciser la considération informelle $n \approx g^\alpha$, avec $\frac{1}{3} < \alpha < \frac{2}{3}$.

Sous ces contraintes, le degré de la partie affine du diviseur $\operatorname{div}\phi$ vérifie :

$$\begin{aligned}\deg \mathrm{Res}_Y(\phi, \mathcal{C}) &\leq \deg_X \phi \deg_Y \mathcal{C} + \deg_Y \phi \deg_X \mathcal{C} \\ &\leq \left(\delta\kappa g^{2/3}\mathcal{M}^{1/3} + \nu n d g^{-1/3}\mathcal{M}^{1/3}\right) \cdot (1 + o(1)) \\ &\leq \kappa(\delta + \nu + o(1)) g^{2/3}\mathcal{M}^{1/3} \qquad \text{(voir équation 10.3)} \\ &= \log_q L(2/3, \kappa(\delta + \nu + o(1))).\end{aligned}$$

Nous pouvons appliquer la proposition 10.1, donnant la probabilité de friabilité d'un tel diviseur sous l'hypothèse heuristique d'usage assimilant son comportement à celui d'un diviseur aléatoire effectif de même degré. En combinant ceci au dénombrement du nombre de fonctions ϕ satisfaisant les contraintes de degré requises, nous pouvons obtenir le nombre de relations collectées.

$$\begin{aligned}\mathrm{Prob}(\text{friabilité}) &= L_{q^g}\left[\frac{1}{3}, -\frac{\kappa(\delta+\nu)}{3b} + o(1)\right], \\ \#\{\phi\} &= q^{\kappa\nu\delta g^{\frac{1}{3}}\mathcal{M}^{\frac{2}{3}}} = L_{q^g}\left[1/3, \kappa\nu\delta\right], \\ \#\{\phi \text{ friable}\} &= L_{q^g}\left[1/3, \kappa\left(\nu\delta - \frac{\nu+\delta}{3b}\right) + o(1)\right].\end{aligned}$$

La valeur idoine pour les paramètres b, ν, δ est une considération d'optimisation. Les contraintes sont les contraintes usuelles. Nous souhaitons minimiser le nombre de fonctions ϕ à tester, tout en garantissant que le nombre de relations produites dépasse le cardinal de la base de facteurs. Enfin, nous souhaitons équilibrer le poids de la phase de collecte de relations avec celui de la résolution du système linéaire. L'ensemble de ces conditions amène à minimiser b sous les contraintes :

$$\kappa\nu\delta - \frac{\kappa(\nu+\delta)}{3b} = b \quad \text{et} \quad \kappa\nu\delta = 2b.$$

La solution optimale, calculée dans [T15], donne, comme énoncé plus haut :

$$\begin{aligned}\nu = \delta = \sqrt[3]{\frac{8}{3\kappa}}, \quad b = \sqrt[3]{\frac{8\kappa}{9}}, \\ t_{\text{rel. collec.}} = t_{\text{linalg}} = L_{q^g}\left[1/3, (64\kappa/9)^{\frac{1}{3}} + o(1)\right].\end{aligned}$$

■

10.4 Calcul des logarithmes individuels

Pour déterminer le logarithme discret modulo ℓ d'un diviseur arbitraire de $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$, nous avons recours à une procédure de *descente* semblable à ce qui a été introduit en 3.5. L'énoncé suivant complète celui du théorème 10.2 plus haut :

Théorème 10.4. *Sous les conditions du théorème 10.2, une fois calculés les logarithmes discrets des éléments de la base de facteurs, le logarithme discret d'un élément quelconque de* $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ *peut heuristiquement être calculé en temps* $L_q[1/3, (8\kappa/9)^{\frac{1}{3}} + \epsilon]$ *pour tout* $\epsilon > 0$.

Démonstration. Nous pouvons, comme cela a été fait en 3.5, séparer la descente en une phase d'amorce et une phase de pas. Toutefois, le contexte particulier que nous étudions nous permet

d'aller jusqu'à l'éviction complète de la phase d'amorce. Aussi nous intéressons-nous au problème de la récriture, dans $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$, d'une place Q de degré au plus

$$D = \log_q L_{q^g}[1/3 + \tau, c] = c(g/\mathcal{M})^{\frac{1}{3}+\tau}\mathcal{M}$$

pour des paramètres c et τ vérifiant initialement $\tau = \frac{2}{3}$ et $c = 1$. Nous imposons en outre à Q d'être de degré d'inertie 1 (comme dit plus haut, cette restriction est insignifiante. Au besoin, randomiser le problème suffit à se ramener au cas que nous étudions). Nous souhaitons récrire Q comme linéairement équivalente à une somme de places satisfaisant une contrainte plus stricte sur le degré.

La consultation des détails qui apparaissent dans [T15] donne le résultat suivant. Il est possible, heuristiquement en temps $L_{q^g}[1/3, e + o(1)]$, de calculer une fonction ϕ telle que $\operatorname{div}\phi - Q$ est D'-friable, avec :

$$D' = \log_q L_{q^g}\left[1/3 + \frac{\tau}{2}, c'\right] \quad \text{et} \quad c' = \frac{2\sqrt{\kappa}}{3e}\sqrt{c+e}.$$

Il est possible de répéter cette opération. Nous obtenons alors une suite d'étages de descente, où les degrés après i étages s'écrivent sous la forme

$$\log_q L_{q^g}\left[\frac{1}{3} + \frac{1}{3\cdot 2^{i-1}}, c_i\right].$$

On démontre ensuite dans [T15] que pour toute constante positive ξ, à l'issue de $O(\log\log g)$ étages, il est possible d'atteindre un degré

$$\log_q L_{q^g}[1/3, c_\infty(1+\xi)],$$

où c_∞ peut être déterminé exactement en fonction de κ et e. Pour atteindre $c_\infty = b$, qui correspond à l'énoncé du théorème 10.4, l'analyse montre qu'il faut choisir $e = b$, ce qui rejoint le résultat annoncé par le théorème 10.4. ■

10.5 Généralisations

Il est possible d'examiner les cas laissés de côté par l'analyse qui vient d'être faite. Notamment, [T15] étudie les cas où l'hypothèse $\frac{n}{(g/\mathcal{M})^{1/3}} \to \infty$ n'est pas vérifiée.

Un cas assez intéressant semble à première vue traité de façon sous-optimale par l'algorithme proposé. Considérons en effet la classe des courbes $C_{a,b}$, soumises aux contraintes de degré

$$a = g^\alpha \mathcal{M}^{1/2-\alpha} \quad \text{et} \quad b = g^\alpha \mathcal{M}^{\alpha-1/2}$$

pour une constante α vérifiant $1/3 < \alpha < 2/3$. En l'absence de singularités affines, ces courbes sont de genre $(a-1)(b-1)/2$, d'où pour cette famille une constante $\kappa = 2$. En appliquant les résultats des théorèmes 10.2 et 10.4, il appert que la complexité heuristique du calcul de logarithmes discrets est dans ce cas de $L_{q^g}[1/3, (128/9)^{1/3} + o(1)]$.

Un examen plus précis montre que dans ce cas précis, il est possible d'atteindre une complexité qui égale celle de l'algorithme GNFS, avec une constante $(64/9)^{1/3}$. Pour cela, il est important de tirer parti de la forme particulière des courbes $C_{a,b}$. Les monômes des fonctions ϕ considérées sont soumis à une contrainte plus précise que dans la preuve du théorème 10.2. Plutôt que d'employer deux bornes indépendantes pour les degrés en X et Y, les monômes X^iY^j apparaissant dans ϕ sont soumis à la contrainte

$$ai + bj \leq \lambda g^{2/3}\mathcal{M}^{1/3}$$

où le paramètre λ est introduit pour jouer conjointement le rôle des paramètres ν et δ qui apparaissent dans la preuve du théorème 10.2. Un calcul attentif des termes du résultant permet d'obtenir une borne meilleure, qui se transcrit *in fine* par une amélioration de la complexité heuristique, atteignant :

$$L_{q^g}[1/3, (64/9)^{1/3} + o(1)].$$

Partie III

Algèbre linéaire

Chapitre 11

Algèbre linéaire pour les algorithmes de crible

Cette partie traite de la résolution de systèmes linéaires creux pour les applications de cryptanalyse, notamment la factorisation d'entiers et le calcul de logarithmes discrets dans les corps finis. Des applications plus exotiques dans le contexte de la cryptanalyse font appel aux mêmes outils de résolution, comme par exemple les problèmes examinés dans [T8, T12]. Nos travaux sur ces thématiques sont particulièrement présents dans les articles [T2, T3, T13, T16, T14]. Une part du travail décrit est aussi présente dans le logiciel CADO-NFS, ainsi que sous la forme de manuscrits en préparation.

Ce chapitre ouvre la partie III. Les éléments présentés ici ont vocation à décrire le contexte général dans lequel s'inscrivent nos travaux.

11.1 Présentation du problème

11.1.1 Systèmes linéaires rencontrés

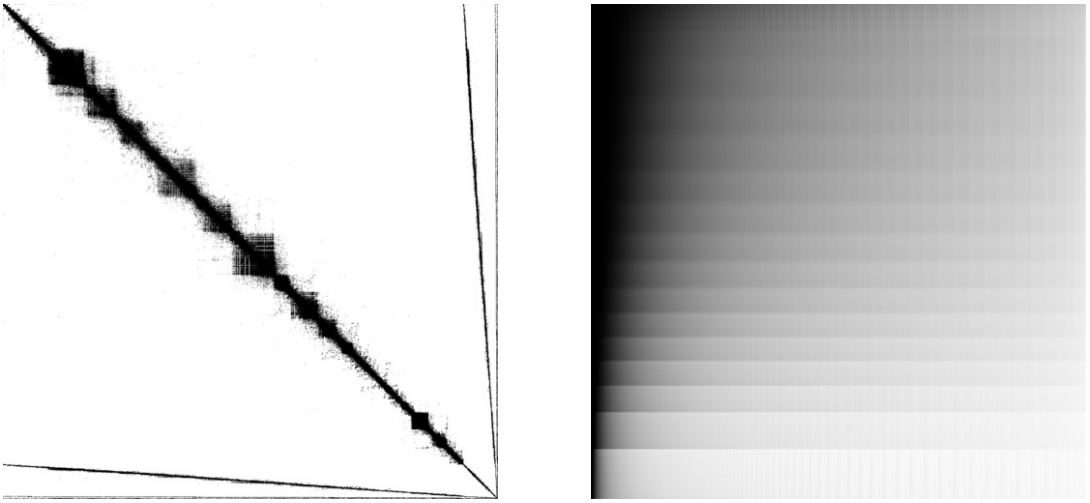
On s'intéresse à un système linéaire à N équations et inconnues, défini par une matrice M, auquel on recherche (par exemple) une solution homogène, c'est-à-dire un vecteur w non trivial tel que

$$Mw = 0.$$

Dans ce cas précis, nous avons supposé implicitement que la matrice M définissant le système était singulière. La résolution de systèmes inhomogènes (avec M non singulière, éventuellement), est également importante. Les systèmes linéaires *creux* (où M a peu de coefficients non nuls) sont partout. La résolution d'équations aux dérivées partielles par la méthode des éléments finis est probablement le pourvoyeur majeur de systèmes linéaires creux. Une grande expertise, et des ressources de calcul importantes sont consacrées à ce type de problèmes. Les systèmes linéaires que nous rencontrons dans le contexte de la cryptanalyse sont fondamentalement différents. Nous donnons ici quelques éléments de contexte permettant de distinguer les problèmes.

Tout d'abord, les problèmes que nous traitons sont définis sur des corps finis. Dans le contexte de la factorisation, on a à traiter des systèmes linéaires définis sur $\mathbb{F}_2$, et dans le contexte du calcul de logarithmes discrets le corps de définition est un corps premier $\mathbb{F}_p$ pour p de taille assez importante (plusieurs centaines de bits). Les coefficients du système, dans ce dernier cas, ne sont toutefois pas, le plus souvent, des entiers de l'ordre de grandeur de p, mais plutôt des entiers petits, typiquement de l'ordre de $O(\log p)$.

Travailler dans un corps fini a en particulier pour implication que les solutions recherchées sont par essence *exactes*. Il n'y a *pas*, sur $\mathbb{F}_p$, de notion de *solution approchée*, ou de *convergence*. Lorsque des méthodes dites indirectes de résolution de systèmes linéaires numériques convergent plus ou moins vite, ici, point de salut : la solution d'un système linéaire à N équations et N inconnues



(a) (b)

FIG. 11.1 : Motifs de densité pour un exemple de matrice de provenant d'un problème de résolution d'équation de la chaleur (a), et pour la matrice RSA-768 (b).

requiert au bas mot N produits matrice $\times$ vecteur[1]. À l'évidence les considérations de propriétés spectrales des matrices et leur impact sur la convergence sont des notions qui n'ont pas lieu d'être dans le contexte qui nous intéresse. Il n'y a pas de distinguo fondamental entre méthodes « directes » (e.g. factorisation LU) et méthodes « indirectes » (e.g. gradient conjugué) qui soit très pertinent dans le contexte de l'algèbre linéaire sur les corps finis.

Enfin, les matrices que nous rencontrons dans le contexte de la factorisation ou du logarithme discret ont une forme caractéristique, mais qui n'est pas celle rencontrée classiquement pour la résolution d'équations aux dérivées partielles, par exemple. La figure 11.1 donne deux exemples, reproduisant des motifs de densité. L'illustration 11.1b reprend le motif de densité de la matrice apparaissant dans le calcul RSA-768 [T13]. L'illustration 11.1a est une matrice provenant d'un problème de résolution d'EDP[2].

Les dimensions des systèmes linéaires considérés sont, dans les cas intéressants, très significatives. Ainsi, la matrice RSA-768 comporte $192,796,550$ lignes et colonnes, et 27×10^9 coefficients non nuls. Cet exemple particulier provient d'un calcul de factorisation, donc à ce titre défini sur $\mathbb{F}_2$. Les exemples provenant de calculs de logarithmes discrets sont définis sur des corps plus grands, et les limites technologiques imposent des dimensions moindres.

Des dimensions semblables à celles qui viennent d'être évoquées proscrivent à l'évidence une représentation *dense* de la matrice considérée. Aussi, les matrices comme celle représentée par la figure 11.1b ne peuvent pas être mise sous forme LU, par exemple, sans rencontrer des problèmes liés à la densité des matrices produites. Il convient donc dans une telle situation d'utiliser des algorithmes « boîte noire » pour lesquels la matrice à traiter n'est jamais modifiée, et est utilisée exclusivement par le biais de l'opération « matrice $\times$ vecteur ».

[1] On verra que les algorithmes par blocs offrent la possibilité d'un compromis, en effectuant des produits un peu plus complexes matrice $\times$ bloc de vecteurs, en quantité moindre.

[2] En l'espèce, il s'agit ici d'un problème d'équation de la chaleur. Exemple récupéré de `http://www.cise.ufl.edu/research/sparse/matrices/Schmid/thermal2.html`.

Résoudre des systèmes linéaires de cette taille nécessite de mettre à contribution des ressources de calcul efficaces. Il est naturel de souhaiter mener de tels calcul en parallèle sur plusieurs machines. Un point important de cette partie est donc consacré à la parallélisation et la distribution de la résolution de systèmes linéaires creux.

11.1.2 Algorithmes utilisés

Le premier point de cette discussion est le choix de l'algorithme utilisé dans la perspective de la résolution du système. Les algorithmes en première ligne sont les algorithmes dits « par blocs », tels l'algorithme de Lanczos par blocs [158] et l'algorithme de Wiedemann par blocs [52].

Le terme d'algorithme « par blocs » se rapporte au fait que l'opération fondamentale utilisée n'est pas *exactement* le produit matrice $\times$ vecteur, mais le produit matrice $\times$ *bloc* de vecteurs. Pour une matrice $N \times N$, un bloc de vecteurs n'est autre qu'une matrice $N \times n$, pour un certain paramètre n (couramment désigné sous le nom de *blocking factor*). Par rapport aux algorithmes ne fonctionnant *pas* par blocs, tels ceux décrits par exemple dans [133], les algorithmes par blocs présentent plusieurs avantages.

- Le premier avantage des algorithmes par blocs est particulièrement apparent pour les systèmes définis sur $\mathbb{F}_2$. En effet, alors que la multiplication d'une matrice creuse par un vecteur de bits est plutôt inefficace, il est aisé, à l'aide d'opérations bit-à-bit telles le XOR, d'effectuer une multiplication d'une matrice creuse par un bloc de 64 vecteurs, en traitant 64 coefficients à la fois[3]. Ceci est particulièrement intéressant car dans le même temps, les algorithmes par blocs, lorsque des blocs de n vecteurs sont ainsi traités simultanément, permettent une réduction proportionnelle du nombre de produits matrice $\times$ vecteur qui doivent être effectués.

- Un autre bénéfice des algorithmes par blocs est lié à la probabilité de succès. En effet, les algorithmes mentionnés dans [133] sont des algorithmes probabilistes. Pour un système défini sur $\mathbb{F}_q$, leur probabilité d'échec est au minimum en $O(\frac{1}{q})$, voire en $O(\frac{N}{q})$. Lorsque q est un grand nombre premier, ceci ne pose pas de problème, tandis qu'à l'évidence pour $q = 2$ une telle probabilité d'échec est inacceptable. Les algorithmes par blocs permettent de rendre cette probabilité d'échec négligeable.

- Il est enfin utile (notamment pour le crible algébrique) de noter que les algorithmes par blocs ont pour caractéristique de fournir naturellement *plusieurs* solutions au système linéaire considéré initialement.

Nous allons voir en outre en 11.2 comment l'algorithme de Wiedemann par blocs permet une certaine distribution du calcul.

L'algorithme de Lanczos par blocs L'algorithme de Lanczos par blocs [158] est une généralisation de l'algorithme de Lanczos, qui lui-même est une adaptation de la procédure d'orthogonalisation de Gram-Schmidt pour des espaces de Krylov. L'algorithme de Lanczos par blocs « orthogonalise » non plus une suite de vecteurs mais une suite de sous-espaces vectoriels.

Nous ne détaillons pas l'algorithme de Lanczos. Ceci est fait dans [158], ainsi que dans [68], ou encore [T4]. Il nous est toutefois utile d'indiquer quelles sont les opérations centrales de cet

[3] On s'est placé ici dans le cas où les mots machines sont des mots de 64 bits.

algorithme. On suppose donnée une matrice M définie sur $\mathbb{F}_2$, et un bloc de n vecteurs de départ noté V_0. L'algorithme de Lanczos par blocs calcule les quantités suivantes, à partir de l'indice $i = 0$:

$$W_i = {}^tMMV_i,$$
$$V_{i+1} = \Phi(W_i, V_i, \ldots).$$

Intentionnellement, nous ne détaillons pas la fonction Φ. Une composante de cette fonction est en particulier l'ajout de contraintes d'orthogonalisation de la suite des espaces vectoriels engendrés par les colonnes des blocs V_i. La fonction Φ est donc non triviale. Le nombre d'itérations requises par l'algorithme de Lanczos par blocs est de $\frac{N}{n-0.76}$, où $n - 0.76$ est le rang moyen d'une matrice symétrique $n \times n$ sur $\mathbb{F}_2$. Puisque chaque itération nécessite une multiplication par M et une multiplication par sa transposée ${}^tM$, soit deux produits matrice $\times$ vecteur au total, il est légitime de dire que l'algorithme de Lanczos par blocs nécessite $O(\frac{2N}{n-0.76})$ produits matrice $\times$ vecteur.

Nous reviendrons ponctuellement sur ces caractéristiques de l'algorithme de Lanczos par blocs à différents moments dans ce chapitre.

Préconditionnement Une problématique importante de la résolution de systèmes linéaires creux est celle du préconditionnement. En effet, les algorithmes que nous étudions sont sensibles par exemple aux invariants de similitude des matrices considérées. Une matrice dont le rang est particulièrement petit par rapport à N, ou bien possédant des facteurs invariants multiples, est susceptible de provoquer des cas d'échec des algorithmes rencontrés. Pour remédier à cette difficulté, il est possible de s'intéresser non plus à la résolution du système $Mw = 0$, mais à celui (par exemple) du système $SMTw = 0$, où S et T sont des matrices de préconditionnement bien choisies. Il est intéressant de déterminer comment les matrices S et T peuvent être choisies de poids minimal afin d'offrir une probabilité de succès maximale. Différents travaux abordent ce problème, on peut par exemple citer [66, 43]. Pour les travaux auxquels nous nous sommes intéressé, le besoin de recourir à des préconditionneurs ne s'est que rarement fait sentir, aussi cet aspect n'est-il pas détaillé.

11.2 Algorithme de Wiedemann par blocs

11.2.1 Présentation

L'algorithme de Wiedemann par blocs [52] est une généralisation de l'algorithme de Wiedemann [217]. Deux valeurs de *blocking factor* m et n sont utilisées. L'algorithme de Wiedemann peut être obtenu en instanciant la description qui suit avec $m = n = 1$. Nous reviendrons sur le choix de m et n.

Soit M une matrice $N \times N$ singulière, définie sur un corps K. On cherche à déterminer un vecteur aléatoire du noyau de M, c'est-à-dire une solution w à l'équation $Mw = 0$. Dans ce qui suit, pour tout anneau A, on note $A^{p\times q}$ l'anneau (et, le cas échéant, l'algèbre) des matrices $p \times q$ à coefficients dans A. On emploie la terminologie « bloc de vecteurs » pour désigner une matrice qui, dans le cas $m = n = 1$, dégénère en un simple vecteur colonne appartenant à $K^{N\times 1} = K^N$.

Soient x et y deux blocs de vecteurs vérifiant $x \in K^{N\times m}$ et $y \in K^{N\times n}$. Posons :

$$a_i = {}^txM^{i+1}y.$$

Une façon simpliste d'obtenir un générateur Les a_i vérifient $a_i \in K^{m\times n}$. Il est aisé de constater qu'ils vérifient une relation de récurrence linéaire. En effet, si $\mu(t) = \sum_{j=0}^{\deg \mu} \mu_j X^j$ désigne le

polynôme minimal de la matrice M, on a :

$$\forall i,\ \sum_{j=0}^{\deg \mu} \mu_j a_{i+j} = {}^t x M^{i+1} \sum_{j=0}^{\deg \mu} \mu_j M^j y = 0.$$

Notons $A = \sum_{i=0}^{\infty} a_i X^i \in K[[X]]^{m \times n}$ la série formelle ayant les a_i comme coefficients. Une reformulation de l'identité précédente est l'existence d'une paire de polynômes $f(X)$, $g(X)$ vérifiant :

$$Af = g, \quad \deg f \leq \deg \mu, \quad \deg g < \deg \mu.$$

Générateurs matriciels Toutefois, en autorisant une relation de dépendance linéaire qui *combine* les colonnes des a_i, il est possible d'obtenir une relation plus courte. En effet, la relation que nous avons ainsi obtenue n'offre rien de particulièrement intéressant par rapport à la considération séparée des différents coefficients des matrices a_i.

On recherche, pour un entier r, deux matrices $f \in K[X]^{n \times r}$ et $g \in K[X]^{n \times r}$, et un entier δ, vérifiant :

$$Af = g, \quad \deg f \leq \delta, \quad \deg g < \delta.$$

Nous verrons au chapitre 12 comment le calcul de (f, g, δ) est possible, en utilisant essentiellement $\frac{N}{m} + \frac{N}{n}$ termes de la série $A(X)$, c'est-à-dire seulement $\frac{N}{m} + \frac{N}{n}$ produits matrice $\times$ vecteur à calculer[4]. Nous verrons aussi que la valeur obtenue pour δ est proche de $\frac{N}{n}$.

Une telle relation peut être utilisée pour calculer une solution potentielle au système linéaire homogène posé initialement. En effet, $Af = g$ signifie en particulier que *tous* les coefficients de degré supérieur ou égal à δ dans le produit Af sont nuls. Ceci peut se récrire en faisant apparaître une quantité privilégiée, dont le produit scalaire avec de nombreux vecteurs s'annule, comme nous le montrons maintenant. Notons pour cela $f = \sum_{i=0}^{\delta} f_i X^i$, où les matrices $f_i \in K^{n \times r}$ donnent les coefficients de f. On a, pour tout $j \geq \delta$:

$$\begin{aligned} [X^j](Af) &= 0, \\ \sum_{i=0}^{\delta} a_{j-i} f_i &= 0, \\ {}^t x M^{j-\delta} \sum_{i=0}^{\delta} M^{1+\delta-i} y f_i &= 0. \end{aligned}$$

Soit alors $w = \sum_{i=0}^{\delta} M^{\delta-i} y f_i$. On a établi :

$$\forall j \geq \delta,\ {}^t x M^{j-\delta} M w = 0.$$

Si le bloc de vecteurs x est par chance tel que l'espace engendré par les colonnes des blocs de vecteurs $\langle x, {}^t M x, \ldots \rangle$ est égal à K^N entier, ceci implique alors $Mw = 0$. Le bloc de vecteurs w étant entièrement déterminé par la valeur de My, on en déduit que w est un bloc de vecteurs appartenant au noyau de M.

L'entier r n'a pas été spécifié pour l'instant. Toute valeur de r non nulle permet d'obtenir une solution, et on verra que le choix de $r = n$ est naturellement privilégié. Nous reviendrons brièvement

[4] L'identité exploitant uniquement le polynôme minimal, qui est celle utilisée dans l'algorithme de Wiedemann simple, nécessite quant à elle d'examiner $2N$ termes de la série $A(X)$.

M	Matrice considérée. On cherche une solution du système $Mw = 0$.
N	nombre de lignes/colonnes de M.
K	corps de définition de M.
X	indéterminée de $K[X]$ et $K[[X]]$.
w	bloc de vecteurs, solutions obtenues ; $w \in K^{N\times r}$.
m, n	paramètres de l'algorithme. Pour $K = \mathbb{F}_2$, $m = 64m'$, $n = 64n'$.
x	vecteur d'intialisation (aléatoire en théorie, creux en pratique) ; $x \in K^{N\times m}$.
y	vecteur d'initialisation ; $y \in K^{N\times n}$.
${}^t x$	transposée de x.
a_i	matrice ${}^t x M^{1+i} y$; $a_i \in K^{m\times n}$.
A	série formelle $A(X) = \sum_{i\geq 0} a_i X^i$; $A \in K[[X]]^{m\times n}$
v_i	bloc de vecteurs $M^i y$; $v_i \in K^{N\times n}$.
r	entier tel que $0 < r \leq n$.
f, g	générateurs vérifiant $Af = g$; $f \in K[X]^{n\times r}$, $g \in K[X]^{m\times r}$.
δ	entier contrôlant $\deg f$ et $\deg g$ comme suit : $\deg f \leq \delta$, $\deg g < \delta$.
L	nombre de termes de la série $A(X)$ calculés dans BW1. On a $L = \frac{N}{m} + \frac{N}{n} + \epsilon$.
Cas de $K = \mathbb{F}_2$, et $n = 64n'$:	
$y^{(k)}$	k-ème sous-bloc (colonnes) de y, pour $0 \leq k < n'$; $y^{(k)} \in K^{N\times 64}$.
$v_i^{(k)}$	k-ème sous-bloc (colonnes) de $v_i = M^i y$, pour $0 \leq k < n'$; $v_i^{(k)} \in K^{N\times 64}$.
$a_i^{(k)}$	k-ème sous-bloc (colonnes) de a_i, pour $0 \leq k < n'$; $a_i^{(k)} \in K^{m\times 64}$.
$f_i^{(k)}$	k-ème sous-bloc (lignes) de f_i, pour $0 \leq k < n'$; $f_i^{(k)} \in K^{64\times r}$.
$w^{(k)}$	k-ème sommant de l'expression de w, obtenu à partir de $y^{(k)}$ et $(f_i^{(k)})_i$.

Fig. 11.2 : Algorithme de Wiedemann par blocs : notations

sur le choix de r au chapitre 12 lors de la discussion sur l'algorithme de Berlekamp-Massey matriciel permettant le calcul du générateur. Pour simplifier l'exposition, nous confondons $r = n$ dans les paragraphes qui suivent. Comme il est apparent dans le tableau 11.2 (qui distingue les rôles de r et n), le paramètre r contrôle en particulier le nombre de solutions produites par l'algorithme *in fine*.

11.2.2 Traits essentiels de l'algorithme

On peut ramener l'algorithme de Wiedemann par blocs à trois étapes essentielles qui sont les suivantes. On ne fait pas figurer dans ces étapes le choix des paramètres m et n, ainsi que des blocs de vecteurs x et y, qui appartiennent à l'intialisation de l'algorithme.

BW1 Calcul de $L = \frac{N}{m} + \frac{N}{m} + \epsilon$ termes[5] de la suite $(a_i)_i$.

BW2 Calcul du polynôme générateur $f \in K[X]^{n\times n}$.

BW3 Calcul de w à partir de f, M, et y.

Nous développons maintenant les aspects les plus essentiels de l'algorithme, qui expliquent en particulier son intérêt par rapport à l'algorithme de Lanczos par blocs esquissé en 11.1.2.

[5]La grandeur ϵ est détaillée lors de l'analyse de l'étape BW2 au chapitre 12.

Simplicité de la récurrence

Le calcul des termes a_i dans l'étape BW1 se fait de manière itérative simple, en s'appuyant exclusivement sur la boîte noire du produit matrice $\times$ vecteur. On peut à cet effet initialiser un bloc de vecteurs v à la valeur $v_0 = y$ (l'indice indiquant l'itération), et répéter les opérations $v_{i+1} = Mv_i$, puis $a_i = {}^t x v_{i+1}$. L'étape BW3 s'effectue de manière similaire ($w \mathrel{+}= v_i f_{\delta-i}$, puis $v_{i+1} = Mv_i$). Ce processus de calcul est considérablement plus simple que celui de l'algorithme de Lanczos par blocs. Outre l'économie de nombreux vecteurs auxiliaires, dont l'occupation mémoire ne peut être complètement négligée, un avantage important de la récurrence de l'algorithme de Wiedemann par blocs sur l'algorithme de Lanczos par blocs est l'absence de multiplication par la transposée ${}^t M$. En effet, cette opération, même si elle peut être effectuée au moyen des mêmes structures de données que celles utilisées pour la matrice M, peut avoir une performance bien moindre. Ceci est dû au fait que la transposition d'un algorithme échange notamment les lectures et écritures. Tandis que des lectures simultanées à partir d'un emplacement mémoire unique sont plutôt une bonne chose en terme d'optimisation, la situation où plusieurs processus concurrents aspirent à modifier une même donnée nécessite une gestion des priorités qui se ressent en terme de performance. Pour ces raisons, l'algorithme de Lanczos par blocs affronte un dilemme, entre l'économie de mémoire centrale suggérant d'utiliser une unique structure de données pour représenter M et ${}^t M$, au prix de l'efficacité, ou bien le choix de deux représentations optimisées séparées, représentant une occupation mémoire nécessairement supérieure.

Choix du paramètre n, et possibilités de distribution

En tant qu'algorithme par blocs, l'algorithme de Wiedemann par blocs offre naturellement un contexte où la boîte noire est, comme cela été décrit plus haut, un produit matrice $\times$ bloc de vecteurs. Nous illustrons notre propos en premier lieu pour le cas $K = \mathbb{F}_2$. Il est alors possible d'utiliser les opérations bit à bit des microprocesseurs, telles le XOR, pour traiter 64 bits à la fois (sur une machine 64 bits). L'implantation d'une telle boîte noire pour $n = 64$ est donc aisée. Nous discutons en 13.1 les problématiques d'implantation pour effectuer ce calcul, mais il ne nous est pas nécessaire pour l'instant d'en connaître le détail.

Une observation cruciale est liée à la situation où le paramètre n est choisi de la forme $n = 64n'$. Le bloc de vecteurs y peut alors être vu comme la concaténation de n' blocs de 64 vecteurs, que nous notons

$$y = \left(y^{(0)} \,\middle|\, \dots \,\middle|\, y^{(n'-1)} \right).$$

Il en va de même pour chacun des blocs de vecteurs v_i intervenant au cours du calcul, ainsi que de chacune des matrices a_i de taille $m \times n$. On a ainsi :

$$v_i = \left(v_i^{(0)} \,\middle|\, \dots \,\middle|\, v_i^{(n'-1)} \right),$$
$$a_i = \left(a_i^{(0)} \,\middle|\, \dots \,\middle|\, a_i^{(n'-1)} \right).$$

La remarque qui suit est primordiale.

REMARQUE. Soit k, k' tels que $0 \leq k, k' < n'$ et $k \neq k'$. Les valeurs $v_i^{(k)}$ et $a_i^{(k)}$ sont indépendantes de $y^{(k')}$.

Cette remarque est le point-clé permettant de distribuer les calculs de l'algorithme de Wiedemann par blocs sur *plusieurs clusters*. En effet, il est tout à fait possible de séparer le calcul en n'

calculs indépendants, se déroulant en des endroits différents. Comme nous venons de le remarquer, aucune communication n'est requise entre les n' tâches, consistant respectivement à calculer les suites $(a_i^{(0)})_i, \ldots, (a_i^{(n'-1)})_i$ dans l'étape BW1. La même remarque vaut pour l'étape BW3, avec une modification technique mineure. Le k-ème « sous-bloc » de vecteur $y^{(k)}$, de largeur 64, est combiné avec 64 *lignes* de la matrice f (ou, de manière équivalente, des matrices scalaires f_i pour chaque indice i considéré). Aussi, nous adoptons la notation $f_i^{(k)}$ pour désigner non pas un bloc de 64 colonnes mais un bloc de 64 lignes de la matrice f_i. La matrice $f_i^{(k)}$ appartient donc à $K^{64\times n}$. Ainsi, w s'écrit :

$$
\begin{aligned}
w &= \sum_{i=0}^{\delta} M^i y f_{\delta-i}, \\
w &= \sum_{k=0}^{n'-1} \sum_{i=0}^{\delta} M^i y^{(k)} f_{\delta-i}^{(k)}, \\
w &= \sum_{k=0}^{n'-1} w^{(k)}, \text{ avec } w^{(k)} = \sum_{i=0}^{\delta} M^i y^{(k)} f_{\delta-i}^{(k)}.
\end{aligned}
$$

L'algorithme de Wiedemann par blocs offre donc une possibilité de *distribution* des calculs. Nous entendons ce terme comme complément de la *parallélisation*, comme suit :

- La *parallélisation* des calculs dans l'algorithme de Wiedemann par blocs désigne le travail qui peut être fait, et qui est mentionné en 13.2, pour répartir le calcul sur plusieurs processeurs d'une machine, ou plusieurs nœuds d'un cluster. Implicitement, la parallélisation induit une forte dépendance vis-à-vis des capacités de communication des entités considérées (cœurs d'un microprocesseur et/ou nœuds d'un cluster). Paralléliser ainsi un produit matrice $\times$ vecteur est une problématique courante, et en tous cas qui existe de manière tout à fait semblable dans l'algorithme de Lanczos par blocs[6].

- La *distribution* indique la possibilité de répartir les calculs *sans* reposer sur des capacités de communication efficace. Les calculs qu'on distribue ainsi ont donc vocation à être exécutés de façon indépendante, sans communication ou synchronisation en dehors du début et de la fin du calcul. L'algorithme de Wiedemann par blocs offre ainsi une capacité de distribution des calculs des étapes BW1 et BW3.

Nous avons ainsi illustré une différence importante entre l'algorithme de Wiedemann par blocs et l'algorithme de Lanczos par blocs. Bien que l'algorithme de Wiedemann par blocs nécessite un plus grand nombre d'itérations que l'algorithme de Lanczos par blocs lorsque la valeur de n est constante (par exemple fixée à 64), on obtient un bénéfice en fixant une valeur plus grande de la forme $n = 64n'$, puisqu'alors le travail peut être réalisé par n' ressources de calcul de taille moyenne (en comparaison avec une ressource de calcul qui permettrait d'utiliser l'algorithme de Lanczos par blocs avec une valeur de n semblable).

Cas d'autres corps de définition

Nous avons décrit comment il était possible de traiter 64 bits simultanément en mode « SIMD » (pour *single-instruction, multiple-data*), à l'aide des opérations d'arithmétique bit-à-bit des microprocesseurs. De la sorte, notre description s'adapte bien au contexte $K = \mathbb{F}_2$. Il convient de remarquer

[6]Nous verrons en 13.2 que certains travaux du domaine « numérique » sur ce sujet ne peuvent toutefois pas s'appliquer à notre contexte de calculs exacts.

que l'organisation des calculs telle que nous la proposons dans les paragraphes précédents peut être adaptée à d'autres contextes, y compris à celui où 64 est remplacé simplement par 1. En effet, lorsque K est un grand corps premier, comme c'est le cas pour le calcul de logarithmes discrets [T2], il peut être pertinent de choisir pour n une petite valeur entière, pour permettre une distribution partielle des calculs (dans [T2], nous avons utilisé $m = n = 4$). Un cas intermédiaire peut être donné par $\mathbb{F}_p$, où p est un *petit* nombre premier. Nous avons à l'esprit des valeurs telles $p = 3$, $p = 17$, $p = 65521$. Dans ces cas, il peut s'avérer payant d'utiliser les instructions SIMD des microprocesseurs modernes. Le contexte d'implantation offre ainsi un contexte naturel dans lequel plusieurs données sont traitées simultanément. Les considérations que nous venons de développer s'étendent *mutatis mutandis*, la constante 64 étant essentiellement la seule à devoir être adaptée. Nous avons effectivement réalisé et utilisé de telles adaptations dans le contexte des travaux sur les problèmes RSA et SDH avec oracle, décrits au chapitre 5.

Chapitre 12

Algorithme de Berlekamp-Massey matriciel

Le premier élément de notre contribution à l'algorithme de Wiedemann par blocs a été la publication d'un algorithme quasi-linéaire [T1, T3] pour le calcul du générateur (f, g, δ) tel que défini en 11.2.1. Cet algorithme s'est avéré essentiel pour la compétitivité de l'algorithme de Wiedemann par blocs dans le contexte de la factorisation d'entiers, puisqu'il a permis d'asseoir son usage prépondérant sur les records de factorisation des quelques dernières années. Nous présentons ici rapidement les points-clés de cet algorithme quasi-linéaire, sans dupliquer les détails publiés notamment dans [T1, 209, T3]. Notons que cet algorithme est désormais ancien (la première publication date de 2001). Nous le décrivons essentiellement afin de pouvoir énoncer en 12.3 des résultats plus récents relatifs à l'utilisation de séquences déséquilibrées. Ces derniers résultats ont été décrits sous une forme préliminaire un peu plus complexe dans [T13].

12.1 Présentation du problème

La donnée initiale du problème est un nombre L de termes d'une série formelle $A \in K[[X]]^{m\times n}$ définie sur un corps K. Le fait que A soit obtenue par un processus semblable à celui décrit en 11.2.1 est sans importance pour le problème du calcul de générateur. On recherche une paire de matrices de polynômes $f \in K[X]^{n\times r}$, $g \in K[X]^{m\times r}$ et un entier δ vérifiant :

$$Af = g, \quad \deg f \leq \delta, \quad \deg g < \delta. \tag{12.1}$$

Comme cela a été déjà discuté, nous allons laisser de côté le choix de l'entier r pour l'instant pour y revenir brièvement ultérieurement. Aussi les paragraphes qui suivent sont-ils rédigés dans le cas $r = n$, et les notations conservées sont donc par exemple $f \in K[X]^{n\times n}$.

Le calcul d'un tel générateur linéaire « matriciel » peut être vu, et c'est le point de vue que nous adopterons, comme une généralisation de l'algorithme de Berlekamp-Massey [22, 145], puisque ce dernier résout précisément le problème que nous avons énoncé dans le cas $m = n = 1$. Il est aussi possible de voir ce problème comme une forme d'algorithme d'Euclide, ou comme un calcul d'approximants de Padé. L'algorithme de Beckermann et Labahn [18] fournit une solution à ce problème en temps quasi-linéaire. L'algorithme que nous avons proposé dans [T3] est très semblable dans sa structure, mais sensiblement meilleur. Les complexités respectives des deux algorithmes sont $O((m+n)^2 m((m+n)^{\omega-2} + \log L)L \log L)$ pour l'algorithme de Beckermann-Labahn, contre $O((m+n)^2((m+n)^{\omega-2}+\log L)L \log L)$ pour l'algorithme que nous proposons, où ω est l'exposant de la multiplication de matrice[1] (nous revenons sur cette comparaison en 12.2.3).

[1] Les matrices multipliées sont de taille petite. Toutefois il est utile de savoir sur quelle partie de la complexité une intervention de l'algorithme de Strassen permet d'obtenir un gain, puisque dans la pratique un ou deux niveaux de récursion sont tout à fait pertinents pour les exemples considérés.

La présentation de l'algorithme de calcul de générateur repose sur le lemme suivant. L'hypothèse de ce lemme est une hypothèse de généricité que nous faisons sur la série formelle apparaissant dans le calcul de générateur de l'étape BW2.

LEMME 12.2. *Soit $A \in K[[X]]^{m\times n}$. On suppose que A dispose d'une description en fraction rationnelle à gauche, c'est-à-dire qu'il existe deux matrices $N \in K[X]^{m\times n}$ et $D \in K[X]^{m\times m}$, avec D unimodulaire (inversible dans $K[[X]]^{m\times m}$, c'est-à-dire telle que $\det D$ est de valuation nulle), telles que $A = D^{-1}N$. Soit maintenant r un entier, trois matrices $f \in K[X]^{n\times r}$, $g \in K[X]^{m\times n}$, $e \in K[[X]]^{m\times r}$, et un entier t vérifiant : $Af = g + X^t e$. On a alors :*

$$t - \max(\deg f, 1 + \deg g) \geq \max(\deg D, 1 + \deg N) \Rightarrow e = 0.$$

DÉMONSTRATION. Il suffit de multiplier le produit Af par la matrice D à gauche. On obtient alors :

$$DAf = Nf = Dg + X^t De.$$

En raisonnant sur les degrés, on obtient (on pose $d = \max(\deg D, 1 + \deg N)$) :

$$\deg N < d, \deg f \leq t - d \Rightarrow \deg Nf < t,$$
$$\deg D \leq d, \deg g < t - d \Rightarrow \deg Dg < t,$$

Il s'ensuit que le produit De est inévitablement nul. Comme la matrice D est inversible, cela implique la nullité de e. ■

12.1.1 Principe de fonctionnement et invariants de la récurrence

Le principe de l'algorithme de calcul de générateur est de maintenir une identité du type

$$AF = G + X^t E$$

à différentes étapes de l'algorithme, indicées par l'entier t. Les objets E,F,G qui apparaissent satisfont :

$$F \in K[X]^{n\times(m+n)}, \quad G \in K[X]^{m\times(m+n)}, \quad E \in K[X]^{m\times(m+n)}.$$

Nous maintenons en outre un vecteur d'entiers $\delta = (\delta_1, \ldots, \delta_{m+n})$ contrôlant les degrés des *colonnes* des matrices F et G comme suit :

$$\max(\deg F_{\cdot,j}, 1 + \deg G_{\cdot,j}) \leq \delta_j.$$

Enfin, nous imposons la contrainte suivante pour chaque valeur de t :

$$\mathrm{rg}([X^0]E) = m.$$

L'algorithme considère donc des matrices E, F, G dont le nombre de colonnes est supérieur au nombre de générateurs voulus. L'objectif est *in fine* d'extraire de ces matrices un sous-ensemble de r colonnes vérifiant les conditions du lemme 12.2. Étant donné la définition de δ_j, cela revient à obtenir un ensemble de r indices j tels que $t - \delta_j$ dépasse le seuil $\max(\deg D, 1 + \deg N)$. Notons que le couple (N, D) est inconnu. Cela n'empêche toutefois pas de travailler sur la base d'une hypothèse heuristique sur le degré correspondant. Le point essentiel est surtout la capacité à garantir que l'évolution de l'algorithme fait *croître* l'écart $t - \delta_j$. C'est heureusement le cas.

12.1.2 Initialisation

L'initialisation de la récurrence est détaillée dans [T3]. Il est suffisant de retenir qu'on choisit $t_0 \geq \lceil \frac{m}{n} \rceil$, et que cette initialisation peut être faite de manière déterministe sous une hypothèse contrôlable. Nous répétons toutefois brièvement le principe de l'initialisation tel qu'évoqué dans [T3], car il nous sera utile d'y revenir lors de la considération du cas des séquences déséquilibrées en 12.3. On fait l'hypothèse que la famille des colonnes des t_0 premières matrices $a_i = [X^i]A$ est de rang plein. Ceci impose naturellement $t_0 \geq \frac{m}{n}$, et heuristiquement il est raisonnable de supposer que cette hypothèse peut être satisfaite pour une valeur de t_0 de l'ordre de $\frac{m}{n} + \epsilon$. En tout état de cause, vérifier que cette hypothèse peut être satisfaite est aisé en considérant uniquement un petit nombre de termes a_i. Sous cette hypothèse, on construit les n premières colonnes de la matrice $F^{(t_0)} \in K[X]^{n\times(m+n)}$ comme la matrice identité I_n. Les m dernières colonnes sont choisies sous la forme de monômes en X qui « extraient » chacun exactement l'une des colonnes des matrices $a_0, \ldots, a_{t_0-1}$. De la sorte, la matrice $[X^{t_0}](AF^{(t_0)})$ est de rang plein (en fait cette propriété est satisfaite par ses m dernières colonnes).

12.1.3 Pas élémentaires de l'algorithme

Pour passer de l'étape t à l'étape $t+1$, il faut considérer la matrice $[X^0]E^{(t)}$, où le suffixe $^{(t)}$ précise qu'on s'intéresse à la valeur de E *à l'étape* t. C'est, par hypothèse, une matrice de rang m. Il est aisé de déterminer une permutation $\sigma \in \mathfrak{S}_{m+n}$ telle que les entiers $\delta_{\sigma(1)}, \ldots, \delta_{\sigma(m+n)}$ sont triés par ordre croissant. Notons alors Σ la matrice de permutation telle que $\Sigma_{\sigma(j),j} = 1$ pour tout j. La j-ème colonne de $F\Sigma$ est la $\sigma(j)$-ème colonne de F. Ainsi, les colonnes des matrices $F\Sigma$, $G\Sigma$, $E\Sigma$ sont triées de sorte que les entiers associés sont donnés par le vecteur d'entiers noté $\delta\Sigma = (\delta_{\sigma(1)}, \ldots, \delta_{\sigma(m+n)})$, trié par ordre croissant.

Il est possible d'effectuer une réduction des colonnes de la matrice $[X^0]E^{(t)}\Sigma$. Ainsi, il existe une matrice inversible $R \in K^{(m+n)\times(m+n)}$ telle que :

- $([X^0]E^{(t)}\Sigma)R$ a exactement m colonnes non nulles ;
- $i > j \Rightarrow R_{i,j} = 0$ (la matrice R est triangulaire supérieure).

La deuxième condition impose que l'élimination se fasse sans induire une modification des valeurs δ_j. En effet, on vérifie que :

$$\begin{aligned}
\max(\deg(F^{(t)}\Sigma R)_{\cdot,j}, 1 + \deg(G^{(t)}\Sigma R)_{\cdot,j}) &\leq \max(\{\deg(F_{\sigma(i)}R_{i,j}), i \leq j\} \cup \\
&\qquad \{1 + \deg(G_{\sigma(j)}R_{i,j}), i \leq j\}), \\
&\leq \max(\{\deg F_{\sigma(i)}, i \leq j\} \cup \{1 + \deg G_{\sigma(i)}, i \leq j\}), \\
&\leq \max\{\delta_{\sigma(i)}, i \leq j\}, \\
&\leq \delta_{\sigma(j)},
\end{aligned}$$

la dernière inégalité découlant du tri des colonnes par ordre de $\delta_{\sigma(i)}$ croissant.

Sans changer les δ_i, on fait donc apparaître n colonnes nulles dans la matrice $([X^0]E^{(t)}\Sigma)R$. On introduit alors une matrice diagonale Δ, dont l'entrée $\Delta_{i,i}$ vaut 1 si la i-ème colonne de $([X^0]E^{(t)}\Sigma)R$ est nulle, et vaut X dans les m autres cas. Ainsi, on garantit

$$\begin{aligned}
[X^0](E^{(t)}\Sigma R\Delta) &= 0, \\
\mathrm{rg}([X^1](E^{(t)}\Sigma R\Delta)) &= \mathrm{rg}([X^0](E^{(t)}\Sigma R)) = m.
\end{aligned}$$

Posons alors :

$$\pi^{(t,t+1)} = \Sigma R \Delta,$$
$$F^{(t+1)} = F^{(t)}\pi^{(t,t+1)}, \quad G^{(t+1)} = G^{(t)}\pi^{(t,t+1)}, \quad E^{(t+1)} = E^{(t)}\pi^{(t,t+1)} \operatorname{div} X,$$
$$\delta^{(t+1)} = (\delta^{(t)}_{\sigma(j)} + \deg \Delta_{j,j}).$$

On vérifie alors que les invariants de la récurrence sont vérifiés à l'étape $t+1$. En outre, la valeur moyenne $\overline{\delta}$ des entiers δ_j croît de seulement $\frac{m}{m+n}$. On a donc l'assurance que l'écart $t - \delta_j$ croît en moyenne, ce qui suffit à garantir que le lemme 12.2 peut être appliqué pour t assez grand.

Dans la pratique, notre hypothèse de généricité pour l'application du lemme 12.2 est qu'il existe une description de A en fraction rationnelle à gauche avec $\max(\deg D, 1 + \deg N) = \lambda$, et que le nombre L de termes de la série A connus vérifie $L \geq t_0 + \frac{m+n}{n}\lambda$. Ainsi, pour $t - t_0 \geq \lambda$, on a $t - \overline{\delta} \geq \lambda$, ce qui permet d'appliquer le lemme 12.2. Dans le cas de l'algorithme de Wiedemann par blocs où la série A est obtenue par l'étape BW1 de l'algorithme, telle que décrite en 11.2.1, pour une matrice M de taille $N \times N$, nous supposons $\lambda = \lceil \frac{N}{m} \rceil + \eta$, avec η petit. Ainsi, le nombre de termes à calculer dans l'étape BW1 est donné par $L = t_0 + \frac{m+n}{n}\lambda = \frac{N}{m} + \frac{N}{n} + \epsilon$, avec ϵ au minimum supérieur à $\lceil \frac{m}{n} \rceil + \frac{m+n}{n}\eta$. On choisit en pratique une valeur arbitraire, petite par rapport à $\frac{N}{m} + \frac{N}{n}$, mais suffisante pour se prémunir d'éventuelles difficultés liées à un défaut partiel de généricité.

Bien que la supposition $\lambda = \lceil \frac{N}{m} \rceil + \eta$ soit valide pour des systèmes génériques, nous convenons que notre approche a ses limites. Les programmes que nous utilisons pour résoudre à l'aide de l'algorithme de Wiedemann par blocs des systèmes linéaires issus, par exemple, de problèmes de factorisation, ne peuvent être utilisés directement pour des systèmes plus particuliers. Par exemple le cas $\operatorname{rg}(M) \ll N$ peut donner lieu à des difficultés, qui peuvent nécessiter l'emploi de préconditionneurs comme évoqué plus haut.

12.2 Structure récursive

12.2.1 Algorithme

Les matrices $\pi^{(t,t+1)}$ introduites précédemment sont de degré 1. Les pas élémentaires de l'algorithme que nous venons de décrire induisent des multiplications entre des objets de degré significatif (tels les matrices $F^{(t)}$), et des objets de petit degré (les matrices $\pi^{(t,t+1)}$). Afin d'obtenir un algorithme sous-linéaire, on s'appuie sur la remarque suivante.

Remarque. Les matrices $\pi^{(t,t+1)}, \ldots, \pi^{(t+k-1,t+k)}$ sont entièrement déterminées par les k premiers coefficients de la matrice $E^{(t)}$, et par le vecteur d'entiers $\delta^{(t)}$.

Définissons la matrice $\pi^{(t,t+k)} = \pi^{(t,t+1)} \ldots \pi^{(t+k-1,t+k)}$. Grâce à l'observation qui précède, nous sommes en mesure de mettre en place une procédure récursive de calcul de $\pi^{(t,t+k)}$ à partir des k premiers coefficients de $E^{(t)}$, et du vecteurs d'entiers $\pi^{(t)}$. Un schéma de fonctionnement est le suivant :

- Soit $j = \lfloor \frac{k}{2} \rfloor$.
- Calculer $\pi^{(t,t+j)}$ à partir de $\delta^{(t)}$ et de $E^{(t)} \bmod X^j$.
- Calculer $E^{(t+j)} \bmod X^{k-j}$ à partir de $E^{(t)}$ et de $\pi^{(t,t+j)}$.
- Calculer $\delta^{(t+j)}$ à partir de $\delta^{(t)}$ et de $\pi^{(t,t+j)}$.

Algorithme $\mathsf{MSLGDC}(k, E, \delta)$

ENTRÉE : k entier, et $E^{(t)} \in K[X]^{m\times(m+n)}$ tel que $\deg E^{(t)} < k$.
δ vecteur de $m+n$ entiers.
SORTIE : $\pi^{(t,t+k)} \in K[X]^{(m+n)\times(m+n)}$.

{
`if (`$k <=$ `threshold) return` $\pi^{(t,t+k)}$; *(procédure itérative)*
$j = \lfloor k/2 \rfloor$
$(E_L, \delta_L) = \left(E^{(t)} \bmod X^j,\ \delta\right)$;
$\pi_L = \mathsf{MSLGDC}(j, E_L, \delta_L)$;
$(E_R, \delta_R) = \left(\left(E\pi_L \bmod X^k\right) \operatorname{div} X^j,\ \delta\pi_L\right)$;
$\pi_R = \mathsf{MSLGDC}(k-j, E_R, \delta_R)$;
`return` $\pi = \pi_L \times \pi_R$;
}

ALGORITHME 12.3 : Pseudo-code d'algorithme récursif pour le calcul de $\pi^{(t,t+k)}$.

- Calculer $\pi^{(t+j,t+k)}$ à partir de $\delta^{(t+j)}$ et $E^{(t+j)} \bmod X^{k-j}$.
- Retourner le produit $\pi^{(t,t+j)} \times \pi^{(t+j,t+k)}$.

Cette trame est reprise par le pseudo-code[2] donné par l'algorithme 12.3. Une matrice $\pi^{(t,t+k)}$ a un degré moyen égal à $\frac{m}{m+n}k$ (seules les matrices Δ intervenant dans sa construction sont de degré supérieur à zéro). Ainsi, on voit que la procédure récursive permet de mener le calcul en mettant une pression plus forte sur des produits de polynômes de plus grand degré. L'emploi d'algorithmes asymptotiquement rapides est donc possible.

Pour préciser l'algorithme 12.3 en une version asymptotiquement rapide, le remplacement de l'opération « produit » par une opération « produit rapide » n'est qu'une partie du chemin, il est possible d'aller plus loin. En effet, dans le contexte d'algorithmes asymptotiquement rapides comme les algorithmes utilisant des transformées de Fourier (voir aussi l'algorithme de Cantor évoqué en 16.4.3), ou plus généralement tout schéma s'appuyant sur les trois étapes d'évaluation, convolution, et interpolation, il est enrichissant de séparer ces trois étapes lorsque les mêmes données sont utilisées pour plusieurs produits, comme c'est le cas pour des produits de matrices. Supposons données trois procédures $\mathsf{transform}_\ell$, conv_ℓ, et $\mathsf{itransform}_\ell$, telles que pour deux polynômes A, B à coefficients dans K dont le produit $C = AB$ vérifie $\deg C \leq \ell$, on ait :

$$C = \mathsf{itransform}_\ell(\mathsf{conv}_\ell(\mathsf{transform}_\ell(A), \mathsf{transform}_\ell(B))).$$

En d'autres termes, ce schéma permet de calculer C par les étapes suivantes :

$$\widehat{A} = \mathsf{transform}_\ell(A),$$
$$\widehat{B} = \mathsf{transform}_\ell(A),$$
$$\widehat{C} = \mathsf{conv}_\ell(\widehat{A}, \widehat{B}),$$

[2]La convention d'écriture $\delta\pi_L$ abrège l'évidence. Pour chaque matrice intervenant dans le calcul de π, l'opération à droite sur le vecteur d'entiers δ est triviale à définir. Le produit $\delta\pi_L$ est donc défini en conséquence. En outre, cette étape de recalcul de δ peut être omise, puisque $\delta\pi_L$ coïncide avec la valeur δ atteinte dans l'appel récursif. Les valeurs δ et t peuvent être considérées comme globales.

Algorithme $\mathsf{MSLGDC\_FFT}(k, E, \delta)$

ENTRÉE : k entier, et $E^{(t)} \in K[X]^{m\times(m+n)}$ tel que $\deg E^{(t)} < k$.
δ vecteur de $m+n$ entiers.
SORTIE : $\pi^{(t,t+k)} \in K[X]^{(m+n)\times(m+n)}$.

{
`if` $(k <= \mathsf{threshold})$ `return` $\pi^{(t,t+k)}$; *(procédure itérative)*
`if` $(k <= \mathsf{fft\_threshold})$ `return` $\mathsf{MSLGDC}(k, E, \delta)$; *(sans FFT)*
$j = \lfloor k/2 \rfloor$
$(E_L, \delta_L) = \left(E^{(t)} \bmod X^j,\ \delta\right)$;
$\pi_L = \mathsf{MSLGDC\_FFT}(j, E_L, \delta_L)$;
$\ell =$ *longueur de transformée* $\geq k + \deg \pi_L \approx k\left(1 + \frac{1}{2}\frac{m}{m+n}\right)$;
$\widehat{E} = \mathsf{transform}_\ell(E)$;
$\widehat{\pi_L} = \mathsf{transform}_\ell(\pi_L)$;
$\widehat{E\pi_L} = \mathsf{conv}_\ell(\widehat{E}, \widehat{\pi_L})$;
$E_R = \mathsf{itransform}_\ell(\widehat{E\pi_L}) \bmod X^k \operatorname{div} X^j$;
$\delta_R = \delta\pi_L$;
$\pi_R = \mathsf{MSLGDC\_FFT}(k-j, E_R, \delta_R)$;
$\widehat{\pi_R} = \mathsf{transform}_\ell(\pi_R)$;
$\widehat{\pi} = \mathsf{conv}_\ell(\widehat{\pi_L}, \widehat{\pi_R})$;
`return` $\pi = \mathsf{itransform}_\ell(\widehat{\pi})$;
}

ALGORITHME 12.4 : Pseudo-code d'algorithme récursif pour le calcul de $\pi^{(t,t+k)}$ (version rapide).

$$C = \mathsf{itransform}_\ell(\widehat{C}).$$

Les algorithmes de multiplication de complexité quasi-linéaire, en fonction du corps de base K, offrent une complexité en $O(\ell \log \ell)$ opérations dans K lorsque K possède des racines ℓ-èmes de l'unité, ou plus généralement une complexité binaire en $O(\ell \log^2 \ell \log\log \ell)$ en utilisant la substitution de Kronecker et l'algorithme de Schönhage-Strassen. La complexité du produit est alors portée par les opérations transform et itransform, tandis que l'opération conv est linéaire. Pour multiplier, comme c'est fait dans l'algorithme 12.3, non plus des polynômes mais des *matrices* à coefficients dans $K[X]$, il suffit d'étendre les fonctions $\mathsf{transform}_\ell$, conv_ℓ, et $\mathsf{itransform}_\ell$. La transformée et la transformée inverse peuvent s'effectuer terme par terme, tandis que le produit de convolution s'exprime comme $O(n^\omega)$ produits de convolution de polynômes, n étant la dimension des matrices considérées, et ω l'exposant de l'algorithme utilisé pour le produit. L'algorithme 12.4 exploite cette idée.

On note que dans l'algorithme 12.4, la longueur de transformée employée peut être améliorée. En effet, il est possible d'utiliser un produit médian [99, 101] pour le calcul de E_R, ce qui permet un gain de complexité.

12.2.2 Complexité

La complexité du calcul s'exprime de façon différente selon le corps de base K et l'algorithme de multiplication qui est employé. Nous donnons la complexité pour calculer un générateur linéaire pour une série de matrices $A \in K[[X]]^{m\times n}$ donnée par ses L premiers termes. L'unité choisie est le nombre d'opérations algébriques sur K, et nous faisons l'hypothèse que le corps K permet une

complexité en $\ell \log \ell$ pour les multiplications de polynômes, au moins pour les longueurs vérifiant $\ell \leq 2L$. Nous supposons enfin que les dimensions m et n vérifient $m, n \in O(\log L)$.

Il est évident que le calcul des L premiers termes de $E^{(t_0)}$ à partir de A se fait en temps linéaire, puisque la matrice $F^{(t_0)}$ est de degré qu'on considère constant (l'initialisation est détaillée dans [209]). Dès lors, la complexité de l'algorithme MSLGCD_FFT s'exprime ainsi :

$$O((m+n)^2((m+n)^{\omega-2} + \log L) L \log L).$$

Cette complexité est meilleure que la complexité en $O((m+n)^\omega L \log^2 L)$ qu'on obtiendrait en effectuant simplement les produits de l'algorithme 12.3 avec un produit rapide.

12.2.3 Comparaison avec l'algorithme de Beckermann-Labahn

L'algorithme SPHPS de Beckermann-Labahn [18] est très similaire à l'algorithme développé ici. Le formalisme développé dans [18] permet de résoudre un ensemble de problèmes généralisant le concept d'approximants de Padé. Au nombre de ceux-ci, on trouve notre problème exposé par l'équation 12.1 page 129. Cette réduction est mentionnée brièvement dans [18], et aussi dans [170] (avec quelques coquilles). À l'instar de ce qui est développé dans ce chapitre, on cherche à résoudre une équation à précision fixée, c'est-à-dire une identité de la forme suivante, où $A \in K[X]^{m \times n}$:

$$Af = g \bmod X^t, \quad f \in K[X]^{n \times 1}, \quad g \in K[X]^{m \times 1}$$

avec en outre la condition sur les degrés $\deg f \leq \delta$, et $\deg g < \delta$. À la différence de l'algorithme que nous avons présenté, les degrés δ et t doivent être donnés comme paramètres d'entrée pour l'algorithme de Beckermann-Labahn. On utilise les paramètres qui sont ceux de la sortie obtenue pour l'algorithme de ce chapitre, à savoir $\delta = \lceil N/n \rceil + \epsilon$ et $t = L = \lceil N/m \rceil + \lceil N/n \rceil + \epsilon \approx \frac{m+n}{m}\delta$. Ceci garantit (en comparant le nombre d'équations et d'inconnues sur K) l'existence de solutions.

Notons f_i (respectivement g_i) la i-ème coordonnée du vecteur colonne f (respectivement de g). Notons $A_{\cdot,j}$ la j-ème colonne de $A \in K[[X]]^{m \times n}$, et en outre e_i le i-ème vecteur de la base canonique de $K[X]^{m \times 1}$. On récrit l'équation ci-dessus comme

$$A_{\cdot,1} f_1 + \cdots + A_{\cdot,n} f_n - (e_1 g_1 + \cdots + e_m g_m) = 0 \bmod X^t,$$

où les f_i et les g_i sont des polynômes. On est donc en présence d'un problème d'approximants de Padé multiples, et vectoriels (on rappelle que les $A_{\cdot,j}$ sont des colonnes de A). L'approche de Beckermann-Labahn passe par l'élimination de l'aspect « vectoriel », en remplaçant les colonnes de A par les coordonnées du vecteur ligne $(1, X, \ldots, X^{m-1}) \times A(X^m)$. En composant l'équation ci-dessus par X^m et en multipliant par le vecteur idoine à gauche, on obtient :

$$\begin{aligned}
\mathbf{F}(X)\mathbf{P}(X^m) &= 0 \bmod X^{mt}, \\
\mathbf{F}_j &= (1, X, \ldots, X^{m-1}) \times A_{\cdot,j}(X^m) \text{ pour } 1 \leq j \leq n, \\
\mathbf{F}_{n+j} &= X^{j-1} \text{ pour } 1 \leq j \leq m, \\
\mathbf{P} &= (f_1, \ldots, f_n, g_1, \ldots, g_m), \\
(\deg \mathbf{P}_i)_i &\leq (\delta, \ldots, \delta, \delta - 1, \ldots, \delta - 1).
\end{aligned}$$

Ce système est semblable au type de système résolu par l'algorithme de Beckermann-Labahn [18, definition 1.1], où les indices (s, m, σ) de [18] sont ici nos indices $(m, m+n, mt)$. La complexité de l'algorithme de Beckermann-Labahn, en reprenant l'analyse de [18, p. 821], est en $O((m+s)m\sigma(m^{\omega-2}+$

$\log\sigma)\log\sigma)$ (la partie $m^{\omega-2}$ est omise dans [18]). En substituant les paramètres qui sont ceux développés dans ce chapitre, la complexité est en $O((m+n)^3((m+n)^{\omega-2}+\log L)\log L$, soit un facteur multiplicatif $(m+n)$ par rapport à l'algorithme développé dans ce chapitre (l'explication tenant à la longueur de transformée utilisée).

12.2.4 Paramètre r de l'algorithme de Wiedemann par blocs

L'algorithme de Wiedemann par blocs exploite un générateur linéaire de taille $n \times r$ de la série A fournie en entrée de l'étape BW2. On a vu que le calcul de générateur linéaire permettait de fournir un générateur jusqu'à une taille $n \times n$. La complexité des étapes élémentaires de l'algorithme de Berlekamp-Massey matriciel n'est pas modifiée par un changement de la valeur de r. Toutefois, le nombre de coefficients nécessaires peut l'être. En effet, un r-uplet de générateurs est obtenu dès le moment où r colonnes des matrices F, G vérifient $t - \max(\deg F_{\cdot,j}, 1 + \deg F_{\cdot,j}) \geq \lambda$, où λ est le degré de la description en fraction rationnelle à gauche, que nous ne connaissons pas. L'écart $t - \max(\deg F_{\cdot,j}, 1+\deg F_{\cdot,j})$ croît de manière uniforme tant qu'aucun générateur n'est obtenu. Dès lors qu'une colonne se fixe sur un générateur, elle n'est plus modifiée, et les écarts $t - \max(\deg F_{\cdot,j}, 1 + \deg F_{\cdot,j})$ ne croissent plus de manière uniforme : l'écart correspondant aux colonnes fixées croît de 1 à chaque étape, ce qui réduit la croissance moyenne pour les autres colonnes. Aussi, considérer $\frac{m+n}{n}\lambda$ coefficients de A n'est valable que dans le cas favorable où il n'existe pas de générateur de degré exceptionnellement bas.

12.3 Séquences déséquilibrées

Nous avons vu comment se construisait la série $A(X)$ en entrée de l'étape BW2 de l'algorithme de Wiedemann par blocs. Nous anticipons quelque peu sur la section 13.3, en considérant la mise en pratique de l'algorithme de Wiedemann par blocs sur plusieurs clusters distincts, le calcul étant séparé en n' calculs indépendants, comme mentionné en 11.2.2. Il est tout à fait naturel, dans un tel cas, d'avoir certains clusters plus rapides que d'autres. Pour que tous atteignent l'objectif du nombre voulu de termes à calculer de manière synchrone, il est possible d'échanger les séquences traitées à un moment bien choisi du calcul. Par exemple, si deux clusters A et B partagent le calcul en deux sous-séquences, et que le cluster A va deux fois plus vite que le cluster B, le cluster A traite $2k/3$ itérations de la première séquence en un temps $T/2$, puis $2k/3$ de la seconde en un temps $T/2$. Cependant, le cluster B aura traité $k/3$ itérations de la seconde séquence pendant la première période de durée $T/2$, et $k/3$ de la première ensuite. Le nombre total d'itérations k calculées à l'issue du temps T pour les deux séquences est ainsi équilibré. Un tel équilibrage est évidemment utile, mais il n'est pas toujours facile de « viser juste » (à plus forte raison lorsque l'efficacité des ressources utilisés varie dans le temps, comme on le verra en 13.3). On s'intéresse ici à la façon dont il est possible de calculer un générateur linéaire dès qu'on a non pas calculé L termes des suites $(a_i^{(j)})_i$ pour chaque j, mais qu'on a réalisé l'objectif légèrement moins précis d'avoir calculé ce nombre de termes *en moyenne*. On s'attend en effet à atteindre cet objectif moyen plus tôt. À l'échelle d'un calcul de grande ampleur avec l'algorithme de Wiedemann par blocs, et en fonction de l'hétérogénéité des ressources de calcul utilisées, le temps ainsi gagné peut être significatif, par exemple de l'ordre de quelques jours pour le calcul RSA-768.

L'objectif de pouvoir traiter une série $A(X)$ dont le nombre de termes connus est déséquilibré en fonction des colonnes répond à un souci pratique tout à fait mineur. Le prix de quelques échanges de séquences est modeste, et ne pose que de minces problèmes d'organisation (liés par exemple aux

fuseaux horaires). Il n'y a aucune nécessité impérieuse d'être en mesure d'exploiter cette situation. Toutefois, il est intéressant d'être capable d'y répondre positivement. Cette section est donc motivée très modérément par l'intérêt pratique, et significativement plus par la curiosité.

Dans l'article [T13], nous détaillons dans un appendice une façon de traiter le cas des séquences déséquilibrées. Cette méthode et son analyse sont dues à Kleinjung. Nous en donnons ici une présentation simplifiée.

On se donne une matrice $A(X) \in K[[X]]$, dont le nombre de termes connus n'est pas le même pour toutes les colonnes. Notons ℓ_j le nombre de termes connus pour la colonne j, où j vérifie $0 \leq j < n$. On peut supposer sans perte de généralité que $0 \leq \ell_1 \leq \ldots \leq \ell_n$. Nous faisons l'hypothèse que la moyenne des ℓ_j est supérieure ou égale à L, soit : $\sum_j \ell_j \geq nL$. Nous notons par ailleurs $\ell = \max_j(\ell_j)$. La description de la procédure à employer pour calculer un générateur linéaire dans ce cas est orthogonale à l'utilisation de la version FFT de l'algorithme. Aussi discutons-nous les modifications à apporter à la version n'utilisant pas la FFT.

L'objectif du calcul du générateur linéaire est d'identifier trois matrices $f \in K[X]^{n\times n}$, $g \in K[X]^{m\times n}$, $e \in K[X]^{m\times n}$ vérifiant les hyptohèses du lemme 12.2 à une étape t du calcul. Puisqu'aucun coefficient de A n'est connu au-delà du degré ℓ, il nous importe bien entendu d'atteindre cet objectif pour une étape $t \leq \ell$. Nous formulons donc notre objectif comme suit :

$$Af = g + X^\ell e,$$
$$\ell - \max(\deg f, 1 + \deg g) \geq \lambda = \max(\deg D, 1 + \deg N).$$

Le moyen pour atteindre cet objectif reste le même, à savoir la considération de matrices F, G, E vérifiant à chaque étape t :

$$AF^{(t)} = G^{(t)} + X^\ell E^{(t)}.$$

Certaines colonnes de A étant connues de manière plus incomplète encore, nous devons imposer une nouvelle condition. Afin que les coefficients de degré ℓ de la j-ème colonne du produit AF puissent être calculés, il est nécessaire d'imposer la condition suivante sur la j-ème *ligne* de la matrice $F^{(\ell)}$.

$$X^{\ell-\ell_j} \mid F^{(\ell)}_{j,\cdot}.$$

Considérons la matrice d'initialisation telle que proposée page 131. La matrice d'initialisation $F^{(t_0)}$ est construite de sorte à forcer les m dernières colonnes de la matrice $[X^{t_0}](AF^{(t_0)})$ à être de rang plein. Les m dernières colonnes de $F^{(t_0)}$ extraient donc m colonnes parmi les colonnes des t_0 premières matrices a_i. Supposons maintenant que l'on impose la contrainte que les colonnes sélectionnées parmi les matrices $(a_i)_{i=0\ldots t_0-1}$ sont toujours parmi les c dernières colonnes de a_i, pour un entier c tel que $0 < c \leq n$. Pour satisfaire la contrainte sur le rang de $[X^{t_0}](AF^{(t_0)})$, ceci implique de prendre au minimum $t_0 = \frac{m}{c} + \epsilon$. Dans un tel cas, le seul coefficient des $n - c$ premières lignes de $F^{(t_0)}$ est celui provenant de la sous-matrice I_n occupant les n premières colonnes de $F^{(t_0)}$. Cette situation est illustrée par a figure 12.5.

Pour traiter le cas de séquences déséquilibrées, nous adoptons ce choix, en ne nous appuyant que sur les c dernières colonnes ayant la valeur ℓ_i maximale (les entiers ℓ_i étant triés, on a donc $\ell_{n-c} < \ell_{n-c+1} = \ldots = \ell_n = \ell$). Ceci n'est toutefois pas suffisant pour garantir la propriété voulue $X^{\ell-\ell_j} \mid F^{(t)}_{j,\cdot}$ dès l'étape $t = t_0$.

Étudions l'évolution de la matrice $F^{(t)}$ au cours des pas de l'algorithme tels qu'ils ont été décrits. La matrice $F^{(t)}$ est multipliée à droite par les matrices $\pi^{(t,t+1)} = \sigma R \Delta$. Elle subit donc des opérations sur les colonnes, en particulier via la matrice triangulaire supérieure R. Si l'on n'impose aucune

$$F^{(t_0)} = \begin{pmatrix} \ddots & & * & * \\ & I_n & \vdots & \vdots \\ & \ddots & * & * \end{pmatrix}; \quad F^{(t_0)} = \left.\begin{pmatrix} \ddots & & 0 & 0 \\ & I_n & * & * \\ & \ddots & * & * \end{pmatrix}\right\} n - c.$$

FIG. 12.5 : Forme de la matrice d'initialisation $F^{(t_0)}$ dans le cas normal (g.) et dans le cas de séquences déséquilibrées (d.).

restriction sur ces opérations, il est peu probable d'obtenir la condition $X^{\ell-\ell_j} \mid F^{(t)}_{j,\cdot}$ « par chance ». Pour y parvenir, nous imposons la condition suivante :

La colonne $F^{(t)}_{\cdot,j}$ n'est jamais utilisée pour l'élimination tant que $t \leq \ell - \ell_j$.

À une étape t donnée, les colonnes telles que $t \leq \ell - \ell_j$ sont appelées « inactives ». Ces colonnes inactives sont au nombre de $n-c$ à l'étape t_0, avec la notation prise plus haut. « Ne pas utiliser » les colonnes inactives pour l'élimination revient à les exclure complètement du calcul, et à les multiplier par X à chaque étape. L'entrée (j, j) est la seule entrée non nulle de la ligne et de la colonne j de R. Quant à la matrice Δ, elle multiplie non pas seulement m mais $m + \alpha$ colonnes par x, ou α désigne le nombre de colonnes inactives. Sous cette condition, la colonne j commence à être utilisée pour l'élimination uniquement à partir de l'étape $t = t_0 + \ell - \ell_k$. Nous décomposons les étapes t en plusieurs périodes en fonction du nombre de colonnes inactives. Certaines périodes peuvent être de durée nulle dans le cas où $\ell_j = \ell_{j+1}$:

- Pour $t_0 \leq t < t_0 + \ell_n - \ell_{n-1}$, il y a $n-1$ colonnes inactives. Le degré de $\pi^{(t,t+1)}$ est $m+n-1$. L'écart $t - \max(\deg F, 1 + \deg G)$ croît de $\frac{1}{m+n}$ en moyenne.
- Pour $t_0 + \ell_n - \ell_{n-k+1} \leq t < t_0 + \ell_n - \ell_{n-k}$, il y a $n - k$ colonnes inactives. Le degré de $\pi^{(t,t+1)}$ est $m + n - k$. L'écart $t - \max(\deg F, 1 + \deg G)$ croît de $\frac{k}{m+n}$ en moyenne.
- Pour $t_0 + \ell_n - \ell_1 \leq t < t_0 + \ell_n$, il n'y a plus de colonne inactive. Le degré de $\pi^{(t,t+1)}$ est m. L'écart $t - \max(\deg F, 1 + \deg G)$ croît de $\frac{n}{m+n}$ en moyenne.

En sommant la croissance des écarts $t - \max(\deg F, 1 + \deg G)$, on obtient :

$$\sum_{i=0}^{n-1} (\ell_{n-i} - \ell_{n-i-1}) \frac{i}{m+n} = \frac{1}{m+n} \sum_i \ell_i \geq \frac{n}{m+n} L.$$

Notons par ailleurs que les matrices $\pi^{(t,t+1)}$ intervenant dans l'une des étapes où il y a un nombre $\alpha > 0$ de colonnes inactives sont particulières. À la permutation près des colonnes induite par le tri des δ_j et la matrice de permutation Σ, ces matrices possèdent la propriété d'être diagonales par blocs et d'avoir un bloc identité de taille $\alpha \times \alpha$. Manipuler de telles matrices peut être fait de manière plus efficace qu'en les considérant comme de simples matrices de taille $(m + n) \times (m + n)$.

Chapitre 13

Algèbre linéaire sur machines

L'algèbre linéaire est une problématique omniprésente du calcul à hautes performances. Toutefois, comme nous l'avons indiqué au chapitre 11, nombre de considérations qui ont cours dans le domaine des calculs *numériques* sont invalides dans le contexte des calculs qui nous intéressent, à savoir principalement la résolution de systèmes linéaires creux intervenant dans des calculs de factorisation ou de logarithme discret.

Le présent chapitre vise à donner quelques éléments de description de notre travail d'implantation de l'algorithme de Wiedemann par blocs, tel qu'il est présent dans `cado-nfs` [90].

13.1 Briques de base pour un produit matrice $\times$ vecteur

L'emploi d'algorithmes *black box* met par définition une pression importante sur le produit matrice $\times$ vecteur. Les algorithme par blocs, et notamment l'algorithme de Wiedemann par blocs, en première ligne dans nos travaux, s'appuient sur la multiplication d'une matrice creuse par un bloc de vecteurs. Considérons une matrice M de taille $N \times N$, de coefficients $m_{i,j}$, et un bloc de k vecteurs noté x. La notation que nous employons pour x est $x = (x_0, \ldots, x_{N-1})$, où les coefficients x_i sont des blocs de k éléments du corps de base. Le produit Mx s'écrit $y = (y_0, \ldots, y_{N-1})$, où :

$$y_i = \sum_j m_{i,j} x_j.$$

Il est naturel de limiter cette somme aux coefficients non nuls de la matrice M. Lorsque le corps de base est $\mathbb{F}_2$, les coefficients non nuls $m_{i,j}$ sont égaux à 1, et les coefficients x_j et y_i peuvent être représentés sous forme de mots machines, codant par exemple 64 éléments distincts de $\mathbb{F}_2$. Aussi, l'opération de produit (par 1) peut être ignorée, et l'opération d'addition revient à un XOR (ou exclusif) bit à bit.

Cette opération de produit matrice $\times$ vecteur est primordiale dans le temps d'exécution des algorithmes *black box*. En présence d'une matrice dense, et dans le cas numérique, cette tâche est confiée à des sous-routines particulièrement optimisées appelées BLAS (*Basic Linear Algebra Subroutines*) [134]. Ces routines calculent un produit matrice vecteur, les coordonnées considérées étant par exemple des nombres flottants double précision IEEE-754.

Dans le cadre des calculs que nous considérons, la différence de corps de base empêche les BLAS de répondre au besoin de briques de base élémentaires pour les produits matrice $\times$ vecteur. D'autre part, les matrices que nous traitons sont creuses. Ceci est une deuxième raison faisant qu'une brique de base telle les BLAS, adaptée au contexte dense, ne peut être présentée comme une solution.

Les matrices creuses, bien entendu, se rencontrent dans le contexte numérique, comme nous l'avons indiqué au chapitre 11. Aussi, des proposition de *sparse BLAS* (BLAS creux) existent [62, 38]. Divers travaux menés dans cette direction produisent les constats essentiels indiquant les priorités d'optimisation [153, 108, 216, 215]. Ces dernières se concentrent sur la difficulté, en termes d'implémentation, que constituent les accès aléatoires à des tableaux de données. En effet, un produit

Algorithme spmv_naïf

Entrée : Un bloc de matrice de taille `nrows` × `ncols` ;
un tableau d'entiers `data` contenant les longuer des lignes et les indices des coefficients non nuls par ligne ;
un tableau de coefficients `coeff_data` contenant les valeurs des coefficients non nuls ;
un vecteur source `src_vec`
un vecteur destination `dst_vec` initialement nul.
Sortie : Le vecteur destination `dst_vec` rempli avec le résultat du produit.

```
for(int i = 0 ; i < nrows ; i++) {
    int row_length = *data++ ;
    for(int j = ; j < row_length ; j++) {
        int jj = *data++ ;
        coeff_type c = *coeff_data++ ;
        coeff_type a = src_vec[jj] ;    /* lecture coûteuse ! */
        dst_vec[i] += a * c ;
    }
}
```

Algorithme 13.1 : Un produit matrice × vecteur naïf.

matrice × vecteur effectué au moyen d'une routine naïve semblable à celle présentée par l'algorithme 13.1 se heurte principalement à la pénalité induite par la lecture du coefficient `src_vec[jj]`, qui ne peut structurellement bénéficier d'aucune des optimisations offertes par le microprocesseurs : mémoires cache de différents niveaux, mémoires tampon de translation d'adresses (TLB), etc.

Il est légitime de tenter de déterminer quelles sont les convergences possibles entre l'étude des BLAS creux dans le contexte numérique et les calculs que nous entendons mener dans le cadre de l'algèbre linéaire sur les corps finis, pour des systèmes émanant de problèmes de factorisation (définis sur $\mathbb{F}_2$) ou de logarithme discret (définis sur $\mathbb{F}_p$). Outre le corps de base, un élément de différenciation important entre les matrices que nous rencontrons et le contexte numérique peut être entrevu à l'observation de la figure 11.1 page 120. Alors que les deux matrices présentées peuvent légitimement être présentées comme « creuses », la distribution spatiale des différentes valeur de densité moyenne est sans comparaison. Les matrices rencontrées dans le contexte de la factorisation ou du logarithme discret n'ont pas de zone structurellement nulle, alors qu'une telle situation est commune dans un contexte de résolution d'EDP par exemple.

Les briques de base utilisées pour un produit matrice × vecteur pour des matrices creuses sur $\mathbb{F}_2$ doivent donc atteindre leur objectif en prenant bien en compte les spécificités suivantes, si l'on souhaite les comparer à leur analogue numérique.

– La différence de corps de base rend difficile de réemployer un code numérique existant (voir toutefois plus bas), et nécessite une nouvelle implantation. Une telle implantation, par exemple pour le cas de calculs sur $\mathbb{F}_2$, doit mesurer la différence de poids des opérations par rapport au contexte numérique. Une multiplication d'un coefficient de la matrice par un coefficient du vecteur (ou bloc de vecteurs) source nécessite uniquement une opération XOR, occupant généralement une fraction de cycle. L'opération équivalente dans le contexte numérique nécessite une opération FMA (Fused Multiply-Add), dont la latence est plus grande. Les points fins de l'implémentation en assembleur sont donc distincts, et il est attendu que le poids des

calculs, dans le cas du corps de base $\mathbb{F}_2$, soit finalement négligeable devant la problématique des accès mémoire.

– Les briques de base que nous désirons doivent s'adapter à un continuum de valeurs de densité. S'il existe des zones dans les matrices de factorisation où la densité des coefficients non-nuls est significative, la fraction du temps d'un produit matrice $\times$ vecteur passée sur ces zones est faible, la plus grande part du temps de calcul concernant les (vastes) zones les plus creuses : quand bien même les coefficients y sont plus rares, le nombre total de coefficients se trouvant dans les parties creuses de la matrice est dominant. Ceci accroît l'importance de l'implémentation utilisée pour traiter les zones les plus creuses.

Sur ces bases, il est approprié de considérer les travaux cités plus haut dans le contexte des BLAS creux numériques comme une source d'inspiration importante, car les constats essentiels sont aussi valides dans le contexte qui nous intéresse. Toutefois, ces constats ont vocation à servir de point de départ à une implémentation adaptée, afin d'obtenir de meilleures performances.

Sur la question de la nécessité d'une implantation adaptée, nous discutons brièvement une approche existante, à savoir celle des bibliothèques FFLAS et FFPACK [64]. Ces bibliothèques se concentrent sur le contexte dense, mais aspirent à réutiliser des composants tels les BLAS pour réaliser des calculs sur les corps finis. Bien que cette approche pragmatique s'avère payante pour traiter, par exemple, des calculs sur $\mathbb{F}_p$ avec p occupant 16 bits, il nous semblerait trop onéreux de traiter le cas de la multiplication d'une matrice creuse sur $\mathbb{F}_2$ par un bloc de vecteurs au moyen d'une approche semblable (i.e. en réutilisant des BLAS creux). La raison essentielle tient en la possibilité d'« encoder » la multiplication d'un coefficient non nul (c'est-à-dire, pour $\mathbb{F}_2$, de la constante 1) par un bloc de vecteurs au moyen d'opérations sur des nombres flottants. Pour qu'une telle opération permette, par exemple, l'accumulation de 16 valeurs sans conversion, il est envisageable de représenter un blocs de 13 coefficients sur $\mathbb{F}_2$ dans les 52 bits de mantisse d'un nombre flottant double précision, ces coefficients étant espacés de 4 bits. Ainsi, la somme d'au plus 16 valeurs de ce type rend possible la réduction. Une telle approche perd donc structurellement un facteur $\frac{64}{13} \approx 5$ en comparaison avec une implantation réalisant des opérations bit à bit.

Nous avons, dans le cadre de `cado-nfs` [90], programmé des briques de base raisonnablement efficaces pour le produit matrice $\times$ vecteur. Nous envisageons d'en fournir une description écrite dans un prochain travail, mais nous n'en donnons pas les détails dans ce mémoire. Il convient toutefois de souligner que ces travaux sont bel et bien des algorithmes plus que de simples implantations efficaces.

13.2 Mise en œuvre sur un réseau rapide

Les problèmes de factorisation et de logarithme discret amènent à considérer des matrices de taille importante. De telles matrices occupent aujourd'hui facilement plusieurs dizaines de Go, et pour les traiter de manière efficace, le recours à plusieurs ordinateurs est, sinon nécessaire, au moins très utile. Se pose alors la question de la possibilité d'exploiter un algorithme tel l'algorithme de Wiedemann par blocs en parallèle sur plusieurs machines. Nous avons indiqué en 11.2.2 comment il était possible avec cet algorithme d'utiliser plusieurs clusters distincts, et nous reviendrons sur ce point en 13.3. Cet aspect de distribution possible n'est pas incompatible avec la parallélisation du produit matrice $\times$ vecteur sur un cluster de plusieurs machines.

Nous nous plaçons, dans les paragraphes qui suivent, exclusivement dans le contexte de l'algorithme de Wiedemann par blocs. Nous étudions donc comment il est possible de calculer les termes successifs de la suite de vecteurs $(M^i v)_i$, où v est un vecteur de départ fixé.

Pour paralléliser ce calcul, il faut en premier lieu décider d'une répartition du travail à effectuer entre les différentes machines prenant part au calcul. Nous supposons que ces machines sont organisées selon une grille à deux dimensions de taille $v \times h$, avec h processeurs par ligne, et v processeurs par colonne.

Une approche peut consister à séparer la matrice en $v \times h$ blocs, le bloc (x, y) contenant les coefficients $m_{i,j}$ pour $\frac{i}{N} \in [\frac{y}{v}, \frac{y}{v} + 1[$ (les bornes étant $0 \leq x < h$ et $0 \leq y < v$). À l'observation de la figure 11.1 page 120, il est évident qu'une telle approche n'est pas une bonne idée, puisque les blocs constitués sont alors extrêmement déséquilibrés.

Il est préférable de choisir deux permutations S et T, et d'appliquer ce découpage par blocs à la matrice $M' = SMT^{-1}$. Il est possible de la sorte, étant donné les matrices rencontrées, d'équilibrer la densité des blocs de manière satisfaisante. De la sorte, on se ramène donc à un problème à deux facettes.

- Sur chacun des processeurs utilisés, on se ramène au problème du produit matrice $\times$ vecteur, au moyen par exemple d'un agencement des briques de base discutées plus haut. Chaque processeur calcule ainsi, en partant d'une fraction de vecteur source, sa contribution locale au vecteur résultat.

- Il convient ensuite de rassembler les contributions locales des différents processeurs, et de redistribuer le vecteur résultat pour la prochaine itération.

Le deuxième point ci-dessus induit des communications entre les lignes, puis entre les colonnes de la grille de taille $v \times h$ qui représente l'organisation de nos processeurs. Ces communications peuvent être programmées au moyen d'une interface comme MPI (Message Passing Interface), qui fournit des interfaces bien définies pour ces opérations dites collectives. Deux approches peuvent être envisagées.

- La première approche est représentée graphiquement par la figure 13.2. Sur une ligne donnée, un processeur en particulier rassemble les contributions locales de tous les processeurs de la ligne et en calcule la somme. Ceci revient à utiliser la primitive MPI appelée `MPI_Reduce`. Ensuite, les parties du vecteur produit Mv ainsi calculées sont redistribuées sur une colonne de la grille de processeurs, par une opération de diffusion. Cette opération de diffusion correspond à la primitive `MPI_Bcast`. Il est nécessaire pour que ce schéma fonctionne de choisir de manière appropriée le processeur, sur la grille, qui collecte les données, mais ceci relève du détail que nous n'explicitons pas.

- La seconde approche est représentée graphiquement par la figure 13.3, et revient à une parallélisation de la précédente. Sur chaque ligne de la grille de processeurs (une ligne comporte h processeurs), l'ensemble des indices traités est divisé en h intervalles. Les contributions locales des processeurs de la ligne correspondant aux indices de chacun des intervalles sont collectées par des processeurs distincts. Ceci correspond à la primitive MPI appelée `MPI_Reduce_Scatter`. Ensuite, pour le produit matrice $\times$ vecteur suivant, les coordonnées du vecteur produit Mv calculé sont redistribuées sur chaque colonne à partir des processeurs sur lesquels ceux-ci ont été calculés. Ceci correspond à la primitive `MPI_Allgather`. Dans ce cas encore, le choix des indices traités par chaque processeur est compliqué, et nous ne le détaillons pas.

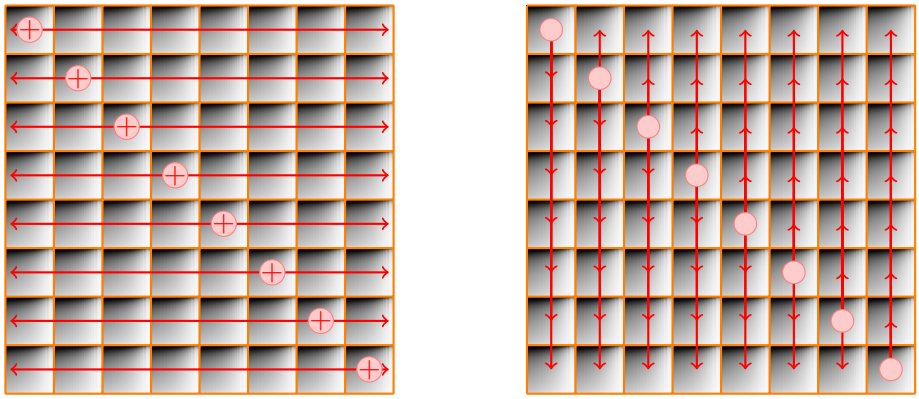


FIG. 13.2 : Approche simple de produit matrice × vecteur en parallèle.

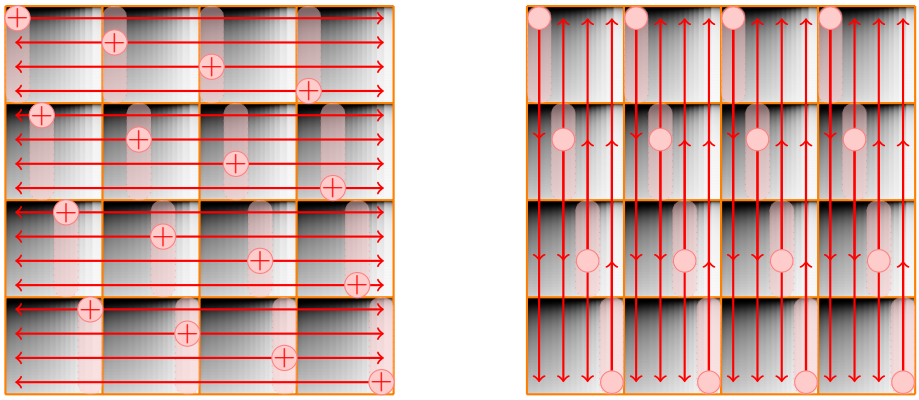


FIG. 13.3 : Approche distribuée pour le produit matrice × vecteur en parallèle.

La seconde approche est plus efficace que la première, car elle évite d'imposer à un processeur en particulier sur une ligne (respectivement, sur une colonne) de recevoir (respectivement, d'émettre) une quantité de données supérieure[1]. Ces deux approches sont implantées dans `cado-nfs`, et ont été utilisées avec succès pour les calculs d'algèbre linéaire pour RSA-768.

Cas de l'algorithme de Lanczos par blocs L'algorithme de Lanczos par blocs, vu en 11.1.2, vise à utiliser une matrice symétrique, pourtant les matrices considérées dans les problèmes que nous rencontrons ne le sont pas. Aussi, après le calcul du produit Mv vient le calcul du produit $M^T v$. Ceci implique, si l'on reprend l'organisation en grille de processeurs que nous venons d'évoquer, que les communications de coordonnées du vecteur calculé ne sont pas acheminées à l'identique. En fait, pour enchaîner une multiplication par M^T après une multiplication par M, chaque processeur ayant calculé une contribution à la coordonnée d'indice i du vecteur Mv a besoin de connaître la valeur finale de cette coordonnée pour effectuer le produit suivant par M^T. Dès lors, la primitive MPI à utiliser est différente, c'est la primitive `MPI_Allreduce`. Outre la différence de nom, il est important de constater qu'une seule opération collective en remplace deux, ce qui offre un léger avantage à

[1]La formulation en terme de quantités de données est contestable, puisque c'est la somme qui est collectée, et que dans le cas d'une diffusion, les données émises vers les différents processeurs sont identiques. Reformuler en termes de latence serait plus correct.

l'algorithme de Lanczos par blocs au niveau des communications pour la parallélisation. Ceci ne résout pas, toutefois, le fait que l'algorithme de Lanczos par blocs est inadapté à la *distribution*, au sens vu en 11.2.2.

Différence de taille avec certaines approches numériques La problématique d'optimisation des communications n'a absolument rien de spécifique au contexte de l'algèbre linéaire sur les corps finis. Toutefois, il convient de remarquer que les solutions apportées au problème analogue dans le cas des calculs numériques sont, elles, souvent spécifiques et inadaptées au contexte des corps finis. Ainsi, une approche comme celle développée en [154] minimise le coût des communications au prix du ralentissement de la convergence d'un calcul d'algèbre linéaire. Une telle approche n'a pas de sens dans le contexte des corps finis, puisque la notion de convergence ou de solution approchée n'existe pas.

13.3 Mise en œuvre sur une grille de calcul

Dans le cadre du calcul RSA-768, nous avons eu à résoudre un système linéaire à 192 millions d'équations et d'inconnues, la matrice correspondante occupant un peu plus de 105 Go dans un format simple, et sous une forme compressée pas moins de 75 Go.

Pour mener à bien ce calcul, nous avons utilisé différentes ressources de calcul. Logistiquement parlant, une méthode pour mener à bien un tel calcul peut impliquer les étapes suivantes :

- Décider des caractéristiques d'un matériel approprié.
- Acheter ce matériel.
- Disposer de l'usage exclusif et permanent de ce matériel.
- Lancer le programme, attendre le résultat.

Cette approche est l'approche « simple ». Parmi les partenaires participant à l'effort d'algèbre linéaire pour RSA-768, T. Kleinjung a employé cette méthode, et K. Aoki a utilisé un cluster déjà existant et mal adapté, mais dont il disposait de l'usage exclusif.

Nous avons souhaité, dans cette entreprise, montrer que l'aspect d'usage exclusif de la ressource n'est pas une nécessité. Ceci induit, par définition, une difficulté. L'accès à une ressource de calcul partagée va avec les contraintes de l'ordonnanceur de tâches (*scheduler*) qui lui est associé. Dans le cadre de la plate-forme Grid'5000 que nous avons employée, ceci implique qu'une tâche, lorsqu'elle démarre, ne dispose d'aucune donnée persistante locale sur les nœuds. Une tâche doit donc nécessairement importer la totalité des données qui lui sont nécessaires pour calculer, et exporter ses résultats. Bien qu'un tel mode de fonctionnement soit manifestement une contrainte, il est typique, lorsqu'on a un accès à une ressource, de n'être nullement en mesure d'en infléchir les politiques de gestion. Nous montrons dans [T14] que dans ce cas précis, il a été possible de travailler malgré cette contrainte.

La plate-forme de calcul que nous avons utilisée est la *grille de calcul* Grid'5000. Cette plate-forme est constituée de plusieurs clusters, répartis sur différents sites. Un cluster donné est (sauf exception) homogène, mais il n'y a pas d'homogénéité globale. La première de nos tâches à consisté à identifier les clusters les plus appropriés pour notre calcul, au vu du niveau possible de performances qu'ils offrent. Ainsi, la mesure de comparaison entre différents clusters est le temps par produit matrice $\times$ vecteur, puisque l'algorithme de Wiedemann par blocs se résume à un grand nombre de

(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)	(k)	(l)
Lausanne	56	2×AMD 2427	2.2	12	16	ib20g	12	144	4.3	4.8	40%
Tokyo	110	2×Pentium-D	3.0	2	5	eth1g	110	220	5.8	7.8	%
Grenoble	34	2×Xeon E5420	2.5	8	8	ib20g	24	144	3.7		30%
Lille	46	2×Xeon E5440	2.8	8	8	mx10g	36	144	3.1	3.3	31%
							32	256	3.8		38%
							24	144	4.4		33%
Nancy	92	2×Xeon L5420	2.5	8	16	ib20g	64	256	2.2	2.4	41%
							36	144	3.0	3.2	31%
							24	144	3.5	4.2	30%
							18	144		5.0	31%
							16	64		6.5	19%
Orsay	120	2×AMD 250	2.4	2	2	mx10g	98	196	2.8	3.9	32%
Rennes	96	2×Xeon 5148	2.3	4	4	mx10g	64	256	2.5	2.7	37%
							49	196	2.9	3.5	33%
Rennes	64	2×Xeon L5420	2.5	8	32	eth1g	49	196	6.2		67%
							24	144	8.4		67%
							18	144	10.0		68%
							8	64		18.0	56%

TAB. 13.4 : Différents temps par itération pour les clusters utilisés. (a) Lieu ; (b) Taille totale du cluster (nombre de nœuds) ; (c) Type de CPU ; (d) Fréquence CPU ; (e) Cœurs par nœud ; (f) RAM par nœud (Go) ; (g) Réseau de connection ; (h) Taille de tâche (nombre de nœuds) ; (i) nombre de cœurs utilisés ; (j) temps par itération en secondes (étape 1) ; (k) temps par itération en secondes (étape 3) ; (l) fraction du temps utilisée par les communications.

telles opérations. Une première donnée est donc constituée par une liste de configurations possibles en nombre de nœuds sur tel ou tel cluster, ordonnée par niveau de performance attendu. Notre approche de calcul se voulant opportuniste, nous avons souhaité avoir la possibilité de choisir la ou les ressources les plus performantes à un moment donné, parmi les ressources disponibles. Une telle liste est donnée par la table 13.4.

La disponibilité des ressources sur une grille de calcul est variable. Puisque plusieurs clusters sont présents sur la grille, dans une configuration favorable il est possible de calculer plusieurs séquences $(M^i v)_i$ en parallèle. Dans l'algorithme de Wiedemann par blocs, comme décrit en 11.2.2, un nombre n' de telles séquences peuvent être calculées indépendamment, de manière non synchronisée. Aussi, nous avons choisi pour le calcul RSA-768 de traiter $n' = 8$ séquences indépendantes, allouées comme suit aux différents partenaires :

- Les séquences $0, 1, 2, 3$ ont été traitées sur Grid'5000.
- Les séquences $4, 5$ ont été traitées à l'ÉPFL (Lausanne).
- Les séquences $6, 7$ ont été traitées à NTT (Tokyo).

Cette configuration a offert la possibilité d'utiliser jusqu'à quatre clusters de Grid'5000 simultanément lorsqu'ils étaient disponibles.

Les tâches essentielles, pour mener à bien une opération de ce type et utiliser une grille comme Grid'5000 pour un calcul comme la résolution du système RSA-768, sont la segmentation du travail

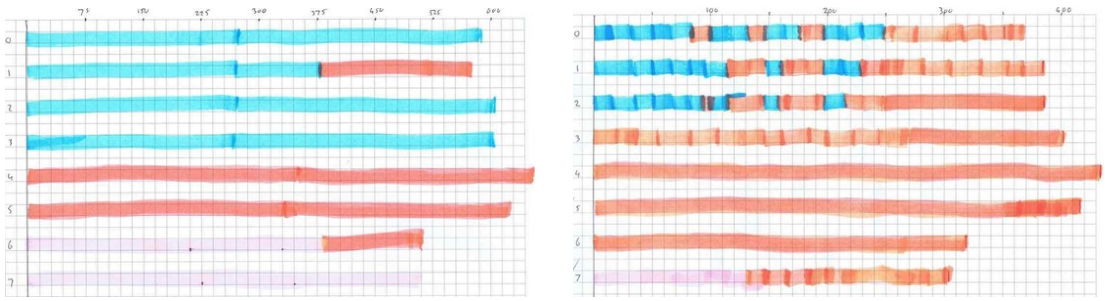


FIG. 13.5 : Suivi de l'avancement de calcul pour l'étape d'algèbre linéaire pour RSA-768 (étapes 1 et 3 de Wiedemann par blocs). (Suivi méticuleux par T. Kleinjung.)

en tâches élémentaires (ici, la réalisation de produits matrice $\times$ vecteur), et l'identification des tailles des données transférées, ainsi que l'ordonnancement de leurs transferts, afin de minimiser les délais de démarrage des tâches, et de ne pas perdre de calcul lors de leur achèvement. Les détails indiquant comment cet objectif a été atteint sont donnés dans [T14]. Un total d'une centaines d'années-CPU ont été utilisées pour ce calcul d'algèbre linéaire, en utilisant une assez large variété de clusters en fonction de leur disponibilité. La table 13.4 donne la liste de ces clusters. L'état d'avancement des calculs a été suivi à gros grain de manière assez artisanale par Thorsten Kleinjung, comme en témoigne la figure 13.5 qui reprend une copie de l'outil de suivi employé. Toute sophistication n'est pas toujours utile !

En montrant qu'il est possible d'exécuter l'algorithme de Wiedemann par blocs sur une grille de calcul, et pas nécessairement sur un ou plusieurs cluster dédiés, nous avons apporté au calcul RSA-768 une particularité par rapport aux records précédents. Cette approche, nous l'avons montré, est viable, mais consommatrice d'une énergie considérable en termes de développement.

Partie IV

Arithmétique

Chapitre 14

Arithmétique rapide

Une partie de nos travaux est de nature purement arithmétique, et nous les abordons dans cette partie. Le présent chapitre vise à offrir quelques éléments de contexte très brefs permettant de positionner nos travaux par rapport à d'autres travaux arithmétiques. Le vaste domaine de l'arithmétique des ordinateurs ne peut qu'être très mal effleuré par un court chapitre, aussi nous renvoyons à des monographies telles [33] pour une présentation considérablement plus aboutie.

14.1 Objets

La structure arithmétique par excellence est l'anneau des entiers $\mathbb{Z}$. Les structures que nous considérons sont bâties les unes après les autres, avec à l'origine du processus de construction l'anneau $\mathbb{Z}$. Les opérations arithmétiques que nous souhaitons effectuer font donc appel *in fine* aux opérations élémentaires sur les entiers, à savoir addition, multiplication, et division euclidienne.

Une distinction fondamentale dans les considérations arithmétiques a trait à la *taille* des objets considérés. Multiplier (par exemple) des entiers occupant plusieurs octets ou plusieurs mégaoctets sont des objectifs qui se distinguent non seulement par l'énoncé, mais aussi par la nature des réponses recherchées, et par les moyens pour y parvenir.

Nous séparons notre présentation des objets considérés selon ce critère essentiel de taille. Nous verrons que ce critère nous amène à identifier des applications distinctes pour les divers calculs que nous pouvons mener.

14.1.1 Arithmétique des petites tailles

Nous nous plaçons dans un contexte d'application où des objets de taille fixe doivent être manipulés. De telles situations se rencontrent fréquemment dans le contexte de la cryptologie. Bien que le cryptographe dispose généralement d'une famille d'instances d'un cryptosystème, paramétrées par un paramètre de sécurité (lequel conditionne en particulier la taille des objets manipulés), c'est bel et bien en général *une* instance particulière qui est déployée pour les fonctionnalités cryptographiques d'une carte à puce, ou d'un logiciel de signature électronique par exemple. Aussi, dans ce contexte, les opérations sur des nombres de taille fixe sont importantes : les optimiser peut se traduire par des économies de ressources, ou par un avantage comparatif face à un cryptosystème concurrent.

D'autre part, un effort de *cryptanalyse* repose souvent sur la capacité de l'attaquant à calculer très efficacement avec les données du cryptosystème. L'attaquant a bien sûr le loisir de mener des calculs échappant à ce cadre, dans des structures mathématiques sans lien direct avec le cryptosystème (par exemple l'algorithme NFS). Toutefois, il est également possible que les calculs de l'attaquant impliquent les mêmes objets que ceux traités par le cryptographe (exemple du calcul du logarithme discret par recherche de collisions [167], ou encore l'algorithme présenté au chapitre 8). Un travail de cryptanalyse nécessitant souvent l'exécution d'un nombre immense d'opérations sur des objets de taille constante, on comprend qu'il est important de rendre celles-ci efficaces. L'optimisation de

ces calculs revêt un objectif double. À la fois elle permet d'atteindre des tailles de problèmes plus importantes à ressources égales, mais elle permet aussi de mieux remplir la mission de la cryptanalyse, qui n'est pas tant en soi de battre des records, mais de donner une vue aussi précise que possible de ce qu'il est possible de réaliser comme calculs avec des ressources données.

Quels sont les « petits » objets ? D'une manière générale, il s'agit d'objets dont la taille est suffisamment petite pour qu'un code « générique », incluant par exemple des boucles, induise une pénalité importante. Un ordre de grandeur convenable peut être donné en nombre de mots machine utilisés pour représenter les données. En-deçà de dix mots machines (soit, pour des entiers sur une machine 64 bits, des entiers inférieurs à 2^{640}), on peut généralement parler de « petite taille ». L'un des apanages de la petite taille est l'idée que seule compte la vérité de l'expérience pour discriminer entre deux choix possibles. Par exemple, déterminer la meilleure approche entre une approche dite *schoolbook* et une approche à la Karatsuba, pour une multiplication de deux nombres entiers occupants six mots machine, n'est pas évident a priori.

Comme nous l'avons mentionné plus haut, le contexte cryptologique offre nombre d'instances de ces calculs en petite taille. Avant d'en lister quelques exemples, nous donnons d'abord un exemple qui à notre sens ne relève *pas* de la petite taille. Les tailles de clé utilisées aujourd'hui avec le cryptosystème RSA atteignent voire dépassent les 2048 bits, soit 32 mots machine (sur un processeur 64 bits). Nous considérons qu'il s'agit là d'objets de taille moyenne. Des objets plus petits interviennent par exemple dans les contextes suivants.

– Tous les systèmes cryptographiques employant des courbes algébriques mettent en avant leur résistance au calcul de logarithmes discrets : seuls les algorithmes en $\sqrt{\#G}$ s'appliquent, pour les systèmes prisés (genre 1 ou 2). Dès lors, pour atteindre un niveau de sécurité de 2^{128} par exemple, il suffit de considérer des courbes elliptiques sur un corps de cardinal environ 2^{256} (en genre 1) ou 2^{128} (en genre 2). On parle donc là d'opérations sur deux ou quatre mots machine. Le même contexte vaut bien entendu pour les travaux de cryptanalyse de ces mêmes systèmes.

  Dans le champ des études sur de tels systèmes cryptographiques, nous relevons donc l'importance de mener efficacement des calculs sur des petits corps finis. Mentionnons aussi, d'autre part, une fois abstraite la couche « corps finis », les travaux d'optimisation de l'arithmétique des diviseurs sur des jacobiennes de courbes. Nous avons brièvement abordé une telle considération en 8.7.1.

– Les tentatives de calcul de logarithmes discrets dans les corps finis, ou par une méthode de calcul d'index en général, reposent à un moment donné sur une étape d'algèbre linéaire. C'est le cas des algorithmes NFS et FFS, mais aussi des algorithmes présentés aux chapitres 8, 9, et 10. Dans tous ces cas, le système linéaire est défini modulo le cardinal du groupe de la forme $\mathbb{Z}/n\mathbb{Z}$ dans lequel le logarithme discret est défini. Ce groupe est généralement choisi de taille juste assez grosse pour contrer les attaques exponentielles, comme cela est fait dans l'algorithme de signature DSA [165]. Dès lors, pour les cas pertinents dans le contexte cryptographique, nous avons à traiter dans la cryptanalyse des objets occupant quelques mots machine, pas davantage.

  Il est également important de noter que ce contexte d'algèbre linéaire, eu égard à la présentation des algorithmes dit « black-box » que nous avons faite en 11.1.1, met particulièrement en avant l'opération « matrice $\times$ vecteur ». Aussi, les opérations induites sur le corps de base sont, pour une bonne part voire la quasi-totalité en fonction de l'algorithme choisi, des multiplications de coefficients de la matrice par des coefficients du vecteur. La matrice, par hypothèse, ne variant pas au cours de l'algorithme, de telles opérations impliquent généralement

des opérandes déséquilibrés, car les coefficients des matrices rencontrées sont généralement représentés par de petits entiers. Cette caractéristique ne peut évidemment être ignorée dans un travail d'implémentation optimisé pour ce contexte d'application.

Les applications que nous venons de citer nous ont ponctuellement amené à nous intéresser à l'arithmétique de ces petites tailles, et tout particulièrement à l'arithmétique des corps finis, comme nous l'abordons au chapitre 15.

14.1.2 Arithmétique des grandes tailles

L'opposé du spectre est bien entendu constitué par les grandes tailles. Le *calcul* connaît peu de limites. Calculer des milliers de milliards de décimales de π, par exemple, nécessite de manipuler des nombres entiers occupant plus de 2^{43} bits. De tels nombres occupent plusieurs téraoctets. Les manipuler efficacement nécessite une réflexion sur les outils algorithmiques utilisés. Ceci est vrai à la fois « au plus haut niveau », c'est-à-dire au niveau des algorithmes de type FFT qui sont adaptés à de telles tailles, mais aussi aux « niveaux inférieurs ». En effet, un algorithme comme l'algorithme de Schönhage-Strassen pour la multiplication de grands entiers ramène le problème à de nombreuses multiplications de nombres de taille moindre. Ces dernières doivent bénéficier, à leur tour, d'un travail d'optimisation afin d'obtenir de bonnes performances.

S'il est une définition qu'on pourrait donner de ce qui est « grand », une option peut être de s'intéresser à la limite entre ce qui est calculatoirement faisable ou pas. Calculer le $\gcd$ de deux entiers occupant plusieurs gigaoctets, par exemple, a un coût prohibitif lorsque des algorithmes quadratiques « naïfs » sont utilisés (plusieurs semaines de calcul avec des machines actuelles). En revanche, l'emploi d'algorithmes asymptotiquement rapides permet d'accomplir cette tache en un temps qui se compte en jours, au plus. L'arithmétique des grandes tailles est donc le domaine où excellent les algorithmes asymptotiquement rapides. Il est coutumier de rappeler les nombreuses prises de position un peu osées à un moment où à un autre, de divers auteurs des années 1970, arguant de la portée essentiellement théorique de tel ou tel algorithme asymptotiquement rapide, étant entendu que jamais les données utilisées « en pratique » ne pourraient mettre en valeur leur supériorité. Ces propos ont toujours été démentis tôt ou tard.

L'exemple que nous venons de donner sur le calcul de décimales de π n'est qu'un parmi d'autres. Sa mention est à double tranchant. S'il est avantageux, pour une fois, de pouvoir expliquer à monsieur tout-le-monde une application de l'arithmétique, la critique assez souvent entendue en réponse est l'inutilité d'une entreprise comme le calcul des décimales de π. Il est bien sûr aisé de trouver de nombreux autres exemples pour lesquels il importe de savoir manipuler des nombres de grande taille. Nous pouvons en donner quelques-uns, toute prétention à l'exhaustivité serait une gageure. Certains des exemples ci-dessous prêtent le flanc à la même critique triviale (et peu informée) d'inutilité, mais cela ne saurait en rien entamer notre intérêt et notre estime pour de tels travaux.

- La recherche de grands nombres premiers, notamment le projet GIMPS [219], repose sur des calculs sur des grands entiers de plusieurs dizaines de millions de bits.

- Le calcul de trinômes primitifs sur $\mathbb{F}_2$ [32,34] nécessite des multiplications et des carrés modulo des polynômes sur $\mathbb{F}_2$ dont le degré dépasse plusieurs millions.

- L'étape de racine carrée de l'algorithme NFS, mentionnée au chapitre 4, nécessite de manipuler des entiers de plusieurs millions de bits.

– L'algorithme de Wiedemann par blocs, notamment lorsqu'il intervient dans le cadre de l'algorithme NFS pour la factorisation d'entiers, amène à multiplier des polynômes sur des algèbres de matrices, d'un degré de l'ordre du million. (Cet algorithme est décrit au chapitre 12 de ce mémoire.)

– La construction de courbes aux bonnes propriétés cryptographiques recourt parfois à des opérations entre de « grands » objets, au sens large, pouvant souvent se ramener à des multiplications de grands entiers. C'est le cas du problème du comptage de points sur diverses courbes. Nous pouvons citer à cet égard [88, 91]. Dans une perspective voisine, la théorie de la multiplication complexe permet de construire des courbes ayant des propriétés choisies (cardinal de la jacobienne, anneau d'endomorphismes). Ces calculs passent par le calcul d'objets appelés polynômes de classes, objets vite considérables par leur taille. Là encore, l'algorithmique rapide est un point-clé de leur manipulation.

Comme nous l'avons dit, une telle liste n'est que très parcellaire. Nombreuses sont les situations où l'algorithmique asymptotiquement rapide est un ingrédient incontournable de travaux dans des domaines divers.

Nos travaux sur les grandes tailles ont principalement abordé le cas de la caractéristique 2 via la bibliothèque `gf2x`. Ces travaux sont abordés au chapitre 16. Nous notons en outre qu'un des aspects des travaux présentés au chapitre 16 est l'optimisation des routines de bas niveau, traitant de petits opérandes, et vues comme des « briques de base » dans la perspective de la multiplication de grands opérandes. Par cet aspect, le travail sur les grandes tailles peut rejoindre des préoccupations présentes aussi lors de l'étude des objets de petite taille.

14.2 Outils

14.2.1 Algorithmes

L'algorithmique efficace commence par l'algorithmique. Pour limiter l'étendue de notre propos, nous pouvons nous concentrer sur les opérations impliquant les entiers ou les polynômes sur des corps finis. Notre intérêt se porte en premier lieu sur l'opération « star », la multiplication.

La multiplication quadratique, aussi dite « naïve » peut parfois être jugée suffisante pour multiplier des objets de petite taille. Par exemple, pour multiplier deux entiers occupant chacun deux mots machine, il est probable que le surcoût en additions de l'algorithme de Karatsuba ne permette pas d'obtenir un code plus rapide que le code naïf utilisant quatre multiplications. Toutefois, ceci n'est pas une évidence absolue, et dépend réellement du coût relatif des additions et des multiplications. À titre d'exemple, dans le contexte de la multiplication de polynômes sur $\mathbb{F}_2$ (représentés de façon similaire à des entiers, en codant 64 coefficients sur un mot machine de 64 bits), une multiplication de Karatsuba « gagne » plus tôt, comme décrit en 16.2.2.

De nombreux algorithmes permettent d'améliorer la complexité. Si certaines bibliothèques s'en passent, ou s'en passaient[1], les bonnes performances d'une bibliothèque arithmétique reposent évidemment sur l'implantation de divers algorithmes lorsqu'ils sont pertinents. L'algorithme de Karatsuba, ou encore les extensions Toom-Cook, Toom-4, et autres, sont chacun performant sur une plage donnée.

[1] La bibliothèque BigNum, développée à la fin des années 1980 par l'INRIA et DEC PRL, n'avait dans sa version en langage `C` qu'une multiplication naïve, par exemple.

Au-delà de ces algorithmes, les algorithmes asymptotiquement rapides (quasi-linéaires) reposant sur des techniques de transformées de Fourier ou analogues (voir chapitre 16) sont utiles pour des nombres de grande taille. Les différents exemples donnés plus haut reposent sur l'existence d'implantations de tels algorithmes.

Cette variété d'algorithmes possibles, outre la mise en évidence d'un nécessaire travail d'implémentation de ces diverses méthodes, fait apparaître un besoin particulier, celui du *tuning*. En effet, la détermination du point à partir duquel l'algorithme X « bat » l'algorithme Y ne peut être qu'expérimentale : ce point dépend par exemple des caractéristiques intrinsèques de la machine sur laquelle est menée l'expérience.

Lorsque le point de vue se déplace de l'opération de multiplication à d'autres opérations (division, multiplication modulaire, racine carrée, ...), alors les techniques les plus efficaces *ramènent* le problème à celui de la multiplication, en utilisant en particulier des techniques telles le relèvement de Newton. Des détails concernant ce type d'algorithmes de réduction sont donnés dans [30] et surtout [33].

14.2.2 Techniques

Nous donnons quelques éléments indiquant les techniques qui peuvent être utilisées pour aboutir à des implémentations efficaces.

Code bas niveau Dès l'époque de l'interface de grands entiers de Pari/GP [47], il est apparu utile d'employer des microprogrammes en code assembleur pour effectuer une poignée de tâches élémentaires bien identifiées. Cette utilité n'est pas démentie aujourd'hui, et une bibliothèque comme GMP [96] qui fait preuve de son efficacité pour les opérations sur les nombres entiers s'appuie de manière cruciale sur ces routines écrites avec soin en assembleur. Bien entendu, la variété des microprocesseurs nécessite un effort constant de mise à jour de ces programmes en fonction de chaque nouvelle microarchitecture.

Cette utilité du travail au plus bas niveau est encore plus évidente lorsqu'on se place dans le contexte des petites tailles mentionné plus haut, et que par essence les seuls calculs menés ont lieu dans ces routines du plus bas niveau. Nous revenons sur l'importance de cet aspect au chapitre 15.

Par « bas niveau », nous entendons aussi la possibilité, et l'utilité, d'employer des instructions assembleur spécifiques à un microprocesseur ou à une famille de microprocesseurs, lorsque cela s'avère bénéfique pour la performance globale. Les travaux que nous mentionnons aux chapitres 15 et 16 exploitent cet aspect.

Utiliser des calculs flottants Un avatar de ce qu'on peut appeler une « considération bas niveau » est l'utilisation de nombres flottants pour effectuer des calculs entiers. Le principe est d'utiliser l'arithmétique présente dans les microprocesseurs pour manipuler des nombres flottants avec des mantisses de 53 bits pour mener des opérations sur des entiers, en veillant à éviter toute perte de précision.

Deux options sont possibles pour représenter des entiers en utilisant des flottants. En ne codant que 26 bits d'un entier dans la mantisse, on peut assurer de ne pas perdre de bit lorsque le produit est calculé. Il est aussi possible de coder 53 bits, et de reposer sur l'existence d'une unité de calcul FMA (fused-multiply-add) sur le microprocesseur pour calculer les bits de poids fort et de poids

faible d'un produit[2]. Cette approche est aujourd'hui assez théorique, puisque les unités FMA sont peu couramment disponibles encore aujourd'hui en 2012[3].

L'avantage potentiel de telles approches est la possibilité de tirer parti des bonnes dispositions des microprocesseurs actuels pour le calcul sur les nombres flottants. En fonction des générations de microprocesseurs, il peut en effet arriver que le coût relatif d'une multiplication de nombres flottants ou entiers soit déséquilibré, aussi bien en termes de latence qu'en termes de débit. Cela était particulièrement vrai lorsque les processeurs ne disposaient que d'opérations sur des entiers de 32 bits. Les processeurs actuels permettant des opérations avec des entiers de 64 bits, et permettant en outre de bonnes performances pour les instructions de multiplication, l'utilité d'une approche flottante est moins évidente. Toutefois, rien n'interdit qu'une évolution future renverse le point de vue.

Dans le contexte des calculs d'algèbre linéaire, recourir à des nombres flottants est une astuce d'organisation logicielle qui peut s'avérer payante, comme nous en faisons mention en 13.1. En effet, pour effectuer des produits de matrices à coefficients réels, un effort considérable a été fait dans l'implémentation des BLAS (*Basic linear algebra subroutines*). Dès lors, plutôt que de réimplanter ce type d'opérations sur des corps finis, il est proposé par [64] de ramener le produit de deux matrices définies sur un corps premier $\mathbb{F}_p$, via le théorème chinois, à de nombreux calculs sur des nombres premiers plus petits, et de déléguer ces sous-calculs à une approche flottante. Nous n'avons pas rencontré ce cas d'usage dans nos travaux, mais nous reconnaissons l'utilité potentielle d'une telle approche, à tout le moins à défaut d'implanter des équivalents des BLAS sur les corps premiers considérés (s'appuyant potentiellement sur la bibliothèque Mpfq discutée au chapitre 15).

[2]Une troisième approche est de réaliser un produit $53 \times 53 \to 126$ en quatre produits flottants. À notre connaissance, toutefois, cette dernière technique n'est pas ou peu employée pour ce type d'applications.

[3]Pour ce qui concerne les microprocesseurs les plus répandus dans les machines de bureau et ordinateurs portables, les microprocesseurs Intel devraient supporter un FMA matériel à partir de 2013, et le processeur AMD *bulldozer* supporte un FMA matériel depuis 2011.

Chapitre 15

Arithmétique rapide des petits corps finis : la bibliothèque `mpfq`

Nous reprenons dans ce chapitre quelques éléments de description de la bibliothèque Mpfq, développée en collaboration avec P. Gaudry. Ces éléments apparaissent dans l'article [T5].

15.1 Présentation

Les travaux que nous menons, notamment dans le domaine de la cryptologie des courbes algébriques, donnent une grande importance à l'arithmétique des corps finis, aussi bien pour le cryptographe, qui souhaite mener les opérations du protocole qu'il emploie avec un minimum de ressources, que pour le cryptanalyste, qui à l'aube de dépenser un temps de calcul important pouvant se compter en mois (éventuellement distribués) est fortement enclin à réfléchir à l'optimisation de ces opérations. Ces deux contextes visent à tirer parti des spécificités du corps fini manipulé afin d'obtenir des implémentations *ad hoc* particulièrement optimisées.

À titre personnel, nous avons (moi-même et Gaudry de même), lors de travaux précédents, choisi cette approche de l'implémentation de code spécifique pour des travaux de cryptanalyse, tels la mise en pratique de l'algorithme présenté au chapitre 8, ou bien encore le comptage de points [89]. Nous avons aussi observé comment une telle approche s'était montrée profitable pour la résolution des *challenges* Certicom pour le calcul de logarithmes discrets, beaucoup des premières instances ayant été résolues par Harley [100] en s'appuyant sur ce principe.

15.1.1 État de l'art précédent

En 2007, le paysage logiciel inclut de nombreuses bibliothèques permettant de manipuler des corps finis, et distribuées sous forme de code source. Nombre d'entre elles existent de longue date. Avant d'illustrer en quoi l'approche proposée par Mpfq apporte une nouveauté, nous donnons quelques exemples de telles bibliothèques.

- La bibliothèque NTL [196] propose entre autres une interface d'arithmétique des corps finis. NTL est une bibliothèque en C++, mais n'utilisant que très peu des fonctionnalités du langage. NTL s'appuie sur la bibliothèque GMP [96], et aussi partiellement sur l'emploi de nombres flottants comme décrit au chapitre précédent en 14.2.2. De la sorte, NTL atteint généralement un niveau d'efficacité très satisfaisant.

- La bibliothèque ZEN [41] est concentrée sur l'arithmétique des corps finis, et possède notamment la capacité d'abstraction nécessaire pour gérer des extensions arbitraires de corps finis (y compris, en particulier, des tours d'extensions). Bien que ZEN soit programmée en C, cette bibliothèque inclut tout un mécanisme orienté objet à la X11, où chaque interface documentée se résout en un appel de fonction indirect via une table attachée à l'objet manipulé. Une

implantation en C++ qui imiterait le concept de ZEN s'appuierait sur le mécanisme de classes virtuelles, ou classes abstraites. La recherche de la meilleure performance nécessite d'éliminer autant que faire se peut cette étape d'indirection. Ceci est possible, mais limité, car le nom exact des fonctions à appeler, qui varient en fonction du corps considéré, relève d'une interface non documentée.

- La bibliothèque Miracl [191] est focalisée sur l'application cryptographique, notamment l'emploi des courbes elliptiques. L'inévitable couche d'arithmétique de corps finis de cette bibliothèque a de bonnes performances.

- La bibliothèque Givaro [63] est une bibliothèque C++, annonçant notamment de bonnes performances pour les calculs sur les corps premiers avec p limité à un mot machine.

15.1.2 Gains à la compilation

L'ensemble des bibliothèques que nous venons de lister ont un point commun. Elles aspirent, via le même code, à pouvoir traiter une variété de corps finis. Ce propos est bien sûr schématique. Suivant les choix, le même code n'a pas vocation à traiter la *totalité* des corps finis, bien entendu. Mais il est fréquent de constater que les opérations sur deux corps premiers, de taille voisine ou pas, empruntent exactement le même chemin de code. S'il est fréquent d'avoir un code séparé pour les corps premiers $\mathbb{F}_p$ avec p tenant dans un mot machine, ou un demi-mot machine, il est en revanche quasi-systématique que le traitement pour p tenant dans (par exemple) trois mots machine ne se différencie nullement de celui utilisé pour p tenant dans dix mots machine.

Pour une application de cryptographie, comme nous l'avons déjà mentionné, le corps fini dans lequel sont menés les calculs est connu à l'avance. Aussi, dès la compilation du code, il est possible de tirer parti de cette information. Ceci s'avère particulièrement important pour manipuler des corps finis de petite taille, où les objets n'occupent pas plus de quelques mots machine.

Diverses optimisations sont possibles, nous en mentionnons quelques-unes.

- Déroulage de boucles, ou *loop unrolling*, et élimination de branchements. Un code pouvant traiter des objets de taille a priori arbitraire emploie un mécanisme de boucle. Lorsque la taille des objets est connue à l'avance, le nombre d'itérations de la boucle est connu aussi. Dès lors, il est possible de remplacer une boucle, effectuant par exemple trois itérations, par le nombre correspondant de répétitions du corps de la boucle. Ce faisant, les instructions de branchements, ou sauts, intervenant dans la boucle peuvent être éliminées.

- Choix du meilleur algorithme. Le meilleur algorithme pour effectuer une tâche donnée (par exemple la multiplication de deux entiers) dépend en particulier de la taille des objets manipulés. Une bibliothèque comme les bibliothèques listées plus haut a vocation à effectuer un tel choix. Lorsque la taille des objets considérés est constante, ce choix est aussi constant. Ceci rentre dans une logique semblable au point précédent. Fixer la taille fixe aussi un grand nombre de valeurs qui déterminent le chemin pris par le programme, ce qui permet de limiter à l'essentiel le code pour une utilisation *ad hoc*.

- Optimisation en fonction du corps fini. Lorsqu'un corps fini précis est spécifié (par son polynôme de définition pour une extension, ou par le module pour un corps premier), alors il est possible d'en tirer parti pour l'opération de réduction. À titre d'exemple, il est assez évident qu'il est possible d'optimiser la réduction modulaire du produits de deux entiers modulo un nombre premier de la forme $2^n - \epsilon$.

Par essence, les bibliothèques à portée généraliste citées plus haut ne peuvent bénéficier de telles optimisations. L'objectif premier de MPFQ est donc de modifier cette donnée initiale, et de proposer une bibliothèque proposant du code adapté à un corps fini au temps de compilation. Implicitement, ceci signifie que le code de la bibliothèque doit être *généré* d'une façon ou d'une autre afin de tirer parti au mieux des points d'optimisation que nous venons de citer.

15.2 Éléments de design

15.2.1 Langage

Une réponse trop hâtive à nos prérequis est celle d'une approche orientée objet. Au même titre que la bibliothèque ZEN mentionnée plus haut, une telle approche permet de rendre agréable la programmation des fonctions qui utilisent la bibliothèque, mais ne répond pas de façon satisfaisante au besoin d'efficacité, puisque les appels de fonction indirects inhérents à ce modèle de programmation ont un coût absolument prohibitif.

Nous n'ignorons pas que le terme de « généricité » recouvre pour certains auteurs le recours à des mécanismes dits de *meta-programming*, utilisant par exemple le recours aux *templates* du langage `C++`. S'il est vrai qu'il est possible avec une telle technique d'« aplatir » une boucle dès lors que son nombre d'exécutions est connu au temps de compilation, notre point de vue sur ce problème est que la variété des optimisations qui sont possibles en considérant des paramètres fixes ne peut qu'être très mal capturée par une approche de *templates*. En effet, nous ne pensons pas qu'il *existe* une règle unique menant à l'écriture d'un code efficace pour toutes les tailles. Aussi, si un mécanisme de *templates* peut se montrer à même de proposer du code efficace, c'est par exemple au moyen de « spécialisations ». Nous ne rejetons pas une telle approche. Simplement, nous nous cantonnons à l'étude de ces spécialisations à proprement parler, et excluons délibérément de notre champ d'étude le « sucre syntaxique » ajouté par le langage autour de ces dernières.

Pour cette raison, MPFQ est bâti autour d'un mécanisme de génération de code de bas niveau, utilisant le langage `perl`. La plupart du code de MPFQ est donc écrite en `perl`, et c'est ce code `perl` qui, en fonction du corps fini sélectionné, effectue le choix d'algorithme, et génère du code en langage `C`.

15.2.2 Types et interface de programmation

Le choix de `perl` ne doit pas être compris comme la porte ouverte à un galimatias de codes générés hétéroclites. Bien au contraire, MPFQ vise à imposer une cohérence au niveau des types utilisés, et des fonctions présentes.

Un corps fini (ou éventuellement une famille de corps finis) pour lequel est généré un code source s'identifie par une étiquette que nous appelons `TAG`. Cette étiquette désigne une implémentation plutôt qu'un corps à proprement parler, puisqu'il est concevable que deux implémentations concurrentes de corps compatibles puissent être générées. Les étiquettes choisies sont utilisées ensuite comme des parties d'identifiants utilisés par le langage `C`, mais sont en dehors de cela libres de choix. À titre d'exemple, l'implémentation standard produite par MPFQ du corps à 2^{63} éléments a pour étiquette `2_63`. Comme autre exemple, l'arithmétique sur des corps premiers $\mathbb{F}_p$ où p est représenté sur trois mots machine, et utilisant la représentation de Montgomery [156], correspond à l'étiquette `pm_3`. Tous les types `C` et les noms de fonctions générés ensuite pour ces corps sont préfixés par `mpfq_TAG`. Ainsi, l'interface générée par MPFQ définit[1] :

[1] L'astuce du tableau de taille 1, utilisée par la bibliothèque GMP, est aussi utilisée par MPFQ. Aussi associe-t-on à un

`mpfq_TAG_field`, `mpfq_TAG_field_ptr` :
: Ce type correspond au corps manipulé. Il est concevable que l'objet en question soit une variable globale du programme, et il est également concevable que cet objet soit structurellement vide et ne comporte aucune information, auquel cas cet argument est ignoré par les fonctions de la bibliothèque. Ces deux cas sont illustrés par les usages suivants. Pour une implémentation de corps premier s'adaptant à divers modules, mais sous la contrainte commune de tenir dans un mot machine, il est utile de communiquer la connaissance du nombre premier p aux fonctions. C'est l'objectif de ce type. A contrario, une implantation réellement spécifique à un corps unique n'en a pas besoin.

`mpfq_TAG_elt`, `mpfq_TAG_{src,dst}_elt` :
: Ces types correspondent aux éléments des corps considérés. Les variantes `{src,dst}` spécifient le caractère `const` ou non des arguments des fonctions, et apparaissent dans les prototypes.

Les fonctions les plus courtes sont produites sous la forme de fonctions `inline` dans un fichier d'en-tête. À titre d'exemple, pour l'addition de deux éléments du corps à 2^{89} éléments, opération assez simple s'il en est, Mpfq produit le code suivant dans un fichier d'en-tête (la machine cible dispose ici d'un processeur 64 bits) :

```
static inline mpfq_2_89_add(mpfq_2_89_field_ptr K,
  mpfq_2_89_dst_elt r, mpfq_2_89_src_elt s1, mpfq_2_89_src_elt s2)
{
    r[0] = s1[0] ^ s2[0];
    r[1] = s1[1] ^ s2[1];
}
```

15.2.3 Génération de code

La capacité de `perl` à effectuer toutes sortes de manipulations de texte et de fichiers rend assez aisée la production des fichiers sources souhaités. D'autre part, le cœur de Mpfq est un moteur, permettant à l'implantation réelle du code `perl` générant les fonctions d'être écrit de manière systématique, déléguant au moteur ces manipulations de texte. À titre d'exemple, la fonction d'addition ci-dessus est générée par le code `perl` suivant, qui hérite des éléments de contexte pertinents (notamment la variable `$eltwidth`), et qui renvoie les informations nécessaires pour que la génération de code soit effectuée. L'adhérence aux prototypes définis par l'interface de programmation de Mpfq étant une contrainte par construction, cette information est suffisante.

```
sub code_for_add {
    my $code = '';
    $code .= "r[$_] = s1[$_] ^ s2[$_];\n" for (0..$eltwidth-1);
    return [ 'inline(K!,r,s1,s2)', $code ];
}
```

Toutefois, pour illustrer mieux encore la puissance de l'utilisation d'un langage de script comme `perl` pour la tâche de génération utile à Mpfq, considérons la programmation de la fonction trace

type quelques types dérivés, compatibles au mot-clé `const` près.

sur un corps de caractéristique 2. Les éléments d'un tel corps sont (par exemple) représentés dans une base polynomiale, relativement à un polynôme de définition fixé. Aussi a-t-on :

$$\mathrm{Tr}_{\mathbb{F}_{2^n}/\mathbb{F}_2}(a_0 + \cdots + a_{n-1}z^{n-1}) = \sum_{i=0}^{n-1} a_i \,\mathrm{Tr}_{\mathbb{F}_{2^n}/\mathbb{F}_2}(z^i).$$

Les valeurs $\mathrm{Tr}_{\mathbb{F}_{2^n}/\mathbb{F}_2}(z^i)$ ci-dessus sont des invariants du corps et peuvent être précalculées. Il est aisé pour un code de génération en `perl`, de déléguer cette tâche à un programme auxiliaire écrit dans un autre langage. Ainsi, MPFQ parvient à générer le code suivant pour calculer la trace dans $\mathbb{F}_{2^{197}}$, défini par le polynôme $z^{197} + z^9 + z^4 + z^2 + 1$:

```
static inline
unsigned long mpfq_2_197_trace(mpfq_2_197_dst_field K, mpfq_2_197_src_elt s)
{
    return ((s[3]>>3) ^ (s[3]>>1) ^ s[0]) & 1 ;
}
```

15.2.4 Fonctionnalités de l'interface

L'interface de programmation du code généré par MPFQ permet les opérations suivantes sur les corps finis considérés.

- Opérations de base : entrées/sorties, conversions (notamment depuis et vers les types GMP), tirages aléatoires, comparaisons.
- Opérations arithmétiques : addition, soustraction, multiplication, inversion, division, puissance, automorphisme de Frobenius, racine carrée.
- Autres opérations : norme, trace, résolution d'équation $x^p + x + a = 0$, transformée de Hadamard d'un quadruplet[2].
- Opérations « non réduites » : le résultat d'une multiplication n'est pas réduit immédiatement mais stocké dans un type étendu (typiquement occupant une taille double), dans lequel plusieurs tels résultats peuvent être accumulés.
- Interfaces additionnelles expérimentales[3] : Manipulations de vecteurs et de polynômes, interface pour la programmation parallèle et l'échange de données via MPI.

Notons que toutes ces opérations ne sont pas nécessairement générées, comme les exemples cités plus haut, en tant que fonctions `inline`. Certaines sont générées dans un fichier `C`, lorsque le coût d'un appel de fonction est tolérable.

15.3 Performances

Nous reprenons ici les tableaux de performance donnés en 2007 dans [T5], qui indiquent comment les opérations élémentaires avec MPFQ se comparent en terme de performances à certaines des autres bibliothèques citées.

[2]La transformation de Hadamard d'un quadruplet (a, b, c, d) est le quadruplet $(a + b + c + d, a - b + c - d, a + b - c - d, a - b - c + d)$. Il est aussi possible de voir cette opération comme la transformation d'un vecteur v à quatre coordonnées en le vecteur $v \times H$, où H est la matrice $M \otimes M$, cette dernière matrice M étant $\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$.

[3]Ces interfaces, utilisées par [91] et [90] ne sont pas disponibles dans la version de MPFQ actuellement téléchargeable.

MPFQ :

	1 mot	2 mots	3 mots	4 mots	$2^{127}-735$	$2^{255}-19$
add	2	4	5	7	4	8
sqr	67	108	170	230	14	30
mul	66	109	180	240	16	45
inv	420	2600	4600	7500	2600	7400

NTL :

	1 mot	2 mots	3 mots	4 mots
add	40	42	36	47
sqr	120	150	230	290
mul	120	150	230	290
inv	1600	4400	6600	9200

ZEN/ZENmgy :

	1 mot	2 mots	3 mots	4 mots
add	8/11	44/44	44/44	48/49
sqr	62/90	270/170	420/270	520/320
mul	68/95	300/180	450/270	600/340
inv	1700/2100	3300/4300	4800/5900	6500/7500

TAB. 15.1 : Temps (nanosecondes, 2 chiffres significatifs) pour les opérations élémentaires sur $\mathbb{F}_p$ (AMD Opteron 250, 2.40 GHz).

15.3.1 Corps premiers

Pour les corps premiers, les tables 15.1 et 15.2 indiquent le temps pris en nanosecondes pour quelques opérations élémentaires, dans le cas de corps dont le cardinal tient sur 1, 2, 3, ou 4 mots machine. Les temps sont donnés sur deux types de processeurs distincts. Pour MPFQ, nous incluons aussi des mesures pour deux corps premiers de forme spécifique (dite « pseudo-Mersenne », soit $2^n - \epsilon$). Inclure ces corps premiers est inutile pour les autres bibliothèques, qui ne sont pas en mesure de tirer parti de la forme spéciale du nombre premier. Les implémentations générées par MPFQ sont sans grande originalité, mais programmées en assembleur pour les opérations essentielles (multiplication et réduction notamment). Pour la bibliothèque ZEN, nous donnons aussi les temps en activant la représentation de Montgomery, qui permet d'obtenir une multiplication plus efficace.

Il est manifeste, à l'observation de ces tables, que le gain obtenu est très significatif, notamment lorsque le nombre premier choisi rend possible une optimisation de la réduction. D'autre part, nous pouvons remarquer que pour ces petites tailles, il est pertinent de garder à l'esprit que les additions et soustractions ne sont nullement « gratuites », comme il est parfois d'usage de considérer. Dès lors, les comparaisons de formules explicites pour l'arithmétique des courbes doivent *aussi* veiller quelque peu aux additions[4].

[4]Ajoutons toutefois qu'à la même échelle que les additions peuvent se trouver les copies, déplacements de registres, défauts de cache et autres, qui amènent à pondérer tout jugement de comparaison de deux algorithmes d'addition sur des jacobiennes de courbes uniquement à la lumière de leurs nombres respectifs d'opérations.

MPFQ :

	1 mot	2 mots	3 mots	4 mots	$2^{127}-735$	$2^{255}-19$
add	1	2	4	8	3	8
sqr	73	110	180	240	17	40
mul	74	120	190	260	19	53
inv	300	2000	3600	5800	2000	5800

NTL :

	1 mot	2 mots	3 mots	4 mots
add	38	45	53	67
sqr	110	130	210	270
mul	110	140	210	270
inv	1200	3400	5800	8000

ZEN/ZENmgy :

	1 mot	2 mots	3 mots	4 mots
add	6/6	41/41	46/46	57/57
sqr	52/52	280/120	400/170	550/250
mul	52/60	280/120	400/180	590/260
inv	1000/1000	2500/3000	3800/4300	5000/5900

TAB. 15.2 : Temps (nanosecondes, 2 chiffres significatifs) pour les opérations élémentaires sur $\mathbb{F}_p$ (Intel Core2 6700, 2.66 GHz).

15.3.2 Corps de caractéristique 2

En caractéristique 2, les graphiques de la table 15.3 donnent une variété de temps pour des corps jusqu'à $\mathbb{F}_{2^{251}}$. Pour la multiplication et le carré, les graphiques font apparaître pour MPFQ la multiplication « non réduite » en plus de l'opération de multiplication. La différence entre ces deux opérations est l'opération de réduction, sensible aux propriétés du polynôme de définition choisi. Les techniques utilisées pour programmer les opérations en caractéristique 2 ressemblent en tout point à celles discutées pour les briques de base élémentaires de la bibliothèque `gf2x` discutée en 16.2.1.

L'observation de la table 15.3 indique que MPFQ offre un gain substantiel par rapport aux bibliothèques concurrentes, exception faite de l'inversion pour laquelle le code présent dans MPFQ n'est pas optimisé.

Nous donnons quelques commentaires additionnels sur ces données. Ces graphiques datent de 2007, et ne prennent pas en compte quelques modifications ultérieures apportées à la bibliothèque MPFQ, qui ont permis d'en améliorer les résultats (en particulier, le positionnement inattendu de « sauts » à 125 et 250 bits a été corrigé, ces derniers sont plus logiquement rencontrés dans la version actuelle à 128 et 256 bits). Ils ne prennent pas non plus en compte l'instruction matérielle `pclmulqdq` disponible sur les processeurs Intel à partir de la génération Westmere (2010). Cette instruction est toutefois discutée dans le chapitre 16. Notons enfin qu'à la suite de nos travaux, et des travaux sur la bibliothèque `gf2x` décrite au chapitre 16, la bibliothèque NTL à compter de la version 5.5 s'appuie sur notre code pour les manipulations de polynômes sur $\mathbb{F}_2$. Aussi les anomalies de performances observables sur les courbes de la table 15.3 ont-elles été corrigées.

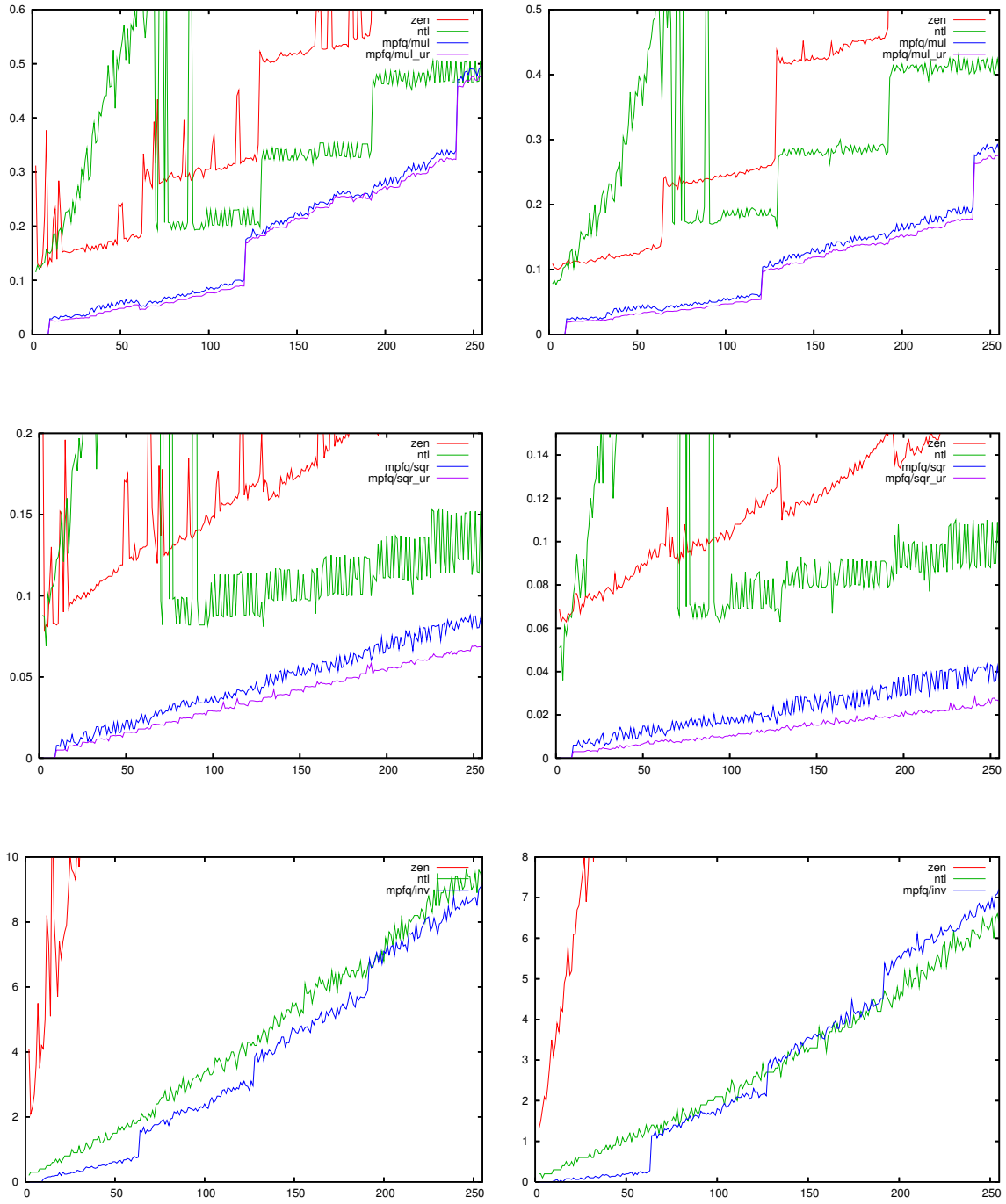


FIG. 15.3 : Temps (microsecondes) pour la multiplication, le carré, et l'inversion dans $\mathbb{F}_{2^n}$. (g) : AMD Opteron 250, 2.40 GHz, (d) : Intel Core 2 6700, 2.66 GHz

Courbe	Corps	AMD Opteron 250, 2.4GHz		Intel Core2 6700, 2.66GHz	
		# cyc./s. mult.	# s. mult. / s	# cyc./s. mult.	# s. mult. / s
`curve25519`	$\mathbb{F}_{2^{255}-19}$	307 000	7 800	386 000	6 900
`surf127eps`	$\mathbb{F}_{2^{127}-735}$	279 000	8 600	410 000	6 500
`curve2_251`	$\mathbb{F}_{2^{251}}$	2 070 000	1 100	1 350 000	2 000
`surf2_113`	$\mathbb{F}_{2^{113}}$	1 060 000	2 200	713 000	3 700

TAB. 15.4 : Exemples de temps obtenus avec MPFQ pour les opérations de base (s. mult. : multiplication scalaire) sur diverses courbes algébriques offrant un niveau de sécurité comparable ($\approx 2^{128}$).

15.4 Application : cryptologie des courbes algébriques

La vocation initiale de MPFQ a été de proposer des implémentations efficaces de cryptosystèmes utilisant des courbes algébriques. L'article [T5] présente quelques résultats obtenus sur des courbes de genre 1 et 2 correspondant approximativement à un niveau de sécurité 2^{128}, avec MPFQ. L'un de ces exemples reproduit le choix `curve25519` proposé par Bernstein [23].

Les temps indiqués par la table 15.4 ont établi des records de vitesse en 2007. Ils ont constitué la première illustration pratique du fait que les courbes de genre 2 pouvaient offrir des performances plus élevées que les courbes de genre 1, notamment grâce aux formules utilisant des fonctions thêta, proposées par Gaudry [87]. Depuis, ces travaux ont été poursuivis par d'autres auteurs [192, 143, 8].

15.5 Statut de la bibliothèque et travaux ultérieurs

La bibliothèque MPFQ est publiée à l'adresse `http://mpfq.org/`. Elle évolue uniquement au gré des usages qu'en font les auteurs. Certains développements récents ne sont pas disponibles sur la version téléchargeable.

La bibliothèque MPFQ a initié plusieurs travaux se plaçant en héritiers. Notamment la bibliothèque EECM-MPFQ a exploité MPFQ pour obtenir de bonnes performances pour l'algorithme ECM [24]. Pour l'application aux opérations sur les courbes algébriques, le flambeau a été repris par divers travaux, notamment [192, 143, 8], qui ont largement amélioré les performances obtenues par MPFQ, en poursuivant les mêmes principes.

Chapitre 16

Arithmétique des polynômes sur $\mathbb{F}_2$: la bibliothèque `gf2x`

Ce chapitre décrit la bibliothèque `gf2x`, qui remplit une unique mission : multiplier des polynômes à coefficients dans $\mathbb{F}_2$. Son interface est éminemment simple, et ne comporte que l'unique fonction nécessaire à cette opération de multiplication. Ce chapitre reprend des éléments de l'article [T10], qui décrit la bibliothèque `gf2x` (collaboration avec R. P. Brent, P. Gaudry, et P. Zimmermann). Quelques compléments relatifs à l'algorithme de Cantor et la variante de Gao-Mateer sont ajoutés.

Le travail sur la bibliothèque `gf2x` a été reconnu puisque cette bibliothèque est désormais utilisée comme composant optionnel de la bibliothèque NTL [196] depuis 2009 (versions 5.5 et suivantes).

16.1 Contexte

Un polynôme à coefficients dans $\mathbb{F}_2$ peut être représenté sur ordinateur par un tableau de mots machine. Un mot de 64 bits peut contenir 64 coefficients, aussi un polynôme de degré n peut-il être stocké sur $\lceil (n+1)/64 \rceil$ mots machine. Une telle représentation est dite « dense », par opposition aux représentations creuses, adaptées aux polynômes ayant particulièrement peu de coefficients non nuls.

Être en mesure de manipuler efficacement des polynômes à coefficients dans $\mathbb{F}_2$ est un point-clé pour l'efficacité des implémentations de briques de base cryptographiques employant des courbes sur les corps finis. D'autres applications sont moins évidentes : par exemple dans le contexte de l'algorithme NFS pour la factorisation d'entiers, l'une des façons d'aborder l'algorithme de Berlekamp-Massey matriciel (décrit au chapitre 12), qui est l'étape centrale de l'étape d'algèbre linéaire, est de fournir une implémentation optimisée de l'arithmétique de polynômes sur $\mathbb{F}_2$. Une autre application est donnée par la recherche de trinômes binaires primitifs [32, 34].

L'algorithmique des polynômes sur $\mathbb{F}_2$ est décrite par de nombreux travaux, par exemple l'ouvrage [84], ainsi que les références indiquées dans [T10]. Notamment, des adaptations (non triviales) des algorithmes asymptotiquement rapides de FFT au contexte de la caractéristique 2 ont été proposées par Schönhage [189] et Cantor [37]. Les discussions sur les choix d'implémentation pour l'arithmétique des polynômes sur $\mathbb{F}_2$ sont présentes dans [48, 98].

Les bibliothèques logicielles permettant de manipuler efficacement les polynômes sur $\mathbb{F}_2$, en revanche, ne sont pas légion. Une implémentation est décrite dans [82, 83, 85], mais elle semble ne plus être disponible. Depuis une dizaine d'années, la seule bibliothèque sur laquelle il s'est avéré possible de s'appuyer pour une implémentation raisonnablement efficace a été la bibliothèque NTL [196][1].

La bibliothèque `gf2x` propose une implémentation « fraîche » de la multiplication de polynômes sur $\mathbb{F}_2$. L'objectif qui a motivé initialement l'écriture de la bibliothèque `gf2x` est la multiplication de polynômes de grand degré, dans le contexte de [32, 34]. Toutefois, l'efficacité en la matière requiert

[1] Naturellement, la comparaison que nous menons avec NTL précède l'inclusion de `gf2x` comme composant de NTL.

une implémentation efficace pour toutes les tailles, puisque les multiplications de polynômes de grand degré se ramènent à des multiplications de polynômes de degré plus petit. Dans ce chapitre, nous présentons rapidement les techniques employées pour obtenir des résultats satisfaisants pour toutes les tailles.

16.2 Petites tailles

16.2.1 Multiplications de 64 bits : `mul1`

Version sans support matériel pour $\mathbb{F}_2[x]$

Nous considérons d'abord comme brique de base l'opération consistant à multiplier deux polynômes binaires de degré au plus 63, représentés chacun sur un mot machine, le résultat étant représenté par deux mots machine. Pour effectuer ce calcul, la formule suivante est bien entendu valable (où $n = 64$, et a_i ou b_i nuls si $i < 0$ ou $i \geq n$) :

$$a(t)b(t) = \sum_{k=0}^{2n-2} \sum_{i=0}^{k} a_i b_{k-i}.$$

Cette même formule vaut aussi pour les entiers, mais le détail de taille tient bien entendu à la caractéristique. Sur $\mathbb{F}_2$, la somme $\sum_{i=0}^{k} a_i b_{k-i}$ n'occupe qu'un unique bit, aucune retenue n'est possible. Légitimement, on est donc enclin à penser que multiplier des polynômes sur $\mathbb{F}_2$ est plus facile que de multiplier des entiers. C'est vrai en matériel, mais moins vrai en logiciel. En effet, alors que les microprocesseurs disposent d'instructions permettant d'effectuer la multiplication de mots machine représentant des nombres entiers, une telle instruction (pour $\mathbb{F}_2$) a seulement fait son apparition pour la première fois sur les microprocesseurs grand public avec la microarchitecture Intel Westmere. Nous discutons en premier lieu comment il est possible de se passer d'une telle instruction, ce qui implique de recourir à des fonctions spécialisées pour effectuer cette opération élémentaire.

Le schéma de programme suivant pour effectuer ce calcul est proposé par [98], le paramètre s étant à fixer par l'implémentation ($s = 4$ est un choix initial raisonnable).

- Précalculer les multiples de $b(t)$ par tous les polynômes de degré strictement inférieur à s (soit une table de 2^s multiples à précalculer). Ces multiples peuvent être calculés au moyen d'un code de Gray.
- Soit $a(t) = \sum_k \alpha_k(t) t^{ks}$, où $\alpha_k(t)$ est un polynôme de degré au plus $s - 1$. Pour chaque bloc $\alpha_k(t)$, calculer $\alpha_k(t)b(t)$ au moyen de la table précédemment obtenue, et accumuler $\alpha_k(t)b(t)t^{ks}$ au résultat.

Deux améliorations de cette approche sont proposées dans [T10]. Un premier point, remarqué initialement dans la version 5.4.1 de la bibliothèque NTL, consiste à se satisfaire de 64 bits pour stocker les produits constitués par la première étape de précalcul (leur degré peut atteindre $63+s-1$). Pour ce faire, une étape de « réparation » permet à peu de frais de « récupérer » les bits perdus à cause des débordements au-delà des 64 bits. Ce mécanisme peut être implémenté sans opérations conditionnelles, comme indiqué dans [T10]. D'autre part, il est possible pour le choix de s, d'offrir un degré de liberté supplémentaire. Le choix de s est sujet à un compromis. Il est possible, avec peu de modifications du code, d'émuler le comportement d'une table correspondant au paramètre $2s$ avec seulement une table correspondant au paramètre s, au prix de deux accès à la table (pour « récupérer » le produit $x(t)b(t)$, où $\deg x < 2s$, on accède d'une part à $(x(t) \operatorname{div} t^s)b(t)$, et d'autre

part à $(x(t) \bmod t^s)b(t)$). Les longueurs respectives des différentes parties du code sont différentes, et cette approche s'avère rentable sur certains processeurs.

Sur un microprocesseur Intel Core2, une implantation des idées pour cette routine `mul1` permet d'obtenir un temps inférieur à 80 cycles pour une multiplication[2].

Une version `mul2` En employant des instructions SIMD (*single instruction multiple data*), telles celles du jeu d'instructions SSE2 présent sur les microprocesseurs Intel Pentium 3 et ultérieurs, il est possible de reprendre l'approche définie plus haut pour en dériver une opération que nous dénommons `mul2`, réalisant la multiplication de deux polynômes de degré au plus 127. Pour cela, la bibliothèque `gf2x` s'appuie non pas sur du code assembleur, mais sur l'interface de programmation en langage `C` définie pour ces fonctions, et supportée par les compilateurs Intel, Gnu, et Microsoft (le ficher d'en-tête `emmintrin.h` offre l'accès à ces instructions appelées *intrinsics*).

Ces techniques permettent, sur un microprocesseur Intel Core2, d'effectuer une opération `mul2` en 130 cycles environ[3].

Utilisation du support matériel

L'instruction `pclmulqdq`, introduite par la microarchitecture Intel Westmere, est la première apparition sur les microprocesseurs grand public d'une opération de multiplication de polynômes sur $\mathbb{F}_2$. À compter de sa version `1.0`, la bibliothèque `gf2x` supporte et utilise cette instruction. Le niveau de performances n'est pas aussi important qu'attendu, puisque l'instruction `pclmulqdq` est considérablement plus lente qu'une multiplication entière[4]. Toutefois, les performances permettent un gain substantiel par rapport à l'état précédent. Il est à espérer que les évolutions futures des microprocesseurs corrigeront l'anomalie que constitue la lenteur relative de cette instruction, améliorant ainsi encore les performances. Toutefois, aucune certitude n'est de mise quant à l'anticipation de telles évolutions.

16.2.2 Extensions : `mul2` à `mul9`

Dans un contexte comme la multiplication de polynômes sur $\mathbb{F}_2$, où l'opération de multiplication, même la plus élémentaire, est particulièrement coûteuse, et où l'addition (le ou exclusif) est pratiquement gratuit, où en outre il n'y a pas de retenue possible, il est naturellement très efficace d'utiliser l'algorithme de Karatsuba dès les petites tailles, comme deux mots. Nous avons vu plus haut que dans un contexte particulier, les instructions du microprocesseur offraient une méthode pour implémenter une routine `mul2` d'un coût comparable au double de `mul1`. Dans ce cas, faire reposer `mul2` sur `mul1` et Karatsuba n'est pas pertinent. En revanche, dans le cas où l'instruction `pclmulqdq` existe, c'est avantageux.

Pour des tailles allant par exemple jusqu'à 9 mots (limite fixée arbitrairement), il est possible d'essayer ainsi diverses stratégies. Outre l'algorithme de Karatsuba, diverses variantes autour du

[2]Le code correspondant dans `gf2x` est en `C`, et s'avère extrêmement sensible au compilateur. Les meilleurs temps rapportés par [T10] (moins de 60 cycles) utilisaient `gcc-4.1`, et il semble difficile de les reproduire. Le temps de 80 cycles est obtenu avec `gcc-4.6` sur un processeur Intel Core i5-2500, avec `gf2x-1.0`.

[3]La remarque précédente concernant `mul1` s'applique encore plus manifestement à `mul2`, où l'implantation en `C` s'appuie sur la gestion par le compilateur des *intrinsics*, qui est pour le moins erratique. Ce temps de 130 cycles peut être aujourd'hui reproduit sur un Intel Core i5-2500, avec `gcc-4.3` (pas les versions suivantes).

[4]L'ordre de grandeur du coût de l'instruction `pclmulqdq` est une latence de 14 cycles, et un débit d'un calcul tous les 8 cycles, pour un processeur Intel Core i5 [74]. À titre de comparaison, pour une multiplication entière ces valeurs sont de 3 et 2 cycles.

principe général d'évaluation-interpolation peuvent être utilisées (ceci inclut l'algorithme de Toom et ses variantes). La bibliothèque gf2x inclut un mécanisme mettant en compétition les différentes implémentations existant pour ces routines, et choisissant la meilleure pour un processeur donné.

16.3 Tailles moyennes

Divers algorithmes peuvent être utilisés pour mener la multiplication de polynômes occupant plusieurs mots machine (pour fixer l'ordre de grandeur, de plusieurs dizaines à plusieurs milliers), en s'appuyant sur des multiplications plus petites. Lorsque ces multiplications de plus petits objets sont quand même non négligeables, il n'est pas utile de pousser l'implémentation spécifique trop loin, et un code capable de gérer toutes les tailles convient. La bibliothèque gf2x implémente ainsi divers algorithmes.

- L'algorithme de Karatsuba.

- L'algorithme de Toom-Cook, qui multiplie deux polynômes en les considérant comme polynômes de degré 2 en la variable $x = t^k$, pour k choisi de sorte à égaler les degrés. Pour constituer les cinq coefficients du résultat (de degré 4 en x), il faut cinq points d'évaluation. Le choix classique des points d'évaluation dans $\mathbb{P}^1(\mathbb{F}_2)$ n'offre que trois choix. Toutefois, il est possible d'utiliser des points dans $\mathbb{P}^1(\mathbb{F}_2[t])$, sans recourir à une extension de corps : les points d'évaluation $(0, 1, \infty, t, 1/t)$ par exemple conviennent. Ceci a été étudié par Quercia, et généralisé plus tard par Bodrato [28], qui montre en particulier que le choix $(0, 1, \infty, t, t + 1)$ est légèrement préférable.

- Nous avons en outre ajouté une variante de l'algorithme précédent, où les points d'évaluation choisis sont $(0, 1, \infty, t^w, t^{-w})$, où w désigne la longueur des mots machine (pour notre cas d'étude, $w = 64$). Ce choix présente l'avantage de limiter les décalages à des nombres entiers de mots machine. La contrepartie est une augmentation du degré des valeurs d'évaluation.

- D'autres variantes sont possibles et testées. Par exemple l'algorithme de Toom se généralise à des divisions des entrées en un nombre différent de parties. Ainsi, un algorithme Toom-4 est aussi employé dans gf2x.

Enfin, un dernier choix possible est celui d'algorithmes intrinsèquement « déséquilibrés ». Il est possible, avec le même mécanisme d'évaluation-interpolation qui permet d'interpréter les formules de Karatsuba et Toom-Cook, d'obtenir des formules pour la multiplication de polynômes de degré 2 et de degré 1 (toujours en une variable intermédiaire $x = t^k$). Ceci est particulièrement utile pour fournir une routine de multiplication efficace pour des opérandes de tailles distinctes.

Une part cruciale du travail d'implantation pour ces tailles moyennes est le *tuning*, qui détermine, pour un degré donné, l'algorithme le plus efficace entre les différents choix possibles que nous venons de lister. La bibliothèque gf2x fait ce choix. Un exemple de résultat obtenu est donné par la table 16.1. Dans la table 16.1, pour chaque algorithme la plage des tailles entre 10 et 2000 mots pour lesquelles il s'avère optimal est mentionnée. Un algorithme donné n'est pas nécessairement optimal sur la totalité de cette plage. Aussi les plages se recouvrent-elles, et chacune indique le pourcentage des tailles incluses pour lesquelles l'algorithme correspondant est effectivement optimal.

Algorithme	Karatsuba	TC3	TC3W	TC4
Plage de tailles	10-65 (50%)	21-1749 (5%)	18-1760 (45%)	166-2000 (59%)

TAB. 16.1 : Plages où sont utilisés les algorithmes de taille moyenne de `gf2x`.

16.4 Grandes tailles

Au-delà de 2000 mots, il est pertinent de s'intéresser aux algorithmes asymptotiquement rapides. Nous donnons quelques considérations sur les approches possibles.

16.4.1 La substitution de Kronecker

Nous mentionnons en premier lieu la substitution de Kronecker, bien qu'il s'agisse d'une stratégie sous-optimale par rapport aux autres. Soient $a(t)$ et $b(t)$ deux polynômes sur $\mathbb{F}_p$ de degré n. Leur produit s'écrit :

$$a(t)b(t) = \sum_{m=0}^{n} \underbrace{\left(\sum_{i=0}^{m} a_i b_{m-i} \right)}_{c_m} t^m.$$

Soit maintenant $(\tilde{a}_i)_i$ et $(\tilde{b}_i)_i$ des relèvements des coefficients a_i et b_i dans $[\![0 \dots p-1]\!]$, et soit K un entier. Considérons les deux entiers suivants, et leur produit :

$$\tilde{A} = \sum_{i=0}^{n} \tilde{a}_i K^i \quad \text{et} \quad \tilde{B} = \sum_{i=0}^{n} \tilde{b}_i K^i,$$
$$\tilde{A}\tilde{B} = \sum_{m=0}^{n} \underbrace{\left(\sum_{i=0}^{m} \tilde{a}_i \tilde{b}_{m-i} \right)}_{\tilde{c}_m} K^m.$$

Chaque terme $\tilde{c}_m$ ci-dessus est un relèvement du coefficient c_m intervenant dans le produit de polynômes. S'il est possible, à partir du produit d'entiers $\tilde{A}\tilde{B}$, de reconnaître sans ambiguïté les coefficients $\tilde{c}_m$, on en déduit par conséquence les c_m. Ceci nécessite de fixer $K > n(p-1)^2$. C'est le principe de la substitution de Kronecker. Dès lors, pour multiplier les deux polynômes donnés en entrées, le coût est :

$$\mathsf{M}(n \times (2 \log p + \log n)),$$

ce qui dans le cas précis $p = 2$, se traduit par $\mathsf{M}(n \log n)$. Lorsqu'une implémentation efficace de la multiplication d'entiers est disponible, il est donc possible de l'utiliser pour multiplier des polynômes binaires.

16.4.2 La FFT ternaire

Une FFT « native » adaptée au cas de la caractéristique 2 se heurte naturellement à l'écueil suivant. Il n'est pas sage d'aller « chercher » des racines 2^k-èmes de l'unité en caractéristique 2, elles ne serviront pas à grand-chose.

L'algorithme de Schönhage [189] reprend l'esprit de l'algorithme de Schönhage-Strassen [190], mais force un point de vue visant à faire apparaître des racines 3^k-èmes de l'unité. Une formulation du cœur de l'algorithme de Schönhage est donnée par la proposition suivante (une généralisation à d'autres corps que $\mathbb{F}_2$ est donnée par [84, exercice 8.30]).

Proposition 16.2. *Soit $k \in \mathbb{N}^*$ et K, L, N vérifiant $K = 3^k,\ L \geq N/K,\ K \mid N,\ K \mid L$. On pose $A = \mathbb{F}_2[t]/t^{2L} + t^L + 1$. Le produit de deux polynômes modulo $t^{2N} + t^N + 1$ peut être calculé en $\widetilde{O}(K)$ additions et $2K$ multiplications dans l'anneau A*

Démonstration. Nous donnons quelques éléments brefs, la structure de l'algorithme étant assez proche de Schönhage-Strassen, et à ce titre classique. Nommons f et g les polynômes à multiplier. On récrit ces polynômes sous forme bivariée en faisant intervenir des segments de $L' = N/K$ coefficients. Ceci revient à poser $T = t^{N/K}$, et définir deux polynômes f' et g' tels que $f(t) = f'(t, T)$ et $g(t) = g'(t, T)$. Les deux polynômes f' et g' sont interprétés comme des polynômes en T à coefficients dans A, les coefficients étant de degré au plus $L' = N/K$. S'il est possible de calculer le produit de ces polynômes modulo $T^{2K} + T^K + 1$, alors l'objectif est atteint.

Pour ce faire, le produit est calculé en deux étapes distinctes. L'élément $\omega = t^{L/K}$ de A est une racine $3K$-ème de l'unité, d'où $w^{2K} + w^K + 1 = 0$. Dès lors, les deux polynômes $P_{1,2}$ ci-dessous sont premiers entre eux, et leurs racines sont données par les ensembles $\mathcal{S}_{1,2}$ comme suit.

$$\begin{aligned} P_1 &= T^K - w^K \in A[T], \\ P_2 &= T^K - w^{2K} \in A[T], \\ \mathcal{S}_1 &= \{x \in A, P_1(x) = 0\} = \{w, w^4, \ldots, w^{3K-2}\}, \\ \mathcal{S}_2 &= \{x \in A, P_2(x) = 0\} = \{w^2, w^5, \ldots, w^{3K-1}\}. \end{aligned}$$

L'algorithme de Schönhage recourt à la multi-évaluation de f' et g' sur les *deux* ensembles $\mathcal{S}_1$ et $\mathcal{S}_2$. Le calcul de ces « transformées », puisqu'on est réellement dans un contexte de transformée de Fourier, a un coût en $\widetilde{O}(K)$ additions dans A, selon un procédé exactement semblable à celui de la FFT, le « butterfly » étant remplacé par une opération à trois entrées et trois sorties. Les produits de convolution donnent lieu à $2K$ multiplications dans A. ■

Une variante, qui est implantée dans `gf2x`, consiste à calculer un produit modulo $t^{2N} + t^N + 1$ au moyen de $3K$ produits modulo $t^{2L} + t^L + 1$. Au prix de la limitation à un unique étage récursif, l'algorithme est simplifié.

L'implantation pratique de l'algorithme de Schönhage, telle que proposée dans `gf2x`, fait appel à une optimisation importante. Dans l'objectif d'éviter l'effet de « marches d'escalier » couramment observé pour la complexité d'algorithmes utilisant des transformées de Fourier, nous avons introduit une approche de division du calcul. Pour constituer un produit de degré N, on calcule deux produits, respectivement modulo $t^{N_1} + 1$ et $t^{N_2} + 1$, où les entiers N_1 et N_2 satisfont $N_1 > N_2 \geq N/2$. Cette approche est simple et efficace, et son apport est visible dans la figure 16.10 page 176, qui est discutée plus bas en 16.4.5.

16.4.3 L'algorithme de Cantor

L'algorithme de Cantor, dit de « FFT additive » est plus spécifique au contexte de la petite caractéristique. Cet algorithme est décrit dans [37, 82]. Le principe reste celui de la multiévaluation, mais en les racines d'un polynôme linéarisé (dont seuls les monômes de degré une puissance de deux sont affectés d'un coefficient non nul). On introduit la suite de polynômes linéarisés suivante :

$$s_0(t) = t \quad \text{et} \quad \forall i \geq 1,\ s_i(t) = s_{i-1}(t)^2 + s_{i-1}(t) = s_1(s_{i-1}(t)) = \underbrace{s_1 \circ \cdots \circ s_1}_{i}(t).$$

On vérifie immédiatement que $s_{2^k} = t^{2^{2^k}} + t$, et les racines de s_{2^k} sont les éléments du corps $\mathbb{F}_{2^{2^k}}$. En toute généralité, pour $0 \leq i \leq 2^k$, l'ensemble des racines de s_i est un sous-espace vectoriel

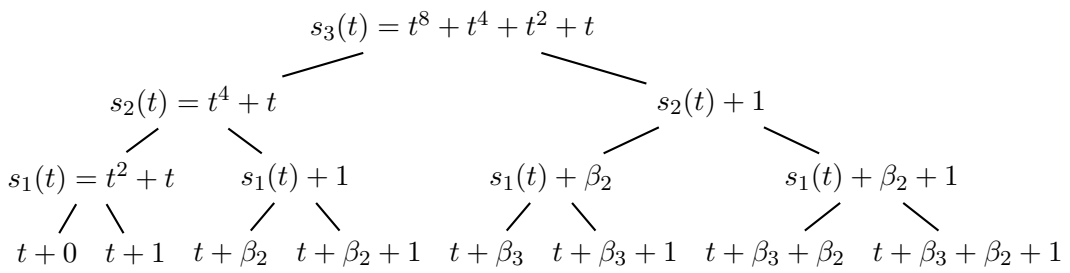


FIG. 16.3 : Arbre de sous-produits intervenant dans l'algorithme de Cantor.

de $\mathbb{F}_{2^{2^k}}$, qu'on note W_i (les W_i forment un drapeau). Pour évaluer un polynôme en les points de W_i, il suffit d'exploiter l'arbre de sous-produits associé. La structure de cet arbre de sous-produits peut être précalculée, en partant de la détermination de vecteurs de base de W_i associés au drapeau. On note ainsi $\beta_0 = 0$, $\beta_1 = 1$, et β_i tel que $s_i(\beta_i) = \beta_{i-1}$. Ainsi, W_i a pour base $(\beta_1, \ldots, \beta_i)$, et l'arbre de sous-produits associé à s_i s'écrit comme donné par la figure 16.3. En exploitant cet arbre de sous-produits, le coût de la multiévaluation sur l'ensemble W_i est donné par la proposition suivante.

PROPOSITION 16.4. *Le coût de l'algorithme de Cantor pour évaluer $f \in \mathbb{F}_{2^{2^k}}[t]$ en les points de $W_i \subset \mathbb{F}_{2^{2^k}}$ (où $0 \leq i \leq 2^k$) est de M_i multiplications et A_i additions dans $\mathbb{F}_{2^{2^k}}$, où*

$$M_i = O\left(i2^i\right), \quad \textit{et} \quad A_i = O\left(i^{\log_2 3}2^i\right).$$

DÉMONSTRATION. Supposons calculé un polynôme f modulo $s_j(t) + \gamma'$. Notons $(c_0, \ldots, c_{2^j-1})$ ses coefficients. Nous souhaitons déterminer le coût de sa réduction modulo $s_{j-1}(t)+\gamma$ et $s_{j-1}(t)+\gamma+1$. Remarquons d'abord que si $f = bq + r$, alors $f = b(q + 1) + (r + q)$. Aussi, pour calculer les deux restes, il est suffisant de n'en calculer qu'un, et d'effectuer ensuite 2^{j-1} additions pour obtenir le second. Pour calculer quotient et reste modulo $s_{j-1}(t)+\gamma$, nous procédons coefficient par coefficient, en traitant d'abord les coefficients de tête. C'est l'objet de l'algorithme 16.5 (on exploite le fait que $s_{j-1}(t) + \gamma$ est unitaire). L'examen de cet algorithme indique que le coût d'une étape de réduction est de 2^{j-1} multiplications, et $2^{j-1} \times (1 + n_{j-1})$ additions, où n_i est le nombre de monômes non nuls du polynôme s_{j-1}. Si on note alors M_i et A_i le nombre de multiplications et d'additions requis pour l'arbre complet, on a :

$$M_i = 2M_{i-1} + 2^{i-1}, \quad \text{et} \quad A_i = 2A_{i-1} + 2^{i-1} \times (1 + n_{i-1}),$$
$$\text{d'où :} \ \frac{M_i}{2^i} = \frac{i}{2}, \quad \text{et} \quad \frac{A_i}{2^i} = \frac{1}{2} \times \sum_{j=0}^{i-1} n_j.$$

L'étude du nombre de coefficients non nuls des s_i donne $\sum_{j=0}^{i-1} n_j = O(i^{\log_2 3})$, d'où l'équivalent asymptotique annoncé. ■

Algorithme de multiplication Pour multiplier deux polynômes de $\mathbb{F}_2$, il suffit de les considérer en premier lieu comme des polynômes à coefficients dans $\mathbb{F}_{2^{2^k}}$ pour un certain k, les coefficients étant regroupés par blocs de longueur 2^{k-1}. Ainsi, un polynôme $f \in \mathbb{F}_2[t]$ est transformé en $f' \in \mathbb{F}_2[t][T]$, avec

$$f(t) = f'(t, t^{2^{k-1}}).$$

Algorithme `cantor_reduce1`

ENTRÉE : j entier, et $(c_0, \ldots, c_{2^j-1})$ coefficients d'un polynôme dans $\mathbb{F}_{2^{2^k}}[t]$.
$\gamma \in \mathbb{F}_{2^{2^k}}$.
SORTIE : $(c_0, \ldots, c_{2^{j-1}-1})$ coefficients du reste modulo $s_{j-1}(t) + \gamma$.
$(c_{2^{j-1}}, \ldots, c_{2^j-1})$ coefficients du reste modulo $s_{j-1}(t) + \gamma + 1$.

```
for(s = 2^j − 1 ; s >= 2^{j−1} ; s--) {
    c_{s−2^{j−1}} += c_s γ ;
    for(t = 0 ; t < j − 1 ; t++) {
        if ([t^{2^t}] s_{j−1} ≠ 0) c_{s−2^{j−1}} += c_s ;
    }
}
for(s = 0 ; s < 2^{j−1} ; s++) {
    c_{s+2^{j−1}} += c_s ;
}
```

ALGORITHME 16.5 : Une étape de réduction dans l'algorithme de Cantor.

Lorsque le produit de f' et g' est de degré au plus 2^i, pour $0 \leq i \leq 2^k$, c'est-à-dire lorsque le produit fg a pour degré au plus $2^{k-1}2^i$, la multiplication de polynômes dans $\mathbb{F}_{2^{2^k}}[T]$ par multi-évaluation sur W_i permet de calculer le produit fg. Des petites valeurs de k sont donc suffisantes pour traiter des entrées de taille conséquente. Les valeurs de 4 à 7 conviennent pour calculer des produits occupant respectivement 64 Ko, 8 Go, 2^{36}Go, 2^{104}Go, soit des limites largement convenables pour la pratique. Pour calculer un produit occupant n bits, un choix en $k = \log\log n$ et $i = \log n$ convient. Dès lors, la complexité est en

$$\mathsf{M}(n) = O\left(n(\log n)^{\log_2 3}\right).$$

On remarque que dans la complexité précédente, ce sont asymptotiquement les *additions* qui dominent. Toutefois, la différence entre $(\log n)^{\log_2 3}$ et $\log n$ n'efface pas le fait que le coût d'une multiplication dans $\mathbb{F}_{2^{2^k}}$ est nettement supérieur à celui d'une addition. Cet aspect asymptotique est donc trompeur. La composante « visible » en pratique est bel et bien donnée par les multiplications. L'apparition de l'instruction `pclmulqdq` change peu la donne. Bien que le coût relatif des multiplications et des additions en soit modifié, le surcoût des multiplications reste très conséquent.

Choix du corps de base Le choix du corps de base est en fait assez libre. Choisir la plus petite valeur convenable pour k, comme fait dans [82], n'est pas forcément optimal. En effet, il convient de mesurer le coût et le nombre des opérations (principalement les multiplications) dans les différents cas possible. La table 16.6 donne ces informations pour les divers choix possibles pour multiplier deux polynômes dont le produit occupe 2^N bits, pour les deux valeurs $N = 14$ et $N = 19$. Cette table livre deux enseignements :

- Les multiplications dominent le coût. En effet, pour ces tailles, le ratio entre A_i et M_i est bien en deçà du ratio des coûts des opérations associés.

- Le choix d'un corps de base tel que $\mathbb{F}_{2^{128}}$ est avantageux. Ceci est dû au fait que la multiplication dans $\mathbb{F}_{2^k}$ a un coût *grosso modo* sous-linéaire en k au moins jusqu'à cette valeur, grâce à l'emploi d'instructions SIMD.

			$N = 16\,384$				$N = 524\,288$			
k	2^k	m_k (cycles)	i	M_i	A_i	$M_i m_k$	i	M_i	A_i	$M_i m_k$
4	16	32	11	10 240	26 624	327 680	16	491 520	2 129 920	15 728 640
5	32	40	10	4 608	11 776	184 320	15	229 376	819 200	9 175 040
6	64	77	9	2 048	5 120	157 696	14	106 496	352 256	8 200 192
7	128	157	8	896	2 432	**140 672**	13	49 152	147 456	**7 716 864**

TAB. 16.6 : Choix du corps de base pour l'algorithme de Cantor (exemple, processeur type Intel Core2).

La table 16.6 ne prend pas en compte les améliorations de `gf2x` consécutives à l'introduction de l'instruction `pclmulqdq`. Dans la table 16.6, on a $m_7/m_6 \approx 2.03$, tandis qu'avec `pclmulqdq`, on obtient plutôt $m_7/m_6 \approx 3$. La sous-linéarité de la multiplication dans ce dernier cas s'arrête donc à $\mathbb{F}_{2^{64}}$, et le choix $k = 6$ semble optimal. Plus précisément, étant donné que $M_6/M_7 \approx 2.16$ (pour l'exemple avec 2^{19} bits), une implantation de l'algorithme de Cantor utilisant `pclmulqdq` et $\mathbb{F}_{2^{64}}$ comme corps de base pourrait, sur le papier, obtenir un facteur d'accélération proche de 30% comparé au corps de base $\mathbb{F}_{2^{128}}$. Ces estimations rapides ne semblent pas confirmées par la pratique, mais ce travail pourrait être poursuivi.

Troncature Pour rendre plus efficace l'algorithme de Cantor, il est important de tenter de combattre l'effet « marches d'escalier » qui semble inhérent à cette méthode, comme à d'autres méthodes employant des transformées de Fourier : il semble que le nombre de points d'évaluation est contraint à être une puissance de deux, ce qui cause a priori des paliers dans le temps de calcul. Pour pallier ce problème, une approche semblable à celle de van der Hoeven [106] a été utilisée dans [T10]. Le principe est très simple, au moins pour l'évaluation. Plutôt que d'évaluer en 2^i points, on évalue en un nombre plus restreint $n < 2^i$. Ceci permet d'économiser des calculs d'évaluation. Pour l'interpolation, une étape de réduction modulo le polynôme annulant les points d'interpolation choisis est nécessaire. En choisissant bien ces points, il est possible d'exploiter les s_i à nouveau, et d'écrire une telle réduction de manière efficace. Les détails apparaissent dans [T10]. L'efficacité de cette approche de troncature peut être estimée au moyen de la figure 16.10 page 176.

16.4.4 L'algorithme de Gao-Mateer

Une amélioration de l'algorithme de Cantor a été proposée par Gao et Mateer dans [81]. Cette approche a été commentée par Bernstein[5], et sa complexité asymptotique est séduisante. Ces éléments n'apparaissant pas dans [T10], nous prenons le temps d'en indiquer le principe.

Soit n une puissance de deux, et $q = 2^n$. Les polynômes s_n et s_{2n} déjà rencontrés plus haut sont :

$$s_n = t^{2^n} + t = t^q + t \quad \text{et} \quad s_{2n} = t^{2^{2n}} + t = t^{q^2} + t.$$

L'algorithme de Gao-Mateer est une réduction spéciale, insérée dans l'algorithme de Cantor, qui permet de descendre efficacement les étages de l'arbre de sous-produits de la figure 16.3. En effet, cet algorithme donne une procédure spécifique pour passer de l'étage $2n$ à l'étage n. Considérons un nœud à l'étage $2n$ dans cet arbre, c'est-à-dire un polynôme $f(t)$ défini modulo $s_{2n}(t)+\beta$. L'élément β a des antécédents par $s_n(t)$, comme dans le schéma classique de l'algorithme de Cantor. Aussi nous

[5] `http://cr.yp.to/f2mult.html`.

Algorithme gao_mateer_taylor_expand

ENTRÉE : n et m entiers (n une puissance de 2, et $m \leq n$). On note $q = 2^n$, $p = 2^m$
$(f_0, \ldots, f_{qp-1})$ coefficients de $f(t) \in \mathbb{F}_{2^{2^k}}[t]$.
SORTIE : $(f_0^*, \ldots, f_{p-1}^*)$ polynômes de degré $q-1$ satisfaisant $f(t) = \sum_{i=0}^{p-1} (t^q + t)^i f_i^*(t)$.

`/* Calculer simultanément quotient et reste modulo` $(t^q + t)^{p/2} = t^{qp/2} + t^{p/2}$ `*/`
`for(i =` $qp - 1$ `; i >=` $qp/2$ `; i--)` $f_{\mathtt{i}+p/2-qp/2}$ `+=` $f_{\mathtt{I}}$ `;`
`gao_mateer_taylor_expand(`n`,`$m-1$`,`$(f_0, \ldots, f_{qp/2-1})$`) ;`
`gao_mateer_taylor_expand(`n`,`$m-1$`,`$(f_{qp/2}, \ldots, f_{qp-1})$`) ;`

ALGORITHME 16.9 : Décomposition en base $t^q + t$ dans l'algorithme de Gao-Mateer.

écrivons $\beta = s_n(\gamma) = \gamma^q + \gamma$, pour un certain γ. Le polynôme $s_{2n}(t) + \beta$ vérifie :

$$\begin{aligned} s_{2n}(t) + \beta &= s_n(s_n(t)) + s_n(\gamma) = s_n(s_n(t) + \gamma), \\ &= \prod_{\alpha \in \mathbb{F}_{2^m}} (s_n(t) + \gamma + \alpha) \quad (\text{où } \mathbb{F}_{2^n} = \mathbb{F}_q). \end{aligned}$$

La réduction de Gao-Mateer vise à calculer simultanément les $q = 2^n$ nœuds de l'étage n. Les polynômes attachés à ces nœuds sont les

$$\{T_n(f, \omega) = f(t) \bmod s_n(t) + \omega,\ \omega \in \gamma + \mathbb{F}_q\}. \tag{16.7}$$

Pour les calculer polynômes, la première étape est l'écriture de $f(t)$ en base $s_n(t) = t^q + t$. On définit ainsi q polynômes de degré au plus $q-1$ tels que :

$$f(t) = f_0^*(t) + (t^q + t) f_1^*(t) + \cdots + (t^q + t)^{q-1} f_{q-1}^*(t). \tag{16.8}$$

Une approche asymptotiquement rapide complètement classique convient, et s'adapte très bien au contexte de la caractéristique 2 comme le montre l'algorithme 16.9.

La suite de l'évaluation consiste à observer l'équation 16.7 donnant les valeurs qu'on souhaite calculer, et l'équation 16.8 définissant les f_i^*. On remarque alors :

$$T_n(f, \omega) = f(t) \bmod t^q + t + \omega = f_0^*(t) + (\omega) f_1^*(t) + \cdots + (\omega)^{q-1} f_{q-1}^*(t).$$

Soit maintenant :

$$h_j(t) = \sum_{i=0}^{q-1} \left([t^j] f_i^*(t) \right) t^i = \sum_{i=0}^{q-1} f_{i,j}^* t^i.$$

Une façon commode de voir les polynômes h_j est de considérer la matrice F^* dont les lignes sont les coefficients des f_i^*. Alors les polynômes h_j ont pour coefficients les colonnes de cette même matrice. Alternativement, et pour prendre un point de vue plus proche de l'implantation, les h_j s'obtiennent à partir des f_i^* par une transposition de matrice de taille $q \times q$. L'introduction des polynômes h_j permet d'écrire :

$$T_n(f, \omega) = f(t) \bmod t^q + t + \omega = \sum_{j=0}^{q-1} h_j(\omega) t^j.$$

Dès lors, les q polynômes cible à calculer s'obtiennent en calculant q multiévaluations des h_j en les points $\{\omega \in \gamma + \mathbb{F}_q\}$, comme nous le voyons maintenant. La matrice F^* définie plus haut peut

s'écrire :

$$F^* = \begin{pmatrix} \longleftarrow f_0^* \longrightarrow \\ \longleftarrow f_1^* \longrightarrow \\ \vdots \\ \longleftarrow f_{q-1}^* \longrightarrow \end{pmatrix} = \begin{pmatrix} \uparrow & \uparrow & & \uparrow \\ h_0 & h_1 & \cdots & h_{q-1} \\ \downarrow & \downarrow & & \downarrow \end{pmatrix}.$$

La multiévaluation des h_j donne les vecteurs que nous notons : $\hat{h}_j = (h_j(\gamma + \alpha))_{\alpha \in \mathbb{F}_q}$. Par commodité, on note $\{\omega_0, \ldots, w_{q-1}\}$ l'ensemble $\gamma + \mathbb{F}_q$.

$$F^* \rightsquigarrow \begin{pmatrix} \uparrow & \uparrow & & \uparrow \\ \hat{h}_0 & \hat{h}_1 & \cdots & \hat{h}_{q-1} \\ \downarrow & \downarrow & & \downarrow \end{pmatrix} = \begin{pmatrix} \longleftarrow T_n(f, \omega_0) \longrightarrow \\ \longleftarrow T_n(f, w_1) \longrightarrow \\ \vdots \\ \leftarrow T_n(f, w_{q-1}) \rightarrow \end{pmatrix}$$

Une étape de réduction de l'étage $2n$ (n étant une puissance de deux) à l'étage n nécessite donc le calcul de $q = 2^n$ transformées afin de calculer les $T_n(f, \omega)$, puis à nouveau du même nombre d'appels récursifs pour calculer effectivement chacune des multiévaluations des $T_n(f, \omega)$ qui correspondent à l'étage n. Pour calculer la multiévaluation d'un polynôme de degré au plus $q^2 = 2^{2n}$ en q^2 points, le coût est donc de :

- Une réduction de Gao-Mateer vers des polynômes de degré $q = 2^n$, coûtant :
  - Un développement en base $t^q + t$, qui nécessite $\log_2 q = n$ étapes récursives, pour un total de $q^2 n$ additions.
  - q calculs de multiévaluations de polynômes de degré q en q points.
- Pour les appels récursifs, à nouveau q calculs de multiévaluations de polynômes de degré q en q points.

In fine, les nombres de multiplications et d'additions pour calculer la transformée d'un polynôme de degré $q^2 = 2^{2n}$ s'écrivent :

$$M_{q^2} = 2 \times q \times M_q \approx q^2 n \approx \frac{1}{2} q^2 \log_2 q^2.$$
$$A_{q^2} = 2 \times q \times A_q + q^2 n \approx q^2 n \log_2 n \approx \frac{1}{2} q^2 \log_2 q^2 \log_2 \log_2 q^2.$$

La réduction de Gao-Mateer permet donc de gagner sur la complexité asymptotique par rapport à l'algorithme de Cantor. La réalité pratique est hélas différente, comme cela apparaît sur la figure 16.10. Des travaux préliminaires d'implantation n'ont pas permis d'obtenir un gain. Il est possible d'avancer plusieurs raisons. En premier lieu, le nombre de multiplications est inchangé avec la réduction de Gao-Mateer. L'échange d'additions contre l'ajout de structures de contrôle moins triviales (transposition notamment) n'est pas un gain immédiat, et qui plus est condamné à être peu visible. D'autre part, la réduction de l'algorithme de Gao-Mateer réduit un calcul à l'étage $2n$ à 2^{n+1} calculs à l'étage n. En comparaison, l'algorithme de Cantor a des calculs à mener à tous les nœuds entre ces deux

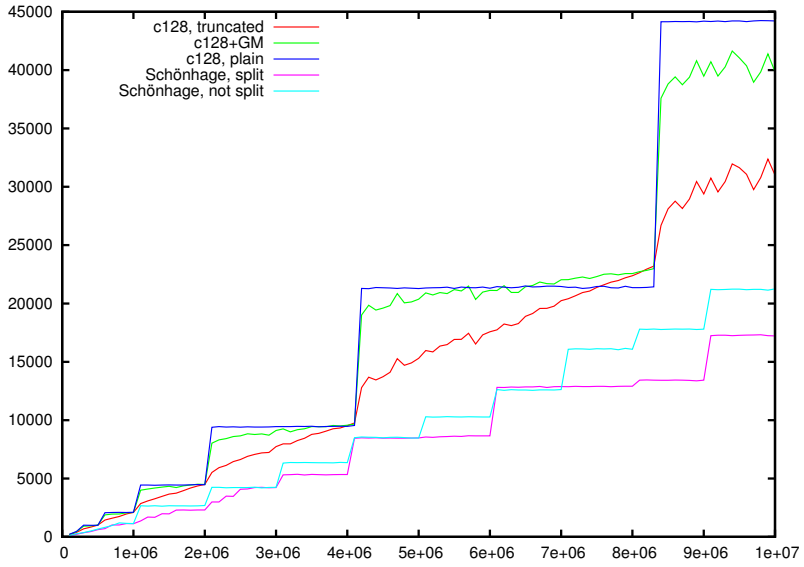


FIG. 16.10 : Temps mesuré pour les algorithmes asymptotiquement rapides dans `gf2x` (abscisses : nombre de mots de 64 bits, ordonnées : millisecondes – processeur Intel Core i5-2500, 3.3 GHz, gcc-4.6.1)

étages, à la suite desquels il se ramène à 2^n calculs à l'étage n. Aussi, le ratio entre les calculs aux étages 0 à n et n à $2n$ dans l'arbre de sous-produits guide-t-il l'intérêt du résultat. Il semble que le déséquilibre de coût entre ces parties haute et basse dans l'algorithme de Cantor, prévu par l'analyse, n'est pas assez grand en pratique pour rendre efficace la réduction de Gao-Mateer. Enfin, les « sauts » de plusieurs étages de l'algorithme de Gao-Mateer font qu'il est difficile d'appliquer la méthode de troncature évoquée plus haut dans l'algorithme de Cantor afin d'obtenir une courbe de temps de calcul plus uniforme. Il est vraisemblable qu'un approfondissement de ces travaux sur l'algorithme de Gao-Mateer permette d'aboutir à des résultats meilleurs, mais ce n'est pas le cas pour l'instant.

16.4.5 Comparaison des différents algorithmes

Nous avons implanté, dans le cadre de la bibliothèque `gf2x` les différents algorithmes indiqués dans cette section[6]. Malgré l'originalité de l'algorithme de Cantor, celui-ci n'apparaît pas meilleur que l'algorithme de Schönhage, au vu des temps indiqués par la figure 16.10. Sur la figure 16.10, outre l'implantation de référence que nous avons établie dans `gf2x` pour l'algorithme de Schönhage (courbe « Schönhage, split ») et pour l'algorithme de Cantor (courbe « c128, truncated ») apparaissent la version sans séparation en deux de l'algorithme de Schönhage, la version non tronquée de l'algorithme de Cantor, ainsi qu'une tentative d'implantation de la réduction de Gao-Mateer au sein de l'algorithme de Cantor. Ces temps sont donnés sur un processeur Intel Core i5-2500 à 3.3 GHz, en utilisant l'instruction `pclmulqdq`.

Il importe de remarquer un point de détail particulièrement important. Les coûts relatifs des

[6]La version *actuelle* de `gf2x` ne contient plus le code pour l'algorithme de Cantor. Celui-ci réapparaîtra dans une autre bibliothèque concentrée sur les différents algorithmes asymptotiquement rapides.

transformées et des convolutions dans les différents algorithmes évoqués (principalement Schönhage et Cantor, donc) sont sans aucun point de comparaison. Pour la multiplication de polynômes d'un million de mots, nous comparons les deux approches sur un processeur Intel Core i5-2500 à 3.3 GHz.

- L'algorithme de Cantor prend 2.1 secondes, dont environ 1.5% (0.03 secondes) dans les produits de convolution point à point.
- L'algorithme de Schönhage, tel qu'implanté dans `gf2x`, prend 1.1 seconde, mais 44% de ce temps est passé dans les produits point à point. Ceci nécessite une petite nuance. L'algorithme de Schönhage se ramène à des produits dans $\mathbb{F}_2[t]/t^{2L} + t^L + 1$ qui sont effectués au moyen de la routine de multiplication de `gf2x`, c'est-à-dire a priori au moyen de l'algorithme de Toom-Cook. Le temps que nous mesurons a pour granularité celle de la séparation que nous avons menée dans l'implémentation : bien que l'algorithme de Toom-Cook comme l'algorithme de Karatsuba peuvent s'incorporer dans un schéma « évaluation-interpolation », nous n'avons pas développé cette interface, aussi notre coût pour les produits point à point considère-t-il Toom-Cook comme un boîte noire, négligeant son aspect « évaluation-interpolation ». Il n'en reste pas moins que la complexité des produits point à point est inhérente au contexte ici, puisque le nombre de points d'évaluation utilisés pour ramener l'algorithme de Toom-Cook à un schéma « évaluation-interpolation » est égal à la complexité de ce même algorithme, et pas linéaire comme dans le cas des algorithmes utilisant des transformées de Fourier.

Au vu de ces données, un contexte applicatif dans lequel les transformées peuvent être employées plusieurs fois donne un coût global bien différent. Ainsi, par exemple, le coût d'une multiplication de deux matrices $n \times n$ est :

$$\begin{aligned} C_{n\times n} &= 2n^2 c_{\text{DFT}} + n^2 c_{\text{IFT}} + n^3 c_{\text{CONV}}, \\ &= n \times \left(2n c_{\text{DFT}} + n c_{\text{IFT}} + n^2 c_{\text{CONV}}\right), \end{aligned}$$

ce qui donne un gain dès la multiplication de matrices 4×4 (cette estimation peut aussi être menée en prenant en compte l'algorithme de Strassen, pour un résultat semblable).

Nous ne connaissons pas d'application « ultime » où des multiplications de polynômes de grand degré sur $\mathbb{F}_2$ sont requises, avec comme caractéristique la possibilité de réutiliser les transformées. Bien entendu, le produit matriciel que nous indiquons a ces caractéristiques, mais il ne saurait constituer une application en soi. Dans l'algorithme NFS, l'étape centrale de l'algorithme de Wiedemann par blocs induit une multiplication de matrices dont les coefficients sont des polynômes sur $\mathbb{F}_2$. Hélas, les matrices sont dans ce contexte de taille non négligeable (256×256 à tout le moins), mais le degré des coefficients justifie à peine une asymptotique rapide : pour l'application RSA-768, au plus un million.

Ceci étant dit, les évolutions futures des microprocesseurs apporteront sans doute des améliorations au coût de l'instruction `pclmulqdq`. On peut en attendre un rapprochement des coûts des algorithmes de Cantor et Schönhage.

Partie V

Conclusion et projets futurs

Chapitre 17

Conclusion et projets futurs

Nous avons exposé dans ce mémoire une large part de nos travaux (exception faite de travaux se raccrochant peu aux travaux mis en valeur, comme [T6] ou [T11]). Les différentes parties de ce mémoire ont permis d'en délimiter les principaux périmètres, l'algorithme du crible algébrique occupant une place centrale, à la fois comme objectif en soi, comme méthodologie (pour le calcul de logarithmes discrets sur les courbes de grand genre, exposé au chapitre 10), ou comme motivation (pour les travaux d'algèbre linéaire dans leur ensemble, ou pour les travaux d'arithmétique sur $\mathbb{F}_2$). La figure 17.1 tente d'illustrer comment les différentes parties de notre travail intéragissent, même si à l'évidence ce type de schéma escamote la vraie richesse des interactions.

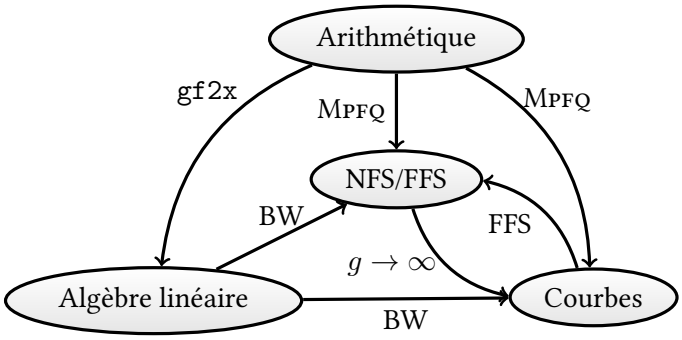


Fig. 17.1 : Liens entre différents aspects des travaux présentés.

Nous considérons que chacun des compartiments de notre travail possède une réelle cohérence, et qu'en outre l'interaction entre ces différents compartiments est riche. Dans nos travaux, chacun des thèmes que nous avons abordés a été examiné sous les angles à la fois théorique et pratique, et nous avons montré comment nous étions en mesure d'offrir des réponses sur plusieurs plans :

- Des réponses d'ordre pratique, passant par des implantations efficaces ont été proposées dans le cadre des travaux décrits en 6.3, 8.7, ainsi qu'aux chapitres 13, 15, et 16.
- Des réponses sous la forme d'algorithmes nouveaux ont été proposées aux chapitres 4, 5, 8, 9, et 10. Les travaux du chapitre 16 relèvent d'une double facette conjugant l'implantation efficace et la réflexion algorithmique.
- Des réponses plus théoriques ont été apportées concernant la formulation mathématique des algorithmes employés (par exemple au chapitre 3) ou la preuve de leur correction (au chapitre 9).

Avant d'aborder pleinement nos perspectives de recherche futures, nous proposons un bref préambule donnant quelques éléments essentiels d'impact du travail décrit dans ce mémoire, ainsi que la façon dont ces axes de travail sont actuellement des composants de nos travaux actuels. À la suite de

cette section 17.1, nous développons nos pistes de recherche futures, organisées pour la plus grande partie d'entre eux sous le chapitre de nos travaux sur l'algorithme NFS et ses algorithmes cousins.

17.1 Impact et poursuite des pistes de recherche actuelles ou passées

17.1.1 Courbes

Nos travaux sur les courbes ont couvert à la fois l'aspect constructif (via la bibliothèque MPFQ présentée au chapitre 15), et l'aspect de cryptanalyse, via les algorithmes de calcul de logarithmes discrets dans diverses classes de courbes, détaillés aux chapitres 8, 9, et 10.

Notre travail en collaboration avec P. Gaudry sur MPFQ a fait date, et depuis 2007 l'idée a été reprise et améliorée, entre autres par [192, 143, 8]. Nous constatons avec satisfaction que pour tenir la corde aujourd'hui en termes d'implantation logicielle, l'approche que nous avons promue est adoptée, en étudiant simultanément l'arithmétique des courbes et l'implémentation efficace des briques de base. Même s'il est probable que nous ne nous réinvestissions pas sur ce terrain pour apporter des améliorations incrémentales, il est évident que nous suivrons avec intérêt les développements sur cette piste de recherche.

Nous avons significativement contribué au problème du logarithme discret sur les courbes. Toutefois, depuis 2006 environ[1], nous considérons cette piste de recherche peu active. L'intérêt d'éventuelles avancées dans ce domaine serait important, mais également particulièrement difficile à anticiper.

Une direction de recherche active aujourd'hui, en lien avec le calcul du logarithme discret est celle visant au transport de ce problème vers une autre instance sur une courbe isogène, notamment via les travaux de Smith [200]. Nous n'envisageons pas de nous investir significativement dans cette voie directement, mais nous retrouvons là l'une des motivations d'une des pistes que nous développons plus loin en 17.3.1.

17.1.2 Algorithme NFS

Nous sommes entré, depuis quelques années, sur la scène internationale de la factorisation d'entiers, de deux façons. D'une part, par la bibliothèque CADO-NFS, qui a le mérite de fournir une implémentation complète de l'algorithme, ne s'appuyant pas sur des composants extérieurs. D'autre part, notre participation à l'effort de factorisation RSA-768 a indiqué notre capacité à nous frotter aux tailles établissant l'état de l'art actuel.

En termes d'impact, outre la visibilité conséquente de ce record, nous considérons que notre contribution personnelle a significativement pesé sur les perspectives futures de tels calculs, et sur la façon de les organiser. Un premier point, plus important qu'il n'y paraît, a trait à la méthode de gestion d'un calcul de cette ampleur. Nous sommes convaincu que l'ensemble des techniques que nous avons employées pour gérer les processus de crible pour RSA-768 ont démontré leur efficacité, particulièrement eu égard à la quantité de temps humain nécessaire comparée au temps machine mobilisé. Cette méthodologie, décrite dans [T16] ainsi que plus brièvement en 6.3.2, permet l'emploi de ressources comme des *grilles de calcul* en mode *best-effort*. Ceci n'est pas nouveau en soi, puisque déjà vingt ans plus tôt, l'idée d'utiliser des travaux de priorité inférieure pour le crible était de mise. Toutefois, nous jugeons que la méthode proposée est particulièrement bien adaptée aux environnements de grilles qui sont aujourd'hui disponibles en maints endroits. Il est raisonnable de penser que

[1] Malgré l'impression laissée par les dates de publication, les idées correspondant aux publications [T9] et [T15] étaient respectivement fixées dès 2005 et 2002.

cette méthodologie « industrielle », sans interdire à un contributeur éventuel de travailler « à la main », sera réemployée pour des calculs futurs.

17.1.3 Algèbre linéaire

Le second point par lequel nos travaux ont pesé sur les calculs de type NFS présents et futurs est la façon d'organiser l'étape d'algèbre linéaire. L'algorithme [T3] que nous avons développé il y a bien longtemps pour le calcul quasi-linéaire de générateurs matriciels a offert un point d'entrée pour l'algorithme de Wiedemann par blocs dans le contexte du crible algébrique. Avec RSA-768, nous avons pu démontrer la supériorité de cette approche sur celle employant l'algorithme de Lanczos par blocs. En effet, une compétition amicale a été lancée avec nos collaborateurs du CWI, dans le but de déterminer qui résoudrait le plus vite le système linéaire lié à RSA-768, l'équipe du CWI tentant d'employer l'algorithme de Lanczos par blocs sur le supercalculateur à leur disposition. Il ne leur a pas été possible de remporter ce mini-défi. On peut partir de ce constat pour juger de l'ampleur des difficultés, notamment logistiques, qu'induit l'algorithme de Lanczos par blocs. Ce dernier nécessite l'emploi d'un unique gros cluster, plus coûteux à requérir que plusieurs clusters de taille moyenne. Nous pensons que l'algorithme de Wiedemann par blocs continuera à être utilisé pour les factorisations record dans le futur.

Relativement toujours à l'organisation des calculs pour l'étape d'algèbre linéaire, l'emploi de *grilles de calcul* est un apport entièrement nouveau de nos travaux, et une originalité importante du calcul RSA-768. Nous considérons cette approche prometteuse, tout en y reconnaissant des points d'amélioration possible, sur lesquels nous entendons poursuivre notre travail.

17.1.4 Arithmétique

L'impact de la partie arithmétique de nos travaux a déjà été mentionnée pour le contexte des courbes, avec Mpfq. Il est tout à fait imaginable d'observer un impact de notre approche dans d'autres domaines où des calculs sur les corps finis représentent la part dominante du coût calculatoire. Ceci étant, pour faire cela intelligemment, il est nécessaire de réfléchir avec l'utilisateur potentiel sur les points à optimiser. Le cadre de travail de Mpfq est utile, mais il n'apporte pas de solution immédiate, en ce sens que l'identification de la fonction la plus importante pour une application en particulier n'est pas un élément qui se décrète. Nous supposons qu'un emploi de Mpfq par exemple pour des calculs en théorie des codes pourrait s'avérer utile, mais ceci n'est pour l'instant qu'une piste vague.

L'impact de nos travaux sur la bibliothèque `gf2x` se mesure avant tout par le fait que cette bibliothèque est montée rapidement en grade, devenant un composant qui peut remplacer la partie équivalente de la bibliothèque NTL, largement utilisée, et auparavant référence en la matière. Nous restons actifs sur `gf2x`, puisque l'introduction des instructions comme `pclmulqdq` est encore récente, et les inévitables évolutions à court terme feront bouger les lignes en matière d'optimisations relatives des différents algorithmes.

17.2 Crible algébrique : RSA-1024 en ligne de mire

Nous l'avons indiqué plus haut, notre positionnement comme acteur sur le thème de la factorisation et du logarithme discret a été bien installé avec le calcul RSA-768. Cette piste de travail mérite d'être poursuivie. Il y a là un travail de continuation, mais surtout de renouvellement. Nous ne souhaitons pas dépenser en continu une énergie importante à grignoter quelques bits sur les records. En effet, un record de calcul est rarement intéressant simplement parce qu'il a montré qu'en poussant

le travail précédent un peu plus loin, avec des ressources de calcul un peu plus puissantes, il est possible d'atteindre un record un peu plus grand. Nous sommes intéressé par les records qui amènent un changement de la méthodologie rendant possible ce qui échappe à une extrapolation à partir des travaux précédents. De telles avancées peuvent se placer à divers niveaux, qu'il s'agisse d'améliorations algorithmiques, d'optimisations d'implantation, ou encore d'optimisations organisationnelles rendant le calcul plus facilement « gérable ».

Dans cette perspective, nous continuerons notre investissement sur l'algorithme NFS pour la factorisation, et cet effort sera étendu au contexte de NFS-DL et FFS pour le logarithme discret sur les corps finis. Ce dernier thème est au cœur du projet ANR Catrel (2013-2016), que nous coordonnons. Peu de records sur le problème du logarithme discret ont été obtenus depuis 2005. Les choix actuels pour la cryptographie à base de couplages amènent à étudier de près l'hypothèse de difficulté du logarithme discret sur les corps finis, ce qui doit naturellement apporter une attention nouvelle sur ce thème de recherche.

Nous pouvons fixer des objectifs à moyen terme pour de tels travaux. Un objectif assez naturel, après RSA-768, est la factorisation de RSA-1024. Il s'agit là d'un objectif à 10 ans. Concernant le problème du logarithme discret sur les corps finis, atteindre la limite des 1024 bits peut apparaître tantôt facile, tantôt particulièrement ardu, en fonction du problème exact que nous considérons : un calcul de logarithmes discrets dans un corps comme $\mathbb{F}_{2^{956}}$, extension de degré 4 de $\mathbb{F}_{2^{239}}$, semble atteignable à court terme. La motivation apportée par le contexte d'usage des couplages sur les courbes elliptiques donne un intérêt à cet exemple particulier, puisque le choix d'une courbe supersingulière sur ce corps de base précis est fait par plusieurs travaux, notamment d'implantation. Le couplage η_T sur une telle courbe n'est pertinent pour une utilisation cryptographique que si le logarithme discret dans $\mathbb{F}_{2^{956}}$ est difficile. Ce corps constitue donc un second objectif, à assez court terme, de nos travaux. Enfin, la limite des 1024 bits pour des problèmes de logarithme discret plus difficiles, comme par exemple le calcul de logarithmes discrets dans $\mathbb{F}_p$ avec p un nombre premier de 1024 bits ou plus, est probablement un objectif à plus long terme, sur lequel nous n'avons pas nécessairement une visibilité excellente.

Nombre de pistes de travail peuvent être évoquées pour progresser vers ces objectifs. La conjonction de travaux dans l'ensemble de ces directions sera nécessaire pour atteindre ces derniers. Nous indiquons quelques-unes de ces pistes, en les présentant par nature des travaux correspondants. Le réceptacle naturel pour les implémentations pratiques de ces travaux, que nous envisageons dans la totalité des cas, est le logiciel Cado-nfs[2].

17.2.1 Optimisations algébriques et algorithmiques de NFS

Un algorithme efficace est d'abord un algorithme auquel on a bien réfléchi. Notre réflexion sur les différentes options existantes, ou envisageables, pour l'amélioration du crible algébrique (par exemple) est constante. Nous entendons poursuivre dans cette voie. Plusieurs points, aujourd'hui, donnent lieu à des travaux possibles à court, moyen ou plus long terme. Chacun de ces travaux est une piste possible pour apporter une amélioration, ponctuelle ou plus globale, sur l'ensemble de l'algorithme NFS.

Pour les points développés ci-dessous, nous savons assez clairement où nous allons, et ce que nous attendons comme retombées. D'une façon plus générale, c'est en poursuivant notre étude poussée de tous les recoins algorithmiques et algébriques de l'algorithme NFS que nous espérons, sous le présent chapitre, obtenir des résultats. Le fruit d'approches de ce type, jusqu'à ce jour, a ouvert

[2]Que celui-ci accueille à terme ou pas les développements consacrés au problème du logarithme discret, ces derniers seront rendus disponibles.

ainsi la voie aux travaux sur le problème RSA avec oracle [T8, T12], ou encore au calcul de logarithmes discrets sur les courbes de genre grand [T15]. Aussi, nous estimons une telle étude comme à la fois riche et enrichissante, de retombées pour des problèmes algorithmiques qui peuvent tout à fait échapper au strict contexte de la factorisation d'entiers ou du logarithme discret dans les corps finis.

NFS avec plusieurs polynômes L'algorithme NFS utilisé aujourd'hui pour factoriser des grands entiers n'est pas optimal. En effet, la variante proposée par Coppersmith [51] améliore marginalement la complexité heuristique, passant de $L_N[1/3, (64/9)^{1/3} = 1.923]$ à $L_N[1/3, 1.902]$. Ceci nécessite de travailler avec plusieurs corps de nombres, en quantité sous-exponentielle $L_N[1/3, 0.125]$. À ce jour, cet algorithme ne s'est pas montré efficace en pratique. Néanmoins, nous proposons conserver cette idée comme une possibilité, et d'étudier dans quelle mesure il est possible de rendre cet algorithme intéressant en pratique, ne serait-ce que pour acquérir un premier résultat expérimental sérieux permettant d'estimer la proximité de la plage de validité de cet algorithme.

Sélection polynomiale pour le logarithme discret dans $\mathbb{F}_p$ Nous pouvons rassembler sous le titre « sélection polynomiale » l'ensemble des choix liés aux structures algébriques intermédiaires (courbes, corps de nombres) construites au sein de la machinerie NFS/FFS pour atteindre l'objectif fixé. Concernant le contexte du logarithme discret, certaines approches sont spécifiques au contexte « corps finis », en comparaison avec le cas de la factorisation. En effet, puisqu'il est possible de calculer une racine d'un polynôme dans un corps fini, ce qu'on ne peut faire dans $\mathbb{Z}/N\mathbb{Z}$, la stratégie de choix d'une paire de polynômes consistant à fixer d'abord le polynôme algébrique, puis ensuite le polynôme rationnel, est viable. Cette stratégie est proposée par exemple dans [116], où le polynôme algébrique est choisi pour avoir un groupe de Galois prescrit, le polynôme rationnel étant optimisé via une étape de réduction de réseau. Une telle approche pour la sélection polynomiale pour le logarithme discret est viable, mais n'est pas l'approche retenue pour le record actuel [125], pour le calcul de logarithmes discrets sur $\mathbb{F}_p$. Avant de prendre à bras-le-corps un record de calcul de logarithmes discrets sur $\mathbb{F}_p$ par NFS, nous entendons mener la comparaison des différentes approches possibles. Il s'agit là d'une tâche courte qui sera menée au moment voulu.

Bases de facteurs Galois-invariantes Concernant le calcul de logarithmes discrets sur des corps de petite caractéristique, il y a matière à développer quelques travaux intéressants. En premier lieu, une cible privilégiée est donnée par les extensions de degré composé, puisque ces dernières sont une cible pour les applications de couplage. À ce jour, les premiers travaux, tels [102], impliquant le choix de bases de facteurs Galois-invariantes comme proposées par [56] nous incitent à bien sûr prendre en compte cet aspect pour des calculs futurs. Une partie des choix techniques faits dans [102, 195, 103] suscitent cependant notre interrogation, et nous incitent à un examen prudent pour les records à venir, notamment en caractéristique 2. D'autre part, l'impact de ces choix de bases de facteurs sur les calculs d'algèbre linéaire nécessite probablement un peu plus de précision que les résultats présentés jusqu'ici.

Élimination de restriction artificielles Un dernier point concernant la sélection polynomiale pour le calcul de logarithmes discrets en petite caractéristique nous amène à considérer un point de vue plus géométrique de l'algorithme FFS, tel que décrit en 3.4. Au regard des difficultés qui sont contournées avec succès dans le cadre NFS (impossibilité de calculer l'ordre maximal, gestion des unités), il est singulier de constater que les présentations communes de l'algorithme FFS se placent dans un cas

où « tout se passe bien », jusqu'à l'excès, alors même que ces obstructions y sont algorithmiquement plus faciles à contourner que dans le contexte NFS. L'algorithme FFS offre une grande latitude dans le choix du polynôme de définition, ce qui explique en partie le parti pris de se simplifier l'existence en posant de telles restrictions. Toutefois, ces restrictions peuvent entraver le souhait, important pour l'optimisation, d'avoir de nombreuses racines dans les petites extensions du corps de base. Aussi, nous entendons proposer dans une version future de CADO-NFS une élimination de ces contraintes que nous jugeons artificielles.

Étape de descente pour le logarithme discret Les algorithmes de la famille NFS, pour le logarithme discret, se concluent par une étape de descente. Nous avons décrit les algorithmes essentiels pour cette étape en 3.5. Un élément manquant aujourd'hui, qui peut se révéler utile pour les travaux à venir sur le thème du logarithme discret, est la formalisation d'une *stratégie* de descente, donnant un guide de choix parmi les différentes possibilités qui s'offrent lorsque la procédure est suivie. En effet, la ramification du calcul en de nombreux sous-calculs, et surtout la nécessité de choisir une branche plutôt qu'une autre pour la suite du calcul, font apparaître l'utilité de disposer d'un tel guide de choix.

17.2.2 Étude des points critiques, lecture fine, et ajustement automatique

Nos travaux produisent des logiciels. Nous souhaitons que ces logiciels soient efficaces. Lorsque ceux-ci s'avèrent d'une importance première pour les calculs que nous menons, il est légitime de dépenser une énergie importante sur la recherche des points d'optimisation possibles, et l'amélioration éventuelle des algorithmes employés.

L'époque de `gprof` est depuis longtemps révolue (n'en déplaise aux amateurs). Nous entendons, par détermination des points d'optimisation possible, un degré de finesse bien supérieur, en lien avec le comportement exact des microprocesseurs. Nous connaissons par exemple les travaux de réflexion menant à l'écriture de routines de base extrêmement optimisées dans le contexte de la bibliothèque Gnu MP. Il est possible aujourd'hui d'aspirer à une explication de chaque cycle CPU passé dans l'opération dominante d'un programme. C'est hélas difficile. Des outils se développent pour le permettre, à la fois en matériel et en logiciel[3]. Nous entendons être très vigilant sur la possibilité d'exploiter une telle source d'information afin d'améliorer, possiblement très significativement, le niveau de performances de nos programmes. Deux cibles naturelles pour de tels travaux d'optimisation sont données par les étapes les plus consommatrices de temps de calcul dans NFS.

Crible L'étape de crible représente la plus grande part du temps de calcul pour un algorithme comme NFS. L'implémentation de référence est due à Franke et Kleinjung, et l'implémentation présente dans le logiciel CADO-NFS est assez proche en termes de performances. Ayant une bonne connaissance des deux implémentations, nous pouvons espérer une amélioration de ces deux implantations, notamment une meilleure gestion des aspects d'exécution parallèle et de localité mémoire. Une quantification précise, au moyen d'outils de mesure et de simulation, des points limitants de telles implémentations, peut laisser entrevoir un ajustement aux caractéristiques de la machine cible, ce qui n'est pas fait actuellement. La variété des hiérarchies mémoire, et la multiplication du nombre de cœurs par processeur rend de telles préoccupations importantes. De tels travaux nécessitent à la fois une bonne connaissance des algorithmes utilisés, ainsi que de la possibilité de modifier ces algorithmes pour prendre en compte une spécificité matérielle. Nous entendons mener ce travail.

[3]Par exemple, les données des tables fournies par A. Fog [74] ne sont en rien « magiques », et peuvent être recalculées.

Briques de base d'algèbre linéaire creuse Dans un ordre d'idées similaire, nous n'avons que brièvement évoqué en 13.1 le nécessaire travail de fabrication de « BLAS creux » pour un produit matrice $\times$ vecteur efficace. Ce travail, tel qu'il existe aujourd'hui dans CADO-NFS, est à nos yeux incomplet. À l'instar de ce que nous venons d'évoquer concernant le crible, nous ne disposons pas concernant ce code d'informations fines sur les points précis qui limitent les performances des algorithmes utilisés, et de leurs implémentations. À nouveau, cette finesse de diagnostic peut permettre un ajustement automatique en fonction des caractéristiques des machines cible. Le développement de telles mesures permettrait de mieux définir les points de passage d'un algorithme à un autre. Nous comptons, dans cette perspective, rapprocher nos travaux de ce qui existe dans le monde numérique. Le travail que nous venons d'évoquer mérite une attention spécifique dans les deux cas distincts ou le corps de base est $\mathbb{F}_2$, ou $\mathbb{F}_p$ avec p assez grand (plusieurs mots machine). En effet, le coût par opération élémentaire étant très différent, il est bien évident qu'une solution unique n'est pas forcément envisageable.

Communications sur réseaux hautes performances La mise en œuvre de nos travaux d'algèbre linéaire nous a amené a travailler sur des réseaux tels les réseaux Infiniband. Nous avons brièvement indiqué quelques points relatifs à ce travail en 13.2. Le débit effectivement employé sur ces matériels lors des calculs que nous menons, comparé aux capacités théoriques maximales du matériel, laisse envisager des travaux d'optimisation possibles. Une réflexion sur les algorithmes collectifs utilisés et leur ordonnancement est susceptible d'apporter des résultats sur ce point.

17.2.3 Utilisation de composants GPU et FPGA

La dernière décennie a vu la démocratisation des FPGA, et l'arrivée des GPU comme plate-forme pour le calcul. Aujourd'hui, une machine de calcul « typique » inclut très souvent un GPU. L'argument de vente souvent donné est le « gigaflop par watt », à l'aune duquel il semble évident au décideur pressé qu'un GPU est avantageux. Bien entendu, pour tirer parti de telles ressources, le travail d'adaptation des algorithmes est considérable. Ceux-ci doivent être repensés afin de s'adapter à un tel modèle de calcul, radicalement différent du processeur superscalaire classique.

Des collaborations actuelles ou naissantes, notamment dans le cadre du projet ANR Catrel, nous amèneront à nous pencher sur de telles questions algorithmiques. À ce jour, notre activité scientifique ne nous a porté ni vers les GPU, ni vers les FPGA. Nous avons cependant des points de départ pour de tels travaux.

Rotation pour la sélection polynomiale de NFS Dans le contexte de la factorisation d'entiers, la sélection polynomiale a permis un saut qualitatif important après la factorisation de RSA-512 en 1999. L'algorithme de Kleinjung [124], succédant à la méthode de Montgomery-Murphy, a rendu possible le choix de polynômes bien meilleurs, pour un gain considérable en terme de relations obtenue par unité de temps (jusqu'à un facteur 2). Aujourd'hui, le nouvel algorithme proposé par Kleinjung [126] est probablement celui qui sera utilisé pour le prochain record. La particularité de ces algorithmes est l'ouverture qu'ils proposent à la technique appelée *rotation*, qui permet à partir d'une paire de polynômes initiale (f, g), de déterminer une combinaison judicieuse $(f + \lambda g, g)$ possédant de bonnes propriétés, notamment concernant les propriétés liées aux racines modulo les petits nombres premiers. L'immense liberté dans le choix du multiplieur λ qui est donnée par le second algorithme de Kleinjung donne lieu à un espace d'exploration immense. Des travaux actuels emploient des GPUs pour mener à bien de tels calculs, et nous avons proposé en outre un algorithme permettant un gain

constant sur la complexité de cette recherche [13]. Dans la mesure où l'étape de sélection polynomiale est le commencement d'un travail de record de factorisation, on peut voir cette piste de travail comme un objectif assez rapide : même si l'effort pour RSA-1024 ne commence pas immédiatement, un calcul par exemple sur RSA-896 pourrait être planifié rapidement. Nous disposons à notre portée des outils pour réaliser un travail de qualité sur ce point, en étudiant de près la pertinence de composants de type GPU.

Cofactorisation L'une des étapes du crible, aussi bien dans le contexte NFS que dans le contexte FFS, est l'étape de cofactorisation, où les facteurs résiduels de la norme issus du crible sont factorisés, afin de déterminer si une relation peut être produite ou pas. Dans le contexte NFS, il s'agit d'une étape de factorisation d'entiers (fort heureusement plus petits que le nombre à factoriser), et dans le contexte FFS d'une factorisation de polynômes. Pour NFS, des implémentations *ad hoc* de l'algorithme ECM, adaptées aux nombres de petite taille considérée, ont été mises en place par Kruppa [132]. Pour FFS, la tâche de cofactorisation est a priori plus aisée, mais un travail similaire est à faire. Ces deux contextes sont des candidats potentiels pour un usage pertinent de puissance de calcul auxiliaire, type GPU ou FPGA.

Algèbre linéaire sur $\mathbb{F}_p$ La résolution de grands systèmes linéaires creux sur $\mathbb{F}_p$ amène au travail d'optimisation du calcul de produits matrice $\times$ vecteur, où $\mathbb{F}_p$ est le corps de définition. Des travaux récents [201, 188] ont montré l'intérêt d'utiliser des GPUs pour effectuer cette opération dans le contexte de la factorisation d'entiers, où le corps de base est $\mathbb{F}_2$. Dans le cadre de la thèse de H. Jeljeli, nous étudions actuellement le gain possible lié à l'emploi de GPUs pour le cas $\mathbb{F}_p$. Les premiers résultats de ce travail sont extrêmement encourageants.

17.3 Travaux de nature plus purement arithmétique

La section précédente a rassemblé le morceau principal de nos projets sous le thème fédérateur de l'algorithme NFS et de ses algorithmes cousins, même si certains des thèmes évoqués débordent sur une thématique plus large. Il nous semble toutefois pertinent d'identifier une seconde partie de nos projets de recherche comme étant de nature plus essentiellement arithmétique (*tous* nos travaux, en un sens, relèvent de l'arithmétique, de près ou de loin). Ici nous entendons au premier chef nous concentrer sur les algorithmes dédiés au calcul de certaines opérations en particulier, et ensuite envisager les applications aux problèmes qui ont motivé ce travail.

17.3.1 Travaux en lien avec les courbes

Calculs de polynômes de classes En tant que pendant à nos travaux passés sur la cryptologie des courbes algébriques, nous avons récemment entamé l'étude d'un autre aspect constructif, plus mathématique, lié aux jacobiennes de courbes de genre 2 et à leur possible usage cryptographique, à savoir le calcul rapide de polynômes de classes d'Igusa (collaboration avec A. Enge). Ces travaux font suite aux thèses de Dupont [65] et Streng [202] et arriveront à maturité à court terme. Grâce à une approche asymptotiquement rapide du calcul de fonctions thêta, nous sommes en mesure de déterminer des jacobiennes de courbes de genre 2 avec un anneau d'endomorphismes prescrit, de nombre de classes dépassant $1\,000$, soit deux ordres de grandeur au-delà de l'état de l'art. Ces travaux nécessitent d'être poursuivis, notamment concernant certains aspects arithmétiques, pour pouvoir aller bien au-delà. Un nombre de classes de l'ordre de $10\,000$ est à ce jour une cible facile, et nous

envisageons qu'il sera bientôt possible, notamment en intégrant certaines améliorations récentes dûes à Streng, d'atteindre aisément 100 000 ou davantage. Sur cette piste de recherche, nous entendons apporter notre capacité à comprendre les concepts mathématiques difficiles, et à en extraire des implémentations particulièrement efficaces. Deux tâches arithmétiques essentielles sont au cœur de ce problème. La première de ces tâches est le calcul en temps quasi-linéaire de valeurs de fonctions thêta en 0, appelées « thêta-constantes » (pour une matrice de périodes Ω et un vecteur de caractéristiques $\left[\begin{smallmatrix}a\\b\end{smallmatrix}\right]$, nous appelons thêta constante la valeur $\Theta_{\left[\begin{smallmatrix}a\\b\end{smallmatrix}\right]}(0,\Omega)$). La seconde est l'optimisation dans un contexte multiprocesseur de l'opération dite « *polyfromroots* » de reconstruction d'un polynôme à partir de ces racines. Pour ces deux opérations, les résultats déjà obtenus sont bons.

Extensions et applications Ces travaux ouvrent la voie à des généralisations (par exemple au calcul de fonctions Θ en des valeurs arbitraires), et à des applications par exemple au calcul explicite d'isogénies, où à la déterminations de classes d'isogénies de certaines jacobiennes de courbes (des avancées conséquentes ont été obtenues sur ces problèmes sur la durée du projet CHIC, dont nous avons coordonné la partie Nancéienne. Ces avancées sont rapportées par les travaux de thèse de Bisson [26], Cosset [53], et Robert [179]).

17.3.2 Arithmétique des entiers et des polynômes

Algorithme de Gao-Mateer En dernier lieu, notre conviction dans la portée des algorithmes asymptotiquement rapides nous amènera à scruter, à moyen terme, les évolutions matérielles susceptibles de contredire le constat relativement décevant du chapitre 16, où l'algorithme de Cantor et la variante de Gao-Mateer sont peu ou pas rentables devant une approche moins « drôle » qui est celle de la FFT ternaire. Le nœud du problème est la complexité de l'opération de multiplication, qui est amenée à être modifiée dans les générations futures de microprocesseurs. Un point qui nous tient particulièrement à cœur est la possibilité de mettre en lumière un cas où un algorithme comme l'algorithme de Cantor, qui offre des produits de convolution particulièrement rapides, peut posséder un avantage décisif sur toute autre approche. Une cible privilégiée, que nous avons déjà indiquée, est l'étape centrale de l'algorithme de Wiedemann par blocs sur $\mathbb{F}_2$, mais pour en faire un cas d'usage pour lequel un gain est possible, des modifications de l'algorithme sont nécessaires.

Algorithme de Fürer Pour conclure sur les algorithmes asymptotiquement rapides, l'algorithme de Fürer [80], qui établit une nouvelle borne supérieure en $O(n(\log n)2^{O(\log^* n)})$ pour la multiplication d'entiers de n bits, pourrait à moyen ou long terme apparaître comme un algorithme important dans certains contextes. Ce n'est pas le cas aujourd'hui, mais il nous semble pertinent de veiller à cette éventualité. L'adaptation de cet algorithme au contexte de la caractéristique 2, en articulation avec le paragraphe précédent, est par ailleurs un sujet intéressant.

Bibliographie

[1] L. M. Adleman, *A subexponential algorithm for the discrete logarithm problem with applications to cryptography*, 20th Annual Symposium on Foundations of Computer Science (FOCS '79), 55–60. IEEE Computer Society Press, 1979. San Juan, Puerto Rico, October 29–31, 1979. ↑ 27, 40, 111

[2] L. M. Adleman, *Factoring numbers using singular integers*, Proc. 23nd Annual ACM Symposium on Theory of Computing (STOC), 64–71. ACM, 1991. ↑ 18

[3] L. M. Adleman, *The function field sieve*. In L. M. Adleman and M.-D. Huang (eds.), ANTS-I, vol. 877 of *Lecture Notes in Comput. Sci.*, 108–121. Springer–Verlag, 1994. 1st Algorithmic Number Theory Symposium, Cornell University, May 6–9, 1994. ↑ 25, 35

[4] L. M. Adleman and J. DeMarrais, *A subexponential algorithm for discrete logarithms over all finite fields*, Math. Comp. **61**(203) (1993), 1–15. ↑ 26

[5] L. M. Adleman, J. DeMarrais, and M.-D. Huang, *A subexponential algorithm for discrete logarithms over the rational subgroup of the jacobians of large genus hyperelliptic curves over finite fields*. In L. M. Adleman and M.-D. Huang (eds.), ANTS-I, vol. 877 of *Lecture Notes in Comput. Sci.*, 28–40. Springer–Verlag, 1994. 1st Algorithmic Number Theory Symposium, Cornell University, May 6–9, 1994. ↑ 83, 89, 92, 111

[6] L. M. Adleman and M.-D. Huang, *Function field sieve methods for discrete logarithms over finite fields*, Inform. and Comput. **151**(1) (1999), 5–16. ↑ 25, 35, 46

[7] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The design and analysis of computer algorithms*, Addison–Wesley, Reading, MA, 1974. ↑ 94

[8] D. F. Aranha and C. P. L. Gouvêa, *RELIC is an Efficient LIbrary for Cryptography*. Homepage at `http://code.google.com/p/relic-toolkit/`. ↑ 163, 182

[9] S. Arita, *Algorithms for computations in Jacobians of C_{ab} curve and their application to discrete-log-based public key cryptosystems*, IEICE Trans. Fundamentals **J82-A**(8) (1999), 1291–1299. In japanese. English translation in [10]. ↑ 191

[10] S. Arita, *Algorithms for computations in Jacobians of C_{ab} curve and their application to discrete-log-based public key cryptosystems*, Proceedings of Conference on The Mathematics of Public Key Cryptography, Toronto, June 12–17, 1999. Translated from japanese [9]. ↑ 109, 191

[11] S. Arita, *Gaudry's variant against C_{ab} curves*. In H. Imai and Y. Zheng (eds.), Public Key Cryptography - PKC 2000, vol. 1751 of *Lecture Notes in Comput. Sci.*, 58–67, 2000. ↑ 109

[12] S. Arita, *An addition algorithm in Jacobian of C_{ab} curves*, Discrete Appl. Math. **130**(1) (2003), 13–31. ↑ 109

[13] S. Bai, R. P. Brent, and E. Thomé, *Root optimization of polynomials in the number field sieve*, 2011. Available at `http://maths.anu.edu.au/~bai/`. Manuscript in preparation– `http://maths.anu.edu.au/~bai/`. ↑ 19, 24, 188

[14] R. Bărbulescu, *Improvements to the discrete logarithm problem in $\mathbb{F}_p^*$*, M2 Internship report, ENS Lyon, 2011. ↑ 43

[15] A. Basiri, A. Enge, J.-C. Faugère, and N. Gürel, *Implementing the Arithmetic of $C_{3,4}$-Curves*. In D. Buell (ed.), ANTS-VI, vol. 3076 of *Lecture Notes in Comput. Sci.*, 87–101. Springer–Verlag, 2004. 6th Algorithmic Number Theory Symposium, Burlington, VT, June 2004. ↑ 109

[16] A. Basiri, A. Enge, J.-C. Faugère, and N. Gürel, *The arithmetic of Jacobian groups of superelliptic cubics*, Math. Comp. **74**(249) (2005), 389–410. ↑ 109

[17] M. Bauer, E. Teske, and A. Weng, *Point counting on Picard curves in large characteristic*, Math. Comp. **74**(185) (2005), 1983–2005. ↑ 109

[18] B. Beckerman and G. Labahn, *A uniform approach for the fast computation of matrix-type Padé approximants*, SIAM J. Matrix Anal. Appl. **15**(3) (1994), 804–823. ↑ 129, 135, 136

[19] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway, *Relations Among Notions of Security for Public-Key Encryption Schemes*. In H. Krawczyk (ed.), Advances in Cryptology – CRYPTO '98, vol. 1462 of *Lecture Notes in Comput. Sci.*, 26–45. Springer–Verlag, 1998. Proc. 18th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 23–27, 1998. ↑ 59

[20] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko, *The One-More-RSA-Inversion Problems and the Security of Chaum's Blind Signature Scheme*, J. Cryptology **16**(3) (2003), 185–215. ↑ 60

[21] M. Bellare and P. Rogaway, *Optimal asymmetric encryption*. In A. De Santis (ed.), Advances in Cryptology – EUROCRYPT '94, vol. 950 of *Lecture Notes in Comput. Sci.*, 92–111, 1995. Proc. Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 1994. ↑ 59

[22] E. R. Berlekamp, *Algebraic coding theory*, McGraw-Hill, 1968. ↑ 129

[23] D. J. Bernstein, *Curve25519 : new Diffie-Hellman speed records*. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin (eds.), Public Key Cryptography - PKC 2006, vol. 3958 of *Lecture Notes in Comput. Sci.*, 207–228. Springer–Verlag, 2006. 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006. ↑ 163

[24] D. J. Bernstein, P. Birkner, T. Lange, and C. Peters, *ECM using Edwards curves*, 2011. Preprint. See `http://eecm.cr.yp.to/`. ↑ 163

[25] D. J. Bernstein and T. Lange, *Explicit formula database*. Available at `http://www.hyperelliptic.org/EFD/`. `http://www.hyperelliptic.org/EFD/`. ↑ 87

[26] G. Bisson, *Endomorphism rings in cryptography*, PhD thesis, T.U. Eindhoven, 2011. ↑ 189

[27] I. F. Blake, R. Fuji-Hara, R. C. Mullin, and S. A. Vanstone, *Computing logarithms in finite fields of characteristic two*, SIAM J. Alg. Disc. Meth. **5**(2) (1984), 276–285. ↑ 40, 42

[28] M. Bodrato, *Towards optimal Toom-Cook multiplication for univariate and multivariate polynomials in characteristic 2 and 0*. In C. Carlet and B. Sunar (eds.), WAIFI 2007, vol. 4547 of *Lecture Notes in Comput. Sci.*, 116–133. Springer–Verlag, 2007. June 21-22, 2007. Madrid, Spain. ↑ 168

[29] University of California, *BOINC, Berkeley Open Infrastructure for Network Computing*, 2002–. `http://boinc.berkeley.edu/`. ↑ 75

[30] R. P. Brent, *Multiple-precision zero-finding methods and the complexity of elementary function evaluation*. In J. F. Traub (ed.), Analytic computational complexity, 151–176. Academic Press, New York, NY, 1975. Available at `http://web.comlab.ox.ac.uk/oucl/work/richard.brent/ftp/rpb028.ps.gz`. ↑ 48, 49, 153

[31] R. Brent, P. Gaudry, E. Thomé, and P. Zimmermann, *Faster Multiplication in GF(2)[x]*. In A. van der Poorten and A. Stein (eds.), ANTS-VIII, vol. 5011 of *Lecture Notes in Comput. Sci.*, 153–166. Springer–Verlag, 2008. ↑ 1, 165, 166, 167, 173 [*clé dans le texte :* [T10]]

[32] R. Brent and P. Zimmermann, *Ten new primitive binary trinomials*, Math. Comp. **78**(266) (2009), 1197–1199. ↑ 151, 165

[33] R. Brent and P. Zimmermann, *Modern Computer Arithmetic*, Cambridge Monographs on Applied and Computational Mathematics, vol. 18, Cambridge University Press, 2010. ↑ 18, 48, 49, 149, 153

[34] R. Brent and P. Zimmermann, *The great trinomial hunt*, Notices Amer. Math. Soc **58**(2) (2011), 233–239. ↑ 151, 165

[35] J. P. Buhler, A. K. Lenstra, and J. M. Pollard, *Factoring integers with the number field sieve*. In A. K. Lenstra and H. W. Lenstra Jr (eds.), The development of the number field sieve, vol. 1554 of *Lecture Notes in Math.*, 50–94. Springer–Verlag, 1993. ↑ 13, 17, 18, 19

[36] S. Burckel, E. Gioan, and E. Thomé, *Mapping Computation with No Memory*, 8th International Conference on Unconventional Computation - UC09, vol. 5715 of *Lecture Notes in Comput. Sci.*, 85–97. Springer–Verlag, 2009. The original publication is available at www.springerlink.com. ↑ 181 [*clé dans le texte :* [T11]]

[37] D. G. Cantor, *On arithmetical algorithms over finite fields*, J. Combin. Theory Ser. A **50** (1989), 285–300. ↑ 165, 170

[38] S. Carney, M. A. Heroux, G. Li, and K. Wu. *A revised proposal for a sparse BLAS toolkit*, Technical Report 94-034, Army High Performance Computing Research Center, 1994. ↑ 139

[39] S. Cavallar, *Strategies in filtering in the number field sieve*. In W. Bosma (ed.), ANTS-IV, vol. 1838 of *Lecture Notes in Comput. Sci.*, 209–231. Springer–Verlag, 2000. 4th Algorithmic Number Theory Symposium, Leiden, The Netherlands, July 2–7, 2000. ↑ 33, 109

[40] S. Cavallar, B. Dodson, A. K. Lenstra, W. Lioen, P. L. Montgomery, B. Murphy, H. J. J. te Riele, K. Aardal, J. Gilchrist, G. Guillerm, P. Leyland, J. Marchand, F. Morain, A. Muffett, C. Putnam, C. Putnam, and P. Zimmermann, *Factorization of a 512-bit RSA modulus*. In B. Preneel (ed.), Advances in Cryptology – EUROCRYPT 2000, vol. 1807 of *Lecture Notes in Comput. Sci.*, 1–18. Springer–Verlag, 2000. Proc. International Conference on the Theory and Application of Cryptographic Techniques, Brugge, Belgium, May 2000. ↑ 64

[41] F. Chabaud and R. Lercier, `ZEN`, *A toolbox for fast computation in finite extensions over finite rings*. Available at `http://www.di.ens.fr/~zen`. ↑ 155

[42] D. Chaum and H. van Antwerpen, *Undeniable signatures*. In G. Brassard (ed.), Advances in Cryptology – CRYPTO '89, vol. 435 of *Lecture Notes in Comput. Sci.*, 212–217. Springer–Verlag, 1990. Proc. 9th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 1989. ↑ 70

[43] L. Chen, W. Eberly, E. Kaltofen, B. D. Saunders, W. J. Turner, and G. Villard, *Efficient matrix preconditioners for black box linear algebra*, Linear Algebra Appl. **343–344** (2002), 119–146. ↑ 122

[44] H. Cheng, G. Hanrot, E. Thomé, E. Zima, and P. Zimmermann, *Time- and Space-Efficient Evaluation of Some Hypergeometric Constants*. In C. W. Brown (ed.), ISSAC 2007, 85–91. ACM Press, 2007. Proc. International Symposium on Symbolic and Algebraic Computation, July 29–August 1st, 2007, London, ON. ↑ 181 [*clé dans le texte :* [T6]]

[45] F. Chung and L. Lu, *The diameter of random sparse graphs*, Adv. Appl. Math. **26** (2001), 257–279. ↑ 107

[46] H. Cohen, *A course in algorithmic algebraic number theory*, Grad. Texts in Math., vol. 138, Springer–Verlag, 1993. ↑ 16

[47] H. Cohen and K. Belabas, *Pari/GP*, 1985–. `http://pari.math.u-bordeaux.fr/`. ↑ 153

[48] H. Cohen and G. Frey, *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, Discrete Math. Appl. (Boca Raton), Chapman & Hall/CRC, 2005. ↑ 165

[49] A. Commeine and I. Semaev, *An algorithm to solve the discrete logarithm problem with the number field sieve*. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin (eds.), Public Key Cryptography - PKC 2006, vol. 3958 of *Lecture Notes in Comput. Sci.*, 174–190. Springer–Verlag, 2006. 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006. ↑ 45, 46

[50] D. Coppersmith, *Fast evaluation of logarithms in fields of characteristic two*, IEEE Trans. Inform. Theory **IT–30**(4) (1984), 587–594. ↑ 25, 38, 40, 45

[51] D. Coppersmith, *Modifications to the number field sieve*, J. Cryptology **6** (1993), 169–180. ↑ 185

[52] D. Coppersmith, *Solving linear equations over* $\mathrm{GF}(2)$ *via block Wiedemann algorithm*, Math. Comp. **62**(205) (1994), 333–350. ↑ 5, 121, 122

[53] R. Cosset, *Applications des fonctions thêta à la cryptographie sur courbes hyperelliptiques*, Thèse, Université Henri Poincaré, 2011. ↑ 189

[54] J.-M. Couveignes, *Computing a square root for the number field sieve*. In A. K. Lenstra and H. W. Lenstra Jr (eds.), The development of the number field sieve, vol. 1554 of *Lecture Notes in Math.*, 95–102. Springer–Verlag, 1993. ↑ 49

[55] J.-M. Couveignes, *Algebraic groups and discrete logarithm*. In K. Alster, J. Urbanowicz, and H. C. Williams (eds.), Public-Key Cryptography and Computational Number Theory, 17–27. Walter de Gruyter, 2001. Sept. 11-15, 2000, Warsaw, Poland. ↑ 89, 111

[56] J.-M. Couveignes and R. Lercier, *Galois Invariant Smoothness Basis*, Algebraic geometry and its applications, May 2008, pp. 142–167. Proc. first SAGA conference, 7-11 May 2007, Papeete. ↑ 185

[57] J. A. Davis and D. B. Holridge, *Factorization using the quadratic sieve algorithm*. In D. Chaum (ed.), Advances in Cryptology – CRYPTO '83, 103–113. Plenum Press, 1984. Proc. Cryptology Workshop, Santa Barbara, CA, August 22–24, 1983. ↑ 41, 65

[58] C. Diem, *An index calculus algorithm for plane curves of small degree*. In F. Hess, S. Pauli, and M. E. Pohst (eds.), ANTS-VII, vol. 4076 of *Lecture Notes in Comput. Sci.*, 543–557. Springer–Verlag, 2006. 7th Algorithmic Number Theory Symposium, Berlin, Germany, July 23-28, 2006. ↑ 103, 104, 110

[59] C. Diem and E. Thomé, *Index calculus in class groups of non-hyperelliptic curves of genus three*, J. Cryptology **21**(4) (2008), 593–611. ↑ 1, 83, 103, 105, 106, 108, 109, 110, 182 [*clé dans le texte :* [T9]]

[60] W. Diffie and M. E. Hellman, *New directions in cryptography*, IEEE Trans. Inform. Theory **IT–22**(6) (1976), 644–654. ↑ 5

[61] B. Dodson and A. K. Lenstra, *NFS with four large primes : an explosive experiment*. In D. Coppersmith (ed.), Advances in Cryptology – CRYPTO '95, vol. 963 of *Lecture Notes in Comput. Sci.*, 372–385. Springer–Verlag, 1995. Proc. 15th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 27–31, 1995. ↑ 95

[62] I. S. Duff, M. Heroux, and R. Pozo, *An overview of the sparse basic linear algebra subprograms : The new standard from the BLAS Technical Forum*, ACM Trans. Math. Software **28** (2002), 239–267. ↑ 139

[63] J.-G. Dumas, T. Gautier, P. Giorgi, J.-L. Roch, and G. Villard, *Givaro, une bibliothèque C++ pour le Calcul Formel*, 1987–. Homepage at `http://ljk.imag.fr/CASYS/LOGICIELS/givaro/`. ↑ 156

[64] J.-G. Dumas, P. Giorgi, and C. Pernet, *Dense Linear Algebra over Word-Size Prime Fields : the FFLAS and FFPACK Packages*, ACM Trans. Math. Software **35**(3) (2008). ↑ 141, 154

[65] R. Dupont, *Moyenne arithmético-géométrique, suites de Borchardt et applications*, Thèse, École Polytechnique, 2006. ↑ 188

[66] W. Eberly and E. Kaltofen, *On randomized Lanczos algorithm*. In W. W. Küchlin (ed.), ISSAC '97, 176–183. ACM Press, 1997. Proc. International Symposium on Symbolic and Algebraic Computation, July 21–23, 1997, Maui, Hawaii. ↑ 122

[67] T. ElGamal, *A public-key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Trans. Inform. Theory **IT–31**(4) (1985), 469–472. ↑ 70

[68] R. M. Elkenbracht-Huizing, *An implementation of the number field sieve*, Experiment. Math. **5**(3) (1996), 231–253. ↑ 121

[69] A. Enge and P. Gaudry, *A general framework for subexponential discrete logarithm algorithms*, Acta Arith. **102**(1) (2002), 83–103. ↑ 29, 89, 105, 111, 112

[70] A. Enge, P. Gaudry, and E. Thomé, *An $L(1/3)$ discrete logarithm algorithm for low degree curves*, J. Cryptology **24**(1) (2011), 24–41. ↑ 1, 43, 45, 83, 111, 112, 113, 114, 115, 182, 185 [*clé dans le texte :* [T15]]

[71] A. Enge and A. V. Sutherland, *Class invariants by the CRT method*. In G. Hanrot, F. Morain, and E. Thomé (eds.), ANTS-IX, vol. 6197 of *Lecture Notes in Comput. Sci.*, 142–156. Springer–Verlag, 2010. 9th Algorithmic Number Theory Symposium, Nancy, France, July 19-23, 2010. ↑ 52, 57

[72] P. Flajolet and A. M. Odlyzko, *Random mapping statistics*. In J.-J. Quisquater and J. Vandewalle (eds.), Advances in Cryptology – EUROCRYPT '89, vol. 434 of *Lecture Notes in Comput. Sci.*, 329–354. Springer–Verlag, 1990. Proc. Eurocrypt '89, Houthalen, April 10–13, 1989. ↑ 97

[73] S. Flon and R. Oyono, *Fast arithmetic on Jacobians on Picard curves*. In F. Bao, R. Deng, and J. Zhou (eds.), Public Key Cryptography - PKC 2004, vol. 2947 of *Lecture Notes in Comput. Sci.*, 55–68, 2004. ↑ 109

[74] A. Fog, *Software optimization resources*, 1996–. `http://www.agner.org/optimize/`. ↑ 167, 186

[75] W. Ford and B. Kaliski, *Server-assisted generation of a strong secret from a password*, Proceedings of the IEEE 9th International Workshop on Enabling Technologies : Infrastructure for Collaborative Enterprises (WET ICE), 176–180. IEEE Computer Society Press, 2000. ↑ 70

[76] M. Fouquet, P. Gaudry, and R. Harley, *Finding secure curves with the Satoh-FGH algorithm and an early-abort strategy*. In B. Pfitzmann (ed.), Advances in Cryptology – EUROCRYPT 2001, vol. 2045 of *Lecture Notes in Comput. Sci.*, 14–29. Springer–Verlag, 2001. ↑ 87

[77] J. Franke and T. Kleinjung, *Continued fractions and lattice sieving*, Special-purpose hardware for attacking cryptographic Systems – SHARCS, 2005. ↑ 37

[78] D. M. Freeman. *Pairing-based identification schemes*, Technical report HPL-2005-154, Hewlett-Packard Laboratories, 2005. ↑ 60, 66

[79] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern, *RSA-OAEP is secure under the RSA assumption*, J. Cryptology **17**(2) (2004), 81–104. ↑ 59

[80] M. Fürer, *Faster integer multiplication*. In U. Feige (ed.), STOC '07, 57–66. ACM, 2007. ↑ 189

[81] S. Gao and T. Mateer, *Additive Fast Fourier Transforms over Finite Fields*, IEEE Trans. Inform. Theory **56**(12) (2010), 6265–6272. ↑ 173

[82] J. von zur Gathen and J. Gerhard. *Arithmetic and factorization of polynomials over* $\mathbb{F}_2$ (extended abstract), Technical report tr-rsfb-96-018, University of Paderborn, Germany, 1996. ↑ 165, 170, 172

[83] J. von zur Gathen and J. Gerhard, *Arithmetic and factorization of polynomials over* $\mathbb{F}_2$ (extended abstract). In Y. N. Lakshman (ed.), ISSAC '96, 1–9. ACM Press, 1996. Proc. International Symposium on Symbolic and Algebraic Computation, July 24–26, 1996, Zurich, Switzerland. ↑ 165

[84] J. von zur Gathen and J. Gerhard, *Modern computer algebra*, Cambridge University Press, Cambridge, England, 1999. ↑ 54, 165, 169

[85] J. von zur Gathen and J. Gerhard, *Polynomial factorization over* $\mathbb{F}_2$, Math. Comp. **71**(240) (2002), 1677–1698. ↑ 165

[86] P. Gaudry, *An algorithm for solving the discrete log problem on hyperelliptic curves*. In B. Preneel (ed.), Advances in Cryptology – EUROCRYPT 2000, vol. 1807 of *Lecture Notes in Comput. Sci.*, 19–34. Springer–Verlag, 2000. Proc. International Conference on the Theory and Application of Cryptographic Techniques, Brugge, Belgium, May 2000. ↑ 83, 89, 90, 91, 92

[87] P. Gaudry, *Fast genus 2 arithmetic based on Theta functions*, J. Math. Cryptol. **1**(3) (2007), 243–265. ↑ 87, 163

[88] P. Gaudry and N. Gürel, *An extension of Kedlaya's algorithm to superelliptic curves*. In C. Boyd and E. Dawson (eds.), Advances in Cryptology – ASIACRYPT 2001, vol. 2248 of *Lecture Notes in Comput. Sci.*, 480–494. Springer–Verlag, 2001. Proc. 7th International Conference on the Theory and Applications of Cryptology and Information Security, Dec. 9–13, 2001, Gold Coast, Queensland, Australia. ↑ 152

[89] P. Gaudry and R. Harley, *Counting Points on Hyperelliptic Curves over Finite Fields*. In W. Bosma (ed.), ANTS-IV, vol. 1838 of *Lecture Notes in Comput. Sci.*, 313–332. Springer–Verlag, 2000. 4th Algorithmic Number Theory Symposium, Leiden, The Netherlands, July 2–7, 2000. ↑ 155

[90] P. Gaudry, A. Kruppa, F. Morain, L. Muller, E. Thomé, and P. Zimmermann, `cado-nfs`, *An Implementation of the Number Field Sieve Algorithm*, 2011. Available at `http://cado-nfs.gforge.inria.fr/`. Release `1.1`. ↑ 1, 24, 47, 56, 139, 141, 159

[91] P. Gaudry and É. Schost, *Genus 2 point counting over prime fields* **47**(4) (2011), 368–400. ↑ 88, 152, 159

[92] P. Gaudry and E. Thomé, *The mpFq library and implementing curve-based key exchanges*, SPEED : Software Performance Enhancement for Encryption and Decryption, 49–64, 2007. ↑ 1, 155, 159, 163 [*clé dans le texte :* [T5]]

[93] P. Gaudry, E. Thomé, N. Thériault, and C. Diem, *A double large prime variation for small genus hyperelliptic index calculus*, Math. Comp. **76**(257) (2007), 475–492. ↑ 1, 83, 91, 95, 96, 97, 104 [*clé dans le texte :* [T7]]

[94] S. Goldwasser, S. Micali, and R. L. Rivest, *A digital signature scheme secure against adaptive chosen-message attacks*, SIAM J. Comput. **17**(2) (1988), 281–308. ↑ 59

[95] D. M. Gordon, *Discrete logarithms in* $\mathrm{GF}(p)$ *using the number field sieve*, SIAM J. Discrete Math. **6**(1) (1993), 124–138. ↑ 25, 46

[96] T. Granlund, `GMP`, *The GNU Multiple Precision Arithmetic Library*, 1991–. `http://www.swox.com/gmp`. ↑ 153, 155

[97] J. L. Hafner and K. S. McCurley, *A rigorous subexponential algorithm for computation of class groups*, J. Amer. Math. Soc. **2**(4) (1989), 837–850. ↑ 103

[98] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer Professional Computing, Springer–Verlag, 2004. ↑ 165, 166

[99] G. Hanrot, M. Quercia, and P. Zimmermann, *The middle product algorithm, I. Speeding up the division and square root of power series*, Appl. Algebra Engrg. Comm. Comput. **14**(6) (2004), 415–438. ↑ 134

[100] R. Harley, *The ECDL project*, 2000. `http://cristal.inria.fr/~harley/ecdl/`. ↑ 155

[101] David Harvey, *The Karatsuba integer middle product*, J. Symbolic Comput. **47**(8) (2012), 954–967. ↑ 134

[102] T. Hayashi, N. Shinohara, L. Wang, S. Matsuo, M. Shirase, and T. Takagi, *Solving a 676-bit discrete logarithm problem in* $\mathrm{GF}(3^{6n})$. In P. Q. Nguyen and D. Pointcheval (eds.), Public Key Cryptography - PKC 2010, vol. 6056 of *Lecture Notes in Comput. Sci.*, 351–367. Springer–Verlag, 2010. Proc. 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. ↑ 25, 185

[103] T. Hayashi, T. Shimoyama, N. Shinohara, and T. Takagi, *Breaking pairing-based cryptosystems using* η_T *pairing over* $GF(3^{97})$, 2012. Available at `http://eprint.iacr.org/2012/345`. ↑ 185

[104] F. Heß, *Computing Riemann-Roch spaces in algebraic function fields and related topics*, J. Symbolic Comput. **33** (2002), 425–445. ↑ 112

[105] F. Heß, *Computing relations in divisor class groups of algebraic curves over finite fields*, 2004. Preprint, submitted to J. Symbolic Comput. Available at `http://www.staff.uni-oldenburg.de/florian.hess/publications/dlog.pdf`. ↑ 111, 112

[106] J. van der Hoeven, *The truncated Fourier transform and applications*. In J. Gutierrez (ed.), ISSAC 2004, 290–296. ACM Press, 2004. Proc. International Symposium on Symbolic and Algebraic Computation, July 4–7, 2004, Santander, Spain. ↑ 173

[107] N. Howgrave-Graham and A. Joux, *New generic algorithms for hard knapsacks*. In Henri Gilbert (ed.), EUROCRYPT, vol. 6110 of *Lecture Notes in Comput. Sci.*, 235–256. Springer–Verlag, 2010. Proc. 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. ↑ 56

[108] E. Im and K. A. Yelick, *Model-based memory hierarchy optimizations for sparse matrices*, Workshop on Profile and Feedback-Directed Compilation, 1998. ↑ 139

[109] S. Janson, T. Luczak, and A. Rucinski, *Random Graphs*, Wiley, New York, 2000. ↑ 97, 107

[110] A. Joux and R. Lercier, *Discrete logarithms in* $\mathrm{GF}(p)$ *- 90 digits*, May 1998. Email to the `NMBRTHRY` mailing-list. ↑ 25

[111] A. Joux and R. Lercier, *Discrete logarithms in* $\mathrm{GF}(p)$ *- 100 digits*, Nov. 1999. Email to the `NMBRTHRY` mailing-list. ↑ 25

[112] A. Joux and R. Lercier, *Discrete logarithms in* $\mathrm{GF}(p)$ *(110 decimal digits)*, Jan. 2001. Email to the `NMBRTHRY` mailing-list. ↑ 25

[113] A. Joux and R. Lercier, *Discrete logarithms in* $\mathrm{GF}(p)$ *(120 decimal digits)*, Apr. 2001. Email to the `NMBRTHRY` mailing-list. ↑ 25

[114] A. Joux and R. Lercier, *Discrete logarithms in* $\mathrm{GF}(2^n)$ *(521 bits)*, Sep. 2001. Email to the `NMBRTHRY` mailing-list. ↑ 25

[115] A. Joux and R. Lercier, *The function field sieve is quite special*. In C. Fieker and D. R. Kohel (eds.), ANTS-V, vol. 2369 of *Lecture Notes in Comput. Sci.*, 431–445. Springer–Verlag, 2002. 5th Algorithmic Number Theory Symposium, Sydney, Australia, July 2002. ↑ 25, 35, 37, 46

[116] A. Joux and R. Lercier, *Improvements to the general number field sieve for discrete logarithms in prime fields. A comparison with the gaussian integer method*, Math. Comp. **72**(242) (2003), 953–967. ↑ 25, 28, 29, 43, 46, 185

[117] A. Joux and R. Lercier, *Discrete logarithms in* $\mathrm{GF}(p)$ *- 130 digits*, June 2005. Email to the `NMBRTHRY` mailing-list. ↑ 25

[118] A. Joux and R. Lercier, *Discrete logarithms in* $\mathrm{GF}(2^{607})$ *and* $\mathrm{GF}(2^{613})$, Sep. 2005. Email to the `NMBRTHRY` mailing-list. ↑ 25

[119] A. Joux and R. Lercier, *Discrete logarithms in* $\mathrm{GF}(65537^{25})$ *- 120 digits - 400 bits*, Oct. 2005. Email to the `NMBRTHRY` mailing-list. ↑ 25

[120] A. Joux and R. Lercier, *Discrete logarithms in* $\mathrm{GF}(370801^{30})$ *- 168 digits - 556 bits*, Nov. 2005. Email to the `NMBRTHRY` mailing-list. ↑ 25

[121] A. Joux, R. Lercier, D. Naccache, and E. Thomé, *Oracle-assisted static Diffie-Hellman is easier than discrete logarithms*. In M. G. Parker (ed.), Cryptography and Coding 2009, vol. 5921 of *Lecture Notes in Comput. Sci.*, 351–367. Springer–Verlag, 2009. ↑ 1, 6, 59, 60, 70, 72, 119, 185 [*clé dans le texte :* [T12]]

[122] A. Joux, R. Lercier, N. P. Smart, and F. Vercauteren, *The number field sieve in the medium prime case*. In C. Dwork (ed.), Advances in Cryptology – CRYPTO 2006, vol. 4117 of *Lecture Notes in Comput. Sci.*, 326–344. Springer–Verlag, 2006. Proc. 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006. ↑ 25, 26, 35

[123] A. Joux, D. Naccache, and E. Thomé, *When* e*-th roots become easier than factoring*. In K. Kurosawa (ed.), Advances in Cryptology – ASIACRYPT 2007, vol. 4833 of *Lecture Notes in Comput. Sci.*, 13–28. Springer–Verlag, 2008. Proc. 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007. ↑ 1, 4, 47, 59, 60, 68, 69, 70, 119, 185 [*clé dans le texte :* [T8]]

[124] T. Kleinjung, *On polynomial selection for the general number field sieve*, Math. Comp. **75**(256) (2006), 2037–2047. ↑ 19, 75, 187

[125] T. Kleinjung, *Discrete logarithms in* $\mathrm{GF}(p)$ *- 160 digits*, May 2007. Email to the `NMBRTHRY` mailing-list. ↑ 25, 185

[126] T. Kleinjung, *Polynomial Selection*, CADO workshop on integer factorization, Nancy, France, 2008. Available at `http://cado.gforge.inria.fr/workshop/abstracts.html`. ↑ 19, 187

[127] T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik, H. te Riele, A. Timofeev, and P. Zimmermann, *Factorization of a 768-bit RSA modulus*. In T. Rabin (ed.), Advances in Cryptology – CRYPTO 2010, vol. 6223 of *Lecture Notes in Comput. Sci.*, 333–350. Springer–Verlag, 2010. Proc. 30th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2010. ↑ 1, 4, 37, 47, 73, 119, 120, 129, 137 [*clé dans le texte :* [T13]]

[128] T. Kleinjung, J. Bos, A. Lenstra, D. A. Osvik, K. Aoki, S. Contini, J. Franke, E. Thomé, P. Jermini, M. Thiémard, P. Leyland, P. Montgomery, A. Timofeev, and H. Stockinger, *A Heterogeneous Computing Environment to Solve the 768-bit RSA Challenge*, Cluster Comput. **15**(1) (2012), 53–68. ↑ 76, 77, 78, 119, 182 [*clé dans le texte :* [T16]]

[129] T. Kleinjung, L. Nussbaum, and E. Thomé, *Using a grid platform for solving large sparse linear systems over GF(2)*, 11th ACM/IEEE International Conference on Grid Computing (Grid 2010), 2010. ↑ 1, 78, 119, 144, 146 [*clé dans le texte :* [T14]]

[130] N. Koblitz, *Elliptic curve cryptosystems*, Math. Comp. **48**(177) (1987), 203–209. ↑ 8

[131] N. Koblitz and A. Menezes, *Another look at non-standard discrete log and Diffie-Hellman problems*, J. Math. Cryptol. **2**(4) (2008), 311–326. Available at `http://eprint.iacr.org/2007/442`. ↑ 60, 66

[132] A. Kruppa, *Factoring into large primes with* $p-1$, $p+1$ *and ECM*, CADO workshop on integer factorization, Nancy, France, 2008. Available at `http://cado.gforge.inria.fr/workshop/`. ↑ 188

[133] B. A. LaMacchia and A. M. Odlyzko, *Solving large sparse linear systems over finite fields*. In A. J. Menezes and S. A. Vanstone (eds.), Advances in Cryptology – CRYPTO '90, vol. 537 of *Lecture Notes in Comput. Sci.*, 109–133. Springer–Verlag, 1990. Proc. 10th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 11–15, 1990. ↑ 20, 121

[134] C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh, *Basic linear algebra subprograms for FORTRAN usage*, ACM Trans. Math. Software **5**(3) (1979), 308–323. ↑ 139

[135] A. K. Lenstra and H. W. Lenstra Jr (eds.), *The development of the number field sieve*, Lecture Notes in Math., vol. 1554, Springer–Verlag, 1993. ↑ 13

[136] A. K. Lenstra, H. W. Lenstra Jr, M. S. Manasse, and J. M. Pollard, *The number field sieve*. In A. K. Lenstra and H. W. Lenstra Jr (eds.), The development of the number field sieve, vol. 1554 of *Lecture Notes in Math.*, 11–42. Springer–Verlag, 1993. ↑ 4, 13

[137] A. K. Lenstra and M. S. Manasse, *Factoring by electronic mail*. In J.-J. Quisquater and J. Vandewalle (eds.), Advances in Cryptology – EUROCRYPT '89, vol. 434 of *Lecture Notes in Comput. Sci.*, 355–371. Springer–Verlag, 1990. Proc. Eurocrypt '89, Houthalen, April 10–13, 1989. ↑ 75

[138] A. K. Lenstra and M. S. Manasse, *Factoring with two large primes*, Math. Comp. **63**(208) (1994), 785–798. ↑ 92, 95

[139] H. W. Lenstra Jr, *Factoring integers with elliptic curves*, Ann. of Math. (2) **126** (1987), 649–673. ↑ 4, 43

[140] R. Lercier and D. Lubicz, *A quasi quadratic time algorithm for hyperelliptic curve point counting*, J. Ramanujan Math. Soc. **12** (2006), 399–423. ↑ 88

[141] R. Lercier and F. Vercauteren, *Discrete logarithms in* $\mathrm{GF}(p^{18})$ *- 101 digits*, 2005. Email to the `NMBRTHRY` mailing-list. ↑ 25

[142] P. Leyland, A. K. Lenstra, B. Dodson, A. Muffett, and S. S. Wagstaff Jr., *MPQS with three large primes*. In C. Fieker and D. R. Kohel (eds.), ANTS-V, vol. 2369 of *Lecture Notes in Comput. Sci.*, 448–462. Springer–Verlag, 2002. 5th Algorithmic Number Theory Symposium, Sydney, Australia, July 2002. ↑ 95

[143] P. Longa and C. H. Gebotys, *Efficient techniques for high-speed elliptic curve cryptography*. In S. Mangard and F.-X. Standaert (eds.), CHES, vol. 6225 of *Lecture Notes in Comput. Sci.*, 80–94. Springer–Verlag, 2010. ↑ 87, 163, 182

[144] D. Lorenzini, *An invitation to arithmetic geometry*, Grad. Stud. Math., vol. 9, Amer. Math. Soc., 1996. ↑ 35

[145] J. L. Massey, *Shift-register synthesis and BCH decoding*, IEEE Trans. Inform. Theory **IT–15**(1) (1969), 122–127. ↑ 129

[146] R. Matsumoto, *Using C_{ab} curves in the function field sieve*, IEICE Trans. Fundamentals **E82-A**(3) (1999), 551–552. ↑ 36

[147] Д. В. Матюхин, *Об асимптотической сложности дискретного логарифмирования в поле* $\mathrm{GF(p)}$, Дискрет. матем. **15**(1) (2003), 28–49.

[148] D. V. Matyukhin, *On asymptotic complexity of computing discrete logarithms over* $\mathrm{GF}(p)$, Discrete Math. Appl. **13**(1) (2003), 27–50.

[149] U. M. Maurer and S. Wolf, *Diffie-Hellman oracles*. In N. Koblitz (ed.), Advances in Cryptology – CRYPTO '96, vol. 1109 of *Lecture Notes in Comput. Sci.*, 268–282. Springer–Verlag, 1996. Proc. 16th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 1996. ↑ 6

[150] A. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*, CRC Press, 1997. ↑ 5

[151] J.-F. Mestre, *Utilisation de l'AGM pour le calcul de $E(F_{2^n})$*, 2000. Lettre adressé à Gaudry et Harley – `http://www.math.jussieu.fr/~mestre/lettreGaudryHarley.ps`. ↑ 87

[152] V. Miller, *Use of elliptic curves in cryptography*. In A. M. Odlyzko (ed.), Advances in Cryptology – CRYPTO '86, vol. 263 of *Lecture Notes in Comput. Sci.*, 417–426. Springer–Verlag, 1987. Proc. 7th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 1986. ↑ 8

[153] N. Mitchell, L. Carter, and J. Ferrante, *Localizing non-affine array references*, Parallel Architectures and Compilation Techniques '99, 1999. ↑ 139

[154] M. Mohiyuddin, M. Hoemmen, J. Demmel, and K. Yelick, *Minimizing communication in sparse matrix solvers*, SC '09, 36 :1–36 :12. ACM, 2009. Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis. ↑ 144

[155] C. Monico, `ggnfs`, *A Number Field Sieve Implementation*, 2004–2005. Available at `http://www.math.ttu.edu/~cmonico/software/ggnfs/`. Release `0.77`. ↑ 23

[156] P. L. Montgomery, *Modular multiplication without trial division*, Math. Comp. **44**(170) (1985), 519–521. ↑ 157

[157] P. L. Montgomery, *Square roots of products of algebraic numbers*. In W. Gautschi (ed.), Mathematics of Computation 1943–1993 : a Half-Century of Computational Mathematics, vol. 48 of *Proc. Sympos. Appl. Math.*, 567–571. Amer. Math. Soc., 1994. ↑ 50, 63, 202

[158] P. L. Montgomery, *A block Lanczos algorithm for finding dependencies over* $\mathrm{GF}(2)$. In L. C. Guillou and J.-J. Quisquater (eds.), Advances in Cryptology – EUROCRYPT '95, vol. 921 of *Lecture Notes in Comput. Sci.*, 106–120, 1995. Proc. International Conference on the Theory and Application of Cryptographic Techniques, Saint-Malo, France, May 1995. ↑ 121

[159] P. L. Montgomery, *Square roots of products of algebraic numbers*, 1997. Unpublished draft, significantly different from published version [157]. Dated May 16, 1997. ↑ 50, 51, 63

[160] F. Morain, *Analyzing PMPQS*, 1993. Available at `http://www.lix.polytechnique.fr/~morain/Articles/pmpqs.ps.gz`. Informal note. ↑ 92

[161] M. A. Morrison and J. Brillhart, *A method of factoring and the factorization of* F_7, Math. Comp. **29**(129) (1975), 183–205. ↑ 13, 92

[162] B. Murphy, *Polynomial selection for the number field sieve integer factorisation algorithm*, PhD Thesis, Australian National University, 1999. ↑ 19, 75

[163] A. Muzereau, N. P. Smart, and F. Vertauteren, *The equivalence between the DHP and DLP for elliptic curves used in practical applications*, LMS J. Comput. Math. 7 (2004), 50–72. ↑ 6

[164] P. Q. Nguyen, *A Montgomery-like square root for the number field sieve*. In J. P. Buhler (ed.), ANTS-III, vol. 1423 of *Lecture Notes in Comput. Sci.*, 151–168. Springer–Verlag, 1998. 3rd Algorithmic Number Theory Symposium, Portland, Oregon, USA, June 21–25, 1998. ↑ 50

[165] National Institute of Standards and Technology. *Digital Signature Standard (DSS)*, FIPS-186-3, `http://csrc.nist.gov/publications/fips`. Third revision, 2009. ↑ 150

[166] A. M. Odlyzko, *Discrete logarithms in finite fields and their cryptographic significance*. In T. Beth, N. Cot, and I. Ingemarsson (eds.), Advances in Cryptology – EUROCRYPT '84, vol. 209 of *Lecture Notes in Comput. Sci.*, 224–314. Springer–Verlag, 1985. Proc. Eurocrypt '84, Paris (France), April 9–11, 1984. ↑ 112

[167] P. C. van Oorschot and M. J. Wiener, *Parallel collision search with cryptanalytic applications*, J. Cryptology **12** (1999), 1–28. ↑ 149

[168] D. Panario, X. Gourdon, and P. Flajolet, *An analytic approach to smooth polynomials over finite fields*. In J. P. Buhler (ed.), ANTS-III, vol. 1423 of *Lecture Notes in Comput. Sci.*, 226–236. Springer–Verlag, 1998. 3rd Algorithmic Number Theory Symposium, Portland, Oregon, USA, June 21–25, 1998. ↑ 112

[169] J. Papadopoulos, `msieve`, *A Library for Factoring Large Integers – `release 1.50`*, 2004–. Available at `http://www.boo.net/~jasonp`. Release `1.50`. ↑ 23

[170] O. Penninga. *Finding column dependencies in sparse matrices over* $\mathbb{F}_2$ *by block Wiedemann*, Report MAS-R9819, Centruum voor Wiskunde en Informatica, Amsterdam, The Netherlands, `http://repository.cwi.nl/search/fullrecord.php?publnr=4642`, 1998. ↑ 135

[171] D. Pointcheval and J. Stern, *Security proofs for signature schemes*. In Ueli Maurer (ed.), Advances in Cryptology - EuroCrypt '96, vol. 1070 of *Lecture Notes in Comput. Sci.*, 387–398. Springer–Verlag, 1996. ↑ 59

[172] J. M. Pollard, *Monte Carlo methods for index computation* $(\mathrm{mod}\ p)$, Math. Comp. **32**(143) (1978), 918–924. ↑ 98

[173] J. M. Pollard, *Factoring with cubic integers*. In A. K. Lenstra and H. W. Lenstra Jr (eds.), The development of the number field sieve, vol. 1554 of *Lecture Notes in Math.*, 4–10. Springer–Verlag, 1993. ↑ 13

[174] J. M. Pollard, *The lattice sieve*. In A. K. Lenstra and H. W. Lenstra Jr (eds.), The development of the number field sieve, vol. 1554 of *Lecture Notes in Math.*, 43–49. Springer–Verlag, 1993. ↑ 34, 37, 44, 63, 76

[175] C. Pomerance, *The quadratic sieve algorithm*. In T. Beth, N. Cot, and I. Ingemarsson (eds.), Advances in Cryptology – EUROCRYPT '84, vol. 209 of *Lecture Notes in Comput. Sci.*, 169–182. Springer–Verlag, 1985. Proc. Eurocrypt '84, Paris (France), April 9–11, 1984. ↑ 13, 20

[176] C. Pomerance, *A tale of two sieves*, Notices Amer. Math. Soc **43** (1996), 1473–1485. ↑ 13

[177] C. Pomerance and J. W. Smith, *Reduction of huge, sparse matrices over finite fields via created catastrophes*, Experiment. Math. **1**(2) (1992), 89–94. ↑ 33

[178] R. L. Rivest, A. Shamir, and L. M. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Comm. ACM **21**(2) (1978), 120–126. ↑ 3

[179] D. Robert, *Fonctions thêta et applications à la cryptographie*, Thèse, Université Henri Poincaré, 2010. ↑ 189

[180] T. Satoh, *The canonical lift of an ordinary elliptic curve over a finite field and its point counting*, J. Ramanujan Math. Soc. **15** (2000), 247–270. ↑ 87

[181] O. Schirokauer, *Discrete logarithms and local units*, Philos. Trans. Roy. Soc. London Ser. A **345**(1676) (1993), 409–423. ↑ 18, 25, 26, 27, 28, 30, 31, 46, 68

[182] O. Schirokauer, *Using number fields to compute logarithms in finite fields*, Math. Comp. **69**(231) (1999), 1267–1283. ↑ 25, 46

[183] O. Schirokauer, *The special function field sieve*, SIAM J. Discrete Math. **16**(1) (2002), 81–98. ↑ 46

[184] O. Schirokauer, *Virtual logarithms*, J. Algorithms **57**(2) (2005), 140–147. ↑ 25, 29, 33

[185] O. Schirokauer, *The impact of the number field sieve on the discrete logarithm problem in finite fields*, Algorithmic Number Theory : Lattices, Number Fields, Curves and Cryptography, 2008, pp. 397–420. ↑ 25, 26

[186] O. Schirokauer, *The number field sieve for integers of low weight*, Math. Comp. **79**(269) (2010), 583–602. ↑ 25

[187] O. Schirokauer, D. Weber, and T. F. Denny, *Discrete logarithms : the effectiveness of the index calculus method*. In H. Cohen (ed.), ANTS-II, vol. 1122 of *Lecture Notes in Comput. Sci.*, 337–361. Springer–Verlag, 1996. 2nd Algorithmic Number Theory Symposium, Talence, France, May 18–23, 1996. ↑ 13, 25

[188] B. Schmidt, H. Aribowo, and H.-V. Dang, *Iterative Sparse Matrix-Vector Multiplication for Integer Factorization on GPUs*. In E. Jeannot, R. Namyst, and J. Roman (eds.), Euro-Par 2011 Parallel Processing, part II, vol. 6853 of *Lecture Notes in Comput. Sci.*, 413–424. Springer–Verlag, 2011. Proc. 17th International Conference, Euro-Par 2011, Bordeaux, France, August 29 - September 2, 2011. ↑ 188

[189] A. Schönhage, *Schnelle Multiplikation von Polynomen über Körpern der Charakteristik* 2, Acta Inform. 7 (1977), 395–398. `http://cr.yp.to/bib/1977/schoenhage.html`. ↑ 165, 169

[190] A. Schönhage and V. Strassen, *Schnelle Multiplikation großer Zahlen*, Computing **7** (1971), 281–292. ↑ 169

[191] M. Scott, *MIRACL : Multiprecision Integer and Rational Arithmetic C/C++ Library*, 1988–. Homepage at `http://www.shamus.ie/`. ↑ 156

[192] M. Scott, *New record breaking implementations of ECC on quadratic extensions using endomorphisms*, 2008. Invited talk at the ECC 2008 Conference. Utrecht, the Netherlands, Sep. 22-24, 2008. ↑ 87, 163, 182

[193] I. A. Semaev, *An algorithm for evaluation of discrete logarithms in some nonprime finite fields*, Math. Comp. **67**(224) (1998), 1679–1689. ↑ 25

[194] I. A. Semaev, *Special prime numbers and discrete logs in finite prime fields*, Math. Comp. **71**(237) (2002), 363–377. ↑ 26, 46

[195] N. Shinohara, T. Shimoyama, T. Hayashi, and T. Takagi, *Key length estimation of pairing-based cryptosystems using η_T pairing*, 2012. Available at `http://eprint.iacr.org/2012/042`. ↑ 185

[196] V. Shoup, *NTL : A library for doing number theory*, 1990–. Homepage at `http://www.shoup.net/ntl/`. ↑ 155, 165

[197] V. Shoup, *OAEP reconsidered*, J. Cryptology **15**(4) (2002), 223–249. ↑ 3, 59

[198] J. H. Silverman, *The arithmetic of elliptic curves*, Grad. Texts in Math., vol. 106, Springer–Verlag, 1986. ↑ 83, 84, 85

[199] R. D. Silverman, *Optimal Parameterization of SNFS*, J. Math. Cryptol. **1**(2) (2007), 105–124. ↑ 20

[200] B. Smith, *Isogenies and the discrete logarithm problem in Jacobians of genus 3 hyperelliptic curves*. In N. Smart (ed.), Advances in Cryptology – EUROCRYPT 2008, vol. 4965 of *Lecture Notes in Comput. Sci.*, 163–180. Springer–Verlag, 2008. ↑ 110, 182

[201] P. Stach, *Optimizations to NFS Linear Algebra*, CADO workshop on integer factorization, Nancy, France, 2008. Available at `http://cado.gforge.inria.fr/workshop`. ↑ 188

[202] M. Streng, *Complex multiplication of abelian surfaces*, PhD Thesis, Universiteit Leiden, 2010. ↑ 188

[203] A. V. Sutherland, *Computing Hilbert class polynomials with the chinese remainder theorem*, Math. Comp. **80**(273) (2011), 501–538. ↑ 52

[204] A. V. Sutherland, *Accelerating the CM method*, 2011. Available at `http://arxiv.org/abs/1009.1082`. Preprint. ↑ 52

[205] R. G. Swan, *Factorization of polynomials over finite fields*, Pacific J. Math. **12** (1962), 1099–1106. ↑ 100

[206] E. Teske, *On random walks for Pollard's rho method*, Math. Comp. **70**(234) (2001), 809–825. ↑ 98

[207] E. Thomé, *Fast computation of linear generators for matrix sequences and application to the block Wiedemann algorithm*. In B. Mourrain (ed.), ISSAC 2001, 323–331. ACM Press, 2001. Proc. International Symposium on Symbolic and Algebraic Computation, July 22–25, 2001, London, Ontario, Canada. ↑ 129 [*clé dans le texte :* [T1]]

[208] E. Thomé, *Computation of discrete logarithms in* $\mathbb{F}_{2^{607}}$. In C. Boyd and E. Dawson (eds.), Advances in Cryptology – ASIACRYPT 2001, vol. 2248 of *Lecture Notes in Comput. Sci.*, 107–124. Springer–Verlag, 2001. Proc. 7th International Conference on the Theory and Applications of Cryptology and Information Security, Dec. 9–13, 2001, Gold Coast, Queensland, Australia. ↑ 119, 127 [*clé dans le texte :* [T2]]

[209] E. Thomé, *Discrete logarithms in* $\mathrm{GF}(2^{607})$, 2002. Email to the `NMBRTHRY` mailing-list. ↑ 25, 129, 135

[210] E. Thomé, *Subquadratic computation of vector generating polynomials and improvement of the block Wiedemann algorithm*, J. Symbolic Comput. **33**(5) (2002), 757–775. ↑ 119, 129, 131, 183 [*clé dans le texte :* [T3]]

[211] E. Thomé, *Algorithmes de calcul de logarithme discret dans les corps finis*, Thèse, École polytechnique, 2003. ↑ 43, 92, 121 [*clé dans le texte :* [T4]]

[212] E. Thomé, *Square Root Algorithms for the Number Field Sieve*. In F. Özbudak and F. Rodríguez-Henríquez (eds.), WAIFI 2012, vol. 7369 of *Lecture Notes in Comput. Sci.*, 208–224. Springer–Verlag, 2012. July 16-19, 2012. Bochum, Germany. ↑ 47 [*clé dans le texte :* [T18]]

[213] N. Thériault, *Index calculus attack for hyperelliptic curves of small genus*. In C. Laih (ed.), Advances in Cryptology – ASIACRYPT 2003, vol. 2894 of *Lecture Notes in Comput. Sci.*, 75–92. Springer–Verlag, 2003. Proc. 9th International Conference on the Theory and Applications of Cryptology and Information Security, Nov. 30 – Dec. 4, 2003, Taipei, Taiwan. ↑ 90, 92

[214] F. Vercauteren, *Computing zeta functions of curves over finite fields*, PhD thesis, Katholieke Universiteit Leuven, 2003. ↑ 88

[215] R. Vuduc, J. W. Demmel, and K. A. Yelick, *OSKI : Optimized Sparse Kernel Interface*, 2007. Version 1.0.1h, `http://bebop.cs.berkeley.edu/oski/`. ↑ 139

[216] R. Vuduc, J. W. Demmel, K. A. Yelick, S. Kamil, R. Nishtala, and B. Lee, *Performance optimizations and bounds for sparse matrix-vector multiply*, High Performance Networking and Computing, 1–35, 2002. Proceedings of the 2002 ACM/IEEE Conference on Supercomputing. ↑ 139

[217] D. H. Wiedemann, *Solving sparse linear equations over finite fields*, IEEE Trans. Inform. Theory **IT–32**(1) (1986), 54–62. ↑ 20, 122

[218] T. Wollinger, J. Pelzl, and C. Paar, *Cantor versus Harley : Optimization and Analysis of Explicit Formulae for Hyperelliptic Curve Cryptosystem*, IEEE Trans. Comput. **54**(7) (2005), 861–872. ↑ 98, 99, 100

[219] G. Woltman, `GIMPS`, *The great Internet Mersenne prime search*, 1996–. `http://www.mersenne.org/`. ↑ 151

[220] P. Zimmermann and B. Dodson, *20 Years of ECM*. In F. Hess, S. Pauli, and M. E. Pohst (eds.), ANTS-VII, vol. 4076 of *Lecture Notes in Comput. Sci.*, 525–542. Springer–Verlag, 2006. 7th Algorithmic Number Theory Symposium, Berlin, Germany, July 23-28, 2006. ↑ 43

Bibliographie personnelle

Travaux de thèse

[T1] E. Thomé, *Fast computation of linear generators for matrix sequences and application to the block Wiedemann algorithm*. In B. Mourrain (ed.), ISSAC 2001, 323–331. ACM Press, 2001. Proc. International Symposium on Symbolic and Algebraic Computation, July 22–25, 2001, London, Ontario, Canada. ↑ 129

[T2] E. Thomé, *Computation of discrete logarithms in* $\mathbb{F}_{2^{607}}$. In C. Boyd and E. Dawson (eds.), Advances in Cryptology – ASIACRYPT 2001, vol. 2248 of *Lecture Notes in Comput. Sci.*, 107–124. Springer–Verlag, 2001. Proc. 7th International Conference on the Theory and Applications of Cryptology and Information Security, Dec. 9–13, 2001, Gold Coast, Queensland, Australia. ↑ 119, 127

[T3] E. Thomé, *Subquadratic computation of vector generating polynomials and improvement of the block Wiedemann algorithm*, J. Symbolic Comput. **33**(5) (2002), 757–775. ↑ 119, 129, 131, 183

[T4] E. Thomé, *Algorithmes de calcul de logarithme discret dans les corps finis*, Thèse, École polytechnique, 2003. ↑ 43, 92, 121

Travaux post-thèse

[T5] P. Gaudry and E. Thomé, *The mpFq library and implementing curve-based key exchanges*, SPEED : Software Performance Enhancement for Encryption and Decryption, 49–64, 2007. ↑ 1, 155, 159, 163

[T6] H. Cheng, G. Hanrot, E. Thomé, E. Zima, and P. Zimmermann, *Time- and Space-Efficient Evaluation of Some Hypergeometric Constants*. In C. W. Brown (ed.), ISSAC 2007, 85–91. ACM Press, 2007. Proc. International Symposium on Symbolic and Algebraic Computation, July 29–August 1st, 2007, London, ON. ↑ 181

[T7] P. Gaudry, E. Thomé, N. Thériault, and C. Diem, *A double large prime variation for small genus hyperelliptic index calculus*, Math. Comp. **76**(257) (2007), 475–492. ↑ 1, 83, 91, 95, 96, 97, 104

[T8] A. Joux, D. Naccache, and E. Thomé, *When e-th roots become easier than factoring*. In K. Kurosawa (ed.), Advances in Cryptology – ASIACRYPT 2007, vol. 4833 of *Lecture Notes in Comput. Sci.*, 13–28. Springer–Verlag, 2008. Proc. 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007. ↑ 1, 4, 47, 59, 60, 68, 69, 70, 119, 185

[T9] C. Diem and E. Thomé, *Index calculus in class groups of non-hyperelliptic curves of genus three*, J. Cryptology **21**(4) (2008), 593–611. ↑ 1, 83, 103, 105, 106, 108, 109, 110, 182

[T10] R. Brent, P. Gaudry, E. Thomé, and P. Zimmermann, *Faster Multiplication in GF(2)[x]*. In A. van der Poorten and A. Stein (eds.), ANTS-VIII, vol. 5011 of *Lecture Notes in Comput. Sci.*, 153–166. Springer–Verlag, 2008. ↑ 1, 165, 166, 167, 173

[T11] S. Burckel, E. Gioan, and E. Thomé, *Mapping Computation with No Memory*, 8th International Conference on Unconventional Computation - UC09, vol. 5715 of *Lecture Notes in Comput. Sci.*, 85–97. Springer–Verlag, 2009. The original publication is available at www.springerlink.com. ↑ 181

[T12] A. Joux, R. Lercier, D. Naccache, and E. Thomé, *Oracle-assisted static Diffie-Hellman is easier than discrete logarithms*. In M. G. Parker (ed.), Cryptography and Coding 2009, vol. 5921 of *Lecture Notes in Comput. Sci.*, 351–367. Springer–Verlag, 2009. ↑ 1, 6, 59, 60, 70, 72, 119, 185

[T13] T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik, H. te Riele, A. Timofeev, and P. Zimmermann, *Factorization of a 768-bit RSA modulus*. In T. Rabin (ed.), Advances in Cryptology – CRYPTO 2010, vol. 6223 of *Lecture Notes in Comput. Sci.*, 333–350. Springer–Verlag, 2010. Proc. 30th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2010. ↑ 1, 4, 37, 47, 73, 119, 120, 129, 137

[T14] T. Kleinjung, L. Nussbaum, and E. Thomé, *Using a grid platform for solving large sparse linear systems over GF(2)*, 11th ACM/IEEE International Conference on Grid Computing (Grid 2010), 2010. ↑ 1, 78, 119, 144, 146

[T15] A. Enge, P. Gaudry, and E. Thomé, *An $L(1/3)$ discrete logarithm algorithm for low degree curves*, J. Cryptology **24**(1) (2011), 24–41. ↑ 1, 43, 45, 83, 111, 112, 113, 114, 115, 182, 185

[T16] T. Kleinjung, J. Bos, A. Lenstra, D. A. Osvik, K. Aoki, S. Contini, J. Franke, E. Thomé, P. Jermini, M. Thiémard, P. Leyland, P. Montgomery, A. Timofeev, and H. Stockinger, *A Heterogeneous Computing Environment to Solve the 768-bit RSA Challenge*, Cluster Comput. **15**(1) (2012), 53–68. ↑ 76, 77, 78, 119, 182

[T17] V. Cortier, J. Detrey, P. Gaudry, F. Sur, E. Thomé, M. Turuani, and P. Zimmermann, *Ballot stuffing in a postal voting system*. In IEEE (ed.), Revote 2011 - International Workshop on Requirements Engineering for Electronic Voting Systems, 27–36. IEEE, 2011.

[T18] E. Thomé, *Square Root Algorithms for the Number Field Sieve*. In F. Özbudak and F. Rodríguez-Henríquez (eds.), WAIFI 2012, vol. 7369 of *Lecture Notes in Comput. Sci.*, 208–224. Springer–Verlag, 2012. July 16-19, 2012. Bochum, Germany. ↑ 47

Édition d'ouvrages

[T19] G. Hanrot, F. Morain, and E. Thomé (eds.), *ANTS-IX*, Lecture Notes in Comput. Sci., vol. 6197, Springer–Verlag, 2010. 9th Algorithmic Number Theory Symposium, Nancy, France, July 19-23, 2010.

Mes publications personnelles peuvent être téléchargées à l'adresse :
`http://www.loria.fr/~thome/publis.html`

Résumé

Le problème de la factorisation et celui du logarithme discret sont deux fondements essentiels de nombreux algorithmes de la cryptographie à clé publique. Dans le champ des algorithmes pour attaquer ces problèmes éminemment ardus, le crible algébrique et ses algorithmes cousins occupent une place de première importance.

La première partie de ce mémoire est consacrée à la présentation de la « famille » du crible algébrique, et à plusieurs de mes contributions dans ce domaine. D'autres travaux sont abordés dans la partie suivante, notamment en lien avec le problème du logarithme discret sur les jacobiennes de courbes, et à ma contribution à de nouveaux algorithmes pour ce problème dans certains cas particuliers.

La partie 3 du mémoire aborde mes travaux sur le thème de l'algèbre linéaire creuse sur les corps finis, motivés par le contexte d'application des algorithmes précédemment cités. La partie 4, enfin, traite de mes travaux dans le domaine de l'arithmétique, notamment concernant l'arithmétique des polynômes sur $\mathrm{GF}(2)$. La proximité des travaux apparaissant dans ces parties 3 et 4 avec des problématiques d'implantation indique le souci permanent, dans mes travaux, de ne pas laisser de côté cet aspect.

Abstract

The integer factorization and discrete logarithm problems are cornerstones of several public-key cryptography algorithms. In the realm of algorithms targeted at solving these highly difficult challenges, the number field sieve algorithm and its siblings are of utmost importance.

Part 1 of this work presents the family of algorithms around the number field sieve, together with several personal contributions in this research area. Other works are detailed in part 2, notably in relationship with the discrete logarithm problem on Jacobians of curves, and my contributions to this problem in some special cases.

Some aspects of my contributions on sparse linear algebra in finite fields, motivated by the aforementioned algorithms, are discussed in part 3 of this work. Part 4 covers my research on computer arithmetic, and in particular efficient arithmetic for binary polynomials. Parts 3 and 4 of this work emphasize a strong connection with the goal of efficient implementation.