UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine
D. F. D. Informatique

Algorithmes et arithmétique pour l'implémentation de couplages cryptographiques

Thèse

présentée et soutenue publiquement le 30 octobre 2013
pour l'obtention du

Doctorat de l'Université de Lorraine
(mention Informatique)

par

Nicolas Estibals

devant un jury composé de

Président :	Bernard Girau	Professeur de l'Université de Lorraine
Rapporteurs :	Anwar Hasan	Professeur à l'Université de Waterloo, Canada
	Reynald Lercier	Ingénieur de l'armement à la DGA et chercheur associé de l'Université de Rennes 1
Examinateurs :	Jean-Claude Bajard	Professeur de l'Université Pierre et Marie Curie
	Fabien Laguillaumie	Professeur à l'Université de Lyon 1
	Arnaud Tisserand	Chargé de Recherche au CNRS
Directeurs :	Jérémie Detrey	Chargé de Recherche à Inria
	Pierrick Gaudry	Directeur de Recherche au CNRS

Loria
Laboratoire lorrain de recherche en informatique et ses applications

UMR 7503

CARAMEL

Remerciements

Je tiens tout d'abord à remercier Anwar Hasan et Reynald Lercier d'avoir accepté la tâche qu'a constitué la relecture de ce manuscrit. Je remercie également Jean-Claude Bajard, Bernard Girau, Fabien Laguillaumie et Arnaud Tisserand d'avoir bien voulu participer à mon jury de thèse.

Toute ma reconnaissance va en premier lieu à mes directeurs de thèse, Jérémie Detrey et Pierrick Gaudry. Ils ont su me guider dans mes recherches avec une grande disponibilité. Ce manuscrit leur doit beaucoup par les nombreuses corrections et conseils qui m'ont aidé à en améliorer la présentation. J'ai eu avec eux de nombreux échanges toujours passionnants, scientifiques bien évidemment mais aussi amicaux quand ils n'étaient pas « trollesques ». Jérémie, Pierrick, merci !

Mes premiers pas dans la recherche ont été accompagnés par Nicolas Brisebarre. Il a compris très vite mes goûts scientifiques, attisé ma curiosité et m'a orienté vers la cryptographie et l'implémentation matérielle. Je l'ai retrouvé à Lyon et son soutien a été sans faille lors de la rédaction de ce document. Pour tout ceci et surtout tout le reste, Nicolas, je vous remercie.

Nicolas m'a également présenté Jean-Luc Beuchat avec qui j'ai pu faire mon premier stage de recherche à l'université de Tsukuba. Jean-Luc m'a confié un sujet très intéressant, m'initiant ainsi aux couplages qui allaient plus tard faire l'objet de ma thèse. Merci aussi pour la découverte du Japon et de 蛇の目寿司.

Je tiens également à remercier mes coauteurs : Diego F. Aranha, Razvan Barbulescu, Francico Rodríguez-Henríquez, Paul Zimmermann et Eiji Okamoto avec qui j'ai toujours un grand plaisir à travailler.

Je remercie aussi tous les membres de l'équipe CARAMEL dont l'accueil, l'enthousiasme et les pauses-café toujours très animées ont largement contribué à rendre mon séjour nancéen très enrichissant. Je pense en particulier à mes « grands frères » : Romain Cosset qui a considérablement complété ma formation mathématique lors de nos discussions sur le chemin du retour, Damien Robert et son habileté aux casse-têtes, Gaëtan Bisson et son inénarrable amitié, ainsi qu'aux « petits frères » : Cyril Bouvier et Hamza Jeljeli. Merci encore à Emmanuel Thomé aux geekeries récréatives[1] et à Marion Videau pour son soutien. Je remercie Stéphane Glondu pour toutes ces distractions qu'apportaient nos discussions de cobureaux.

J'ai eu un grand plaisir à rejoindre le LORIA, je ne pourrais citer ici toutes les personnes que j'ai eu la chance d'y croiser mais je tiens quand même à remercier les équipes administratives et de direction qui font vivre ce laboratoire très accueillant, un merci particulier à Emmanuelle Deschamps qui m'a assisté dans les préparations de mes missions ainsi qu'aux membres du conseil du laboratoire qui n'ont pas hésité à me montrer les dessous de son fonctionnement. Le LORIA m'a aussi offert des rencontres amicales, merci à Vincent Nivoliers

1. Souvent utiles comme les bouts de codes LaTeXqui ont servi à cette thèse.

et Maxime Rio pour les nombreuses escapades culturelles et culinaires, les picnics du mercredi, et les soirées à refaire le monde ; à quand la prochaine palette de Sonia ? Merci aussi à Matthieu, Aurélien (le « jeune »), Hugo, Olivier, Jean-Christophe, ...

J'ai eu l'occasion, pour ma dernière année de thèse, de redécouvrir les membres de l'équipe Aric ; j'avais déjà eu la chance de suivre leurs cours dans mes années d'études lyonnaises et j'ai beaucoup apprécié de me retrouver de « l'autre coté de la barrière » avec eux. J'ai une pensée spéciale pour Guillaume Hanrot que j'avais déjà côtoyé à Nancy, Bruno Salvy aux défis mathématiques percutants pendant les pauses-café et à tout le groupe de travail sur les réseaux euclidiens, en particulier Damien Stehlé qui m'a fait découvrir ces objets passionnants, ainsi qu'à Adeline, Marc, Nicolas, Rishiraj, et Serge qui ont partagé mon bureau. Je remercie également toute l'équipe enseignante du DI qui m'a beaucoup aidé à pleinement développer mes enseignements.

Je remercie grandement encore Francisco avec qui j'ai échangé dès le début de ma thèse. Il m'a invité à passer un séjour au CINVESTAV à Mexico dont je garde un souvenir impérissable, en grande partie grâce à ses étudiants Nidia Cortez-Duarte, Jorge E. González-Díaz, René Henríquez et les autres qui nous ont montré les trésors du Mexique. ¡Muchas gracias !

J'ai rejoint les communautés scientifiques des algorithmiciens de l'arithmétique et des cryptographes. Je remercie les organisateurs des RAIM, de l'école ARCHI, des journées C2 et des JNCF d'avoir permis ces rencontres scientifiques, ainsi que les équipes de Rennes, Versailles, Caen et Lyon de m'avoir invité à leurs séminaires. Je remercie en particulier Laurent Imbert, Vanessa Vitse dont le manuscrit de thèse si clair a aidé à la rédaction du mien, Peter Schwabe et sa grande sympathie, Sorina Ionica ainsi que tous les autres.

Merci à mes étudiants de l'ÉSIAL et de l'ÉNS de Lyon qui ont supporté mes premiers pas dans l'enseignement avec curiosité...

Merci aux « matheux d'en face » : Arnaud et Julie pour leur nombreuses soirées sans jeux, Michaël et son humour décalé pendant les formations CIES, Romain et ses propositions festives, Paul et son barbecue d'expert, Estelle et les spaghetti basilic... Je remercie en particulier Armand dont l'amitié m'est maintenant précieuse.

Merci encore aux membres du Nybi Construction Club d'avoir détourné chaque mercredi soir mon attention de ma thèse.

Je remercie aussi tous ceux que j'ai croisés à Lyon, en particulier Bruno, sa « grande gueule » et sa présence, Pascal un peu brutal mais toujours sincère, Nico et nos échanges profonds, Flo et ses traités d'œnologie alternative, Manu, Issac, Julien, Pierre et les zébus.

Merci à tous ceux que j'ai connus plus jeune et qui sont toujours là : Vince, Pooki, Matmat, Vivien, Hélène, Kéké, Joe et tous les Moulins, Dom, Blandine, Stib, Greg, Inès, Fred, ... Un merci particulier à Tod qui a partagé une coloc joyeuse avec moi et supporté les humeurs conséquentes à une journée de rédaction ; merci aussi Arakis ! Enfin, je remercie chaleureusement Loïc pour son amitié et nos bricolages de plus en plus fous !

Un clin d'œil aussi à Spag et Didje pour les concerts de punk !

Je remercie aussi mes parents qui, au delà de leur générosité, m'ont toujours entouré. Patrick, Élise, merci et à vous de jouer !

Il y a aussi tous ceux que j'ai oubliés lors de ce difficile exercice qu'est l'écriture de remerciements mais qui, j'en suis sûr, se reconnaîtront et pardonneront ma maladresse.

Enfin je conclues ces remerciements par celle qui a égayé ma thèse et à qui ce manuscrit doit beaucoup de par sa grande patience et ses nombreuses corrections. Merci Aurelia pour ta présence qui me réconforte toujours, les joies que tu m'inspires et notre complicité toujours renouvelée.

Table des matières

Chapitre 1

Introduction

Nous vivons, depuis quelques décennies, une époque de développement rapide des technologies numériques qui prennent désormais une part substantielle, si ce n'est essentielle, de nos activités quotidiennes. Conséquemment, nous observons actuellement une massification et une intensification des **échanges numériques**. Dès lors, la **sécurisation** de ces flux de données devient un enjeu sociétal et économique. En effet, la dématérialisation de nos échanges doit garantir, *a minima*, les mêmes propriétés que leurs analogues physiques. Pensons en particulier au respect de la vie privée lorsque le courrier devient électronique, à la confiance en un commerçant quand il ouvre un site marchand ou encore à celle en une banque en ligne. Ces garanties peuvent être apportées par la **cryptographie** qui donne une réponse mathématique au problème d'assurer, entre autres, la confidentialité, l'authenticité et l'intégrité des messages empruntant des voies de communication non sûres telles que le réseau Internet.

De plus, la sécurité des échanges numériques est un enjeu qui dépasse le cadre de nos ordinateurs personnels. En effet, nous avons besoin d'assurer cette sécurité dans de nombreux objets du quotidien : des systèmes embarqués à la puissance de calcul limité. S'il est courant de penser en premier lieu à la carte bancaire, d'autres exemples sont peut-être encore plus percutant : comment mettre à jour de façon sûre le calculateur du système de freins d'une voiture ? ou encore les paramètres d'un pacemaker ?

La cryptographie est une science multi-millénaire dont les premières traces remontent à l'antiquité, initialement pour des applications militaires. Il faut cependant attendre 1883 pour qu'elle se dote d'un cadre formel [Ker83a ; Ker83b]. A. Kerckhoffs introduit dans ses articles la **notion de clé** : le système cryptographique (entendre par là l'algorithme, le circuit ou bien encore la machine) est supposé connu, et la sécurité repose uniquement sur le fait qu'un paramètre du système, la clé, reste secret ; ce principe est toujours valable de nos jours.

Les premiers cryptosystèmes qui sont apparus fonctionnent sur le même principe : l'expéditeur et le destinataire s'échangent au préalable une clé secrète partagée, cette clé servant alors à la fois au chiffrement et au déchiffrement. De tels cryptosystèmes sont dits **symétriques**.

Cependant, l'échange préalable des clés secrètes est problématique car il demande à la fois que les interlocuteurs aient disposé d'un canal de communication privé et qu'ils soient en mesure de stocker de manière sécurisée l'ensemble des clés de leurs correspondants potentiels. Prenons l'exemple des cartes bancaires : il n'est pas envisageable que chaque distributeur de billets ait une clé pour chaque carte produite, et encore moins que chaque carte connaisse les clés de tous les distributeurs en service à travers le monde. Ce problème de **distribution**

des clés a été résolu en 1976 lorsque W. Diffie et M. Hellman ont montré comment rompre la symétrie des précédents cryptosystèmes [DH76] : la clé comporte une partie publique avec laquelle tout le monde peut chiffrer un message ou vérifier une signature, mais seul le possesseur de la partie secrète de la clé peut déchiffrer ou construire une signature valide. En plus de proposer des alternatives à clé publique pour le chiffrement et la signature, W. Diffie et M. Hellman ont conçu un protocole permettant à deux parties de s'accorder sur une clé privée commune sans que leurs échanges ne révèlent d'information sur celle-ci. Ainsi, il devient possible d'utiliser les algorithmes de cryptographie symétrique sans qu'il soit nécessaire de partager un secret commun au préalable. L'avantage de cette approche hybride est de permettre l'utilisation des primitives de cryptographie symétrique généralement plus rapides en éliminant le besoin de l'échange de clé préalable.

Qu'il s'agisse de systèmes symétriques ou asymétriques, leur sécurité repose sur l'**impossibilité calculatoire** de décrypter un message ou de forger une signature sans la clé privée. Le niveau de sécurité d'un cryptosystème, mesuré en bits, représente l'ordre de grandeur du nombre d'opérations élémentaires nécessaires pour briser celui-ci à l'aide du meilleur algorithme connu. Attaquer un système ayant un niveau de sécurité de n bits demandera ainsi de l'ordre de 2^n opérations. À l'heure actuelle, la plupart des institutions gouvernementales (NIST, ANSSI, BSI) recommandent au minimum **128 bits de sécurité**[1].

Cryptographie à clé publique

Cette thèse prend pour cadre la **cryptographie à clé publique**. Afin de créer l'asymétrie, il faut, en général, se ramener à des **problèmes mathématiques** dont la construction d'une instance et la vérification d'une solution sont calculatoirement aisées mais dont la résolution est difficile. Le plus connu de ces problèmes est certainement la **factorisation des entiers** sur laquelle repose la sécurité du système RSA [RSA78]. La clé secrète y est constituée de deux nombres premiers et la clé publique est le produit de ceux-ci.

Un autre problème utilisé en cryptographie asymétrique est celui du logarithme discret. Il est presque aussi connu et a été introduit quelques années avant dans [DH76].

Définition 1.1 (Problème de logarithme discret)**.** Soient $\mathbb{G}$ un groupe cyclique d'ordre ℓ noté multiplicativement, g un générateur de ce groupe et $h \in \mathbb{G}$. Le **problème du logarithme discret** — *discrete logarithm problem* en anglais (DLP) — consiste à calculer $a \in \mathbb{Z}/\ell\mathbb{Z}$ tel que $h = g^a$ étant donnée seulement la connaissance de g et h. Nous notons alors $\mathrm{dlog}_g(h) = a$.

Dans les faits, les problèmes durs sur lesquels reposent ces protocoles sont au plus aussi sûrs que le DLP, et potentiellement plus faibles pour certains. Cependant, les problèmes usuellement employés sont conjecturés équivalents au DLP, comme par exemple le problème de Diffie-Hellman calculatoire.

Définition 1.2. Soient $\mathbb{G} = \langle g \rangle$ d'ordre ℓ, ainsi que $a, b \in \mathbb{Z}/\ell\mathbb{Z}$. Le **problème de Diffie-Hellman calculatoire** — *computational Diffie–Hellman problem* (CDH) — consiste à calculer g^{ab} étant donnée seulement la connaissance de g, g^a et g^b.

Il existe également une version décisionnelle de CDH appelée le problème de Diffie-Hellman décisionnel (DDH). Celle-ci est parfois plus facile que la version calculatoire.

1. Les diverses recommandations académiques ou institutionnelles sont regroupées sur le site http://keylength.com

Le DLP est un problème qu'il faut instancier en choisissant un groupe particulier. En effet, ce problème est trivial pour certains groupes ($(\mathbb{Z}/\ell\mathbb{Z}, +)$ par exemple). W. Diffie et M. Hellman ont proposé initialement d'utiliser le groupe multiplicatif d'un corps fini $(\mathbb{F}_q^*, \times)$ [DH76]. Ce groupe est toujours utilisé en pratique pour l'authentification sur les serveurs SSH et la signature des courriers électroniques par PGP par exemple.

Au milieu des années 1980, la communauté de théorie algorithmique des nombres travaille activement sur les courbes elliptiques : H. Lenstra crée une méthode de factorisation reposant sur ces courbes (ECM) [Len87], S. Goldwasser et J. Kilian les utilisent pour tester et prouver la primalité [GK86]. C'est dans ce contexte que V. Miller et N. Koblitz ont indépendamment proposé d'utiliser comme groupe cyclique $\mathbb{G}$ le groupe des points d'une courbe elliptique [Mil86b ; Kob87] avec l'idée que le DLP sur les courbes elliptiques est un problème calculatoirement plus dur que celui introduit par W. Diffie et M. Hellman.

Rapidement, N. Koblitz a étendu cette idée aux jacobiennes de courbes hyperelliptiques [Kob89]. Plus tard, des structures dérivées des groupes algébriques ont été utilisées, comme les variétés de trace nulle [Fre01] ou les surfaces de Kummer [SS99].

D'autres objets mathématiques furent également utilisés pour créer des cryptosystèmes à clé publique. Parmi ceci, nous pouvons citer les codes correcteurs [McE78], le problème du sac à dos [MH78], les systèmes polynomiaux [Pat96], les groupes non-commutatifs. Cependant, certains de ces derniers, notamment les systèmes reposant sur les groupes de tresses [AAG99], ne sont maintenant plus utilisables étant donné que des attaques efficaces existent.

Enfin, les problèmes liés aux réseaux euclidiens offrent une voie très prometteuse. Elle s'est ouverte en 1996 lorsque M. Atjai a donné la première réduction pire cas/moyen cas pour différents problèmes durs sur les réseaux euclidiens [Ajt96]. Parallèlement, des efforts ont été faits pour fournir des systèmes cryptographiques praticables avec la conception de NTRU [HPS98]. Récemment, les réseaux euclidiens se sont illustrés dans leurs utilisations cryptographiques d'une part en donnant lieu à des schémas reposant sur des preuves de sécurité utilisant des hypothèses faibles [Reg09], et d'autre part en permettant la résolution du problème du chiffrement complètement homomorphe [Gen09].

Couplages cryptographiques

De façon très remarquable, les courbes elliptiques, et plus généralement les variétés jacobiennes, viennent avec une structure supplémentaire qui permet de définir des applications bilinéaires sur celles-ci. Ces applications, appelées **couplages**, sont l'objet principal de cette thèse.

Initialement, les couplages — *pairings* en anglais — ont été vus comme un outil pour attaquer les courbes elliptiques, et en particulier les courbes supersingulières. Ils ont ainsi d'abord été introduits à la communauté des cryptologues par les attaques Menezes-Okamoto-Vanstone [MOV93] et Frey-Rück [FR94].

Dans un second temps, A. Joux a proposé d'utiliser les **couplages de manière constructive** [Jou00 ; Jou04]. Dès lors, un grand nombre de protocoles cryptographiques avancés ont vu le jour. Nous pensons tout d'abord au **chiffrement reposant sur l'identité** [BF01 ; BF03] qui était un problème ouvert depuis que A. Shamir l'avait posé [Sha85] et à la **signature courte** [BLS01 ; BLS04]. Depuis, ils ont permis d'obtenir des propriétés plus complexes que simplement l'authenticité et la confidentialité au sein de cryptosystèmes plus ambitieux tels que les systèmes de vote ou de monnaie électronique.

Ces avancées s'expliquent par le fait que les couplages apportent une fonctionnalité importante à la cryptographie sur les groupes : **combiner deux secrets sans avoir à les révéler**. Cet aspect s'exprime plus abstraitement dans la définition formelle d'un couplage cryptographique ci-après.

Définition 1.3 (Couplage cryptographique)**.** Soient $(\mathbb{G}_1, +), (\mathbb{G}_2, +)$ deux groupes d'exposant ℓ et $(\mathbb{G}_T, \times)$ un groupe cyclique d'ordre ℓ. Un **couplage cryptographique** est une application

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$$

vérifiant les propriétés suivantes.

(*i*) **Non-dégénérescence** : pour tout $u \in \mathbb{G}_1$, il existe $v \in \mathbb{G}_2$ tel que $e(u, v) \neq 1_{\mathbb{G}_T}$.

(*ii*) **Bilinéarité** : pour tous $u, u' \in \mathbb{G}_1$ et tous $v, v' \in \mathbb{G}_2$,

$$\begin{cases} e(u + u', v) = e(u, v) \cdot e(u', v) \text{ et} \\ e(u, v + v') = e(u, v) \cdot e(u, v'). \end{cases}$$

(*iii*) **Calculabilité** : il existe un algorithme efficace calculant e.

En cryptographie, les couplages utilisés sont dérivés des couplages de Weil et de Tate sur les courbes elliptiques et hyperelliptiques. Les groupes $\mathbb{G}_1$ et $\mathbb{G}_2$ sont alors des groupes de points et $\mathbb{G}_T$ le groupe multiplicatif d'un corps fini.

Afin de construire des protocoles reposant sur des couplages, il faut *a minima* supposer que le DLP est dur dans les trois groupes $\mathbb{G}_1, \mathbb{G}_2$ et $\mathbb{G}_T$. Cependant, cette hypothèse n'est pas suffisante pour construire des protocoles cryptographiques et nous distinguons trois types de couplages qui permettront d'utiliser différentes hypothèses de sécurité et d'exploiter différentes fonctionnalités. Nous avons ainsi les couplages

— de **type I**, ou symétriques, lorsque $\mathbb{G}_1 = \mathbb{G}_2$ ou lorsqu'il existe un isomorphisme explicite entre ces deux groupes ;
— de **type II** si le seul homomorphisme calculable efficacement est de $\mathbb{G}_2$ dans $\mathbb{G}_1$; et
— de **type III** s'il n'existe aucun homomorphisme explicite entre les deux groupes.

Selon leur type, les fonctionnalités et les hypothèses de sécurité associées aux couplages sont différentes. Par exemple, s'il existe un couplage de type I de $\mathbb{G}_1 \times \mathbb{G}_1$ dans $\mathbb{G}_T$, le problème de Diffie-Hellman décisionnel devient trivial dans $\mathbb{G}_1$.

Certains protocoles parmi les plus fameux (Diffie-Hellman tripartite [Jou00 ; Jou04] et chiffrement basé sur l'identité [BF01 ; BF03]) utilisent des couplages de type I et l'hypothèse calculatoire associée la plus classique est le **problème de Diffie-Hellman bilinéaire** (BDH).

Définition 1.4 (BDH)**.** Soit P un générateur de $\mathbb{G}_1$ sur lequel est construit $e(.,.)$, un couplage de type I. Le problème de Diffie-Hellman bilinéaire consiste à calculer $e(P, P)^{abc}$ étant donnés P, aP, bP, cP pour des entiers a, b, c inconnus.

L'hypothèse de la difficulté de BDH est plus forte que celle du DLP dans $\mathbb{G}_1$ et $\mathbb{G}_T$; en effet, s'il est possible de résoudre le problème du logarithme discret dans ces groupes, alors il suffit d'une exponentiation pour résoudre BDH.

D'autres protocoles sont écrits dans un cadre plus général et requièrent, selon le cas, des couplages de type II ou III. Il existe alors une très grande variété d'hypothèses calculatoires sur lesquelles s'appuient ces protocoles. Il est difficile de donner une comparaison explicite de ces hypothèses de sécurité car leur difficulté relative n'est souvent que conjecturée. Cependant

le dénominateur commun est toujours que le DLP doit être dur dans chacun des groupes concernés.

Nous donnons l'exemple d'une de ces hypothèses : co-DH est difficile. C'est l'une des premières utilisées, notamment par Boneh-Lynn-Shacham dans leur protocole de signature courte [BLS01 ; BLS04]. Nous donnons ici la version calculatoire de ce problème.

Définition 1.5 (co-DH)**.** Soient P un générateur de $\mathbb{G}_1$ et Q un générateur de $\mathbb{G}_2$. Le problème co-DH consiste à calculer aQ étant donnés P, aP et Q pour un entier a inconnu.

Bien que ce problème ne fasse pas intervenir directement de couplage dans sa définition, l'existence d'un tel couplage pourrait permettre une attaque plus directe que de simplement calculer le logarithme discret de aP en base P et de calculer ensuite aQ. L'hypothèse co-DH est ainsi une hypothèse de sécurité concernant la structure apportée par le couplage.

Objectifs et orientation de la thèse

Le but de cette thèse est de fournir des implémentations efficaces de couplages à 128 bits de sécurité. Cette finalité pose des questions multiples tant au niveau du choix des courbes à utiliser, des couplages à calculer, des algorithmes à mettre en œuvre que du choix des cibles et plateformes pour l'implémentation. En effet, calculer un couplage à ce niveau de sécurité demande une quantité d'opérations non négligeable qu'il convient d'optimiser pour rendre le coût d'utilisation de couplages en cryptographie raisonnable.

Par exemple, l'intégration de primitives cryptographiques à des systèmes embarqués représente un défi du point de vue de différentes contraintes : surface limitée des puces concernées, consommation électrique réduite avec malgré tout le besoin de performances raisonnables. Qui attendrait plus de quelques secondes l'autorisation de retrait devant un distributeur automatique de billets ? Au vue de ces contraintes, il est difficile d'envisager des implémentations logicielles embarquées de primitives cryptographiques, en particulier de couplages.

Il est également un autre domaine où les implémentations matérielles sont très utiles : les protocoles utilisant les couplages font souvent intervenir un serveur qui, en pratique, devra interagir avec de nombreux clients. Dans ce cas, une implémentation logicielle, même pour des processeurs très performants, peut s'avérer insuffisante pour supporter la charge de calcul. Ici, un coprocesseur cryptographique spécifique peut être nécessaire pour atteindre le débit attendu.

Pour ces raisons, nous avons choisi de nous concentrer sur les **implémentations matérielles**, et plus particulièrement sur FPGA — *Field-Programmable Gate Array*. Ces **circuits reconfigurables** ont l'intérêt d'être un outil de prototypage rapide évitant certaines des difficultés inhérentes à la description de circuits intégrés spécifiques. Ils nous ont ainsi permis de tester nos idées d'implémentations et de nous comparer au reste de la littérature. De plus, leur reconfigurabilité leur confère également un intérêt propre : ils sont ainsi très adaptés aux cas d'un coprocesseur spécifique pour un serveur dont les besoins peuvent changer au cours du temps (évolution des algorithmes et des tailles de clés par exemple).

Ensuite, nous devons identifier des courbes munies de couplages dont le calcul sera rapide. La courbe associée à un couplage détermine également son type (I, II ou III). Devant la grande variété des courbes adaptées aux couplages, nous nous sommes restreints aux courbes supersingulières car leurs propriétés mathématiques sont particulièrement intéressantes vis à vis des couplages. Premièrement, elles sont les seules à fournir des couplages symétriques (ou

de type I). Deuxièmement, l'arithmétique de ces courbes est particulière et efficace et nous espérons bénéficier de cet aspect dans nos implémentations. Enfin, les courbes supersingulières intéressantes d'un point de vue cryptographique sont définies sur des corps de petite caractéristique (2 ou 3) ; cela signifie que l'arithmétique de ce corps ne demande aucune propagation de retenue et est ainsi adaptée aux implémentations matérielles.

Contributions

Cette thèse regroupe un certain nombre de contributions au domaine des couplages cryptographiques. Tout d'abord, nous donnons une présentation unifiée des couplages utilisés dans des implémentations cryptographiques. Jusqu'alors, chaque couplage (Eta, Eta T, Ate, et leurs variantes « optimales ») bénéficiait de constructions *ad hoc* dont la présentation se limitait à une classe de courbes limitées. Nous donnons dans ce manuscrit une construction unique intégrant chacune des techniques ayant permis la découverte de ces couplages pour les courbes elliptiques et hyperelliptiques.

Nous avons ensuite appliqué ce cadre général à des couplages connus. Cela nous a permis, par exemple, de montrer le fait apparemment méconnu que les couplages Ate et Eta en caractéristiques 2 et 3 ont exactement la même complexité.

Nous avons également appliqué ce cadre général aux courbes hyperelliptiques supersingulières de genre 2 et de caractéristique 2. En collaboration avec D. Aranha, J.-L. Beuchat et J. Detrey, nous avons pu en déduire un nouvel algorithme plus efficace pour calculer un couplage sur ces courbes [Ara+12].

L'efficacité d'une implémentation reposant en majeure partie sur l'implémentation de l'arithmétique sous-jacente, nous avions besoin de ce fait d'algorithmes de multiplication sous-quadratique à la Karatsuba. C'est pourquoi nous avons également travaillé sur le problème du rang bilinéaire. Nous avons ainsi conçu un algorithme de recherche exhaustive de formules de multiplication dont nous avons amélioré la complexité. Cela nous a permis de contribuer au domaine de la complexité algébrique en prouvant le rang de certaines formes bilinéaires en caractéristique 2 et 3. Ces travaux ont été faits en collaboration avec R. Barbulescu, J. Detrey et P. Zimmermann ; ils sont présentés dans [Bar+12].

Enfin, nous avons réalisé des implémentations FPGA de tous les algorithmes de couplages étudiés. En collaboration avec J.-L. Beuchat, J. Detrey, E. Okamoto et F. Rodríguez-Henríquez, nous avons conçu dans un premier temps un accélérateur *ad-hoc* parallèle donnant des temps de calcul très compétitifs [Beu+09a ; Beu+11]. Cependant, il a fallu concevoir une architecture radicalement différente pour réaliser la première implémentation matérielle de couplage à 128 bits de sécurité [Est10] et d'obtenir ses performances qui sont restées les meilleures jusqu'à récemment. Nous avons également implémenté notre nouvel algorithme pour le genre 2 [Ara+12] qui permet d'atteindre des performances très comparables à celles des autres accélérateurs pour courbes supersingulières malgré la complexité accrue des algorithmes.

Avertissement

Depuis février 2013, la communauté de théorie algorithmique des nombres a connu un regain d'activités important sur le problème du logarithme discret pour le groupe multiplicatif des corps finis de petite caractéristique. Ces travaux ont commencé initialement en utili-

sant des idées récemment développées pour le logarithme discret en moyenne caractéristique (depuis décembre 2012) [Jou12 ; Jou13a].

Dès lors, l'activité sur ce sujet s'est intensifiée et de nombreux records de calcul de logarithme discret en caractéristique 2 ont été annoncés sur la liste de diffusion NMBRTHRY [Jou12 ; Jou13a ; Jou13b ; Gra+13a ; Jou13c ; Gra+13b ; Jou13d]. Ces améliorations successives ont enfin conduit à l'élaboration d'un algorithme de complexité quasi-polynomiale pour le logarithme discret en petite caractéristique [Bar+13b]. Ces avancées récentes remettent en cause une partie des estimations de complexité utilisées dans nos travaux.

Cependant, la communauté travaille toujours très activement sur ce problème et il est encore trop tôt pour en mesurer les conséquences et faire de nouvelles estimations précises, spécialement pour la caractéristique 3 pour laquelle aucune implémentation n'a été réalisée pour le moment. Par conséquent, toutes les estimations de sécurité données dans ce document doivent être comprises comme des estimations valables avant l'année 2013.

Nous nous attendons néanmoins à ce que ce nouvel algorithme ait un impact très fort sur la cryptographie reposant sur les couplages, et plus particulièrement lorsqu'il est fait usage de courbes supersingulières comme c'est le cas dans cette thèse.

Nous ne mentionnerons plus ces travaux récents dans ce document, si ce n'est par des notes aux endroits concernés renvoyant au présent paragraphe.

Organisation du document

La réalisation d'une implémentation de primitive cryptographique consiste en une progression partant de la définition mathématique des objets manipulés jusqu'à la conception des opérateurs. Elle demande de mettre en adéquation des choix mathématiques, algorithmiques, arithmétiques et architecturaux. Nous avons structuré ce document à l'image de cette progression.

Nous avons ainsi regroupé dans une première partie les définitions et constructions des courbes elliptiques et hyperelliptiques (Chapitre 2) et des couplages (Chapitre 3) avant de mettre en perspective l'utilisation de ceux-ci en cryptographie (Chapitre 4).

Dans une seconde partie, nous avons abordé les aspects calculatoires et algorithmiques des couplages : d'abord en exposant les techniques classiques de calcul de couplages (Chapitre 5), permettant alors de construire, conjointement avec leurs algorithmes de calcul, d'autres couplages plus efficaces (Chapitre 6). Nous détaillons alors ces constructions et algorithmes pour les trois familles de courbes supersingulières qui ont retenu notre attention (Chapitre 7).

Afin de pouvoir implémenter efficacement ces algorithmes, il est fondamental d'étudier l'arithmétique des corps finis que nous manipulons (Chapitre 8). Ce fut l'occasion de découvrir un nouvel algorithme de recherche exhaustive de formules de multiplication et de l'appliquer aux besoins de nos algorithmes de couplage (Chapitre 9).

Dans une dernière partie, après une brève introduction à la conception de circuits (Chapitre 10), nous décrivons les aspects architecturaux de l'ensemble de nos implémentations matérielles. Celles-ci se séparent en deux familles : les accélérateurs *ad-hoc* parallèles (Chapitre 11) et ceux reposant sur un coprocesseur arithmétique (Chapitre 12). Nous discutons enfin de ces architectures et de leur positionnement par rapport à celles de la littérature (Chapitre 13).

Première partie

Courbes elliptiques et hyperelliptiques

Chapitre 2

Courbes elliptiques et hyperelliptiques

Nous étudierons dans ce chapitre les fondements mathématiques des objets que nous utiliserons dans cette thèse. Comme nous l'avons vu dans l'introduction, un des moyens d'obtenir une cryptographie asymétrique est d'utiliser des groupes sur lesquels le problème du logarithme discret est difficile ou tout du moins considéré comme tel. Les courbes elliptiques sont actuellement très utilisées pour obtenir des primitives pour la cryptographie à clé publique ; leur utilisation a été proposé par Koblitz en 1987 [Kob87]. Par la suite, les cryptographes ont commencé à se servir d'autres objets mathématiques tels que les courbes hyperelliptiques ou encore les variétés abéliennes qui forment le cadre le plus général des groupes issus de la géométrie algébrique utilisés en cryptographie [Kob89].

Nous nous sommes tout particulièrement intéressés durant cette thèse à la construction et au calcul de couplages. Ces applications bilinéaires peuvent être construites sur tous les objets que nous avons précédemment cités, mais nous nous restreindrons aux cas que nous utiliserons dans ce manuscrit : les courbes elliptiques et hyperelliptiques. Cependant, la construction de couplages (Chapitre 3), particulièrement de couplages calculables efficacement (Chapitre 6), et le choix des courbes adaptées (Chapitre 4) demandent d'utiliser un grand nombre de propriétés des objets algébriques sur lesquels ils sont construits. C'est pourquoi nous nous attarderons dans ce chapitre à décrire les fondements théoriques qui conduisent à l'obtention des courbes elliptiques et hyperelliptiques, ainsi que leurs propriétés mathématiques.

Ce chapitre s'organise ainsi dans ce but. Nous commençons par décrire les **variétés algébriques** qui forment le cadre le plus général des objets que nous manipulerons, ainsi que leurs propriétés essentielles à la définition et au calcul de couplages (Section 2.1). Nous appliquerons alors deux restrictions et montrerons les propriétés qu'elles induisent. Dans un premier temps nous nous restreindrons aux **courbes algébriques** (c'est-à-dire aux variétés de dimension 1), ceci nous permettra d'introduire les groupes que nous utiliserons pour tous les couplages considérés ici : les **jacobiennes de courbes**. Ce sont des variétés abéliennes construites à partir de courbes (Section 2.2). Enfin, nous introduirons l'hypothèse d'hyperellipticité : elle peut se résumer dans l'essentiel à l'existence d'une **involution hyperelliptique** sur les points de la courbe. Elle permet de développer une arithmétique efficace simplifiant la manipulation des points de la courbe et de sa jacobienne (Section 2.3).

Nous étudierons dans ce chapitre des objets algébriques, définis sur un corps K que nous considérerons dans la suite parfait et commutatif. En pratique, la cryptographie se limite à l'emploi de **corps finis** $\mathbb{F}_q$ afin de bénéficier d'une représentation efficace et ces hypothèses

sont alors vérifiées. Cependant, nous travaillerons également avec des extensions de ce corps, telles que sa clôture algébrique $\overline{\mathbb{F}_q}$ ou une extension finie $\mathbb{F}_{q^k}$. Nous appellerons alors K le corps de base.

2.1 Variétés algébriques et abéliennes

2.1.1 Variétés algébriques

Nous introduisons ici la notion de variété algébrique. Elle permet de formaliser la géométrie des solutions d'un système d'équations polynomiales. Ces variétés définissent ainsi la notion de points — les solutions d'un tel système — qui sont les éléments les plus élémentaires avec lesquels nous travaillerons. Cette notion nous permet d'introduire une première difficulté associée à l'étude des ces objets algébriques : nous avons besoin, selon le contexte, de se placer dans un espace **affine** ou un espace **projectif**. Nous ne donnerons ici aucune preuve, mais nous invitons le lecteur à consulter [Sil86] pour celles-ci, une grande partie des propositions étant tirées de cette référence.

§ 1. Variété affine. Nous commençons par donner la définition d'un espace affine.

Définition 2.1 (Espace affine)**.** L'espace affine de dimension n sur K, noté $\mathbb{A}^n(K)$, est l'ensemble des n-uplets sur K :

$$\mathbb{A}^n(K) = \{P = (x_1, \ldots, x_n) \mid x_i \in K\}\,.$$

Nous travaillerons essentiellement dans ce chapitre dans la clôture algébrique de K et nous noterons $\mathbb{A}^n = \mathbb{A}^n(\overline{K})$ lorsque le contexte est clair. Afin de définir formellement l'ensemble des points affines solutions d'un système polynomial, nous nous intéressons à l'idéal I engendré par les polynômes du système considéré, et nous supposerons que cet idéal est premier.

Définition 2.2 (Variété affine)**.** Une variété affine $\mathcal{V}$ sur $\overline{K}$ est un ensemble de points

$$\mathcal{V} = \{P \in \mathbb{A}^n \mid \forall f \in I, f(P) = 0\}\,,$$

où I est un **idéal premier** de l'anneau des polynômes à n variables $\overline{K}[X_1, \ldots, X_n]$.

Si $\mathcal{V}$ est une variété affine, l'idéal qui la définit est donné par

$$I(\mathcal{V}) = \left\{ f \in \overline{K}[X_1, \ldots, X_n] \;\middle|\; \forall P \in \mathcal{V}, f(P) = 0 \right\}.$$

§ 2. Variété projective. Certaines propriétés des objets que nous allons manipuler nécessitent de nous placer dans un contexte projectif. Pour cela, il est nécessaire de se limiter à l'usage de **polynômes homogènes**.

Définition 2.3 (Polynôme homogène)**.** Soit $f \in \overline{K}[X_0, \ldots, X_n]$, f est un polynôme homogène de degré d si

$$\forall \lambda \in \overline{K}, f(\lambda x_0, \ldots, \lambda x_n) = \lambda^d \cdot f(x_0, \ldots, x_n).$$

De façon équivalente, un polynôme f est homogène de degré d s'il est une **somme de monômes de degré** d.

Nous construisons alors l'**espace projectif** dans lequel les variétés du même type seront définies. Cet espace est tel que les **zéros** des polynômes homogènes sont bien définis en tant que points projectifs.

Définition 2.4 (Espace projectif)**.** L'espace projectif de dimension n sur $\overline{K}$, noté $\mathbb{P}^n(\overline{K})$ ou $\mathbb{P}^n$, est l'ensemble des droites de $\overline{K}^{n+1}$ passant par l'origine. De façon plus pratique, cet espace est représenté par tous les points de $\overline{K}^{n+1}$ différents de l'origine et munis de la relation d'équivalence suivante :

$$(x_0, \ldots, x_n) \sim (x'_0, \ldots, x'_n) \Leftrightarrow \exists \lambda \in \overline{K}^* \text{ tel que } \forall i \in \{0, \ldots, n\}, x_i = \lambda \cdot x'_i.$$

La classe d'équivalence de $(x_0, \ldots, x_n)$ sera alors notée $(x_0 : \cdots : x_n)$.

Nous parlerons d'**idéal homogène** si celui-ci peut être généré par des polynômes homogènes. Remarquons alors que, pour un idéal homogène I de $\overline{K}[X_0, \ldots, X_n]$, nous pouvons définir

$$\mathcal{V}_I = \{P \in \mathbb{P}^n \mid \forall f \text{ homogène} \in I, f(P) = 0\}.$$

En effet, chaque point projectif est défini à un scalaire près, qui devient de même un facteur multiplicatif lorsque nous lui appliquons un polynôme homogène : l'appartenance à $\mathcal{V}_I$ ne dépend donc pas du choix de la représentation des points projectifs.

Nous pouvons maintenant définir l'équivalent projectif des variétés que nous avons vues précédemment.

Définition 2.5 (Variété projective)**.** Une variété projective $\mathcal{V}$ sur $\overline{K}$ est un ensemble de points

$$\mathcal{V} = \{P \in \mathbb{P}^n \mid \forall f \text{ homogène} \in I, f(P) = 0\},$$

où I est un **idéal homogène premier** de l'anneau des polynômes à $(n+1)$ variables $\overline{K}[X_0, \ldots, X_n]$.

Si $\mathcal{V}$ est une variété projective, l'idéal homogène qui la définit est donné par

$$\left\{ f \text{ homogène} \in \overline{K}[X_0, \ldots, X_n] \;\middle|\; \forall P \in \mathcal{V}, f(P) = 0 \right\}.$$

§ 3. Homogénéisation. Nous explicitons dans cette section les liens entre variétés projective et affine. Pour ce faire, nous commençons par associer un point projectif à un point affine relativement à la i-ième variable :

$$(x_1, \ldots, x_n) \longmapsto (x_1 : \cdots : x_i : 1 : x_{i+1} : \cdots : x_n) \ ;$$

ainsi que la transformation inverse pour les points de $\mathbb{P}^n$ dont la i-ième coordonnée est non nulle

$$(x_0 : \cdots : x_n) \longmapsto \left(\frac{x_0}{x_i}, \ldots, \frac{x_{i-1}}{x_i}, \frac{x_{i+1}}{x_i}, \ldots, \frac{x_n}{x_i} \right).$$

Ainsi, le passage des coordonnées projectives aux coordonnées affines ne permet plus de décrire les points projectifs dont la i-ième coordonnée est nulle. Ces points sont vus comme des **points à l'infini** selon cette projection relativement à la i-ième variable.

De même, il est possible d'homogénéiser un polynôme f à n variables par rapport à la i-ième variable :

$$X_i^d \cdot f\left(\frac{X_0}{X_i}, \ldots, \frac{X_{i-1}}{X_i}, \frac{X_{i+1}}{X_i}, \ldots, \frac{X_n}{X_i} \right),$$

où d est le degré de f (les dénominateurs disparaissent et nous obtenons alors un polynôme homogène). Par cette construction, si le point affine P est un zéro d'un polynôme f, l'homogénéisation de P est un zéro de l'homogénéisation de f.

En conséquence, ces morphismes permettent de passer d'une variété affine à une variété projective et inversement.

Définition 2.6 (Clôture projective)**.** Soit $\mathcal{V} \subset \mathbb{A}^n$ une variété affine, sa clôture projective $\overline{\mathcal{V}}$ relativement à la i-ième variable est la variété projective définie par l'idéal engendré par l'ensemble des homogénéisés de f relativement à la i-ième variable, où f décrit l'idéal $I(\mathcal{V})$.

Cette définition permet d'obtenir les propriétés naturelles auxquelles nous nous attendons : le projeté sur $\mathbb{A}^n$ de la clôture projective d'une variété affine $\mathcal{V}$ est cette même variété $\mathcal{V}$ et la clôture projective du projeté sur $\mathbb{A}^n$ d'une variété projective $\mathcal{V}$ est bien la variété $\mathcal{V}$.

Par la suite nous utiliserons ces deux représentations selon le contexte ; en effet, les variétés affines sont plus pratiques pour étudier les propriétés locales d'une variété alors que la vision projective permet d'en étudier les propriétés globales.

2.1.2 Corps de définition et K-rationalité

Jusqu'alors nous avons défini différents objets relativement à la clôture algébrique d'un corps K or, celle-ci n'étant pas finie, elle ne présente qu'un intérêt théorique et nous avons besoin de ramener ces définitions relativement au corps K dans un intérêt pratique. L'outil que nous utilisons pour ce faire est le groupe de Galois de $\overline{K}$ sur K.

Définition 2.7 (Groupe de Galois)**.** Le groupe de Galois d'une extension L sur K est l'ensemble des automorphismes de L qui laissent K invariant :

$$\mathbb{G}_{L/K} = \left\{ \sigma \in \operatorname{Aut} L \;\middle|\; \sigma_{|K} = \mathrm{Id}_{|K} \right\}.$$

Une variété affine $\mathcal{V}$ est définie sur K (ce que nous notons $\mathcal{V}/K$) si son idéal peut être engendré par des polynômes dans $K[X_1, \ldots, X_n]$. Dans ce cas, l'ensemble de ses points dits L-rationnels (où L est une extension algébrique de K) est

$$\mathcal{V}(L) = \left\{ P \in \mathcal{V} \;\middle|\; \forall \sigma \in \mathbb{G}_{\overline{K}/L}, P^\sigma = (\sigma(x_1), \ldots, \sigma(x_n)) = P \right\},$$

c'est-à-dire les points à coordonnées dans L : $\mathcal{V} \cap \mathbb{A}^n(L)$.

De même, nous nous intéressons aux variétés projectives. Cependant, il nous faut d'abord définir ce qu'il en est pour l'espace projectif :

$$\begin{aligned}
\mathbb{P}^n(L) &= \left\{ P \in \mathbb{P}^n \;\middle|\; \forall \sigma \in \mathbb{G}_{\overline{K}/L}, P^\sigma = P \right\} \\
&= \left\{ (x_0 : \cdots : x_n) \;\middle|\; \forall i, j, \frac{x_j}{x_i} \in L \text{ si } x_i \neq 0 \right\} \\
&= \{ (x_0 : \cdots : x_n) \mid \forall i, x_i \in L \}.
\end{aligned}$$

Ainsi, il s'agit de l'ensemble des points pour lesquels il existe une représentation n'utilisant que des coordonnées dans L.

Dès lors une variété projective $\mathcal{V}$ est définie sur K si son idéal peut être généré par des polynômes homogènes à coefficients dans K. L'ensemble des points L-rationnels est alors :

$$\mathcal{V}(L) = \mathcal{V} \cap \mathbb{P}^n(L).$$

Remarquons que le fait d'être défini sur K reste vrai après clôture projective. Notons également que si une variété affine ou projective est définie sur un corps K, elle l'est aussi sur toute extension L de K.

2.1.3 Corps de fonctions

Un des outils essentiels du calcul de couplages est la notion de **fonction rationnelle** ; en effet, ces fonctions permettent d'associer un scalaire à chaque point d'une variété sauf, éventuellement, un nombre fini d'entre eux. Elles sont ainsi nécessaires à l'écriture de formules pour le calcul de couplages. Plus généralement, les fonctions rationnelles forment un corps associé à sa variété et ce corps est un outil puissant pour l'étude de la variété correspondante. Nous donnons d'abord la définition du corps de fonctions d'une variété affine.

Définition 2.8 (Corps de fonctions)**.** Le corps de fonctions $\overline{K}(\mathcal{V})$ d'une variété affine $\mathcal{V}$ est le corps des fractions de son **anneau des coordonnées** :

$$\overline{K}(\mathcal{V}) = \operatorname{Frac} \overline{K}[\mathcal{V}] \text{ où } \overline{K}[\mathcal{V}] = \frac{\overline{K}[X_1, \ldots, X_n]}{I(\mathcal{V})}.$$

Comme l'idéal $I(\mathcal{V})$ est premier, l'anneau $\overline{K}[\mathcal{V}]$ est intègre et son corps de fonctions existe.

Pour une variété projective $\mathcal{V}$, il est possible de définir son corps de fonctions comme celui de la variété $\mathcal{V} \cap \mathbb{A}^n$, nous allons cependant en donner une définition équivalente plus explicite.

Définition 2.9. Étant donnée $\mathcal{V}$ une variété projective, son corps de fonctions $\overline{K}(\mathcal{V})$ est l'ensemble des fractions $F(X_0, \ldots, X_n) = f(X_0, \ldots, X_n)/g(X_0, \ldots, X_n)$ telles que
(*i*) $f, g \in \overline{K}[X_0, \ldots, X_n]$,
(*ii*) f et g sont homogènes de même degré, et
(*iii*) $g \notin I(\mathcal{V})$,
muni de la relation d'équivalence suivante :

$$\frac{f}{g} \sim \frac{f'}{g'} \Leftrightarrow fg' - f'g \in I(\mathcal{V}).$$

Dans les deux cas affine et projectif, la définition du corps de fonctions est telle que l'évaluation de la fonction en un point de la variété est bien définie au sens où elle ne dépend pas de la représentation choisie pour la fonction.

Notons cependant que les fonctions rationnelles d'une variété projective ne sont pas nécessairement définies en tous ses points ; en effet, g peut s'annuler en certains points. Nous verrons à l'Exemple 2.37 comment caractériser ces points, que nous appellerons **pôles** de F, dans le cas particulier des courbes algébriques.

Si une variété $\mathcal{V}$ est définie sur K, il est alors possible de se restreindre à $K(\mathcal{V})$, la partie K-rationnelle de son corps de fonctions, en restreignant les polynômes à l'anneau $K[X_0, \ldots, X_n]$. Notons qu'une fois encore, ceci peut être caractérisé par l'action du groupe de Galois $\mathbb{G}_{\overline{K}/K}$: si $\sigma \in \mathbb{G}_{\overline{K}/K}$, f^σ est le polynôme ou la fonction rationnelle dont les coefficients sont les images par σ des coefficients de f. Le corps $K(\mathcal{V})$ est alors la partie de $\overline{K}(\mathcal{V})$ laissée invariante par $\mathbb{G}_{\overline{K}/K}$.

2.1.4 Singularité

Pour continuer l'étude des variétés nous avons besoin de définir leur dimension ; c'est un invariant qui correspond, dans le cas général, au nombre de variables libres dans les équations définissant la variété. Cette notion prendra toute son importance lorsque, dans la section suivante, nous nous restreindrons aux courbes, c'est-à-dire aux variétés de dimension 1.

Définition 2.10. La dimension d'une variété $\mathcal{V}$, notée $\dim \mathcal{V}$, est le degré de transcendance de $\overline{K}(\mathcal{V})$ par rapport à $\overline{K}$.

Cette dimension nous permet aussi de nous intéresser à la lissité, ou encore la non singularité, de la variété en un de ses points : un point $P = (x_1, \ldots, x_n)$ de $\mathcal{V}$ est lisse si et seulement si

$$\operatorname{rg}\left(\frac{\partial f_j}{\partial X_i}(P)\right)_{1\leq i\leq n, 1\leq j\leq m} = n - \dim \mathcal{V},$$

où la famille $(f_j)_{1\leq j\leq m}$ génère l'idéal de $\mathcal{V}$.

Cette propriété étant locale, la singularité d'un point sur une variété projective est définie en projetant la variété dans $\mathbb{A}^n$.

2.1.5 Applications entre variétés

Après avoir défini le corps de fonctions d'une variété, il est naturel de vouloir étendre les classes de fonctions que nous utilisons aux applications entre variétés. Celles-ci interviennent en plusieurs endroits de la construction des couplages. En premier lieu, nous construirons par la suite les variétés abéliennes en munissant d'une loi de groupe respectant les hypothèses d'applications entre variétés. De plus, cela nous permet de comparer des variétés en introduisant une notion d'isomorphisme. Enfin, lorsque nous nous restreindrons aux endomorphismes — particulièrement l'endomorphisme de Frobenius — nous déduirons la structure des groupes qui interviennent dans les couplages et bénéficierons d'un outil puissant pour réduire le temps de calcul de ceux-ci.

§ 1. Définition, régularité et morphisme. Ici nous nous éloignons du point de vue local et cherchons à considérer chaque variété dans son ensemble. Ainsi, il est plus naturel de travailler avec des variétés projectives.

Définition 2.11. Soient $\mathcal{V}_1, \mathcal{V}_2$ deux variétés projectives et n tel que $\mathcal{V}_2 \subset \mathbb{P}^n$. Une application rationnelle de $\mathcal{V}_1$ vers $\mathcal{V}_2$ est une application ϕ de la forme

$$\begin{array}{rccc} \phi : & \mathcal{V}_1 & \longrightarrow & \mathcal{V}_2 \\ & P & \longmapsto & (f_0(P) : \cdots : f_n(P)) \end{array}$$

où les f_i sont $n+1$ fonctions rationnelles de $\overline{K}(\mathcal{V}_1)$ telles que, en tout point P où toutes les f_i sont définies, $(f_0(P) : \cdots : f_n(P)) \in \mathcal{V}_2$.

S'il existe un scalaire $\lambda \in \overline{K}^*$ tel que $\lambda \cdot f_0, \ldots, \lambda \cdot f_n$ soient définies sur K, ϕ est alors **définie sur** K. Cette notion de rationalité sur K possède encore une fois une formulation équivalente en termes d'action du groupe de Galois $\mathbb{G}_{\overline{K}/K}$. Pour tout $\sigma \in \mathbb{G}_{\overline{K}/K}$, nous écrivons ϕ^σ l'application $(f_0^\sigma : \cdots : f_n^\sigma)$. L'application ϕ est alors définie sur K si et seulement si

$$\forall \sigma \in \mathbb{G}_{\overline{K}/K}, \phi^\sigma = \phi.$$

Il est possible de choisir d'autres représentations pour une même application ϕ. En effet, si $g \in \overline{K}(\mathcal{V}_1)$, $(f_0 : \cdots : f_n)$ et $(gf_0 : \cdots : gf_n)$ définissent la même application. De plus, ceci permet éventuellement d'évaluer ϕ en un point où l'une des fonctions f_i n'est pas définie. Nous dirons que ϕ est régulière en P s'il est possible d'évaluer ϕ en P, ce que nous formalisons par la définition suivante.

Définition 2.12 (Régularité en un point)**.** Soient $\phi : \mathcal{V}_1 \to \mathcal{V}_2, P \mapsto (f_0(P) : \cdots : f_n(P))$ et $P \in \mathcal{V}_1$. L'application ϕ est régulière en P s'il existe $g \in \overline{K}(\mathcal{V}_1)$ tel que :
(i) toutes les fonctions gf_i sont définies en P,
(ii) il existe i tel que $gf_i(P) \neq 0$.

La régularité en tout point est ainsi ce qui nous permet de mettre en relation deux variétés dans leur ensemble ; nous donnons alors les définitions naturelles de morphisme et isomorphisme de variétés qui s'ensuivent.

Définition 2.13 (Morphisme de variétés)**.** Une fonction rationnelle $\phi : \mathcal{V}_1 \to \mathcal{V}_2$ régulière en tout point est appelée un **morphisme**.

Définition 2.14 (Isomorphisme de variétés)**.** Deux variétés $\mathcal{V}_1$ et $\mathcal{V}_2$ sont isomorphes (ce que nous noterons $\mathcal{V}_1 \cong \mathcal{V}_2$) s'il existe deux morphismes $\phi : \mathcal{V}_1 \to \mathcal{V}_2$ et $\psi : \mathcal{V}_2 \to \mathcal{V}_1$ tels que $\psi \circ \phi$ et $\phi \circ \psi$ soient l'identité sur $\mathcal{V}_1$ et $\mathcal{V}_2$, respectivement.

Si ces morphismes sont définis sur K, $\mathcal{V}_1$ et $\mathcal{V}_2$ sont dites isomorphes sur K, ou encore K-isomorphes. Remarquons que si $\mathcal{V}_1$ et $\mathcal{V}_2$ sont $\overline{K}$-isomorphes, elle ne sont pas nécessairement K-isomorphes, nous dirons alors que $\mathcal{V}_1$ et $\mathcal{V}_2$ sont **tordues** l'une de l'autre — ou *twist* en anglais. Cette distinction prendra toute son importance lorsque que nous chercherons à calculer des couplages sur des courbes ordinaires (Section 4.1.3).

§ 2. Application surjective. Nous étudions maintenant le cas des applications surjectives entre variétés ; en effet, cette hypothèse nous permet d'ajouter un certain nombre de définitions et d'outils pour mieux les comprendre. Nous supposons ici que les variétés $\mathcal{V}_1$ et $\mathcal{V}_2$ sont définies sur K ; cependant, la surjectivité des applications que nous considérerons continue de s'entendre pour la clôture algébrique. La surjectivité d'une application rationnelle permet, de fait, de construire une application entre les corps de fonctions des variétés considérées et ainsi d'utiliser les outils de l'algèbre des corps pour poursuivre notre étude.

Définition 2.15 (Tiré en arrière — *pull-back*)**.** Soit une application rationnelle surjective $\phi : \mathcal{V}_1 \to \mathcal{V}_2$ définie sur K. Il est alors possible de construire une injection de $K(\mathcal{V}_2)$ dans $K(\mathcal{V}_1)$:

$$\begin{array}{rccc} \phi^* : & K(\mathcal{V}_2) & \longrightarrow & K(\mathcal{V}_1) \\ & f & \longmapsto & f \circ \phi. \end{array}$$

Cette construction se résume par le diagramme suivant :

$$\begin{array}{ccc} \mathcal{V}_1 & & \\ {\scriptstyle\phi}\downarrow & \searrow^{\phi^* f} & \\ \mathcal{V}_2 & \xrightarrow[f]{} & K \end{array}$$

Pour toute application rationnelle ϕ telle que définie précédemment, le corps de fonctions $K(\mathcal{V}_1)$ est une extension de corps finie de $\phi^* K(\mathcal{V}_2)$, ce qui nous permet de définir le **degré** de cette application ϕ.

Définition 2.16 (Degré d'une fonction rationelle)**.** Soit une application rationnelle $\phi : \mathcal{V}_1 \to \mathcal{V}_2$ définie sur K. Si ϕ n'est pas surjective, son degré est défini comme nul, sinon

$$\deg \phi = [K(\mathcal{V}_1) : \phi^* K(\mathcal{V}_2)].$$

Nous dirons que ϕ est **séparable**, **inséparable**, ou encore **purement inséparable** selon la séparabilité de l'extension de corps $K(\mathcal{V}_1)/\phi^* K(\mathcal{V}_2)$.

Proposition 2.17. *Soit une application rationnelle surjective $\phi : \mathcal{V}_1 \to \mathcal{V}_2$. L'application ϕ est un isomorphisme si et seulement si elle est de degré* 1*.*

2.1.6 Variétés abéliennes

Afin d'obtenir des primitives pour la cryptographie asymétrique, nous avons besoin de construire des **groupes**, ce que nous faisons en ajoutant une loi interne aux variétés. Là encore, nous donnons les résultats sans justification et renvoyons le lecteur à [Mil86c] pour les preuves.

Définition 2.18 (Variété abélienne)**.** Une variété abélienne $\mathcal{A}$ sur K est une variété projective non-vide (il y a au moins un point K-rationnel) définie sur K, munie d'une loi de groupe que nous noterons $+$ et telle que les applications $(P, Q) \mapsto P+Q$ et $P \mapsto -P$ sont des morphismes de variétés définis sur K.

Nous notons également $[x]$ la multiplication scalaire associée à cette loi de groupe :

$$\text{pour tout } x \in \mathbb{Z}, [x]P = \underbrace{P + \cdots + P}_{x \text{ fois}}.$$

Cette définition induit une structure algébrique riche dont nous pouvons extraire quelques propriétés remarquables.

Proposition 2.19. *Pour toute variété abélienne $\mathcal{A}$, $(\mathcal{A}, +)$ est un groupe commutatif.*

Proposition 2.20. *Si $\mathcal{A}$ est une variété abélienne, c'est aussi une variété non-singulière.*

Nous étendons alors les notions de morphisme et d'isomorphisme de variétés algébriques en ajoutant la condition d'être un homomorphisme de groupes pour devenir respectivement des morphisme et isomorphisme de variétés abéliennes.

2.1.7 Isogénies

Définition 2.21 (Isogénie)**.** Une isogénie est un morphisme surjectif de variétés abéliennes — c'est-à-dire un morphisme de variétés compatible avec la loi de groupe — dont le noyau est fini.

Si $\phi : \mathcal{A} \to \mathcal{B}$ est une isogénie, son degré en tant que morphisme de variétés correspond au cardinal de son noyau :

$$\deg \phi = \#\mathrm{Ker}(\phi).$$

De plus, $\mathcal{A}$ et $\mathcal{B}$ sont dites isogènes et elles sont de même dimension.

Théorème 2.22 (Isogénie duale)**.** *Pour toute isogénie $\phi : \mathcal{A} \to \mathcal{B}$, il existe une isogénie duale $\hat{\phi}$ telle que*

$$\hat{\phi} \circ \phi = [\deg \phi]_{\mathcal{A}} \text{ et } \phi \circ \hat{\phi} = [\deg \phi]_{\mathcal{B}}.$$

Notons que $\hat{\phi}$ est de même degré que ϕ.

Parmi les isogénies, nous trouvons par exemple la multiplication scalaire par ℓ ; celle-ci est de degré ℓ^{2g} quand le corps de définition K est de caractéristique 0 ou première avec ℓ.

Définition 2.23 (Endomorphisme)**.** Un morphisme d'une variété abélienne vers elle-même est appelé un endomorphisme.

Le composé de deux endomorphismes est de nouveau un endomorphisme. De plus, la loi de groupe permet de définir un nouvel endomorphisme en additionnant deux endomorphismes. Il n'est pas difficile de voir que, muni de ces deux lois de compositions internes $\circ$ et $+$, l'ensemble des endomorphismes d'une variété abélienne $\mathcal{A}$ est un anneau. Il est appelé **anneau d'endomorphismes** de $\mathcal{A}$ et noté $\mathrm{End}(\mathcal{A})$.

Les isogénies d'une variété abélienne $\mathcal{A}$ vers elle-même sont des endomorphismes, et en particulier, pour tout ℓ, la multiplication par ℓ appartient à $\mathrm{End}(\mathcal{A})$. Ainsi, l'anneau $\mathbb{Z}$ est toujours inclus dans $\mathrm{End}(\mathcal{A})$. En caractéristique 0, il y a parfois égalité, mais en caractéristique positive, ça n'est jamais le cas, comme mis en évidence dans le paragraphe suivant.

§ 1. Frobenius et Verschiebung. Nous supposons désormais que le corps dans lequel nous travaillons est de caractéristique non nulle. Une autre isogénie classique, dont nous nous servirons plus tard pour définir certains couplages, est le **morphisme de Frobenius** relatif à l'élévation à la puissance q, où q est une puissance de la caractéristique de K. Nous définissons ainsi

$$\begin{array}{rccc} \pi_q : & \mathcal{A} & \longrightarrow & \mathcal{A}^{(q)} \\ & (x_0 : \cdots : x_n) & \longmapsto & (x_0^q : \cdots : x_n^q), \end{array}$$

où $\mathcal{A}^{(q)}$ est la variété obtenue en élevant à la puissance q chacun des coefficients intervenant dans la définition de l'idéal. Remarquons que si $\mathcal{A}$ est définie sur $\mathbb{F}_q$, cette variété est identique à $\mathcal{A}$; π_q est alors un endomorphisme. Le Frobenius π_q est de degré q et son isogénie duale est appelée le **Verschiebung**. Notons également que le morphisme de Frobenius est **purement inséparable** ; en effet, si nous considérons $\pi_q^*(K(\mathcal{A}^{(q)})) = \left\{ f \circ \pi_q \;\middle|\; f \in K(\mathcal{A}^{(q)}) \right\}$, par linéarité de l'élévation à la puissance q, nous avons $\pi_q^*(K(\mathcal{A}^{(q)})) = \{f^q \mid f \in K(\mathcal{A})\} = K(\mathcal{A})^q$; ainsi nous déduisons que le morphisme de Frobenius est purement inséparable de degré q.

2.1.8 Structure de la ℓ-torsion

Soit $\mathcal{A}$ un variété abélienne sur un corps K. Nous cherchons maintenant à déduire des informations sur la **structure de groupe** de $\mathcal{A}$. Pour ce faire, nous considérons sa ℓ-torsion sur la clôture algébrique de K :

$$\mathcal{A}[\ell] = \{P \in \mathcal{A} \mid [\ell]P = 0_{\mathcal{A}}\} = \mathrm{Ker}\,[\ell].$$

D'un point de vue cryptographique, notre intérêt se porte sur ce sous-groupe ; en effet, nous cherchons à construire des groupes d'ordre premier.

La description de ce groupe diffère selon la primalité de ℓ avec p, la caractéristique de K.

Théorème 2.24. *Soient p premier, ℓ un entier tel que $p \nmid \ell$ et $\mathcal{A}$ une variété abélienne de dimension g sur un corps de caractéristique p. La structure de groupe de la ℓ-torsion est la suivante :*

$$\mathcal{A}[\ell] \cong (\mathbb{Z}/\ell\mathbb{Z})^{2g}.$$

Ce théorème permet de retrouver le degré de la multiplication scalaire par ℓ en tant qu'isogénie.

Lorsque nous nous intéressons à la p-torsion, la structure de groupe change et dépend de la variété considérée.

Théorème 2.25 (p-rang)**.** *Soit $\mathcal{A}$ une variété abélienne de dimension g définie sur un corps de caractéristique p. Le p-rang de $\mathcal{A}$ est l'entier r tel que*

$$\mathcal{A}[p] \cong (\mathbb{Z}/p\mathbb{Z})^{r}.$$

Cette entier est compris entre 0 et g.

Une variété de p-rang nul est dite **très spéciale**. À l'inverse, si le p-rang est maximal, nous disons qu'elle est **ordinaire**. Pour les cas intermédiaires, il n'y a pas de dénomination consacrée.

Si le corps de base K est un corps fini, nous pouvons combiner les résultats précédents avec le théorème de structure des groupes abéliens finis.

Théorème 2.26. *Soit $\mathcal{A}$ une variété abélienne de dimension g définie sur un corps fini K. L'ensemble de ses éléments K-rationnels admet la structure de groupe suivante :*

$$\mathcal{A}(K) \cong (\mathbb{Z}/n_1\mathbb{Z}) \oplus (\mathbb{Z}/n_2\mathbb{Z}) \oplus \cdots \oplus (\mathbb{Z}/n_{2g}\mathbb{Z}),$$

avec $n_1 \mid n_2 \mid \cdots \mid n_{2g}$.

2.1.9 Supersingularité

La notion de supersingularité sera cruciale dans nos travaux. Nous en donnons ici les définitions ainsi que les propriétés générales. D'autres propriétés, plus liées aux couplages, seront énoncées plus tard.

Définition 2.27. Soit $\mathcal{E}$ une variété abélienne de dimension 1 sur un corps de caractéristique p non-nulle. La variété $\mathcal{E}$ est dite supersingulière si son p-rang est nul, *i.e.* $\mathcal{E}[p] = \{0\}$.

Remarque 2.28. Nous verrons à la Section 2.3 que les variétés abéliennes de dimension 1 sont les courbes elliptiques, d'où la notation $\mathcal{E}$.

Le théorème suivant donne une caractérisation plus complète des variétés abéliennes supersingulières de dimension 1.

Théorème 2.29 (Variétés abéliennes supersingulières de dimension 1)**.** *Soit $\mathcal{E}$ une variété abélienne de dimension 1 sur un corps de caractéristique p. Les propriétés suivantes sont équivalentes :*

(i) *$\mathcal{E}$ est supersingulière ;*
(ii) *$\mathcal{E}$ est très spéciale : $\mathcal{E}[p] = \{0\}$;*
(iii) *$\forall r \in \mathbb{N}^*, \mathcal{E}[p^r] = \{0\}$;*

(*iv*) $\#\mathcal{E}(\mathbb{F}_q) \equiv 1 \pmod{p}$;
(*v*) $\mathrm{End}(\mathcal{E})$ *est non-commutatif ;*
(*vi*) *la multiplication par p est un endomorphisme purement inséparable.*

Nous pouvons désormais donner une définition de la supersingularité pour les variétés abéliennes de dimension supérieure.

Définition 2.30 (Variété abélienne supersingulière)**.** Une variété abélienne $\mathcal{A}$ de dimension g est **supersingulière** si et seulement si elle est isogène à la puissance g-ième d'une variété abélienne supersingulière de dimension 1.

Si de plus $\mathcal{A}$ est isomorphe à la puissance g-ième d'une variété abélienne supersingulière de dimension 1, alors $\mathcal{A}$ est dite **superspéciale**.

Remarquons que, contrairement à la dimension 1, les variétés abéliennes très spéciales ($\mathcal{A}[p] = \{0\}$) de dimension $g > 2$ ne sont pas nécessairement supersingulières.

Le Théorème 2.29 ne s'étend pas aux dimensions supérieures au sens nous n'avons plus d'équivalence mais une implication seulement.

Théorème 2.31. *Soit $\mathcal{A}$ une variété abélienne supersingulière. La variété $\mathcal{A}$ vérifie alors les propriétés équivalentes suivantes :*
(*i*) *$\mathcal{A}$ est très spéciale, c'est-à-dire de p-rang nul ;*
(*ii*) $\mathrm{End}(\mathcal{A})$ *est non-commutatif ;*
(*iii*) *la multiplication par p est un endomorphisme purement inséparable.*

2.2 Courbes algébriques

Dans la suite de cette étude nous nous restreignons à un cas particulier de variétés projectives : les courbes algébriques.

Définition 2.32 (Courbe algébrique)**.** Une courbe algébrique est une variété projective de dimension 1.

Cette restriction est importante et permet, tout du moins de façon plus simple, d'étudier des **comportements locaux** ; en effet, l'idée essentielle est qu'un seul paramètre permet de modéliser le comportement de la courbe autour d'un point : ceci n'est plus vrai par exemple pour les surfaces et complexifie notoirement les études locales. Nous commençons par définir proprement la notion de **multiplicité** d'un zéro ou d'un pôle d'une fonction rationnelle, ce qui nous permet par la suite de développer la théorie des diviseurs qui sont les éléments que nous manipulerons pour le calcul de couplages.

2.2.1 Valuations et uniformisantes

Afin de définir la valuation d'une fonction en un point, nous définissons M_P l'ensemble des fonctions rationnelles qui s'annulent en un point P de la courbe $\mathcal{C}$. La valuation d'une fonction $f \in K(\mathcal{C})$ définie en P est nulle si et seulement si f ne s'annule pas en ce point. Dans le cas contraire, il s'agit de compter la multiplicité de ce zéro.

Définition 2.33 (Valuation)**.** Soit $P \in \mathcal{C}$ un point non singulier tel que $f \in K(\mathcal{C})$ est définie en P ; la valuation de f en P est alors

$$\mathrm{ord}_P(f) = \sup\left\{ d \in \mathbb{N} \;\middle|\; f \in M_P^d \right\}.$$

Nous construisons alors les uniformisantes afin d'étendre la notion de valuation aux pôles de f.

Définition 2.34 (Uniformisante)**.** Une uniformisante pour une courbe algébrique $\mathcal{C}$ en un point lisse $P \in \mathcal{C}$ est une fonction $t_P \in \overline{K}(\mathcal{C})$ telle que

$$\mathrm{ord}_P(t_P) = 1.$$

Il est possible de montrer qu'il existe au moins une uniformisante pour chaque point de la courbe. Nous en exhiberons dans le cas hyperelliptique.

Si P est un pôle de f, sa valuation en P sera

$$\mathrm{ord}_P(f) = -\min\left\{d \in \mathbb{N} \;\middle|\; ft_P^d \text{ est définie en } P\right\},$$

où t_P est une uniformisante en P.

Remarquons que la valuation forme un morphisme de groupes :

$$\mathrm{ord}_P(f \cdot g) = \mathrm{ord}_P(f) + \mathrm{ord}_P(g).$$

2.2.2 Morphismes de courbes

L'existence d'une valuation et d'uniformisantes permet d'obtenir des propriétés plus puissantes pour les morphismes de courbes que de variétés.

Proposition 2.35. *Soit une fonction rationnelle $\phi : \mathcal{C} \to \mathcal{V}$ où $\mathcal{C}$ est une courbe algébrique et $\mathcal{V}$ une variété projective. Soit $P \in \mathcal{C}$, si P est un point lisse, alors ϕ est régulière en P.*

Si $\mathcal{C}$ est une courbe non singulière, alors ϕ est un morphisme.

Proposition 2.36. *Tout morphisme $\phi : \mathcal{C}_1 \to \mathcal{C}_2$ est soit constant, soit surjectif.*

Exemple 2.37 (Fonction rationnelle). Soit $\mathcal{C}$ une courbe non singulière et $F \in K(\mathcal{C})$. Il est possible d'associer à F le morphisme de courbes suivant

$$\begin{array}{ccc} \mathcal{C} & \longrightarrow & \mathbb{P}^1 \\ P & \longmapsto & (F(P) : 1). \end{array}$$

Ce morphisme de courbes aurait également pu être défini en donnant le numérateur de F pour la première coordonnée et le dénominateur pour la deuxième. Les pôles de F correspondent alors aux points auxquels est associé $(1 : 0)$. Comme c'est un morphisme, si F n'est pas constante, nous pouvons en déduire qu'elle a au moins un pôle.

Dorénavant, toutes les courbes considérées seront **lisses**, de sorte que nous pourrons définir la valuation d'une fonction en tout point de la courbe.

2.2.3 Diviseurs

Le groupe des diviseurs de $\mathcal{C}$, noté additivement, est le groupe abélien libre généré par les points de la courbe $\mathcal{C}$ sur une clôture algébrique :

$$\mathrm{Div}(\mathcal{C}) = \left\{ \sum_{P \in \mathcal{C}} n_P(P) \;\middle|\; n_P \in \mathbb{Z} \text{ et seulement un nombre fini de } n_p \text{ sont non nuls} \right\}.$$

Soit $D = \sum_{P \in \mathcal{C}} n_P(P)$. Nous appelons support de D l'ensemble des points P pour lesquels n_P est non nul :

$$\operatorname{supp} D = \{P \in \mathcal{C} \mid n_P \neq 0\}.$$

Son degré est

$$\deg D = \sum_{P \in \mathcal{C}} n_P.$$

Les diviseurs de degré nul forment un sous-groupe de $\operatorname{Div}(\mathcal{C})$ que nous noterons $\operatorname{Div}^0(\mathcal{C})$.

Nous cherchons maintenant à définir la partie K-rationnelle des diviseurs de $\mathcal{C}/K$. Il ne s'agit pas seulement de construire le groupe libre sur $\mathcal{C}(K)$ mais de faire agir le groupe de Galois $\mathbb{G}_{\overline{K}/K}$ sur $\operatorname{Div}(\mathcal{C})$:

$$\forall \sigma \in \mathbb{G}_{\overline{K}/K}, D^\sigma = \sum_{P \in \mathcal{C}} n_P(P^\sigma).$$

$\operatorname{Div}_K(\mathcal{C})$ est alors la partie de $\operatorname{Div}(\mathcal{C})$ laissée invariante par l'action de $\mathbb{G}_{\overline{K}/K}$. Remarquons que le support de $D \in \operatorname{Div}_K(\mathcal{C})$ peut contenir des points dont les coordonnées ne sont pas dans K : soient $\sigma \in \mathbb{G}_{\overline{K}/K}$ et $P \in \mathcal{C}$, l'ensemble $P, P^\sigma, \ldots, P^{\sigma^k} = P$ des conjugués de P par σ peuvent être dans le support de D à condition qu'ils apparaissent tous avec le même coefficient. Nous définissons de même $\operatorname{Div}_K^0(\mathcal{C})$ le sous-groupe des diviseurs de degré nul de $\operatorname{Div}_K(\mathcal{C})$.

Nous munissons également les diviseurs d'une relation d'ordre partiel :

$$D_1 \geq D_2 \Leftrightarrow \text{tous les coefficients de } D_1 - D_2 \text{ sont positifs.}$$

Si $D \geq 0$, D est dit **effectif**.

Soit $D = \sum_{P \in \mathcal{C}} n_P(P)$, il peut être utile de considérer la partie effective de D :

$$\epsilon(D) = \sum_{\substack{P \in \mathcal{C} \\ n_P > 0}} n_P(P).$$

2.2.4 Diviseurs principaux

À chaque fonction rationnelle non nulle, nous associons un diviseur par le morphisme de groupe suivant :

$$\begin{array}{rccc} \operatorname{div} : & \overline{K}(\mathcal{C})^* & \longrightarrow & \operatorname{Div}(\mathcal{C}) \\ & f & \longmapsto & \sum_{P \in \mathcal{C}} \operatorname{ord}_P(f)(P). \end{array}$$

Le diviseur $\operatorname{div}(f)$ est une structure qui code les zéros et pôles de f en tenant compte de leur multiplicité. Notons que ce morphisme est correctement défini grâce au fait qu'une fonction rationnelle non nulle ne contient qu'un nombre fini de zéros et de pôles.

Remarquons que ce morphisme est compatible avec l'action du groupe de Galois $\mathbb{G}_{\overline{K}/K}$:

$$\forall \sigma \in \mathbb{G}_{\overline{K}/K}, \operatorname{div}(f^\sigma) = (\operatorname{div} f)^\sigma.$$

Ainsi, le diviseur d'une fonction K-rationnelle sera également K-rationnel.

Proposition 2.38. *Soit $f \in \overline{K}(\mathcal{C})^*$. Nous avons*

(i) $\operatorname{div}(f) = 0 \Leftrightarrow f \in \overline{K}^*$,

(ii) $\deg \operatorname{div}(f) = 0$.

Définition 2.39 (Diviseur principal)**.** Un diviseur est principal s'il est le diviseur d'une fonction rationnelle. Nous notons $\mathrm{Princ}(\mathcal{C})$ l'ensemble des diviseurs principaux ; il forme un sous-groupe de $\mathrm{Div}^0(\mathcal{C})$.

Un diviseur principal définit une fonction rationnelle à une constante multiplicative près.

Remarque 2.40. Un argument de cohomologie galoisienne permet de vérifier que cela reste valable sur K : à tout diviseur K-rationnel $D \in \mathrm{Princ}(\mathcal{C})$ correspond une fonction $f \in K(\mathcal{C})^*$ telle que $\mathrm{div}\, f = D$ [Gal12, Théorème 7.8.3] [Was08, Lemme 11.10]. En d'autres termes, nous avons

$$\mathrm{Princ}_K(\mathcal{C}) = \{\mathrm{div}\, f \mid f \in K(\mathcal{C})^*\} = \mathrm{Princ}(\mathcal{C})^{\mathbb{G}_{\overline{K}/K}}.$$

2.2.5 Groupe de Picard et jacobienne

Comme nous avons vu que $\mathrm{Princ}(\mathcal{C})$ est un sous-groupe de $\mathrm{Div}^0(\mathcal{C})$, il est naturel de considérer le groupe quotient.

Définition 2.41 (Groupe de Picard)**.** Nous définissons le **groupe de Picard**, aussi appelé le **groupe des classes de diviseurs**, comme $\mathrm{Pic}(\mathcal{C}) = \mathrm{Div}(\mathcal{C})/\mathrm{Princ}(\mathcal{C})$. Deux diviseurs D_1 et D_2 sont dits linéairement équivalents, ce que nous notons $D_1 \sim D_2$, si $D_1 - D_2$ est principal. Pour tout diviseur $D \in \mathrm{Div}(\mathcal{C})$, nous utiliserons la notation $\overline{D}$ pour désigner sa classe d'équivalence dans $\mathrm{Pic}(\mathcal{C})$.

Nous pouvons aussi définir le groupe de classe des diviseurs de degré 0, noté $\mathrm{Pic}^0(\mathcal{C})$, de la même manière : $\mathrm{Pic}^0(\mathcal{C}) = \mathrm{Div}^0(\mathcal{C})/\mathrm{Princ}(\mathcal{C})$.

Les constructions précédentes peuvent être résumées dans la suite exacte suivante :

$$1 \longrightarrow \overline{K}^* \longrightarrow \overline{K}(\mathcal{C})^* \longrightarrow \mathrm{Div}^0(\mathcal{C}) \longrightarrow \mathrm{Pic}^0(\mathcal{C}) \longrightarrow 0.$$

Théorème 2.42 (Variété jacobienne)**.** *Étant donnée une courbe algébrique $\mathcal{C}$ sur K, s'il existe un point K-rationnel $P_0 \in \mathcal{C}(K)$, alors le groupe $\mathrm{Pic}^0(\mathcal{C})$ peut-être muni d'une structure de variété abélienne, appelée la* ***variété jacobienne*** *de la courbe $\mathcal{C}$ et notée $\mathrm{Jac}_{\mathcal{C}}$ [Lor96, Section XI.3].*

Définition 2.43 (Genre)**.** La dimension de la variété jacobienne associée à une courbe $\mathcal{C}$ est appelée le **genre** de $\mathcal{C}$.

Notons que l'hypothèse de l'existence d'un point P_0 K-rationnel sur $\mathcal{C}$, nécessaire pour identifier $\mathrm{Jac}_{\mathcal{C}}$ à $\mathrm{Pic}^0(\mathcal{C})$, est vérifiée dans la plupart des cas. Ainsi, nous parlerons dans la suite de jacobienne afin de respecter les habitudes de la littérature, mais nous utiliserons la construction du groupe de Picard pour travailler explicitement dans cette structure.

Encore une fois, nous nous intéressons plus particulièrement à la partie K-rationnelle de la jacobienne, notée $\mathrm{Jac}_{\mathcal{C}}(K)$. Il s'agit des éléments fixés par l'action du groupe de Galois de $\overline{K}$ sur K : $\mathrm{Jac}_{\mathcal{C}}(K) = \mathrm{Jac}_{\mathcal{C}}^{\mathbb{G}_{\overline{K}/K}} = \mathrm{Pic}^0(\mathcal{C})^{\mathbb{G}_{\overline{K}/K}}$.

Il est a noter que, si K est un corps fini, nous avons aussi

$$\mathrm{Pic}^0(\mathcal{C})^{\mathbb{G}_{\overline{K}/K}} = \mathrm{Div}^0(\mathcal{C})^{\mathbb{G}_{\overline{K}/K}}/\mathrm{Princ}(\mathcal{C})^{\mathbb{G}_{\overline{K}/K}}$$

[Lor96, Remarque XI.5.5]. Nous pouvons alors identifier $\mathrm{Jac}_{\mathcal{C}}(K)$ au groupe des classes de diviseurs K-rationnels de degré 0, défini comme $\mathrm{Pic}^0_K(\mathcal{C}) = \mathrm{Div}^0_K(\mathcal{C})/\mathrm{Princ}_K(\mathcal{C})$.

Afin de travailler avec les éléments du groupe de Picard, nous avons besoin de choisir un représentant pour chaque classe de diviseurs. Le théorème de Riemann-Roch permet de déduire une forme de diviseur réduit. Soit $P_0 \in \mathcal{C}$, tout diviseur $D \in \mathrm{Div}(\mathcal{C})$ peut s'écrire sous la forme réduite équivalente suivante :

$$\sum_{i=0}^{m}(P_i) - m(P_0), \text{ avec } m \leq g.$$

2.2.6 Propriétés des fonctions et des diviseurs

Nous étendons ici l'évaluation de fonctions rationnelles en un point à l'évaluation en un diviseur.

Définition 2.44. Soit $f \in \overline{K}(\mathcal{C})$. Pour tout diviseur $D = \sum_{P\in\mathcal{C}} n_P(P)$ de $\mathcal{C}$ de support disjoint de celui de $\mathrm{div}\, f$, nous définissons l'évaluation de f en D par le produit suivant

$$f(D) = \prod_{P\in\mathcal{C}} f(P)^{n_P}.$$

Nous avons alors un théorème qui sera essentiel pour le calcul de couplages car il permet notamment de vérifier le bien-fondé des constructions mathématiques de couplages que nous exposons dans le chapitre suivant.

Théorème 2.45 (Réciprocité de Weil)**.** *Soient $\mathcal{C}$ une courbe lisse et f, g deux fonctions rationnelles non nulles telles que*

$$\mathrm{supp}\,\mathrm{div}(f) \cap \mathrm{supp}\,\mathrm{div}(g) = \emptyset.$$

Nous avons alors l'égalité suivante :

$$f(\mathrm{div}\, g) = g(\mathrm{div}\, f).$$

Dans la suite de cette section, nous considérons deux courbes $\mathcal{C}_1$, $\mathcal{C}_2$ et un morphisme $\phi : \mathcal{C}_1 \to \mathcal{C}_2$ entre ces deux courbes. Il s'agit désormais d'étudier les interactions entre fonctions et diviseurs dans cette configuration. Nous commençons par définir un tiré-en-arrière et un poussé-en-avant permettant de transporter les diviseurs d'une courbe sur l'autre en utilisant le morphisme ϕ.

Définition 2.46. Soit $\phi : \mathcal{C}_1 \to \mathcal{C}_2$ un morphisme de courbes.

- Soit D un diviseur de $\mathcal{C}_2$, le **tiré-en-arrière** (*pull-back*) $\phi^*(D)$ de D est le diviseur obtenu en sommant les antécédents par ϕ de chaque point de D, comptés avec leur multiplicité.
- Soit D un diviseur de $\mathcal{C}_1$, le **poussé-en-avant** (*push-forward*) $\phi_*(D)$ de D est le diviseur obtenu en sommant les images par ϕ de chaque point de D.

Nous omettons volontairement ici d'expliciter le tiré-en-arrière par une formule ; le lecteur pourra compléter cette présentation par [Sil86, Sections II.2 et II.3] pour se familiariser avec la notion de ramification permettant de formaliser la notion de multiplicité dans ce cas. Nous donnons cependant une formule pour le poussé-en-avant, celle-ci s'exprimant de façon plus élémentaire :

$$\begin{array}{rccc} \phi_* : & \mathrm{Div}\,(\mathcal{C}_1) & \longrightarrow & \mathrm{Div}\,(\mathcal{C}_2) \\ & \sum_i (P_i) & \longmapsto & \sum_i (\phi(P_i)). \end{array}$$

Nous continuons cette présentation par quelques propriétés qui nous seront utiles lorsque nous chercherons à construire des couplages calculables efficacement dans le Chapitre 6.

Proposition 2.47. *Soit $\phi : \mathcal{C}_1 \to \mathcal{C}_2$ un morphisme de courbes.*
a. $\operatorname{div} \phi^*(f) = \phi^*(\operatorname{div} f)$.
b. Si D est principal, alors $\phi_*(D)$ est principal.
c. $\phi^* \circ \phi_*$ est la multiplication par $\deg \phi$ dans $\operatorname{Div}(\mathcal{C}_1)$.

2.2.7 Bornes de Hasse-Weil

Soit $\mathcal{C}$ une courbe de genre g définie sur un corps fini $\mathbb{F}_q$. L'action du Frobenius qui élève les coordonnées des points de $\mathcal{C}$ à la puissance q s'étend en un endomorphisme de sa jacobienne ; cela donne exactement l'endomorphisme de Frobenius évoqué en Section 2.1.7. Au sein de l'anneau $\operatorname{End}(\operatorname{Jac}_\mathcal{C})$, l'endomorphisme de Frobenius annule un polynôme de degré $2g$, appelé **polynôme caractéristique**, et noté $\chi_\mathcal{C}(T)$. L'existence même d'un tel polynôme, ainsi que ses propriétés, désormais prouvées, ont été conjecturées par Weil.

Théorème 2.48 (Conjectures de Weil)**.** *Soit $\mathcal{C}$ une courbe de genre g définie sur $\mathbb{F}_q$. L'endomorphisme de Frobenius associé à $\mathcal{C}$ admet un polynôme annulateur $\chi_\mathcal{C}(T)$ appelé polynôme caractéristique qui vérifie :*
— *$\chi_\mathcal{C}(T)$ est de degré $2g$ et est à coefficients entiers ;*
— *$\frac{T^{2g}}{q^g}\chi_\mathcal{C}(q/T) = \chi_\mathcal{C}(T)$;*
— *toutes les racines (complexes) de $\chi_\mathcal{C}(T)$ sont de module $\sqrt{q}$.*

Il y a un lien entre l'ensemble des points de $\operatorname{Jac}_\mathcal{C}$ définis sur $\mathbb{F}_q$ et l'endomorphisme de Frobenius, puisque celui-ci fixe les points rationnels. Cela se traduit *in fine* par $\#\operatorname{Jac}_\mathcal{C}(\mathbb{F}_q) = \chi_\mathcal{C}(1)$. Plus précisément, nous avons le théorème suivant.

Théorème 2.49 (Bornes de Hasse-Weil)**.** *Soit $\mathcal{C}$ une courbe de genre g définie sur $\mathbb{F}_q$. Le nombre de points $\mathbb{F}_q$-rationnel de la courbe est borné par*

$$|\#\mathcal{C}(\mathbb{F}_q) - q - 1| \leq 2g\sqrt{q}.$$

De même, le nombre d'éléments de sa jacobienne est borné par

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

Corollaire 2.50. *On a*

$$|\#\operatorname{Jac}_\mathcal{C}(\mathbb{F}_q) - q^g| = O(q^{g-1/2}).$$

Les propriétés du polynôme caractéristique du Frobenius fournissent un autre type d'information. Soit ℓ un entier premier différent de la caractéristique du corps. L'endomorphisme de Frobenius envoie un point de ℓ-torsion sur un point de ℓ-torsion, il agit donc sur $\operatorname{Jac}_\mathcal{C}[\ell]$. Or, nous avons vu que $\operatorname{Jac}_\mathcal{C}[\ell]$ est isomorphe à $(\mathbb{Z}/\ell\mathbb{Z})^{2g}$; c'est donc un $\mathbb{Z}/\ell\mathbb{Z}$-espace vectoriel. Comme l'endomorphisme de Frobenius commute avec la loi de groupe, son action sur $\operatorname{Jac}_\mathcal{C}[\ell]$ est donc une action d'endomorphisme d'espace vectoriel sur $\mathbb{Z}/\ell\mathbb{Z}$. Le polynôme caractéristique de cette action est alors exactement $\chi_\mathcal{C}(T)$ modulo ℓ.

Ainsi, une racine de $\chi_\mathcal{C}(T)$ modulo ℓ peut être interprétée comme une valeur propre de l'action du Frobenius sur la ℓ-torsion. Et le deuxième point du Théorème 2.48 implique que si v est une telle valeur propre modulo ℓ, alors $q/v \mod \ell$ est aussi une valeur propre du Frobenius. Les espaces propres associés à ces valeurs propres joueront un rôle important pour définir des couplages efficaces.

2.3 Courbes (hyper)elliptiques et leurs jacobiennes

Nous avons maintenant à notre disposition tous les outils théoriques qui nous permettront de définir les couplages et leurs algorithmes de calcul. Cependant, nous avons besoin d'un **modèle de courbe** pour rendre ces calculs effectifs. En effet, il ne nous reste plus qu'à poser un cadre dans lequel nous pourrons manipuler efficacement les diviseurs qui interviendront dans le calcul de couplages.

Nous axons donc cette section sur la description d'une famille de courbes qui fournit une arithmétique efficace pour la courbe elle-même et sa jacobienne : les courbes hyperelliptiques.

2.3.1 Définition et propriétés des courbes

En toute généralité, une courbe hyperelliptique est une courbe algébrique qui constitue un recouvrement de degré 2 de la droite projective : à chaque point de la droite projective sont associés au plus deux points de la courbe hyperelliptique. Cependant, nous cherchons une description plus simple, notamment d'un point de vue calculatoire. Nous donnons donc une définition volontairement plus limitée.

Définition 2.51 (Courbe hyperelliptique)**.** Une courbe hyperelliptique de genre g sous forme imaginaire est une courbe qui admet une équation affine de la forme

$$\mathcal{H}/K : y^2 + h(x)y = f(x),$$

avec $f, g \in K[X]$, $\deg f = 2g + 1$, $\deg h \leq g$, contenant au moins un point K-rationnel et n'acceptant pas de point singulier dans sa partie affine.

Il est possible de vérifier que cette définition est compatible avec la notion générale de genre que nous avons donnée à la Définition 2.43.

Notre définition de courbe hyperelliptique n'inclut pas toutes les courbes classiquement considérées comme hyperelliptiques (celles ne possédant pas de point K-rationnel) mais cela nous permet de considérer les **courbes elliptiques** comme une sous-famille des courbes hyperelliptiques : celles de **genre** 1.

Toutefois, le cas des courbes elliptiques est plus simple et nous devons garder à l'esprit que, pour ces courbes, la jacobienne est isomorphe à la courbe elle-même. D'ailleurs, inversement, toute variété abélienne de dimension 1 est une courbe elliptique. Tout ce qui suit, et notamment la loi de groupe, se simplifie donc grandement dans le cas elliptique : nous retombons sur la règle classique « corde et tangente ».

§ 1. Non singularité. Une courbe hyperelliptique est par définition une **courbe lisse** sur sa partie affine ; nous donnons ici les conditions que cela impose sur le choix des polynômes f et h. Un point affine singulier est une solution du système suivant :

$$\begin{cases} F & = y^2 + h(x)y - f(x) & = 0, \\ \partial F/\partial x & = h'(x)y - f'(x) & = 0, \\ \partial F/\partial y & = 2y + h(x) & = 0. \end{cases}$$

Il est possible de montrer que cela impose que $f(x) + h(x)^2/4$ n'ait pas de racine double en caractéristique différente de 2, ou que h soit un polynôme non nul premier avec $f'(x)^2 - f(x)h(x)$.

§ 2. Point à l'infini. Bien que nous décrivions une courbe hyperelliptique par son équation affine dans un souci de concision, sa nature est de fait projective. La clôture projective d'une courbe imaginaire possède en effet un **unique point à l'infini** $P_\infty = (0 : 1 : 0)$. Pour le genre supérieur ($g > 1$), ce point est singulier ; cependant, il est possible de construire un modèle désingularisé de la courbe. Nous pourrons donc utiliser les outils associés aux morphismes de courbes. Nous n'exhibons pas cette construction mais invitons le lecteur à consulter [Gal12, Section 10.1] pour en trouver une preuve. Désormais nous n'utiliserons plus que l'écriture affine et posons

$$\mathcal{H} = \left\{(x, y) \in \overline{K}^2 \;\middle|\; y^2 + h(x)y = f(x)\right\} \cup \{P_\infty\}.$$

§ 3. Involution hyperelliptique. Les courbes hyperelliptiques possèdent un automorphisme particulier : l'**involution hyperelliptique**. L'existence de celle-ci est la conséquence directe du fait que la courbe constitue un recouvrement de degré 2. Elle est également à l'origine d'un certain nombre des propriétés qui expliquent l'efficacité arithmétique de ces courbes.

Définition 2.52 (Involution hyperelliptique)**.** Une courbe hyperelliptique $\mathcal{H}$ possède l'automorphisme suivant, involution hyperelliptique :

$$\begin{array}{rccc} \iota : & \mathcal{H} & \longrightarrow & \mathcal{H} \\ & (x, y) & \longmapsto & (x, -y - h(x)). \end{array}$$

Il est facile de vérifier que cette définition donne bien un automorphisme et une involution.

§ 4. Uniformisante. Afin de pouvoir efficacement évaluer des couplages, nous aurons besoin de définir une normalisation des fonctions rationnelles ; celle-ci s'appuie sur les uniformisantes (Section 5.3.1).

Proposition 2.53 (Uniformisante d'un point d'une courbe hyperelliptique)**.** *Nous donnons ici un choix d'uniformisante au point P (Figure 2.1) :*

- *si $P = P_\infty$, $t_{P_\infty} = x^g/y$,*
- *si $P = \iota(P)$, $t_P = y - y_P$,*
- *si $P \neq \iota(P)$, $t_P = x - x_P$.*

2.3.2 Représentation de la jacobienne

Nous connaissons déjà une représentation d'un élément de la jacobienne d'une courbe (Section 2.2.5) grâce à un diviseur de petite taille : sa partie effective contient au plus g points. Le cas hyperelliptique nous offre cependant une représentation plus précise.

Théorème 2.54. *Tout élément $\overline{D}$ de la jacobienne d'une courbe hyperelliptique admet un représentant **unique** de la forme*

$$\sum_{i=1}^{m} (P_i) - m(P_\infty),$$

avec les conditions suivantes

(i) $m \leq g$,

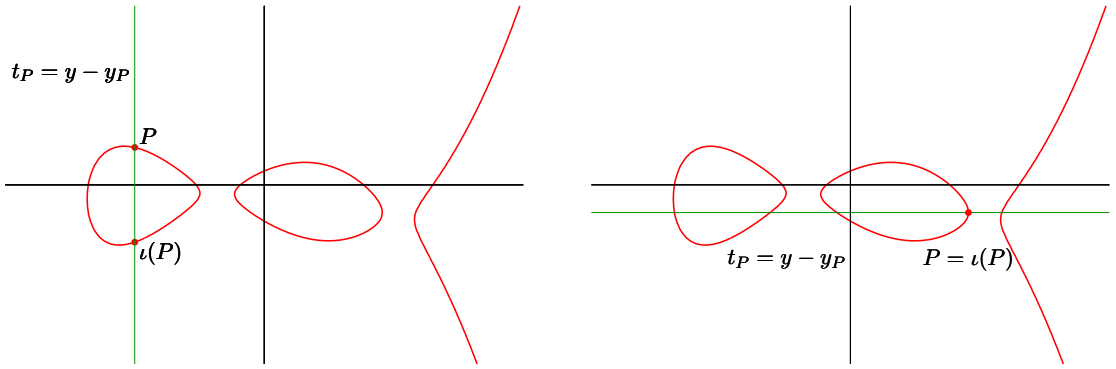


FIGURE 2.1 – Uniformisante en P pour une courbe hyperelliptique.

(*ii*) *les P_i sont des points affines, et*
(*iii*) *$\iota(P_i) \neq P_j$ pour tout i, j tels que $i \neq j$.*

L'unicité de cette représentation est un atout car elle permet de définir un morphisme de réduction. Nous parlerons ainsi du **diviseur réduit** $\rho(\overline{D})$ associé à la classe d'équivalence $\overline{D}$:

$$\rho(\overline{D}) = \sum_{i=1}^{m}(P_i) - m(P_\infty) \in \overline{D}.$$

Toutefois, il nous reste à régler les problèmes de K-rationalité. En effet, un diviseur K-rationnel peut contenir des points vivant dans une extension de K. La **représentation de Mumford** permet de s'affranchir de ce problème.

Théorème 2.55 (Représentation de Mumford)**.** *Les éléments de la jacobienne de $\mathcal{H}$ définie sur K sont en bijection avec les paires de polynômes $[u(x), v(x)]$ à coefficients dans K tels que*
(*i*) *u est unitaire,*
(*ii*) $\deg u \leq g$,
(*iii*) $\deg v < \deg u$,
(*iv*) $u \mid v^2 + hv - f$.

De plus, de par la remarque suivante, la correspondance entre la représentation de Mumford et le diviseur réduit est explicite.

Remarque 2.56. Soit $D = [u, v]$ un diviseur réduit donné par sa représentation de Mumford. Comme u est unitaire, il se factorise sous la forme suivante sur $\overline{K}[x]$: $u(x) = \prod_{i=0}^{m}(x - x_i)$. Le diviseur D s'écrit alors

$$D = \sum_{i=0}^{m}(P_i) - m(P_\infty) \text{ où } P_i = (x_i, y_i) \text{ et } y_i = v(x_i).$$

Remarquons que, lors de la factorisation de u, des racines non K-rationnelles peuvent apparaître.

Cette correspondance permet de mieux comprendre la représentation de Mumford. La première coordonnée u encode les abscisses (avec multiplicité) des points du support effectif et la deuxième v donne leurs ordonnées[1]. Enfin, il peut y avoir plusieurs fois le même point dans un diviseur réduit ; grâce à la condition $u \mid v^2 + hv - f$, le polynôme v porte aussi l'information de la multiplicité de ces points.

Quant au passage du diviseur réduit à sa représentation de Mumford, il suffit de prendre u comme le produit précédent et d'effectuer une interpolation de Lagrange pour les coordonnées v.

Comme nous calculerons avec les représentations de Mumford, il est naturel de considérer la taille d'une telle représentation. Dans le cas général, u est unitaire de degré g et v est un polynôme de degré $g-1$: **$2g$ coefficients dans K** suffisent donc à représenter $\mathrm{Jac}_{\mathcal{H}}(K)$.

Le propre d'une bonne représentation d'un objet mathématique est non seulement d'être compacte, mais aussi de fournir un moyen de calcul efficace. Dans ce cas, nous souhaitons calculer la loi de groupe de la jacobienne directement sur la représentation de Mumford. C'est le rôle de l'**algorithme de Cantor** que nous décrirons plus tard à la Section 5.2. Nous en extrairons alors des informations utiles aux calculs de couplage.

1. La condition $\iota(P_i) \neq P_j$ pour $i \neq j$ assure qu'il n'y ait qu'un seul point du support d'abscisse x_i.

Chapitre 3

Couplages de Weil et Tate

Nous cherchons dans ce chapitre à définir les couplages dont les groupes sont construits à partir de courbes hyperelliptiques dans le but de pouvoir les utiliser comme des couplages cryptographiques (voir Définition 1.3). Pour ce faire, nous commençons par présenter deux couplages classiques : le couplage de Weil [Wei40] et celui de Tate [Tat66]. Nous commençons par introduire le couplage de Weil afin de donner les premières problématiques de la définition d'une telle fonction. Cependant, nous ne donnerons de preuves que pour le couplage de Tate car c'est celui sur lequel nous nous sommes concentrés lors de cette thèse. En effet, les techniques utilisées dans ces preuves (calculs de diviseurs de fonctions, réciprocité de Weil) sont à l'origine des théorèmes qui permettent des constructions avancées de couplages efficaces pour le calcul (Chapitre 6).

3.1 Couplage de Weil

Avant de pouvoir définir le couplage de Weil, nous devons nous fixer un groupe cyclique à partir duquel nous construirons le couplage. Soit $\mathcal{H}$ une courbe hyperelliptique définie sur un corps K de caractéristique non-nulle ; nous travaillerons dans sa jacobienne $\mathrm{Jac}_{\mathcal{H}}$ et plus particulièrement dans sa ℓ-torsion où ℓ est un entier premier avec la caractéristique p du corps K. Bien que la définition et les propriétés du couplage de Weil énoncées ci-après ne nécessitent pas cette hypothèse, nous nous restreindrons dans le cadre de cette thèse au cas où ℓ est premier.

Le groupe d'arrivée du couplage de Weil sera un sous-groupe multiplicatif de $\overline{K}$, le groupe des racines ℓ-ièmes de l'unité :

$$\mu_\ell = \left\{ x \in \overline{K} \;\middle|\; x^\ell = 1 \right\}.$$

Afin de pouvoir expliciter le couplage de Weil, nous introduisons ici les fonctions de Miller [Mil86a ; Mil04]. Cette famille de fonctions rationnelles est paramétrée par un entier et un diviseur.

Définition 3.1 (Fonction de Miller)**.** Soient $n \in \mathbb{Z}$ et $D \in \mathrm{Div}^0(\mathcal{H})$. La fonction de Miller correspondante, notée $f_{n,D}$, est donnée par son diviseur :

$$\mathrm{div}\, f_{n,D} = nD - \rho(nD).$$

Remarquons que ces fonctions ne sont définies qu'à une constante multiplicative près.

Nous donnons maintenant la construction du couplage de Weil [Wei40].

Définition 3.2 (Couplage de Weil)**.** Étant donnée une courbe hyperelliptique $\mathcal{H}$ définie sur K et un premier ℓ distinct de la caractéristique de K, le couplage de Weil, noté w_ℓ, est défini par

$$\begin{array}{rccccc} w_\ell : & \mathrm{Jac}_{\mathcal{H}}[\ell] & \times & \mathrm{Jac}_{\mathcal{H}}[\ell] & \longrightarrow & \mu_\ell \\ & (\overline{D_1} & , & \overline{D_2}) & \longmapsto & \frac{f_{\ell,D_1}(D_2)}{f_{\ell,D_2}(D_1)}, \end{array}$$

où $D_1 \in \overline{D_1}$ et $D_2 \in \overline{D_2}$ sont tels que $\operatorname{supp} D_1 \cap \operatorname{supp} D_2 = \emptyset$.

Comme $\overline{D_1}$ et $\overline{D_2}$ sont des éléments de ℓ-torsion, $\ell D_1 \sim \ell D_2 \sim 0$. Les diviseurs des fonctions de Miller associées sont ainsi plus simples :

$$\operatorname{div} f_{\ell,D_1} = \ell D_1 \text{ et } \operatorname{div} f_{\ell,D_2} = \ell D_2.$$

Par conséquent les supports de $\operatorname{div} f_{\ell,D_1}$ et D_2 sont disjoints et il est possible d'évaluer f_{ℓ,D_1} en D_2 (Définition 2.44). Il en va de même pour l'évaluation de f_{ℓ,D_2} en D_1.

Il reste alors à vérifier que le couplage de Weil est bien défini, c'est-à-dire que la valeur $w_\ell(\overline{D_1}, \overline{D_2})$ ne dépend pas du choix des diviseurs D_1 et D_2 comme représentants des classes d'équivalence $\overline{D_1}$ et $\overline{D_2}$ de la jacobienne. Nous invitons le lecteur à consulter [Sil86, Section III.8] pour en obtenir une preuve ainsi que celles des propriétés élémentaires que nous donnons dans le théorème suivant.

Théorème 3.3. *Le couplage de Weil vérifie les propriétés suivantes.*

(*i*) ***Bilinéarité*** *: pour tous* $\overline{D_1}, \overline{D_1'}, \overline{D_2}, \overline{D_2'} \in \mathrm{Jac}_{\mathcal{H}}[\ell]$,

$$\begin{cases} w_\ell(\overline{D_1} + \overline{D_1'}, \overline{D_2}) = w_\ell(\overline{D_1}, \overline{D_2}) \cdot w_\ell(\overline{D_1'}, \overline{D_2}), \\ w_\ell(\overline{D_1}, \overline{D_2} + \overline{D_2'}) = w_\ell(\overline{D_1}, \overline{D_2}) \cdot w_\ell(\overline{D_1}, \overline{D_2'}). \end{cases}$$

(*ii*) ***Non-dégénérescence*** *: si* $w_\ell(\overline{D_1}, \overline{D_2}) = 1$ *pour tout* $\overline{D_1} \in \mathrm{Jac}_{\mathcal{H}}[\ell]$ *alors* $\overline{D_2}$ *est trivial* ($\overline{D_2} = \overline{0}$)*.*

(*iii*) ***Alternance*** *: pour tout* $\overline{D} \in \mathrm{Jac}_{\mathcal{H}}[\ell], w_\ell(\overline{D}, \overline{D}) = 1$.

(*iv*) ***Invariance galoisienne*** *: pour tout automorphisme* $\sigma \in \mathbb{G}_{\overline{K}/K}$ *et pour tous* $\overline{D_1}, \overline{D_2} \in \mathrm{Jac}_{\mathcal{H}}[\ell]$,

$$w_\ell(\overline{D_1}^\sigma, \overline{D_2}^\sigma) = \sigma(w_\ell(\overline{D_1}, \overline{D_2})).$$

Cette liste de propriétés n'est pas exhaustive mais nous suffira à continuer notre étude.

3.2 Couplage de Tate

Nous définissons ici le couplage de Tate [Tat66] qui est à l'origine de tous les couplages que nous avons étudiés durant cette thèse. Pour ce faire, nous nous restreignons au cas où K est un corps fini $\mathbb{F}_q$ de caractéristique p, et $\mathcal{H}$ une courbe hyperelliptique définie sur $\mathbb{F}_q$. Étant donné un premier ℓ, nous commençons par nous intéresser au corps de définition des racines ℓ-ièmes de l'unité dans $\overline{\mathbb{F}_q}$.

Définition 3.4 (Degré de plongement)**.** Le degré de plongement (*embedding degree* en anglais) des racines ℓ-ièmes de l'unité dans $\mathbb{F}_q$ est le plus petit entier $k > 0$ tel que

$$\mu_\ell = \left\{ x \in \overline{\mathbb{F}_q} \;\middle|\; x^\ell = 1 \right\} \subset \mathbb{F}_{q^k}^*.$$

De manière équivalente, c'est l'ordre de q dans $(\mathbb{Z}/\ell\mathbb{Z})^*$: c'est-à-dire le plus petit entier k tel que

$$\ell \mid q^k - 1.$$

Nous sommes désormais en mesure de définir le couplage de Tate.

Définition 3.5 (Couplage de Tate)**.** Soient $\mathcal{H}$ une courbe hyperelliptique définie sur $\mathbb{F}_q$ et ℓ un premier ne divisant pas q. Nous posons k le degré de plongement des racines ℓ-ièmes de l'unité dans $\mathbb{F}_q$. Le couplage de Tate est alors défini par

$$\begin{array}{rccccc} \langle .,. \rangle_\ell : & \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell] & \times & \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})/\ell\,\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k}) & \longrightarrow & \mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^\ell \\ & (\overline{D_1} & , & \overline{D_2}) & \longmapsto & f_{\ell,D_1}(D_2), \end{array}$$

où $D_1 \in \overline{D_1}$ et $D_2 \in \overline{D_2}$ sont deux diviseurs $\mathbb{F}_{q^k}$-rationnels de supports disjoints et où la fonction de Miller f_{ℓ,D_1} est prise dans $\mathbb{F}_{q^k}(\mathcal{H})^*$.

Remarquons en tout premier lieu que le couplage de Tate ne demande pas de se placer dans la clôture algébrique du corps de définition de la courbe mais permet directement de se placer dans une extension finie de celui-ci : le corps de plongement des racines ℓ-ièmes de l'unité.

À l'instar du couplage de Weil, nous utilisons ici des fonctions de Miller de forme particulière. Ainsi, nous avons

$$\operatorname{div} f_{\ell,D_1} = \ell D_1 - \rho(\ell D_1) = \ell D_1,$$

car $\overline{D_1}$ est de ℓ-torsion. Notons que la Remarque 2.40 nous permet de toujours choisir f_{ℓ,D_1} dans $\mathbb{F}_{q^k}(\mathcal{H})^*$. Une preuve plus constructive sera donnée avec l'algorithme de Miller (Chapitre 5).

Nous commençons par montrer que cette définition est correcte, c'est-à-dire que la valeur du couplage ne dépend pas du choix de D_1 et D_2 dans $\mathrm{Div}^0_{\mathbb{F}_{q^k}}(\mathcal{H})$ comme représentants des éléments $\overline{D_1}$ et $\overline{D_2}$ de la jacobienne, ni du choix de la fonction $\mathbb{F}_{q^k}$-rationnelle f_{ℓ,D_1} de diviseur ℓD_1, ni enfin du choix de $\overline{D_2}$ comme représentant de sa classe d'équivalence dans $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})/\ell\,\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})$.

En effet, l'expression du couplage de Tate de la Définition 3.5 présente ces quatre choix, et nous devons vérifier que son calcul n'est pas influencé par ceux-ci.

Proposition 3.6. *La valeur du couplage ne dépend pas du choix de la fonction f_{ℓ,D_1} ni du représentant choisi pour $\overline{D_1}$.*

Preuve. La fonction $f_{\ell,D_1} \in \mathbb{F}_{q^k}(\mathcal{H})^*$ est donnée par son diviseur, elle est donc définie à une constante multiplicative près. Soient f et f' deux fonctions $\mathbb{F}_{q^k}$-rationnelles de diviseur ℓD_1. Il existe alors $c \in \mathbb{F}_{q^k}^*$ tel que $f' = c \cdot f$. Soit $D_2 \in \overline{D_2}$ respectant les hypothèses de la définition. Il peut alors s'écrire sous la forme $\sum n_R(R)$ et, comme il est de degré nul, nous avons $\sum n_R = \deg D_2 = 0$. L'évaluation du couplage donne alors

$$f'(D_2) = \prod f'(R)^{n_R} = \prod (cf)(R)^{n_R} = c^{\deg D_2} f(D_2) = f(D_2),$$

ce qui montre l'indépendance face au choix de la fonction de Miller.

Soit maintenant $D_1' \in \mathrm{Div}^0_{\mathbb{F}_{q^k}}(\mathcal{H})$ tel que $D_1' \sim D_1$ (c'est-à-dire $D_1' \in \overline{D_1}$). Il existe alors une fonction $\mathbb{F}_{q^k}$-rationnelle h telle que $D_1' = D_1 + \operatorname{div} h$. Montrons que nous obtenons la même fonction de Miller modulo les puissances ℓ-ièmes. Nous avons ainsi

$$\begin{aligned}\operatorname{div} f_{\ell,D_1'} &= \ell D_1' \\ &= \ell D_1 + \ell \operatorname{div} h \\ &= \operatorname{div} f_{\ell,D_1} + \ell \operatorname{div} h,\end{aligned}$$

donc les fonctions $f_{\ell,D_1'}$ et $f_{\ell,D_1} \cdot h^\ell$ sont égales à une constante multiplicative près. Leurs évaluations en D_2 donnent donc la même valeur à une puissance ℓ-ième près. □

Nous laissons pour le moment de côté le fait que le couplage ne dépend pas du choix de $\overline{D_2}$ comme représentant de sa classe d'équivalence dans le groupe quotient $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})/\ell\,\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})$ car c'est une conséquence immédiate des propriétés de bonne définition et de bilinéarité que nous montrons au Théorème 3.8.

Proposition 3.7. *La valeur du couplage ne dépend pas du choix du diviseur D_2 comme représentant de $\overline{D_2}$.*

Preuve. Soient D_2 et D_2' deux représentants $\mathbb{F}_{q^k}$-rationnels de $\overline{D_2}$ et de supports disjoints de celui de D_1. Ces diviseurs sont alors linéairement équivalents, et il existe une fonction $\mathbb{F}_{q^k}$-rationnelle h telle que $D_2' = D_2 + \operatorname{div} h$. La réciprocité de Weil nous donne alors l'égalité suivante :

$$\begin{aligned}f_{\ell,D_1}(D_2') &= f_{\ell,D_1}(D_2) \cdot f_{\ell,D_1}(\operatorname{div} h) \\ &= f_{\ell,D_1}(D_2) \cdot h(\ell D_1) \\ &= f_{\ell,D_1}(D_2) \cdot h(D_1)^\ell \\ &\equiv f_{\ell,D_1}(D_2) \pmod{(\mathbb{F}_{q^k}^*)^\ell}.\end{aligned}$$

□

Théorème 3.8. *Le couplage de Tate vérifie les propriétés suivantes.*

(*i*) ***Bonne définition*** *:*
- *pour tout* $\overline{D_2} \in \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})$, $\langle \overline{0}, \overline{D_2}\rangle_\ell = 1$, *et*
- *pour tous* $\overline{D_1} \in \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell]$ *et* $\overline{D_2} \in \ell\,\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})$, $\langle \overline{D_1}, \overline{D_2}\rangle_\ell \in (\mathbb{F}_{q^k}^*)^\ell$.

(*ii*) ***Non-dégénérescence*** *:*
- *pour tout* $\overline{D_1} \in \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell], \overline{D_1} \neq \overline{0}$, *il existe* $\overline{D_2} \in \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})$ *tel que* $\langle \overline{D_1}, \overline{D_2}\rangle_\ell \notin (\mathbb{F}_{q^k}^*)^\ell$, *et*
- *pour tout* $\overline{D_2} \in \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k}), \overline{D_2} \notin \ell\,\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})$, *il existe* $\overline{D_1} \in \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell]$ *tel que* $\langle \overline{D_1}, \overline{D_2}\rangle_\ell \notin (\mathbb{F}_{q^k}^*)^\ell$.

(*iii*) ***Bilinéarité*** *: pour tous* $\overline{D_1}, \overline{D_1'} \in \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell]$ *et* $\overline{D_2}, \overline{D_2'} \in \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})$, *nous avons*
- $\langle \overline{D_1} + \overline{D_1'}, \overline{D_2}\rangle_\ell \equiv \langle \overline{D_1}, \overline{D_2}\rangle_\ell \cdot \langle \overline{D_1'}, \overline{D_2}\rangle_\ell \pmod{(\mathbb{F}_{q^k}^*)^\ell}$, *et*
- $\langle \overline{D_1}, \overline{D_2} + \overline{D_2'}\rangle_\ell \equiv \langle \overline{D_1}, \overline{D_2}\rangle_\ell \cdot \langle \overline{D_1}, \overline{D_2'}\rangle_\ell \pmod{(\mathbb{F}_{q^k}^*)^\ell}$.

(*iv*) ***Invariance galoisienne*** *: pour tout automorphisme* $\sigma \in \mathbb{G}_{\overline{\mathbb{F}_q}/\mathbb{F}_q}$ *et pour tous* $\overline{D_1} \in \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell]$ *et* $\overline{D_2} \in \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})$,

$$\langle \overline{D_1}^\sigma, \overline{D_2}^\sigma\rangle_\ell \equiv \sigma(\langle \overline{D_1}, \overline{D_2}\rangle_\ell) \pmod{(\mathbb{F}_{q^k}^*)^\ell}.$$

Preuve. **Bonne définition**. La fonction de Miller $f_{\ell,0}$ est une constante car elle est de diviseur nul. Son évaluation en n'importe quel diviseur D_2 donnera donc 1 car nous pouvons choisir $f_{\ell,0} = 1$ grâce à la Proposition 3.6.

Soient maintenant $\overline{D_2} \in \ell \operatorname{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})$, et $\overline{D} \in \operatorname{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})$ tels que $\overline{D_2} = \ell\overline{D}$. Nous choisissons alors les diviseurs $D_2 \in \overline{D_2}$ et $D \in \overline{D}$ tous deux $\mathbb{F}_{q^k}$-rationnels. Nous avons ainsi $D_2 \sim \ell D$. Pour tout élément $\overline{D_1} \in \operatorname{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell]$, représenté par $D_1 \in \operatorname{Div}^0_{\mathbb{F}_{q^k}}(\mathcal{H})$ de support disjoint de ceux de D_2 et D, nous obtenons donc :

$$\begin{aligned}\langle \overline{D_1}, \overline{D_2}\rangle_\ell &= f_{\ell,D_1}(D_2) \\ &\equiv f_{\ell,D_1}(\ell D) \pmod{(\mathbb{F}_{q^k}^*)^\ell} \\ &= f_{\ell,D_1}(D)^\ell \in (\mathbb{F}_{q^k}^*)^\ell.\end{aligned}$$

Non-dégénérescence. La preuve de non-dégénérescence du couplage de Tate utilise des arguments dont les bases théoriques n'ont pas pu être abordées durant cette thèse, aussi nous laissons le lecteur se référer à [FR94, Proposition 2.1] pour une preuve utilisant la cohomologie et à [Hes04a, Théorème 4] pour une preuve plus élémentaire.

Bilinéarité. Nous commençons par montrer la linéarité par rapport au premier argument du couplage. Soient $\overline{D_1}, \overline{D_1'} \in \operatorname{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell]$, représentés par $D_1, D_1' \in \operatorname{Div}^0_{\mathbb{F}_{q^k}}(\mathcal{H})$ et $\overline{D_2} \in \operatorname{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})$. Nous avons l'égalité de diviseurs suivante :

$$\begin{aligned}\operatorname{div} f_{\ell,D_1+D_1'} &= \ell(D_1 + D_1') \\ &= \ell D_1 + \ell D_1' \\ &= \operatorname{div} f_{\ell,D_1} + \operatorname{div} f_{\ell,D_1'},\end{aligned}$$

d'où

$$f_{\ell,D_1+D_1'} = f_{\ell,D_1} \cdot f_{\ell,D_1'} \text{ (à une constante multiplicative près)},$$

et donc

$$\langle \overline{D_1} + \overline{D_1'}, \overline{D_2}\rangle_\ell \equiv \langle \overline{D_1}, \overline{D_2}\rangle_\ell \cdot \langle \overline{D_1'}, \overline{D_2}\rangle_\ell \pmod{(\mathbb{F}_{q^k}^*)^\ell}.$$

La linéarité en la deuxième variable est une simple conséquence de la définition de l'évaluation en un diviseur d'une fonction rationnelle (Définition 2.44).

Invariance galoisienne. Soient D_1, D_2 deux diviseurs respectant les hypothèses de la définition du couplage de Tate et $\sigma \in \mathbb{G}_{\overline{\mathbb{F}_q}/\mathbb{F}_q}$. Nous avons alors

$$\begin{aligned}\sigma(\langle \overline{D_1}, \overline{D_2}\rangle_\ell) &= \sigma(f_{\ell,D_1}(D_2)) \\ &= f^\sigma_{\ell,D_1}(D_2^\sigma),\end{aligned}$$

or

$$\operatorname{div} f^\sigma_{\ell,D_1} = (\ell D_1)^\sigma = \ell(D_1^\sigma) = \operatorname{div} f_{\ell,D_1^\sigma}$$

car D_1^σ est également un diviseur de ℓ-torsion ($\mathcal{H}$ est définie sur $\mathbb{F}_q$).

Nous pouvons donc conclure que

$$\begin{aligned}\sigma(\langle \overline{D_1}, \overline{D_2}) &= f^\sigma_{\ell,D_1}(D_2^\sigma) \\ &\equiv f_{\ell,D_1^\sigma}(D_2^\sigma) \pmod{(\mathbb{F}_{q^k}^*)^\ell} \\ &= \langle \overline{D_1}^\sigma, \overline{D_2}^\sigma\rangle_\ell.\end{aligned}$$

□

La Définition 3.5 du couplage de Tate n'est pas directement utilisable telle qu'elle est écrite. En effet, elle fait intervenir deux groupes quotients qui ne permettent pas facilement, par exemple, d'obtenir un représentant unique. Nous commençons donc par simplifier le groupe d'arrivée du couplage en définissant le couplage de Tate réduit. Cela consiste à rajouter une étape d'**exponentiation finale** afin d'obtenir une racine ℓ-ième de l'unité. En effet, comme $\mathbb{F}_{q^k}^*$ est cyclique, l'application $\mathbb{F}_{q^k}^* \to \mathbb{F}_{q^k}^*, x \mapsto x^{\frac{q^k-1}{\ell}}$ a $(\mathbb{F}_{q^k}^*)^\ell$ pour noyau et μ_l pour image, définissant ainsi un isomorphisme entre $\mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^\ell$ et μ_ℓ.

Définition 3.9 (Couplage de Tate réduit)**.** Le couplage de Tate réduit, notée e_ℓ, est défini comme

$$\begin{array}{rccccc} e_\ell : & \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell] & \times & \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})/\ell\,\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k}) & \longrightarrow & \mu_\ell \\ & (\overline{D_1} & , & \overline{D_2}) & \longmapsto & \langle \overline{D_1}, \overline{D_2} \rangle_\ell^{\frac{q^k-1}{\ell}}. \end{array}$$

Il nous reste alors à nous occuper du groupe du second argument du couplage :

$$\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})/\ell\,\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k}).$$

Comme nous avons pris ℓ premier, nous pouvons représenter ce groupe quotient par les éléments de ℓ-torsion afin de travailler avec une définition plus symétrique du couplage de Tate. Cette transformation est conditionnée par le fait qu'il n'existe aucun diviseur $\mathbb{F}_{q^k}$-rationnel d'ordre ℓ^2 dans la jacobienne ; c'est une hypothèse que nous atteindrons facilement en pratique pour les courbes que nous choisirons.

Théorème 3.10. *Pour ℓ premier, s'il n'existe aucun diviseur $\mathbb{F}_{q^k}$-rationnel d'ordre ℓ^2 dans $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})$, alors le couplage de Tate réduit peut être calculé dans les ensembles suivants :*

$$e_\ell : \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell] \times \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell] \longrightarrow \mu_\ell.$$

Preuve. Nous commençons cette preuve par un argument de structure montrant que la ℓ-torsion $\mathbb{F}_{q^k}$-rationnelle $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell]$ et le quotient $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})/\ell\,\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})$ sont isomorphes. Remarquons que ceci est vrai sans avoir à supposer qu'il n'existe aucun de diviseur d'ordre ℓ^2. La structure de groupe de la jacobienne est connue :

$$\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k}) \cong (\mathbb{Z}/n_1\mathbb{Z}) \oplus \cdots \oplus (\mathbb{Z}/n_{2g}\mathbb{Z}),$$

avec $n_1 \mid \cdots \mid n_{2g}$. Nous considérons alors d la dimension de la ℓ-torsion $\mathbb{F}_{q^k}$-rationnelle en tant que $\mathbb{Z}/\ell\mathbb{Z}$-espace vectoriel. Nous avons alors que

$$\ell \mid n_{2g-d+1} \mid \cdots \mid n_{2g} \text{ et, que pour tout } i \leq 2g-d, \mathrm{pgcd}\,(\ell, n_i) = 1 \text{ car } \ell \text{ est premier.}$$

Nous pouvons alors calculer la structure du groupe quotient :

$$\begin{aligned} \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})/\ell\,\mathrm{Jac}_{\mathcal{H}}(F_{q^k}) &\cong ((\mathbb{Z}/n_1\mathbb{Z}) \oplus \cdots \oplus (\mathbb{Z}/n_{2g}\mathbb{Z}))/\ell((\mathbb{Z}/n_1\mathbb{Z}) \oplus \cdots \oplus (\mathbb{Z}/n_{2g}\mathbb{Z})) \\ &\cong (\mathbb{Z}/n_1\mathbb{Z})/\ell(\mathbb{Z}/n_1\mathbb{Z}) \oplus \cdots \oplus (\mathbb{Z}/n_{2g}\mathbb{Z})/\ell(\mathbb{Z}/n_{2g}\mathbb{Z}) \\ &\cong \{0\} \oplus \cdots \oplus \mathbb{Z}/\ell\mathbb{Z} \\ &\cong (\mathbb{Z}/\ell\mathbb{Z})^d, \end{aligned}$$

qui est également la structure de la ℓ-torsion $\mathbb{F}_{q^k}$-rationnelle : $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell] \cong (\mathbb{Z}/\ell\mathbb{Z})^d$.

En vue de rendre explicite l'isomorphisme entre ces deux groupes, nous considérons l'injectivité de l'application suivante :

$$\begin{array}{ccc} \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell] & \longrightarrow & \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})/\ell\,\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k}) \\ \overline{D} & \longmapsto & \overline{D}. \end{array}$$

Soient $\overline{D_1}$ et $\overline{D_2}$ deux éléments de $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell]$ appartenant à la même classe d'équivalence modulo $\ell\,\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})$. Il existe alors $\overline{D} \in \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})$ tel que

$$\overline{D_1} - \overline{D_2} = \ell\overline{D}.$$

En multipliant cette relation par ℓ, nous obtenons

$$0 = \ell\overline{D_1} - \ell\overline{D_2} = \ell^2\overline{D},$$

car $\overline{D_1}$ et $\overline{D_2}$ sont des éléments de la ℓ-torsion. Comme il n'existe pas d'éléments d'ordre ℓ^2, $\overline{D}$ est donc nul ou d'ordre ℓ, d'où $\overline{D_1} = \overline{D_2}$. Nous avons ainsi l'injection souhaitée. L'argument de cardinalité montre alors que cette injection est un isomorphisme.

En composant cet isomorphisme avec le couplage de Tate en sa deuxième variable, nous obtenons la forme désirée. □

Dans la suite nous supposerons qu'il n'existe pas d'élément $\mathbb{F}_{q^k}$-rationnel d'ordre ℓ^2 dans la jacobienne et nous écrirons ainsi le couplage de Tate réduit sous la forme suivante :

$$\begin{array}{rccccc} e_\ell : & \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell] & \times & \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell] & \longrightarrow & \mu_\ell \\ & (\overline{D_1} & , & \overline{D_2}) & \longmapsto & f_{\ell,D_1}(D_2)^{\frac{q^k-1}{\ell}}. \end{array}$$

Chapitre 4

Couplages : considérations cryptographiques

Nous avons maintenant à notre disposition tous les objets mathématiques qui conduiront à l'obtention de couplages valables pour une utilisation cryptographique. Nous commençons donc ici leur construction. Elle se poursuivra dans la partie suivante où nous aurons à cœur de rendre calculable efficacement les couplages que nous définissons ici. Dans une première section, nous verrons comment utiliser le couplage de Tate pour obtenir un couplage conforme à la Définition 1.3 ; puis nous étudierons la sécurité de la primitive cryptographique ainsi obtenue. Enfin, nous donnerons une liste de courbes utilisables en pratique dans une dernière section.

4.1 Couplage de Tate réduit pour la cryptographie

4.1.1 Choix des groupes $\mathbb{G}_1$, $\mathbb{G}_2$ et $\mathbb{G}_T$

Soient $\mathcal{H}$ une courbe hyperelliptique définie sur $\mathbb{F}_q$, ℓ un nombre premier ne divisant pas q et k le degré de plongement correspondant, c'est-à-dire l'ordre de q dans $(\mathbb{Z}/\ell\mathbb{Z})^*$. Nous supposons également qu'il n'existe pas d'élément d'ordre ℓ^2 dans $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})$. Nous nous retrouvons ainsi dans la situation étudiée jusqu'ici, et nous pouvons définir le couplage de Tate réduit sur la ℓ-torsion de $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})$:

$$e_\ell : \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell] \times \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell] \longrightarrow \mu_\ell.$$

Nous cherchons maintenant à nous placer dans des sous-groupes cycliques d'ordre ℓ afin d'obtenir un couplage cryptographique conforme à la Définition 1.3, c'est-à-dire de la forme

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T,$$

où $\mathbb{G}_1$, $\mathbb{G}_2$ et $\mathbb{G}_T$ sont des groupes cycliques d'ordre premier ℓ.

Nous devrons donc sélectionner deux sous-groupes cycliques d'ordre ℓ de $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell]$ pour $\mathbb{G}_1$ et $\mathbb{G}_2$ tout en vérifiant que la restriction du couplage à $\mathbb{G}_1 \times \mathbb{G}_2$ soit non-dégénérée. Le groupe d'arrivée $\mathbb{G}_T$ ne demande quant à lui aucun choix et nous prendrons directement l'ensemble μ_ℓ des racines ℓ-ièmes de l'unité.

Sans perte de généralité, nous supposons de plus que $\ell \mid \#\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)$, c'est-à-dire qu'il existe un point non trivial $\mathbb{F}_q$-rationnel de ℓ-torsion. Bien que rien ne garantisse l'existence d'un tel point, nous pourrons toujours nous placer dans une extension $\mathbb{F}_{q^d}$ du corps de base $\mathbb{F}_q$ telle que $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^d})[\ell] \neq \{0\}$ car $\ell \nmid q$. Nous pouvons alors définir le couplage de Tate réduit en prenant $\mathbb{F}_{q^d}$ comme corps de base :

$$e_\ell : \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{(q^d)^{k'}})[\ell] \times \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{(q^d)^{k'}})[\ell] \longrightarrow \mu_\ell,$$

avec $k' = \mathrm{ppcm}\,(d,k)/d = k/\,\mathrm{pgcd}\,(d,k)$ l'ordre de q^d dans $(\mathbb{Z}/\ell\mathbb{Z})^*$. Il est à noter que, dans ce cas, les racines ℓ-ièmes de l'unité sont quant à elles toujours définies sur $\mathbb{F}_{q^k}$, qui est un sous-corps propre de $\mathbb{F}_{(q^d)^{k'}}$ lorsque $d \nmid k$.

Dorénavant, nous supposerons toujours que $\ell \mid \#\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)$. Nous pouvons donc prendre un sous-groupe cyclique d'ordre ℓ de $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)[\ell]$ pour $\mathbb{G}_1$ ou $\mathbb{G}_2$. Cependant, lorsque $k > 1$, nous ne pouvons pas choisir $\mathbb{G}_1$ et $\mathbb{G}_2$ tous deux $\mathbb{F}_q$-rationnels : pour tous $\overline{D_1}, \overline{D_2} \in \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)[\ell]$ et tout $\sigma \in \mathbb{G}_{\overline{\mathbb{F}_q}/\mathbb{F}_q}$, nous avons

$$\sigma(e_\ell(\overline{D_1}, \overline{D_2})) = e_\ell(\overline{D_1}^\sigma, \overline{D_2}^\sigma) = e_\ell(\overline{D_1}, \overline{D_2}) \ ;$$

ainsi $e_\ell(\overline{D_1}, \overline{D_2}) \in \mathbb{F}_q$. Or toute racine ℓ-ième de l'unité différente de 1 ne peut être définie sur un sous-corps propre de $\mathbb{F}_{q^k}$. Soit $\zeta \in \mu_\ell \setminus \{1\}$. Si nous supposons que $\zeta \in \mathbb{F}_{q^{k'}}$ avec $k' < k$, alors comme μ_ℓ est cyclique et ℓ est premier, le groupe des racines ℓ-ième et engendré par ζ et $\ell \mid q^{k'} - 1$, ce qui est contradictoire avec la définition du degré de plongement k.

Ainsi, le couplage de Tate restreint à $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)[\ell] \times \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)[\ell]$ est toujours dégénéré quand $k > 1$. Par la même démonstration, nous pouvons également montrer une condition nécessaire (mais non suffisante) à l'obtention d'un couplage non dégénéré : il faut qu'au moins l'un des groupes $\mathbb{G}_1$ ou $\mathbb{G}_2$ contienne des éléments définis sur $\mathbb{F}_{q^k}$ et non dans un sous-corps propre de celui-ci.

Le cas particulier de $k = 1$ est traité avec plus de détails dans la Section IX.7 de [BSS05] avec notamment l'opportunité qu'il fournit de construire des autocouplages (*self-pairings*), c'est-à-dire des couplages sur $\mathbb{G}_1 \times \mathbb{G}_1$.

Nous nous plaçons ici dans le cas où $k > 1$. Il s'agit maintenant de construire le second groupe cyclique dont les éléments sont définis sur $\mathbb{F}_{q^k}$. Soit $\overline{D}$ un élément non trivial de $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)[\ell]$. Considérons alors l'action de π_q, l'endomorphisme de Frobenius associé à l'élévation à la puissance q sur la ℓ-torsion $\mathrm{Jac}_{\mathcal{H}}[\ell]$ vue comme un $\mathbb{Z}/\ell\mathbb{Z}$-espace vectoriel. Comme $\overline{D}$ est $\mathbb{F}_q$-rationnel, $\pi_q(\overline{D}) = \overline{D}$. L'élément $\overline{D}$ est donc un vecteur propre de valeur propre 1. Comme vu en Section 2.2.7, l'action de π_q sur $\mathrm{Jac}_{\mathcal{H}}[\ell]$ admet alors aussi la valeur propre q modulo ℓ. Soit $\overline{D'} \in \mathrm{Jac}_{\mathcal{H}}[\ell]$ un vecteur propre de valeur propre q modulo ℓ. Nous vérifions alors que k est le plus petit entier tel que

$$\pi_{q^k}(\overline{D'}) = \pi_q^k(\overline{D'}) = q^k \overline{D'} = \overline{D'},$$

puisque $k > 1$ est le plus petit entier tel que $q^k \equiv 1 \pmod{\ell}$. Ainsi $\overline{D'}$ est un élément de $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell]$ mais n'est défini sur aucun sous-corps propre de $\mathbb{F}_{q^k}$. Il est alors possible de prendre pour $\mathbb{G}_1$ ou $\mathbb{G}_2$ le sous-groupe cyclique engendré par $\overline{D'}$.

Remarquons que cette construction nous donne une caractérisation équivalente du degré de plongement : k est le plus petit entier tel que le sous-espace de valeur propre q de la ℓ-torsion est $\mathbb{F}_{q^k}$-rationnel [Gra+07].

Dans la suite, nous considérerons toujours que l'un des deux groupes $\mathbb{G}_1$ ou $\mathbb{G}_2$ est défini sur $\mathbb{F}_q$, et l'autre sur $\mathbb{F}_{q^k}$. Bien souvent, il sera souhaitable de choisir ce dernier dans le sous-espace propre de π_q associé à la valeur propre $q \bmod \ell$. Il est à noter que la non-dégénérescence du couplage de Tate restreint à deux tels groupes n'est pas garantie pour autant, et devra être vérifiée au cas par cas.

Cependant, le cas des courbes elliptiques ($g = 1$) est plus simple et nous avons alors une preuve de la non-dégénérescence du couplage de Tate ainsi restreint. Celle-ci provient de la structure de la ℓ-torsion en tant qu'espace vectoriel, de dimension au plus $2g = 2$. Nous supposons ici $k > 1$. Comme nous venons d'exhiber deux vecteurs linéairement indépendants $(\overline{D}, \overline{D'})$, nous avons une base de la ℓ-torsion. Par la non-dégénérescence du couplage de Tate nous savons qu'il existe $\overline{D''}$ tel que $\langle \overline{D}, \overline{D''} \rangle_\ell \notin (\mathbb{F}_{q^k})^\ell$. Nous pouvons alors écrire $\overline{D''}$ selon la base $(\overline{D}, \overline{D'})$: $\overline{D''} = [a]\overline{D} + [b]\overline{D'}$, avec $b \not\equiv 0 \pmod{\ell}$. Nous aurons alors

$$\langle \overline{D}, \overline{D''} \rangle_\ell = \langle \overline{D}, \overline{D} \rangle_\ell^a \cdot \langle \overline{D}, \overline{D'} \rangle_\ell^b = \langle \overline{D}, \overline{D'} \rangle_\ell^b \notin (\mathbb{F}_{q^k}^*)^\ell.$$

Ce qui montre que le couplage restreint à $\langle \overline{D} \rangle \times \langle \overline{D'} \rangle$ est non-dégénéré. Il en va de même pour le couplage restreint à $\langle \overline{D'} \rangle \times \langle \overline{D} \rangle$. La propriété essentielle des courbes elliptiques que nous avons utilisées ici est que la dimension de la ℓ-torsion est au plus 2, ce qui n'est plus vrai en genre supérieur.

4.1.2 Utilisation d'une *distortion map*

Nous commençons par définir la notion de *distortion map*.

Définition 4.1. Étant donnée une courbe hyperelliptique $\mathcal{H}$, une **distortion map** pour le couplage de Tate et associée à $\overline{D_1}$ et $\overline{D_2}$, deux éléments d'ordre premier ℓ de la jacobienne de $\mathcal{H}$, est un endomorphisme ψ de $\mathrm{Jac}_\mathcal{H}$ tel que $e_\ell(\overline{D_1}, \psi(\overline{D_2})) \neq 1$.

Remarque 4.2. Une conséquence de l'existence d'une *distortion map* associée à $\overline{D_1}$ et $\overline{D_2}$ est la possibilité de déduire du couplage de Tate un couplage non-dégénéré sur les groupes $\mathbb{G}_1 = \langle \overline{D_1} \rangle$ et $\mathbb{G}_2 = \langle \overline{D_2} \rangle$.

Théorème 4.3 (Verheul-Schoof [Ver01, Théorème 5 ; Gal+09, Théorème 2.1])**.** *Si la courbe hyperelliptique $\mathcal{H}$, et donc sa jacobienne, sont supersingulières, alors pour tous $\overline{D_1}$ et $\overline{D_2}$ éléments d'ordre ℓ de la jacobienne, il existe un endomorphisme ψ de la jacobienne* $\mathrm{Jac}_\mathcal{H}$ *tel que $e_\ell(\overline{D_1}, \psi(\overline{D_2})) \neq 1$.*

Quels que soient les groupes cycliques $\mathbb{G}_1$ et $\mathbb{G}_2 \subset \mathrm{Jac}_\mathcal{H}[\ell]$ choisis, il est ainsi possible de construire un couplage non-dégénéré restreint au domaine $\mathbb{G}_1 \times \mathbb{G}_2$. En particulier, nous pouvons choisir de restreindre les deux arguments du couplage de Tate à un même sous-groupe $\mathbb{G}_1$, possiblement $\mathbb{F}_q$-rationnel, pour obtenir un couplage de type I (encore appelé couplage symétrique) [GPS08]. Nous introduisons alors la définition du couplage de Tate modifié qui reflète ce choix de sous-groupes.

Définition 4.4 (Couplage de Tate modifié)**.** Le couplage de Tate modifié, aussi appelé le couplage de Tate de type I, est défini comme

$$\begin{array}{rccccc} \hat{e}_\ell : & \mathbb{G}_1 & \times & \mathbb{G}_1 & \longrightarrow & \mu_\ell \\ & (\overline{D_1} & , & \overline{D_2}) & \longmapsto & e_\ell(\overline{D_1}, \psi(\overline{D_2})), \end{array}$$

où ψ est une *distortion map* pour le couplage de Tate associée à $\overline{D}$, générateur de $\mathbb{G}_1$.

L'intérêt de pouvoir avoir un couplage non-dégénéré sur $\mathbb{G}_1 \times \mathbb{G}_2$, avec $\mathbb{G}_1$ et $\mathbb{G}_2$ tous deux $\mathbb{F}_q$-rationnels, est multiple. D'une part, nous bénéficions de groupes dont la représentation est bien plus compacte que si l'un des groupes est strictement $\mathbb{F}_{q^k}$-rationnel lorsque $k > 1$. L'efficacité de communication d'un protocole reposant sur les couplages s'en trouve donc automatiquement améliorée. D'autre part, ces protocoles utilisent souvent des calculs supplémentaires dans les groupes $\mathbb{G}_1$ et $\mathbb{G}_2$ tels que des multiplications scalaires : ceux-ci pourront être effectués dans $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)$ et non dans $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})$, ce qui peut représenter un gain en performance important dès lors que $k > 1$. Cependant, cela n'est pas toujours possible, car l'existence de *distortion maps* est conditionnée par la proposition suivante.

Proposition 4.5 (Pellikaan-Verheul [Ver04, Théorème 6])**.** *Soient $\overline{D_1}$ et $\overline{D_2} \in \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)[\ell]$ non nuls. Si $k > 1$ et s'il existe une distortion map pour le couplage de Tate associée à $\overline{D_1}$ et $\overline{D_2}$, alors l'anneau d'endomorphismes de $\mathrm{Jac}_{\mathcal{H}}$ est non-commutatif.*

Preuve. Soit ψ la *distortion map* en question. Nous avons donc $e_\ell(\overline{D_1}, \psi(\overline{D_2})) \neq 1$. Comme $k > 1$ et $\overline{D_1} \in \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)$, alors $\psi(\overline{D_2})$ n'est pas $\mathbb{F}_q$-rationnel. Ainsi, $\psi \circ \pi_q(\overline{D_2}) = \psi(\overline{D_2}) \neq \pi_q \circ \psi(\overline{D_2})$. Les deux endomorphismes π_q et ψ ne commutent donc pas. □

Nous déduisons de ce théorème et de la caractérisation des courbes elliptiques supersingulières (Théorème 2.29) que seules celles-ci nous permettent la construction du couplage de Tate modifié dans le cas du genre 1.

Corollaire 4.6. *En genre 1 et dans le cas où $k > 1$, seules les courbes supersingulières possèdent une distortion map pour calculer le couplage de Tate modifié sur $\mathbb{G}_1 \times \mathbb{G}_1$, avec $\mathbb{G}_1$ $\mathbb{F}_q$-rationnel.*

4.1.3 Utilisation d'une tordue

Nous supposons ici que $k > 1$. Comme nous l'avons vu plus haut, dans le cas où il n'existe pas de *distortion map* permettant d'avoir un couplage non-dégénéré sur deux sous-groupes de ℓ-torsion $\mathbb{F}_q$-rationnels, seul l'un des deux groupes $\mathbb{G}_1$ ou $\mathbb{G}_2$ peut être défini sur $\mathbb{F}_q$. Considérons sans perte de généralité qu'il s'agit ici de $\mathbb{G}_1$, et que $\mathbb{G}_2$ ne peut alors être défini que sur $\mathbb{F}_{q^k}$. Il est tout de même possible de tirer parti de l'existence de tordues (Section 2.1.5) pour pouvoir définir un groupe $\mathbb{G}_2'$ sur un sous-corps propre de $\mathbb{F}_{q^k}$ ainsi qu'un isomorphisme explicite entre $\mathbb{G}_2'$ et $\mathbb{G}_2$.

Définition 4.7 (Tordue)**.** Soient $\mathcal{H}$ et $\mathcal{H}'$ deux courbes hyperelliptiques définies sur $\mathbb{F}_q$. La courbe $\mathcal{H}'$ est une **tordue** d'ordre d de $\mathcal{H}$ s'il existe un isomorphisme ψ_d de $\mathcal{H}'$ vers $\mathcal{H}$ défini sur $\mathbb{F}_{q^d}$, avec d minimal pour cette propriété.

Nous détaillons maintenant l'intérêt de l'existence d'une tordue pour le calcul de couplage. Soient $\mathcal{H}$ une courbe hyperelliptique définie sur $\mathbb{F}_q$, ℓ un nombre premier divisant $\#\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)$ et k le degré de plongement associé. Nous supposons $k > 1$ et que $\mathcal{H}$ admet une tordue $\mathcal{H}'$ de degré d divisant k et telle que $\ell \mid \#\mathrm{Jac}_{\mathcal{H}'}(\mathbb{F}_{q^e})$ avec $e = k/d$. L'isomorphisme $\psi_d : \mathcal{H}' \to \mathcal{H}$ issu de la définition de tordue nous donne alors une injection explicite de $\mathrm{Jac}_{\mathcal{H}'}(\mathbb{F}_{q^e})$ dans $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})$. Si de plus $\ell \nmid d$, alors $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)[\ell] \cap \psi_d(\mathrm{Jac}_{\mathcal{H}'}(\mathbb{F}_{q^e})[\ell]) = \{\overline{0}\}$. Il est donc possible de définir un couplage non-dégénéré entre $\mathbb{G}_1 \subset \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)[\ell]$ et $\mathbb{G}_2' \subset \mathrm{Jac}_{\mathcal{H}'}(\mathbb{F}_{q^e})[\ell]$ en se basant sur le couplage de Tate réduit entre $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)[\ell]$ et $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell]$:

$$\begin{array}{rccccc} \hat{e}_\ell : & \mathbb{G}_1 & \times & \mathbb{G}_2' & \longrightarrow & \mu_\ell \\ & (\overline{D_1} & , & \overline{D_2}) & \longmapsto & e_\ell(\overline{D_1}, \psi_d(\overline{D_2})). \end{array}$$

Nous manipulerons donc a priori des points dans une extension de degré d fois moindre que sans tordue. Cela peut faire une énorme différence en pratique.

Comme détaillé dans [HSV06], l'existence de telles tordues est conditionnée par la structure du groupe d'automorphismes de $\mathcal{H}$. En effet, étant donnée $\mathcal{H}'$ une tordue d'ordre d de $\mathcal{H}$, ainsi que l'isomorphisme associé ψ_d, la construction $\psi_d^{\pi_q} \circ \psi_d^{-1}$ — où $\psi_d^{\pi_q}$ dénote l'action du Frobenius π_q relatif à l'élévation à la puissance q-ième sur les coefficients de ψ_d — définit un automorphisme de $\mathcal{H}$. Si cet automorphisme est de surcroît défini sur $\mathbb{F}_q$ (comme cela est le cas pour les courbes elliptiques ordinaires, par exemple), alors il est d'ordre d : pour qu'une tordue d'ordre d existe, il faut donc que $\mathrm{Aut}(\mathcal{H})$ contiennent un sous-groupe cyclique d'ordre d. Comme les courbes ayant des automorphismes non triviaux sont bien classifiées au moins pour les courbes hyperelliptiques qui nous intéressent (à commencer par le genre $g = 1$ et le genre $g = 2$[1]), cela fournit des candidats pour construire des courbes admettant des tordues.

4.2 Cryptanalyse des couplages

La sécurité d'un couplage $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ repose sur l'impossibilité, tout du moins calculatoire, de résoudre des problèmes conjecturés durs tels que ceux de Diffie-Hellman bilinéaire, l'inversion de couplage et la résolution du logarithme discret sur les groupes $\mathbb{G}_1$, $\mathbb{G}_2$ et $\mathbb{G}_T$. Nous dirons alors qu'un couplage atteint n bits de sécurité si la plus efficace des attaques connues auxquelles il est sujet demande au moins 2^n opérations élémentaires pour être menée à bien.

Les connaissances actuelles se limitent à des algorithmes pour résoudre le DLP dans ces différents groupes et n'offrent pas de solution pour les autres problèmes (Diffie-Hellman bilinéaire, inversion). Nous allons donc estimer la complexité du DLP dans chacun des groupes afin de définir la sécurité d'un couplage. Ainsi, nous pourrons choisir correctement ses paramètres en fonction du niveau de sécurité requis.

Rappelons que le problème du logarithme discret pour un groupe $(\mathbb{G}, \cdot)$ cyclique consiste à calculer l'entier a modulo $\#\mathbb{G}$ tel que $h = g^a$, étant seulement donnés g un générateur de $\mathbb{G}$ et $h \in \mathbb{G}$. Il est important de noter que nous nous sommes réduits au cas où $\mathbb{G}$ est cyclique sans perte de généralité ; en effet, dans le cas contraire, il suffirait de travailler dans $\langle g \rangle$, le sous-groupe engendré par g.

Nous décrivons donc dans ce chapitre les attaques aux problèmes du logarithme discret pour les différents groupes qui interviennent (essentiellement la ℓ-torsion d'une jacobienne de courbe hyperelliptique et le groupe de racines ℓ-ièmes de l'unité) et conclurons alors en donnant des recommandations sur le choix de la taille du corps de base q et le degré de plongement k pour différents niveaux de sécurité.

4.2.1 Attaques génériques sur les jacobiennes de courbes

Nous cherchons dans cette section à évaluer la complexité du DLP dans le groupe de torsion $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell]$ dans lequel un couplage prend ses entrées.

Un groupe générique, selon le modèle de Shoup [Sho97] par exemple, est un groupe dont la représentation est inconnue et qui est muni d'un certain nombre d'oracles pour calculer les opérations « simples » comme la loi de groupe, tirer un élément au hasard, ou encore tester

1. Les cas des genres supérieurs n'est résolu que plus partiellement[LR12].

l'égalité. Il existe une grande variété dans ces modèles mais le résultat essentiel reste que le problème du logarithme discret dans un groupe générique cyclique d'ordre premier demande au plus $\Omega(\sqrt{\#\mathbb{G}})$ opérations élémentaires pour être résolu.

Cette complexité donne ainsi une borne supérieure sur la sécurité atteignable par un groupe de cardinal donné. Il est alors intéressant de se munir de groupes se comportant comme des groupes génériques, au moins de façon conjecturale. Les courbes algébriques sont de bonnes candidates pour construire de tels groupes ; en effet, leur structure algébrique dévoile, *a priori*, trop peu d'information exploitable pour résoudre le DLP. Cependant, il existe certaines familles de courbes qui sont sujettes à des attaques efficaces. Nous pensons en premier lieu aux courbes qui ont un « petit » degré de plongement et pour lesquelles Menezes, Okamoto et Vanstone d'une part, et Frey et Rück d'autre part, ont montré que le DLP se transfère de la courbe au groupe d'arrivée du couplage [MOV93 ; FR94]. Nous verrons dans la prochaine section que les courbes de genre ≥ 3 sont elles aussi attaquables. Enfin, il faut aussi éviter un certain nombre de cas particuliers : les courbes anomales (dont la trace est 1) [Sma99], le groupe de p-torsion en caractéristique p [Sem98], etc.

De plus, la ℓ-torsion est un groupe d'exposant premier ℓ ; ainsi, quel que soit le sous-groupe cyclique non trivial considéré, il n'est pas possible d'utiliser l'algorithme de Pohlig-Hellman qui accélère les calculs quand l'ordre du groupe est composé [PH78].

§ 1. Algorithme de Shanks. Soit $\mathbb{G} = \langle g \rangle$ un groupe cyclique d'ordre n, et h un élément de $\mathbb{G}$ dont nous cherchons le logarithme discret par rapport à g. L'algorithme de Shanks — ou encore *baby-step giant-step* (BSGS) — est un compromis temps-mémoire permettant de calculer le logarithme en base g de h : $\mathrm{dlog}_g\, h$. L'algorithme consiste en deux étapes dépendant d'un paramètre N :

— le calcul des **pas de bébé**, c'est-à-dire la liste des g^i pour i allant de 0 à $N-1$,
— le calcul des **pas de géant** : pour chaque j dans $\{0, \ldots, \lfloor n/N \rfloor\}$, chercher si $h \cdot g^{-jN}$ est présent dans la liste précédente.

Lorsque qu'une telle collision est trouvée, nous obtenons une relation de la forme

$$g^i = h \cdot g^{-jN},$$

dont nous pouvons facilement déduire le logarithme discret :

$$\mathrm{dlog}_g(h) \equiv i + jN \pmod{n}.$$

Cette collision existe et sera trouvée par l'algorithme de Shanks car i et j correspondent au reste et au quotient de la division euclidienne de $\mathrm{dlog}_g(h)$ par N.

En prenant $N \approx \sqrt{n}$, il faut $O(\sqrt{n})$ opérations de groupe pour un calcul de logarithme discret. Cependant, cet algorithme demande aussi de stocker $O(\sqrt{n})$ éléments du groupe, ce qui représente une hypothèse irréaliste dans le cas d'un groupe de taille cryptographique.

§ 2. Algorithme ρ de Pollard. Afin d'obtenir la même complexité en utilisant un espace mémoire de taille constante, Pollard a proposé d'utiliser le **paradoxe des anniversaires** [Pol78]. Le principe est de parcourir de façon aléatoire le groupe pour construire une relation entre une puissance du générateur et une puissance de l'élément dont nous cherchons le logarithme discret.

Soient $\mathbb{G} = \langle g \rangle$ un groupe cyclique d'ordre n, $h \in \mathbb{G}$ et $f : \mathbb{G} \to \mathbb{G}$ une fonction définissant une marche déterministe pseudo-aléatoire $x_0, x_1 = f(x_0), \ldots, x_i = f(x_{i-1}) = f^i(x_0), \ldots$

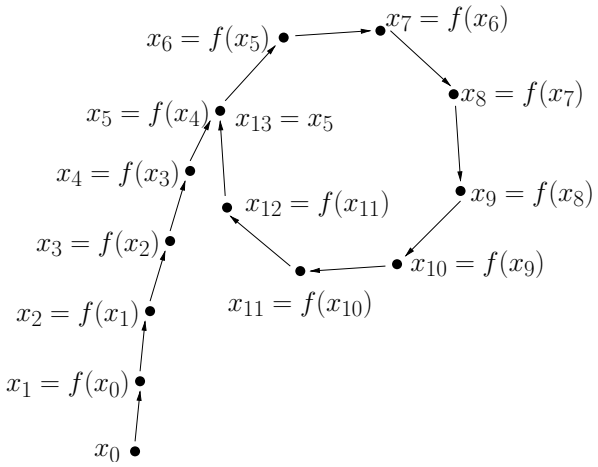


FIGURE 4.1 – Algorithme ρ de Pollard.

dans $\mathbb{G}$. Le paradoxe des anniversaires implique alors que nous trouvions une collision $x_i = x_j$ avec $i \neq j$ après, en moyenne, $\sqrt{\pi n/2}$ étapes. Comme cette marche aléatoire forme une suite ultimement périodique, elle est souvent représentée sous la forme de la lettre ρ (Figure 4.1), c'est ce qui a donné son nom à cet algorithme. La détection de la collision peut se faire grâce à la méthode de détection de cycle de Floyd [Knu97, Section 3.1, exercice 6] et ne demande alors de stocker que deux éléments du groupe en mémoire : au départ les deux éléments sont initialisés à x_0, puis à chaque étape nous appliquons une fois f à l'un des éléments et deux fois à l'autre avant de tester la collision. C'est pourquoi cette méthode est aussi appelée celle du lièvre et de la tortue.

Dans le but de retrouver effectivement la valeur de $\mathrm{dlog}_g(h)$, la fonction f est construite ainsi :

$$\begin{array}{rccc} f : & \mathbb{G} & \longrightarrow & \mathbb{G} \\ & x & \longmapsto & x \cdot g^{\alpha_{\varphi(x)}} \cdot h^{\beta_{\varphi(x)}}, \end{array}$$

où $\varphi : \mathbb{G} \to \{0, \ldots, r-1\}$ découpe $\mathbb{G}$ en r espaces distincts de taille comparable et les α_i, β_i sont des constantes dans $\mathbb{Z}/(\#\mathbb{G})\mathbb{Z}$. En partant de $x_0 = g$ par exemple, une collision donnera une égalité de la forme :

$$g^a \cdot h^b = g^{a'} \cdot h^{b'},$$

dont nous déduisons le logarithme discret : $\mathrm{dlog}_g(h) \equiv (a - a')/(b - b') \pmod{n}$. Dans le cas où $b \equiv b' \pmod{n}$, il faut recommencer le processus en partant d'un x_0 différent.

Cette méthode peut être généralisée afin de pouvoir être exécutée en parallèle sur une grappe de calcul par exemple. Il s'agit de la méthode λ de Pollard qui consiste à calculer plusieurs marches aléatoires partant d'éléments distincts du groupe et d'arrêter ces marches lorsqu'elles passent par un élément distingué de $\mathbb{G}$. Une collision est alors cherchée parmi ces éléments distingués [OW99]. Un bon choix de la proportion d'éléments de $\mathbb{G}$ qui sont distingués permet d'atteindre la même complexité asymptotique que l'algorithme ρ de Pollard [Gal10]. Des travaux récents [GH12 ; GPR13] permettent également d'améliorer la constante de complexité devant le $\sqrt{\#\mathbb{G}}$.

De plus amples détails sur les méthodes de calcul de logarithme discret utilisant des marches aléatoires peuvent être trouvés au Chapitre 14 de [Gal12].

TABLE 4.2 – Accélération du DLP pour les groupes de torsion de quelques courbes.

Courbe	Groupe de torsion considéré	$\#\mathrm{Aut}(\mathcal{H})$	Facteur d'accélération
E_{BN}	$E_{\mathrm{BN}}(\mathbb{F}_p)[\ell]$	6	$\frac{1}{\sqrt{6}}$
E_2	$E_2(\mathbb{F}_{2^m})[\ell]$	24	$\frac{1}{\sqrt{24 \cdot m}}$
E_3	$E_3(\mathbb{F}_{3^m})[\ell]$	12	$\frac{1}{\sqrt{12 \cdot m}}$
H_2	$\mathrm{Jac}_{H_2}(\mathbb{F}_{2^m})[\ell]$	32	$\frac{1}{\sqrt{32 \cdot m}}$

Voir Section 4.3 pour la définition de ces courbes.

§ 3. Utilisation des automorphismes de courbes. Dans le cas particulier des groupes de torsion sur une courbe elliptique ou hyperelliptique $\mathcal{H}$, Duursma, Gaudry et Morain ont montré en 1999 qu'il est possible d'utiliser le groupe d'automorphismes du sous-groupe cyclique d'ordre ℓ considéré pour réduire l'espace de recherche [DGM99]. L'attaque est alors conduite en utilisant l'algorithme ρ de Pollard sur le groupe quotient $\mathbb{G}/\sim$ où

$$\forall g, h \in \mathbb{G}, (g \sim h \Leftrightarrow \exists \sigma \in \mathrm{Aut}(\mathbb{G}) \text{ tel que } g = \sigma(h)) .$$

Cette idée a aussi été présentée plus tôt la même année, dans le cas des courbes elliptiques seulement, par Weiner et Zuccherato [WZ99]. Cette approche permet de réduire d'un facteur $\#\mathrm{Aut}(\mathbb{G})$ la taille du groupe dans lequel le DLP est calculé et ainsi de gagner un facteur $1/\sqrt{\#\mathrm{Aut}(\mathbb{G})}$ pour cette attaque.

Nous nous intéressons maintenant plus particulièrement au DLP dans $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)[\ell]$. Les automorphismes que nous pourrons utiliser sont alors, d'une part, les automorphismes de la courbe $\mathcal{H}$ et, d'autre part, l'ensemble des itérés de l'endomorphisme de Frobenius fixant le sous-groupe cyclique d'ordre ℓ considéré. De ce fait, si $q = p^{r \cdot s}$ et si la courbe est définie sur $\mathbb{F}_{p^r}$, il est possible d'utiliser l'ensemble des s endomorphismes itérés de Frobenius $(x, y) \mapsto (x^{p^{r \cdot i}}, y^{p^{r \cdot i}})$ pour $i \in \{1, \ldots, s\}$ qui fixent $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)[\ell]$. Si nous nous plaçons dans le cas fréquent où $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)[\ell]$ est cyclique, les restrictions à $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)[\ell]$ de chacun des s endomorphismes de Frobenius sont différentes. Dans ce cas, le nombre moyen de pas dans la marche aléatoire avant de trouver une collision sera

$$\sqrt{\frac{\pi \cdot n}{s \cdot \#\mathrm{Aut}(\mathcal{H})}}.$$

Le Théorème III.10.1 de [Sil86] nous fournit une caractérisation du groupe d'automorphismes pour les courbes elliptiques selon leur j-invariant. Un théorème d'Igusa dénombre les automorphismes des courbes hyperelliptiques supersingulières de genre et caractéristique 2 [CNP05] ; celles-ci sont également décrites précisément à la Section 8 de [Gal+09]. Nous en déduisons la Table 4.2 montrant l'avantage apporté par les automorphismes pour les différentes courbes que nous listons en fin de chapitre et qui sont communément utilisées pour le calcul de couplage.

Ces techniques sont mises en œuvre pour attaquer la courbe ECC2K-131 du challenge proposé par Certicom [Cer97]. Cette attaque qui demande d'effectuer environ $2^{60.9}$ pas,

a été implémentée sur différentes architectures (CPU, GPU, FPGA) et son calcul est en cours [2] [Bai+09].

4.2.2 Courbes de genre ≥ 3

Le DLP pour des courbes de genre supérieur à 3 peut être calculé de façon plus efficace que pour des courbes de genre 1 et 2. Parmi ces courbes de genre supérieur, il faut distinguer celles de genre « grand » pour lesquelles des algorithmes de complexité sous-exponentielle existent [Cou01 ; Hes04b]. Pour le genre fixé « petit », il faut utiliser une adaptation du calcul d'indice. D'un point de vue très schématique, les techniques de calcul d'indice pour un groupe $(\mathbb{G}, \cdot)$ consistent en quatre étapes :

— **Construction d'une base de facteurs** $\mathcal{F}$: choisir un ensemble de « petits » éléments π_i dans $\mathbb{G}$.

— **Recherche de relations** : chercher un ensemble d'au moins $\#\mathcal{F}$ relations de la forme

$$g^a = \prod_i \pi_i^{c_i}.$$

Cela revient à tester la « friabilité » d'un grand nombre de puissances de g.

— **Algèbre linéaire** : utiliser ces relations pour construire un système d'équations linéaires sur $\mathbb{Z}/(\#\mathbb{G})\mathbb{Z}$ de la forme

$$\begin{cases} & \vdots & \\ a & \equiv & \sum_i c_i \cdot \mathrm{dlog}_g(\pi_i) \pmod{\#\mathbb{G}} \\ & \vdots & \end{cases}$$

et le résoudre pour obtenir le logarithme en base g des éléments de la base de facteurs.

— **Logarithme individuel** : chercher une relation de la forme

$$g^a \cdot h^b = \prod_i \pi_i^{c_i} \text{ avec } b \not\equiv 0 \pmod{\#\mathbb{G}},$$

et l'utiliser pour obtenir finalement $\mathrm{dlog}_g(h) \equiv \left(\sum_i c_i \cdot \mathrm{dlog}_g(\pi_i)\right) / b \pmod{\#\mathbb{G}}$.

Nous appliquons ensuite ce schéma à la jacobienne d'une courbe qui est un groupe noté additivement ; nous avons cependant préféré garder une notation multiplicative jusqu'ici, le vocabulaire étant plus adapté.

Dans le cas d'une courbe hyperelliptique $\mathcal{H}$, la base de facteurs $\mathcal{F}$ est construite comme l'ensemble des diviseurs dégénérés dont la représentation de Mumford est $[x - x_0, y_0]$ avec $(x_0, y_0) \in \mathcal{H}(\mathbb{F}_q)$. La notion de friabilité correspondante pour un diviseur $D = [u(x), v(x)]$ est alors le fait que le polynôme $u(x)$ se scinde sur $\mathbb{F}_q[x]$. Tant que le genre est beaucoup plus petit que la taille du corps de base, chaque diviseur a une probabilité en $1/g!$ d'être friable, ce qui est raisonnable, il faudra donc tester environ $qg!$ diviseurs pour obtenir $\#\mathcal{H}(\mathbb{F}_q) \approx q$ relations et pouvoir passer à la phase d'algèbre linéaire. L'algèbre linéaire aura alors un coût quadratique en q et déterminera la complexité totale de l'algorithme en $O(q^2)$.

Il est cependant possible de réduire cette complexité en utilisant une idée due à Harley [Thé03] : réduire la base de facteurs afin de passer moins de temps dans la phase d'algèbre

2. L'avancement du projet peut être consulté sur http://ecc-challenge.info/.

linéaire et équilibrer le coût des phases de recherche de relations et d'algèbre linéaire. Soit $0 < \alpha \leq 1$, en choisissant une base de facteurs de taille q^α (il suffit d'éliminer artificiellement certains diviseurs de degré 1), chaque diviseur aura une probabilité $q^{(\alpha-1)g}/g!$ d'être friable et il faudra essayer environ $q^{\alpha+(1-\alpha)g}g!$ diviseurs pour obtenir q^α relations. La phase d'algèbre linéaire aura alors une complexité en $q^{2\alpha}$. En choisissant la valeur de α qui égalise ces deux complexités, l'algorithme a alors une complexité en $O(q^{2-\frac{2}{g+1}})$.

Enfin, il existe des variantes *single*, puis *double large prime* consistant à autoriser que la décomposition se fasse avec un ou deux diviseurs dans une seconde base de facteurs légèrement plus grande. Cette variante permet d'obtenir une complexité en $\tilde{O}(q^{2-\frac{2}{g}})$ [Théo3 ; Gau+07].

Comme $\#\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q) \approx q^g$ (Théorème 2.49), le calcul d'indice devient plus efficace que les méthodes reposant sur une marche aléatoire à partir de $g = 3$. Par conséquent, les courbes de genre ≥ 3 ne sont généralement pas utilisées en cryptographie.

Ces méthodes sont décrites plus en détail dans [Gau08, Section 2.2] et [Vit12, Section 5.4].

4.2.3 Attaques sur le groupe des racines de l'unité

Dans cette section, nous décrivons le calcul du logarithme discret dans le groupe des racines ℓ-ièmes de l'unité, groupe où le couplage de Tate réduit prend ses valeurs. Comme $\mu_\ell \subset \mathbb{F}_{q^k}^*$, le problème du logarithme discret peut être attaqué soit comme pour le cas d'un groupe générique d'ordre ℓ comme nous l'avons vu en Section 4.2.1, soit par des algorithmes exploitant la structure algébrique de $\mathbb{F}_{q^k}$. Nous verrons que nous obtiendrons des algorithmes sous-exponentiels en q^k, le degré de plongement nous permettant ainsi de contrôler la difficulté du DLP sur μ_ℓ par rapport à celle du DLP sur la courbe. Afin de mesurer ces complexités, nous introduisons ici la fonction L.

Définition 4.8 (Fonction sous-exponentielle)**.** Soient $0 \leq \alpha \leq 1$ et $c > 0$, la fonction sous-exponentielle de paramètres α et c est définie par

$$L_n[\alpha, c] = \exp(c \log(n)^\alpha \log(\log(n))^{1-\alpha}).$$

Nous dénoterons par $L_n[\alpha]$ l'ensemble des fonctions sous-exponentielles $L_n[\alpha, c]$ pour $c > 0$.

Pour des entrées de taille $\log n$, un algorithme sera dit de complexité :
- **polynomiale** s'il demande $O(L_n[0, c]) = O((\log n)^c)$ opérations,
- **exponentielle** dans le cas où $O(L_n[1, c]) = O(n^c)$ opérations sont nécessaires, et enfin
- **sous-exponentielle** pour les valeurs de α intermédiaires.

Les méthodes de marche aléatoire dans un groupe $\mathbb{G}$, par exemple, sont ainsi des algorithmes en $O(L_{\#\mathbb{G}}[1, 1/2])$.

Lorsque la caractéristique est petite, une première idée est d'utiliser le calcul d'indice en utilisant la notion naturelle de friabilité sur les polynômes représentant les éléments de $\mathbb{F}_{q^k}$. La base de facteurs est alors l'ensemble des polynômes irréductibles de taille plus petite qu'une certaine borne. Le théorème de Canfield-Erdős-Pomerance indique qu'avec une borne de friabilité en $L_{q^k}[1/2]$, un polynôme est friable avec une probabilité $\frac{1}{L_{q^k}[1/2]}$ [CEP83]. Ainsi, le DLP sur $\mathbb{F}_{q^k}^*$ peut être attaqué avec une complexité en $L_{q^k}[1/2]$ grâce au calcul d'indice. Cet algorithme est dû à Adleman [Adl79].

Il est aussi possible de résoudre ce problème grâce à des techniques de crible algébrique qui vont utiliser d'autres notions de friabilité et faire baisser la complexité pour obtenir des

algorithmes en $L_{q^k}[1/3]$. Les notions de friabilité vont dépendre de la forme du corps considéré. Dans notre cas, nous cherchons à résoudre le logarithme discret sur $\mathbb{F}_{q^k}^*$ avec $q = p^m$. Or, l'algorithme à utiliser dépend du rapport de taille entre la caractéristique du corps p et son degré d'extension $s = k \cdot m$; il faut distinguer trois cas :

— **petite caractéristique** quand $2 \log \log p \leq \log s$, l'algorithme de crible de corps de fonctions — *function field sieve* (FFS) — est utilisé ;
— **grande caractéristique** quand $\log \log p \geq 2 \log s$, le crible de corps de nombres — *number field sieve* (NFS) — est alors choisi ;
— **cas moyen** pour les valeurs intermédiaires de p, une variante de NFS pour les grands degrés est adoptée (NFS-HD).

En ce qui concerne les couplages, les courbes choisies sont définies soit sur un corps de petite caractéristique $p = 2$ ou 3, si bien qu'il est clair qu'il faut utiliser FFS [3], soit sur un corps premier, auquel cas $\mathbb{F}_{p^k}^*$ tombe dans le domaine de NFS pour les degrés de plongement k considérés en pratique.

§ 1. Crible de corps de nombres. Dans le cas de la grande caractéristique, la factorisation dans un corps de nombres est utilisée [Gor93 ; Sch93] et permet d'obtenir une complexité en

$$L_{q^k}\left[\frac{1}{3}, \sqrt[3]{\frac{64}{9}}\right].$$

§ 2. Crible de corps de fonctions. Ici nous considérons le cas de la petite caractéristique. Le crible algébrique est alors instancié en utilisant une notion de friabilité sur les corps de fonctions. Cette variante a été introduite en 1994 par Adleman [Adl94] et améliorée en 1999 par Adleman et Huang [AH99] pour obtenir une complexité meilleure que celle de NFS en

$$L_{q^k}\left[\frac{1}{3}, \sqrt[3]{\frac{32}{9}}\right].$$

FFS a fait l'objet d'implémentations qui ont permis des calculs records, notamment en 2005, où Joux et Lercier ont résolu le DLP sur $\mathbb{F}_{2^{607}}$ et $\mathbb{F}_{2^{613}}$ [JL05]. Plus récemment, Hayashi *et al.* ont résolu le DLP pour deux cas intéressants vis à vis des couplages : sur $\mathbb{F}_{3^{6\cdot 71}}$ (corps de 675 bits) [Hay+10] et sur $\mathbb{F}_{2^{6\cdot 97}}$ (corps de 923 bits) [Hay+12]. Barbulescu *et al.* ont également résolu le DLP sur $\mathbb{F}_{2^{809}}$ [Bar+13a].

4.2.4 Attaques par descente de Weil

Nous présentons dans cette section un ensemble d'attaques que nous pouvons appliquer à une jacobienne $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)$ quand $q = q'^{m'}$. Dans ce cas, et selon les valeurs de m', la sécurité apportée par la partie $\mathbb{F}_q$-rationnelle de la jacobienne est amoindrie.

En effet, la descente de Weil permet de construire un isomorphisme entre une courbe $\mathcal{H}$ définie sur une extension de corps $\mathbb{F}_q = \mathbb{F}_{q'^{m'}}$ et une variété abélienne de dimension m' définie sur $\mathbb{F}_{q'}$. Cette variété, la **restriction de Weil**, se note $W_{\mathbb{F}_q/\mathbb{F}_{q'}}(\mathcal{H})$ et se définit en choisissant une base de $\mathbb{F}_q$ en tant que $\mathbb{F}_{q'}$-espace vectoriel et en projetant l'équation de la courbe sur celle-ci.

3. Depuis 2013, il faut considérer les travaux évoqués dans la section d'avertissement du Chapitre 1.

§ 1. Attaque Gaudry-Heß-Smart. L'attaque Gaudry-Heß-Smart (GHS) construit ainsi une courbe de genre au moins m' sur $\mathbb{F}_{q'}$ sur laquelle le problème du logarithme discret est transféré. Si la courbe obtenue est effectivement de genre m', cela fournit ainsi une attaque en $\tilde{O}((q')^{2-\frac{2}{m'+1/2}})$ en résolvant le DLP sur cette nouvelle courbe grâce aux algorithmes de calcul d'indice que nous avons vus précédemment [GHS02].

Plus tard, il a été montré que cette complexité était applicable même quand GHS construit une courbe de genre plus grand que m' [Die11 ; Gau09] avec cependant une plus mauvaise constante, dépendant de n, cachée derrière la notation $\tilde{O}(\cdot)$.

§ 2. Problème de Diffie-Hellman statique. L'utilisation d'une courbe sur laquelle la restriction de Weil s'applique introduit également une vulnérabilité face à un autre problème dérivé, mais plus faible, du problème de Diffie-Hellman calculatoire (Définition 1.2).

Définition 4.9. SDHP (*Static Diffie–Hellman problem*). Soient $\mathbb{G}$ un groupe cyclique, $s \in \mathbb{Z}$ et g un générateur de $\mathbb{G}$. Étant donnés g, g^s et un oracle $x \mapsto x^s$, après une phase d'apprentissage faisant appel à l'oracle, calculer h^s pour un élément h de $\mathbb{G}$ choisi au hasard.

Ce problème a été formalisé pour la première fois par Brown et Gallant en 2004 [BG04]. Certains protocoles n'ont pas besoin d'un groupe résistant au SDHP, mais d'autres reposent dessus, notamment lorsqu'une clé privée doit être réutilisée un grand nombre de fois. Le chiffrement d'ElGamal [ElG84] est ainsi un exemple de protocole demandant une résistance au SDHP.

Afin de résoudre le SDHP, [Jou+09] [4], [Gra10] et [JV11] montrent qu'il est possible d'adapter un algorithme de calcul d'indice et d'utiliser l'oracle pour obtenir la solution du problème sur les éléments de la base de facteurs. Granger obtient ainsi un algorithme appelant $O((q')^{1-\frac{1}{m'+1}})$ fois l'oracle et résolvant le SDHP en un temps $\tilde{O}((q')^{1-\frac{1}{m'+1}})$.

Il est important de noter que si ce problème est plus facilement attaquable que le DLP, il n'est pas aussi critique d'obtenir une bonne sécurité contre celui-ci dans tous les cas. En effet, mettre en place une politique de révocation des clés suffisamment fréquente permet de supprimer l'accès à l'oracle pour l'attaquant.

4.3 Courbes adaptées aux couplages

Nous avons supposé qu'il existe des éléments $\mathbb{F}_q$-rationnels d'ordre ℓ pour construire un couplage de Tate restreint à des groupes $\mathbb{G}_1$ et $\mathbb{G}_2$ respectant les hypothèses d'un couplage cryptographique. Cela impose que ℓ soit un facteur de $\#\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)$. De plus la contrainte de sécurité sur le DLP dans $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)[\ell]$ impose que ℓ soit un grand facteur premier (de taille au moins $2s$ bits environ où s est la sécurité visée). Cependant si le théorème de Hasse-Weil nous dit que le cardinal de $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)$ est de l'ordre de q^g (Corollaire 2.50), le plus grand facteur premier de $\#\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)$ peut être bien plus petit. Cette différence entre ℓ et q^g est ainsi mesurée par le paramètre ρ.

Définition 4.10 (ρ-valeur)**.** Soient $\mathcal{H}$ une courbe hyperelliptique de genre g définie sur $\mathbb{F}_q$ et ℓ un premier divisant l'ordre de la partie $\mathbb{F}_q$-rationnelle de la jacobienne de $\mathcal{H}$. Nous définissons la ρ-valeur de $\mathcal{H}$ pour ℓ par

$$\rho = \frac{\log q^g}{\log \ell} = g\frac{\log q}{\log \ell}.$$

4. Les auteurs attaquent dans ce cas le groupe multiplicatif d'un corps fini.

TABLE 4.3 – Estimation de valeurs de k équilibrant la sécurité de la courbe et du groupe des racines ℓ-ièmes pour différents niveaux de sécurité.

Sécurité	Petite caractéristique			Grande caractéristique		
	ℓ	q^k	$k \cdot \rho/g$	ℓ	q^k	$k \cdot \rho/g$
64 bits	$\gtrsim 2^{128}$	$\gtrsim 2^{768}$	6	$\gtrsim 2^{128}$	$\gtrsim 2^{512}$	4
80 bits	$\gtrsim 2^{160}$	$\gtrsim 2^{1600}$	10	$\gtrsim 2^{160}$	$\gtrsim 2^{1024}$	6
128 bits	$\gtrsim 2^{256}$	$\gtrsim 2^{4096}$	16	$\gtrsim 2^{256}$	$\gtrsim 2^{3072}$	12
192 bits	$\gtrsim 2^{384}$	$\gtrsim 2^{11520}$	30	$\gtrsim 2^{384}$	$\gtrsim 2^{6912}$	18
256 bits	$\gtrsim 2^{512}$	$\gtrsim 2^{22528}$	44	$\gtrsim 2^{512}$	$\gtrsim 2^{15360}$	30

Ainsi, une courbe fournissant une ρ-valeur proche de 1 donne, a priori, des clés de taille minimale. Il faut cependant tenir compte également de la sécurité dans le groupe d'arrivée.

Une première idée permettant de minimiser la taille des clés est alors d'équilibrer la résistance aux attaques sur la courbe avec celles sur le groupe des racines ℓ-ièmes de l'unité. Ceci est possible en choisissant une bonne valeur pour le degré de plongement k qui définit le rapport entre la taille de la courbe et celle de l'extension de corps fini dans laquelle vivent les racines de l'unité. Plus précisément, il s'agit d'adapter l'ordre du groupe de torsion ℓ au cardinal de l'extension du corps fini q^k ; comme ℓ est de taille $g\frac{\log q}{\rho}$ par définition de ρ, ce rapport est

$$\frac{\log q^k}{\log \ell} = k \log q \frac{\rho}{g \log q} = \frac{k\rho}{g}.$$

Ainsi, c'est la valeur de $k\rho/g$ qu'il faut ajuster en fonction des exigences de sécurité. En effet, la complexité des techniques de crible algébrique étant meilleure que celle des attaques sur une courbe, la valeur de $k\rho/g$ « optimale » en ce sens augmente avec le niveau de sécurité requis.

Nous donnons ces valeurs dans la Table 4.3. Cette table a été calculée en utilisant les valeurs théoriques de complexité fournies dans les sections précédentes. Ceci est un choix optimiste si nous les comparons aux recommandations disponibles sur le site `keylength.com` qui prend en compte l'amélioration dans le temps des algorithmes et des machines. Cependant, nous évitons d'utiliser ces valeurs pessimistes, étant donné qu'elle ne sont pas disponibles pour le calcul du logarithme discret par l'algorithme FFS. De plus, nous souhaitons pouvoir nous comparer aux autres implémentations de couplages de la littérature et préférons ainsi utiliser les mêmes estimations que le reste de la communauté. [5]

En pratique, nous ne pouvons choisir le degré de plongement librement. En effet, ce degré d'extension k est en général du même ordre de grandeur que q pour une courbe tirée au hasard [Sil86, Section XI.6, Remarque 6.3]. Ainsi, le calcul d'un tel couplage est d'ordinaire inenvisageable pour la jacobienne d'une courbe de taille cryptographique à cause de la taille de l'extension à considérer.

Cependant, il existe des familles de courbes pour lesquelles ce degré de plongement est connu et suffisamment petit. Ces courbes sont dites **adaptées aux couplages** (*pairing-friendly* en anglais). Dès lors, nous nous trouvons dans la situation où il faut obtenir un

5. Rappelons qu'il faudrait désormais prendre en compte les travaux récents signalés au Chapitre 1.

TABLE 4.4 – Bornes supérieures pour le rapport k/g des variétés supersingulières [RS02, Table 1].

Genre g	1	2	3	4	5	6
q est un carré	3	3	3	$\frac{15}{4}$	$\frac{11}{5}$	$\frac{7}{2}$
q non carré, $p > 3$	2	3	$\frac{14}{3}$	$\frac{15}{4}$	$\frac{22}{5}$	7
q non carré, $p = 2$	4	6	–	5	–	6
q non carré, $p = 3$	6	2	6	$\frac{15}{2}$	–	7

(q cardinal du corps de base, p sa caractéristique.)

degré de plongement k suffisamment « petit » pour rendre le calcul de couplage possible et suffisamment « grand » pour ne pas diminuer la sécurité apportée par la courbe.

Malheureusement, nous ne disposons que d'un bestiaire limité de courbes adaptées au calcul de couplages. Comme nous le verrons dans la partie suivante, obtenir la meilleure performance pour un couplage à une sécurité donnée dépend d'un grand nombre de paramètres algébriques et arithmétiques et non uniquement de la valeur de $k \cdot \rho/g$. C'est pourquoi il est souvent préférable de choisir une courbe possédant de bonnes propriétés même si elle ne donne pas une valeur de $k \cdot \rho/g$ « optimale ». Nous donnons dans la suite de cette section quelques exemples de courbes elliptiques et hyperelliptiques adaptées aux couplages.

4.3.1 Courbes supersingulières

Nous nous intéressons ici à une première famille de courbes, identifiée de longue date, qui permet d'atteindre des degrés de plongement petits : les courbes supersingulières. De plus, comme il existe toujours des *distortion maps* pour ces courbes, nous pourrons ainsi construire des couplages de type I avec elles.

Plus précisément, les degrés de plongement qu'il est possible d'atteindre avec ces courbes sont bornés et ont été donnés par Rubin et Silverberg [RS02]. Nous décrivons ces bornes dans la Table 4.4.

Remarque 4.11. Certaines des bornes données dans la Table 4.4, et notamment l'optimum $k/g = 7,5$ en genre 4, ne sont pas atteignables a priori pour des jacobiennes de courbes, mais seulement pour des variétés abéliennes, pour lesquelles notre définition de couplage n'est pas suffisante. Étant données les attaques décrites dans la Section 4.2.2, ce n'est pas limitant puisqu'il existe des attaques pour les courbes de genre supérieur à 3. Enfin, nous exhibons des courbes hyperelliptiques atteignant les bornes pour le genre 1 et 2 ; nous introduisons donc aucune restriction en ne considérant que les courbes hyperelliptiques.

Nous constatons que pour le genre 1 et 2, les courbes en caractéristique 2 et 3 sont les plus attractives, car fournissant les degrés de plongements les mieux adaptés aux niveaux de sécurité.

Nous listons donc ici les trois familles de courbes que nous allons considérer dans la suite du manuscrit :

— $E_2/\mathbb{F}_2 : y^2 + y = x^3 + x + b$ avec $b \in \{0, 1\}$,
— $E_3/\mathbb{F}_3 : y^2 = x^3 - x + b$ avec $b \in \{-1, 1\}$,

— $H_2/\mathbb{F}_2 : y^2 + y = x^5 + x^3 + d$ avec $d \in \{0, 1\}$.

§ 1. Courbe E_2. Les premières courbes que nous considérons sont les courbes elliptiques supersingulières définies sur $\mathbb{F}_2$:

$$E_2 : y^2 + y = x^3 + x + b,$$

avec $b \in \{0, 1\}$. Elles permettent toutes deux d'atteindre un degré de plongement optimal $k = 4$. Afin de bénéficier de groupes de taille suffisante, nous considérerons ces courbes sur un corps de base $\mathbb{F}_{2^m}$ avec m impair. Le cardinal des parties $\mathbb{F}_{2^m}$-rationnelles est connu :

$$\#E_2(\mathbb{F}_{2^m}) = 2^m + \nu 2^{\frac{m+1}{2}} + 1, \text{ avec } \nu = \begin{cases} (-1)^b & \text{si } m \equiv \pm 1 \pmod 8, \\ -(-1)^b & \text{si } m \equiv 4 \pm 1 \pmod 8. \end{cases}$$

Il s'agit des facteurs aurifeuilliens des nombres de Cunningham en base 2. C'est un fait intéressant lorsqu'il s'agira de choisir effectivement le corps de base car les factorisations de beaucoup de ces nombres sont connues et tabulées [Wag ; Ley].

§ 2. Courbe E_3. Les deux courbes que nous étudions ensuite sont les courbes elliptiques supersingulières définies sur $\mathbb{F}_3$:

$$E_3 : y^2 = x^3 - x + b,$$

avec $b \in \{-1, 1\}$. Considérées sur $\mathbb{F}_{3^m}$ avec $\operatorname{pgcd}(6, m) = 1$, elles sont de degré de plongement $k = 6$ et leur cardinal est

$$\#E_3(\mathbb{F}_{3^m}) = 3^m + \mu b 3^{\frac{m+1}{2}} + 1, \text{ où } \mu = \begin{cases} 1 & \text{si } m \equiv \pm 1 \pmod{12}, \\ -1 & \text{si } m \equiv 6 \pm 1 \pmod{12}. \end{cases}$$

Les cardinaux de ces courbes sont également des facteurs aurifeuilliens.

§ 3. Courbe H_2. Enfin, nous avons utilisé les courbes hyperelliptiques de genre 2 définies sur $\mathbb{F}_2$:

$$H_2 : y^2 + y = x^5 + x^3 + d,$$

avec $d \in \{0, 1\}$. Sur $\mathbb{F}_{2^m}$ avec m copremier avec 6, le degré de plongement de ces courbes est $k = 12$ et le cardinal de la partie $\mathbb{F}_{2^m}$-rationnelle de la jacobienne est :

$$\#\mathrm{Jac}_{H_2}(\mathbb{F}_{2^m}) = 2^{2m} + \delta 2^{\frac{3m+1}{2}} + 2^m + \delta 2^{\frac{m+1}{2}} + 1,$$
$$\text{où } \delta = \begin{cases} (-1)^d & \text{si } m \equiv \pm 1, \pm 7 \pmod{24}, \\ -(-1)^d & \text{si } m \equiv \pm 5, \pm 11 \pmod{24}. \end{cases}$$

Nous donnons dans la Table 4.5 des estimations sur la taille des corps de base pour ces trois familles de courbes en fonction du niveau de sécurité requis.

TABLE 4.5 – Choix du corps de base pour une courbe supersingulière à différents niveaux de sécurité.

Sécurité	E_2 sur $\mathbb{F}_{2^m}$		E_3 sur $\mathbb{F}_{3^m}$		H_2 sur $\mathbb{F}_{2^m}$	
	m	Attaque	m	Attaque	m	Attaque
64 bits	$\gtrsim 196$	FFS	$\gtrsim 88$	ρ de Pollard	$\gtrsim 70$	ρ de Pollard
80 bits	$\gtrsim 330$	FFS	$\gtrsim 140$	FFS	$\gtrsim 110$	FFS
128 bits	$\gtrsim 1024$	FFS	$\gtrsim 420$	FFS	$\gtrsim 350$	FFS
192 bits	$\gtrsim 2800$	FFS	$\gtrsim 1200$	FFS	$\gtrsim 950$	FFS
256 bits	$\gtrsim 5700$	FFS	$\gtrsim 2400$	FFS	$\gtrsim 1900$	FFS

Nous avons pris en compte ici l'accélération fournie à la méthode ρ de Pollard par le groupe d'automorphismes de la courbe concernée et des itérés du Frobenius.

4.3.2 Courbes ordinaires

Nous avons vu qu'avec les courbes supersingulières, le degré de plongement est limité. Pour de plus hauts niveaux de sécurité, il peut être souhaitable d'obtenir des courbes avec k plus grand. Il faut pour cela considérer des courbes non-supersingulières.

Comme nous ne pouvons espérer un degré de plongement suffisamment petit en cherchant des courbes au hasard, nous avons besoin de constructions *ad-hoc*, qui sont issues *in fine* de la théorie de la multiplication complexe. Nous pouvons citer, parmi ces constructions, les méthodes de Cocks-Pinch [CP01] et de Brezing-Weng [BW05]. Nous renvoyons à la taxonomie de Freeman, Scott et Teske dans [FST10] pour un survol de ces constructions. Cependant, ces méthodes sont intrinsèquement limitées à la grande caractéristique ; en effet, celles-ci demandent de tirer aléatoirement le cardinal du corps sur lequel la courbe sera définie et la probabilité d'obtenir une puissance d'un petit premier est beaucoup plus faible que celle d'obtenir un premier.

Nous donnons toutefois l'exemple des courbes Barreto-Naehrig [BN06] qui, grâce à un degré de plongement de 12, sont bien adaptées pour le niveau de sécurité de 128 bits. De plus, ces courbes sont très utilisées dans les implémentations records [Ara+11 ; Yao+13] et nous les utiliserons pour comparer nos implémentations au cas ordinaire.

Du fait de leur construction, les familles de courbes ordinaires *pairing-friendly* sont souvent paramétrées par un entier z. Dans le cas des courbes Barreto-Naehrig, la taille $p(z)$ du corps de définition ainsi que le cardinal $n(z)$ de la courbe sont donnés par des polynômes :

$$n(z) = 36z^4 + 36z^3 + 18z^2 + 6z + 1,$$
$$p(z) = 36z^4 + 36z^3 + 24z^2 + 6z + 1,$$

ce qui implique que la trace du Frobenius est

$$t(z) = 6z^2 + 1.$$

Les courbes Barreto-Naehrig ont alors pour équation

$$E_{\text{BN}} : y^2 = x^3 + b.$$

TABLE 4.6 – Choix du corps de base pour une courbe BN à différents niveaux de sécurité.

Sécurité	p	Meilleure attaque
64 bits	$\gtrsim 128$	ρ de Pollard
80 bits	$\gtrsim 160$	ρ de Pollard
128 bits	$\gtrsim 256$	ρ de Pollard et NFS
192 bits	$\gtrsim 512$	NFS
256 bits	$\gtrsim 1024$	NFS

Elles ont un groupe d'automorphismes de cardinal 6. Comme il est possible de l'espérer, cette courbe possède ainsi une tordue de degré 6. Nous pouvons alors définir le couplage de Tate sur un groupe $\mathbb{F}_p$-rationnel et un groupe $\mathbb{F}_{p^2}$-rationnel (la construction se trouve en Section 3 de [BN06]). Dans la Table 4.6, nous donnons différentes estimations de taille de corps pour différents niveaux de sécurité ainsi que la meilleure attaque dans chacun de ces cas.

Nous mentionnons également d'autres familles de courbes qui ont été utilisées dans des implémentations matérielles ou logicielles. Elles permettent d'atteindre différents degrés de plongement au prix de ρ-valeurs plus ou moins élevées.

— (MIYAJI, NAKABAYASHI et TAKANO, 2001) [MNT01] les courbes E_{MNT_k} permettent d'atteindre des degrés de plongement k égaux à 3, 4, ou 6 ;
— (BARRETO, LYNN et SCOTT, 2003) [BLS03] ;
— (KACHISA, SCHAEFER et SCOTT, 2008) [KSS08].

Notons également que des travaux récents proposent des constructions pour le genre 2 [GV11]. Elles sont valables pour tout degré de plongement ; cependant, pour $5 \leq k \leq 35$, la ρ-valeur des courbes obtenues est élevée : $2,25 \leq \rho \leq 4$.

Deuxième partie

Calcul de couplage

Chapitre 5

Algorithme de Miller

Nous présentons dans ce chapitre les techniques de base permettant de calculer un couplage. Comme nous l'avons vu au Chapitre 3, les couplages peuvent être définis grâce à des fonctions de Miller. En 1986, Miller a décrit un algorithme permettant de calculer ces fonctions [Mil86a ; Milo4]. Bien qu'il ait détaillé ces techniques pour le couplage de Weil, celles-ci s'adaptent aisément au couplage de Tate que nous utilisons principalement dans cette thèse.

5.1 Algorithme général

Nous rappelons que nous cherchons ici à calculer le couplage de Tate réduit

$$\begin{array}{rccccc} e_\ell : & \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell] & \times & \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell] & \longrightarrow & \mu_\ell \\ & (\overline{D_1} & , & \overline{D_2}) & \longmapsto & f_{\ell,D_1}(D_2)^{\frac{q^k-1}{\ell}}, \end{array}$$

où D_1 et D_2 sont de supports disjoints et $f_{n,D}$ est la fonction de Miller donnée par

$$\mathrm{div}\, f_{n,D} = nD - \rho(nD).$$

Les fonctions de Miller sont donc définies à une constante multiplicative près et nous verrons plus tard (Section 5.3.1) comment la fixer.

Le calcul du couplage revient ainsi à construire la fonction f_{ℓ,D_1} et à l'évaluer en D_2. Remarquons que le choix des représentants D_1 et D_2 dans leur classe d'équivalence est libre tant que leur support est disjoint : les Propositions 3.6 et 3.7 montrent que la valeur du couplage est indépendante du choix de D_1 et D_2. Cependant, nous imposons que le diviseur D_1 soit réduit pour simplifier le calcul de la fonction de Miller. Le diviseur D_2 ne peut ainsi être réduit car, dans ce cas, le point P_∞ serait à la fois dans le support de D_1 et dans celui de D_2.

5.1.1 Récurrence sur les fonctions de Miller

Soit D un diviseur réduit. Nous remarquons en premier lieu que les fonctions de Miller de paramètre 0 ou 1 sont des fonctions constantes. En effet,

$$\mathrm{div}\, f_{0,D} = 0 \cdot D - \rho(0 \cdot D) = 0 \text{ et } \mathrm{div}\, f_{1,D} = D - \rho(D) = D - D = 0.$$

Nous cherchons alors une identité permettant le calcul par récurrence des fonctions de Miller :

$$\begin{aligned}\operatorname{div} f_{n+n',D} &= (n+n')D - \rho((n+n')D) \\ &= \underbrace{nD - \rho(nD)}_{\operatorname{div} f_{n,D}} + \underbrace{n'D - \rho(n'D)}_{\operatorname{div} f_{n',D}} + \underbrace{\rho(nD) + \rho(n'D) - \rho((n+n')D)}_{\operatorname{div} g_{\rho(nD),\rho(n'D)}},\end{aligned}$$

où $g_{D,D'}$ est une fonction qui intervient dans le calcul de la somme réduite de D et D'. Son diviseur est

$$\operatorname{div} g_{D,D'} = D + D' - \rho(D + D').$$

Celle-ci apparaît explicitement dans l'algorithme d'addition-réduction de Cantor que nous développons dans la Section 5.2. Nous avons donc, à une constante multiplicative près, l'égalité suivante :

$$f_{n+n',D} = f_{n,D} \cdot f_{n',D} \cdot g_{\rho(nD),\rho(n'D)}. \tag{5.1}$$

Ainsi, le calcul de $f_{n,D}$ est lié au calcul de la multiplication scalaire $[n]D = \rho(nD)$. Il suffit alors de choisir n'importe quelle chaîne d'addition pour n pour obtenir un moyen de calculer la fonction de Miller voulue. Celle-ci s'exprimera comme le produit de fonctions de la forme $g_{[k_1]D,[k_2]D}$ où k_1 et k_2 sont des éléments intermédiaires de la chaîne d'addition considérée.

5.1.2 Algorithme de Miller

Un choix classique de chaîne d'addition est de suivre un algorithme de doublement-addition commençant par les poids forts de n. Il suffit de le modifier pour calculer $f_{n,D}$ en même temps que $[n]D$. Nous présentons cette méthode dans l'Algorithme 5.1, où nous intégrons l'évaluation de la fonction de Miller dans le corps de la boucle pour limiter la taille des variables manipulées.

Algorithme 5.1 Algorithme de Miller.

Entrées : Un entier n donné par sa représentation binaire $\overline{n_{t-1} \dots n_0}$ et $D, D' \in \mathrm{Div}^0_{\mathbb{F}_{q^k}}(\mathcal{H})$ avec D réduit et D' de support disjoint avec ceux de toutes les fonctions $g_{V,V}$ et $g_{V,D}$ intervenant dans l'algorithme.

Sortie : $f_{n,D}(D')$

1. $f \leftarrow 1$; $V \leftarrow 0$
2. **for** $i \leftarrow t-1, t-2, \dots, 0$ **do**
3. $\quad f \leftarrow f^2 \cdot g_{V,V}(D')$; $V \leftarrow \rho(2V)$
4. $\quad$ **if** $n_i = 1$ **then**
5. $\quad\quad f \leftarrow f \cdot g_{V,D}(D')$; $V \leftarrow \rho(V + D)$
6. $\quad$ **end if**
7. **end for**
8. **return** f

Le calcul des fonctions $g_{V,V}$ et $g_{V,D}$ des lignes 3 et 5 est issu du calcul du doublement et de l'addition de diviseurs explicité dans la section suivante.

Il est aisé de vérifier la correction de l'algorithme en montrant qu'après l'itération i, f contient $f_{\overline{n_{t-1}\dots n_i},D}(D')$ et V le diviseur réduit $[\overline{n_{t-1}\dots n_i}]D$ par induction en utilisant la relation (5.1).

Remarquant que, pour certaines courbes, les formules de triplement sont moins complexes que celles pour le doublement, Duursma et Lee proposèrent d'utiliser une variante de cet algorithme en triplement-addition [DL03]. Nous détaillerons plus précisément ceci à la Section 7.2.

5.2 Réduction de Cantor

Comme nous l'avons vu dans la section précédente, le calcul de fonction de Miller demande de savoir effectuer une addition dans la jacobienne d'une courbe hyperelliptique. Cette addition se décompose en deux parties : l'addition des représentants D_1 et D_2 en tant que diviseurs puis la réduction de la somme obtenue. Cette méthode a été décrite par Cantor en 1987 [Can87]. Nous chercherons ici à exhiber le calcul de la fonction g_{D_1,D_2} correspondante.

Afin de traiter la réduction de Cantor dans toute sa généralité, nous supposons ici que la courbe $\mathcal{H} : y^2 + h(x)y = f(x)$ est définie sur K. Nous décrivons ici l'addition dans $\mathrm{Jac}_{\mathcal{H}}(L)$ où L est une extension de K.

Soient D_1 et D_2 deux diviseurs L-rationnels réduits et $[u_1, v_1]$ et $[u_2, v_2]$ leurs représentations de Mumford respectives. Comme D_1 et D_2 sont réduits, ils s'écrivent chacun sous la forme

$$D_i = \sum_{j=1}^{r_i}(P_{i,j}) - r_i(P_\infty) \text{ avec } r_i \leq g \text{ et } P_{i,j} \in \mathcal{H}(\overline{K}).$$

Nous avons également deux conditions supplémentaires qui assurent l'unicité de la réduction : $P_{i,j} \neq P_\infty$ et $P_{i,j} \neq \iota(P_{i,j'})$ pour $j \neq j'$. Nous rappelons que la représentation de Mumford permet d'encoder la partie effective d'un diviseur : les abscisses des $P_{i,j}$ sont les zéros du polynôme $u_i \in L[x]$ et les ordonnées correspondantes sont obtenues grâce au polynôme v_i (*cf.* Section 2.3.2).

Algorithme 5.2 Algorithme de réduction-addition de Cantor [CFA06, Algorithme 14.7].

Entrées : $D_1 = [u_1, v_1], D_2 = [u_2, v_2]$ deux diviseurs réduits et L-rationnels sur une courbe hyperelliptique $\mathcal{H} : y^2 + h(x)y = f(x)$

Sortie : $D = \rho(D_1 + D_2)$ et g_{D_1,D_2}

1. $d_1 \leftarrow \mathrm{pgcd}\,(u_1, u_2)$; garder les coefficients de Bézout $d_1 = e_1u_1 + e_2u_2$
2. $d \leftarrow \mathrm{pgcd}\,(d_1, v_1 + v_2 + h)$; garder les coefficients de Bézout $d = c_1d_1 + c_2(v_1 + v_2 + h)$
3. $G \leftarrow d$
4. $s_1 \leftarrow c_1e_1$; $s_2 \leftarrow c_1e_2$; $s_3 \leftarrow c_2$
5. $u \leftarrow \frac{u_1u_2}{d^2}$; $v \leftarrow \frac{s_1u_1v_2+s_2u_2v_1+s_3(v_1v_2+f)}{d} \mod u$
6. **while** $\deg u > g$ **do**
7. $\quad \tilde{u} \leftarrow \frac{f-vh-v^2}{u}$; $\tilde{v} \leftarrow (-v - h) \mod \tilde{u}$
8. $\quad G \leftarrow G \cdot \frac{y-v}{\tilde{u}}$
9. $\quad u \leftarrow \tilde{u}$; $v \leftarrow v'$
10. **end while**
11. Rendre u unitaire
12. **return** $[u, v], G$

La première étape consiste à former le diviseur non-réduit $D_1 + D_2$ en prenant pour u le produit $u_1 u_2$ et en construisant v par interpolation de Lagrange (Algorithme 5.2, ligne 5). Cependant, ce diviseur peut contenir des points qui sont involués l'un de l'autre. Il ne pourra donc pas être directement représenté sous la forme $[u, v]$. Nous commençons donc par les éliminer deux à deux en utilisant les fonctions rationnelles correspondantes (lignes 1 à 5). Pour ce faire, nous construisons tout d'abord la fonction d_1 dont les zéros sont les abscisses communes entre les points des supports de D_1 et D_2 (ligne 1), puis nous ne retenons dans d que celles qui correspondent à des paires de points involués l'un de l'autre (ligne 2). Le diviseur de d est ainsi

$$\operatorname{div} d = \sum_{\substack{P \in \operatorname{supp}(\epsilon(D_1)) \\ \iota(P) \in \operatorname{supp}(\epsilon(D_2))}} \min\left(\operatorname{ord}_P(D_1), \operatorname{ord}_{\iota(P)}(D_2)\right)((P) + (\iota(P)) - 2(P_\infty)).$$

Certains auteurs disent alors que le diviseur obtenu est semi-réduit. La contribution à la fonction g_{D_1,D_2} est directement donnée par la fonction d, et prise en compte à la ligne 3.

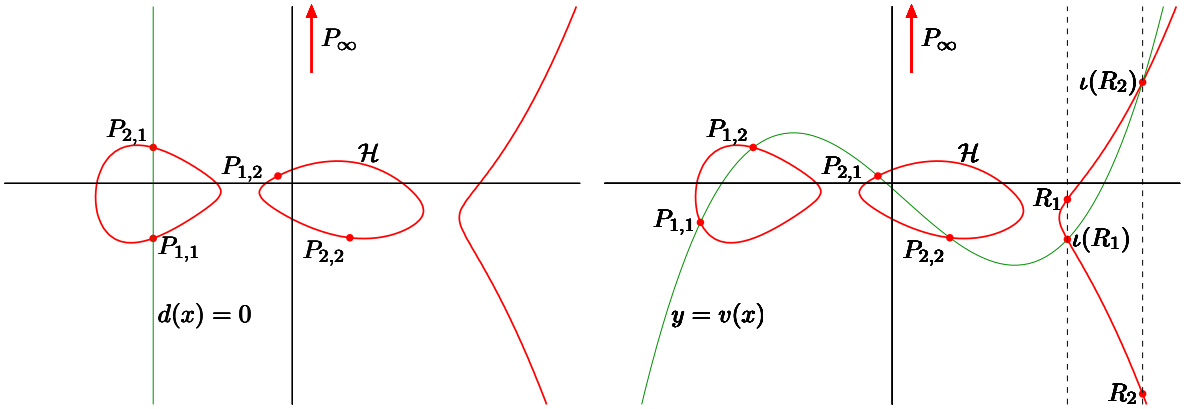
L'étape suivante, appelée réduction, consiste à appliquer autant de fois que nécessaire une construction qui diminue de une ou deux unités la taille du support effectif de $D = [u, v]$ jusqu'à ce qu'il soit complètement réduit, c'est-à-dire qu'il contienne au plus g points (ligne 10). En effet, à la ligne 5, $D = [u, v]$ peut contenir jusqu'à $2g$ points.

La boucle (ligne 6 à 10) termine quand il reste au plus g points dans le diviseur courant (c'est-à-dire $\deg u \leq g$). Le corps de cette boucle consiste à construire une fonction rationnelle qui permettra d'échanger des points de D contre d'autres moins nombreux. Cette fonction $\frac{y-v}{\tilde{u}}$ est accumulée multiplicativement (ligne 8) afin d'obtenir g_{D_1,D_2} à la fin de l'algorithme. Le nouveau diviseur ainsi construit $[\tilde{u}, \tilde{v}]$ contient moins de points puisque

$$\begin{aligned}
\deg \tilde{u} &\leq \max\{\deg f, \deg vh, \deg v^2\} - \deg u \\
&\leq \max\{2g+1, g + \deg u - 1, 2\deg u - 2\} - \deg u \\
&\leq \max\{2g + 1 - \deg u, g - 1, \deg u - 2\} \\
&\leq \max\{g, g-1, \deg u - 2\} \\
&< \deg u.
\end{aligned}$$

Exemple 5.1 (Courbes elliptiques). Dans le cas des courbes elliptiques, l'algorithme de Cantor peut être réécrit de façon plus simple. Nous avons déjà vu qu'en genre 1 la courbe et la jacobienne sont isomorphes par $P \mapsto (P) - (P_\infty)$. Trois cas se présentent alors : soit nous cherchons à additionner deux points opposés et nous obtenons l'élément neutre P_∞ correspondant au diviseur nul sur la jacobienne ; soit nous doublons un point et nous pouvons utiliser directement une formule de doublement ; soit nous avons une addition. Il s'agit de la loi classique de la corde et de la tangente.

Exemple 5.2 (Courbes hyperelliptique de genre 2). La réduction de Cantor pour les courbes hyperelliptiques de genre 2 est également plus simple que pour le genre supérieur. En premier lieu, les diviseurs à sommer peuvent être soit dégénérés (leurs supports contiennent strictement moins de deux points) soit génériques. Si les deux diviseurs sont dégénérés alors il n'y a aucune réduction à faire. Sinon il suffira d'une étape de réduction qui n'aura lieu que si aucun point des supports n'est l'involué d'un autre (c'est-à-dire quand la semi-réduction n'est pas suffisante). La somme de deux diviseurs génériques sur une courbe hyperelliptique de genre 2 est illustrée dans la Figure 5.3.



(a) Semi-réduction (b) Réduction

FIGURE 5.3 – Loi de groupe pour la jacobienne d'une courbe hyperelliptique.

5.3 Premières optimisations pour le calcul du couplage de Tate

Dans cette section, nous étudions deux optimisations du calcul du couplage de Tate de portée générale avant de construire d'autres couplages dans le chapitre suivant afin de réduire encore le coût de calcul d'un couplage cryptographique.

5.3.1 Évaluation en des diviseurs effectifs

Une première idée permettant de réduire le coût de calcul d'un couplage est d'évaluer la fonction de Miller non pas en un diviseur D' représentant $\overline{D'}$ mais en la partie effective du diviseur réduit correspondant : $\epsilon(\rho(D'))$. Pour ce faire, nous avons besoin de **normaliser** la fonction de Miller au point à l'infini, et donc d'étudier son comportement au voisinage de ce point. Cela nous amène à la notion de coefficient dominant.

Définition 5.3. Soient $\mathcal{H}$ une courbe définie sur K, P un point sur $\mathcal{H}$, u_P une uniformisante en P et f une fonction rationnelle. Le **coefficient dominant** (*leading coefficient*) de f au point P relativement à l'uniformisante u_P est défini par

$$\mathrm{lc}_P(f) = \left(u_P^{-\mathrm{ord}_P(f)} f\right)(P).$$

Commençons par montrer que le coefficient dominant ne dépend pas du choix de l'uniformisante dans notre cas en utilisant le lemme suivant.

Lemme 5.4. *Soient $u_{P_\infty}, u'_{P_\infty}$ deux uniformisantes $\mathbb{F}_{q^k}$-rationnelles en P_∞ et $\overline{D} \in \mathrm{Jac}_{\mathcal{H}}[\ell]$. Nous notons lc_{P_∞} et lc'_{P_∞} les coefficients dominants correspondants. Nous avons alors :*

$$\frac{\mathrm{lc}_{P_\infty}(f_{\ell,D})}{\mathrm{lc}'_{P_\infty}(f_{\ell,D})} \in (\mathbb{F}^*_{q^k})^\ell.$$

Preuve. Nous constatons tout d'abord que

$$\operatorname{div} f_{\ell,D} = \ell D - \rho(\ell D) = \ell D,$$

et donc que

$$\operatorname{ord}_{P_\infty}(f_{\ell,D}) = -\ell r \text{ où } r = \deg \epsilon(D).$$

$$\begin{aligned}
\frac{\operatorname{lc}_{P_\infty}(f_{\ell,D})}{\operatorname{lc}'_{P_\infty}(f_{\ell,D})} &= \frac{\left({u_{P_\infty}}^{-\operatorname{ord}_{P_\infty}(f_{\ell,D})} f\right)(P_\infty)}{\left({u'_{P_\infty}}^{-\operatorname{ord}_{P_\infty}(f_{\ell,D})} f\right)(P_\infty)} \\
&= \left(\frac{u_{P_\infty}}{u'_{P_\infty}}(P_\infty)\right)^{-\operatorname{ord}_{P_\infty}(f_{\ell,D})} \\
&= \left(\frac{u_{P_\infty}}{u'_{P_\infty}}(P_\infty)\right)^{\ell r} \\
&\in (\mathbb{F}_{q^k}^*)^\ell \text{ car } P_\infty \in \mathcal{H}(\mathbb{F}_q) \text{ et } u_{P_\infty}/u'_{P_\infty} \in \mathbb{F}_{q^k}(\mathcal{H})^*.
\end{aligned}$$

□

Nous montrons maintenant le résultat attendu : si la fonction construite par l'algorithme de Miller est normalisée — c'est-à-dire que son coefficient dominant au point P_∞ est une puissance ℓ-ième —, alors il suffit de l'évaluer en un diviseur effectif. C'est la généralisation hyperelliptique d'une idée d'abord apparue pour le cas elliptique dans [Bar+02, Théorème 1]. Plus précisément :

Théorème 5.5. *Soient $\overline{D}$, $\overline{D'}$ deux éléments de $\operatorname{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell]$ avec D' un diviseur réduit. Si $\operatorname{lc}_{P_\infty}(f_{\ell,D}) \in (\mathbb{F}_{q^k}^*)^\ell$, alors nous avons que*

$$\langle \overline{D}, \overline{D'} \rangle_\ell \equiv f_{\ell,D}(\epsilon(D')) \pmod{(\mathbb{F}_{q^k}^*)^\ell}$$

tant que $\operatorname{supp}(D) \cap \operatorname{supp}(\epsilon(D')) = \emptyset$.

Preuve. Soit u_{P_∞} une uniformisante en P_∞ dont le support est disjoint de celui de $\epsilon(D')$. Nous notons $r = \deg \epsilon(D)$ et constatons alors que la fonction $u_{P_\infty}^{r\ell} f_{\ell,D}$ ne possède pas P_∞ dans son support. De plus, comme $\operatorname{supp}(D) \cap \operatorname{supp}(\epsilon(D')) = \emptyset$ par hypothèse, le support de $\epsilon(D')$ est disjoint de celui de $u_{P_\infty}^{rl} f_{\ell,D}$.

Soit $D'' \in \overline{D'}$ tel que D'' ait un support disjoint de ceux de D et $\operatorname{div} u_{P_\infty}$. Par conséquent, D'' est également de support disjoint de celui de $u_{P_\infty}^{rl} f_{\ell,D}$.

Nous posons $r' = \deg \epsilon(D')$ et nous avons ainsi $D' = \epsilon(D') - r'(P_\infty)$.

Nous fixons maintenant une fonction $\mathbb{F}_{q^k}$-rationnelle h telle que $\operatorname{div} h = D'' - D' = D'' - \epsilon(D') + r'(P_\infty)$. Le support de h est donc contenu dans $\operatorname{supp}(D'') \cup \operatorname{supp}(\epsilon(D')) \cup \{P_\infty\}$. Il est ainsi disjoint de celui de $u_{P_\infty}^{rl} f_{\ell,D}$.

Nous exprimons alors le couplage de Tate modulo les puissances ℓ-ièmes.

$$\begin{aligned}
\langle \overline{D}, \overline{D'} \rangle_\ell &= f_{\ell,D}(D'') \\
&\equiv \left(u_{P_\infty}^{r\ell} f_{\ell,D}\right)(D'') \pmod{(\mathbb{F}_{q^k}^*)^l} \\
&= \left(u_{P_\infty}^{r\ell} f_{\ell,D}\right)(\epsilon(D') - r'(P_\infty) + \operatorname{div} h) \\
&= u_{P_\infty}(\epsilon(D'))^{r\ell} \cdot f_{\ell,D}(\epsilon(D')) \cdot \frac{\left(u_{P_\infty}^{r\ell} f_{\ell,D}\right)(\operatorname{div} h)}{\operatorname{lc}_{P_\infty}(f_{\ell,D})^{r'}} \\
&\equiv f_{\ell,D}(\epsilon(D')) \cdot h\left(\operatorname{div}\left(u_{P_\infty}^{r\ell} f_{\ell,D}\right)\right) \pmod{(\mathbb{F}_{q^k}^*)^l} \\
&= f_{\ell,D}(\epsilon(D')) \cdot h(r\ell \operatorname{div} u_{P_\infty} + \ell D) \\
&= f_{\ell,D}(\epsilon(D')) \cdot h(r \operatorname{div} u_{P_\infty} + D)^\ell \\
&\equiv f_{\ell,D}(\epsilon(D')) \pmod{(\mathbb{F}_{q^k}^*)^l}.
\end{aligned}$$

Remarquons que ces calculs sont rendus possibles par les différentes disjonctions de support que nous avons montrées plus haut. □

Afin de nous placer dans les hypothèses du Théorème 5.5, nous modifions l'Algorithme 5.2 de réduction de Cantor et normalisons les fonctions $g_{D,D'}$ obtenues. Le lecteur pourra trouver dans [Gra+07, Annexe A.1] le détail de cette méthode. Le plus simple est alors de normaliser de façon à ce que $\operatorname{lc}_{P_\infty}(f_{\ell,D}) = 1$. Nous utilisons l'uniformisante en P_∞ suivante

$$u_{P_\infty}(x, y) = x^g / y.$$

Soit $p \in \overline{K}[x]$. Avec cette uniformisante, nous avons

$$\operatorname{lc}_{P_\infty}(p(x)) = \operatorname{lc}(p(x)) \text{ et}$$

$$\operatorname{lc}_{P_\infty}(y - p(x)) = \begin{cases} 1 & \text{si } \deg(p(x)) \leq g, \\ -\operatorname{lc}(p(x)) & \text{si } \deg(p(x)) > g, \end{cases}$$

où $\operatorname{lc}(p(x))$ désigne le coefficient dominant du polynôme p. Dans l'algorithme de Cantor, il suffira alors de s'assurer que la fonction pgcd (ligne 2, Algorithme 5.2) renvoie bien un polynôme unitaire et nous aurons alors $\operatorname{lc}_{P_\infty}(d) = 1$ (ligne 3). Pour les autres contributions à G, il faudra rendre u' unitaire (ligne 7) et diviser G par $\operatorname{lc}_{P_\infty}(y - v)$ à la ligne 8.

Remarque 5.6. La dernière condition du Théorème 5.5 ($\operatorname{supp}(D) \cap \operatorname{supp}(\epsilon(D')) = \emptyset$) est trivialement assurée dans le cas elliptique lorsque nous imposerons qu'un des éléments de la jacobienne en entrée du couplage soit $\mathbb{F}_q$-rationnel. En effet, comme l'autre argument du couplage sera alors nécessairement non-$\mathbb{F}_q$-rationnel (cf. Section 4.1.1), chaque support effectif ne sera constitué que d'un seul point, l'un sera de coordonnées $\mathbb{F}_q$-rationnelles et l'autre pas.

5.3.2 Élimination des verticales

Dans cette section, nous allons nous atteler à éliminer une partie du calcul de la fonction g_{D_1,D_2} correspondant à la réduction dans l'algorithme de Cantor : en se restreignant aux degrés de plongement pairs et en prenant les arguments du couplage dans les sous-espaces

propres de valeur propre 1 et $q \bmod \ell$, il n'est en effet pas nécessaire de tenir compte de la contribution des fonctions rationnelles correspondant à des produits de droites verticales [Bar+02]. Pour voir cela, nous commençons par montrer les lemmes suivants.

Lemme 5.7. *Soient $\mathcal{H} : y^2 + h(x)y = f(x)$ une courbe hyperelliptique définie sur $\mathbb{F}_q$, ℓ un premier divisant $\#\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)$, k le degré de plongement correspondant, et $\overline{D} \in \mathrm{Jac}_{\mathcal{H}}[\ell]$ tel que $\pi_q(\overline{D}) = [q]\overline{D}$. Notons $D = [u, v] \in \overline{D}$ le diviseur réduit correspondant ainsi que sa représentation de Mumford. Si k est pair, alors $\pi_q^{k/2}(D) = \iota(D)$ et u est défini sur $\mathbb{F}_{q^{k/2}}$.*

Preuve. Comme k est le degré de plongement de ℓ dans $\mathbb{F}_q$, en le supposant pair, nous avons

$$\ell \mid q^k - 1 \text{ et } \ell \nmid q^{k/2} - 1.$$

Or, comme $q^k - 1 = (q^{k/2} - 1)(q^{k/2} + 1)$ et ℓ est premier, nous avons $\ell \mid q^{k/2} + 1$ et, par conséquent, $q^{k/2} = -1 \pmod{\ell}$.

L'élément $\overline{D}$ étant de ℓ-torsion, nous établissons alors l'égalité suivante :

$$\pi_q^{k/2}(D) = [q^{k/2}]D = [-1]D = \iota(D).$$

En représentation de Mumford, comme $D = [u, v]$ et $\iota(D) = [u, (-v - h) \bmod u]$, nous avons alors bien $\pi_q^{k/2}(u) = u$. □

Lemme 5.8. *Nous supposons ici que la courbe $\mathcal{H}$ est de genre au plus 2. Dans l'Algorithme 5.2, si les diviseurs D_1 et D_2 vérifient tous deux $\pi_q^{k/2}(D_i) = \iota(D_i)$, alors les polynômes d (ligne 2) et $\tilde{u}$ (ligne 7) sont tous deux définis sur $\mathbb{F}_{q^{k/2}}$.*

Preuve. En notant $D = \rho(D_1 + D_2)$, commençons par remarquer que, par linéarité de l'endomorphisme de Frobenius, nous avons

$$\pi_q^{k/2}(D) = \rho(\pi_q^{k/2}(D_1) + \pi_q^{k/2}(D_2)) = \rho(\iota(D_1) + \iota(D_2)) = \iota(\rho(D_1 + D_2)) = \iota(D).$$

Notons alors $[u_1, v_1]$ et $[u_2, v_2]$ les représentations de Mumford respectives de D_1 et D_2, et considérons d, le PGCD de u_1, u_2 et $v_1 + v_2 + h$ calculé à la ligne 2 de l'Algorithme 5.2. Remarquons tout d'abord que, comme u_1 et u_2 sont unitaires, alors d l'est aussi.

L'hypothèse $g \leq 2$ nous permet ensuite de distinguer deux cas dans l'algorithme de Cantor, selon le nombre d'itérations de la boucle de réduction (lignes 6 à 10). En effet, comme $\deg(u_1), \deg(u_2) \leq g$, alors $\deg(u_1 u_2/d^2) \leq 2g \leq 4$, et au plus une seule itération de cette boucle est nécessaire pour obtenir le diviseur réduit $D = \rho(D_1 + D_2)$.

— Si aucune itération n'est nécessaire, alors $D = [u, v]$ pour les u et v calculés ligne 5. Comme nous savons que $\pi_q^{k/2}(D) = \iota(D)$, nous avons que $u = u_1 u_2/d^2$ est défini sur $\mathbb{F}_{q^{k/2}}$. Or, par hypothèse, nous avons aussi que u_1 et u_2 sont $\mathbb{F}_{q^{k/2}}$-rationnelles, ce qui nous donne de même que d^2 est défini sur $\mathbb{F}_{q^{k/2}}$. Enfin, d étant unitaire, il est par conséquent lui aussi $\mathbb{F}_{q^{k/2}}$-rationnel.

— Si une itération de réduction est nécessaire, alors nous pouvons constater que $g < \deg(u_1 u_2/d^2) \leq 2g - 2\deg(d)$, ce qui implique que $\deg(d) < g/2$. Puisque $g \leq 2$, nous avons alors que $\deg(d) = 0$ et donc $d = 1$. Le résultat D est construit comme $[\tilde{u}, \tilde{v}]$, avec les $\tilde{u}$ et $\tilde{v}$ calculés ligne 7. Comme $\pi_q^{k/2}(D) = \iota(D)$, nous avons alors bien que $\pi_q^{k/2}(\tilde{u}) = \tilde{u}$ et donc que $\tilde{u}$ est défini sur $\mathbb{F}_{q^{k/2}}$. □

Nous sommes alors en mesure d'énoncer le résultat suivant.

Proposition 5.9. *Soient $\mathbb{G}$ et $\mathbb{G}'$ deux sous-groupes cycliques de $\mathrm{Jac}_{\mathcal{H}}[\ell]$, chacun inclus dans un sous-espace propre de valeur propre 1 ou $q \bmod \ell$ du Frobenius π_q. Soient aussi $\overline{D_1}, \overline{D_2} \in \mathbb{G}$ et $\overline{D'} \in \mathbb{G}'$, ainsi que D_1, D_2 et D' les trois diviseurs réduits correspondants. Supposons enfin que les supports de D_1 et D_2 sont disjoints de celui de $\epsilon(D')$. Alors, dans le calcul de $g_{D_1,D_2}(\epsilon(D'))$ par l'algorithme de Cantor (Algorithme 5.2), la contribution des fonctions d (ligne 2) et $\tilde{u}$ (ligne 7) évaluées en $\epsilon(D')$ est définie sur $\mathbb{F}_{q^{k/2}}$.*

Preuve. Si $D' = [u', v']$ est de valeur propre 1 pour π_q, alors $D' \in \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)$ et donc u' est défini sur $\mathbb{F}_q$. Dans le cas contraire, D' est de valeur propre $q \bmod \ell$ et, par le Lemme 5.7, u' est $\mathbb{F}_{q^{k/2}}$-rationnel.

De même, si D_1 et D_2 sont de valeur propre 1, alors D_1, D_2 sont $_q$-rationnels, ainsi que les fonctions d et $\tilde{u}$ dans l'algorithme de Cantor. Si par contre D_1 et D_2 sont de valeur propre q, alors, par le Lemme 5.7, $\pi_q^{k/2}(D_1) = \iota(D_1)$ et $\pi_q^{k/2}(D_2) = \iota(D_2)$. En appliquant le Lemme 5.8, nous assurons enfin que les fonctions d et $\tilde{u}$ sont bien définies sur $\mathbb{F}_{q^{k/2}}$.

Ainsi, quelles que soient les valeurs propres choisies pour D_1, D_2 et D', les fonctions d, $\tilde{u}$ et u' sont toujours définies sur $\mathbb{F}_{q^{k/2}}$.

Pour conclure, il suffit de remarquer que d et $\tilde{u}$ sont des polynômes en x uniquement : ces deux fonctions correspondent à des produits de droites verticales. Leur évaluation en $\epsilon(D')$ est donc donnée par

$$d(\epsilon(D')) = \mathrm{Res}(d, u') \text{ et } \tilde{u}(\epsilon(D')) = \mathrm{Res}(\tilde{u}, u'),$$

qui sont tous deux des éléments de $\mathbb{F}_{q^{k/2}}$. □

Lorsque ses hypothèses sont réunies (degré de plongement pair et arguments du couplage restreints aux sous-espaces propres de valeur propre 1 et $q \bmod \ell$ du Frobenius), cette proposition nous permet de simplifier une partie du calcul de la fonction de Miller $f_{n,D}(\epsilon(D'))$ en évitant de calculer les contributions des verticales $d(\epsilon(D'))$ et $\tilde{u}(\epsilon(D'))$, définies sur un sous-corps propre de $\mathbb{F}_{q^k}$, lors de l'évaluation de chaque fonction $g_{[k1]D,[k2]D}$ en $\epsilon(D')$. Bien entendu, cela implique de travailler modulo les puissances ℓ-ièmes, ou bien que le couplage ait une exponentiation finale pour éliminer les éléments définis dans un sous-corps propre de $\mathbb{F}_{q^k}$.

Cette optimisation peut être très intéressante en pratique, car le dénominateur de $f_{n,D}$ est uniquement formé d'un produit de telles fonctions $\tilde{u}$. Nous parlons alors d'**élimination des verticales** ou d'**élimination des dénominateurs**.

De la même manière, pour $D = [u, v]$, la fonction $g_{D,[-1]D}$ est égale à u. Ainsi, si D et D' sont des éléments de $\mathrm{Jac}_{\mathcal{H}}[\ell]$ de valeur propre 1 ou $q \bmod \ell$ pour π_q, alors $g_{D,[-1]D}(D')$ est dans $\mathbb{F}_{q^{k/2}}$ et peut donc aussi être simplifié.

Chapitre 6

Approche unifiée pour la construction d'autres couplages

Nous introduisons dans ce chapitre deux techniques permettant de construire d'autres couplages. La première consiste à étudier l'action d'un endomorphisme sur les diviseurs en entrée du couplage de Tate et d'en déduire de nouveaux couplages. C'est cette technique qui conduit aux couplages Eta [DL03 ; Bar+07] et Ate [HSV06 ; Gra+07]. La seconde a été introduite par Hess et Vercauteren [Ver10 ; Hes08]. Elle consiste à explorer différents choix de paramètre pour la fonction de Miller grâce à la recherche de vecteurs courts dans un réseau euclidien. Nous verrons comment cette méthode permet de retrouver simplement d'autres couplages construits à partir de Ate et Eta, notamment le couplage Eta T [Bar+07] que nous avons utilisé dans la plupart de nos implémentations.

Ces deux techniques sont décrites dans les deux sections qui suivent. Elles sont appliquées afin d'obtenir les couplages que nous avons effectivement implémentés lors de cette thèse.

6.1 Action d'endomorphisme

Nous montrons dans cette section une construction unifiée des couplages Eta et Ate. Nous avons tenté de rassembler ici, pour plus de clarté, les preuves de [Bar+07 ; HSV06 ; Gra+07] en les étendant au cas des courbes hyperelliptiques. Bien que ces couplages aient été découverts séparément, ils reposent tous de façon essentielle sur le choix d'un endomorphisme de la courbe qu'il n'est pas facile d'exhiber dans les premières preuves ayant conduit à la construction de ces couplages. Cet endomorphisme interagit avec les fonctions de Miller, cela nous permet d'exploiter l'égalité donnée par le Lemme suivant [HSV06, Lemme 4 ; Gra+07, Lemme 5].

Lemme 6.1. *Soit ϕ un endomorphisme purement inséparable de $\mathcal{H}$ tel que $\phi(P_\infty) = P_\infty$. Soit D un diviseur réduit de $\mathcal{H}$. Nous avons alors, pour tout n et à un coefficient scalaire près,*

$$f_{n,\phi_*(D)} \circ \phi_* = f_{n,D}^{\deg \phi}.$$

Classiquement, l'endomorphisme de courbe est à choisir entre l'endomorphisme de Frobenius relatif à la puissance q-ième et son dual, le Verschiebung, quand il vérifie les conditions du lemme ; nous détaillons alors dans deux sous-sections les spécificités de leur utilisation. Nous

gardons pour le moment une formulation indépendante dans le Théorème 6.2 qui permet de couvrir les couplages Ate et Eta.

Théorème 6.2. *Soient $\mathcal{H}$ une courbe hyperelliptique définie sur $\mathbb{F}_q$ et ℓ un diviseur premier de $\#\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)$. Nous supposons que k, le degré de plongement de ℓ dans $\mathbb{F}_q$, est strictement plus grand que 1.*

Étant donné ϕ un endomorphisme purement inséparable de $\mathcal{H}$, pour toute valeur propre $\lambda \in \mathbb{Z}/\ell\mathbb{Z}$ de l'action de ϕ_* sur $\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell]$, nous notons

$$\mathbb{G}_{\phi,\lambda} = \mathrm{Ker}(\phi_* - [\lambda]) \cap \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell]$$

le sous-espace propre correspondant. En supposant que 1 et $q \bmod \ell$ sont deux telles valeurs propres de ϕ, nous fixons deux groupes cycliques d'ordre ℓ, $\mathbb{G}_1 \subset \mathbb{G}_{\phi,q}$ et $\mathbb{G}_2 \subset \mathbb{G}_{\phi,1}$.

Pour tous $\overline{D_1} \in \mathbb{G}_1$ et $\overline{D_2} \in \mathbb{G}_2$, nous avons alors

$$e_\ell(\overline{D_1}, \overline{D_2}) = f_{q,D_1}(D_2)^{kq^{k-1}}.$$

De plus, pour tout entier L premier avec k et tel que $\ell \mid L \mid q^k - 1$, nous pouvons définir le couplage

$$\begin{array}{ccccc} \mathbb{G}_1 \subset \mathbb{G}_{\phi,q} & \times & \mathbb{G}_2 \subset \mathbb{G}_{\phi,1} & \longrightarrow & \mu_\ell \\ (\overline{D_1} & , & \overline{D_2}) & \longmapsto & f_{q,D_1}(D_2)^{\frac{q^k-1}{L}}, \end{array}$$

qui est non-dégénéré si le couplage de Tate restreint aux mêmes groupes l'est et que $\ell^2 \nmid q^k - 1$.

Sous réserve de rencontrer ces conditions, le théorème précédent nous permet de construire d'autres couplages dont l'algorithme de Miller correspondant sera éventuellement plus court ($\log q$ itérations contre $\log \ell$ itérations pour Tate) et/ou plus simple (le poids de Hamming de q est éventuellement plus faible que celui de ℓ). Notons qu'afin de construire un couplage à partir de ce théorème, il nous faudra bien évidement restreindre le couplage à des groupes inclus dans $\mathrm{Ker}(\phi - [q]) \times \mathrm{Ker}(\phi - [1])$.

Nous commençons la preuve du Théorème 6.2 par une série de lemmes techniques autour des fonctions de Miller.

Lemme 6.3. *Soient $\overline{D} \in \mathrm{Jac}_{\mathcal{H}}[\ell]$, $m \in \mathbb{Z}$ et N tel que $\ell \mid N$, nous avons alors l'égalité des fonctions normalisées suivantes :*

$$f_{N,D}^m = f_{mN,D}.$$

Preuve. Comme $\overline{D}$ est de ℓ-torsion, nous avons

$$\mathrm{div}\, f_{mN,D} = mND = m\,\mathrm{div}\, f_{N,D} = \mathrm{div}\left(f_{N,D}^m\right). \qquad \square$$

Corollaire 6.4. *Soient N un entier tel que $\ell \mid N \mid q^k - 1$, et $\overline{D}, \overline{D'} \in \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell]$ avec D, D' de supports disjoints. Nous avons*

$$e_\ell(\overline{D}, \overline{D'}) = f_{N,D}(D')^{\frac{q^k-1}{N}}.$$

Preuve. Il s'agit d'une application immédiate du lemme précédent avec $N \leftarrow \ell$ et $m \leftarrow N/\ell$. $\square$

Ce dernier corollaire est déjà en soit une première optimisation pour le couplage de Tate étant donné qu'il permet par exemple de choisir un multiple N de ℓ dont le poids de Hamming sera faible pour une base bien choisie et ainsi donner un algorithme de Miller moins exigeant en calculs.

Le lemme suivant nous permet de changer le paramètre de la fonction de Miller d'une unité, c'est ce qui nous permettra de faire apparaître une puissance dans ce paramètre et d'utiliser le Lemme 6.6.

Lemme 6.5. *Soient $\overline{D} \in \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell]$ avec D un diviseur réduit. Si N est un multiple de ℓ, alors*

$$f_{N,D} = f_{N+1,D}.$$

Preuve. Nous appliquons la formule (5.1) de récurrence sur les fonctions de Miller :

$$\begin{aligned} f_{N+1,D} &= f_{N,D} \cdot f_{1,D} \cdot g_{[N]D,D} \\ &= f_{N,D} \cdot g_{0,D} \\ &= f_{N,D} \end{aligned}$$

car $\ell \mid N$ et $\overline{D}$ est un élément de ℓ-torsion. □

Le lemme suivant offre également une opportunité de réduction de la taille de la boucle de Miller. En effet, exprimer un couplage comme un produit de fonctions de Miller de même paramètre permet de partager une partie du calcul et d'en extraire également plus de parallélisme.

Lemme 6.6. *Pour tous diviseur D dans $\mathrm{Div}^0(\mathcal{H})$ et entiers $a, n \in \mathbb{Z}$, nous avons l'égalité suivante après normalisation :*

$$f_{n^a,D} = \prod_{i=0}^{a-1} \left(f_{n,[n^i]D}\right)^{n^{a-i-1}}.$$

Preuve. Pour prouver cette égalité il suffit de prouver l'égalité des diviseurs. Nous avons, d'une part,

$$\mathrm{div}\, f_{n^a,D} = n^a D - [n^a]D$$

et, d'autre part,

$$\mathrm{div}\left(f_{n,[n^i]D}\right)^{n^{a-i-1}} = n^{a-i-1} \cdot (n[n^i]D - [n^{i+1}]D) = n^{a-i}[n^i]D - n^{a-(i+1)}[n^{i+1}]D,$$

ce qui nous donne, par télescopage de la somme obtenue sur les diviseurs,

$$\mathrm{div} \prod_{i=0}^{a-1} \left(f_{n,[n^i]D}\right)^{n^{a-i-1}} = n^{a-0}[n^0]D - n^{a-(a-1+1)}[n^{(a-1)+1}]D = n^a D - [n^a]D. \qquad \square$$

Le dernier ingrédient permettant de construire les couplages Ate et Eta est l'exploitation d'un endomorphisme de la courbe $\mathcal{H}$ sur laquelle le couplage est construit. Soit ϕ un endomorphisme de $\mathcal{H}$ qui fixe le point à l'infini. Celui-ci s'étend naturellement en un morphisme ϕ_*

de diviseurs de $\mathcal{H}$ (*cf.* Définition 2.46). Le Lemme 6.1 donne alors l'égalité de fonctions de Miller suivante, à un coefficient scalaire près :

$$f_{n,\phi_*(D)} \circ \phi_* = f_{n,D}^{\deg \phi}.$$

Nous en donnons maintenant la preuve.

Preuve du Lemme 6.1. Nous commençons cette preuve en montrant l'égalité des fonctions appliquées aux points de $\mathcal{H}$:

$$f_{n,\phi_*(D)} \circ \phi = f_{n,D}^{\deg \phi}.$$

Remarquons en premier lieu que nous imposons $\phi(P_\infty) = P_\infty$ afin que l'image d'un diviseur réduit par le poussé-en-avant de ϕ soit alors lui aussi réduit. Étant donné $D = \sum_{i=1}^{d \leq g}(P_i) - d(P_\infty)$ un diviseur réduit, nous avons alors

$$\phi_*(D) = \sum_{i=1}^{d}(\phi(P_i)) - d(P_\infty).$$

Comme ϕ est purement inséparable, nous avons que $\forall i, \phi(P_i) \neq P_\infty$ et $\forall i \neq j, \phi(P_i) \neq \iota(\phi(P_j))$, ainsi nous obtenons bien un diviseur réduit.

Dans la suite de la preuve nous considérons le tiré-en-arrière de l'endomorphisme ϕ tel que défini en 2.15 : nous obtenons

$$\operatorname{div}(f_{n,\phi_*(D)} \circ \phi) = \operatorname{div} \phi^*(f_{n,\phi(D)}).$$

La Proposition 2.47.a. nous permet alors de transposer celui-ci aux diviseurs :

$$\operatorname{div} \phi^*(f_{n,\phi_*(D)}) = \phi^*(\operatorname{div} f_{n,\phi_*(D)}).$$

Nous continuons alors en intégrant la définition de la fonction de Miller :

$$\phi^*(\operatorname{div} f_{n,\phi_*(D)}) = \phi^*(n\phi_*(D) - [n]\phi_*(D)).$$

Pour pouvoir finir le calcul, nous devons alors montrer que la multiplication réduite par n et le poussé-en-avant ϕ_* commutent. En notant g et h les fonctions rationnelles qui interviennent dans la réduction de nD et $n\phi_*(D)$, nous avons

$$\begin{aligned}\phi_*([n]D) &= \phi_*(nD + \operatorname{div} g)\\ &= n\phi_*(D) + \phi_*(\operatorname{div} g)\\ &= [n]\phi_*(D) - \operatorname{div} h + \phi_*(\operatorname{div} g).\end{aligned}$$

Mais d'après la Proposition 2.47.b., $\phi_*(\operatorname{div} g)$ est principal, et donc $\phi_*([n]D)$ et $[n]\phi_*(D)$ ne diffèrent que d'un diviseur principal. Comme de plus $\phi_*([n]D)$ et $[n]\phi_*(D)$ sont tous deux réduits (ϕ est purement inséparable), ces deux diviseurs sont égaux.

Ensuite nous utilisons le fait que $\phi^* \circ \phi_*$ correspond à la multiplication non réduite par $\deg \phi$ (Proposition 2.47.c.) :

$$\begin{aligned}\operatorname{div}(f_{n,\phi_*(D)} \circ \phi) &= \phi^*(n\phi_*(D) - [n]\phi_*(D))\\ &= \phi^*(\phi_*(nD) - \phi_*([n]D))\\ &= \phi^*(\phi_*(nD - [n]D))\\ &= \deg \phi \cdot \operatorname{div} f_{n,D}.\end{aligned}$$

Cette égalité de diviseurs nous donne alors l'égalité suivante après normalisation des fonctions :

$$f_{n,\phi_*(D)} \circ \phi = f_{n,D}^{\deg \phi}.$$

Nous étendons maintenant cette égalité aux diviseurs. Soit $D' = \sum_i (P_i)$ un diviseur de support disjoint de celui de D. Grâce aux définitions de l'application d'une fonction en un diviseur et du poussé-en-avant, nous avons

$$\begin{aligned}
(f_{n,\phi_*(D)} \circ \phi_*)(D') &= f_{n,\phi_*(D)}\left(\phi_*\left(\textstyle\sum_i (P_i)\right)\right) \\
&= f_{n,\phi_*(D)}\left(\textstyle\sum_i (\phi(P_i))\right) \\
&= \textstyle\prod_i f_{n,\phi_*(D)}(\phi(P_i)) \\
&= \textstyle\prod_i f_{n,D}^{\deg \phi}(P_i) \\
&= f_{n,D}^{\deg \phi}(D'). \qquad \square
\end{aligned}$$

Nous avons maintenant obtenu tous les lemmes qui permettent le calcul d'un nouveau couplage et pouvons maintenant détailler la preuve du Théorème 6.2.

Preuve du Théorème 6.2. Nous commençons par montrer l'égalité principale du Théorème :

$$\begin{aligned}
e_\ell(\overline{D_1}, \overline{D_2}) &= f_{q^k-1,D_1}(D_2) && \text{d'après le Corollaire 6.4,} \\
&= f_{q^k,D_1}(D_2) && \text{d'après le Lemme 6.5,} \\
&= \prod_{i=0}^{k-1} f_{q,[q^i]D_1}(D_2)^{q^{(k-i-1)}} && \text{d'après le Lemme 6.6,} \\
&= \prod_{i=0}^{k-1} f_{q,\phi_*^i(D_1)}(\phi_*^i(D_2))^{q^{(k-i-1)}} \\
&= \prod_{i=0}^{k-1} f_{q,D_1}(D_2)^{q^{(k-i-1)}\cdot q^i} && \text{d'après le Lemme 6.1,} \\
&= \left(f_{q,D_1}(D_2)\right)^{kq^{k-1}}.
\end{aligned}$$

Ce qui montre la première partie du théorème. Pour la deuxième, nous élevons l'égalité à la puissance $\frac{q^k-1}{L}$:

$$\underbrace{e_\ell(\overline{D_1}, \overline{D_2})^{\frac{q^k-1}{L}}}_{\in \mu_\ell \subset \mu_L} = \underbrace{\left(f_{q,D_1}(D_2)^{\frac{q^k-1}{L}}\right)^{kq^{k-1}}}_{\in \mu_L},$$

et nous pouvons enfin écrire notre nouveau couplage comme une puissance du couplage de Tate :

$$f_{q,D_1}(D_2)^{\frac{q^k-1}{L}} = e_\ell(\overline{D_1}, \overline{D_2})^{\frac{q^k-1}{L}((kq^{k-1})^{-1} \bmod L)},$$

car nous avons supposé L premier avec k. Ainsi, ce nouveau couplage renvoie bien une racine ℓ-ième de l'unité. C'est une puissance non triviale du couplage de Tate car

$$\frac{q^k-1}{L}((kq^{k-1})^{-1} \bmod L) \not\equiv 0 \pmod{\ell}.$$

En effet, nous avons $\ell^2 \nmid q^k - 1$ donc $\ell \nmid \frac{q^k-1}{L}$ et comme k et q sont premiers avec L, $(kq^{k-1})^{-1} \bmod L$ l'est aussi.

Si, de plus, le couplage de Tate restreint à $\mathbb{G}_1 \times \mathbb{G}_2$ est non-dégénéré, le nouveau couplage est lui aussi non-dégénéré. □

Les optimisations de la Section 5.3 nous imposent déjà que l'un des groupes soit $\mathbb{F}_q$-rationnel et que l'autre soit inclus dans le sous-espace propre de valeur propre $q \bmod \ell$ du Frobenius. Il est donc naturel de penser au Frobenius de degré q : π_q. Dès lors une seconde branche s'offre à notre exploration, c'est d'utiliser le Verschiebung correspondant : $\hat{\pi}_q$, l'isogénie duale du Frobenius. Nous développons ces deux cas dans les deux sections qui suivent.

6.1.1 Frobenius

Comme nous l'avons vu à la Section 2.1.7 § 1, le Frobenius est un endomorphisme de la courbe purement inséparable, nous allons donc pouvoir utiliser le Théorème 6.2 avec $\phi \leftarrow \pi_q$. Nous aurons dans ce cas :

$$\begin{cases} \mathbb{G}_1 \subset \mathbb{G}_{\pi_q,q} = \mathrm{Jac}_{\mathcal{H}}[\ell] \cap \mathrm{Ker}((\pi_q)_* - [q]) \\ \mathbb{G}_2 \subset \mathbb{G}_{\pi_q,1} = \mathrm{Jac}_{\mathcal{H}}[\ell] \cap \mathrm{Ker}((\pi_q)_* - [1]) = \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)[\ell] \end{cases} \quad \text{donc} \quad \begin{cases} \pi_{q\,|\mathbb{G}_1} = [q], \\ \pi_{q\,|\mathbb{G}_2} = [1]. \end{cases}$$

Ainsi, en restreignant le couplage à $\mathbb{G}_1 \times \mathbb{G}_2$ et en utilisant l'endomorphisme π_q, nous remplissons les conditions du Théorème 6.2 et

$$\begin{array}{ccccc} \mathbb{G}_1 & \times & \mathbb{G}_2 & \longrightarrow & \mu_\ell \\ \overline{D_1} & , & \overline{D_2} & \longmapsto & f_{q,D_1}(D_2)^{\frac{q^k-1}{L}} \end{array}$$

est un couplage non-dégénéré quand le couplage de Tate restreint aux même groupe l'est et que ℓ est premier avec k et q.

Le lemme suivant nous permet de nous affranchir du besoin de calcul d'exponentiation finale en obtenant quand même une racine ℓ-ième de l'unité.

Lemme 6.7. *Soient $\overline{D_1}, \overline{D_2} \in \mathbb{G}_{\pi_q,q} \times \mathbb{G}_{\pi_q,1}$ avec D_1, D_2 de supports disjoints. Nous avons*

$$f_{q,D_1}(D_2) \in \mu_\ell.$$

Preuve. Nous commençons par montrer que l'évaluation en D_2 de f_{q,D_1} ne dépend pas du choix de D_2 en tant que représentant pour $\overline{D_2}$. Soit $h \in \mathbb{F}_q(\mathcal{H})^*$ de support disjoint avec celui de la fonction f_{q,D_1}, nous avons alors :

$$\begin{aligned} f_{q,D_1}(\mathrm{div}\, h) &= h(\mathrm{div}\, f_{q,D_1}) && \text{Réciprocité de Weil, Théorème 2.45} \\ &= h(qD_1 - [q]D_1) \\ &= h(qD_1 - (\pi_q)_*(D_1)) \\ &= h(D_1)^q / h((\pi_q)_*(D_1)) \\ &= h(D_1)^q / h(D_1)^q && \text{car } h \text{ est } \mathbb{F}_q\text{-rationnel} \\ &= 1. \end{aligned}$$

Comme D_2 est $\mathbb{F}_q$-rationnel, nous avons bien montré que l'évaluation de f_{q,D_1} en D_2 ne dépend pas du représentant choisi.

Il ne nous reste alors plus qu'à évaluer la puissance ℓ-ième de $f_{q,D_1}(D_2)$:

$$f_{q,D_1}(D_2)^\ell = f_{q,D_1}(\ell D_2) = f_{q,D_1}([\ell]D_2) = f_{q,D_1}(0) = 1. \qquad \square$$

Ainsi, nous avons défini un nouveau couplage qui est le couplage Ate dont nous donnons maintenant la définition.

Définition 6.8 (Couplage Ate)**.** Soient $\mathcal{H}$ une courbe hyperelliptique et ℓ un premier divisant $\#\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)$. Pour deux groupes $\mathbb{G}_1$ et $\mathbb{G}_2$ d'ordre ℓ tels que $\mathbb{G}_1 \subset \mathbb{G}_{\pi_q,q}$ et $\mathbb{G}_2 \subset \mathbb{G}_{\pi_q,1}$, nous définissons le **couplage Ate** par

$$\begin{array}{rccccc} a : & \mathbb{G}_1 & \times & \mathbb{G}_2 & \longrightarrow & \mu_\ell \\ & \overline{D_1} & , & \overline{D_2} & \longmapsto & f_{q,D_1}(D_2). \end{array}$$

Ce couplage est non-dégénéré lorsque le couplage de Tate restreint aux mêmes groupes l'est aussi, que $\ell^2 \nmid q^k - 1$ et que ℓ est premier avec k.

Cependant, cette définition du couplage Ate n'est pas nécessairement la plus pratique quand il s'agit d'obtenir une implémentation efficace de couplage : nous ne pouvons nous reposer sur l'exponentiation finale pour effectuer l'élimination de verticales décrite à la Section 5.3.2. Comme elle représente un gain algorithmique important, il est certainement préférable de pouvoir en bénéficier et d'ajouter pour cela une exponentiation finale. Dans ce cas l'exposant $q^{k/2} - 1$ sera suffisant.

Remarquons que nous n'avons pas nécessairement réduit la longueur de la boucle de Miller avec cette construction puisque

$$\log q = \frac{\rho}{g} \log \ell.$$

Cependant, nous avons obtenu un paramètre pour la fonction de Miller de forme généralement plus « simple », notamment pour les courbes en petite caractéristique où $q = p^m$. Nous nous servirons aussi du couplage Ate dans la Section 6.2 pour construire des couplages avec un nombre réduit d'itérations de Miller.

6.1.2 Verschiebung

Nous allons maintenant étudier l'utilisation du Verschiebung, le dual de l'endomorphisme du Frobenius, c'est-à-dire l'endomorphisme $\hat{\pi}_q$ tel que :

$$\pi_q \circ \hat{\pi}_q = \hat{\pi}_q \circ \pi_q = [\deg \pi_q] = [q].$$

Remarquons que le Verschiebung n'est pas nécessairement un endomorphisme purement inséparable de la courbe : c'est un endomorphisme de courbe seulement si la multiplication scalaire par q agit directement sur la courbe et non seulement sur la jacobienne[1] ; et il est purement inséparable si et seulement si la courbe $\mathcal{H}$ est très spéciale (ce qui est le cas si elle est supersingulière). Nous allons cependant supposer dans la suite de cette section que $\hat{\pi}_q$ est purement inséparable sur la courbe et nous verrons dans la suite comment utiliser cette construction quand ce n'est pas le cas.

1. C'est ainsi toujours le cas pour les courbes elliptiques.

Nous commençons, à l'instar de la construction du couplage Ate, à étudier l'action du Verschiebung sur deux groupes $\mathbb{G}_1$ et $\mathbb{G}_2$ cycliques d'ordre ℓ tels que :

$$\begin{cases} \mathbb{G}_1 \subset \mathbb{G}_{\hat{\pi}_q,q} = \mathrm{Jac}_{\mathcal{H}}[\ell] \cap \mathrm{Ker}((\hat{\pi}_q)_* - [q]) = \mathrm{Jac}_{\mathcal{H}}[\ell] \cap \mathrm{Ker}((\pi_q)_* - [1]) \text{ et} \\ \mathbb{G}_2 \subset \mathbb{G}_{\hat{\pi}_q,1} = \mathrm{Jac}_{\mathcal{H}}[\ell] \cap \mathrm{Ker}((\hat{\pi}_q)_* - [1]) = \mathrm{Jac}_{\mathcal{H}}[\ell] \cap \mathrm{Ker}((\pi_q)_* - [q]). \end{cases}$$

Nous avons donc bien :

$$\begin{cases} \hat{\pi}_q|_{\mathbb{G}_1} = [q] \\ \hat{\pi}_q|_{\mathbb{G}_2} = [1] \end{cases}.$$

Nous pouvons écrire ces égalités car π_q et $\hat{\pi}_q$ sont des morphismes injectifs de la jacobienne et que

$$\begin{aligned} \hat{\pi}_q \circ (\pi_q - [1]) &= [q] - \hat{\pi}_q, \text{ et} \\ \pi_q - [q] &= \pi_q \circ ([1] - \hat{\pi}_q). \end{aligned}$$

En restreignant le couplage en cours de construction à $\mathbb{G}_1 \times \mathbb{G}_2$, nous vérifions alors également les deux premières conditions du Théorème 6.2 avec $\phi \leftarrow \hat{\pi}_q$.

Définition 6.9 (Couplage Eta)**.** Soit $\mathcal{H}$ une courbe hyperelliptique définie sur $\mathbb{F}_q$ telle que $\hat{\pi}_q$ soit un endomorphisme purement inséparable de la courbe. Le **couplage Eta** est défini par

$$\begin{array}{rccccc} \eta : & \mathbb{G}_1 & \times & \mathbb{G}_2 & \longrightarrow & \mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^\ell \\ & \overline{D_1} & , & \overline{D_2} & \longmapsto & f_{q,D_1}(D_2). \end{array}$$

Ce couplage est non-dégénéré lorsque le couplage de Tate restreint aux mêmes groupes l'est aussi, que $\ell^2 \nmid q^k - 1$ et que ℓ est premier avec k.

Comme cette fois $\mathbb{G}_2$ n'est plus $\mathbb{F}_q$-rationnel, nous n'avons pas d'équivalent du Lemme 6.7 et une exponentiation finale est nécessaire pour obtenir une racine ℓ-ième de l'unité.

6.2 Couplages optimaux

Les couplages optimaux, introduits par Hess et Vercauteren [Ver10 ; Hes08], reposent sur l'observation simple qu'un produit, ou un rapport, de couplages donne aussi un couplage et que celui-ci ne sera toujours que le couplage de Tate à une puissance fixée. Comme nous l'avons vu avec Ate et Eta, une puissance du couplage de Tate peut être plus simple à calculer que le couplage de Tate lui-même. Vercauteren n'a appliqué sa technique qu'au couplage Ate bien que celle-ci fonctionne également dans le cas Eta comme nous l'avons montré dans [Ara+12]. Nous présentons donc ici une approche unifiée de la technique.

Dans la suite, nous utiliserons à plusieurs reprises la formule de récurrence suivante sur les fonctions de Miller :

$$f_{ab,D} = f_{a,[b]D} \cdot f_{b,D}^a, \tag{6.1}$$

qu'un simple calcul sur les diviseurs permet de vérifier.

6.2.1 Une première généralisation de Ate et Eta

Avant la construction des couplages optimaux, nous devons étendre les couplages Ate et Eta afin de pouvoir remplacer q par q^i dans le paramètre de la fonction de Miller associée à ces couplages.

Nous nous plaçons dans les conditions du Théorème 6.2. Soit ϕ un endomorphisme purement inséparable de la courbe $\mathcal{H}$. Nous restreignons donc toujours $\overline{D_1}$ à $\mathbb{G}_{\phi,q}$ et $\overline{D_2}$ à $\mathbb{G}_{\phi,1}$ et nous choisissons les représentants D_1 et D_2 de supports disjoints.

Pour vérifier que ces extensions de Ate et Eta sont bien des couplages, nous montrons que $f_{q^i,D_1}(D_2)$ est une puissance de $f_{q,D_1}(D_2)$. Plus précisément, nous montrons par récurrence l'égalité

$$f_{q^i,D_1}(D_2) = f_{q,D_1}^{iq^{i-1}}(D_2).$$

L'initialisation pour $i = 0$ est claire. Supposons l'égalité vraie pour i et développons l'égalité pour $i+1$:

$$\begin{aligned}
f_{q^{i+1},D_1}(D_2) &= f_{q,D_1}^{q^i}(D_2) \cdot f_{q^i,[q]D_1}(D_2) && \text{d'après l'Équation (6.1),}\\
&= f_{q,D_1}^{q^i}(D_2) \cdot f_{q^i,\phi_*(D_1)}(\phi_*(D_2)) &&\\
&= f_{q,D_1}^{q^i}(D_2) \cdot f_{q^i,D_1}^{q}(D_2) && \text{d'après le Lemme 6.1,}\\
&= f_{q,D_1}^{q^i}(D_2) \cdot (f_{q,D_1}^{iq^{i-1}}(D_2))^q && \text{par hypothèse de récurrence,}\\
&= f_{q,D_1}^{(i+1)q^i}(D_2). &&
\end{aligned}$$

Si, en plus d'être premier avec k et q, ℓ est premier avec i, nous pouvons définir les couplages $a_{q^i} = a^{iq^{i-1}}$ ou $\eta_{q^i} = \eta^{iq^{i-1}}$ selon que $\phi = \pi_q$ ou $\hat{\pi}_q$.

6.2.2 Constructions des couplages optimaux

Nous partons d'une puissance du couplage de Tate pour construire une famille de couplages. Comme nous voulons pouvoir choisir la forme de l'exposant final, nous choisissons un entier L tel que $\ell \mid L \mid q^k - 1$ et nous utilisons le Corollaire 6.4 pour exprimer le couplage de Tate :

$$e_\ell(\overline{D_1}, \overline{D_2}) = f_{L,D_1}(D_2)^{\frac{q^k-1}{L}}.$$

Nous choisissons alors un entier N multiple de L et posons M tel que $N = ML$. Le Lemme 6.3 nous donne alors l'égalité suivante :

$$e_\ell(\overline{D_1}, \overline{D_2})^M = f_{N,D_1}(D_2)^{\frac{q^k-1}{L}}.$$

L'idée de la construction des couplages optimaux est alors d'écrire N en base q :

$$N = \sum_{i=0}^{n-1} c_i q^i,$$

et d'exprimer la fonction f_{N,D_1} comme un produit de fonctions de Miller en utilisant cette décomposition :

$$\begin{aligned} f_{N,D_1} &= f_{\sum_{i=0}^{n-1} c_i q^i, D_1} \\ &= \prod_{i=0}^{n-1} \left(f_{c_i q^i, D_1} \cdot g_{[c_i q^i]D_1, [s_i]D_1} \right), \end{aligned}$$

où $s_i = c_{i+1} q^{i+1} + \cdots + c_{n-1} q^{n-1}$. Or nous avons également :

$$f_{c_i q^i, D_1} = f_{q^i, D_1}^{c_i} \cdot f_{c_i, [q^i]D_1}.$$

Nous imposons alors à D_1 et à D_2 de vérifier les mêmes conditions que précédemment :

$$\begin{cases} D_1 \in \mathbb{G}_{\phi,q} = \mathrm{Ker}(\phi_* - [q]) \cap \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell], \\ D_2 \in \mathbb{G}_{\phi,1} = \mathrm{Ker}(\phi_* - [1]) \cap \mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_{q^k})[\ell]. \end{cases}$$

Dès lors nous pouvons finir la réécriture de $f_{c_i q^i, D_1}$:

$$f_{c_i q^i, D_1}(D_2) = f_{q, D_1}^{i c_i q^{i-1}} \cdot f_{c_i, (\phi_*)^i(D_1)}((\phi_*)^i(D_2)) = f_{q, D_1}^{i c_i q^{i-1}} \cdot f_{c_i, D_1}(D_2)^{q^i},$$

ainsi que celle de f_{N,D_1} en produit de fonctions de Miller :

$$f_{N,D_1}(D_2) = \left[\prod_{i=0}^{n-1} \left(f_{c_i, D_1}^{q^i} \cdot g_{[c_i q^i]D_1, [s_i]D_1} \right)(D_2) \right] \cdot f_{q, D_1}(D_2)^{\sum_{i=0}^{n-1} i c_i q^{i-1}}.$$

En choisissant N tel que les chiffres c_i de sa décomposition en base q soient « petits », l'expression entre crochets n'utilise que des fonctions de Miller de « petits » paramètres. Nous utiliserons donc ces fonctions pour définir un couplage optimal. Nous verrons dans la section suivante comment le choix d'un tel N se fait. Nous posons alors

$$e_{(c_0,\ldots,c_{n-1})}(\overline{D_1}, \overline{D_2}) = \left(\prod_{i=0}^{n-1} \left(f_{c_i, D_1}^{q^i} \cdot g_{[c_i q^i]D_1, [s_i]D_1} \right)(D_2) \right)^{\frac{q^k-1}{L}},$$

et pouvons maintenant écrire une égalité de couplages :

$$e_\ell(\overline{D_1}, \overline{D_2})^M = e_{(c_0,\ldots,c_{n-1})}(\overline{D_1}, \overline{D_2}) \cdot f_{q, D_1}(D_2)^{\frac{q^k-1}{L} \sum_{i=0}^{n-1} i c_i q^{i-1}}.$$

Cette égalité montre que le couplage optimal $e_{(c_0,\ldots,c_{n-1})}$ est une puissance du couplage de Tate :

$$e_{(c_0,\ldots,c_{n-1})} = e_\ell^{M - \frac{q^k-1}{L}((kq^{k-1})^{-1} \bmod \ell) \sum_{i=0}^{n-1} i c_i q^{i-1}}.$$

Ainsi, ce couplage sera non-dégénéré quand le couplage de Tate restreint aux mêmes groupes $\mathbb{G}_1$ et $\mathbb{G}_2$ l'est et que

$$M \not\equiv \frac{q^k-1}{L}((kq^{k-1})^{-1} \mod \ell) \sum_{i=0}^{n-1} i c_i q^{i-1} \pmod{\ell},$$

ou, plus simplement, quand

$$Mkq^{k-1} \not\equiv \frac{q^k-1}{L}\sum_{i=0}^{n-1} ic_i q^{i-1} \pmod{\ell}.$$

Nous pouvons alors définir les couplages Ate et Eta optimal.

Définition 6.10 (Couplage Ate optimal)**.** Soient $\mathcal{H}$ une courbe hyperelliptique, ℓ un premier divisant $\#\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)$ et k le degré de plongement de ℓ dans $\mathbb{F}_q$. Soient également un entier L tel que $\ell \mid L \mid q^k-1$ et un multiple $N = \sum_{i=0}^{n-1} c_i q^i$ de L. Nous posons M tel que $N = ML$. Pour deux groupes $\mathbb{G}_1$ et $\mathbb{G}_2$ d'ordre ℓ tels que $\mathbb{G}_1 \subset \mathbb{G}_{\pi_q,q}$ et $\mathbb{G}_2 \subset \mathbb{G}_{\pi_q,1}$, nous définissons le **couplage Ate optimal** par

$$\begin{array}{rccccc} a_{(c_0,\ldots,c_{n-1})}: & \mathbb{G}_1 & \times & \mathbb{G}_2 & \longrightarrow & \mu_\ell \\ & \overline{D_1} & , & \overline{D_2} & \longmapsto & \left(\prod_{i=0}^{n-1} f_{c_i,D_1}^{q^i} \cdot g_{[c_iq^i]D_1,[s_i]D_1}\right)(D_2)^{\frac{q^k-1}{L}}. \end{array}$$

Ce couplage est non-dégénéré quand

$$Mkq^{k-1} \not\equiv \frac{q^k-1}{L}\sum_{i=0}^{n-1} ic_i q^{i-1} \pmod{\ell},$$

et que le couplage de Tate restreint aux mêmes groupes l'est aussi.

Définition 6.11 (Couplage Eta optimal)**.** Soient $\mathcal{H}$ une courbe hyperelliptique, ℓ un premier divisant $\#\mathrm{Jac}_{\mathcal{H}}(\mathbb{F}_q)$ et k le degré de plongement de ℓ dans $\mathbb{F}_q$. Soit également un entier L tel que $\ell \mid L \mid q^k-1$ et un multiple $N = \sum_{i=0}^{n-1} c_i q^i$ de L. Nous posons M tel que $N = ML$. Pour deux groupes $\mathbb{G}_1$ et $\mathbb{G}_2$ d'ordre ℓ tels que $\mathbb{G}_1 \subset \mathbb{G}_{\hat{\pi}_q,q}$ et $\mathbb{G}_2 \subset \mathbb{G}_{\hat{\pi}_q,1}$, nous définissons le **couplage Eta optimal** par

$$\begin{array}{rccccc} \eta_{(c_0,\ldots,c_{n-1})}: & \mathbb{G}_1 & \times & \mathbb{G}_2 & \longrightarrow & \mu_\ell \\ & \overline{D_1} & , & \overline{D_2} & \longmapsto & \left(\prod_{i=0}^{n-1} f_{c_i,D_1}^{q^i} \cdot g_{[c_iq^i]D_1,[s_i]D_1}\right)(D_2)^{\frac{q^k-1}{L}}. \end{array}$$

Ce couplage est non-dégénéré quand

$$Mkq^{k-1} \not\equiv \frac{q^k-1}{L}\sum_{i=0}^{n-1} ic_i q^{i-1} \pmod{\ell},$$

et que le couplage de Tate restreint aux mêmes groupes l'est.

Il est intéressant de constater qu'il reste des fonctions rationnelles issues de l'algorithme de Cantor dans ces expressions ($g_{[c_iq^i]D,[s_i]D}$). Nous verrons sur des exemples plus tard que nous pourrons extraire des verticales de celles-ci et les éliminer ainsi que décrit à la Section 5.3.2.

6.2.3 Choix des paramètres et argument d'optimalité

Étant donnée la construction précédente, nous cherchons maintenant un multiple N de L dont l'écriture en base q ne fait intervenir que de petits coefficients :

$$N = ML = \sum_{i=0}^{n-1} c_i q^i, \text{ avec les } c_i \text{ « petits »}.$$

Afin de garantir la non-dégénérescence du couplage optimal obtenu, nous devons vérifier la congruence suivante :

$$Mkq^{k-1} \not\equiv \frac{q^k - 1}{L} \sum_{i=0}^{n-1} i c_i q^{i-1} \pmod{\ell}.$$

De sorte à limiter notre espace de recherche, nous commençons par constater que le polynôme cyclotomique Φ_k donne une relation à petits coefficients entre les puissances de q ($\Phi_k(q) \equiv 0 \pmod{\ell}$). Il n'est donc pas nécessaire de chercher des multiples ayant plus de φ_k chiffres dans leur écriture en base q.

Qui plus est, en choisissant $L = \ell$ et $N = \Phi_k(q)$, nous aurons bien $\ell \mid \Phi_k(q)$ avec N s'écrivant avec de petits coefficients mais obtiendrons toujours un couplage dégénéré [Ver10, Corollaire 6].

L'idée est alors de constater que les écritures en base q avec au plus φ_k chiffres des multiples de L forment un réseau euclidien dont une base est donnée par les lignes de la matrice

$$\begin{pmatrix} L & 0 & \cdots & \cdots & 0 \\ -q & 1 & \ddots & & \vdots \\ -q^2 & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ -q^{\varphi_k - 1} & 0 & \cdots & 0 & 1 \end{pmatrix}.$$

En cherchant parmi les vecteurs courts de ce réseau, nous aurons une liste de candidats pour obtenir des couplages optimaux.

Enfin, comme le volume du réseau euclidien que nous considérons est L, nous pouvons espérer trouver des vecteurs courts de normes plus petite environ que $L^{\frac{1}{\varphi_k}}$.

Chapitre 7

Application au cas des courbes supersingulières

Nous nous attachons, dans ce chapitre, à appliquer les algorithmes de couplage que nous avons développés lors des deux chapitres précédents aux courbes supersingulières qui ont retenu notre attention lors de cette thèse. Outre l'avantage arithmétique qu'elles fournissent en étant définies sur des corps de petite caractéristique, ces courbes ont une géométrie particulière qu'il convient d'exploiter pour obtenir des algorithmes efficaces de calcul de couplages. C'est cet aspect que nous étudions dans ce chapitre.

Rappelons qu'outre une arithmétique efficace, les courbes supersingulières fournissent une représentation compacte du groupe non-$\mathbb{F}_q$-rationnel par l'intermédiaire d'une *distortion map* ψ. Cependant, pour pouvoir utiliser les couplages Ate et Eta, nous devons nous restreindre à des groupes qui sont des sous-espaces propres du Frobenius π_q. Nous choisirons donc la *distortion map* par des techniques de diagonalisation de la matrice du Frobenius π_q dans la ℓ-torsion pour que son image soit un sous-espace propre.

Nous commençons par étudier le cas elliptique (Sections 7.1 et 7.2). Celui-ci bénéficie d'un grand nombre de simplifications par rapport au cadre général ; en effet, la jacobienne de courbe est isomorphe à la courbe dans ce cas, et ainsi tout diviseur réduit pourra être identifié à un point de la courbe :

$$\begin{array}{rcl} E(\mathbb{F}_{q^i}) & \longrightarrow & \mathrm{Jac}_E(\mathbb{F}_{q^i}) \\ P & \longmapsto & (P) - (P_\infty) = [x - x_P, y_P], \end{array}$$

et

$$\begin{array}{rcl} \mathrm{Jac}_E(\mathbb{F}_{q^i}) & \longrightarrow & E(\mathbb{F}_{q^i}) \\ \overline{D} = [x - x_P, y_P] & \longmapsto & P \text{ où } \rho(D) = (P) - (P_\infty). \\ 0 & \longmapsto & P_\infty \end{array}$$

Les coordonnées que nous utiliserons pour les calculs seront donc directement celles des points ; remarquons qu'elles sont très proches de celles de Mumford.

Nous continuerons alors l'étude par les courbes supersingulières de genre et caractéristique 2 à la Section 7.3.

7.1 Courbes elliptiques en caractéristique 2

Nous nous intéressons ici à la courbe

$$E_2/\mathbb{F}_2 : y^2 + y = x^3 + x + b \text{ avec } b \in \{0, 1\}.$$

Comme nous l'avons vu à la Section 4.3.1 § 1, celle-ci est de degré de plongement $k = 4$ et nous travaillerons avec sa jacobienne sur un corps $\mathbb{F}_{2^m}$ avec m impair ; celle-ci est isomorphe à la courbe et de cardinal

$$\#E_2(\mathbb{F}_{2^m}) = 2^m + \nu 2^{\frac{m+1}{2}} + 1, \text{ avec } \nu = \begin{cases} (-1)^b & \text{si } m \equiv \pm 1 \pmod 8, \\ -(-1)^b & \text{si } m \equiv \pm 3 \pmod 8. \end{cases}$$

7.1.1 *Distortion map* et endomorphismes de E_2

Nous exhibons tout d'abord une *distortion map* de E_2. Comme c'est un automorphisme de la courbe, nous la cherchons parmi ceux-ci grâce à la caractérisation donnée par Silverman dans sa preuve de [Sil86, Proposition A.1.2]. C'est également un automorphisme $\mathbb{F}_{2^{4m}}$-rationnel qui n'est ni $\mathbb{F}_{2^m}$-rationnel, ni $\mathbb{F}_{2^{2m}}$-rationnel ; en appliquant ces conditions à la description de Silverman, les *distortion maps* sont à chercher parmi les automorphismes :

$$\sigma_{s,t} : (x, y) \longmapsto (x + s^2, y + sx + t),$$

avec $s^2 + s + 1 = 0$ et $t^2 + t + s = 0$. Nous fixons alors deux racines s et t qui définissent la tour d'extensions suivante que nous utiliserons pour le calcul de couplage :

$$\begin{array}{c} \mathbb{F}_{2^{4m}} = \mathbb{F}_{2^{2m}}[t] \\ | \\ \mathbb{F}_{2^{2m}} = \mathbb{F}_{2^m}[s] \\ | \\ \mathbb{F}_{2^m} \end{array}$$

Nous souhaitons alors utiliser la *distortion map* ψ définie par $\sigma_{s,t}$; cependant, nous devons encore vérifier qu'elle envoie bien $\mathbb{G}_{\pi_{2^m},1} = E_2(\mathbb{F}_{2^m})[\ell]$ dans $\mathbb{G}_{\pi_{2^m},2^m} = \mathrm{Ker}(\pi_{2^m} - [2^m]) \cap E_2(\mathbb{F}_{2^m})[\ell]$. Soit $P \in E_2(\mathbb{F}_{2^m})$, montrons que $\psi(P) \in \mathbb{G}_{\pi_{2^m},2^m}$, c'est-à-dire que $\pi_{2^m}(\psi(P)) = [2^m]\psi(P)$.

$$\begin{aligned} \pi_{2^m}(\psi(P)) &= (x_P + (s^2)^{2^m}, \quad y_P + s^{2^m} x_P + t^{2^m}) \\ &= (x_P + s, \qquad y_P + (s+1)x_P + t + s + \lfloor m/2 \rfloor) \end{aligned}$$

Nous donnons alors la formule du doublement itéré pour la courbe E_2, qui se déduit aisément de la loi de la corde et la tangente :

$$[2^i](x, y) = (x^{2^{2i}} + i, y^{2^{2i}} + i x^{2^{2i}} + \lfloor i/2 \rfloor). \tag{7.1}$$

Le calcul se poursuit ainsi :

$$\begin{aligned}
[2^m]\psi(P) &= (x_P + (s^2)^{2^{2m}} + 1, & y_P + s^{2^{2m}} x_P + t^{2^{2m}} + x_P + (s^2)^{2^{2m}} + \lfloor m/2 \rfloor) \\
&= (x_P + (s+1) + 1, & y_P + s x_P + t + 1 + x_P + s + 1 + \lfloor m/2 \rfloor) \\
&= (x_P + s, & y_P + (s+1)x_P + t + s + \lfloor m/2 \rfloor) \\
&= \pi_{2^m}(\psi(P)).
\end{aligned}$$

Nous avons alors montré qu'il est possible d'utiliser le Frobenius π_{2^m} pour obtenir des couplages Ate.

Afin de construire des couplages Eta, nous utiliserons le Verschiebung $\hat{\pi}_{2^m}$. Comme nous l'avons vu en Section 6.1.2, la seule hypothèse supplémentaire à vérifier est que le Verschiebung est bien un endomorphisme de la courbe : c'est le cas car nous sommes en genre 1 et que la jacobienne est isomorphe à la courbe. C'est un endomorphisme purement inséparable car la courbe E_2 est très spéciale. L'expression du Verschiebung s'obtient grâce à l'identité $\pi_{2^m} \circ \hat{\pi}_{2^m} = [2^m]$:

$$\hat{\pi}_{2^m} = \gamma_{E_2} \circ \pi_{2^m} \text{ où } \gamma_{E_2}(x, y) = (x + 1, y + x + \lfloor m/2 \rfloor).$$

7.1.2 Construction de couplages optimaux

Soit ℓ un grand diviseur premier du cardinal de la courbe $E_2(\mathbb{F}_{2^m})$. Nous cherchons maintenant à fixer le L intervenant dans la construction des couplages optimaux (Section 6.2). Comme nous voulons bénéficier d'une exponentiation finale efficace, nous choisissons $L = \#E_2(\mathbb{F}_{2^m})$. L'exposant final correspondant est alors

$$\frac{q^k - 1}{L} = \frac{2^{4m} - 1}{2^m + \nu 2^{\frac{m+1}{2}} + 1} = (2^{2m} - 1)(2^m - \nu 2^{\frac{m+1}{2}} + 1) \ ;$$

ainsi, nous obtenons une forme structurée qui permettra de développer un algorithme *ad-hoc*. De plus, cet exposant est bien un multiple de $q^{k/2} - 1 = 2^{2m} - 1$ et nous pourrons donc bénéficier de l'élimination de verticales (Section 5.3.2).

Nous cherchons alors à définir un couplage optimal et devons donc chercher des vecteurs courts dans le réseau euclidien formé par les lignes la matrice

$$\begin{pmatrix} 2^m + \nu 2^{\frac{m+1}{2}} + 1 & 0 \\ -2^m & 1 \end{pmatrix}.$$

Parmi les vecteurs courts de ce réseau, nous trouvons

$$\left(2^{\frac{m-1}{2}} + \nu, -2^{\frac{m-1}{2}}\right) \text{ et } \left(2^{\frac{m+1}{2}} + \nu, \nu\right).$$

Nous écartons le premier vecteur car bien qu'il conduise à une boucle de Miller plus courte d'une itération, il nécessite de construire deux fonctions de Miller en même temps.

En notant le second vecteur $(c_0, c_1) = (2^{\frac{m+1}{2}} + \nu, \nu)$, le couplage $e_{(c_0,c_1)}$ ainsi obtenu est donnée par le produit

$$e_{(c_0,c_1)}(P, .) = f_{c_0,P} \cdot g_{[c_0]P,[\nu 2^m]P} \cdot f_{\nu,P}^{2^m} \cdot g_{[\nu 2^m]P,[0]P}.$$

En remarquant que $c_0 + \nu 2^m = \nu \# E_2(\mathbb{F}_{2^m})p \equiv 0 \pmod{\ell}$, la fonction $g_{[c_0]P,[\nu 2^m]P}$ est une verticale que nous pouvons éliminer. Enfin, selon que $\nu = 1$ ou -1, la fonction de Miller $f_{\nu,P}$ est soit triviale, soit une verticale que nous éliminons aussi. Nous avons alors le couplage

$$e_{(c_0,c_1)}(P,.) \equiv f_{c_0,P} \pmod{(\mathbb{F}_{2^{4m}}^*)^\ell}.$$

Remarquons que nous n'avons pas encore eu à choisir entre une version Ate ou Eta du couplage.

7.1.3 Algorithme de Miller et arithmétique de courbe

Afin de simplifier l'itération de Miller, nous choisissons de développer la fonction obtenue :

$$\begin{aligned} f_{2^{\frac{m+1}{2}}+\nu,P} &= f_{2^{\frac{m+1}{2}},P} \cdot g_{\left[2^{\frac{m+1}{2}}\right]P,[\nu]P} \\ &= \left(\prod_{i=0}^{\frac{m-1}{2}} g_{[2^i]P,[2^i]P}^{2^{\frac{m-1}{2}-i}} \right) \cdot g_{\left[2^{\frac{m+1}{2}}\right]P,[\nu]P}. \end{aligned}$$

Le calcul revient donc à l'accumulation d'équations de droites de doublement :

$$g_{P,P}(X,Y) = (x_P^2 + 1) \cdot X + Y + y_P + b.$$

Afin de limiter l'impact sur la mémoire, nous intégrons l'évaluation de la fonction de Miller à son calcul. La stratégie est alors d'appliquer la *distortion map* le plus tard possible afin de conserver des coordonnées compactes pour les points de $\mathbb{G}_{\pi_{2^m},2^m} = \psi(\mathbb{G}_{\pi_{2^m},1})$. C'est-à-dire soit dans le calcul, soit dans l'évaluation des fonctions $g_{P,P}$ selon que nous sommes dans le cas Eta ou Ate :

$$\begin{array}{rlcccl} g_{\psi(P),\psi(P)}(Q) = & \left[(x_P^2+1)x_Q + y_P + y_Q + b\right] & + & \left[x_P + x_Q\right] s & + & t, \\ g_{P,P}(\psi(Q)) = & \left[(x_P^2+1)(x_Q+1) + y_P + y_Q + b\right] & + & \left[x_P^2 + x_Q + 1\right] s & + & t. \end{array}$$

Nous pouvons remarquer que ces deux formules fournissent un multiplicande creux. Nous exploiterons alors cette propriété afin de réduire le coût de chaque itération. En utilisant les résultats et l'Algorithme 9.2 de la partie suivante, nous pouvons déduire qu'une formule pour ce produit, optimale en nombre de multiplications sur $\mathbb{F}_{2^m}$, utilisera 6 multiplications et 14 additions. Nous verrons cependant que ce n'est pas nécessairement l'algorithme à utiliser en adéquation avec l'architecture cible.

Les deux formules précédentes donnent des algorithmes avec des coûts très similaires et montrent que le choix entre les couplages Ate et Eta ne fournit pas d'amélioration significative de la performance du couplage. Nous utiliserons alors le couplage Eta dans la suite afin de pouvoir se comparer aux travaux de la littérature. Il est à noter que le couplage ainsi défini correspond au couplage η_T présenté par Barreto *et al.* dans [Bar+07].

Avant de pouvoir expliciter un algorithme, nous devons donner une formule pour la dernière droite à accumuler, c'est-à-dire celle qui passe par $\left[2^{\frac{m+1}{2}}\right] P$ et $[\nu]P$. Celle-ci est de forme particulière car les points ont des coordonnées simples :

$$\begin{aligned} \left[2^{\frac{m+1}{2}}\right] P &= (x_P^2 + \alpha, y_P^2 + \alpha x_P^2 + \delta + b), \\ [\nu]P &= (x_P, y_P + \delta x_P), \end{aligned}$$

où $\alpha = \frac{m+1}{2} \bmod 2$. L'évaluation de la droite finale donne alors :

$$g_{\left[2\frac{m+1}{2}\right]P,[\nu]P}(\psi(Q)) = \left(x_P^2 + (x_P+\alpha)(x_Q+\alpha) + x_Q + y_P + y_Q + \delta + 1\right) + s\cdot(x_P + x_Q + 1 - \alpha) + t.$$

Nous pouvons alors écrire un algorithme implémentant les formules précédentes (Algorithme 7.1) en remarquant

— que le premier produit d'accumulation n'est pas nécessaire, et
— qu'une partie de l'évaluation de la droite finale est déjà contenue dans la boucle de Miller précédente.

Afin d'expliciter le coût arithmétique de cet algorithme, nous introduisons les notations suivantes :

a	addition dans $\mathbb{F}_{2^m}$
a′	ajout de 1
f	Frobenius dans $\mathbb{F}_{2^m}$ $(x \mapsto x^2)$
f′	Frobenius inverse dans $\mathbb{F}_{2^m}$ $(x \mapsto \sqrt{x})$
F	Frobenius dans $\mathbb{F}_{2^{4m}}$ $(x \mapsto x^2)$
m	multiplication dans $\mathbb{F}_{2^m}$
M	multiplication dans $\mathbb{F}_{2^{4m}}$
M′	multiplication « creux par plein » $(a_0 + a_1 s + t)(b_0 + b_1 s + b_2 t + b_3 st)$
M″	multiplication « creux par creux » $(a_0 + a_1 s + t)((a_0 + a_2) + (a_1 + 1)s + t)$
i	inversion dans $\mathbb{F}_{2^m}$

7.1.4 Raffinements algorithmiques

Nous donnons ici deux idées qui permettent de transformer l'algorithme précédent et de diminuer son coût. Elles sont extraites des travaux présentés par Beuchat *et al.* dans [Beu+08a].

La première idée, que nous détaillons dans l'Algorithme 7.2, est de calculer la boucle de Miller dans l'autre sens, c'est-à-dire d'incrémenter i plutôt que de le décrémenter. Ceci permet d'éliminer la nécessité d'élever au carré la variable d'accumulation F à chaque itération mais introduit en contrepartie le besoin d'une opération supplémentaire : le Frobenius inverse $(\sqrt{.})$ [1]. Cette variante permet également d'avancer le calcul de la droite finale au début de l'algorithme, simplifiant ainsi le produit correspondant. Cet algorithme introduit un nouveau paramètre :

$$\beta = \begin{cases} b & \text{si } m \equiv 1 \text{ ou } 3 \pmod 8, \\ 1-b & \text{si } m \equiv 5 \text{ ou } 7 \pmod 8. \end{cases}$$

La deuxième variante proposée par Beuchat *et al.* et présentée dans l'Algorithme 7.3 cherche à éliminer le calcul des racines carrées, supprimant ainsi le besoin d'avoir un opérateur supplémentaire dédié à cette opération.

1. Il est intéressant de noter que pour certains corps binaires, le calcul du Frobenius inverse (racine carrée) est moins coûteux que celui du Frobenius (carré).

Algorithme 7.1 Algorithme pour le couplage Eta optimal sur E_2.

Entrées : $P = (x_P, y_P)$, $Q = (x_Q, y_Q) \in E_2(\mathbb{F}_{2^m})[\ell]$.
Sortie : $\eta_T(P,Q) = \eta_{(c_0,c_1)}(P,Q) \in \mathbb{F}^*_{2^{4m}}$.

1. $y_Q \leftarrow y_Q + b$ — $(b\ \mathsf{a}')$
2. $g_1 \leftarrow x_P + x_Q$ — $(1\ \mathsf{a})$
3. $x_V \leftarrow x_P^2 + 1$ — $(1\ \mathsf{f} + 1\ \mathsf{a}')$
4. $g_0 \leftarrow x_V \cdot x_Q + y_V + y_Q$ — $(1\ \mathsf{m} + 2\ \mathsf{a})$
5. $F \leftarrow (g_0 + g_1 s + t)$
6. $x_V \leftarrow x_V^2$ — $(1\ \mathsf{f})$
7. $y_V \leftarrow y_V^4 + x_V + 1$ — $(2\ \mathsf{f} + 1\ \mathsf{a} + 1\ \mathsf{a}')$
8. **for** $i = \frac{m-3}{2}$ **to** 0 **do**
9. $\quad F \leftarrow F^2$ — $(4\ \mathsf{F})$
10. $\quad g_1 \leftarrow x_V + x_Q$ — $(1\ \mathsf{a})$
11. $\quad x_V \leftarrow x_V^2 + 1$ — $(1\ \mathsf{f} + 1\ \mathsf{a}')$
12. $\quad g_0 \leftarrow x_V \cdot x_Q + y_V + y_Q$ — $(1\ \mathsf{m} + 2\ \mathsf{a})$
13. $\quad F \leftarrow F \cdot (g_0 + g_1 s + t)$ — $(1\ \mathsf{M}')$
14. $\quad x_V \leftarrow x_V^2$ — $(1\ \mathsf{f})$
15. $\quad y_V \leftarrow y_V^4 + x_V + 1$ — $(2\ \mathsf{f} + 1\ \mathsf{a} + 1\ \mathsf{a}')$
16. **end for**
17. $g_0 \leftarrow g_0 + x_P + x_Q + \alpha$ — $(2\ \mathsf{a} + \alpha\ \mathsf{a}')$
18. $g_1 \leftarrow g_1 + 1$ — $(1\ \mathsf{a}')$
19. $F \leftarrow F \cdot (g_0 + g_1 s + t)$ — $(1\ \mathsf{M}')$
20. **return** $F^{\frac{2^{4m}-1}{L}}$

Algorithme 7.2 Calcul du couplage η_T en caractéristique 2 avec le retournement de boucle.

Entrées : $P = (x_P, y_P)$, $Q = (x_Q, y_Q) \in E_2(\mathbb{F}_{2^m})[\ell]$.
Sortie : $\eta_T(P,Q) \in \mathbb{F}^*_{2^{4m}}$.

1. $x_P \leftarrow x_P$; $y_P \leftarrow y_P + \bar{\delta}$ ($\bar{\delta}$ a′)
2. $u \leftarrow x_P + \alpha$; $v \leftarrow x_Q + \alpha$ (2α a′)
3. $w \leftarrow y_P + y_Q + \beta$ (1 a + β a′)
4. $g_0 \leftarrow u \cdot v + w$ (1 m + 1 a)
5. $g_1 \leftarrow u + x_Q$; $g_2 \leftarrow v + (x_P)^2$; (2 a + 1 f)
6. $F \leftarrow (g_0 + g_2) + (g_1 + 1)\, s + t$
7. $G \leftarrow g_0 + g_1 s + t$
8. $F \leftarrow F \cdot G$ (1 M″)
9. **for** $i = 1$ **to** $\frac{m-1}{2}$ **do**
10. $x_P \leftarrow \sqrt{x_P}$; $y_P \leftarrow \sqrt{y_P}$ (2 f′)
11. $x_Q \leftarrow (x_Q)^2$; $y_Q \leftarrow (y_Q)^2$ (2 f)
12. $u \leftarrow x_P + \alpha$; $v \leftarrow x_Q + \alpha$ (2α a′)
13. $w \leftarrow y_P + y_Q + \beta$ (1 a + β a′)
14. $g_0 \leftarrow u \cdot v + w$ (1 m + 1 a)
15. $g_1 \leftarrow u + x_Q$ (1 a)
16. $G \leftarrow g_0 + g_1 s + t$
17. $F \leftarrow F \cdot G$ (1 M′)
18. **end for**
19. **return** $F^{\frac{2^{4m}-1}{L}}$

Algorithme 7.3 Calcul du couplage η_T en caractéristique 2 avec le retournement de boucle sans racine carrée [Beu+08a, Algorithme 2].

Entrées : $P = (x_P, y_P)$, $Q = (x_Q, y_Q) \in E_2(\mathbb{F}_{2^m})[\ell]$.
Sortie : $\eta_T(P, Q) \in \mathbb{F}^*_{2^{4m}}$.

1. $x_P \leftarrow x_P$; $y_P \leftarrow y_P + \bar{\delta}$ ($\bar{\delta}$ a′)
2. $x_P \leftarrow {x_P}^2$; $y_P \leftarrow {y_P}^2$ (2 f)
3. $u \leftarrow x_P + 1$; $y_P \leftarrow y_P + b$ ($(1+b)$ a′)
4. $g_1 \leftarrow u + x_Q$; (1 a)
5. $g_0 \leftarrow x_P \cdot x_Q + y_P + y_Q + g_1$ (1 m + 3 a)
6. $x_Q \leftarrow x_Q + 1$ (1 a′)
7. $g_2 \leftarrow x_P^2 + x_Q$; (1 a + 1 f)
8. $F \leftarrow (g_0 + g_2) + (g_1 + 1)\, s + t$
9. $G \leftarrow g_0 + g_1 s + t$
10. $F \leftarrow F \cdot G$ (1 M″)
11. **for** $i = 1$ **to** $\frac{m-1}{2}$ **do**
12. $\quad F \leftarrow F^2$ (4 F)
13. $\quad x_Q \leftarrow x_Q^4$; $y_Q \leftarrow y_Q^4$ (4 f)
14. $\quad x_Q \leftarrow x_Q + 1$; $y_Q \leftarrow y_Q + x_Q$ (1 a + 1 a′)
15. $\quad g_0 \leftarrow u \cdot x_Q + y_P + y_Q$ (1 m + 2 a)
16. $\quad g_1 \leftarrow x_P + x_Q$ (1 a)
17. $\quad G \leftarrow g_0 + g_1 s + t$
18. $\quad F \leftarrow F \cdot G$ (1 M′)
19. **end for**
20. **return** $F^{\frac{2^{4m}-1}{L}}$

7.1.5 Exponentiation finale

Nous détaillons maintenant le calcul de l'exponentiation finale pour le couplage $\eta_{(c_0,c_1)}$: c'est-à-dire que nous cherchons à élever la valeur du couplage non réduit à la puissance

$$\frac{2^{4m}-1}{\#E_2} = \frac{2^{4m}-1}{2^m+\nu 2^{(m+1)/2}+1} = \left(2^{2m}-1\right)\left(2^m-\nu 2^{(m+1)/2}+1\right).$$

L'Algorithme 7.4 détaille ce calcul ainsi que son coût en terme d'opérations sur le corps de base. Nous présentons ici les points-clés qui nous permettent d'obtenir un tel algorithme.

Algorithme 7.4 Calcul de l'exponentiation finale pour le couplage η_T sur la courbe E_2.

Entrées : $U = u_0 + u_1 s + u_2 t + u_3 st \in \mathbb{F}^*_{2^{4m}}$
Sortie : $U^{(2^{2m}-1)(2^m-\nu\cdot 2^{(m+1)/2}+1)}$

1. $m_0 \leftarrow u_0^2$; $m_1 \leftarrow u_1^2$; $m_2 \leftarrow u_2^2$; $m_3 \leftarrow u_3^2$ — (4 f)
2. $T_0 \leftarrow (m_0 + m_1) + m_1 s$; $T_1 \leftarrow (m_2 + m_3) + m_3 s$ — (2 a)
3. $T_2 \leftarrow m_3 + m_2 s$; $T_3 \leftarrow (u_0 + u_1 s)\cdot(u_2 + u_3 s)$ — (3 m + 4 a)
4. $T_4 \leftarrow T_0 + T_2$; $D \leftarrow T_3 + T_4$ — (4 a)
5. $D \leftarrow D^{-1}$ — (3 m + 4 a + 1 f + 1 i)
6. $T_5 \leftarrow T_1 \cdot D$; $T_6 \leftarrow T_4 \cdot D$ — (6 m + 8 a)
7. $V_0 \leftarrow T_5 + T_6$; $V_1 \leftarrow T_5$; $V \leftarrow V_0 + V_1 t$ — (2 a)
8. **if** $\nu = -1$ **then**
9. $\quad W_0 \leftarrow V_0$
10. **else**
11. $\quad W_0 \leftarrow T_6$
12. **end if**
13. $W_1 \leftarrow T_5$; $W \leftarrow W_0 + W_1 t$
14. $V \leftarrow V^{2^m+1}$ — (5 m + 14 a + 2 f)
15. $W \leftarrow W^{2^{\frac{m+1}{2}}}$ — (6 a + (2m + 2) f)
16. **return** $V \cdot W$ — (1 M)

La première étape du calcul (lignes 1 à 7) consiste à calculer $V = U^{2^{2m}-1}$: nous posons $U = U_0 + U_1 t$ avec $U_0, U_1 \in \mathbb{F}_{2^{2m}}$,

$$\begin{aligned} U^{2^{2m}-1} &= \frac{U_0 + U_1 + U_1 t}{U_0 + U_1 t} \\ &= \frac{(U_0 + U_1 + U_1 t)^2}{(U_0 + U_1 t)(U_0 + U_1 + U_1 t)} \\ &= \frac{U_0^2 + U_1^2 + U_1^2 s + U_1^2 t}{U_0^2 + U_0 U_1 + U_1^2 s}. \end{aligned}$$

Ce calcul est grandement simplifié par le fait que l'élévation à la puissance 2^{2m} correspond à l'endomorphisme de Frobenius laissant $\mathbb{F}_{2^{2m}}$ invariant. Notons cependant que nous devons effectuer l'inversion sur $\mathbb{F}_{2^{2m}}$ de $D = U_0^2 + U_0 U_1 + U_1^2 s$ (ligne 5) pour obtenir V. Celle-ci peut être ramenée à une inversion sur $\mathbb{F}_{2^m}$ ainsi que 3 multiplications, 4 additions et une application du Frobenius π_2 grâce à l'application du petit théorème de Fermat (algorithme de Itoh & Tsujii, Section 8.1.4).

Pour la suite de l'algorithme, nous allons calculer V^{2^m+1} d'une part et $W = V^{-\nu \cdot 2^{\frac{m+1}{2}}}$ d'autre part avant d'en prendre le produit pour obtenir le résultat final (ligne 16). Ces deux opérations vont utiliser le fait que V appartient à une structure algébrique particulière à l'arithmétique efficace : le tore $T_2(\mathbb{F}_{2^{2m}}) = \{X_0 + X_1 t \mid X_0^2 + X_0 X_1 + X_1^2 s = 1\}$. En effet, l'élévation à la puissance $2^{2m} - 1$ assure l'appartenance à ce sous-groupe multiplicatif d'ordre $2^{2m} + 1$.

La première partie du calcul consiste à appliquer le Frobenius π_{2^m} à V, ce qui n'implique que des additions sur $\mathbb{F}_{2^m}$, puis à multiplier le résultat par V. Cette opération est plus simple qu'une multiplication complète sur $\mathbb{F}_{2^{4m}}$ grâce à la structure du tore. L'Algorithme 7.5 donne les formules à appliquer pour ce calcul.

Algorithme 7.5 Calcul de V^{2^m+1} dans le tore $T_2(\mathbb{F}_{2^{2m}})$.

Entrées : $V = v_0 + v_1 s + v_2 t + v_3 st \in T_2(\mathbb{F}_{2^{2m}})$
Sortie : V^{2^m+1}

1.	$m_0 \leftarrow (v_0 + v_1) \cdot (v_2 + v_3)$	(1 m + 2 a)
2.	$m_1 \leftarrow v_0 \cdot v_1$	(1 m)
3.	$m_2 \leftarrow v_0 \cdot v_3$	(1 m)
4.	$m_3 \leftarrow v_1 \cdot v_2$	(1 m)
5.	$m_4 \leftarrow v_2 \cdot v_3$	(1 m)
6.	$t_0 \leftarrow (v_0 + v_1)^2$	(1 a + 1 f)
7.	$t_1 \leftarrow (v_2 + v_3)^2$	(1 a + 1 f)
8.	**if** $\frac{m+1}{2} \equiv 1 \pmod 2$ **then**	
9.	$\quad v'_0 \leftarrow m_1 + m_2 + m_3 + t_0$	(3 a)
10.	$\quad v'_1 \leftarrow m_0 + m_2 + m_4 + t_1$	(3 a)
11.	$\quad v'_2 \leftarrow m_2 + m_3 + m_4 + t_1$	(3 a)
12.	**else**	
13.	$\quad v'_0 \leftarrow m_0 + m_1 + m_2 + t_0$	(3 a)
14.	$\quad v'_1 \leftarrow m_0 + m_3 + m_4 + t_1$	(3 a)
15.	$\quad v'_2 \leftarrow m_2 + m_3$	(1 a)
16.	**end if**	
17.	$v'_3 \leftarrow m_4 + t_1$	(1 a)
18.	**return** $v'_0 + v'_1 s + v'_2 t + v'_3 st$	

Pour le calcul de $V^{-\nu \cdot 2^{\frac{m+1}{2}}}$ nous avons d'abord à éventuellement inverser V. Pour ce faire, nous utilisons la structure de tore :

$$(V_0 + tV_1)(V_0 + V_1 + tV_1) = V_0^2 + V_0 V_1 + V^2 s = 1.$$

Ainsi, ce calcul ne demande que 2 additions supplémentaires sur le corps de base $\mathbb{F}_{2^m}$ s'il est nécessaire ($\nu = 1$) (lignes 8 à 12, Algorithme 7.4).

Il nous reste alors à élever $W = V^{-\nu}$ à la puissance $2^{\frac{m+1}{2}}$, c'est-à-dire appliquer $\frac{m+1}{2}$ fois le Frobenius π_2 à W. Comme le Frobenius est une opération linéaire, nous avons la formule suivante pour $W = (w_0 + w_1 s) + (w_2 + w_3 s)t$ avec $w_j \in \mathbb{F}_{2^m}$:

$$W^{2^i} = ((w'_0 + \gamma_1 w'_1 + \gamma_2 w'_2 + \gamma_3 w'_3) + (w'_1 + \gamma_1 w'_2 + \gamma_2 w'_3)s) + ((w'_2 + \gamma_1 w'_3) + w'_3 s)t,$$

avec $w'_j = w_j^{2^i}$ et $\gamma_1 = i \pmod 2$, $\gamma_2 = \left\lfloor \frac{i}{2} \right\rfloor \pmod 2$, $\gamma_3 = 1$ si $i \equiv 1 \pmod 4$ et $\gamma_3 = 0$ sinon. Ainsi, ce calcul demande $2m + 2$ carrés et quelques additions (au plus 6).

TABLE 7.6 – Coût arithmétique des différents algorithmes pour le couplage Eta optimal sur E_2.

(a) Décompte avec les opérations dans les extensions.

Algorithme	Multiplication	Addition	Frobenius	Inv.
Naïf (Alg. 7.1)	$\frac{m+1}{2}$ M′ + $\frac{m+1}{2}$ m	$(2m+4)$ a + $(m+1+\alpha+b)$ a′	$\frac{m-1}{2}$ F + $(2m+2)$ f	–
Renversé (Alg. 7.2)	$\frac{m-1}{2}$ M′ + 1 M″ + $\frac{m+1}{2}$ m	$(3\frac{m+1}{2}+1)$ a + $((2\alpha+\beta)\frac{m+1}{2}+\bar{\delta})$ a′	m f + $(m-1)$ f′	–
Renversé sans $\sqrt{\cdot}$ (Alg. 7.3)	$\frac{m-1}{2}$ M′ + 1 M″ + $\frac{m+1}{2}$ m	$(2m+3)$ a + $(\frac{m+1}{2}+\bar{\delta}+2+b)$ a′	$\frac{m-1}{2}$ F + $(4\frac{m+1}{2}-1)$ f	–
Exp. finale (Alg. 7.4)	1 M + 17 m	44 a	$(2m+9)$ f	1 i

(b) Décompte des opérations ramenées dans le corps de base.

Algorithme	Multiplication	Addition	Frobenius	Inv.
Naïf (Alg. 7.1)	$7\frac{m+1}{2}$ m	$(11m+9)$ a + $(m+1+\alpha+b)$ a′	$4m$ f	–
Renversé (Alg. 7.2)	$(7\frac{m+1}{2}-4)$ m	$(17\frac{m+1}{2}-8)$ a + $((2\alpha+\beta)\frac{m+1}{2}+\bar{\delta}+2)$ a′	$(m+1)$ f + $(m-1)$ f	–
Renversé sans $\sqrt{\cdot}$ (Alg. 7.3)	$(7\frac{m+1}{2}-4)$ m	$11m$ a + $(\frac{m+1}{2}+\bar{\delta}+4+b)$ a′	$4m$ f	–
Exp. finale (Alg. 7.4)	26 m	64 a	$(2m+9)$ f	1 i

Nous donnons maintenant le coût arithmétique des différentes approches algorithmiques que nous avons envisagées dans la Table 7.6. Cette table donne deux versions des coûts : l'une où les opérations dans $\mathbb{F}_{2^{4m}}$ ont été comptées séparément, l'autre où ces opérations ont été implémentées en séquences d'opérations sur le corps de base, optimales en nombre de multiplications.

7.2 Courbes elliptiques en caractéristique 3

La deuxième courbe qui a retenu notre attention est :

$$E_3/\mathbb{F}_3 : y^2 = x^3 - x + b, \text{ avec } b \in \{-1, 1\}.$$

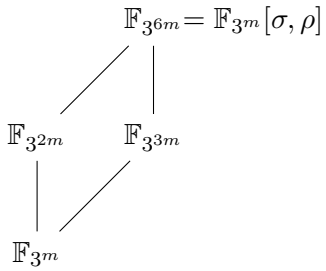
Elle est décrite à la Section 4.3.1 § 2. C'est une courbe de degré de plongement $k = 6$. Nous considérerons alors la courbe sur un corps de degré d'extension m premier avec 6, où

elle est de cardinal :

$$\#E_3(\mathbb{F}_{3^m}) = 3^m + \mu 3^{\frac{m+1}{2}} + 1, \text{ où } \mu = \begin{cases} b & \text{si } m \equiv \pm 1 \pmod{12}, \\ -b & \text{si } m \equiv \pm 5 \pmod{12}. \end{cases}$$

7.2.1 *Distortion map* et endomorphismes de E_3

À l'instar de la caractéristique 2, nous commençons par chercher une *distortion map* que Silverman nous fournit dans sa preuve de [Sil86, Proposition A.1.2] [2]. En fixant la tour d'extensions suivante



avec $\sigma \in \mathbb{F}_{3^2}$ et $\rho \in \mathbb{F}_{3^3}$ tels que $\sigma^2 + 1 = 0$ et $\rho^3 - \rho - b = 0$, nous pouvons alors considérer la *distortion map*

$$\psi : (x, y) \longmapsto (\rho - x, \sigma y).$$

Vérifions qu'elle définit bien le groupe $\mathbb{G}_{\pi_{3^m}, 3^m}$ une fois appliquée à $\mathbb{G}_{\pi_{3^m},1}$. Pour tout $P \in \mathbb{G}_{\pi_{3^m},1}$, $P \neq P_\infty$, nous avons

$$\begin{aligned} \pi_{3^m}(\psi(P)) &= (\rho^{3^m} - x_P, & \sigma^{3^m} y_P) \\ &= (\rho + mb - x_P, & -\sigma y_P). \end{aligned}$$

Étant donnée la formule du triplement itéré pour E_3

$$[3^i](x, y) = (x^{3^{2i}} - ib, (-1)^i y^{3^{2i}}), \tag{7.2}$$

nous pouvons finir la vérification de $\mathbb{G}_{\pi_{3^m}, 3^m} = \psi(G_{\pi_{3^m},1})$:

$$\begin{aligned} [3^m]\psi(P) &= (\rho^{3^{2m}} - x_P - mb, & -\sigma^{3^{2m}} y_P) \\ &= (\rho - mb - x_P - mb, & -\sigma y_P) \\ &= \pi_{3^m}(\psi(P)). \end{aligned}$$

Comme nous sommes en genre 1, le Frobenius π_{3^m} et le Verschiebung $\hat{\pi}_{3^m}$ agissent directement sur la courbe. Nous avons alors :

$$\hat{\pi}_{3^m} = \gamma_{E_3} \circ \pi_{3^m} \text{ où } \gamma_{E_3}(x, y) = (x - mb, -y).$$

Comme γ_{E_3} est un automorphisme et que le Frobenius est purement inséparable, le Verschiebung l'est aussi. Nous pourrons donc utiliser indifféremment les versions Ate et Eta.

2. Silverman ne fournit qu'une caractérisation des automorphismes de E_3 dont nous déduisons par le calcul le seul automorphisme exactement $\mathbb{F}_{3^{6m}}$-rationnel.

7.2.2 Construction de couplages optimaux

Soit ℓ un diviseur premier de $\#E_3(\mathbb{F}_{3^m})$. Ainsi qu'en caractéristique 2, nous choisissons $L = \#E_3(\mathbb{F}_{3^m})$ et nous aurons l'exposant final suivant :

$$\frac{q^k-1}{L} = \frac{3^{6m}-1}{3^m + \mu 3^{\frac{m+1}{2}} + 1} = (3^{3m}-1)(3^m+1)(3^m - \mu 3^{\frac{m+1}{2}} + 1).$$

Cet exposant bénéficie des mêmes avantages que précédemment : une structure permettant un algorithme efficace d'exponentiation et l'élimination des verticales.

Nous cherchons maintenant un couplage optimal, c'est-à-dire un vecteur court dans le réseau euclidien donné par les lignes de la matrice

$$\begin{pmatrix} 3^m + \mu 3^{\frac{m+1}{2}} + 1 & 0 \\ -3^m & 1 \end{pmatrix}.$$

Le vecteur court qui nous intéressera est alors $(3^{\frac{m+1}{2}} + \mu, \mu)$. Nous avons donc une construction similaire à la caractéristique 2 et devrons ainsi calculer un couplage reposant sur la fonction de Miller $f_{3^{\frac{m+1}{2}}+\mu,P}$, comme le couplage η_T introduit par Barreto *et al.* dans [Bar+07].

7.2.3 Algorithme de Miller et arithmétique de courbe

Duursma et Lee ont montré que la courbe E_3 permet d'obtenir des formules très efficaces pour le triplement de point [DL03]. Celles-ci nous permettent d'envisager un algorithme de Miller en base 3 et d'exploiter la forme creuse en base 3 du paramètre de la fonction à calculer. Nous avons alors une expression de cette fonction très similaire à la caractéristique 2 :

$$f_{3^{\frac{m+1}{2}}+\mu,P} = \left(\prod_{i=0}^{\frac{m-1}{2}} f_{3,[3^i]P}^{3^{\frac{m-1}{2}-i}} \right) \cdot g_{\left[3^{\frac{m+1}{2}}\right]P,[\mu]P}.$$

Nous avons déjà donné la formule de triplement (7.2), il ne nous reste plus que l'équation du produit de droites associées (une fois la verticale éliminée) :

$$f_{3,P}(X,Y) = y_P^3 Y - (x_P^3 - X + b)^2.$$

Nous appliquons alors la *distortion map* à P dans le cas Ate et au point d'évaluation pour Eta :

$$\begin{aligned} f_{3,\psi(P)}(Q) &= -\rho^2 - (x_P^3 + x_Q + b)\rho - y_P^3 y_Q \sigma - (x_P^3 + x_Q + b)^2, \\ f_{3,P}(\psi(Q)) &= -\rho^2 - (x_P^3 + x_Q + b)\rho + y_P^3 y_Q \sigma - (x_P^3 + x_Q + b)^2. \end{aligned}$$

Ici encore, les versions Ate et Eta sont indifférentiables : les formules des fonctions de Miller que nous accumulons ne diffèrent que par un signe. Nous choisirons encore une fois les versions Eta, par cohérence avec le reste de la littérature. Nous donnons alors l'évaluation de la droite finale dans ce cas :

$$g_{\left[3^{\frac{m+1}{2}}\right]P,[\mu]P}(\psi(Q)) = -\lambda y_P \rho + y_Q \sigma + \lambda y_P (x_P + x_Q - \nu),$$

où $\lambda = (-1)^{\frac{m+1}{2}}$ et $\nu = \mu\lambda$, ce qui donne

$$\lambda = \begin{cases} +1 & \text{si } m \equiv 7 \text{ ou } 11 \pmod{12}, \\ -1 & \text{si } m \equiv 1 \text{ ou } 5 \pmod{12}, \end{cases}$$
$$\nu = \begin{cases} +b & \text{si } m \equiv 5 \text{ ou } 11 \pmod{12}, \\ -b & \text{si } m \equiv 1 \text{ ou } 7 \pmod{12}. \end{cases}$$

Nous introduisons alors les notations suivantes afin de compter les opérations dans les algorithmes que nous développons dans la suite.

a	addition dans $\mathbb{F}_{3^m}$
a′	ajout de ± 1
f	Frobenius dans $\mathbb{F}_{3^m}$ $(x \mapsto x^3)$
f′	Frobenius inverse dans $\mathbb{F}_{3^m}$ $(x \mapsto \sqrt[3]{x})$
F	Frobenius dans $\mathbb{F}_{3^{3m}}$
m	multiplication dans $\mathbb{F}_{3^m}$
M	multiplication dans $\mathbb{F}_{3^{6m}}$
M′	multiplication « creux par plein » $(a_2^2 + a_1\sigma + a_2\rho)(b_0 + b_1\sigma + b_2\rho + b_3\sigma\rho + b_4\rho^2 + b_5\sigma\rho^2)$
M″	multiplication « creux par creux » $(a_2^2 + a_1\sigma + a_2\rho + \rho^2)(b_0 + b_1\sigma + b_2\rho + \rho^2)$

Nous pouvons ainsi présenter une première version de l'algorithme de couplage η_T sur E_3 (Algorithme 7.7).

7.2.4 Raffinements algorithmiques

Une étude extensive de cet algorithme a été réalisée par Beuchat *et al.* dans [Beu+08b] ; elle permet d'optimiser ainsi l'organisation des calculs afin d'améliorer son coût en termes d'opérations arithmétiques.

Les mêmes optimisations qu'en caractéristique 2 (renversement de la boucle de Miller, utilisation ou non du Frobenius inverse) sont applicables. Cependant, la caractéristique 3 permet une autre optimisation. Une fois la nécessité d'appliquer le Frobenius sur l'accumulateur à chaque itération éliminée, il est possible de dérouler la boucle : en groupant par deux le calcul de fonctions de Miller, nous avons alors deux éléments creux à multiplier à l'accumulateur. Il devient alors intéressant de multiplier d'abord entre eux les évaluations de fonctions de Miller car leur produit donne encore une fois un élément creux. Ainsi, il est possible d'économiser 3 produits sur $\mathbb{F}_{3^m}$ toutes les 2 itérations. Cette possibilité n'existait pas en caractéristique 2 car le produit de deux éléments creux issus des droites doublements n'est pas un élément creux de $\mathbb{F}_{2^{4m}}$.

Devant la variété des algorithmes que ces raffinements peuvent produire, nous invitons le lecteur à consulter [Beu+08b] pour obtenir chaque version dans le détail. Comme chacune de ces variantes présente un intérêt différent selon l'architecture ciblée (ordonnancement, coût relatif des opérations arithmétiques, disponibilité du Frobenius inverse, etc.), nous donnons tout de même une table résumant les coûts arithmétiques de celles-ci (Table 7.8) ainsi que les deux algorithmes que nous avons utilisés pour nos implémentations parallèles [Beu+09a ; Beu+11] et compactes [Est10].

Algorithme 7.7 Algorithme pour le couplage Eta optimal sur E_3.

Entrées : $P = (x_P, y_P)$, $Q = (x_Q, y_Q) \in E_3(\mathbb{F}_{3^m})[\ell]$.
Sortie : $\eta_T(P,Q) = \eta_{\left(3^{\frac{m+1}{2}}+\mu,\mu\right)}(P,Q) \in \mathbb{F}^*_{3^{6m}}$.

1. $x_Q \leftarrow x_Q + b$ (1 a′)
2. $y_V \leftarrow y_P^3$ (1 f)
3. $g_1 \leftarrow y_V y_Q$ (1 m)
4. $x_V \leftarrow x_Q^3$ (1 f)
5. $g_2 \leftarrow x_V + x_Q$ (1 a)
6. $g_0 \leftarrow g_2^2$ (1 m)
7. $F \leftarrow -g_0 + g_1\sigma - g_2\rho - \rho^2$
8. **for** $i = \frac{m-1}{2}$ **to** 2 **do**
9. $\quad F \leftarrow F^3$ (1 F)
10. $\quad y_V \leftarrow y_V^3$ (1 f)
11. $\quad g_1 \leftarrow y_V y_Q$ (1 m)
12. $\quad x_V \leftarrow x_V^3$ (1 f)
13. $\quad g_2 \leftarrow x_V + x_Q$ (1 a)
14. $\quad F \leftarrow -F \cdot (g_2^2 - g_1\sigma + g_2\rho + \rho^2)$ (1 M′ + 8 a)
15. $\quad x_V \leftarrow x_V^3 - b$ (1 f + 1 a′)
16. $\quad y_V \leftarrow -y_V^3$ (1 f)
17. **end for**
18. $g_0 \leftarrow \lambda y_P \cdot (x_P + x_Q - \nu)$ (1 m + 1 a)
19. $F \leftarrow F \cdot (g_0 + y_Q\sigma - \lambda y_P \rho)$ (1 M′)
20. **return** $F^{\frac{3^{6m}-1}{L}}$

Pour cette première implémentation [Beu+09a ; Beu+11], nous voulions utiliser un multiplieur pipeliné. Les contraintes d'ordonnancement étaient donc très fortes et nous ne pouvions nous permettre d'utiliser le déroulement de boucle car elle complexifiait notablement le corps de la boucle et son ordonnancement. Cependant, nous étions obligés de dupliquer les opérateurs chargés de la mise à jour des coordonnées des points. Il n'était ainsi pas dommageable de rajouter le support du Frobenius inverse. Nous avons donc utilisé l'Algorithme 7.9.

Algorithme 7.9 Algorithme Eta optimal pour E_3 avec renversement de boucle [Beu+08b, Algorithme 3].

Entrées : $P = (x_P, y_P)$, $Q = (x_Q, y_Q) \in E_3(\mathbb{F}_{3^m})[\ell]$.
Sortie : $\eta_T(P,Q) = \eta_{\left(3^{\frac{m+1}{2}}+\mu,\mu\right)}(P,Q) \in \mathbb{F}^*_{3^{6m}}$.

1. $x_V \leftarrow x_P - \nu b$; $y_V \leftarrow -\mu y_P$ (1 a′)
2. $t \leftarrow x_V + x_Q$ (1 a)
3. $F \leftarrow -(\lambda y_V + y_Q\sigma - \lambda y_P\rho) \cdot (t^2 + \lambda y_P y_Q \sigma + t\rho + \rho^2)$ (6 m + 6 a + 1 f)
4. **for** $j \leftarrow 1$ **to** $\frac{m-1}{2}$ **do**
5. $\quad x_V \leftarrow \sqrt[3]{x_V}$; $y_V \leftarrow \sqrt[3]{y_V}$ (2 f′)
6. $\quad x_Q \leftarrow x_Q^3$; $y_Q \leftarrow y_Q^3$ (2 f)
7. $\quad t \leftarrow x_V + x_Q$ (1 a)
8. $\quad u \leftarrow y_V y_Q$ (1 m)
9. $\quad G \leftarrow -t^2 - \lambda u\sigma - t\rho - \rho^2$
10. $\quad F \leftarrow F \cdot G$ (1 M′ + 8 a)
11. **end for**
12. **return** $F^{\frac{3^{6m}-1}{L}}$

A contrario, l'implémentation compacte [Est10] consiste à utiliser un coprocesseur travaillant sur le corps de base $\mathbb{F}_{3^m}$. Dans ce cas, le support du Frobenius inverse ($x \mapsto \sqrt[3]{x}$) aurait représenté un surcoût matériel non négligeable. Cependant, le flot d'exécution est beaucoup plus souple et nous avons donc pu bénéficier du déroulement de boucle et utilisons l'Algorithme 7.10.

7.2.5 Exponentiation finale

Le cas de la caractéristique 3 utilise des méthodes similaires à celles de la caractéristique 2. Nous donnons ici l'algorithme que nous avons utilisé (Algorithme 7.11). Il s'agit ici d'élever la valeur U du couplage non réduit à la puissance

$$\frac{3^{6m}-1}{\#E_3} = \frac{3^{6m}-1}{3^m + \mu \cdot 3^{(m+1)/2} + 1} = \left(3^{3m}-1\right)(3^m+1)\left(3^m - \mu 3^{(m+1)/2} + 1\right).$$

L'exposant est ainsi très similaire au cas précédent et nous commençons également par calculer $V = U^{3^{3m}-1}$. En posant $U = U_0 + U_1\sigma$ avec $U_0, U_1 \in \mathbb{F}_{3^{3m}}$, nous avons

$$V = \frac{U_0^2 - U_1^2 + U_0U_1\sigma}{U_0^2 + U_1^2},$$

dont nous déduisons les lignes 1 à 6 de l'Algorithme 7.11. Ce calcul implique une inversion (ligne 5) sur $\mathbb{F}_{3^{3m}}$ que nous réalisons par une application de l'algorithme d'Itoh & Tsujii.

TABLE 7.8 – Coûts arithmétiques de différents algorithmes pour le couplage Eta optimal sur E_3.

(a) Décompte avec les opérations dans les extensions.

Algorithme	Multiplication	Addition	Frobenius	Inv.
Direct sans $\sqrt[3]{\cdot}$ (Alg. 7.7)	$\frac{m-1}{2}\,\mathsf{M}' + (\frac{m-1}{2}+2)\,\mathsf{m}$	$(9\frac{m-1}{2}-7)\,\mathsf{a} + \frac{m-1}{2}\,\mathsf{a}'$	$(\frac{m-1}{2}-1)\,\mathsf{F} + (2m-2)\,\mathsf{f}$	–
Direct avec $\sqrt[3]{\cdot}$ [Beu+08b, Alg. 2]			$(m-1)\,\mathsf{f} + (m-1)\,\mathsf{f}'$	
Renversé avec $\sqrt[3]{\cdot}$ (Alg. 7.9)	$\frac{m-1}{2}\,\mathsf{M}' + (\frac{m-1}{2}+6)\,\mathsf{m}$	$(9\frac{m-1}{2}+7)\,\mathsf{a} + 1\,\mathsf{a}'$	$m\,\mathsf{f} + (m-1)\,\mathsf{f}'$	–
Renversé sans $\sqrt[3]{\cdot}$ [Beu+08b, Alg. 2]			$(\frac{m-1}{2}-1)\,\mathsf{F} + (2m+3)\,\mathsf{f}$	
Déroulé ($\frac{m-1}{2}$ pair) (Alg. 7.10)	$\frac{m-1}{4}\,(\mathsf{M}+\mathsf{M}'') + (3\frac{m-1}{4}+6)\,\mathsf{m}$	$(3\frac{m-1}{4}+7)\,\mathsf{a} + m\,\mathsf{a}'$	$\frac{m-1}{2}\,\mathsf{F} + (5\frac{m-1}{2}+3)\,\mathsf{f}$	–
Exp. finale (Alg. 7.11)	$1\,\mathsf{M} + 58\,\mathsf{m}$	$111\,\mathsf{a}$	$(3m+3)\,\mathsf{f}$	$1\,\mathsf{i}$

(b) Décompte des opérations ramenées dans le corps de base.

Algorithme	Multiplication	Addition	Frobenius	Inv.
Direct sans $\sqrt[3]{\cdot}$ (Alg. 7.7)	$(7m+2)\,\mathsf{m}$	$(33m+11)\,\mathsf{a} + \frac{m-1}{2}\,\mathsf{a}'$	$(5m-7)\,\mathsf{f}$	–
Direct avec $\sqrt[3]{\cdot}$ [Beu+08b, Alg. 2]		$(59\frac{m-1}{2}+15)\,\mathsf{a} + \frac{m-1}{2}\,\mathsf{a}'$	$(m-1)\,\mathsf{f} + (m-1)\mathsf{f}'$	
Renversé avec $\sqrt[3]{\cdot}$ (Alg. 7.9)	$(7m-1)\,\mathsf{m}$	$(30m-23)\,\mathsf{a} + 1\,\mathsf{a}'$	$m\,\mathsf{f} + (m-1)\,\mathsf{f}'$	–
Renversé sans $\sqrt[3]{\cdot}$ [Beu+08b, Alg. 2]		$(67\frac{m-1}{2}+7)\,\mathsf{a} + 1\,\mathsf{a}'$	$(5m-2)\,\mathsf{f}$	
Déroulé ($\frac{m-1}{2}$ pair) (Alg. 7.10)	$(25\frac{m-1}{4}+6)\,\mathsf{m}$	$(107\frac{m-1}{4}+11)\,\mathsf{a}$	$(11\frac{m-1}{2}+3)\,\mathsf{f}$	–
Exp. finale (Alg. 7.11)	$73\,\mathsf{m}$	$178\,\mathsf{a}$	$(3m+3)\,\mathsf{f}$	$1\,\mathsf{i}$

Algorithme 7.10 Algorithme Eta optimal pour E_3 avec déroulement de boucle [Beu+08b, Algorithme 5] (Cas $\frac{m-1}{2} \equiv 0 \pmod 2$).

Entrées : $P = (x_P, y_P)$, $Q = (x_Q, y_Q) \in E_3(\mathbb{F}_{3^m})[\ell]$.
Sortie : $\eta_T(P,Q) = \eta_{\left(3^{\frac{m+1}{2}}+\mu,\mu\right)}(P,Q) \in \mathbb{F}^*_{3^{6m}}$.

1. $x_V \leftarrow x_P + b\,;\ y_V \leftarrow -\mu b y_P$ — $(1\ a')$
2. $x_Q \leftarrow x_Q^3\,;\ y_Q \leftarrow y_Q^3$ — $(2\ \mathsf{f})$
3. $t \leftarrow x_V + x_Q$ — $(1\ a)$
4. $R \leftarrow (\lambda y_V - \lambda y_Q \sigma - \lambda y_P \rho) \cdot (-t^2 + y_P y_Q \sigma - t\rho - \rho^2)$ — $(6\ \mathsf{m} + 6\ \mathsf{a} + 1\ \mathsf{f})$
5. **for** $j \leftarrow 1$ **to** $\frac{m-1}{4}$ **do**
6. $\quad R \leftarrow R^9$ — $(2\ \mathsf{F})$
7. $\quad x_Q \leftarrow x_Q^9 - b\,;\ y_Q \leftarrow y_Q^9$ — $(1\ \mathsf{a}' + 4\ \mathsf{f})$
8. $\quad t \leftarrow x_V + x_Q$ — $(1\ \mathsf{a})$
9. $\quad u \leftarrow y_V y_Q$ — $(1\ \mathsf{m})$
10. $\quad S \leftarrow (-t^2 - u\sigma - t\rho - \rho^2)^3$ — $(1\ \mathsf{m} + 1\ \mathsf{a} + 2\ \mathsf{a}' + 2\ \mathsf{f})$
11. $\quad x_Q \leftarrow x_Q^9 - b\,;\ y_Q \leftarrow y_Q^9$ — $(1\ \mathsf{a}' + 4\ \mathsf{f})$
12. $\quad t \leftarrow x_V + x_Q$ — $(1\ \mathsf{a})$
13. $\quad u \leftarrow y_V y_Q$ — $(1\ \mathsf{m})$
14. $\quad S' \leftarrow -t^2 + u\sigma - t\rho - \rho^2$
15. $\quad S \leftarrow S \cdot S'$ — $(1\ \mathsf{M}'')$
16. $\quad R \leftarrow R \cdot S$ — $(1\ \mathsf{M})$
17. **end for**
18. **return** $R^{\frac{3^{6m}-1}{L}}$

Algorithme 7.11 Calcul de l'exponentiation finale pour le couplage η_T sur E_3.

Entrées : $U = u_0 + u_1\sigma + u_2\rho + u_3\sigma\rho + u_4\rho^2 + u_5\sigma\rho^2 \in \mathbb{F}^*_{3^{6m}}$
Sortie : $U^{(3^{3m}-1)(3^m+1)(3^m-\mu 3^{(m+1)/2}+1)}$

1. $m_0 \leftarrow (u_0 + u_2\rho + u_4\rho^2)^2$ — $(5\ \mathsf{m} + 7\ \mathsf{a})$
2. $m_1 \leftarrow (u_1 + u_3\rho + u_5\rho^2)^2$ — $(5\ \mathsf{m} + 7\ \mathsf{a})$
3. $m_2 \leftarrow (u_0 + u_2\rho + u_4\rho^2) \cdot (u_1 + u_3\rho + u_5\rho^2)$ — $(6\ \mathsf{m} + 12\ \mathsf{a})$
4. $a_0 \leftarrow m_0 - m_1\,;\ a_1 \leftarrow m_0 + m_1$ — $(6\ \mathsf{a})$
5. $i \leftarrow a_1^{-1}$ — $(12\ \mathsf{m} + 11\ \mathsf{a} + 1\ \mathsf{i})$
6. $V_0 \leftarrow a_0 \cdot i\,;\ V_1 \leftarrow m_2 \cdot i\,;\ V \leftarrow V_0 + V_1\sigma$ — $(12\ \mathsf{m} + 24\ \mathsf{a})$
7. $V' \leftarrow V^{3^m+1} = V'_0 + V'_1\sigma$ — $(9\ \mathsf{m} + 19\ \mathsf{a})$
8. $W_0 \leftarrow V'_0\,;\ W_1 \leftarrow -\mu V'_1\,;\ W \leftarrow W_0 + W_1\sigma$
9. $V' \leftarrow V'^{3^m+1}$ — $(9\ \mathsf{m} + 19\ \mathsf{a})$
10. $W \leftarrow W^{3^{\frac{m+1}{2}}}$ — $(6\ \mathsf{a} + (3m+3)\ \mathsf{f})$
11. **return** $V' \cdot W$ — $(1\ \mathsf{M})$

Cet élément fait alors partie du tore $T_2(\mathbb{F}_{3^{3m}}) = \{X_0 + X_1\sigma \mid X_0^2 + X_1^2 = 1\}$ dont nous utiliserons l'arithmétique efficace.

Contrairement au cas précédent nous devons élever V à la puissance $3^m + 1$ avant de pouvoir reprendre une stratégie similaire ($V' \leftarrow V^{3^m+1}$). Ce calcul est facilité par l'utilisation du tore et nous le décrivons dans l'Algorithme 7.12.

Algorithme 7.12 Calcul de V^{3^m+1} dans le tore $T_2(\mathbb{F}_{3^{3m}})$.

Entrées : $V = v_0 + v_1\sigma + v_2\rho + v_3\sigma\rho + v_4\rho^2 + v_5\sigma\rho^2 \in T_2(\mathbb{F}_{3^{3m}})$
Sortie : V^{3^m+1}

1. $a_0 \leftarrow v_0 + v_1$; $a_1 \leftarrow v_2 + v_3$; $a_2 \leftarrow v_4 - v_5$ (3 a)
2. $m_0 \leftarrow v_0 \cdot v_4$; $m_1 \leftarrow v_1 \cdot v_5$; $m_2 \leftarrow v_2 \cdot v_4$ (3 m)
3. $m_3 \leftarrow v_3 \cdot v_5$; $m_4 \leftarrow a_0 \cdot a_2$; $m_5 \leftarrow v_1 \cdot v_2$ (3 m)
4. $m_6 \leftarrow v_0 \cdot v_3$; $m_7 \leftarrow a_0 \cdot a_1$; $m_8 \leftarrow a_1 \cdot a_2$ (3 m)
5. $a_3 \leftarrow m_5 + m_6 - m_7$; $a_4 \leftarrow -m_2 - m_3$ (3 a)
6. $a_5 \leftarrow -m_2 + m_3$; $a_6 \leftarrow -m_0 + m_1 + m_4$ (3 a)
7. **if** $m \equiv 1 \pmod 6$ **then**
8. $\quad v_0 \leftarrow 1 + m_0 + m_1 + ba_4$ (3 a)
9. $\quad v_1 \leftarrow bm_5 - bm_6 + a_6$ (2 a)
10. $\quad v_2 \leftarrow -a_3 + a_4$ (1 a)
11. $\quad v_3 \leftarrow m_8 + a_5 - ba_6$ (2 a)
12. $\quad v_4 \leftarrow -ba_3 - ba_4$ (1 a)
13. $\quad v_5 \leftarrow bm_8 + ba_5$ (1 a)
14. **else**
15. $\quad v_0 \leftarrow 1 + m_0 + m_1 - ba_4$ (3 a)
16. $\quad v_1 \leftarrow -bm_5 + bm_6 + a_6$ (2 a)
17. $\quad v_2 \leftarrow a_3$
18. $\quad v_3 \leftarrow m8 + a_5 + ba_6$ (2 a)
19. $\quad v_4 \leftarrow -ba_3 - ba_4$ (1 a)
20. $\quad v_5 \leftarrow -bm_8 - ba_5$ (1 a)
21. **end if**
22. **return** $v_0 + v_1\sigma + v_2\rho + v_3\sigma\rho + v_4\rho^2 + v_5\sigma\rho^2$

À l'instar de la caractéristique 2, le calcul se sépare en deux : V'^{3^m+1} et $V'^{-\mu 3^{\frac{m+1}{2}}}$. Nous venons de décrire l'algorithme à appliquer à nouveau pour la première partie. Pour la seconde nous devons éventuellement inverser V' ($\mu = 1$), ce calcul est simple grâce au fait que V' est également dans le tore $T_2(\mathbb{F}_{3^{3m}})$:

$$V'^{-1} = \frac{1}{V'_0 + V'_1\sigma} = \frac{V'_0 - V'_1\sigma}{V'^2_0 + V'^2_1} = V'_0 - V'_1\sigma.$$

Ainsi, cette inversion ne demandera qu'un changement de signe pour 3 coefficients, ce qui peut être absorbé dans les opérations suivantes.

Nous utilisons alors la linéarité du Frobenius π_3 pour obtenir la formule suivante : pour $A = a_0 + a_1\sigma + a_2\rho + a_3\sigma\rho + a_4\rho^2 + a_5\sigma\rho^2$ avec $a_j \in \mathbb{F}_{3^m}$ et $a'_j = a_j^{3^i}$, nous avons

$$A^{3^i} = ((a'_0 - \epsilon_1 a'_2 + \epsilon_2 a'_4) + \epsilon_3(a'_1 - \epsilon_1 a'_3 + \epsilon_2 a'_5)\sigma) + ((a'_2 + \epsilon_1 a'_4) + \epsilon_3(a'_3 + \epsilon_1 a'_5)\sigma)\rho + (a'_4 + \epsilon_3 a'_5\sigma)\rho^2,$$

avec $\epsilon_1 = -ib \mod 3$, $\epsilon_2 = i^2 \mod 3$ et $\epsilon_3 = (-1)^i$. Ainsi cette opération demandera $3m + 3$ applications du Frobenius et au plus 6 additions.

7.3 Courbes de genre 2 en caractéristique 2

La troisième et dernière courbe sur laquelle notre attention s'est portée durant cette thèse est une courbe supersingulière de genre 2 en caractéristique 2 que nous avons décrite à la Section 4.3.1 § 3. Nous rappelons ici son équation :

$$H_2/\mathbb{F}_2 : y^2 + y = x^5 + x^3 + d \text{ avec } d \in \{0, 1\}.$$

Comme il s'agit d'une courbe de genre supérieur, nous travaillerons dans sa jacobienne dont le cardinal est

$$\#\mathrm{Jac}_{H_2}(\mathbb{F}_{2^m}) = 2^{2m} + \delta 2^{\frac{3m+1}{2}} + 2^m + \delta 2^{\frac{m+1}{2}} + 1,$$
$$\text{où } \delta = \begin{cases} (-1)^d & \text{si } m \equiv \pm 1, \pm 7 \pmod{24}, \\ -(-1)^d & \text{si } m \equiv \pm 5, \pm 11 \pmod{24}. \end{cases}$$

Rappelons également que cette courbe est de degré de plongement $k = 12$ et qu'il permet ainsi d'atteindre le même rapport de taille entre la courbe et le groupe d'arrivée que la courbe E_3 avec $k/g = 6$; cependant, cette courbe est de caractéristique 2 et nous pouvons ainsi espérer en tirer un avantage arithmétique. Nous avons fait l'étude de la construction de couplages sur cette courbe dans l'article [Ara+12].

7.3.1 *Distortion map* et endomorphismes de H_2

Le choix de la *distortion map* est cette fois un point crucial de la construction d'un couplage sur H_2. En effet, contrairement à E_2 et E_3, un plus grand choix s'offre à nous et les différentes possibilités donnent des applications dont le calcul est plus ou moins aisé. Ces endomorphismes sont décrits dans l'article [Gal+09, Section 8] ; nous reproduisons ici cette caractérisation.

Les automorphismes de H_2 sont les

$$\sigma_\omega : (x, y) \longmapsto (x + \omega, y + s_{\omega,2}x^2 + s_{\omega,1}x + s_{\omega,0}),$$

tels que

(*i*) ω est une racine du polynôme $x^{16} + x^8 + x^2 + x$,
(*ii*) $s_{\omega,2} = \omega^8 + \omega^4 + \omega$,
(*iii*) $s_{\omega,1} = \omega^4 + \omega^2$, et
(*iv*) $s_{\omega,0}$ est une racine de $y^2 + y + \omega^5 + \omega^3$.

Afin d'exhiber une base de ces automorphismes, nous fixons alors une racine τ de $x^6 + x^5 + x^3 + x^2 + 1$ (remarquons que nous avons bien que $x^6 + x^5 + x^3 + x^2 + 1 \mid x^{16} + x^8 + x^2 + x$) et une racine $s_{\tau,0}$ de $y^2 + y + \tau^5 + \tau^3$. Posons également $\theta = \tau^4 + \tau^2 + \tau$, $s_{\theta,0} = s_{\tau,0} + \tau^5 + \tau^2 + \tau + 1$, $\xi = \tau^4 + \tau^2$ et $s_{\xi,0} = \tau^4 + \tau^2$. Il est alors aisé de s'assurer que σ_τ, σ_θ et σ_ξ sont bien des automorphismes de H_2. La Proposition 8.1 de [Gal+09] nous indique alors que toutes les *distortion maps* sont à chercher dans $\mathbb{Z}[\pi_{2^m}, \sigma_\tau, \sigma_\theta]$.

Notons également que les endomorphismes 1, σ_τ, σ_θ et σ_ξ forment une base de la $\mathbb{Q}(\pi_{2^m})$-algèbre des endomorphismes de Jac_{H_2} et qu'un calcul fastidieux nous donne les relations suivantes :

$$\begin{aligned}
\pi_{2^m}\sigma_\tau\pi_{2^m}^{-1} &= [2^m]\sigma_\tau\pi_{2^m}^{-2} + [\epsilon 2^{2m}]\sigma_\theta\pi_{2^m}^{-4},\\
\pi_{2^m}\sigma_\theta\pi_{2^m}^{-1} &= [-2^{3m}]\sigma_\theta\pi_{2^m}^{-6},\\
\pi_{2^m}\sigma_\xi\pi_{2^m}^{-1} &= [2^{4m}]\sigma_\xi\pi_{2^m}^{-8} + [\epsilon 2^{5m}]\pi_{2^m}^{-10},
\end{aligned}$$

où $\epsilon = (-1)^e$ et $e = 0$ quand $m \equiv \pm 1 \pmod{12}$, et 1 sinon.

Afin de fixer une *distortion map* ψ qui permettra de définir un couplage Ate ou Eta (c'est-à-dire compatible avec l'hypothèse du Théorème 6.2), nous étudions la structure de la ℓ-torsion. C'est un $\mathbb{Z}/\ell\mathbb{Z}$ espace vectoriel de dimension 4 dont une base est formée par $\overline{D_1}$ un élément non trivial de $\mathbb{G}_{\pi_{2^m},1} = \mathrm{Jac}_{H_2}(\mathbb{F}_{2^m})$, $\overline{D_\xi} = \sigma_\tau(\overline{D_1})$, $\overline{D_\theta} = \sigma_\theta(\overline{D_1})$ et, $\overline{D_\xi} = \sigma_\xi(\overline{D_1})$. Nous explicitons alors le Frobenius π_{2^m} dans cette base grâce aux relations précédentes :

$$\pi_{2^m} \equiv \begin{pmatrix} 1 & 0 & 0 & \epsilon 2^{5m} \\ 0 & 2^m & 0 & 0 \\ 0 & \epsilon 2^{2m} & -2^{3m} & 0 \\ 0 & 0 & 0 & 2^{4m} \end{pmatrix} \pmod{\ell}.$$

La première idée est de chercher le sous-espace associé à la valeur propre 2^m afin de pouvoir utiliser le couplage Ate optimal donné par Vercauteren dans [Ver10, Section IV-G]. Comme la matrice du Frobenius n'est pas diagonale, il faut soit abandonner l'utilisation d'une *distortion map* (et avec ceci, l'obtention d'un couplage symétrique et une représentation compacte des diviseurs en entrée) soit diagonaliser cette matrice. Le calcul montre alors que l'application $\psi = (2^{3m} + \pi_{2^m})\sigma_\tau$ permet de définir le sous-espace propre associé à la valeur propre 2^m de π_{2^m}. Nous n'avons cependant pas retenu cette idée car elle conduit à l'utilisation d'une *distortion map* dont le calcul est complexe (une addition-réduction de Cantor par application de la *distortion map*) et qui ralentirait chaque itération de Miller.

Le cas Eta est quant à lui complètement obstrué par le fait que le Verschiebung $\hat{\pi}_{2^m}$ est un endomorphisme purement inséparable de Jac_{H_2} mais pas de la courbe H_2 elle-même. C'est la conséquence directe du fait que la courbe H_2, bien que supersingulière, n'est pas superspéciale.

Afin d'outrepasser ces limitations, nous nous reposons sur une observation de Barreto *et al.* [Bar+07] qui nous permet d'utiliser le sous-espace propre associé à la valeur propre 2^{3m} pour l'action du Verschiebung $\hat{\pi}_{3^m}$. En effet, les difficultés précédentes viennent du fait que la multiplication scalaire $[2^m]$ n'agit que sur la jacobienne et pas sur la courbe. Considérons un diviseur dégénéré $D = (P) - (P_\infty)$, le calcul montre que $[8]D = [x + x_P^{64} + 1, x_P^{128} + y_P^{64} + 1]$ et ainsi que l'octuplement agit non seulement sur la jacobienne mais également sur la courbe :

$$[8]P = (x_P^{64} + 1, x_P^{128} + y_P^{64} + 1).$$

Par extension, $[2^{3m}]$ agit également sur la courbe : $[2^{3m}]P = (x_P^{2^{3m}} + 1, x_P^{2^{3m+1}} + y_P^{2^{3m}} + \nu)$ avec $\nu = \frac{m+1}{2} \mod 2$. Nous considérons alors le Frobenius $\pi_{2^{3m}}$ et le Verschiebung $\hat{\pi}_{2^{3m}}$ et leurs écritures dans la base $(\overline{D_1}, \overline{D_\tau}, \overline{D_\theta}, \overline{D_\xi})$:

$$\pi_{2^{3m}} \equiv \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2^{3m} & 0 & 0 \\ 0 & 0 & 2^{3m} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \pmod{\ell} \quad \text{et} \quad \hat{\pi}_{2^{3m}} \equiv \begin{pmatrix} 2^{3m} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2^{3m} \end{pmatrix} \pmod{\ell}.$$

Cette fois nous obtenons directement des matrices diagonales et, si nous nous intéressons aux couplages constructibles dans les deux premières dimensions, nous pourrons utiliser la *distortion map*

$$\psi = \sigma_\tau.$$

Notons qu'à la différence de la *distortion map* précédente, celle-ci est définie de la courbe vers la courbe et non vers la jacobienne et ne demande aucune réduction.

Nous avons choisi d'utiliser le Verschiebung $\hat{\pi}_{2^{3m}}$ et de construire ainsi le couplage Eta, puisque

$$\forall D \in \mathrm{Jac}_{H_2}(\mathbb{F}_{2^m})[\ell], \hat{\pi}_{2^{3m}}(D) = [2^{3m}]D \text{ et } \hat{\pi}_{2^{3m}}(\psi(D)) = \psi(D).$$

Il est également possible de choisir la version Ate mais, à l'instar des couplages en genre 1, nous ne pouvons espérer une grande différence de coût arithmétique.

7.3.2 Construction d'un couplage Eta optimal

Encore une fois, nous devons fixer l'exponentiation finale. Pour ce faire nous choisissons $L = \#\mathrm{Jac}_{H_2}(\mathbb{F}_{2^m})$. L'exposant est alors

$$\frac{2^{12m}-1}{2^{2m}+\delta 2^{\frac{3m+1}{2}}+2^m+\delta 2^{\frac{m+1}{2}}+1} = (2^{6m}-1)(2^{2m}+1)(2^{2m}-\delta 2^{\frac{3m+1}{2}}+2^m-\delta 2^{\frac{m+1}{2}}+1).$$

Nous détaillons ici la construction optimale pour le couplage Eta reposant sur le Verschiebung $\hat{\pi}_{2^{3m}}$. Nous utiliserons donc les deux premières dimensions de la ℓ-torsion et posons $\mathbb{G}_1 = \langle \overline{D_1} \rangle$ et $\mathbb{G}_2 = \langle \overline{D_\tau} \rangle$. Comme nous ne pouvons pas utiliser la valeur propre 2^m, nous avons réduit la dimension du réseau euclidien à explorer. En effet, il n'est pas de dimension $\varphi(12) = 4$ mais $\varphi(12/3) = 2$. Le réseau que nous avons considéré est donc engendré par les lignes de la matrice

$$\begin{pmatrix} 2^{2m}+\delta 2^{\frac{3m+1}{2}}+2^m+\delta 2^{\frac{m+1}{2}}+1 & 0 \\ -2^{3m} & 1 \end{pmatrix},$$

et nous utiliserons

$$(c_0, c_1) = \left(\delta 2^{\frac{m-1}{2}}+1, 2^m+\delta 2^{\frac{m-1}{2}}\right)$$

parmi ses vecteurs courts.

L'expression du couplage correspondant est alors, pour $(\overline{D_1}, \overline{D_2}) \in \mathbb{G}_1 \times \mathbb{G}_2$

$$\eta_{\left(\delta 2^{\frac{m-1}{2}}+1, 2^m+\delta 2^{\frac{m-1}{2}}\right)}(\overline{D_1}, \overline{D_2}) = \left(f_{\delta 2^{\frac{m-1}{2}}+1, D_1} \cdot f^{2^{3m}}_{2^m+\delta 2^{\frac{m-1}{2}}, D_1}\right)(D_2)^{\frac{2^{12m}-1}{L}}.$$

En effet, comme $\left(2^m+\delta 2^{\frac{m-1}{2}}\right) \cdot 2^{3m} \equiv -\left(\delta 2^{\frac{m-1}{2}}+1\right) \pmod{\ell}$, les droites

$$g_{\left[\delta 2^{\frac{m-1}{2}}+1\right]D_1, \left[\left(2^m+\delta 2^{\frac{m-1}{2}}\right)\cdot 2^{3m}\right]D_1}$$

qui apparaissent dans la construction optimale sont des verticales que nous pouvons éliminer.

Nous transformons alors ce produit de fonctions de Miller afin d'obtenir un produit de fonctions de Miller dont les paramètres sont de poids de Hamming minimaux. Ainsi, nous pourrons déduire une itération efficace pour le calcul de ce couplage Eta optimal.

$$\begin{aligned} f_{2^{\frac{m-1}{2}}+1,D_1} &= f_{\delta 2^{\frac{m-1}{2}},D_1} \cdot g_{\left[\delta 2^{\frac{m-1}{2}}\right]D_1,D_1}, \\ f_{2^m+\delta 2^{\frac{m-1}{2}},D_1} &= f_{2^m,D_1} \cdot f_{\delta 2^{\frac{m-1}{2}},D_1} \cdot g_{[2^m]D_1,\left[\delta 2^{\frac{m-1}{2}}\right]D_1}. \end{aligned}$$

Nous éliminons alors quelques verticales pour obtenir :

$$\begin{aligned} f_{\delta 2^{\frac{m-1}{2}},D_1} &= f_{2^{\frac{m-1}{2}},[\delta]D_1}, \text{ et} \\ f_{2^m,D_1} &= f_{\delta 2^{\frac{m-1}{2}}\cdot\delta 2^{\frac{m+1}{2}},D_1} \\ &= f^{\delta 2^{\frac{m+1}{2}}}_{2^{\frac{m-1}{2}},[\delta]D_1} \cdot f_{2^{\frac{m+1}{2}},\left[2^{\frac{m-1}{2}}\right]D_1}. \end{aligned}$$

Enfin, nous obtenons

$$\begin{aligned} f_{c_0,D_1} &= f_{2^{\frac{m-1}{2}},[\delta]D_1} \cdot g_{\left[\delta 2^{\frac{m-1}{2}}\right]D_1,D_1}, \\ f_{c_1,D_1} &= f^{\delta 2^{\frac{m+1}{2}}+1}_{2^{\frac{m-1}{2}},[\delta]D_1} \cdot f_{2^{(m+1)/2},\left[2^{\frac{m-1}{2}}\right]D_1} \cdot g_{[2^m]D_1,\left[\delta 2^{\frac{m-1}{2}}\right]D_1}. \end{aligned}$$

Ainsi, le couplage optimal Eta que nous proposons se ramène essentiellement au calcul de deux fonctions de Miller de paramètres $(m-1)/2$ et $(m+1)/2$. Ceci représente une amélioration de 33% en terme de longueur de boucle de Miller par rapport au couplage Eta T décrit dans [Bar+07].

Nous avons utilisé le fait que l'octuplement agit sur la courbe de manière fondamentale dans la section précédente afin de permettre la construction d'un couplage Eta, et nous continuons d'utiliser ce fait : la fonction de Miller $f_{8,D}$ est d'expression plus simple que celle du doublement car elle ne fait pas intervenir une réduction de Cantor complète : chaque point du support de D est indépendant, de ce point de vue. Afin donc d'exploiter l'octuplement au mieux, nous considérons deux cas selon la valeur de m modulo 6.

— Si $m \equiv 1 \pmod 6$, $(m-1)/2$ est un multiple de 3 et $f_{2^{\frac{m-1}{2}},D_1}$ se calcule avec $(m-1)/6$ octuplements ; $f_{2^{(m+1)/2},[2^{(m-1)/2}]D_1}$ a contrario demande $(m-1)/6$ octuplements et un doublement.

— Si $m \equiv 5 \pmod 6$, $(m-1)/2$ n'est plus un multiple de 3. Afin de limiter le nombre de doublements, nous calculerons dans ce cas $\eta_{(2c_0,2c_1)}$ grâce aux formules suivantes :

$$\begin{aligned} f_{2c_0,D_1} &= f_{2^{(m+1)/2},[\delta]D_1} \cdot f_{2,D_1} \cdot g_{[\delta 2^{(m+1)/2}]D_1,[2]D_1}, \\ f_{2c_1,D_1} &= f^{\delta 2^{(m+1)/2}+1}_{2^{(m+1)/2},[\delta]D_1} \cdot f_{2^{(m+1)/2},[2^{(m+1)/2}]D_1} \cdot g_{[2^{m+1}]D_1,[\delta 2^{(m+1)/2}]D_1} ; \end{aligned}$$

il nous faudra alors $(m+1)/6$ octuplements pour calculer chacune des fonctions de Miller de paramètre $2^{\frac{m+1}{2}}$, et un doublement pour $f_{2,D}$.

Nous présentons un premier algorithme (Algorithme 7.13) découlant directement de cette analyse. Le coût d'un tel algorithme est alors de $\lfloor m/3 \rceil$ octuplements et un doublement.

Algorithme 7.13 Calcul du couplage Eta optimal sur H_2.

Entrées : $\overline{D_1}, \overline{D_2} \in \mathrm{Jac}_{H_2}(\mathbb{F}_{2^m})[\ell]$
Sortie : $\eta_{(c_0,c_1)}(\overline{D_1}, \overline{D_2})$ ou $\eta_{(c_0,c_1)}(\overline{D_1}, \overline{D_2})^2 \in \mu_\ell \subseteq \mathbb{F}^*_{2^{12m}}$, selon que $m \equiv 1$ ou $5 \pmod 6$.

1. **if** $m \equiv 1 \pmod 6$ **then** $m' \leftarrow m-1$ **else** $m' \leftarrow m+1$ **end if**
2. $G_1 \leftarrow 1$; $R_1 \leftarrow [\delta]D_1$; $E_2 \leftarrow \epsilon(\psi(D_2))$
3. **for** $i \leftarrow 1$ **to** $m'/6$ **do**
4. $\quad G_1 \leftarrow G_1^8 \cdot f_{8,R_1}(E_2)$
5. $\quad R_1 \leftarrow [8]R_1$ $\qquad (R_1 = [\delta 8^i]D_1)$
6. **end for** $\qquad (G_1 = f_{2^{m'/2},[\delta]D_1}(E_2)$ et $R_1 = [\delta 2^{m'/2}]D_1)$
7. $G_2 \leftarrow G_1^\delta$; $R_2 \leftarrow [\delta]R_1$
8. **for** $i \leftarrow 1$ **to** $m'/6$ **do**
9. $\quad G_2 \leftarrow G_2^8 \cdot f_{8,R_2}(E_2)$
10. $\quad R_2 \leftarrow [8]R_2$ $\qquad (R_2 = [8^{\frac{m'}{6}+i}]D_1)$
11. **end for** $\qquad (G_2 = f_{2^{m'},D_1}(E_2)$ et $R_2 = [2^{m'}]D_1)$
12. **if** $m \equiv 1 \pmod 6$ **then**
13. $\quad G_2 \leftarrow G_2^2 \cdot f_{2,R_2}(E_2)$ $\qquad (G_2 = f_{2^m,D_1}(E_2))$
14. $\quad F_0 \leftarrow G_1 \cdot g_{R_1,D_1}(E_2)$ $\qquad (F_0 = f_{c_0,D_1}(E_2))$
15. $\quad F_1 \leftarrow G_1 \cdot G_2 \cdot g_{[2]R_2,R_1}(E_2)$ $\qquad (F_1 = f_{c_1,D_1}(E_2))$
16. **else**
17. $\quad F_0 \leftarrow G_1 \cdot f_{2,D_1}(E_2) \cdot g_{R_1,[2]D_1}(E_2)$ $\qquad (F_0 = f_{2c_0,D_1}(E_2))$
18. $\quad F_1 \leftarrow G_1 \cdot G_2 \cdot g_{R_2,R_1}(E_2)$ $\qquad (F_1 = f_{2c_1,D_1}(E_2))$
19. **end if**
20. **return** $\left(F_1^{2^{3m}} \cdot F_0\right)^{\frac{2^{12m}-1}{L}}$

7.3.3 Arithmétique de la courbe et évaluation de complexité

L'algorithme précédent ramène le calcul du couplage optimal Eta à l'évaluation successive de fonctions de Miller $f_{8,[\pm 8^i]D_1}$ en un diviseur effectif $\epsilon(\psi(D_2))$. Afin d'évaluer le coût arithmétique global de cet algorithme, nous nous intéressons donc à ce calcul.

Avant d'étudier ce calcul, nous revenons sur le choix des entrées du couplage. Étant donné que nous travaillons maintenant en genre supérieur, nous avons plus de choix sur la forme des diviseurs utilisés. En genre 2, il peuvent être de deux types :

— **dégénéré** : $D = (P) - (P_\infty)$, ou
— **générique** : $D = (P_1) + (P_2) - 2(P_\infty)$.

Notons que, comme l'octuplement et notre *distortion map* $\psi = \sigma_\tau$ agissent directement sur la courbe, le caractère dégénéré d'un diviseur en entrée du couplage se conserve durant l'essentiel de l'exécution de l'Algorithme 7.13. De plus, un diviseur dégénéré demande deux fois moins de coordonnées qu'un diviseur générique et engendre également moins de calculs. Nous envisagerons donc plusieurs versions de notre algorithme selon qu'il ne prendra que des diviseurs génériques, dégénérés ou un de chaque. Selon le protocole qui utilisera le couplage que nous calculons, il est possible de choisir une version dégénérée et ainsi de gagner un facteur approximativement 2 à chaque fois que nous restreignons une entrée aux diviseurs dégénérés [3]. Par exemple, Chatterjee *et al.* [CHM10] ont proposé une variante de la signature courte BLS [BLS04] permettant d'utiliser un diviseur dégénéré pour l'une des entrées.

Le choix de la *distortion map* $\psi = \sigma_\tau$ a fixé une partie de la tour d'extension qui nous permet de représenter $\mathbb{F}_{2^{12m}}$ mais il nous reste cependant un choix pour l'extension de degré 6 sous-jacente : il nous faut un élément τ tel que $\tau^6 + \tau^5 + \tau^3 + \tau^2 + 1 = 0$, mais nous pouvons passer par une extension intermédiaire quadratique ou cubique. Une étude de ces deux solutions a montré que passer par une extension quadratique nous fournissait des formules moins complexes, nous posons donc l'extension suivante :

$$\begin{array}{l} \mathbb{F}_{2^{12m}} = \mathbb{F}_{2^{6m}}[s_{\tau,0}] \text{ où } s_{\tau,0}^2 + s_{\tau,0} + \tau^2 + i\tau + \tau = 0 \\ \quad | \\ \mathbb{F}_{2^{6m}} = \mathbb{F}_{2^{2m}}[\tau] \text{ où } \tau^3 + i\tau^2 + i\tau + i = 0 \\ \quad | \\ \mathbb{F}_{2^{2m}} = \mathbb{F}_{2^m}[i] \text{ où } i^2 + i + 1 = 0 \\ \quad | \\ \mathbb{F}_{2^m} \end{array}$$

dont nous vérifions la compatibilité avec les définitions précédentes.

Avec cette représentation, nous obtenons les coûts arithmétiques données en Table 7.14. Enfin nous donnons également le coût total du calcul du couplage Eta optimal pour le corps de base $\mathbb{F}_{2^{367}}$ que nous avons utilisé dans notre implémentation. Nous y donnons également le coût de l'exponentiation finale. Pour celle-ci, nous ne donnons pas l'algorithme mais il est très proche du cas elliptique.

3. Nous pouvons constater ce facteur 2 dans la Table 7.14.

TABLE 7.14 – Coût arithmétique des différentes opérations utilisées dans le couplage Eta optimal sur H_2.

Opération	D_1	D_2	**Opérations sur $\mathbb{F}_{2^m}$**			
			Mult.	**Add.**	**Frob.**	**Inv.**
Addition dans $\mathbb{F}_{2^{12m}}$	—	—	0	12	0	0
Frobenius π_2 sur $\mathbb{F}_{2^{12m}}$	—	—	0	21	12	0
Multiplication dans $\mathbb{F}_{2^{12m}}$	—	—	45	199	0	0
$[\pm 8^i]D_1 \mapsto [\pm 8^{i+1}]D_1$	Dég.	—	0	2	13	0
	Gén.	—	0	5	24	0
$f_{4,[\pm 8^i]D_1}(\epsilon(\psi(D_2)))$	Dég.	Dég.	3	11	1	0
	Gén.	Dég.	19	40	2	0
	Gén.	Gén.	83	247	17	0
$f_{2,[\pm 4\cdot 8^i]D_1}(\epsilon(\psi(D_2)))$	Dég.	Dég.	2	9	1	0
	Gén.	Dég.	16	34	2	0
	Gén.	Gén.	81	236	17	0
Itération de Miller	Dég.	Dég.	61	315	68	0
$\begin{cases} G_i \leftarrow G_i^8 \cdot f_{8,R_i}(E_2) \\ R_i \leftarrow [8]R_i \end{cases}$	Gén.	Dég.	121	512	130	0
	Gén.	Gén.	254	949	160	0
Exp. finale sur $\mathbb{F}_{2^{367}}$	—	—	303	1 386	2 234	1
Couplage Eta Optimal	Dég.	Dég.	7 894	40 356	11 571	1
$\eta_{[c_0,c_1]}(\overline{D}_1, \overline{D}_2)$	Gén.	Dég.	15 293	64 644	15 472	1
over $C_0(\mathbb{F}_{2^{367}})$	Gén.	Gén.	31 644	118 382	19 161	1

Troisième partie

Arithmétique de corps fini

Chapitre 8

Représentation et opérations arithmétiques

Dans ce chapitre, nous nous attardons sur la description de la représentation et de l'arithmétique des corps finis. Les corps finis sont les objets mathématiques de plus bas niveau que nous manipulons et tous nos algorithmes sont exprimés en tant que séquences d'opérations élémentaires sur des éléments de corps finis. Aussi nous devons les présenter et montrer comment il est possible d'optimiser ces opérations. Dans le cadre de cette thèse, nous n'utilisons que des corps finis de petite caractéristique, dont nous rappelons la représentation en Section 8.1. Comme nous l'avons déjà vu dans la partie précédente, l'opération critique est la multiplication. Afin d'optimiser le calcul de couplages, nous avons cherché à en réduire le nombre nécessaire dans la partie précédente ; ici (Section 8.2) nous cherchons à en optimiser le coût. Cette optimisation passe également par la production de nouvelles formules de multiplication ; nous décrirons au chapitre suivant un nouvel algorithme permettant de les produire.

8.1 Corps de caractéristique 2 et 3

Pour des raisons cryptographiques[1], nous nous sommes limités aux caractéristiques $p = 2$ et 3 dans cette thèse. Ces corps finis de petite caractéristique présentent deux particularités importantes au vu des calculs que nous faisons avec eux. En premier lieu, ce sont des extensions de corps de petite taille dont l'arithmétique est triviale, ainsi il convient de travailler ici l'arithmétique d'extension, notamment en choisissant correctement le polynôme de définition de l'extension. La deuxième particularité est l'existence d'un automorphisme de Frobenius ($x \mapsto x^p$) facile à calculer pour peu que le polynôme de définition soit bien choisi et dont le calcul de couplage fait largement usage.

8.1.1 Représentation de $\mathbb{F}_p$

En caractéristique 2, le corps de base est $\mathbb{F}_2$ et il correspond ainsi à un bit muni des opérations XOR (addition) et AND (multiplication).

1. Il existe peu de courbes supersingulières cryptographiquement intéressantes en petite caractéristique supérieure ou égale à 5.

En revanche, le choix d'une représentation est moins évident pour la caractéristique 3. Une première idée est de coder sur deux bits les chiffres 0, 1 et 2. Cependant nous avons plutôt utilisé une représentation redondante où chaque élément est donné par 2 bits suivant la représentation *borrow-save* [DM91]. Dans cette représentation, chaque élément $a \in \mathbb{F}_3$ est donné par deux bits a_+ et a_- tel que $a = a_+ - a_-$. Cette représentation redondante a pour avantage de donner un calcul d'opposé gratuit en matériel (il suffit d'échanger les deux bits a_+ et a_-) et de simplifier l'expression du produit de deux éléments de $\mathbb{F}_3$.

8.1.2 Représentation du corps

Nous travaillons dans des extensions de corps $\mathbb{F}_{q^n}$ avec n premier. Dans le cas où le degré d'extension n est composé, il faudra utiliser une structure de tour d'extensions en répétant le mécanisme d'extension de corps. *In fine* la tour d'extensions reposera sur des extensions $\mathbb{F}_{q^n}$ avec n premier dont il faudra implémenter l'arithmétique. Nous nous concentrons donc ici sur ce cas. Le modèle de coût change dramatiquement lorsque le corps de base n'est plus un corps premier de petite taille et que le degré d'extension est de petite taille. C'est un cas que nous avons rencontré pour le calcul de couplage : le résultat d'un couplage vit dans une extension particulière fixée par le choix de la courbe. Nous avons également rencontré ce cas lors d'une implémentation où nous avons choisi un corps de définition de degré d'extension composé pour accélérer les calculs, celle-ci est décrite à la Section 12.3. Dans ces cas, il convient de faire une étude *ad hoc* de l'arithmétique de ces extensions.

Un corps $\mathbb{F}_{q^n}$ est construit comme le corps de rupture d'un polynôme irréductible f de degré n sur $\mathbb{F}_q$. Durant cette thèse, nous avons privilégié l'utilisation de la base polynomiale pour représenter les éléments du corps $\mathbb{F}_{q^n}$: chaque élément est ainsi représenté par un polynôme sur $\mathbb{F}_q$ modulo f, et, le plus souvent, par un représentant canonique : un polynôme de degré au plus $n-1$. Nous aurions également pu utiliser des bases normales qui permettent d'obtenir un calcul de l'automorphisme de Frobenius gratuit en matériel (il s'agit alors d'un décalage des coefficients) mais celles-ci compliquent le calcul du produit.

8.1.3 Choix du polynôme irréductible de définition et automorphisme de Frobenius

Lors du calcul de la multiplication de deux éléments ou du Frobenius, le résultat obtenu dans un premier temps est non réduit et il faut alors le réduire modulo f où f est le polynôme irréductible de définition de $\mathbb{F}_{q^n}$. Afin de réduire le coût de cette réduction, il convient de choisir convenablement ce polynôme.

La première idée est de choisir un polynôme irréductible f de poids de Hamming faible, ainsi les coefficients qui doivent être réduits n'auront pas à être additionnés un trop grand nombre de fois pour obtenir le résultat. C'est une considération particulièrement importante pour les implémentations matérielles car elle permet de réduire la taille de l'arbre d'addition calculant chaque coefficient du résultat. Pour les cas que nous avons rencontrés, il a presque toujours été possible de trouver des polynômes irréductibles de poids minimal, c'est-à-dire des trinômes, ou dans les autres cas des pentanômes.

Dans le calcul du Frobenius, les éléments à réduire ont une forme particulière (1 coefficient sur 2 nul en caractéristique 2, 2 sur 3 en caractéristique 3) qui donne une réduction plus efficace encore en choisissant le trinôme ou pentanôme irréductible. Nous référons le lecteur à [Beu+11, IV. A. 6] et [RML08] pour plus de détails.

8.1.4 Algorithme d'inversion

Lors du calcul de couplage, une inversion est nécessaire dans l'exponentiation finale. Aussi c'est une opération non critique mais coûteuse. Il est donc intéressant de calculer ces inverses en ne demandant pas d'introduire de nouvelle primitives pour ce calcul comme c'est le cas pour l'algorithme d'Euclide étendu. Nous utiliserons donc l'algorithme d'Itoh & Tsujii [IT88 ; GP02].

L'algorithme d'Itoh & Tsujii (Algorithme 8.1) consiste à ramener le calcul d'un inverse dans une extension de corps $\mathbb{F}_{q^n}$ à un calcul d'inverse dans le corps de base $\mathbb{F}_q$ en utilisant l'identité suivante du petit théorème de Fermat : pour tout $a \in \mathbb{F}_{q^n}$, $a \neq 0$,

$$a^{-1} = a^{q^n-2}.$$

Ainsi, il faudra élever a à la puissance

$$q^n - 2 = \underbrace{\overline{(q-1)\dots(q-1)}}_{n-1 \text{ fois}}\overline{(q-2)}^{(q)}$$

pour obtenir son inverse. L'algorithme se déduit alors de l'écriture en base q de l'exposant. Dans un premier temps (lignes 1 à 5), il faut calculer

$$r = a^{\frac{q^{n-1}-1}{q-1}} = a^{\overline{\underbrace{1\dots1}_{n-1}}^{(q)}}.$$

Pour ce faire, nous définissons la séquence suivante

$$\forall i \in \{0,\dots,n-1\}, w_i = a^{\frac{q^i-1}{q-1}} = a^{\overline{\underbrace{1\dots1}_{i}}^{(q)}},$$

pour laquelle nous avons la relation suivante

$$w_{i+j} = w_i \cdot (w_j)^{q^i} = w_i \cdot a^{\overline{\underbrace{1\dots1}_{j}\underbrace{0\dots0}_{i}}^{(q)}}.$$

Étant donnée une chaîne d'addition pour $n-1$ de longueur l, il est alors possible de calculer $r = w_{n-1}$ avec l multiplication et $n-2$ applications du q-Frobenius ($x \mapsto x^q$). Nous choisissons, en général, d'utiliser une chaîne d'addition de Brauer, c'est-à-dire une chaîne où chaque nouvelle somme est l'addition de la dernière calculée et d'une autre précédemment calculée. Pour $n \leq 2500$, ces chaînes sont en effet optimales.

Puis nous calculons $s = r^q$ en appliquant une fois de plus le q-Frobenius qu'il faudra multiplier à

$$u = a^{(q-2)\cdot\frac{q^n-1}{q-1}} = a^{\overline{\underbrace{(q-2)\dots(q-2)}_{n}}^{(q)}}$$

pour obtenir l'inverse souhaité. Pour ce faire, nous commençons par obtenir $t = a^{\frac{q^n-1}{q-1}}$ (ligne 7) et constatons que c'est un élément du corps de base $\mathbb{F}_q$. Ainsi, élever cet élément de $\mathbb{F}_q$ à la puissance $q-2$ correspond à son inversion (petit théorème de Fermat). L'algorithme termine alors par la multiplication de s et u qui est particulière étant donné que u appartient au corps de base $\mathbb{F}_q$.

Algorithme 8.1 Algorithme d'Itoh & Tsujii pour le calcul d'inverse sur $\mathbb{F}_{q^n}$.

Entrées : $a \in \mathbb{F}_{q^n}^*$, $(B_i)_{0 \leq i \leq l-1}$ une chaîne d'addition de Brauer pour $n-1$.
Sortie : a^{-1}

1. $w_1 \leftarrow a$
2. **for** $i \in 1, \ldots, l-1$ **do**
3. $\quad w_{B_i} \leftarrow w_{B_i - B_{i-1}} \cdot (w_{B_{i-1}})^{q^{B_i - B_{i-1}}}$ $\quad ((B_i - B_{i-1})$ Frob., 1 mul. sur $\mathbb{F}_{q^n})$
4. **end for**
5. $r \leftarrow w_{n-1}$
6. $s \leftarrow r^q$ $\quad$ (1 Frob. sur $\mathbb{F}_{q^n}$)
7. $t \leftarrow s \cdot a$ $\quad$ (1 mul. sur $\mathbb{F}_{q^n}$)
8. $u \leftarrow t^{-1}$ $\quad$ (1 inv. sur $\mathbb{F}_q$)
9. **return** $s \cdot u$ $\quad$ (n mul. sur $\mathbb{F}_q$)

Le coût total de l'algorithme pour une inversion sur $\mathbb{F}_{q^n}$ est donc de l multiplications, $n-1$ applications du q-Frobenius sur l'extension $\mathbb{F}_{q^n}$, n multiplications et une inversion sur le corps de base $\mathbb{F}_q$.

Nous utiliserons cet algorithme notamment sur des corps de la forme $\mathbb{F}_{2^m}$ ou $\mathbb{F}_{3^m}$. Dans ces cas, il faut remarquer que l'opération d'inverse sur le corps de base est triviale puisque que l'inversion, sur $\mathbb{F}_2^*$ comme sur $\mathbb{F}_3^*$, correspond à l'identité. De plus, quand $q = 2$, t est nécessairement l'unité et le résultat de l'algorithme est directement s.

Exemple 8.1 (Calcul d'un inverse sur $F_{3^{5 \cdot 97}}$). Nous montrons ici l'exemple du calcul d'un inverse sur $\mathbb{F}_{3^{5 \cdot 97}}$ que nous utiliserons dans une implémentation décrite à la Section 12.3. Pour ce faire, nous appliquons une première fois l'Algorithme 8.1 avec $3^{97} \to q$ et $5 \to n$. Nous notons m l'opération de multiplication sur $\mathbb{F}_{3^{97}}$, a celle d'addition et i l'inversion. Pour les multiplications dans le corps $\mathbb{F}_{3^{5 \cdot 97}}$, nous utilisons l'Algorithme 9.8 qui utilise 11 multiplications et 47 additions.

Entrées : $A = a_0 + a_1 X + a_2 X^3 + a_3 X^3 + a_4 X^4 \in F_{3^{5 \cdot 97}} \cong \mathbb{F}_{3^{97}}[X]/(X^5 - X + 1)$, $A \neq 0$.
Sortie : A^{-1}

1. $W_1 \leftarrow A$
2. $W_2 \leftarrow W_1 \cdot W_1^{3^{97}}$ $\quad$ (11m, 53a)
3. $W_4 \leftarrow W_2 \cdot W_2^{3^{2 \cdot 97}}$ $\quad$ (11m, 54a)
4. $s_0 + s_1 X + s_2 X^3 + s_3 X^3 + s_4 X^4 \leftarrow W_4^{3^{97}}$ $\quad$ (6a)
5. $t \leftarrow s_0 \cdot a_0 - s_1 \cdot a_4 - s_2 \cdot a_3 - s_3 \cdot a_2 - s_4 \cdot a_1$ $\quad$ (5m, 4a)
6. $u \leftarrow t^{-1}$ $\quad$ (1i)
7. **return** $s_0 \cdot u + s_1 \cdot uX + s_2 \cdot uX^3 + s_3 \cdot uX^3 + s_4 \cdot uX^4$ $\quad$ (5m)

Ainsi, il faudra 32 multiplications, 117 additions et une inversion dans $\mathbb{F}_{3^{97}}$ pour ce calcul. Si nous cherchons à ne pas utiliser l'inverse sur $\mathbb{F}_{3^{97}}$, il faut alors utiliser une autre instance de l'algorithme d'Itoh & Tsujii avec $3 \to q$ et $97 \to n$. Nous notons f le 3-Frobenius sur $\mathbb{F}_{3^{97}}$.

Entrées : $a \in \mathbb{F}_{3^{97}}^*$
Sortie : a^{-1}

1. $w_1 \leftarrow a$
2. $w_2 \leftarrow w_1 \cdot w_1^3$ $\quad$ (1m, 1f)
3. $w_4 \leftarrow w_2 \cdot w_2^{3^2}$ $\quad$ (1m, 2f)
4. $w_8 \leftarrow w_4 \cdot w_4^{3^4}$ $\quad$ (1m, 4f)
5. $w_{16} \leftarrow w_8 \cdot w_8^{3^8}$ $\quad$ (1m, 8f)

TABLE 8.2 – Chaînes d'addition de Brauer utilisées dans cette thèse.

n	l	$(B_i)_{0\leq i\leq l-1}$
4	3	$1, 2, 4$
6	4	$1, 2, 4, 6$
8	4	$1, 2, 4, 8$
10	5	$1, 2, 4, 8, 10$
42	8	$1, 2, 4, 8, 10, 14, 28, 42$
52	8	$1, 2, 4, 8, 10, 18, 26, 52$
66	8	$1, 2, 4, 8, 16, 32, 33, 66$
70	9	$1, 2, 4, 8, 12, 14, 28, 42, 70$
72	8	$1, 2, 4, 8, 16, 20, 36, 72$
78	9	$1, 2, 4, 8, 16, 24, 26, 52, 78$
82	9	$1, 2, 4, 8, 16, 32, 64, 80, 82$
88	9	$1, 2, 4, 8, 12, 20, 40, 80, 88$
96	8	$1, 2, 4, 8, 16, 32, 64, 96$
102	9	$1, 2, 4, 8, 16, 32, 34, 68, 102$
162	10	$1, 2, 4, 8, 16, 32, 64, 128, 160, 162$
166	10	$1, 2, 4, 8, 16, 32, 34, 66, 132, 166$
192	9	$1, 2, 4, 8, 16, 32, 64, 128, 192$
226	11	$1, 2, 4, 8, 16, 32, 64, 128, 192, 224, 226$
238	11	$1, 2, 4, 8, 16, 32, 34, 68, 136, 204, 238$
312	12	$1, 2, 4, 8, 16, 32, 64, 96, 104, 208, 312, 313$
456	12	$1, 2, 4, 8, 16, 32, 64, 128, 256, 384, 448, 456$
556	13	$1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 544, 552, 556$
612	12	$1, 2, 4, 8, 16, 32, 64, 68, 136, 272, 544, 612$
690	13	$1, 2, 4, 8, 16, 32, 64, 128, 144, 272, 544, 688, 690$

Cette table a été extraite du site http://wwwold.iit.cnr.it/staff/giovanni.resta/ac/.

6. $w_{32} \leftarrow w_{16} \cdot w_{16}^{3^{16}}$ (1m, 16f)
7. $w_{64} \leftarrow w_{32} \cdot w_{32}^{3^{32}}$ (1m, 32f)
8. $w_{96} \leftarrow w_{32} \cdot w_{64}^{3^{32}}$ (1m, 32f)
9. $s \leftarrow w_{96}^{3}$ (1f)
10. $t \leftarrow s \cdot a$ (1m)
11. **return** $s \cdot t$ (1m)

Notons que les deux dernières multiplications sont particulières puisque la variable t contient ± 1 ; selon les possibilités de l'architecture d'exécution, il est possible de réduire le coût de celles-ci. Ainsi, il est possible de remplacer une inversion sur $\mathbb{F}_{3^{97}}$ par 9 multiplications et 96 applications du 3-Frobenius.

8.2 Multiplication

Afin de multiplier deux éléments d'une extension de corps, l'idée naturelle est de multiplier leurs représentants dans la base polynomiale puis de les réduire modulo le polynôme irréductible de définition. Nous verrons dans le chapitre suivant une méthode pour chercher des formules donnant directement le produit réduit dans le cas d'une extension de petit degré. Pour des degrés supérieurs nous devons cependant passer par une multiplication polynomiale et une réduction. Ainsi, nous présentons dans cette section les algorithmes de multiplication polynomiale que nous avons utilisés.

Comme nous nous intéressons à des multiplications de petite taille ou de taille cryptographique, nous ne présentons pas ici les algorithmes asymptotiquement optimaux de multiplication tels que l'algorithme de Schönhage & Strassen [SS71].

Nous donnons une présentation succincte des constructions classiques d'arithmétique rapide pour les tailles moyennes. Nous considérons ici la multiplication de polynômes définis sur un anneau pour plus de simplicité. Nous verrons dans le chapitre suivant une formalisation plus satisfaisante en nous plaçant dans une K-algèbre (Section 9.1).

8.2.1 Multiplication naïve

Nous commençons par rappeler l'algorithme de multiplication naïve. Il consiste à calculer tous les produits des coefficients des opérandes deux à deux et à en faire la somme. Soient $A = \sum_{i=0}^{n} a_i X^i$ et $B = \sum_{j=0}^{m} b_j X^j$. Le produit $C = A \cdot B$ s'exprime alors par le somme suivante :

$$C = \sum_{k=0}^{n+m} \left(\sum_{i+j=k} a_i b_j \right) X^k.$$

Il requiert ainsi $(n+1)(m+1)$ produits et nm additions de coefficients. Nous avons ainsi une complexité quadratique en la taille des opérandes.

8.2.2 Algorithme de Karatsuba

La première découverte significative concernant le calcul de la multiplication a eu lieu en 1962 lorsque Karatsuba a proposé un premier algorithme de complexité sous-quadratique pour la multiplication [KO63]. L'algorithme consiste à séparer chaque opérande en parties

haute et basse et à utiliser l'identité remarquable suivante :

$$a'b + ab' = (a + a')(b + b') - ab - a'b'.$$

Afin que cet algorithme puisse fonctionner, nous devons nous limiter à des opérandes de même degré n. Nous notons A_H et A_L les parties haute et basse de A, ainsi que B_H, B_L celles de B. Nous avons alors

$$\begin{cases} A = A_L + X^{\lceil n/2 \rceil} A_H, \text{ et} \\ B = B_L + X^{\lceil n/2 \rceil} B_H. \end{cases}$$

Le produit s'écrit alors

$$\begin{aligned} A \cdot B &= A_H B_H X^{2\lceil n/2 \rceil} + (A_H B_L + A_L B_H) X^{\lceil n/2 \rceil} + A_L B_L \\ &= A_H B_H X^{2\lceil n/2 \rceil} + ((A_H + A_L)(B_H + B_L) - A_H B_H - A_L B_L)\, X^{\lceil n/2 \rceil} + A_L B_L, \end{aligned}$$

et nous pouvons ainsi utiliser cette formule pour ne calculer que trois produits de taille moitié. En utilisant cette méthode récursivement, nous aurons alors une complexité en $O(n^{\log_2 3})$ ($\log_2 3 \approx 1.58$).

Notons qu'il existe une variante de l'algorithme de Karatsuba où les opérandes sont découpés en trois. Il faut alors utiliser trois fois l'identité remarquable et nous obtenons 6 sous-produits à la place de 9. Cette alternative n'offre aucune amélioration asymptotique ($\log_3 6 \approx 1.63$) mais peut être intéressante lorsque la taille du polynôme est divisible par 3.

§ 1. Découpage pair-impair. L'algorithme de Karatsuba a d'abord été conçu pour la multiplication de grands entiers et le découpage des opérandes s'est donc limité aux cas où la propagation de retenue est de complexité raisonnable. Cependant nous travaillons ici avec des polynômes et nous avons donc plus de liberté. Hanrot et Zimmermann ont proposé, dans un cadre légèrement différent — celui des séries formelles — de découper les opérandes selon la parité de leurs indices [HZ04]. Nous séparons, dans ce cas les polynômes en parties paire et impaire :

$$\begin{cases} A = A_E(X^2) + X A_O(X^2), \text{ et} \\ B = B_E(X^2) + X B_O(X^2). \end{cases}$$

Nous pouvons alors utiliser la même identité remarquable pour obtenir le produit avec seulement trois produits de taille moitié :

$$A \cdot B = (A_E B_E + X A_O B_O)(X^2) + X\left((A_E + A_O)(B_E + B_O) - A_E B_E - A_O B_O\right)(X^2).$$

Cette formulation a l'avantage de faciliter la reconstruction du produit complet ; en effet, les arbres d'addition conduisant à chaque coefficient du résultat sont de profondeur 3 au maximum et non 4 comme dans la variante classique. Cette amélioration pourra être sensible sur la latence d'un opérateur matériel pour la multiplication.

Nous avons introduit une nouvelle version de cette idée en découpant les opérandes en trois selon le résidu de leurs indices modulo 3 [Beu+08a]. Nous donnons cette variante dans l'Algorithme 8.3.

Algorithme 8.3 Algorithme de multiplication « à la Karatsuba » avec un découpage selon l'indice modulo 3.

Entrées : A, B deux polynômes.
Sortie : $C = A \cdot B$

1. $A \to A_0(X^3) + XA_1(X^3) + X^2A_2(X^3)$
2. $B \to B_0(X^3) + XB_1(X^3) + X^2B_2(X^3)$
3. $A'_0 \leftarrow A_1 + A_2\,; \quad B'_0 \leftarrow B_1 + B_2$
4. $A'_1 \leftarrow A_0 + A_2\,; \quad B'_1 \leftarrow B_0 + B_2$
5. $A'_2 \leftarrow A_1 + A_0\,; \quad B'_2 \leftarrow B_1 + B_0$
6. $p_0 \leftarrow A_0 \cdot B_0\,; \quad p'_0 \leftarrow A'_0 \cdot B'_0$
7. $p_1 \leftarrow A_1 \cdot B_1\,; \quad p'_1 \leftarrow A'_1 \cdot B'_1$
8. $p_2 \leftarrow A_2 \cdot B_2\,; \quad p'_2 \leftarrow A'_2 \cdot B'_2$
9. **return** $(p_0 + X(p'_0 - p_1 - p_2))(X^3)$
   $+X(p'_2 - p_0 - p_1 + Xp_2)(X^3)$
   $+X^2(p_1 + p'_1 - p_2 - p_0)(X^3)$

§ 2. Astuce de Montgomery. Montgomery décrit dans [Mon05, Section 2.3] une astuce permettant d'économiser un sous-produit lorsque l'algorithme de Karatsuba est appliqué à des opérandes ayant un nombre impair de coefficients. Supposons que nous multiplions des polynômes de taille $2n+1$. Classiquement, l'algorithme de Karatsuba nous donne :

$$A \cdot B = A_H B_H X^{2n} + ((A_H + A_L)(B_H + B_L) - A_H B_H - A_L B_L)\, X^n + A_L B_L,$$

où $A = A_H X^n + A_L$ et $B = B_H X^n + B_L$. Notons que le produit $A_L B_L$ est alors de taille n, les deux autres de taille $n+1$. En calculant $(A_H + XA_L)(B_H + XB_L)$ à la place de $(A_H + A_L)(B_H + B_L)$, la reconstruction du produit complet est toujours possible mais nous constatons qu'alors le coefficient de poids le plus faible de $(A_H + XA_L)(B_H + XB_L)$ est le même que pour $A_H B_H$. Il est donc inutile de calculer ce sous-produit deux fois et nous pouvons l'économiser.

8.2.3 Schéma d'évaluation-interpolation

L'année suivant la découverte de Karatsuba a vu arriver une technique bien plus générale pour la construction d'algorithme de multiplication : Toom & Cook ont proposé d'utiliser un schéma d'évaluation-interpolation [Too63 ; Coo66].

Soient A un polynôme de degré n et B un polynôme de degré m. Le produit $C = A \cdot B$ étant un polynôme de degré $n + m$, il suffit de connaître la valeur de C en $(n + m + 1)$ points pour pouvoir le reconstruire. Pour calculer C en ces points d'interpolation, il suffit de multiplier les évaluations respectives des polynômes A et B. Notons que pour que l'interpolation puisse fonctionner, il faut que les différences des points soient des éléments inversibles.

Afin d'obtenir un algorithme efficace, il faut utiliser des points d'interpolation suffisamment simples pour que l'évaluation ne demande pas de produit complet et que l'étape d'interpolation ne fasse pas intervenir de division coûteuse. C'est pourquoi il est courant d'utiliser au moins 0, 1, -1 et ∞ (évaluer en l'infini consiste à sélectionner le coefficient de poids fort).

L'interpolation consiste alors à inverser la matrice d'évaluation de C en les points d'inter-

polation :

$$\begin{pmatrix} c_0 \\ \\ \vdots \\ \\ c_{n+m} \end{pmatrix} = \begin{pmatrix} 1 & x_0 & \cdots & x_0^{n+m-1} & x_0^{n+m} \\ 1 & x_1 & \cdots & x_1^{n+m-1} & x_1^{n+m} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_{n+m-1} & \cdots & x_{n+m-1}^{n+m-1} & x_{n+m-1}^{n+m} \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} C(x_0) \\ C(x_1) \\ \vdots \\ C(x_{n+m-1}) \\ C(\infty) \end{pmatrix}.$$

Il est aisé de constater que l'algorithme de Karatsuba correspond à un schéma d'évaluation-interpolation en $0, 1$ et ∞. Cependant, la multiplication de polynômes de degré 2 est plus efficace par l'algorithme de Toom-Cook avec les points $0, 1, -1, 2$ et ∞ car elle ne demande que 5 produits. Ceci est vrai dès lors que ces points sont bien distincts dans l'anneau considéré et que leurs différences sont inversibles. Or, comme nous sommes particulièrement intéressés par la multiplication de polynômes sur $\mathbb{F}_2$ et $\mathbb{F}_3$, nous ne pourrons utiliser cette méthode directement et devrons rajouter des racines à notre corps de base pour pouvoir les utiliser comme points d'interpolation. C'est ce que nous faisons dans la section suivante.

8.2.4 Algorithme reposant sur le théorème des restes chinois

L'algorithme que nous présentons ici peut-être vu comme un schéma d'évaluation-interpolation où les points d'interpolation ne sont pas choisis dans le corps de base mais parmi les racines de polynômes bien choisis [Win80].

Nous cherchons ici à multiplier deux polynômes A et B de degrés respectifs n et m. Nous choisissons alors une famille $\{f_i\}$ de polynômes premiers entre eux et un entier ω tels que le degré de $F = \prod f_i$ soit égal à $n + m + 1 - \omega$. Pour effectuer le produit de A par B, il faudra alors calculer les sous-produits suivants :

$$C_i \equiv A \cdot B \pmod{f_i},$$

ce qui peut se faire en réduisant dans un premier temps A et B puis en multipliant les deux restes obtenus grâce à n'importe quelle formule valide. Il faut également calculer le produit

$$C_\infty \equiv A \cdot B \pmod{(X - \infty)^\omega},$$

qui est le produit court donnant les ω coefficients de tête du produit $A \cdot B$. Remarquons que le produit court de taille n est en général moins coûteux que le produit de deux polynômes de taille n. De même, nous n'avons besoin de connaître les C_i que modulo f_i, il se peut ainsi que ce calcul soit plus simple et demande moins de produits que la multiplication de deux polynômes de même taille. Ce point est discuté plus en détail dans les Sections 9.2.3 et 9.2.4 du chapitre suivant. L'exemple le plus notable est le cas où $f_i = X^k$, le produit modulo f_i est alors un produit court.

Nous montrons maintenant comment reconstruire le résultat $A \cdot B$ à partir des C_i et de C_∞. Il s'agit dans un premier temps d'une application classique du théorème des restes chinois pour calculer $C' = A \cdot B \bmod F$:

$$C' = \sum C_i \cdot \Big((F/f_i)^{-1} \bmod f_i \Big) \cdot (F/f_i).$$

Remarquons que C' est ainsi une combinaison linéaire des sous-produits que nous pouvons précalculer.

Pour obtenir $C = A \cdot B$, nous utilisons alors le fait que les ω coefficients de tête de C sont ceux de C_∞ et que $C' = C \mod F$. Il s'agit donc de calculer $C'' = C \bmod X^{n+m-\omega+1}$; or, nous avons que

$$C = C_\infty X^{n+m-\omega+1} + C'' \equiv C' \pmod F,$$

ce qui donne

$$C'' = C' + (C_\infty X^{n+m-\omega+1} \mod F),$$

car $\deg C' \leq n + m - \omega$.

Ainsi, un algorithme reposant sur le théorème des restes chinois aura la forme générale suivante : calcul des $A \bmod f_i$ et $B \bmod f_i$ (combinaisons linéaires des coefficients d'entrée), calcul des sous-produits en appelant éventuellement d'autres algorithmes de multiplication rapide, et enfin reconstruction du produit par combinaison linéaire.

Étant donné le degré des polynômes en entrée, il faut choisir les polynômes irréductibles f_i et le nombre de coefficients de poids fort à calculer séparément. Ces choix dépendent du nombre de polynômes irréductibles disponibles pour chaque degré et donc de la taille du corps sur lequel sont construits nos multiplicandes. Ils dépendent également de l'efficacité des algorithmes de multiplication utilisés pour les sous-produits et le produit court. Des tables sont disponibles pour les caractéristiques 2 et 3 dans [Fan+07 ; CÖ08 ; CÖ09].

Nous donnons en guise d'exemple un algorithme de multiplication sur une extension de degré 5 d'un corps de caractéristique 3. Cette multiplication a été utilisée dans une implémentation présentée dans [Est10], l'Algorithme 8.4 étant alors le meilleur algorithme connu pour cette opération. Nous avons pu depuis améliorer celui-ci grâce aux travaux présentés dans le chapitre suivant.

Algorithme 8.4 Algorithme de multiplication pour $\mathbb{F}_{3^5} \cong \mathbb{F}_3[\alpha]$ avec $\alpha^5 - \alpha + 1 = 0$ (12 mul., 53 add.)

Entrées : $A = a_0 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3 + a_4\alpha^4$ et $B = b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3 + b_4\alpha^4$ où $a_i, b_i \in \mathbb{F}_3$.
Sortie : $C = A \cdot B$

1. $a_5 \leftarrow a_0 + a_1$; $b_5 \leftarrow b_0 + b_1$; $a_6 \leftarrow a_2 + a_3$; $b_6 \leftarrow b_2 + b_3$
2. $a_7 \leftarrow a_2 - a_3$; $b_7 \leftarrow b_2 - b_3$; $a_8 \leftarrow a_0 + a_4$; $b_8 \leftarrow b_0 + b_4$
3. $a_9 \leftarrow a_4 + a_5$; $b_9 \leftarrow b_4 + b_5$
4. $p_0 \leftarrow a_0 \cdot b_0$; $p_1 \leftarrow a_1 \cdot b_1$; $p_2 \leftarrow a_4 \cdot b_4$; $p_3 \leftarrow a_5 \cdot b_5$
5. $p_4 \leftarrow (a_1 - a_3) \cdot (b_1 - b_3)$; $p_5 \leftarrow (a_1 - a_6) \cdot (b_1 - b_6)$
6. $p_6 \leftarrow (a_2 - a_8) \cdot (b_2 - b_8)$; $p_7 \leftarrow (a_6 + a_9) \cdot (b_6 + b_9)$
7. $p_8 \leftarrow (a_6 - a_9) \cdot (b_6 - b_9)$; $p_9 \leftarrow (a_0 - a_4 + a_7) \cdot (b_0 - b_4 + b_7)$
8. $p_{10} \leftarrow (a_1 - a_7 - a_8) \cdot (b_1 - b_7 - b_9)$; $p_{11} \leftarrow (a_3 - a_4 + a_5) \cdot (b_3 - b_4 + b_5)$
9. $t_0 \leftarrow p_1 - p_3$; $t_1 \leftarrow p_7 + p_{10}$; $t_2 \leftarrow p_7 - p_{10}$
10. $t_3 \leftarrow p_2 + p_4$; $t_4 \leftarrow p_8 - p_6$; $t_5 \leftarrow p_{11} - t_0$
11. $c_0 \leftarrow t_4 - t_0 - t_2 - p_4$; $c_1 \leftarrow p_2 - p_0 - p_9 - t_1 - t_5$
12. $c_2 \leftarrow p_5 - p_8 - t_3 - t_5$; $c_3 \leftarrow t_2 - t_3 + t_4$
13. $c_4 \leftarrow p_4 - p_0 - p_6 + t_1$
14. **return** $C = c_0 + c_1\alpha + c_2\alpha^2 + c_3\alpha^3 + c_4\alpha^4$

8.2.5 Algorithmes *ad hoc* de Montgomery

Enfin, Montgomery a fait la découverte de formules *ad-hoc* en 2005 pour la multiplication de polynômes de degré 4, 5 et 6 [Mon05]. Ces formules utilisent respectivement 13, 17 et 22 produits de coefficients. Cette découverte est étonnante par le fait que les formules de Montgomery n'utilisent aucune des structures connues jusque là. Elle sont en effet le fruit d'une recherche exhaustive. Nous avons repris les travaux de Montgomery et étendu son algorithme de recherche ainsi qu'amélioré sa complexité. Cette étude fait l'objet du chapitre suivant.

Chapitre 9

Algorithme pour la recherche de formules de multiplication

Ce chapitre présente un algorithme pour la recherche de formules de multiplications, et plus généralement pour le calcul de toute application bilinéaire. Ces travaux sont le fruit d'une collaboration avec Razvan Barbulescu, Jérémie Detrey et Paul Zimmermann ; ils ont été publiés à la conférence WAIFI 2012 [Bar+12].

Pour calculer une application bilinéaire, il faut faire un certain nombre de sommes puis effectuer des produits avant de reconstituer le résultat grâce à des additions. Partant de cette constatation, nous avons cherché à optimiser le calcul des applications bilinéaires en trouvant des formules minimisant le nombre de produits. En effet, la multiplication est, en général, l'opération la plus coûteuse.

Ce problème est bien connu et dépasse le champ de la communauté des arithméticiens. Il trouve ses fondements théoriques dans les études de complexité algébrique. Il est alors communément appelé le **problème du rang bilinéaire** [BCS97, Chapitre 14].

Les premières réponses pratiques au problème du calcul du rang bilinéaire sont arrivées en 1962 quand Karatsuba a montré que seules 3 multiplications au lieu de 4 suffisent à calculer le produit de deux polynômes de degré 1 [KO63] et en 1963 lorsque Toom & Cook montrèrent qu'il en suffit de 5 pour deux polynômes de degré 2. Dès lors, trouver des formules pour la multiplication a été une course très productive dont nous avons présenté une partie du fruit au chapitre précédent. Retenons ici les schémas en évaluation-interpolation et les formules reposant sur le théorème du reste chinois.

Cependant, toutes ces constructions sont *ad-hoc* et reposent sur une structure mathématique. À partir de cette constatation, Montgomery a conduit une recherche systématique de formules pour la multiplication de polynômes de taille 5, 6 et 7 en 2005 [Mon05]. Cette idée originale de recherche systématique lui a ainsi permis de trouver de nouvelles formules ne suivant aucune construction connue. Fan et Hasan ont utilisé ces nouvelles formules pour améliorer la multiplication pour des tailles supérieures [FH07], montrant ainsi que toute amélioration des algorithmes de multiplication pour les petites tailles peut conduire à des améliorations pour les tailles supérieures du fait de l'utilisation récursive de ceux-ci. L'idée de recherche systématique a été réutilisée deux ans plus tard, en 2007, par Chung et Hasan qui ont trouvé ainsi des formules asymétriques pour le carré sur $\mathbb{Q}$ [CH07]. Enfin, Oseledets proposa en 2008 un autre algorithme de recherche systématique de formules pour la multi-

plication de deux polynômes de taille n modulo X^n et $X^n - 1$ [Oseo8]. Cependant, conduire une recherche exhaustive est une tâche qui devient rapidement hors de portée des puissances de calcul actuellement disponibles et les précédents travaux ont dû soit limiter l'espace de recherche, soit faire une exploration heuristique pour pouvoir mener à bien leurs calculs.

Nous proposons ici une formalisation différente du problème qui a pour avantage d'en capturer toute la généralité (nous ne nous limitons pas à la multiplication mais considérons n'importe quelle application bilinéaire) et qui nous permet une recherche exhaustive des formules.

Nous organisons ce chapitre de la manière suivante : nous commençons par fournir un cadre formel au problème (Section 9.1), puis nous montrons comment quelques applications concrètes rentrent dans ce formalisme (Section 9.2). Enfin, nous transformons le problème en un problème d'algèbre linéaire (Section 9.3) avant d'en donner la résolution (Section 9.4) et de montrer comment reconstruire les formules (Section 9.5). Nous donnons alors les résultats de nos expérimentations (Section 9.6).

9.1 Formalisation du problème

Nous cherchons ici à montrer une formalisation du **problème de rang bilinéaire** qui puisse englober une grande variété de situations. Parmi celles que nous considérerons, il y a bien évidemment la multiplication de polynômes, mais aussi la multiplication dans une extension de corps ou de matrices.

Soit K un corps. Afin d'obtenir toute la généralité des formules que nous obtiendrons, nous ne considérons pas uniquement les applications bilinéaires sur K mais également leur extension sur n'importe quelle K-algèbre $\mathbb{K}$, non nécessairement commutative. En effet, la même formule permet d'évaluer une application bilinéaire quelle que soit l'algèbre $\mathbb{K}$ considérée. Martin Albrecht fournit un bel exemple de ce fait dans [Alb11] : il utilise ainsi les formules de multiplication pour les polynômes de n termes sur $\mathbb{F}_2$ dans le but d'évaluer un produit de matrices $r \times r$ sur $\mathbb{F}_{2^n}$; il considère ainsi la $\mathbb{F}_2$-algèbre des matrices $r \times r$ et représente $\mathbb{F}_{2^n}^{r\times r}$ comme $\mathbb{F}_2^{r\times r}[X]/\langle f(X) \cdot \mathbb{F}_2^{r\times r}[X]\rangle$ où f est un polynôme irréductible de degré n sur $\mathbb{F}_2$. Il a ainsi pu utiliser des formules efficaces, notamment celles de Montgomery (Section 8.2.5).

Cet exemple nous permet de distinguer deux catégories d'opérations :

— les **multiplications complètes** de deux éléments dérivés des entrées de l'application bilinéaire, ces éléments appartenant à l'algèbre $\mathbb{K}$;

— les additions ou les multiplications par un scalaire $\lambda \in K$.

Les multiplications complètes sont considérées bien plus coûteuses que les additions ou les multiplications scalaires. Cela correspond bien aux cas pratiques et plus particulièrement quand l'algèbre $\mathbb{K}$ est plus grande que le corps K. L'idée intuitive pour réduire le coût du calcul d'une application bilinéaire est ainsi de minimiser le nombre de multiplications complètes et c'est ce que propose l'algorithme que nous présentons.

Étant donné un corps K, nous notons $\mathcal{B}(K^n, K^m; K^\ell)$ l'ensemble des applications bilinéaires $K^n \times K^m \to K^\ell$. Pour $\phi \in \mathcal{B}(K^n, K^m; K^\ell)$, le **problème du rang bilinéaire** est de trouver une formule évaluant ϕ en utilisant un nombre minimal de multiplications complètes ; ce nombre est appelé le **rang bilinéaire** de l'application ϕ.

Chaque application bilinéaire ϕ peut être écrite comme un ℓ-uplet $\phi = (\gamma_0, \ldots, \gamma_{\ell-1})$ de formes bilinéaires $\gamma_i \in \mathcal{B}(K^n, K^m; K)$. Nous cherchons alors à représenter les formes bilinéaires. Soit $\gamma \in \mathcal{B}(K^n, K^m; K)$, celle-ci peut s'écrire $\gamma : \mathbf{a}, \mathbf{b} \mapsto \sum_{i,j} \gamma_{i,j} a_i b_j$ où $\mathbf{a} =$

$(a_0, \ldots, a_{n-1}) \in K^n$ et $\mathbf{b} = (b_0, \ldots, b_{m-1}) \in K^{m-1}$. Cette forme γ est donc entièrement déterminée par les coefficients $\gamma_{i,j} \in K$. De ce fait, $\mathcal{B}(K^n, K^m; K)$ peut être représenté comme un K-espace vectoriel V de dimension nm. De même, une application bilinéaire $\phi \in \mathcal{B}(K^n, K^m; K^\ell)$ peut être représentée comme un ℓ-uplet d'éléments de V.

Les formes bilinéaires de rang 1 sont celles qui peuvent s'évaluer avec une seule multiplication complète : étant donné $\gamma \in \mathcal{B}(K^n, K^m; K)$,

$$\gamma \text{ est de rang 1} \Leftrightarrow \left(\begin{array}{c} \exists \alpha, \beta \in \mathrm{Hom}(K^n, K) \times \mathrm{Hom}(K^m, K)/ \\ \forall \mathbf{a}, \mathbf{b} \in K^n \times K^m, \gamma(\mathbf{a}, \mathbf{b}) = \alpha(\mathbf{a}) \cdot \beta(\mathbf{b}) \end{array} \right),$$

où $\mathrm{Hom}(K^n, K)$ dénote l'ensemble des formes linéaires de K^n dans K. Remarquons tout d'abord que l'évaluation des formes linéaires α et β ne demande que des additions et des multiplications par des scalaires.

Parmi les formes bilinéaires de rang 1 nous pouvons extraire une base de l'espace des formes bilinéaires : $e_{i,j} : \mathbf{a}, \mathbf{b} \mapsto a_i b_j$. La représentation d'une forme ou d'une application bilinéaire sur la base $(e_{i,j})$ permet ainsi d'obtenir une borne supérieure sur le rang de celles-ci ; cependant, nous cherchons le nombre minimal de multiplications à effectuer.

Le cas des formes bilinéaires est simple ; en effet, le rang bilinéaire de $\gamma = \sum_{i,j} \gamma_{i,j} e_{i,j}$ est le rang de la matrice $(\gamma_{i,j})_{0 \leq i < n, 0 \leq j < m}$. Cependant, le cas des applications bilinéaires est beaucoup plus complexe ; l'intuition qui permet de s'en rendre compte est que les différentes formes bilinéaires peuvent partager des produits.

Définition 9.1 (Problème du rang bilinéaire [BCS97])**.** Soit $\mathcal{G} \subset V$ un ensemble de $n \times m$ formes bilinéaires de rang 1 et $\mathcal{T} = \{t_0, \ldots, t_{\ell-1}\} \subseteq \mathrm{Span}\, \mathcal{G}$ un ensemble de formes bilinéaires cibles. Le problème du rang bilinéaire consiste à générer tous les éléments de $\mathcal{T}$ par des combinaisons K-linéaires d'un sous-ensemble de taille k minimale des générateurs. Une fois le rang k déterminé, il est aussi intéressant d'énumérer toutes les solutions à k générateurs.

Partant de cette définition, nous pouvons supposer sans perte de généralité que l'ensemble des cibles est linéairement indépendant car il est alors possible de reconstruire linéairement des cibles liées sans avoir à rajouter de générateurs. Nous supposerons donc dans la suite que $\mathcal{T}$ est une famille libre.

Nous avons aussi cherché à étudier le calcul de carré qui ne correspond plus à une application bilinéaire mais à un ensemble de formes quadratiques. Le cadre formel que nous avons décrit s'adapte parfaitement à ce cas : soit $\sigma \in \mathcal{Q}(K^n; K)$ une forme quadratique ; elle est uniquement déterminée par ses coefficients $\sigma_{i,j}$ tels que $\sigma : \mathbf{a} \mapsto \sum_{0 \leq i \leq j < n} \sigma_{i,j} a_i \cdot a_j$. Ainsi, $\mathcal{Q}(K^n; K)$ forme un espace vectoriel sur K de dimension $n(n+1)/2$ qu'il suffira de prendre comme l'espace V de la Définition 9.1. Les formes de rang 1 seront alors similaires :

$$\sigma \text{ est de rang 1} \Leftrightarrow \left(\begin{array}{c} \exists \alpha, \alpha' \in \mathrm{Hom}(K^n, K)/ \\ \forall \mathbf{a} \in K^n, \sigma(\mathbf{a}) = \alpha(\mathbf{a}) \cdot \alpha'(\mathbf{a}) \end{array} \right).$$

Notons que la notion de rang ne nous permet pas ici de distinguer les carrés ($\alpha = \alpha'$) des multiplications.

Le rang bilinéaire des applications quadratiques[1] est en fait un problème plus général que celui des applications bilinéaires. En effet, une forme $n \times m$-bilinéaire peut être vue comme une

1. Bien que le terme d'« application quadratique » ne soit pas courant, nous l'utiliserons pour désigner un ℓ-uplet de n-formes quadratiques dont l'ensemble sera noté : $\mathcal{Q}(K^n; K^\ell)$.

$(n+m)$-forme quadratique où certains produits ne sont pas utilisés (ceux entre les n premières variables et ceux entre les m dernières).

Le problème du rang bilinéaire est prouvé être NP-dur par réduction à la décomposition d'un tenseur d'ordre 3 [BCS97, section 14.2] connu pour être également NP-dur [Hås90]. Ainsi, nous n'espérons pas un algorithme polynomial pour le calcul du rang bilinéaire, mais nous cherchons ici un algorithme efficace pour les petites instances du problème.

9.2 Quelques exemples d'applications

Nous présentons ici quelques applications bilinéaires et quadratiques que nous avons utilisées lors de nos calculs. Nous n'écrivons pas ici ces applications complètement car nous pouvons nous contenter d'écrire leur application aux vecteurs **a** et **b**.

9.2.1 Multiplication de polynômes

La première application est le produit d'un polynôme $A = \sum a_i X^i$ à n termes par un polynôme $B = \sum b_i X^i$ à m termes sur un corps K. L'ensemble des $n+m-2$ cibles est alors

$$\mathcal{T} = \{a_0b_0, a_0b_1 + a_1b_0, \ldots, a_{n-1}b_{m-1}\},$$

et nous utiliserons l'ensemble de générateurs suivant :

$$\mathcal{G} = \left\{ \left(\sum_{i=0}^{n-1} \alpha_i a_i\right) \cdot \left(\sum_{j=0}^{m-1} \beta_j b_j\right) \;\middle|\; \alpha_i \in K, \beta_j \in K \right\}.$$

Exemple 9.2 (Produit de polynômes de taille 2×3). Par la suite, nous nous référerons souvent à l'exemple du produit polynomial de taille 2×3. L'algorithme naïf consiste à calculer tous les 6 produits partiels. Cependant, il est possible de n'utiliser que 5 produits en utilisant la formule de Karatsuba :

$$A \cdot B = g_3X^3 + (g_1 + g_2)X^2 + (g_4 - g_2 - g_0)X + g_0,$$

avec $g_0 = a_0b_0$, $g_1 = a_0b_2$, $g_2 = a_1b_1$, $g_3 = a_1b_2$ et $g_4 = (a_0+a_1)(b_0+b_1)$. Comme le produit $A \cdot B$ contient 4 termes, un schéma d'évaluation-interpolation (Section 8.2.3) permettra donc de faire le calcul avec seulement 4 produits dès lors que K contient au moins 3 éléments [2]. Comme les 4 termes du résultat sont bien linéairement indépendants, 4 est effectivement le nombre de produits optimal dès que $\#K \geq 3$; cependant, la question se pose lorsque K est le corps à deux éléments ($\mathbb{F}_2$). Pour étudier ce problème sur $\mathbb{F}_2$, nous considérerons l'ensemble de tous les produits (formes bilinéaires de rang 1) :

$$\begin{aligned}
\mathcal{G} = \{ & a_0 \cdot b_0, && a_1 \cdot b_0, && (a_0 + a_1) \cdot b_0, \\
& a_0 \cdot b_1, && a_1 \cdot b_1, && (a_0 + a_1) \cdot b_1, \\
& a_0 \cdot (b_0 + b_1), && a_1 \cdot (b_0 + b_1), && (a_0 + a_1) \cdot (b_0 + b_1), \\
& a_0 \cdot b_2, && a_1 \cdot b_2, && (a_0 + a_1) \cdot b_2, \\
& a_0 \cdot (b_0 + b_2), && a_1 \cdot (b_0 + b_2), && (a_0 + a_1) \cdot (b_0 + b_2), \\
& a_0 \cdot (b_1 + b_2), && a_1 \cdot (b_1 + b_2), && (a_0 + a_1) \cdot (b_1 + b_2), \\
& a_0 \cdot (b_0 + b_1 + b_2), && a_1 \cdot (b_0 + b_1 + b_2), && (a_0 + a_1) \cdot (b_0 + b_1 + b_2)\}.
\end{aligned}$$

2. Le point à l'infini fournit un des quatre points d'interpolation (ici le produit a_1b_2).

9.2.2 Carré de polynômes

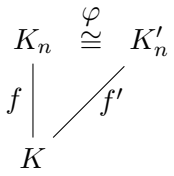
Nous pouvons également étudier les applications quadratiques telles que le carré de polynômes dans notre cadre formel. Par exemple nous pouvons chercher des formules pour le carré d'un polynôme $a_1X + a_0$ sur $\mathbb{F}_3$. L'ensemble des générateurs que les formules pourront utiliser est alors :

$$\begin{aligned}\mathcal{G} = \{&a_0 \cdot a_0,\\ &a_0 \cdot a_1, && a_1 \cdot a_1,\\ &a_0 \cdot (a_0 + a_1), && a_1 \cdot (a_0 + a_1), && (a_0 + a_1) \cdot (a_0 + a_1),\\ &a_0 \cdot (a_0 - a_1), && a_1 \cdot (a_0 - a_1), && (a_0 + a_1) \cdot (a_0 - a_1), && (a_0 - a_1) \cdot (a_0 - a_1)\}.\end{aligned}$$

9.2.3 Multiplication dans une extension de corps

Le problème de multiplication dans une extension de corps de degré n est très similaire à la multiplication $n \times n$ de polynômes puisque qu'il est possible de représenter les éléments de l'extension par des polynômes modulo un polynôme irréductible de degré n. Cependant, lors de la réduction, il peut arriver que l'une ou plusieurs des multiplications complètes permettant d'obtenir le produit de polynômes ne soient plus utiles [CÖ10]. Aussi la multiplication dans une extension de corps est un problème différent dont le rang bilinéaire est éventuellement plus faible pour une taille n donnée.

Soit un corps K et K_n une extension de degré n de K construite grâce au polynôme irréductible f. Nous considérons la multiplication sur K_n : c'est une application K-bilinéaire. Nous commençons par montrer que le rang bilinéaire est indépendant du choix du polynôme irréductible f. Soit donc $f' \in K[X]$ un autre polynôme irréductible de degré n et $K'_n = K[X]/(f'(x))$. Les corps K_n et K'_n sont alors isomorphes, notons φ cet isomorphisme.



Le produit dans K'_n est également une application K-bilinéaire et nous avons

$$\times_{K'_n} = \varphi \circ (\times_{K_n}) \circ (\varphi^{-1}, \varphi^{-1}). \tag{9.1}$$

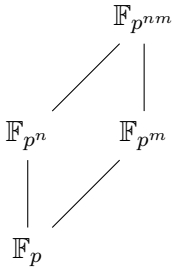
Soit $(g_i : \mathbf{a}, \mathbf{b} \mapsto (\sum \alpha_{i,j} a_j)(\sum \beta_{i,j} b_j))_{0 \leq i < k}$ l'ensemble des produits d'une formule pour la multiplication dans K_n. Nous savons alors qu'il existe une combinaison K-linéaire telle que

$$\times_{K_n} = \left(\sum_j \gamma_{i,j} g_j \right)_{0 \leq i < n}.$$

En posant $g'_i = g_i \circ (\varphi^{-1}, \varphi^{-1})$, nous aurons bien une solution à k produits pour la multiplication dans K'_n. En effet, φ est également un isomorphisme d'espaces vectoriels ; par conséquent, les combinaisons linéaires permettant d'obtenir les g_i et de reconstruire le résultat seront bien transformées en d'autres combinaisons linéaires. L'Équation (9.1) provenant de l'isomorphisme entre les corps K_n et K'_n nous donne alors la validité de la formule obtenue.

Comme nous avons montré que le rang bilinéaire du produit dans une extension de corps ne dépend pas du polynôme choisi pour la représenter, nous pouvons nous contenter de chercher les formules pour une seule représentation.

Nous utiliserons dans la suite ces formules pour des corps finis de la forme $\mathbb{F}_{p^{nm}}$ avec n et m premiers entre eux. Nous chercherons les formules de multiplications dans $\mathbb{F}_{p^n}$ et les utiliserons dans $\mathbb{F}_{p^{nm}}$, les coefficients a_i, b_i seront alors des éléments du corps $\mathbb{F}_{p^m}$.



9.2.4 Produit court

Il peut être parfois utile de considérer le produit de deux polynômes non pas modulo un polynôme irréductible mais modulo un polynôme quelconque. Dans ce cas, nous n'avons plus la structure d'extension de corps et il faudra faire une recherche de formules pour chaque module considéré. Nous nous intéressons ici au cas particulier de la multiplication de deux polynômes de taille n modulo X^n, il s'agit du produit court introduit par Mulders en 2000 [Mul00]. Ce produit est d'un intérêt très particulier car il est utilisé dans un grand nombre d'algorithmes liés aux calculs sur les nombres flottants ou en virgule fixe. Il est également utilisé dans les algorithmes de multiplication reposant sur le théorème des restes chinois (voir la Section 8.2.4).

Dans le formalisme que nous développons dans ce chapitre, nous aurons pour ensemble des cibles la famille à n éléments suivante :

$$\mathcal{T} = \{a_0b_0, a_0b_1 + a_1b_0, \ldots, a_0b_{n-1} + \cdots + a_{n-1}b_0\}.$$

9.2.5 Multiplication de matrices

Notre cadre nous permet également de considérer la multiplication de matrices. Nous nous restreignons ici aux matrices carrées par souci de concision mais nous aurions également pu considérer le produit des matrices $n \times m$ par les matrices $m \times l$. Les matrices de taille $n \times n$ sur K forment un espace vectoriel de dimension n^2. Si nous considérons les matrices sur $K = \mathbb{F}_p$, l'ensemble des générateurs pour effectuer un produit est alors de cardinal $\#\mathcal{G} = (\frac{p^{n^2}-1}{p-1})^2$.

Le rang bilinéaire est connu pour les matrices de taille 2×2, il est égal à 7 : Strassen a exhibé en 1969 une formule avec 7 produits [Str69] et Winograd a montré deux ans plus tard que cette formule est optimale [Win71]. Cependant, nous n'avons que des bornes pour les matrices 3, le rang est compris entre 19 [Blä03] et 23 [CBH11]. Afin de considérer ce problème sur $\mathbb{F}_2$, nous devrons travailler avec $\#\mathcal{G} = 261\,121$. Nous verrons plus tard que la complexité de notre algorithme de recherche exhaustive de formule dépend fortement du nombre de générateurs ; nous n'avons donc pu améliorer les bornes pour la multiplication de matrices 3×3.

9.2.6 Variantes creuses

L'un des avantages de notre formalisme est qu'il permet d'appréhender n'importe quelle application bilinéaire. En particulier, nous pouvons donc considérer des variantes creuses des différentes multiplications que nous avons envisagées jusqu'alors. Il arrive régulièrement, lors de l'implémentation d'un algorithme arithmétique, que nous connaissions la forme des éléments que nous avons à multiplier. Par exemple, il se peut que l'un de nos opérandes soit un polynôme creux dont les coefficients suivent une structure particulière :

$$A = a_0 + a_1X + a_2X^2 + (a_1 + a_2)X^4,$$

ou encore une matrice particulière.

Il est alors difficile de trouver une formule optimale en nombre de multiplications complètes, ou tout du moins une bonne formule. Notre algorithme permettra alors à l'implémenteur de chercher des formules utiles à son problème particulier.

Nous illustrons cette recherche de formules par un exemple tiré du calcul de couplage η_T sur une courbe elliptique supersingulière de caractéristique 2. À chaque itération de la boucle de Miller, il faut multiplier

$$G = g_0 + g_1 s + t,$$

à

$$F = f_0 + f_1 s + f_2 t + f_3 st,$$

où $s^2+s+1 = 0$, $t^2+t+s = 0$ et les coefficients appartiennent à un corps de caractéristique 2. Nous commençons alors par retirer la partie linéaire en étudiant le produit de $G - t$ par F. Nous avons alors une application bilinéaire de $\mathcal{B}((\mathbb{F}_2)^2, (\mathbb{F}_2)^4; (\mathbb{F}_2)^4)$ dont l'ensemble des cibles qui correspond est alors :

$$\mathcal{T} = \{f_0g_0 + f_1g_1, f_0g_1 + g_1f_0 + f_1g_1, f_2g_0 + f_3g_1, f_2g_1 + g_1f_2 + f_3g_1\}.$$

Notre algorithme a permis de vérifier que la formule utilisée précédemment dans la littérature (Alg. 11.10) est bien optimale.

9.3 Expression du problème en terme d'algèbre linéaire

Nous nous concentrons sur le cas des applications bilinéaires bien que la construction soit très similaire pour les applications quadratiques. Nous rappelons que nous notons V le K-espace vectoriel de dimension nm isomorphe à l'ensemble des formes $n \times m$-bilinéaires $\mathcal{B}(K^n, K^m; K)$. Chaque forme cible ($\in \mathcal{T}$) et chaque produit faisant partie de l'ensemble des générateurs $\mathcal{G}$ correspond alors à un vecteur dans V. Cette conversion étant réalisée par le morphisme suivant :

$$\begin{array}{rcl} \mathcal{B}(K^n, K^m; K) & \longrightarrow & V \\ (\mathbf{a}, \mathbf{b} \mapsto a_ib_j) & \longmapsto & e_{im+j}, \end{array}$$

où e_k est le k-ème vecteur de la base canonique de V. Par un abus de notation, nous n'écrirons pas ce morphisme explicitement dans la suite du chapitre et écrirons a_ib_j à la place de e_{im+j}.

Nous pouvons alors exprimer le problème du rang bilinéaire en termes d'algèbre linéaire : il s'agit de trouver $\mathcal{W}$ de cardinal minimal k tel que

— $\mathcal{W} \subset \mathcal{G}$ (nous cherchons un ensemble de générateurs) et
— $\mathcal{T} \subset W = \operatorname{Span} \mathcal{W}$ (qui génère linéairement les formes cibles).

Dans un souci de clarté, nous notons en lettres cursives ($\mathcal{A}$) les ensembles de vecteurs de V et en lettres droites (A) les sous-espaces de V engendrés par ceux-ci ($\mathcal{A} = \operatorname{Span} A$).

Cette expression du problème en termes d'algèbre linéaire permet d'utiliser une représentation matricielle plus pratique pour les calculs : à chaque forme linéaire correspond un vecteur ligne et à chaque ensemble une matrice. Notons que les sous-espaces peuvent être également représentés par une matrice, celle des vecteurs d'une de ses bases ; nous choisirons par la suite une forme échelon pour ceux-ci, dans un souci d'efficacité des calculs.

Si nous reprenons notre exemple de la multiplication polynomiale de taille 2×3, nous aurons alors

$$\mathcal{T} = \begin{array}{c} \begin{array}{cccccc} a_0b_0 & a_0b_1 & a_0b_2 & a_1b_0 & a_1b_1 & a_1b_2 \end{array} \\ \left(\begin{array}{cccccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right) \end{array} \begin{array}{l} a_0b_0 \\ a_0b_1 + a_1b_0 \\ a_0b_2 + a_1b_1 \\ a_1b_2 \end{array},$$

et

$$\mathcal{G} = \begin{array}{c} \begin{array}{cccccc} a_0b_0 & a_0b_1 & a_0b_2 & a_1b_0 & a_1b_1 & a_1b_2 \end{array} \\ \left(\begin{array}{cccccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{array} \right) \end{array} \begin{array}{l} a_0b_0 \\ a_1b_0 \\ (a_0 + a_1)b_0 \\ \vdots \\ a_1(b_0 + b_1 + b_2) \\ (a_0 + a_1)(b_0 + b_1 + b_2) \end{array}.$$

9.4 Algorithmes de résolution

Nous cherchons dans ce travail non seulement à trouver le rang bilinéaire (c'est-à-dire le nombre minimal de multiplications complètes à effectuer) mais aussi à produire les formules qui correspondent. Ainsi, nous avons conçu un algorithme qui énumère toutes les formules utilisant k produits où k est une des entrées de l'algorithme. Pour trouver le rang, il suffira de lancer plusieurs fois l'algorithme en faisant croître k ; le premier k pour lequel des formules sont trouvées est alors le rang bilinéaire.

9.4.1 Approche naïve

Un premier algorithme naïf consiste à énumérer tous les sous-ensembles $\mathcal{W}$ à k éléments de l'ensemble des générateurs $\mathcal{G}$ et à tester si le sous-espace $W = \operatorname{Span} \mathcal{W}$ recouvre l'ensemble des vecteurs cibles $\mathcal{T}$.

Pour faire ce test, nous commençons par constater qu'il suffit de vérifier que $T \subset W$, avec $T = \operatorname{Span} \mathcal{T}$; cette condition est vérifiable plus efficacement car nous pouvons alors représenter T sous la forme d'une matrice échelon. Soit $\mathcal{W} = \{g_1, \ldots, g_k\}$. Tester $T \subset W$ consistera à réduire chaque vecteur ligne d'une base échelon de T dans la matrice échelon W ; si tous les vecteurs de T se réduisent à 0, nous aurons alors l'inclusion $T \subset W$.

Dans un souci de simplicité, nous n'analysons que la complexité combinatoire de nos algorithmes, c'est-à-dire le nombre d'appels à la routine du test de l'inclusion $T \subset W$. Ainsi,

la complexité de l'algorithme naïf est

$$\binom{\#\mathcal{G}}{k}.$$

Par exemple, chercher les formules optimales pour la multiplication sur $\mathbb{F}_2[X]$ de taille 2×3 demande de tester $\binom{21}{5} = 20\,349$ sous-ensembles $\mathcal{W}$.

9.4.2 Une première idée d'amélioration de l'algorithme

Le principal inconvénient de l'approche naïve précédente est que nous testons chaque sous-ensemble $\mathcal{W}$ alors que certains d'entre eux génèrent le même sous-espace W. Intuitivement, nous pouvons constater que c'est le cas pour une grande partie des sous-ensembles $\mathcal{W}$ car le nombre de générateurs considérés ($\#\mathcal{G}$) est en général bien plus grand que la dimension de l'espace V des formes bilinéaires ; il existe donc un grand nombre de combinaisons linéaires nulles de ces générateurs.

La première illustration de ce fait peut se remarquer dans l'article [Mon05] de Montgomery : il donne un ensemble de formules pour la multiplication de deux polynômes de taille 3 sur $\mathbb{Q}$ sous la forme suivante

$$\begin{aligned}(a_0+a_1X+a_2X^2)(b_0+b_1X+b_2X^2) = a_0b_0(C+1-X-X^2)+\\ \begin{array}{lcll} a_1b_1(C-X+X^2-X^3) & + & a_2b_2(C-X^2-X^3+X^4) & + \\ (a_0+a_1)(b_0+b_1)(-C+X) & + & (a_0+a_2)(b_0+b_2)(-C+X^2) & + \\ (a_1+a_2)(b_1+b_2)(-C+X^3) & + & (a_0+a_1+a_2)(b_0+b_1+b_2)C, & \end{array}\end{aligned} \tag{9.2}$$

où C peut-être un polynôme choisi arbitrairement de $\mathbb{Q}[X]$. Dans notre formulation en termes d'algèbre linéaire, cela revient à dire que chaque sous-ensemble de taille 6 parmi les 7 générateurs suivants :

$$\begin{aligned}\mathcal{G} = \{&a_0b_0, a_1b_1, a_2b_2, (a_0+a_1)(b_0+b_1), (a_0+a_2)(b_0+b_2),\\ &(a_1+a_2)(b_1+b_2), (a_0+a_1+a_2)(b_0+b_1+b_2)\},\end{aligned}$$

fournit une solution. Cependant, nous pouvons constater que ces 7 générateurs sont linéairement dépendants :

$$\begin{aligned}(a_0+a_1+a_2)(b_0+b_1+b_2) = (a_0+a_1)(b_0+b_1) + (a_0+a_2)(b_0+b_2)\\ + (a_1+a_2)(b_1+b_2) - a_0b_0 - a_1b_1 - a_2b_2,\end{aligned}$$

et que chacun des sous-ensembles de 6 générateurs donne le même sous-espace vectoriel. Il suffit donc de tester une seule fois ce sous-espace vectoriel pour constater qu'il permet de produire des solutions.

Nous proposons donc de prendre avantage de cette redondance en n'énumérant non pas les sous-ensembles $\mathcal{W}$ mais les sous-espaces W. Les conditions que W doit vérifier pour fournir une solution sont alors :

(*i*) $T \subset W$, c'est-à-dire que l'espace cible T est recouvert par W ;

(*ii*) $\mathrm{Span}\,(W \cap \mathcal{G}) = W$, c'est-à-dire que W peut être généré uniquement par des éléments de $\mathcal{G}$; et

(*iii*) $\dim W = k$, c'est-à-dire que seulement k générateurs suffisent.

Nous pouvons alors déduire un algorithme récursif pour explorer tous les sous-espaces W solutions. Il consiste à partir de l'espace nul et ajouter progressivement des générateurs de $\mathcal{G}$ qui ne sont pas déjà dans l'espace W en cours de construction (voir Algorithme 9.1).

Algorithme 9.1 Énumération de tous les sous-espaces W de dimension k générés par des éléments de $\mathcal{G}$ seulement et qui contiennent l'espace cible T. (Premier algorithme)

Entrées : V un espace vectoriel, $\mathcal{G}$ une famille finie de V, T un sous-espace inclus dans $\operatorname{Span}\mathcal{G}$ et un entier k tel que $\dim T \leq k \leq \operatorname{rg}\mathcal{G}$.

Sortie : L'ensemble $\mathcal{S}$ de tous les sous-espaces W de V tels que $T \subset W$, $\operatorname{Span}(W \cap \mathcal{G}) = W$ et $\dim W = k$.

```
1.  procedure expand_subspace(W, H)
2.      if dim W = k and T ⊂ W then
3.          W est solution
4.      else if dim W < k then
5.          H ← H \ W
6.          while H ≠ ∅ do
7.              Dépiler un élément g de H
8.              expand_subspace(W ⊕ Span(g), H)
9.          end while
10.     end if
11. end procedure
12. expand_subspace({0}, G)
```

Notons que cet algorithme ne permet pas d'assurer que nous testons une seule fois chaque espace de dimension k généré uniquement par des éléments de $\mathcal{G}$ mais il évite tout de même d'avoir à énumérer toutes les bases dans $\mathcal{G}$.

L'analyse de complexité de cet algorithme de parcours de l'arbre des sous-espaces vectoriels générés par des éléments de $\mathcal{G}$ est complexe et dépend fortement de la forme de $\mathcal{G}$. Aussi nous ne pouvons borner la complexité de cet algorithme uniquement par la même borne que celle de l'approche naïve :

$$\binom{\#\mathcal{G}}{k}.$$

Cependant, cette approche apporte déjà une amélioration en pratique pour notre algorithme de recherche. En effet, pour notre exemple de la multiplication de taille 2×3 sur $\mathbb{F}_2[X]$, il suffira de tester 14 616 sous-espaces W à la place des 20 349 sous-ensembles $\mathcal{W}$ de générateurs.

9.4.3 Utilisation de l'espace cible

Une deuxième idée nous permettant d'améliorer la complexité est de constater que les espaces solutions W que nous cherchons sont partiellement connus. En effet, la condition (*i*) indique que nous cherchons des sur-espaces du sous-espace cible T. Cependant, il y a potentiellement beaucoup plus de sur-espaces de T que de sous-ensembles $\mathcal{W}$. Comme nous cherchons des espaces qui puissent être entièrement générés par des vecteurs de $\mathcal{G}$, le théorème de la base incomplète montre qu'il suffit d'énumérer les sur-espaces obtenus en ajoutant des générateurs

dans $\mathcal{G}$. Cette idée a précédemment été utilisée par Oseledets [Oseo8] pour des algorithmes heuristiques. Nous montrons ici qu'il est possible d'utiliser cette idée sans sacrifier l'exhaustivité de la recherche. Pour cela, nous modifions la condition (ii) pour prendre en compte l'idée d'étendre T par des vecteurs de $\mathcal{G}$:

$$(ii') \quad \exists \mathcal{W}' \subset \mathcal{G} \text{ tel que } W = T \oplus \operatorname{Span} \mathcal{W}'.$$

Lemme 9.3. *Tout sous-espace W qui vérifie (i) et (ii) vérifie également (ii').*

Preuve. Soit W un sous-espace vérifiant (i) et (ii'), le résultat découle alors directement de la proposition suivante avec $\mathcal{H} = W \cap \mathcal{G}$ et $\mathcal{F}$ étant la famille libre des cibles $\mathcal{T}$. □

Proposition 9.4 ([Art91, Prop. 3.15])**.** *Soit W un espace vectoriel sur un corps K généré par une ensemble fini de générateurs $\mathcal{H}$. Soit $\mathcal{F}$ une famille libre de W. Il existe une sous-famille $\mathcal{I} \subset \mathcal{H}$ telle que $\mathcal{F} \cup \mathcal{I}$ soit une base de W.*

Preuve. Si $\mathcal{F}$ génère W, il suffit de prendre $\mathcal{I} = \emptyset$.

Sinon ($\operatorname{Span} \mathcal{F} \subsetneq W$). La famille $\mathcal{H}$ ne peut être incluse dans $\operatorname{Span} \mathcal{F}$ car elle ne génèrerait pas W et il existe ainsi $h \in \mathcal{H} \setminus \operatorname{Span} \mathcal{F}$. Ainsi, la famille $\mathcal{F}' = \mathcal{F} \cup \{h\}$ est également libre. Le processus peut ainsi être itéré jusqu'à ce que tout l'espace W soit généré. Comme il n'est pas possible de choisir deux fois un élément $h \in \mathcal{H}$, ce processus termine. Nous prenons alors pour $\mathcal{I}$ l'ensemble des générateurs h ajoutés à $\mathcal{F}$. □

Ainsi, nous pouvons modifier l'algorithme précédent pour énumérer tous les ensembles $\mathcal{W}'$ tels que $W = T \oplus \operatorname{Span} \mathcal{W}'$ vérifie les conditions (i), (ii') et (iii) ; puis sélectionner ceux d'entre eux qui vérifient également la condition (ii). Cette modification est implémentée à l'Algorithme 9.2 et sa correction est prouvée par le Théorème 9.5.

Algorithme 9.2 Énumération de tous les sous-espaces W de dimension k générés par des éléments de $\mathcal{G}$ seulement et qui contiennent l'espace cible T.

Entrées : V un espace vectoriel, $\mathcal{G}$ une famille finie de V, T un sous-espace inclus dans $\operatorname{Span} \mathcal{G}$ et un entier k tel que $\dim T \leq k \leq \operatorname{rg} \mathcal{G}$.

Sortie : L'ensemble $\mathcal{S}$ de tous les sous-espaces W de V tels que $T \subset W$, $\operatorname{Span}(W \cap \mathcal{G}) = W$ et $\dim W = k$.

1. $\mathcal{S} \leftarrow \emptyset$
2. **procedure** `expand_subspace`(W, $\mathcal{H}$)
3. **if** $\dim W = k$ **and** $\operatorname{rg}(W \cap \mathcal{G}) = k$ **then**
4. $\mathcal{S} \leftarrow \mathcal{S} \cup \{W\}$
5. **else if** $\dim W < k$ **then**
6. $\mathcal{H} \leftarrow \mathcal{H} \setminus W$
7. **while** $\mathcal{H} \neq \emptyset$ **do**
8. Dépiler un élément h de $\mathcal{H}$
9. `expand_subspace`($W \oplus \operatorname{Span}(h)$, $\mathcal{H}$)
10. **end while**
11. **end if**
12. **end procedure**
13. `expand_subspace`(T, $\mathcal{G}$)
14. **return** $\mathcal{S}$

Théorème 9.5 (Correction de l'Algorithme 9.2)**.** *Pour tout k, l'Algorithme 9.2 renvoie l'ensemble de tous les sous-espaces W de V qui vérifient les conditions (i), (ii) et (iii).*

Preuve. Soit W un sous-espace de V vérifiant (i), (ii) et (iii). Nous montrons que ce sous-espace W est bien inclus dans la sortie de l'Algorithme 9.2.

Le Lemme 9.3 nous indique que W vérifie également la condition (ii'), il existe donc $\mathcal{W}' = \{g_1, \dots, g_{k'}\}$ tel que $W = T \oplus \operatorname{Span} \mathcal{W}'$. Quitte à réordonner les g_i, nous pouvons supposer qu'ils apparaissent dans le même ordre que celui dans lequel l'algorithme les dépile. Nous supposons également que la famille $\mathcal{W}'$ est libre (il suffit d'en prendre une sous-famille libre et de même rang si ce n'est pas le cas).

Démontrons par induction que l'Algorithme 9.2 testera l'espace $T \oplus \operatorname{Span} \mathcal{W}'$. Supposons que l'algorithme ait déjà construit l'espace $W_r = T \oplus \operatorname{Span}\{g_1, \dots, g_r\}$ et montrons qu'il construira $W_{r+1} = T \oplus \operatorname{Span}\{g_1, \dots, g_{r+1}\}$. Nous considérons le premier appel de `expand_subspace` sur W_r. Si $g_{r+1} \in W_r$, nous avons que $W_r = W_{r+1}$. Sinon $g_{r+1} \notin W_r$, alors g_{r+1} est dans la pile $\mathcal{H}$ car nous sommes au premier appel sur W_r et il reste donc dans la pile tous les générateurs non liés à W_r qui apparaissent après g_r. L'algorithme appellera donc `expand_subspace` sur l'espace $W_{r+1} = W_r \oplus \operatorname{Span} g_{r+1}$. Ainsi, nous avons montré que `expand_subspace` est appelé à un instant sur $W = T \oplus \operatorname{Span} \mathcal{W}'$. Or W vérifie (ii) et (iii) et il sera ainsi sélectionné par le **if** à la ligne 3 pour faire partie des sorties de l'algorithme.

Réciproquement, soit W un sous-espace solution donné par l'Algorithme 9.2. Nous montrons alors que W vérifie les conditions (i), (ii) et (iii). La condition (i) est triviale, en effet, W est construit en étendant T. Il en est de même pour la condition (iii) qui est testée par le **if** à la ligne 3. Enfin, ce même **if** assure que $\operatorname{rg}(W \cap \mathcal{G}) = k$, or $\operatorname{Span} W \cap \mathcal{G} \subset W$ et $\dim W = k$, nous avons donc que $\operatorname{Span}(W \cap \mathcal{G}) = W$ (condition (ii)). □

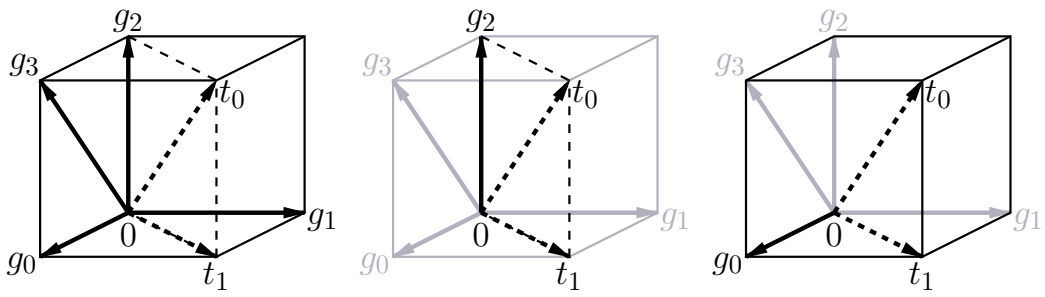
Corollaire 9.6. *En particulier, si l'Algorithme 9.2 ne trouve aucune solution pour tout k en dessous d'une borne stricte k_0, il prouve que le rang bilinéaire est au moins égal à k_0.*

Nous continuons d'utiliser l'exemple de la multiplication polynomiale de taille 2×3 en caractéristique 2 pour illustrer l'Algorithme 9.2. Nous constatons immédiatement qu'il n'existe pas de solution pour $k = 4$ car $\operatorname{rg} \mathcal{T} = 4$ et $\operatorname{rg}(T \cap \mathcal{G}) = 3$. Comme nous avons déjà exhibé des formules à 5 multiplications, ceci prouve que le rang bilinéaire de ce problème est 5.

Exemple 9.7. Nous illustrons maintenant le fonctionnement de l'Algorithme 9.2 dans $\mathbb{Q}^3$ à la Figure 9.3. Nous cherchons à générer $\mathcal{T} = \{t_0, t_1\}$ où $t_0 = (1,1,1)$ et $t_1 = (1,1,0)$ à partir des générateurs $g_0 = (1,0,0)$, $g_1 = (0,1,0)$, $g_2 = (0,0,1)$ et $g_3 = (1,0,1)$ (Figure 9.3a).

Il est immédiat de constater que l'algorithme ne trouve pas de solution pour $k = 2$: en effet, à la première exécution de `expand_subspace`, $W = T = \operatorname{Span} \mathcal{T}$ et $W \cap \mathcal{G} = \{g_2\}$ ainsi le rang de $W \cap \mathcal{G}$ est 1 et l'algorithme s'arrête (Figure 9.3b). Pour chercher les solutions avec $k = 3$, nous devrons faire un étage de récursion supplémentaire en examinant les espaces construits en ajoutant un vecteur de $\mathcal{G} \setminus T = \{g_0, g_1, g_3\}$ à T. Par exemple, nous construirons $W = T \oplus \operatorname{Span}\{g_0\}$ et constaterons que son intersection avec $\mathcal{G}$ est $\{g_0, g_1, g_2, g_3\}$ et de rang 3, ce qui fait que cet espace est solution (Figure 9.3c). L'algorithme continuera et donnera également $T \oplus \operatorname{Span}\{g_1\}$ et $T \oplus \operatorname{Span}\{g_3\}$ comme espaces solutions.

Exemple 9.8 (Schéma d'évaluation/interpolation). Nous examinons ici le cas de la multiplication de polynômes de taille n par des polynômes de taille m. Les formes bilinéaires correspondant à l'évaluation du produit en un scalaire $\kappa \in K^*$ sont des formes de rang 1 (elles peuvent s'écrire $(\sum_i a_i \kappa^i)(\sum_j b_j \kappa^j)$) et appartiennent à l'espace but (elles s'écrivent aussi $\sum_i t_i \kappa^i$)) ;



(a) $\mathcal{G}$ et $\mathcal{T}$ (b) $\mathcal{G} \cap T$ (c) $T \oplus \text{Span}\{g_0\}$

FIGURE 9.3 – Exemple d'exécution de l'Algorithme 9.2 dans $\mathbb{Q}^3$.

il en est de même pour l'évaluation à l'infini qui correspond aux coefficients de poids forts : $t_{n+m-2} = a_{n-1}b_{m-1}$. Ainsi, nous connaissons déjà $\#K$ générateurs qui appartiennent à l'espace but. Il est alors possible que toute sous-famille de ces évaluations de cardinal au plus $n+m-2$ forme une solution : quand $\#K \geq n+m-2$, T est alors un espace solution.

§ 1. Analyse de complexité. Il reste difficile de faire une analyse précise de la complexité de cette modification de l'algorithme. Mais nous pouvons en donner une borne :

$$\binom{\#\mathcal{G}}{k - \operatorname{rg}\mathcal{T}},$$

qui correspond au nombre de générateurs qu'il faut ajouter à T pour obtenir un espace candidat. Cette borne est relativement grossière et n'est jamais atteinte en pratique. Atteindre cette borne signifierait en particulier que $\mathcal{G}$ est une famille libre ce qui n'est jamais le cas en pratique.

9.4.4 Variation du parcours de l'arbre des sous-espaces

Afin d'améliorer encore le temps d'exécution pour l'Algorithme 9.2, nous devons étudier l'arbre des appels à la procédure `expand_subspace` (Figure 9.4). C'est un arbre dont la racine correspond au premier appel, c'est-à-dire `expand_subspace(`T`,` $\mathcal{G}$`)`, et de profondeur $k - \operatorname{rg}\mathcal{T}$. Les feuilles de cet arbre donnent tous les sous-espaces W dont nous testons le fait qu'ils soient solutions. Comme nous le verrons à la section suivante, ce test est coûteux et nous avons donc tout intérêt à limiter la taille de cet arbre.

Le première idée consiste à se rendre compte qu'un sous espace peut apparaître plusieurs fois à une même profondeur de l'arbre. L'Exemple 9.7 montre que le même sous-espace (V l'espace complet en l'occurrence) sera construit de trois manières différentes et testé donc trois fois. Afin d'éviter ce phénomène, nous pouvons remplacer la ligne 6 de l'Algorithme 9.2 par

$$\mathcal{H} \leftarrow \mathcal{H}/\!\sim_W \setminus \{0\} \text{ où } h \sim_W h' \Leftrightarrow W \oplus \text{Span}\{h\} = W \oplus \text{Span}\{h'\}.$$

Ce test n'introduit pas de calcul d'algèbre linéaire supplémentaire car chaque espace $W \oplus \text{Span}\{h\}$ est nécessairement construit. Nous enlevons alors le vecteur nul qui correspond à

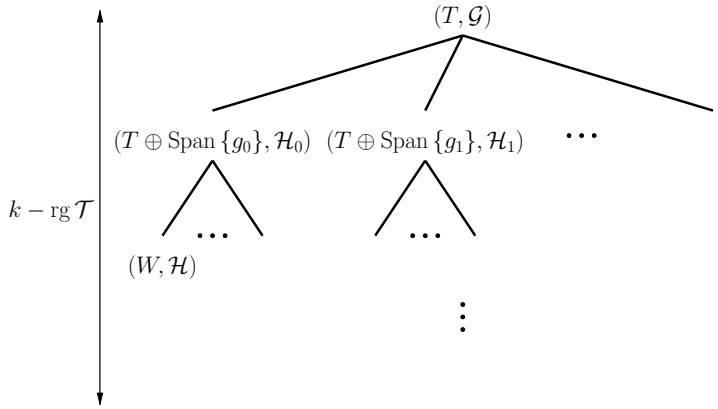


FIGURE 9.4 – Arbre d'appel de la procédure `expand_subspace` dans l'Algorithme 9.2.

la classe des générateurs déjà contenus dans le sous-espace W. Nous reprenons la Figure 9.4 pour expliquer le fonctionnement de cette optimisation. Supposons que $g_0 \sim_W g_1$, nous aurons donc $T \oplus \mathrm{Span}\{g_0\} = T \oplus \mathrm{Span}\{g_1\}$ et comme $\mathcal{H}_0 = \mathcal{H}_1 \cup \{g_0\}$, le sous-arbre partant du nœud $T \oplus \mathrm{Span}\{g_1\}$ est un sous-arbre de celui partant de $T \oplus \mathrm{Span}\{g_0\}$. Cette exploration sera donc effectuée deux fois dans l'algorithme original mais une seule fois dans l'algorithme modifié sans perdre un seul espace candidat.

Cette première idée peut être généralisée : il s'agit en fait de faire un parcours en largeur de l'arbre pour un niveau de récursion. Nous pouvons aller plus loin en augmentant la profondeur sur laquelle nous faisons un parcours en largeur ; cependant, pour des raisons d'efficacité nous devons limiter ce parcours en profondeur à une largeur permettant de garder tous les sous-espaces construits en mémoire. Ainsi, il n'est pas possible d'éliminer toutes les redondances dans l'exploration des sous-espaces mais nous pouvons tout de même limiter cet effet en modifiant le parcours de l'arbre présenté à la Figure 9.4.

9.4.5 Algorithmique des primitives d'algèbre linéaire

Nous avons pour le moment cantonné la description de notre algorithme à l'utilisation de primitives d'algèbre linéaire de haut niveau. Nous discutons ici la représentation des sous-espaces manipulés ainsi que leur calcul.

Les calculs que nous aurons à effectuer sont :
- l'élimination des vecteurs construisant le même sous-espace $(\mathcal{H}/\sim_W \setminus\{0\})$;
- l'ajout d'un vecteur à un sous-espace $(W \oplus \mathrm{Span}\{h\})$;
- l'intersection d'une famille et d'un espace $(W \cap \mathcal{G})$;
- le calcul du rang de la famille obtenue $(\mathrm{rg}\,(W \cap \mathcal{G}))$.

Afin de réduire le coût de ces opérations, nous représentons ces espaces par des matrices échelonnées réduites : chaque ligne de la matrice a un nombre de zéros en tête qui augmente strictement et chacun des pivots (c'est-à-dire le premier coefficient non nul de chaque ligne) est 1.

Commençons par étudier les deux premières opérations. Calculer le quotient $\mathcal{H}/\sim_W$ consiste à réduire chacun des vecteurs de $\mathcal{H}$ dans la matrice échelon représentant W : soit $h \in \mathcal{H}$, nous cherchons le vecteur h' tel que $h = w + h'$ où $w \in W$ et h' ne peut se réduire plus

dans W. Les vecteurs appartenant à la même classe se réduirons alors en le même vecteur réduit et nous n'avons plus qu'à retirer les doublons. De même, si $h \in W$, nous aurons $h' = 0$. Le calcul d'un sous-espace $W \oplus \text{Span}\{h\}$ consiste alors simplement à insérer le vecteur réduit h' divisé par son premier coefficient non nul dans la matrice représentant W. Nous réduisons ainsi $\mathcal{H}$ au fur et à mesure que nous descendons dans l'arbre : quand $\mathcal{H}$ a déjà été réduit par W, obtenir $\mathcal{H}/\sim_{W\oplus\text{Span}\{h\}}$ consiste simplement à réduire $\mathcal{H}$ par $\text{Span}\{h\}$ et non pas les sous-espaces complets $W \oplus \text{Span}\{h\}$, ce qui réduit encore la quantité de calculs à effectuer.

Lorsque la dimension de W a atteint k (ce qui se mesure en comptant le nombre de lignes de la matrice échelon représentant W), il faut alors calculer l'intersection $W \cap \mathcal{G}$, ce qui se fait simplement en ne gardant que les vecteurs qui se réduisent à 0 dans W. Nous calculons pour ce faire le rang de cette famille en calculant l'espace $\text{Span}(W \cap \mathcal{G})$, c'est-à-dire en insérant un à un les vecteurs de l'intersection dans une matrice échelonnée. Si la matrice obtenue contient k lignes, l'espace qu'elle représente est solution.

Nous reprenons une fois de plus la multiplication polynomiale de taille 2×3 sur $\mathbb{F}_2[X]$ pour illustrer ces méthodes. Nous avons déjà montré les matrices représentant T et $\mathcal{G}$ à la Section 9.3. La première famille $\mathcal{H}$ que nous construisons (à la racine de l'arbre d'exécution) est :

$$\mathcal{H} = \begin{array}{c} \begin{matrix} a_0b_0 & a_0b_1 & a_0b_2 & a_1b_0 & a_1b_1 & a_1b_2 \end{matrix} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array} \begin{matrix} a_1b_0 \\ a_1b_1 \\ a_1(b_0+b_1) \end{matrix},$$

et ne contient ainsi que 3 vecteurs à la place des 21 générateurs initiaux. L'une des branches que nous explorerons consistera à ajouter le générateur $g = a_1b_1$ et nous obtiendrons ainsi l'espace

$$W = \begin{array}{c} \begin{matrix} a_0b_0 & a_0b_1 & a_0b_2 & a_1b_0 & a_1b_1 & a_1b_2 \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \begin{matrix} a_0b_0 \\ a_0b_1 + a_1b_0 \\ a_0b_2 + a_1b_1 \\ g = a_1b_1 \\ a_1b_2 \end{matrix},$$

qui contient 9 générateurs formant une famille de rang 5 et fournit ainsi un espace solution.

9.5 Reconstruction des formules

Nous avons ainsi vu que l'Algorithme 9.2 permet de calculer le rang bilinéaire en le bornant inférieurement (Corollaire 9.6) et en exhibant des espaces solutions. Nous montrons ici comment reconstruire des formules à partir des sous-espaces W.

Soit W un espace solution, c'est-à-dire une espace vérifiant les conditions (i), (ii) et (iii). Comme $\mathcal{T} \subset W$ il est possible de construire par combinaisons linéaires chaque vecteur cible t_i à partir des éléments de $\mathcal{G}$ qui génèrent W ; le fait que seulement les éléments de $\mathcal{G} \cap W$ suffisent est assuré par la condition (ii). Nous pouvons même nous limiter à seulement k de ceux-ci car ils forment alors une base de W. Ainsi, chaque base de W incluse dans la famille $\mathcal{G}$ nous fournit une formule.

Pour chaque base $\{g_1, \ldots, g_k\}$ de W dans $\mathcal{G}$, il est alors clair qu'il est possible d'obtenir tous les g_i par des combinaisons linéaires des a_i et des b_i et par k multiplications complètes.

Ensuite, il suffira d'exprimer chaque vecteur cible t_i dans la base $\{g_1, \ldots, g_k\}$ pour retrouver la combinaison linéaire des g_i qu'il faut calculer pour l'obtenir.

Nous illustrons cette reconstruction par l'exemple de la multiplication de taille 2×3 sur $\mathbb{F}_2[X]$. Nous avons trouvé 3 espaces solutions et chacun d'eux peut s'écrire avec 54 bases différentes d'éléments de $\mathcal{G}$; ainsi il y a 162 formules de multiplications pour ce problème. Le premier espace solution que nous trouvons est

$$W = \mathrm{Span}\,\{a_0b_0, a_0b_2, a_0(b_0+b_2), a_1b_1, a_1b_2, a_1(b_1+b_2), \\ (a_0+a_1)b_2, (a_0+a_1)(b_0+b_1), (a_0+a_1)(b_0+b_1+b_2)\}\,.$$

Par exemple, la famille $\{g_0 = a_0b_0, g_1 = a_0b_2, g_2 = a_1b_2, g_3 = a_1(b_1+b_2), g_4 = (a_0+a_1)(b_0+b_1+b_2)\}$ forme une base de cet espace dont nous pouvons déduire la formule suivante :

$$(a_0 + a_1X) \cdot (b_0 + b_1X + b_2X^2) = g_0 + (g_0+g_1+g_3+g_4)X + (g_1+g_2+g_3)X^2 + g_4X^3.$$

9.6 Expérimentation et résultats

Nous présentons dans cette section les résultats que nous avons obtenus sur différentes instances du problème du rang bilinéaire.

L'algorithme que nous avons décrit jusque là ne dépend pas du choix du corps de base sur lequel est construit l'algèbre dans laquelle nous nous plaçons. Cependant, il dépend de l'ensemble $\mathcal{G}$ des générateurs. Plus exactement, cet ensemble de générateurs se doit d'être fini pour que notre algorithme termine : la complexité de l'algorithme est en effet bornée par $\binom{\#\mathcal{G}}{k-\mathrm{rg}\,\mathcal{T}}$.

Nous pouvons alors soit sacrifier l'exhaustivité de la recherche en sélectionnant un sous-ensemble des générateurs, soit utiliser un corps fini pour corps de base. Nous avons préféré ce dernier cas pour nos expérimentations.

Nous avions un intérêt propre à nous restreindre aux corps finis dans le cadre de cette thèse car nous cherchions des formules pour accélérer nos calculs de couplages. Mais résoudre le problème du rang sur un corps fini premier permet également d'obtenir des bornes inférieures pour le rang bilinéaire sur $\mathbb{Z}$. En effet, s'il existe une formule sur $\mathbb{Z}$, alors sa réduction modulo p donnera une formule valide contenant au plus le même nombre de produits.

9.6.1 Implémentation de l'algorithme

Afin de pouvoir atteindre les différents problèmes du rang bilinéaire qui nous ont intéressés, nous avons eu besoin d'une implémentation efficace et massivement parallèle de l'algorithme que nous avons développé dans ce chapitre.

La parallélisation de l'Algorithme 9.2 est simplifiée par le fait qu'il consiste en l'exploration d'un arbre. Cependant, les sous-arbres de celui-ci ne sont pas de taille prévisible du fait que nous utilisons un certain nombre de critères dynamiques pour arrêter l'exploration dans certaines branches. Ainsi, nous avons dû nous contenter d'un découpage heuristique de l'arbre d'exploration et n'avons pu assurer une complexité constante des tâches en lesquelles nous avons divisé notre algorithme. Cependant, ces problèmes de parallélisation ne sont pas gênants en pratique.

Par ailleurs, les primitives d'algèbre linéaire que nous manipulons (opérations sur les formes échelons) sont particulièrement bien adaptées à l'utilisation des instructions vectorielles

que fournissent les différentes extensions du jeu d'instruction `x86`. Nous avons notamment pu tirer profit de l'extension AVX qui fournit, entre autres, des instructions à trois opérandes et des instructions SIMD (*Single Instruction Multiple Data*) sur quatre mots de 64 bits.

Enfin notre implémentation permet de considérer $\mathbb{F}_2$ et $\mathbb{F}_3$ comme corps de base. Dans le second cas, les coefficients sont manipulées grâces à des techniques de *bit slicing*.

Les temps de calcul que nous présentons dans la suite sont obtenus sur des processeurs Intel Xeon L5640 cadencés à 2.27 GHz.

9.6.2 Cas particulier des applications bilinéaire symétriques

Un forme $n \times n$-bilinéaire $\gamma : \mathbf{a}, \mathbf{b} \mapsto \sum_{i,j} \gamma_{i,j} a_i b_j$ est dite symétrique si pour tout i, j, nous avons $\gamma_{i,j} = \gamma_{j,i}$. Cette notion s'étend alors naturellement aux applications bilinéaires : ϕ est symétrique si c'est un uplet de formes symétriques ou, de façon équivalente, $\mathcal{T}$ ne contient que des formes bilinéaires symétriques. Parmi les problèmes que nous avons choisi de résoudre, un certain nombre font intervenir des applications bilinéaires symétriques : c'est le cas de la multiplication de polynômes de même taille ou dans une extension de corps.

Afin d'accélérer la recherche, nous pouvons restreindre l'ensemble des générateurs $\mathcal{G}$ aux formes de rang 1 symétriques : $(\sum_i \alpha_i a_i)(\sum_j \alpha_j b_j)$. De la sorte, nous faisons passer le nombre de générateurs de $(\#K^n - 1)^2$ à seulement $\#K^n - 1$ et réduisons ainsi grandement la complexité de l'algorithme de recherche de formules. Néanmoins, se limiter aux générateurs symétriques ne permet pas d'explorer toutes les formules et il se peut que les formules optimales utilisent des générateurs non symétriques. Par exemple, pour calculer $a_0b_1 + a_1b_0$ qui est symétrique, il suffit de 2 produits non symétriques alors qu'il en faut 3 si nous nous limitons aux produits symétriques. Plus de détails sur les applications bilinéaires symétriques et leur calcul avec des formes symétriques peuvent être trouvés dans [Com+08]. Ainsi, ceci doit être considéré comme une heuristique et ne nous permettra pas de prouver des bornes sur le rang bilinéaire des problèmes concernés. Malgré cela, nous utiliserons cette heuristique dans certaines de nos expérimentations pour atteindre des tailles supérieures.

9.6.3 Résultats

Le premier problème que nous avons chercher à résoudre est celui de la multiplication de polynômes. Nous présentons dans les Tables 9.5 et 9.6 les résultats pour les polynômes de petites tailles en caractéristique 2 et 3. Pour chacun des problèmes que nous avons exploré, nous donnons :

- le nombre de générateurs,
- la valeur k du rang bilinéaire quand notre algorithme a pu le trouver, ou à défaut, une borne inférieure sur celui-ci,
- le nombre de tests, c'est-à-dire le nombre de feuilles de l'arbre d'exploration, ou encore le nombre d'espaces candidats dont nous avons calculé le rang de l'intersection avec $\mathcal{G}$,
- le nombre d'espaces solutions,
- le nombre de formules que nous pouvons déduire de ces espaces, ainsi que
- le temps de calcul nécessaire à la découverte des solutions.

Quand la taille du problème ne nous a pas permis de trouver de solution, nous avons testé tous les espaces de rang au maximum k et nous prouvons ainsi qu'une formule pour ce problème demande au moins $k + 1$ produits.

TABLE 9.5 – Résultats pour la multiplication de taille $n \times m$ sur $\mathbb{F}_2[X]$.

$n \times m$	$\#\mathcal{G}$	k	# de tests	# de solutions	# de formules	Temps de calcul [s]
2×2	9	3	1	1	1	0.00
	(Sym.) 3	3	1	1	1	0.00
3×2	21	5	2	3	162	0.00
3×3	49	6	9	3	9	0.00
	(Sym.) 7	6	1	1	7	0.00
4×2	45	6	5	4	108	0.00
4×3	105	8	700	33	423	0.00
4×4	225	9	$6.60 \cdot 10^3$	4	4	0.03
	(Sym.) 15	9	10	4	4	0.00
5×2	93	8	56	28	790 272	0.00
5×3	217	10	$1.46 \cdot 10^5$	366	48 195	0.51
5×4	465	12	$3.13 \cdot 10^8$	4 113	66 153	$2.86 \cdot 10^3$
5×5	961	13	$9.65 \cdot 10^9$	27	27	$2.28 \cdot 10^5$
	(Sym.) 31	13	$2.10 \cdot 10^3$	20	20	0.00
6×2	189	9	250	64	1 404 928	0.00
6×3	441	11	$2.05 \cdot 10^6$	3	243	11.5
6×4	945	13	$7.69 \cdot 10^9$	9	15	$1.62 \cdot 10^5$
6×5	1 953	14	$2.01 \cdot 10^{11}$	—	—	$9.97 \cdot 10^6$
6×6	3 969	14	$4.37 \cdot 10^9$	—	—	$6.03 \cdot 10^5$
	(Sym.) 63	17	$8.08 \cdot 10^6$	6	54	17.7
7×2	381	11	$9.14 \cdot 10^3$	960	∞	0.07
7×3	889	13	$2.52 \cdot 10^9$	87	63 423	$3.66 \cdot 10^4$
7×4	1 905	14	$1.47 \cdot 10^{11}$	—	—	$6.34 \cdot 10^6$
7×7	16 129	—	—	—	—	—
	(Sym.) 127	22	$3.38 \cdot 10^{12}$	2 618	19 550	$1.59 \cdot 10^7$
8×2	765	12	$7.80 \cdot 10^4$	4 096	∞	0.75
8×3	1 785	14	$5.27 \cdot 10^{10}$	—	—	—

Pour les problèmes symétriques (produit de polynômes de taille $n \times n$) que nous avons pu résoudre entièrement, nous donnons également la résolution ne considérant que des produits symétriques. Cette restriction a pour effet de réduire drastiquement le nombre de générateurs et d'accélérer le calcul, rendant possible la résolution du problème. Notons que rien ne prouve alors que la valeur de k obtenue soit le rang bilinéaire. Cependant, nous n'avons pas trouvé de taille pour laquelle ces valeurs diffèrent bien que nous ayons pu mettre en évidence des espaces solutions qui ne soient pas générés que par des produits symétriques. Nos expérimentations sur $\mathbb{F}_2[X]$ permettent de montrer l'optimalité des formules obtenues par Montgomery dans [Mon05] dans le cadre symétrique pour les tailles 5, 6 et 7.

Il est également intéressant de noter que pour la caractéristique 3, le rang bilinéaire de la multiplication polynomiale est plus faible qu'en caractéristique 2 pour différentes tailles. Ce phénomène avait déjà été observé dans [CÖ10] sans qu'il n'ait pu être prouvé.

Comme le calcul de couplage fait intervenir des extensions de faible degré de corps, calculer le rang bilinéaire de la multiplication dans ces extensions était un problème central pour nous. Nous avons déjà vu que ce rang est éventuellement plus faible que celui de la

TABLE 9.6 – Résultats pour la multiplication de taille $n \times m$ sur $\mathbb{F}_3[X]$.

$n \times m$	$\#\mathcal{G}$	k	# de tests	# de solutions	# de formules	Temps de calcul [s]
2×2	16	3	1	1	4	0.00
	(Sym.) 4	3	1	1	4	0.00
3×2	52	4	1	1	1	0.00
3×3	169	6	24	22	1 493	0.00
	(Sym.) 13	6	1	1	1 170	0.00
4×2	160	6	9	13	38 880	0.00
4×3	520	7	164	12	48	0.00
4×4	1 600	9	$4.11 \cdot 10^5$	726	50 640	14.9
	(Sym.) 40	9	32	9	36 864	0.00
5×2	484	7	24	36	93 312	0.00
5×3	1 573	9	$2.81 \cdot 10^5$	1 116	94 629	9.33
5×4	4 840	10	$4.75 \cdot 10^6$	48	768	$1.01 \cdot 10^3$
5×5	14 641	11	$4.89 \cdot 10^7$	—	—	$4.02 \cdot 10^4$
	(Sym.) 121	12	$3.93 \cdot 10^4$	31	6 460	0.14
6×2	1 456	8	69	81	104 976	0.01
6×3	4 732	10	$3.24 \cdot 10^6$	240	4 272	566
6×4	14 560	11	$4.55 \cdot 10^7$	—	—	$3.31 \cdot 10^4$
6×5	44 044	12	$4.58 \cdot 10^8$	—	—	$1.31 \cdot 10^6$
6×6	132 496	—	—	—	—	—
	(Sym.) 364	15	$2.37 \cdot 10^8$	4	1 024	$3.79 \cdot 10^3$
7×2	4 372	10	$2.27 \cdot 10^4$	10 530	∞	2.84
7×3	14 209	11	$3.15 \cdot 10^7$	—	—	$1.84 \cdot 10^4$
7×4	43 720	12	$4.16 \cdot 10^8$	—	—	$1.03 \cdot 10^6$
7×7	1 194 649	—	—	—	—	—
	(Sym.) 1 093	17	$2.69 \cdot 10^{10}$	—	—	$1.50 \cdot 10^6$
8×2	13 120	11	$2.01 \cdot 10^5$	85 293	∞	53.6
8×3	42 640	12	$2.90 \cdot 10^8$	—	—	$5.46 \cdot 10^5$

TABLE 9.7 – Résultats pour la multiplication sur les petites extensions de $\mathbb{F}_2$ et $\mathbb{F}_3$.

Corps fini	**$\#\mathcal{G}$**	**k**	**# de tests**	**# de solutions**	**# de formules**	**Temps de calcul [s]**
$\mathbb{F}_{2^2}$	9	3	3	3	3	0.00
$\mathbb{F}_{2^3}$	49	6	$7.03 \cdot 10^3$	105	147	0.01
$\mathbb{F}_{2^4}$	225	9	$2.57 \cdot 10^9$	2 025	2 025	$1.13 \cdot 10^4$
$\mathbb{F}_{2^5}$	961	9	$3.10 \cdot 10^{10}$	—	—	$8.11 \cdot 10^5$
	(Sym.) 31	13	$3.49 \cdot 10^6$	2 015	2 015	6.24
$\mathbb{F}_{2^6}$	(Sym.) 63	15	$2.21 \cdot 10^{10}$	21	21	$6.63 \cdot 10^4$
$\mathbb{F}_{2^7}$	(Sym.) 127	15	$1.34 \cdot 10^{12}$	—	—	$6.17 \cdot 10^6$
$\mathbb{F}_{3^2}$	16	3	3	4	16	0.00
$\mathbb{F}_{3^3}$	169	6	$2.42 \cdot 10^5$	11 843	105 963	1.08
$\mathbb{F}_{3^4}$	1 600	8	$2.27 \cdot 10^{11}$	—	—	$1.08 \cdot 10^7$
	(Sym.) 40	9	$1.10 \cdot 10^5$	234	615 240	0.45
$\mathbb{F}_{3^5}$	(Sym.) 121	11	$2.66 \cdot 10^9$	121	121	$1.45 \cdot 10^4$
$\mathbb{F}_{3^6}$	(Sym.) 364	12	$3.01 \cdot 10^{12}$	—	—	$4.50 \cdot 10^7$

multiplication polynomiale correspondante. Cependant, la recherche exhaustive de formules est plus complexe dans ce cas que pour les polynômes : la complexité de la recherche dépend de $k - \operatorname{rg} \mathcal{T}$ et les cibles sont ici de rang n contrairement à la multiplication polynomiale où il atteint $2n - 1$.

Nous présentons les résultats obtenus pour les petites extensions de $\mathbb{F}_2$ et $\mathbb{F}_3$ dans la Table 9.7. Nous pouvons observer que nous obtenons un gain par rapport à la multiplication polynomiale pour les extensions $\mathbb{F}_{2^5}$, $\mathbb{F}_{2^6}$ et $\mathbb{F}_{3^5}$. Cenk & Özbudak avaient trouvé des formules pour $\mathbb{F}_{2^5}$ et $\mathbb{F}_{2^6}$ sans prouver leur optimalité [CÖ10]. À notre connaissance, les formules que nous produisons pour $\mathbb{F}_{3^5}$ n'étaient pas connues et nous améliorons ainsi la compléxité bilinéaire de la multiplication dans les extensions de degré 5 en caractéristique 3. Ce dernier cas fut d'un intérêt particulier pour nous car nous avions besoin d'une telle multiplication pour l'implémentation de couplage présentée dans [Est10]. Nous avons ainssi pu passer de 12 multiplications à 11 et nous verrons que cela apporte un gain de 8% en cycles dans l'implémentation concernée. Nous donnons l'Algorithme 9.8 obtenue à partir de l'une des 121 formules pour le produit dans une extension de degré 5 en caractéristique 3.

Enfin nous donnons dans la Table 9.9 les résultats obtenus pour le produit court. Nous confirmons alors l'optimalité des formules données dans [Ose08] en caractéristique 2 et étendons ces travaux à la caractéristique 3.

Algorithme 9.8 Multiplication sur $\mathbb{F}_{q^5} \cong \mathbb{F}_q[\alpha]$ avec $3 \mid q$ et $\alpha^5 - \alpha + 1 = 0$ (11 mul., 47 add.)

Entrées : $A = a_4\alpha^4 + a_3\alpha^3 + a_2\alpha^2 + a_1\alpha + a_0 \in \mathbb{F}_{q^5}$ et $B = b_4\alpha^4 + b_3\alpha^3 + b_2\alpha^2 + b_1\alpha + b_0 \in \mathbb{F}_{q^5}$ avec $a_i, b_i \in \mathbb{F}_q$

Sortie : $A \cdot B = c_4\alpha^4 + c_3\alpha^3 + c_2\alpha^2 + c_1\alpha + c_0$.

1. $a_5 \leftarrow a_1 - a_3$; $b_5 \leftarrow b_1 - b_3$
2. $a_6 \leftarrow a_2 - a_4$; $b_6 \leftarrow b_2 - b_4$
3. $a_7 \leftarrow -a_0 + a_2$; $b_7 \leftarrow -b_0 + b_2$
4. $a_8 \leftarrow a_4 + a_5$; $b_8 \leftarrow b_4 + b_5$
5. $p_0 \leftarrow a_4 \cdot b_4$; $p_1 \leftarrow a_8 \cdot b_8$
6. $p_2 \leftarrow (a_0 - a_3) \cdot (b_0 - b_3)$; $p_3 \leftarrow (a_1 - a_4) \cdot (b_1 - b_4)$
7. $p_4 \leftarrow (a_3 - a_4) \cdot (b_3 - b_4)$; $p_5 \leftarrow (a_1 - a_7) \cdot (b_1 - b_7)$
8. $p_6 \leftarrow (a_0 + a_2) \cdot (b_0 + b_2)$; $p_7 \leftarrow (a_3 + a_6) \cdot (b_3 + b_6)$
9. $p_8 \leftarrow (a_5 - a_7) \cdot (b_5 - b_7)$; $p_9 \leftarrow (a_0 + a_8) \cdot (b_0 + b_8)$
10. $p_{10} \leftarrow (a_5 + a_6) \cdot (b_5 + b_6)$
11. $t_0 \leftarrow p_6 - p_5$; $t_1 \leftarrow p_1 + p_2$; $t_2 \leftarrow p_4 + p_{10}$
12. $t_3 \leftarrow p_8 - t_2$; $t_4 \leftarrow p_3 - p_5$; $t_5 \leftarrow p_7 - t_0$
13. $t_6 \leftarrow p_8 - t_1$
14. $c_0 \leftarrow p_0 - p_4 - t_4 - t_6$
15. $c_1 \leftarrow p_2 + p_3 + t_0 - t_3$
16. $c_2 \leftarrow -p_3 - t_5 + t_6$
17. $c_3 \leftarrow -p_0 - p_7 - t_3 - t_4$
18. $c_4 \leftarrow p_9 + t_1 + t_3 + t_5$
19. **return** $c_4\alpha^4 + c_3\alpha^3 + c_2\alpha^2 + c_1\alpha + c_0$

TABLE 9.9 – Résultats pour la multiplication dans $\mathbb{F}_p[X]/(X^n)$ et $\mathbb{F}_p[X]/(X^n-1)$ pour $p=2$ et 3.

Anneau	n	$\#\mathcal{G}$	k	**# de tests**	**# de solutions**	**# de formules**	**Temps de calcul [s]**
$\mathbb{F}_2[X]/(X^n)$	2	9	3	3	3	10	0.00
	3	49	5	590	12	40	0.00
	4	225	8	$5.17 \cdot 10^7$	1 440	9 248	230
	5	961	9	$2.66 \cdot 10^{10}$	—	—	$6.70 \cdot 10^5$
		(Sym.) 31	11	$3.64 \cdot 10^5$	112	736	0.48
	6	(Sym.) 63	14	$2.63 \cdot 10^9$	384	2 816	$7.66 \cdot 10^3$
	7	(Sym.) 127	15	$1.16 \cdot 10^{12}$	—	—	$5.46 \cdot 10^6$
$\mathbb{F}_2[X]/(X^n-1)$	2	9	3	3	3	10	0.00
	3	49	4	21	3	3	0.00
	4	225	8	$2.69 \cdot 10^7$	1 440	9 248	124
	5	961	9	$1.39 \cdot 10^{10}$	—	—	$3.65 \cdot 10^5$
		(Sym.) 31	10	$7.46 \cdot 10^4$	25	25	0.09
	6	(Sym.) 63	12	$2.33 \cdot 10^7$	31	148	50.0
	7	(Sym.) 127	13	$2.55 \cdot 10^9$	1	49	$1.24 \cdot 10^4$
$\mathbb{F}_3[X]/(X^n)$	2	16	3	4	4	39	0.00
	3	169	5	$7.94 \cdot 10^3$	90	1 539	0.07
	4	1 600	7	$5.54 \cdot 10^8$	—	—	$3.22 \cdot 10^4$
		(Sym.) 40	8	$3.17 \cdot 10^5$	252	40 095	0.14
	5	(Sym.) 121	10	$1.45 \cdot 10^8$	243	13 122	$2.28 \cdot 10^3$
	6	(Sym.) 364	11	$4.79 \cdot 10^{10}$	—	—	$8.22 \cdot 10^5$
$\mathbb{F}_3[X]/(X^n-1)$	2	16	2	1	1	1	0.00
	3	169	5	$4.45 \cdot 10^3$	90	1 539	0.04
	4	1 600	5	767	4	16	0.07
	5	(Sym.) 121	10	$8.74 \cdot 10^7$	234	615 240	$1.39 \cdot 10^3$
	6	(Sym.) 364	10	$3.37 \cdot 10^8$	9	2 025	$1.68 \cdot 10^4$

Quatrième partie

Trois implémentations matérielles de couplages

Chapitre 10

Introduction aux problématiques de l'implémentation FPGA

Nous souhaitons donner dans ce court chapitre une introduction aux problématiques de l'implémentation matérielle et plus spécifiquement pour la plateforme FPGA. Nous renvoyons le lecteur souhaitant plus de détails à [PH11, Appendice B].

10.1 Notions d'implémentation matérielle

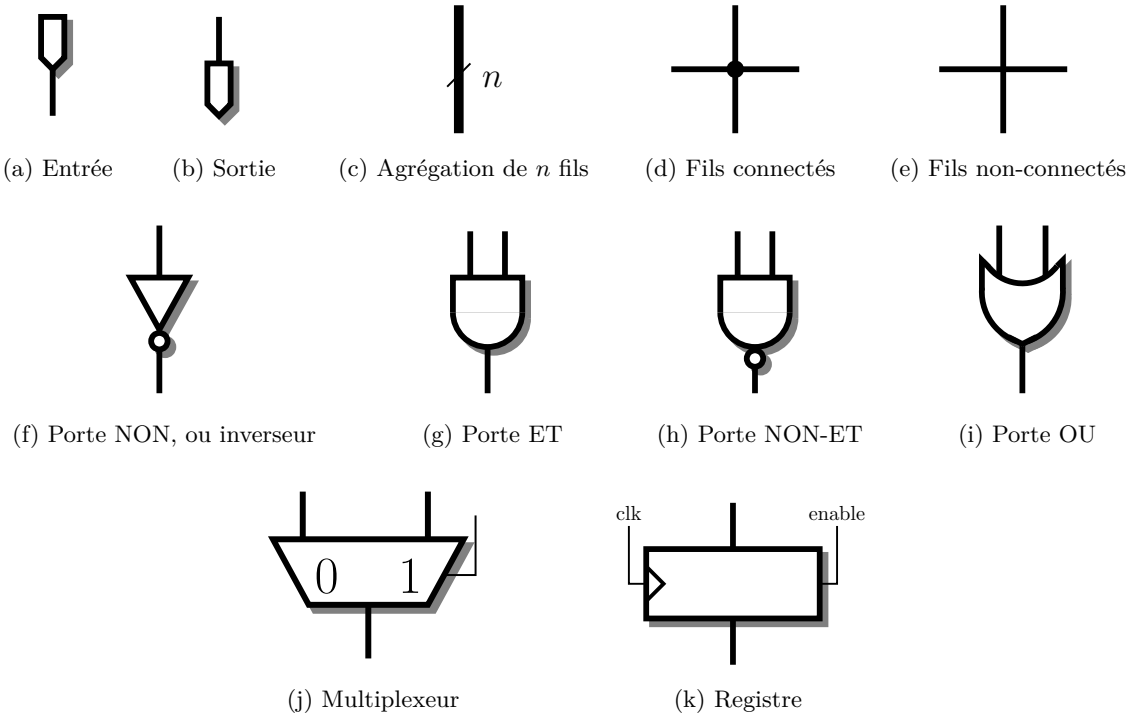
10.1.1 Représentation schématique de circuits

La réalisation d'une implémentation matérielle consiste à fournir une description de l'architecture du circuit. C'est-à-dire, un ensemble d'instances de composants de base et les connexions entre celles-ci. Nous donnons dans la Figure 10.1 la liste des composants et des conventions de dessins d'architecture que nous avons utilisés dans les schémas de cette thèse.

Les **portes logiques** telles que les portes ET, OU et NON permettent de réaliser n'importe quel circuit combinatoire. Pour aider à la concision et la compréhension des schémas, nous utiliserons également le **multiplexeur** (Figure 10.1j) : il permet de sélectionner l'une de ses entrées pour la recopier sur sa sortie grâce à un bit de contrôle.

Nous avons aussi besoin de pouvoir séquentialiser des calculs. Pour ce faire, nous utilisons des **registres** (Figure 10.1k). Ils conservent une valeur dans leur état interne qu'ils recopient sur leur sortie. Comme nous travaillons avec des circuits synchrones, la mise à jour des états de tous les registres d'un circuit se fait en même temps. Ceci est contrôlé par le **signal d'horloge** *clk* : un signal carré de période régulière. Lors de chacun des fronts montants de ce signal, et si le bit de contrôle *enable* est à la valeur 1, un registre recopie dans son état interne la valeur de son entrée ; cette valeur sera alors immédiatement recopiée sur la sortie du registre. Le bit de contrôle *enable* est optionnel et sa valeur sera supposée être 1 quand nous l'omettrons dans les prochains schémas.

Comme tous les composants non purement combinatoires sont connectés au signal d'horloge, nous ne le représentons généralement pas sur les schémas dans un souci de lisibilité. Cependant, nous signalons qu'un composant est sensible aux fronts montants de l'horloge par un petit triangle sur le coté de la boîte le représentant comme dans la Figure 10.1k.



(a) Entrée (b) Sortie (c) Agrégation de n fils (d) Fils connectés (e) Fils non-connectés

(f) Porte NON, ou inverseur (g) Porte ET (h) Porte NON-ET (i) Porte OU

(j) Multiplexeur (k) Registre

FIGURE 10.1 – Liste des symboles d'architecture matérielle utilisés.

10.1.2 Mesure de performance

Contrairement à une implémentation logicielle où il suffit de donner un temps de calcul, la mesure de performance d'une implémentation matérielle est multicritère. Premièrement, nous mesurons la quantité de composants utilisés : ce paramètre est la **surface** et son unité dépend fortement de la technologie utilisée. Pour les ASIC — *Application-Specific Integrated Circuits* — la surface est généralement exprimée en nombre de portes de base ou, plus directement, en mm^2 de silicium occupée par le circuit intégré. Nous discuterons dans la section suivante ce qu'il en est pour les FPGA auxquels nous nous sommes intéressés.

Vient ensuite le **temps de calcul**. Celui-ci dépend d'un autre paramètre : la **fréquence d'horloge**. À chaque cycle d'horloge, tous les registres du circuit sont chargés en même temps ; il faut donc attendre que les entrées des registres soient toutes stabilisées avant de pouvoir passer au cycle suivant. Ainsi, la fréquence maximale de fonctionnement d'un circuit est déterminée par le temps maximal de propagation d'un signal partant de la sortie d'un registre vers l'entrée d'un autre. Le chemin réalisant ce maximum est appelé le **chemin critique**.

Le temps de calcul est alors le nombre de cycles nécessaires au calcul multiplié par la durée de ceux-ci. Parfois, il est plus intéressant de considérer l'inverse de cette valeur, c'est-à-dire le nombre de calculs pouvant être réalisés par unité de temps.

Lors du développement d'un circuit, il convient donc d'identifier les chemins critiques et de minimiser leur longueur. Selon le circuit à concevoir, une technique permettant de réduire la longueur des chemins critiques est de réaliser des opérateurs pipelinés en insérant

des registres sur les chemins critiques. Cela va permettre de diviser leur longueur et donc d'accélérer tout le circuit en augmentant sa fréquence d'horloge. Cependant, les opérateurs pipelinés ne permettront pas d'obtenir le résultat plus rapidement : il faudra plusieurs cycles pour obtenir le résultat. Par contre, il sera possible de fournir un jeu de données par cycle et nous augmenterons ainsi le débit de l'opérateur.

Afin de comparer des implémentations différentes, il peut être intéressant d'utiliser le produit temps-surface ou le ratio débit/surface. En effet, si, pour un même calcul, une architecture est deux fois plus lente mais aussi deux fois plus compacte qu'une autre, alors les même performances en termes de débit peuvent être atteintes en utilisant deux instances de la plus petite architecture. Ainsi, si une latence faible n'est pas l'objectif principal de l'implémentation, il sera intéressant de considérer l'architecture plus compacte.

10.2 Circuits reconfigurables FPGA

10.2.1 Présentation générale

Les circuits reconfigurables sont des circuits intégrés dont la (re)configuration permet d'émuler n'importe quel circuit. Nous avons utilisé des FPGA — *Field-Programmable Gate Arrays* — et nous décrivons maintenant leur architecture. Comme leur nom l'indique, les FPGA sont constitués de cellules reconfigurables disposées régulièrement sur une grille 2D. Chaque cellule, très simple, peut alors être configurée pour réaliser une porte logique et connectée aux autres pour réaliser un circuit complet.

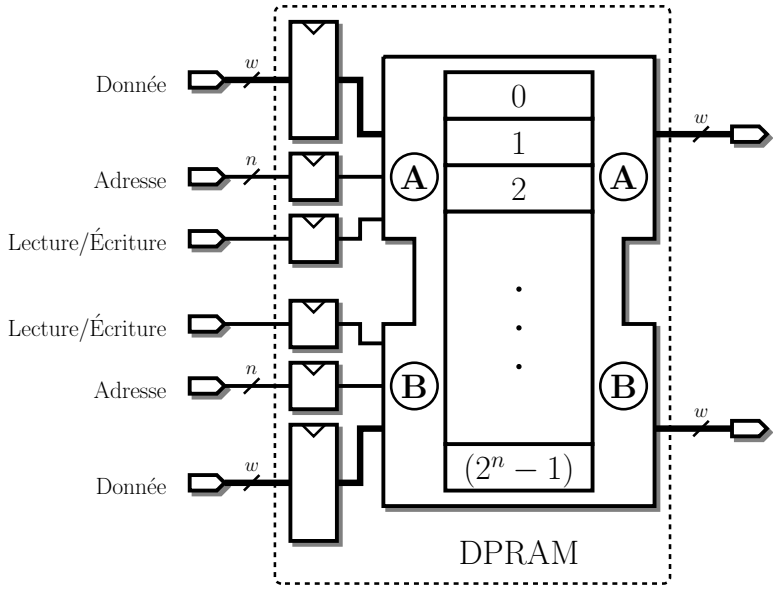
Les FPGA sont ainsi des circuits très flexibles qui peuvent être soit utilisés comme une plateforme de prototypage rapide soit directement comme un accélérateur spécifique. Ils sont fréquemment utilisés dans des applications diverses : logique d'interface, traitement du signal, périphériques réseaux filaires et sans-fil, calcul haute-performance, traitement vidéo et, pour ce qui nous intéresse le plus, coprocesseurs cryptographiques.

Nous avons choisi d'utiliser les FPGA pour deux raisons qui font leur popularité dans leurs applications. D'une part, leur programmation est plus simple que la description d'un ASIC et nous permet ainsi de tester plus rapidement nos idées architecturales. D'autre part, ils permettent d'obtenir des opérateurs matériels spécifiques donnant de meilleures performances que celles obtenues grâce à des implémentations logicielles, à coût raisonnable.

10.2.2 Spécificités architecturales

Comme nous l'avons vu, les FPGA sont constitués d'une grille de cellules configurables. Celles-ci contiennent chacune une ou plusieurs tables de correspondance — *lookup tables* en anglais (LUT) — permettant, selon la famille de FPGA concernée, d'implémenter n'importe quelle fonction booléenne à n entrées ($4 \leq n \leq 6$) et à m sorties ($1 \leq m \leq 2$). Afin de pouvoir réaliser une classe plus grande de circuits combinatoires, ces LUT sont interconnectées par une matrice de routage elle-même configurable.

Pour réaliser un calcul, il faut également un moyen de séquentialiser les opérations : il faut pouvoir mémoriser. Dans les FPGA nous trouvons des éléments de mémorisation sous deux formes. La première est présente sous forme de registres connectés aux sorties de chacune des LUT. Il est ensuite possible de choisir entre la sortie directe de la LUT ou celle du registre correspondant comme entrée des portes suivantes, selon que nous souhaitions ou non insérer un étage de registres à ce niveau. La seconde est donnée par des blocs de mémoire à accès



Chaque port d'une DPRAM est constitué de trois entrées (une pour sélectionner l'adresse qui va être lue ou écrite selon la valeur de l'entrée « lecture/écriture » et une entrée pour les données à éventuellement écrire) et d'une sortie donnant la valeur présente à l'adresse fournie au cycle précédent. Les deux ports A et B accèdent parallèlement à la même mémoire.

FIGURE 10.2 – Mémoire à accès direct double port (DPRAM).

direct à double port répartis sur toute la surface du FPGA. Certaines de nos architectures reposent sur ces blocs de mémoire dont nous donnons un schéma Figure 10.2.

Dans les FPGA modernes, nous trouvons également des blocs DSP (*Digital Signal Processing*), initialement présents pour les applications de traitement du signal, qui permettent de réaliser plus rapidement qu'avec les LUT certaines opérations courantes comme la multiplication et l'accumulation de petits entiers. Ces blocs ne présentent pas d'intérêt dans nos implémentations en petite caractéristique mais nous verrons que les implémentations pour courbes BN en font un grand usage.

La configuration du FPGA est l'ensemble des valeurs contenues dans les LUT et la topologie de la matrice de routage. Elle s'obtient à partir de la description dans le langage VHDL ou Verilog du circuit à implémenter. Des outils de synthèse effectuent ensuite la transformation en plusieurs étapes. Dans un premier temps, ils transforment le circuit en un circuit composé uniquement d'éléments présents sur le FPGA, puis ils choisissent la place de chacun de ceux-ci sur le FPGA avant de chercher comment connecter les différents blocs ainsi instanciés et placés (étape de placement/routage). Les performances d'un circuit découlent directement de la configuration du FPGA ainsi obtenue.

10.2.3 Les FPGA Xilinx

Il existe une grande variété de FPGA dans le commerce. Deux constructeurs principaux — Xilinx et Altera — fournissent des architectures relativement différentes décrites par un vocabulaire séparé. Les comparaisons de circuits implantés pour différentes marques de FPGA

ne sont pas aisées car elles fournissent des cellules de base différentes. Comme le monde académique de la cryptographie matérielle utilise principalement les FPGA de marque Xilinx, nous avons conservé ce choix afin de pouvoir nous comparer aux architectures existantes. Les cellules de base des FPGA Xilinx sont appelées les **slices** et sont l'unité de mesure de surface que nous utiliserons.

Les FPGA Xilinx sont divisés en plusieurs familles : nous avons utilisé les Virtex pour les applications hautes performances et les Spartan, moins performants mais aussi moins chers et plus économes en énergie, pour les applications embarquées. Ces familles se divisent en plusieurs générations (nous ne décrivons que celles que nous trouvons dans la littérature concernant les couplages) :

— Virtex-II, le plus ancien que nous avons utilisé et noté XC2V,
— Spartan-3, noté XC3S,
— Virtex-4, noté XC4V,
— Virtex-5, qui change la structure de la slice en passant d'une slice comportant deux LUT à 4 entrées/1 sortie à une slice comportant 4 LUT à 6 entrées/1 sortie ou 5 entrées/2 sorties, noté XC5V,
— Virtex-6, noté XC6V.

Ces générations se déclinent encore en différentes versions selon l'existence et le nombre de cellules spéciales disponibles (ports d'entrées-sorties rapides pour les applications réseaux, cellules DSP, cœurs RISC complets, etc.). À l'exception du Virtex-II qui est de conception plus ancienne et que nous avons utilisé dans sa version « Pro » légèrement plus récente, les FPGA Xilinx se divisent en trois familles :

— LX, orientée logique, comprenant les FPGA avec une plus grande proportion de LUT,
— SX, orientée traitement du signal, comprenant les FPGA avec une plus grande proportion de cellules DSP, et
— FX, orientée systèmes embarqués, comprenant les FPGA intégrants un ou plusieurs cœurs PowerPC.

Comme nous n'utilisons pas ces fonctionnalités spéciales, nous avons utilisé les versions « LX » de ces FPGA. Ceci est dénoté par l'adjonction de LX à la notation précédente.

Certains de ces FPGA possèdent également des terminaisons pour leur connections à des bus PCIe et des contrôleurs d'accès Ethernet. Lorsque qu'ils ont cette connectivité avancée, un « T » est adjoint à la notation.

Enfin, les FPGA Xilinx sont disponibles en plusieurs tailles ce qui apparaît avec un nombre accolé à la notation précédente : plus ce nombre est grand, plus le FPGA dispose de cellules logiques ou de blocs DSP. Le dernier élément de cette notation est le *speedgrade*, noté par un entier négatif n. Il représente la fréquence maximale atteignable par le FPGA : plus le *speedgrade* n est petit, plus cette fréquence est grande.

Par exemple, nous avons utilisé le FPGA XC6VLX75T−3 qui est le plus petit Virtex-6 LXT avec le *speedgrade* le plus performant.

Chapitre 11

Accélérateur *ad-hoc* parallèle

Nous présentons dans ce chapitre la première implémentation de couplage à laquelle nous avons contribué durant cette thèse [Beu+09a ; Beu+11]. Cette implémentation a été construite de manière à minimiser le temps de calcul pour un couplage ; nous nous intéressons ainsi aux applications demandant de faibles latences. Nous verrons plus tard que les techniques que nous avons adoptées nous ont aussi permis d'obtenir un accélérateur minimisant également le produit temps-surface ; ainsi cette architecture prend aussi son intérêt en tant que coprocesseur dans des serveurs avec des services cryptographiques à forte charge tels que ceux des banques.

Cette implémentation repose sur les courbes elliptiques supersingulières de caractéristique 2 ou 3. Nous bénéficions ainsi de courbes permettant d'obtenir un couplage symétrique (type I). Nous disposons également du couplage η_T sur ces courbes et nous réduisons ainsi la longueur de la boucle de Miller. De plus, ces courbes ont une arithmétique efficace : le doublement ou le triplement de point peuvent être calculés par des formules simples en caractéristique 2 ou 3 respectivement (voir Sections 7.1 et 7.2)

Comme nous l'avons vu, la multiplication est l'opération arithmétique critique dans le calcul de couplage. Nous avons ainsi cherché à optimiser son calcul et réduire sa latence au maximum en articulant notre architecture autour d'un multiplieur parallèle pipeliné que nous décrivons dans la prochaine section (Section 11.1). La principale difficulté est alors de tirer au maximum avantage du multiplieur. En effet, le multiplieur étant parallèle, il occupe une surface très importante sur le FPGA ; il est donc important que celui-ci soit utilisé à chaque cycle en lui fournissant un produit à effectuer à chaque top d'horloge. Celui-ci étant également pipeliné, chaque multiplication prend un certain nombre de cycles ; ainsi la dépendance entre les opérations arithmétiques doit être étudiée afin de maintenir le pipeline du multiplieur plein. Il faut ainsi exploiter le parallélisme intrinsèque au calcul de couplage. Comme ce parallélisme ne se trouve qu'au niveau des opérations arithmétiques, il est nécessaire d'étudier précisément les algorithmes de calcul du couplage et nous devons ainsi étudier séparément les cas de la caractéristique 3 (Section 11.2) et 2 (Section 11.3).

Cependant, il est important de noter que le calcul du couplage η_T se découpe en deux étapes principales quelle que soit la caractéristique : le calcul du couplage non réduit par la boucle de Miller et le calcul de l'exponentiation finale. La première étape — la boucle de Miller — est la plus coûteuse et consiste en une suite d'itérations régulières pour laquelle il est possible de concevoir une architecture *ad-hoc* autour d'un multiplieur parallèle. La

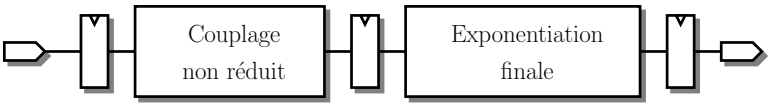


FIGURE 11.1 – Architecture globale de l'accélérateur *ad-hoc*.

deuxième étape — l'exponentiation finale — est moins longue mais plus irrégulière. Il est ainsi dommageable à l'architecture pour le calcul de la boucle de Miller d'être adaptée pour supporter l'exponentiation finale. Nous avons ainsi choisi de dessiner un accélérateur par étape et de les pipeliner (voir Figure 11.1). Afin de maximiser l'efficacité de l'architecture, le calcul d'exponentiation est réalisé par un coprocesseur le plus compact possible qui permet d'effectuer le calcul en moins de temps que celui du couplage non réduit. Comme il s'agit d'un calcul irrégulier nous avons choisi de concevoir un coprocesseur générique pour les corps finis ; celui-ci est décrit au chapitre suivant (Chapitre 12). Comme ce coprocesseur bénéficie d'une conception très générique, nous verrons comment nous l'avons utilisé pour calculer d'autres couplages.

11.1 Multiplieur parallèle pipeliné

Nous construisons dans cette section des multiplieurs matériels parallèles et pipelinés pour corps finis de caractéristique 2 et 3.

Avant de décrire l'architecture du multiplieur, nous nous attardons sur la représentation des éléments du corps fini $\mathbb{F}_{p^m}$ dans le circuit. Chaque élément est représenté par un polynôme de degré au plus $m-1$ sur $\mathbb{F}_p$. En caractéristique 2, chaque coefficient peut être stocké sur un bit, ainsi l'addition correspond à l'opérateur logique `Xor` et la multiplication au `And`. En caractéristique 3 cependant, il faut deux bits pour représenter les trois valeurs possibles. Nous utilisons le schéma *borrow-save* introduit par Duprat et Muller en 1991 [DM91] : un trit $a \in \{0, 1, -1\}$ est représenté par deux fils a_+ et a_- de telle sorte que $a = a_+ - a_-$. Nous utilisons alors l'architecture en LUT des FPGA, avec deux LUT à quatre entrées pour réaliser chacune des opérations $+$ et $\times$. En caractéristique 3, il faut aussi pouvoir prendre l'opposé d'un coefficient ; c'est une opération gratuite en matériel vue la représentation choisie : il suffit d'échanger le bit $+$ et le bit $-$.

11.1.1 Architecture du multiplieur

L'architecture globale de notre multiplieur sur $\mathbb{F}_{p^m}$ consiste à décomposer ce produit en un produit sur les polynômes de degré au plus $m-1$ représentant les éléments du corps à multiplier puis à effectuer la réduction modulo le polynôme de définition de $\mathbb{F}_{p^m}$. Le schéma de l'opérateur est donné à la Figure 11.2. En choisissant un polynôme irréductible de poids de Hamming faible (trinôme, pentanôme) pour polynôme de définition, la réduction consiste en la somme d'un coefficient de poids faible et de quelques coefficients de poids fort.

Notre multiplieur est pipeliné : si nous lui donnons un jeu d'entrées à chaque cycle d'horloge, il en ressortira, après quelques cycles de latence, un résultat à chaque cycle. Pour ce faire, il y a des registres tout le long du calcul. Comme cette architecture est générée automatiquement et que le générateur permet de choisir la profondeur du pipeline, nous avons inséré des registres optionnels dans le circuit que le générateur choisira ou non d'utiliser. La

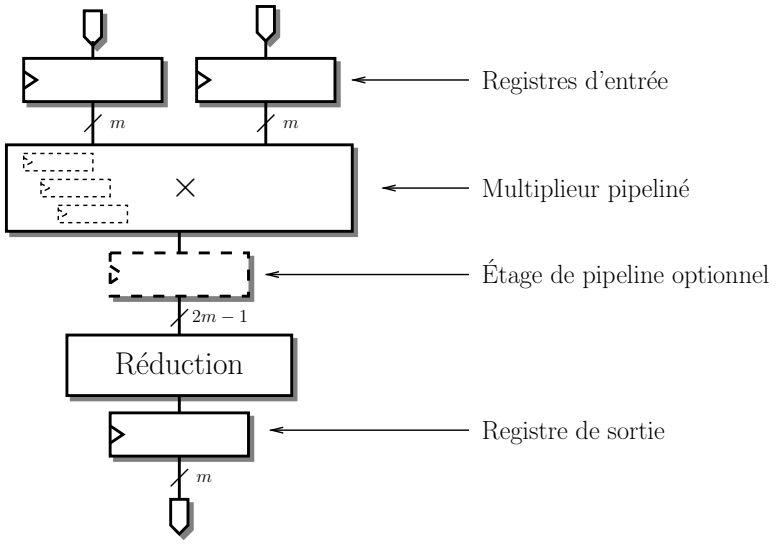


FIGURE 11.2 – Architecture globale du multiplieur dans un corps fini $\mathbb{F}_{2^m}$ ou $\mathbb{F}_{3^m}$.

fréquence du multiplieur ainsi pipeliné est alors déterminée par la taille du chemin le plus long entre deux registres. Afin de conserver les performances de notre multiplieur quand nous le plaçons dans un circuit complet, nous l'avons muni de registres en entrée et en sortie.

Comme ce multiplieur servira à effectuer les produits sur le corps de base des courbes elliptiques supersingulières que nous utiliserons, nous aurons, au vu des différentes exigences de sécurité, à multiplier des éléments de corps finis de 200 à 800 bits ou 100 à 300 trits.

À partir de ces tailles de corps — tailles dites **cryptographiques** —, il devient intéressant de considérer des algorithmes de complexité sous-quadratique comme ceux étudiés dans la Partie III. Ils ont pour particularité de pouvoir être utilisés récursivement : le schéma global de ces algorithmes consiste en la décomposition des opérandes pour obtenir un certain nombre de sous-produits à calculer et en la recomposition du produit à partir des sous-produits. L'architecture du multiplieur sur les polynômes prend alors naturellement la forme présentée à la Figure 11.3. Plus précisément, chaque algorithme de multiplication consiste en quelques additions pour découper les entrées en plusieurs jeux d'entrées pour les multiplications de polynômes de degré inférieur. Les sorties de ces multiplieurs sont alors récupérées par un composant qui les rassemble, au prix de quelques additions, en le résultat souhaité.

Dans le but de pipeliner le multiplieur, nous pouvons générer des registres à l'entrée de la multiplication ou à la sortie des multiplieurs de taille inférieure. Nous avons choisi ces emplacements pour les registres de sorte à garantir une certaine précision dans le choix de leur placement tout en conservant la compatibilité entre les différents algorithmes.

Les composants pour la décomposition en sous-produits et recomposition du résultat sont constitués de sommes d'un certain nombre de coefficients. Afin de minimiser la latence de telles sommes, il faut minimiser la profondeur des arbres d'additions. Il y a ici une dépendance assez forte à la technologie FPGA utilisée. En effet, l'architecture nous permet de faire la somme de 2 à 4 (ou 6 sur Virtex-5 ou autre FPGA Xilinx plus récent) termes en caractéristique 2 pour le même coût que la somme de deux termes. Du fait qu'il faille deux signaux pour représenter un coefficient en caractéristique 3, la somme de deux (ou trois sur Virtex-5) termes maximum nécessite deux LUT.

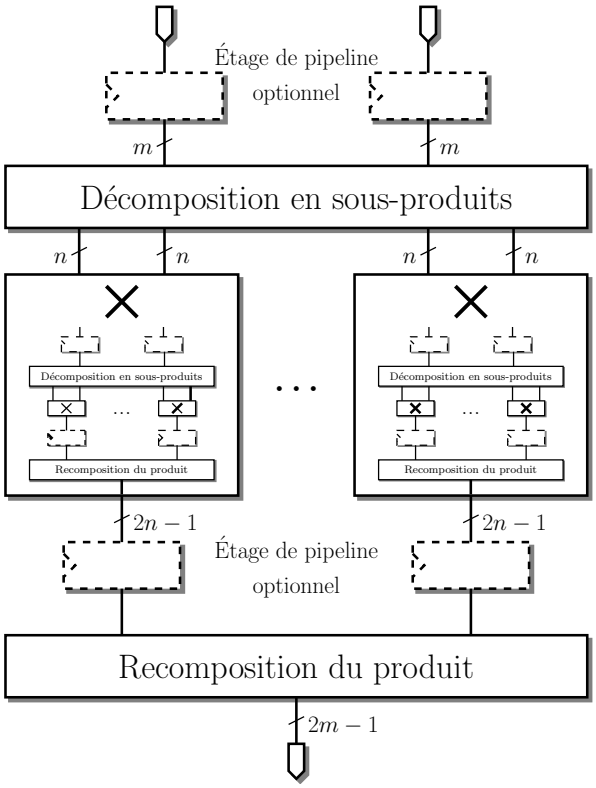


FIGURE 11.3 – Architecture du multiplieur parallèle pour les polynômes sur $\mathbb{F}_2$ ou $\mathbb{F}_3$.

11.1.2 Choix des algorithmes de multiplications

Dans cette section nous nous intéressons au choix des algorithmes de multiplication à utiliser pour chaque étage de récursion dans l'architecture précédente. En effet, l'architecture précédente ne demande pas d'utiliser les mêmes algorithmes pour chaque étage et grâce à notre générateur, nous pouvons explorer une grande variété de combinaisons de ces algorithmes.

Tout d'abord, nous nous sommes intéressés au dernier étage de la récursion. En effet, les algorithmes de multiplication sous-quadratiques sont notoirement inefficaces pour les polynômes de petit degré. Il est donc important d'utiliser des multiplieurs naïfs pour ceux-ci car ils ont une complexité moins grande. Ces multiplieurs implémentent directement l'algorithme que nous apprenons à l'école primaire : production des produits partiels et accumulation de ceux-ci. De façon surprenante, nous verrons dans la Table 11.4 que les multiplieurs sous-quadratiques restent avantageux jusqu'à des tailles relativement petites : nous utilisons un multiplieur naïf pour des polynômes ayant moins de 10 coefficients en caractéristique 2 et moins de 4 en caractéristique 3.

Parmi les différentes formules de multiplications que nous avons vues à la Section 8.2, toutes ne sont pas utilisables dans cette architecture. Nous avons utilisé les différentes variations de l'algorithme de Karatsuba présenté dans la Section 8.2.2. Pour les raisons exposées dans la Section 8.2.3, il est difficile d'utiliser les schémas de multiplication par évaluation-interpolation en petite caractéristique. En effet, nous ne disposons pas de suffisamment de points d'interpolation dans le corps où vivent les coefficients (ici $\mathbb{F}_2$ ou $\mathbb{F}_3$) et nous devons alors faire intervenir la variable libre définissant l'anneau de polynômes. Ainsi, le coût de la recomposition du produit deviendrait prohibitif car il ferait intervenir des produits supplémentaires. Lors de la conception de ce multiplieur, nous n'avions pas encore étudié les formules de Montgomery ni conçu l'algorithme du Chapitre 9, il serait donc intéressant de poursuivre la conception de ce multiplieur parallèle en intégrant ces nouvelles formules. Cependant, les complexités asymptotiques de celles-ci étant moins bonnes que celles fournies par les différentes variations de l'algorithme de Karatsuba, nous n'envisageons pas qu'elles puissent conduire à une nette amélioration des performances observées sur cette architecture.

Nous nous intéressons maintenant au comportement des différentes formules de multiplication issues de celle de Karatsuba. La première variation consiste à découper les opérandes en 2 ou 3 parties. Les extensions $\mathbb{F}_{p^m}$ que nous utilisons généralement étant de degré m premier, le découpage de ces opérandes n'est jamais exacte et ainsi les sous-produits à calculer voient leurs coefficients de poids fort éventuellement nuls. Cette variation est alors essentielle car elle permet de minimiser le nombre de zéros qui apparaissent au cours du calcul ce qui permet éventuellement d'obtenir de meilleures performances.

La deuxième variation a été introduite par Hanrot et Zimmermann lors d'un travail sur le produit court de Mudlers [HZ04]. Elle consiste, dans le cas d'une découpe en 2 parties, à séparer les opérandes en parties paire et impaire plutôt qu'en parties haute et basse. Cette amélioration prend toute son importance dans le cas d'une implémentation matérielle car elle permet de réduire la profondeur des additions à effectuer lors de l'étape de reconstruction du résultat sans rajouter de sous-produit à effectuer : elle passe ainsi de 4 à 3. Nous avons étendu cette approche au cas du découpage en 3 parties, faisant passer la profondeur de la somme pour la reconstruction de 10 à 4.

Nous avons implémenté un générateur de VHDL pour cette architecture. Nous pouvons ainsi tester un grand nombre de multiplieurs en utilisant différents algorithmes pour chaque étage de la récursion. Nous présentons quelques-uns de ces multiplieurs dans la Table 11.4

TABLE 11.4 – Différents multiplieurs parallèles générés pour Virtex-II Pro.

Corps	**Récursion**	**Temps [ns]**	**Fréq. [MHz]**	**Surface [slices]**	**AT [μs·slices]**
$\mathbb{F}_{2^{239}}$	$239 \xrightarrow{2} 120 \xrightarrow{2} 60 \xrightarrow{2} 30 \xrightarrow{2} 15 \xrightarrow{2} 8$	7.477	134	9028	67.5
	$239 \xrightarrow{3'} 80 \xrightarrow{2} 40 \xrightarrow{2} 20 \xrightarrow{2} 10 \xrightarrow{2} 5$	6.798	147	9218	62.7
$\mathbb{F}_{2^{313}}$	$313 \xrightarrow{2} 157 \xrightarrow{2} 79 \xrightarrow{2} 40 \xrightarrow{2} 20 \xrightarrow{2} 10$	8.143	123	13708	112
$\mathbb{F}_{3^{97}}$	$97 \xrightarrow{2} 49 \xrightarrow{2} 25 \xrightarrow{2} 13 \xrightarrow{2} 13 \xrightarrow{2} 7 \xrightarrow{2} 4$	7.987	125	10403	83.1
	$97 \xrightarrow{2'} 49 \xrightarrow{2'} 25 \xrightarrow{2'} 13 \xrightarrow{2'} 13 \xrightarrow{2'} 7 \xrightarrow{2'} 4$	7.694	130	10332	79.5
	$97 \xrightarrow{3'} 33 \xrightarrow{3'} 11 \xrightarrow{3'} 4$	7.298	137	12736	92.9
$\mathbb{F}_{3^{167}}$	$167 \xrightarrow{2'} 84 \xrightarrow{2'} 42 \xrightarrow{2'} 21 \xrightarrow{2'} 11 \xrightarrow{2'} 6 \xrightarrow{2'} 3$	8.999	111	27099	244
	$167 \xrightarrow{3'} 56 \xrightarrow{3'} 19 \xrightarrow{3'} 7 \xrightarrow{2'} 4$	9.102	110	28662	261

Les multiplieurs ont été pipelinés en 5 étages pour la caractéristique 2 et 7 pour la caractéristique 3 afin de correspondre aux exigences des accélérateurs que nous présentons aux sections suivantes. Nous notons $\xrightarrow{2}$ l'utilisation de Karatsuba, $\xrightarrow{3}$ pour la version avec un découpage en 3 des opérandes. L'ajout de « $'$ » dénote l'utilisation des variantes paire-impaire.

parmi les meilleurs obtenus. Nous avons pu observer qu'en caractéristique 3 l'utilisation des variantes paire-impaire représente un gain non négligeable de performance.

11.2 Caractéristique 3

Nous décrivons ici une architecture *ad-hoc* pour le calcul du couplage η_T sur une courbe supersingulière en caractéristique 3.

11.2.1 Ordonnancement de la boucle de Miller

Nous étudions maintenant le corps de la boucle de Miller afin d'en déduire un ordonnancement des opérations arithmétiques et de construire une architecture adaptée à celui-ci. La boucle de Miller usuelle (Algorithme 7.7) pour le couplage η_T sur la courbe E_3 est essentiellement un algorithme de triplement-addition où une fonction rationnelle est calculée à chaque étape et où son évaluation en le point courant est accumulée multiplicativement. Ainsi, le corps de la boucle pour le calcul de $\eta_T(P,Q)$ est constitué de quatre étapes :

— mise à jour des coordonnées de Q :

$$x_Q \leftarrow x_Q^9 \pm 1;$$
$$y_Q \leftarrow y_Q^9;$$

— calcul de la fonction rationnelle et évaluation en $\psi(Q)$:

$$t \leftarrow x_P + x_Q;$$
$$G \leftarrow -t^2 + y_P y_Q \sigma - t\rho - \rho^2;$$

— accumulation du résultat (notons que c'est un produit dans $\mathbb{F}_{3^{6m}}$) :

$$F \leftarrow F \cdot G;$$

— élévation à la puissance 3 de l'accumulateur (application de l'automorphisme de Frobenius) :

$$F \leftarrow F^3;$$

Comme nous utiliserons un multiplieur pipeliné, nous nous intéressons en premier lieu à la répartition des produits dans le corps de la boucle. Nous avons besoin d'effectuer 2 multiplications sur le corps de base pour l'évaluation de la fonction rationnelle et un produit sur l'extension $\mathbb{F}_{3^{6m}}$. Le choix d'un algorithme de multiplication pour cette accumulation demande une attention particulière. En effet, un des opérandes G est creux ; l'approche classique est de minimiser le nombre de produits sur le corps de base en utilisant l'algorithme de Gorla *et al.* [GPS07]. Celui-ci demande 12 multiplications et 59 additions sur le corps de base. Cependant nous disposons d'un multiplieur rapide pour cette architecture et l'utilisation de cet algorithme demandera d'utiliser beaucoup de surface du FPGA pour effectuer les additions suffisamment rapidement par rapport aux produits. Comme cette approche se montre inefficace, nous utiliserons un autre algorithme de multiplication échangeant la moitié des additions contre 3 multiplications. Nous utiliserons ainsi l'algorithme de Beuchat *et al.* [Beu+10a] qui ne demande que 15 multiplications et 29 additions afin de profiter au mieux du multiplieur parallèle.

Nous portons maintenant notre attention sur l'ordonnancement des multiplications. En effet, le multiplieur que nous utilisons est pipeliné et nous devons nous assurer qu'il est possible d'effectuer les 17 multiplications du corps de la boucle sans introduire de bulle dans le pipeline du multiplieur. Or nous pouvons observer qu'il y a une dépendance entre l'élévation à la puissance 3 de l'accumulateur ($F \leftarrow F^3$) et le produit $F \cdot G$ de l'itération suivante. Le problème qui en résulte est que le calcul du Frobenius F^3 agit comme une barrière de synchronisation car il faut, pour obtenir chaque coordonnée de F^3, la plupart des coordonnées de F. Ainsi, il faut avoir terminé le calcul d'un produit d'accumulation pour pouvoir commencer le suivant.

Nous utiliserons donc un autre algorithme pour le calcul du couplage non réduit : nous utiliserons la variante qui exécute la boucle de Miller en partant de la fin et fait apparaître des racines cubiques pour faire disparaître le calcul de F^3 (Algorithme 7.9).

Nous récrivons alors cet algorithme afin d'exhiber un corps de boucle plus régulier et d'être plus proche de l'architecture de l'accélérateur que nous construisons (Algorithme 11.5). Nous retrouvons dans cette version de l'algorithme quasiment les mêmes étapes que précédemment :

— mise à jour des coordonnées de P et Q (lignes 8 et 9),
— calcul de la fonction rationnelle (lignes 10 et 6),
— accumulation du résultat (ligne 7).

Nous avons maintenant décrit l'algorithme complet pour le calcul du couplage non réduit dont le coût total de chaque itération est 17 multiplications, 30 additions, 2 applications du Frobenius (cube) et 2 du Frobenius inverse (racine cubique).

Nous nous attardons maintenant sur l'accumulation du résultat en présentant l'algorithme de multiplication creuse que nous utilisons (voir Algorithme 11.6). Il est alors possible de trouver un ordonnancement des 17 produits de la boucle de Miller tel que nous pourrons utiliser un multiplieur comportant jusqu'à 7 étages de pipeline. L'ordonnancement final des opérations arithmétiques (voir Figure 11.8) est construit en même temps que l'architecture que nous décrivons à la section suivante.

Algorithme 11.5 Calcul du couplage η_T non réduit en caractéristique 3.

Les variables notées en majuscules vivent dans l'extension $\mathbb{F}_{3^{6m}}$, et celles en minuscules sont dans le corps de base $\mathbb{F}_{3^m}$. Les exposants entre parenthèses représentent l'itération où la variable est utilisée.

Entrées : $P = (x_P, y_P)$ et $Q = (x_Q, y_Q) \in E(\mathbb{F}_{3^m})[\ell]$.
Sortie : $f_{3^{\frac{m+1}{2}}+\mu,P}(Q) \in \mathbb{F}^*_{3^{6m}}$.

1. $x_P^{(0)} \leftarrow x_P - \nu$; $y_p^{(0)} \leftarrow -\mu y_P$;
2. $x_Q^{(0)} \leftarrow x_Q$; $y_Q^{(0)} \leftarrow -\lambda y_Q$;
3. $t^{(0)} \leftarrow x_P^{(0)} + x_Q^{(0)}$;
4. $F^{(-1)} \leftarrow \lambda y_P^{(0)} \cdot t^{(0)} - \lambda y_Q^{(0)}\sigma - \lambda y_P^{(0)}\rho$;
5. **for** $i = 0$ to $(m-1)/2$ **do**
6. $\quad G^{(i)} \leftarrow -\left(t^{(i)}\right)^2 + y_P^{(i)} y_Q^{(i)}\sigma - t^{(i)}\rho - \rho^2$;
7. $\quad F^{(i)} \leftarrow F^{(i-1)} \cdot G^{(i)}$;
8. $\quad x_P^{(i+1)} \leftarrow \sqrt[3]{x_P^{(i)}}$; $y_P^{(i+1)} \leftarrow \sqrt[3]{y_P^{(i)}}$;
9. $\quad x_Q^{(i+1)} \leftarrow \left(x_Q^{(i)}\right)^3$; $y_Q^{(i+1)} \leftarrow \left(y_Q^{(i)}\right)^3$;
10. $\quad t^{(i+1)} \leftarrow x_P^{(i)} + x_Q^{(i)}$;
11. **end for**
12. **return** $F^{((m-1)/2)}$;

11.2.2 Architecture de l'accélérateur

Les opérations arithmétiques impliquées dans le calcul du couplage non réduit peuvent être classées en deux ensembles relativement à la latence des opérateurs les réalisant : d'une part nous trouvons les opérations **linéaires** comme les additions et les applications du Frobenius, d'autre part les **multiplications**. Comme la fréquence de l'accélérateur complet est celle de sa partie la moins rapide, nous avons tout intérêt à utiliser un multiplieur fonctionnant à fréquence élevée ; ainsi nous utilisons le maximum d'étages de pipeline possibles pour notre multiplieur afin d'augmenter sa fréquence. Cependant, l'ordonnancement des opérations nous autorise 7 étages au maximum ce qui fait qu'un additionneurs à deux opérandes dans le corps de base $\mathbb{F}_{3^m}$ bénéficie encore d'une fréquence de fonctionnement plus élevée que le multiplieur. C'est pourquoi nous avons choisi d'effectuer nos additions avec un additionneur à 4 opérandes, ceci nous permet également de s'occuper des 30 additions de chaque itération en les 17 cycles impartis. En effet, la multiplication creuse demande d'additionner jusqu'à 4 valeurs et nous ne pourrions utiliser un additionneur avec plus d'entrées.

Afin de concevoir l'accélérateur, nous avons commencé par isoler l'étape de mise à jour des coordonnées de P et Q (lignes 8, 9 et 10, Algorithme 11.5) et nous avons développé un circuit réalisant ce calcul qui est présenté dans la partie supérieure de la Figure 11.7. Cette partie de l'opérateur permet aussi d'effectuer les calculs d'initialisation de ces coordonnées (lignes 1, 2 et 3) et constitue ainsi le point d'entrée de l'accélérateur. Nous avons besoin ici de deux opérateurs pour le calcul de l'automorphisme de Frobenius ($x \mapsto x^3$) et du Frobenius inverse ($x \mapsto \sqrt[3]{x}$) ; ces opérations peuvent être calculées en effectuant des sommes de différents coefficients de l'entrée pour chaque coefficient en sortie. En choisissant correctement le

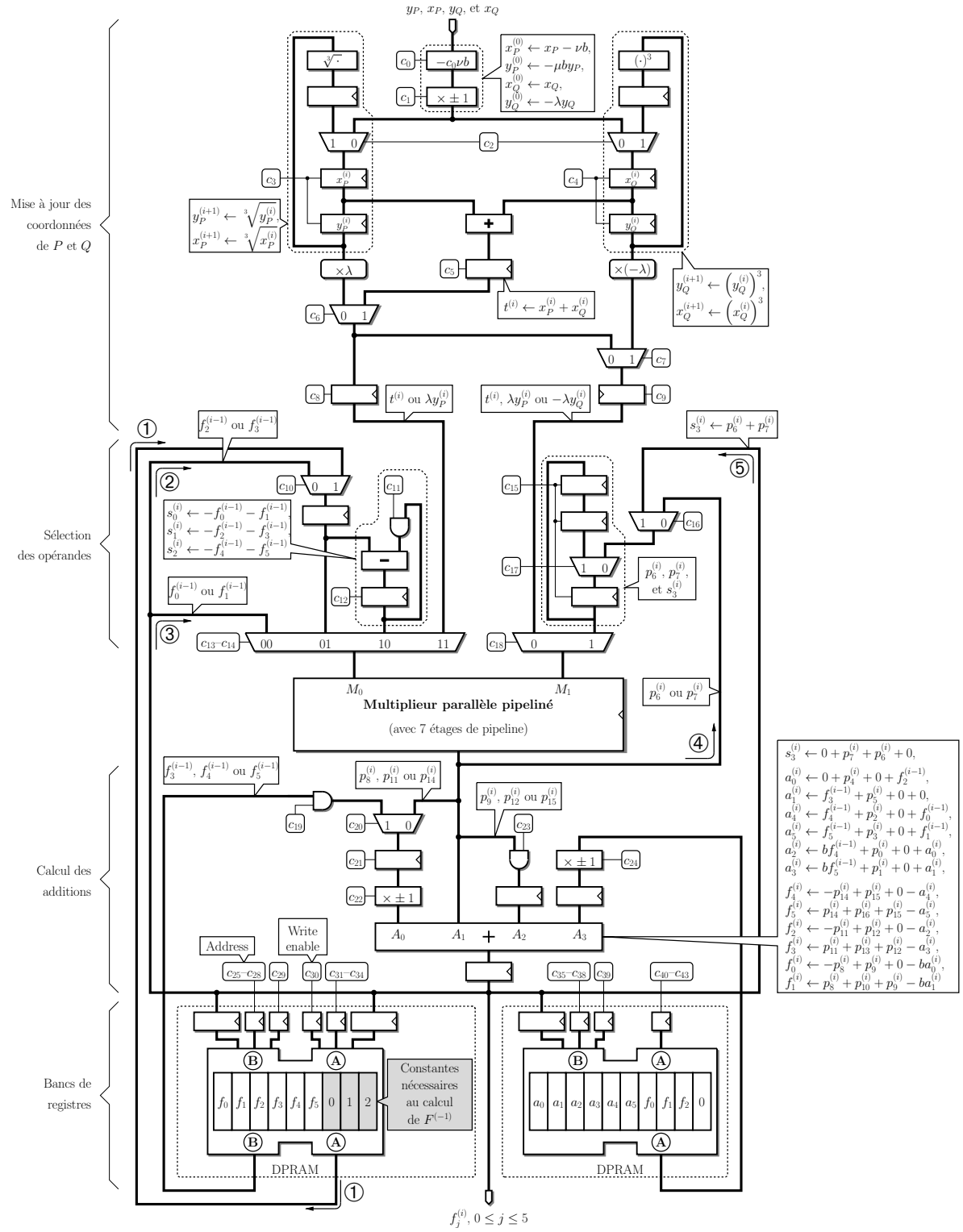


FIGURE 11.7 – Architecture de l'accélérateur pour la caractéristique 3.

Algorithme 11.6 Multiplication creuse sur $\mathbb{F}_{3^{6m}}$ pour le calcul du couplage η_T.

Entrées : $b \in \{-1, 1\}$; $t^{(i)}$, $y_P^{(i)}$, et $y_Q^{(i)} \in \mathbb{F}_{3^m}$; $F^{(i-1)} \in \mathbb{F}_{3^{6m}}$.

Sortie : $F^{(i)} = F^{(i-1)} \cdot G^{(i)} \in \mathbb{F}_{3^{6m}}$, où $G^{(i)} = -\left(t^{(i)}\right)^2 + y_P^{(i)} y_Q^{(i)} \sigma - t^{(i)} \rho - \rho^2$.

1. $p_0^{(i)} \leftarrow f_0^{(i-1)} \cdot t^{(i)}$; $p_1^{(i)} \leftarrow f_1^{(i-1)} \cdot t^{(i)}$; $p_2^{(i)} \leftarrow f_2^{(i-1)} \cdot t^{(i)}$;
2. $p_3^{(i)} \leftarrow f_3^{(i-1)} \cdot t^{(i)}$; $p_4^{(i)} \leftarrow f_4^{(i-1)} \cdot t^{(i)}$; $p_5^{(i)} \leftarrow f_5^{(i-1)} \cdot t^{(i)}$;
3. $p_6^{(i)} \leftarrow t^{(i)} \cdot t^{(i)}$; $p_7^{(i)} \leftarrow -y_P^{(i)} \cdot y_Q^{(i)}$;
4. $s_0^{(i)} \leftarrow -f_0^{(i-1)} - f_1^{(i-1)}$; $s_1^{(i)} \leftarrow -f_2^{(i-1)} - f_3^{(i-1)}$;
5. $s_2^{(i)} \leftarrow -f_4^{(i-1)} - f_5^{(i-1)}$; $s_3^{(i)} \leftarrow p_6^{(i)} + p_7^{(i)}$;
6. $a_0^{(i)} \leftarrow f_2^{(i-1)} + p_4^{(i)}$; $a_2^{(i)} \leftarrow br_4^{(i-1)} + p_0^{(i)} + a_0^{(i)}$; $a_4^{(i)} \leftarrow f_0^{(i-1)} + f_4^{(i-1)} + p_2^{(i)}$;
7. $a_1^{(i)} \leftarrow f_3^{(i-1)} + p_5^{(i)}$; $a_3^{(i)} \leftarrow br_5^{(i-1)} + p_1^{(i)} + a_1^{(i)}$; $a_5^{(i)} \leftarrow f_1^{(i-1)} + f_5^{(i-1)} + p_3^{(i)}$;
8. $p_8^{(i)} \leftarrow f_0^{(i-1)} \cdot p_6^{(i)}$; $p_9^{(i)} \leftarrow f_1^{(i-1)} \cdot p_7^{(i)}$; $p_{10}^{(i)} \leftarrow s_0^{(i)} \cdot s_3^{(i)}$;
9. $p_{11}^{(i)} \leftarrow f_2^{(i-1)} \cdot p_6^{(i)}$; $p_{12}^{(i)} \leftarrow f_3^{(i-1)} \cdot p_7^{(i)}$; $p_{13}^{(i)} \leftarrow s_1^{(i)} \cdot s_3^{(i)}$;
10. $p_{14}^{(i)} \leftarrow f_4^{(i-1)} \cdot p_6^{(i)}$; $p_{15}^{(i)} \leftarrow f_5^{(i-1)} \cdot p_7^{(i)}$; $p_{16}^{(i)} \leftarrow s_2^{(i)} \cdot s_3^{(i)}$;
11. $f_0^{(i)} \leftarrow -ba_0^{(i)} - p_8^{(i)} + p_9^{(i)}$; $f_1^{(i)} \leftarrow -ba_1^{(i)} + p_8^{(i)} + p_9^{(i)} + p_{10}^{(i)}$;
12. $f_2^{(i)} \leftarrow -a_2^{(i)} - p_{11}^{(i)} + p_{12}^{(i)}$; $f_3^{(i)} \leftarrow -a_3^{(i)} + p_{11}^{(i)} + p_{12}^{(i)} + p_{13}^{(i)}$;
13. $f_4^{(i)} \leftarrow -a_4^{(i)} - p_{14}^{(i)} + p_{15}^{(i)}$; $f_5^{(i)} \leftarrow -a_5^{(i)} + p_{14}^{(i)} + p_{15}^{(i)} + p_{16}^{(i)}$;
14. **return** $f_0^{(i)} + f_1^{(i)}\sigma + f_2^{(i)}\rho + f_3^{(i)}\sigma\rho + f_4^{(i)}\rho^2 + f_5^{(i)}\sigma\rho^2$;

trinôme ou le pentanôme définissant le corps de base $\mathbb{F}_{3^m}$, ces sommes n'impliquent que peu de coefficients d'entrée ; ainsi ces opérateurs sont rapides et nos expérimentations ont montré que le chemin critique du circuit ne se trouve jamais dans les opérateurs pour le Frobenius et le Frobenius inverse. Cette partie du calcul ne demandant aucun produit, elle n'est connectée au reste du circuit que par deux chemins permettant de fournir les valeurs de $t^{(i)}$, $\lambda y_P^{(i)}$ et $-\lambda y_Q^{(i)}$ nécessaires lors de l'initialisation (ligne 4, Algorithme 11.5) et au calcul de la fonction rationnelle (ligne 6).

Le reste du circuit s'articule autour d'un multiplieur parallèle pipeliné à 7 étages : une partie du circuit servant à la sélection des opérandes en entrée du multiplieur, une autre du calcul des additions en sortie de celui-ci. Le calcul du couplage demandant un grand nombre de variables intermédiaires qui ne sont pas accédées suivant un schéma suffisamment régulier, celles-ci ne peuvent être stockées dans des registres répartis dans le circuit. Ainsi, nous avons doté le circuit de deux bancs de registres pour stocker d'une part les variables intermédiaires, d'autre part la valeur de la variable d'accumulation F. Notons que bien que nous ayons implémenté les deux bancs de registres grâce à des blocs de RAM à deux ports, la pression en lecture sur ceux-ci est telle que nous devons stocker deux copies des variables f_0, f_1 et f_2 pour pouvoir y accéder en même temps.

Afin d'étudier le circuit plus en détail, nous portons maintenant notre attention sur la Figure 11.8 qui montre l'ordonnancement de toutes les opérations arithmétiques pour une itération de la boucle de Miller, l'essentiel des opérations consistant en le produit creux d'accumulation sur l'extension $\mathbb{F}_{3^{6m}}$ (Algorithme 11.6). En effet, l'ordonnancement est l'étape

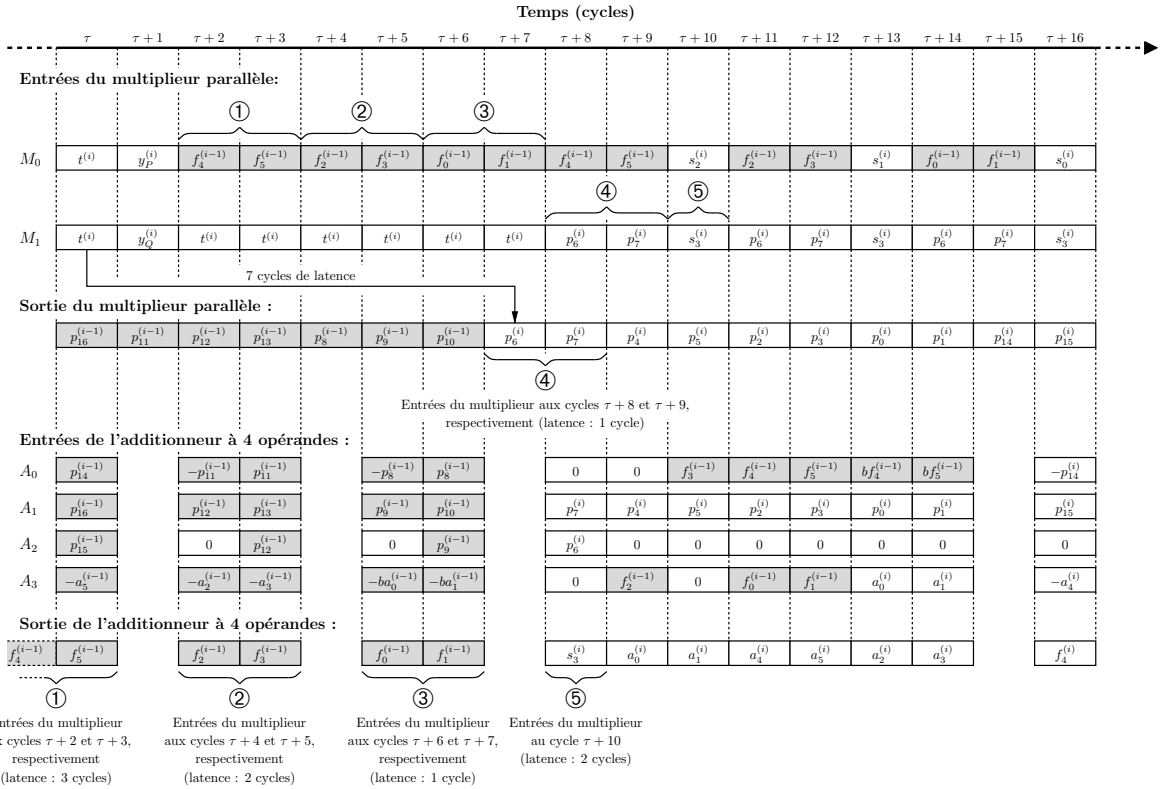


FIGURE 11.8 – Ordonnancement des opérations arithmétiques de la boucle de Miller pour le couplage η_T en caractéristique 3.

cruciale dans la conception de cette architecture, et afin de fournir le multiplieur en opérandes nous avons mis 5 chemins (notés de ① à ⑤) pour alimenter le multiplieur depuis les bancs de registres, la sortie de l'additionneur et du multiplieur. En effet, le temps qui sépare la production d'un produit et son utilisation (après d'éventuelles additions) dans le multiplieur varie selon les coefficients. Ainsi, certaines valeurs sont accédées depuis les bancs de registres (chemin ①), d'autres sont prises depuis l'additionneur directement (chemin ③) ou en passant par un registre afin d'ajouter une éventuelle latence (chemins ② et ⑤), enfin d'autres multiplicandes viennent du multiplieur directement (chemin ④). Le reste du circuit permet alors d'aller chercher les termes des différentes additions. Notons que 3 de ces additions ne sont pas calculées dans l'additionneur à 4 opérandes mais sont réalisées dans le circuit de sélections des multiplicandes (calcul des coefficients s_0, s_1 et s_2).

Cet accélérateur est contrôlé par 44 bits (notées c_0 à c_{43} sur la Figure 11.7) qui constituent les entrées de sélection pour les multiplexeurs et le contrôle des bancs de registres (lecture/écriture, adresses). Ces bits sont générés par un automate fini qui les lit en mémoire et gère les itérations de la boucle de Miller.

In fine, le calcul de boucle de Miller demande 17 cycles par itération auxquels il faut ajouter 17 cycles pour l'initialisation. L'accélérateur a donc une latence totale de $17\frac{m+3}{2}$ cycles.

11.3 Caractéristique 2

Nous décrivons maintenant l'architecture correspondante pour le calcul du couplage η_T sur une courbe elliptique supersingulière de caractéristique 2. Elle a été construite avec les mêmes idées que celle pour la caractéristique 3 mais les différences entre les deux algorithmes imposent de concevoir une architecture nouvelle.

11.3.1 Ordonnancement de la boucle de Miller

Pour les mêmes raisons qu'en caractéristique 3, nous utilisons une version de l'algorithme de doublement-addition pour calcul du couplage η_T qui ne demande pas d'appliquer le Frobenius (ici, la mise au carré) à la variable d'accumulation. En effet, cela jouerait également le rôle de barrière de synchronisation et nous ne pourrions utiliser efficacement le multiplieur pipeliné pour le calcul du produit creux d'accumulation sur l'extension $\mathbb{F}_{2^{4m}}$. Ainsi, nous utiliserons également une version de l'algorithme où la boucle de Miller a été renversée et la mise à jour des coordonnées des points P et Q modifiée pour que le carré sur $\mathbb{F}_{2^{4m}}$ disparaisse (Algorithme 7.2). Nous réécrivons cet algorithme afin d'identifier un corps de boucle compact permettant d'éloigner les dépendances entre les différentes variables (Algorithme 11.9).

En caractéristique 2, les calculs se passent dans une extension de degré moins grand qu'en caractéristique 3 et ainsi le produit d'accumulation sur l'extension (ligne 14, Algorithme 11.9) demande moins de multiplications dans le corps de base pour être calculé. En effet, nous avons choisi un algorithme qui apparaît dans [Beu+08a] et qui minimise le nombre de produits à effectuer sur le corps de base comme le montrent facilement les méthodes proposées au Chapitre 9. Cet algorithme ne demande que 6 multiplications et 14 additions.

Ainsi, il faut calculer 7 produits, 17 sommes, 2 applications du Frobenius ($x \mapsto x^2$) et 2 du Frobenius inverse ($x \mapsto \sqrt{x}$) par itération de la boucle de Miller. Nous avons alors cherché un ordonnancement de cet algorithme afin de dimensionner correctement la profondeur du pipeline du multiplieur. L'ordonnancement de la Figure 11.11 montre qu'il est possible d'utiliser jusqu'à 5 étages de pipeline. En effet, nous pouvons constater qu'il n'est pas possible d'en mettre plus sans introduire de bulle dans le pipeline. Comme il y a 7 multiplications, il faut être capable d'ordonnancer l'itération en 7 cycles, or les produits m_0 et m_1 forment un cycle avec les deux sommes f_0 et f_1 qui prennent chacun un cycle : il n'en reste alors que 5 pour chaque multiplication.

11.3.2 Architecture de l'accélérateur

Tout comme en caractéristique 3, l'accélérateur est construit de façon *ad-hoc* autour des opérations impliquées dans la boucle et adapté pour les étapes d'initialisation. Nous nous attardons maintenant sur les opérations itérées.

Comme la quantité d'opérations demandées par une itération de la boucle de Miller est beaucoup moins grande qu'en caractéristique 3, nous avons pu obtenir un ordonnancement plus régulier ; nous pouvons ainsi nous passer de DPRAM pour implémenter un banc de registre et chaque variable intermédiaire est ainsi stockée dans des registres et des files de registres (FIFO) répartis dans le circuit. Nous décrivons donc, dans la suite, chacune des parties du circuit ainsi que les valeurs qu'elles enregistrent.

L'accélérateur est présenté à la Figure 11.12 et nous détaillons ici son fonctionnement. La première partie du circuit est dédiée à la mise à jour des points P et Q (lignes 17 et 18,

Algorithme 11.9 Calcul du couplage η_T non réduit en caractéristique 2.

Entrées : $P = (x_P, y_P)$, $Q = (x_Q, y_Q) \in E_2(\mathbb{F}_{2^m})[\ell]$.
Sortie : $f_{2^{\frac{m+1}{2}}+\nu,P}(Q) \in \mathbb{F}^*_{2^{4m}}$.

1. $x_P^{(0)} \leftarrow x_P$; $\quad y_P^{(0)} \leftarrow y_P + \bar{\delta}$;
2. $x_Q^{(0)} \leftarrow x_Q$; $\quad y_Q^{(0)} \leftarrow y_Q$;
3. $u^{(0)} \leftarrow x_P^{(0)} + \alpha$; $\quad v^{(0)} \leftarrow x_Q^{(0)} + \alpha$;
4. $w^{(0)} \leftarrow y_P^{(0)} + y_Q^{(0)} + \beta$;
5. $g_0^{(0)} \leftarrow u^{(0)} \cdot v^{(0)} + w^{(0)}$;
6. $g_1^{(0)} \leftarrow u^{(0)} + v^{(0)} + \alpha$; $g_2^{(0)} \leftarrow v^{(0)} + \left(x_P^{(0)}\right)^2$;
7. $x_P^{(1)} \leftarrow \sqrt{x_P^{(0)}}$; $\quad y_P^{(1)} \leftarrow \sqrt{y_P^{(0)}}$;
8. $x_Q^{(1)} \leftarrow \left(x_Q^{(0)}\right)^2$; $\quad y_Q^{(1)} \leftarrow \left(y_Q^{(0)}\right)^2$;
9. $u^{(1)} \leftarrow x_P^{(1)} + \alpha$; $\quad v^{(1)} \leftarrow x_Q^{(1)} + \alpha$;
10. $w^{(1)} \leftarrow y_P^{(1)} + y_Q^{(1)} + \beta$;
11. $F^{(-1)} \leftarrow \left(g_0^{(0)} + g_2^{(0)}\right) + \left(g_1^{(0)} + 1\right) s + t$;
12. **for** $i = 1$ **to** $\frac{m-1}{2}$ **do**
13. $\quad G^{(i)} \leftarrow g_0^{(i)} + g_1^{(i)} s + t$;
14. $\quad F^{(i)} \leftarrow F^{(i-1)} \cdot G^{(i)}$;
15. $\quad g_0^{(i+1)} \leftarrow u^{(i+1)} \cdot v^{(i+1)} + w^{(i+1)}$;
16. $\quad g_1^{(i+1)} \leftarrow u^{(i+1)} + v^{(i+1)} + \alpha$;
17. $\quad x_P^{(i+2)} \leftarrow \sqrt{x_P^{(i+1)}}$; $\quad y_P^{(i+2)} \leftarrow \sqrt{y_P^{(i+1)}}$;
18. $\quad x_Q^{(i+2)} \leftarrow \left(x_Q^{(i+1)}\right)^2$; $\quad y_Q^{(i+2)} \leftarrow \left(y_Q^{(i+1)}\right)^2$;
19. $\quad u^{(i+2)} \leftarrow x_P^{(i+2)} + \alpha$; $\quad v^{(i+2)} \leftarrow x_Q^{(i+2)} + \alpha$;
20. $\quad w^{(i+2)} \leftarrow y_P^{(i+2)} + y_Q^{(i+2)} + \beta$;
21. **end for**
22. **return** $F^{((m-1)/2)}$;

Algorithme 11.10 Multiplication creuse sur $\mathbb{F}_{2^{4m}}$ pour le calcul du couplage η_T.

Entrées : $G^{(i)} = g_0^{(i)} + g_1^{(i)}s + t \in \mathbb{F}_{2^{4m}}$ et
$F^{(i-1)} = f_0^{(i-1)} + f_1^{(i-1)}s + f_2^{(i-1)}t + f_3^{(i-1)}st \in \mathbb{F}_{2^{4m}}$.
Sortie : $F^{(i)} = G^{(i)} \cdot F^{(i-1)}$.

1. $a_0^{(i)} \leftarrow g_0^{(i)} + g_1^{(i)}$; $a_1^{(i)} \leftarrow f_0^{(i-1)} + f_1^{(i-1)}$;
2. $a_2^{(i)} \leftarrow f_2^{(i-1)} + f_3^{(i-1)}$;
3. $m_0^{(i)} \leftarrow g_0^{(i)} \cdot f_0^{(i-1)}$; $m_1^{(i)} \leftarrow g_1^{(i)} \cdot f_1^{(i-1)}$;
4. $m_2^{(i)} \leftarrow a_0^{(i)} \cdot a_1^{(i)}$; $m_3^{(i)} \leftarrow g_0^{(i)} \cdot f_2^{(i-1)}$;
5. $m_4^{(i)} \leftarrow g_1^{(i)} \cdot f_3^{(i-1)}$; $m_5^{(i)} \leftarrow a_0^{(i)} \cdot a_2^{(i)}$;
6. $f_0^{(i)} \leftarrow m_0^{(i)} + m_1^{(i)} + f_3^{(i-1)}$;
7. $f_1^{(i)} \leftarrow m_0^{(i)} + m_2^{(i)} + f_2^{(i-1)} + f_3^{(i-1)}$;
8. $f_2^{(i)} \leftarrow m_3^{(i)} + m_4^{(i)} + f_0^{(i-1)} + f_2^{(i-1)}$;
9. $f_3^{(i)} \leftarrow m_3^{(i)} + m_5^{(i)} + f_1^{(i-1)} + f_3^{(i-1)}$;
10. **return** $f_0^{(i)} + f_1^{(i)}s + f_2^{(i)}t + f_3^{(i)}st$;

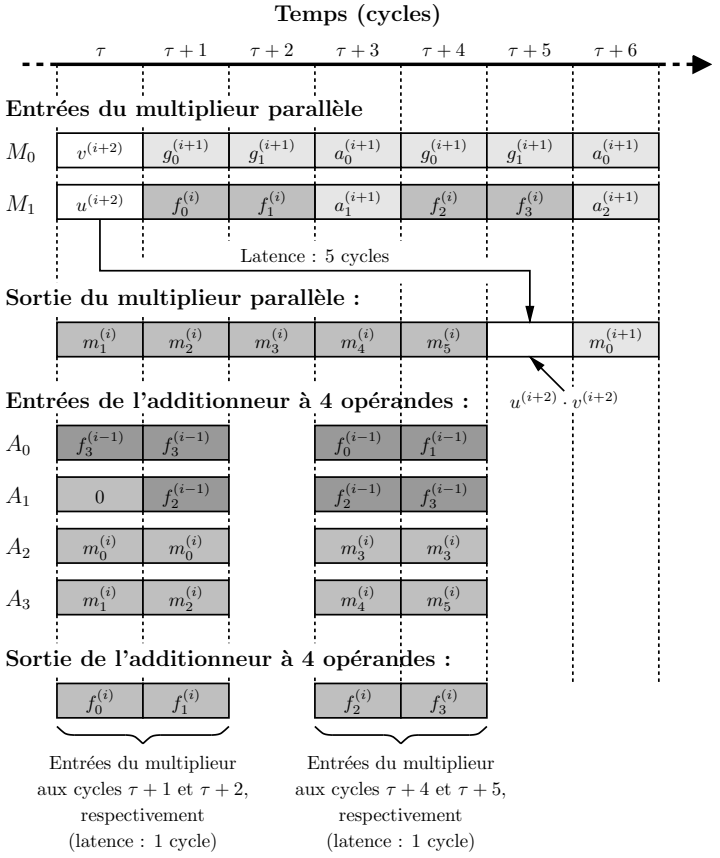


FIGURE 11.11 – Ordonnancement des opérations arithmétiques de la boucle de Miller pour le couplage η_T en caractéristique 2.

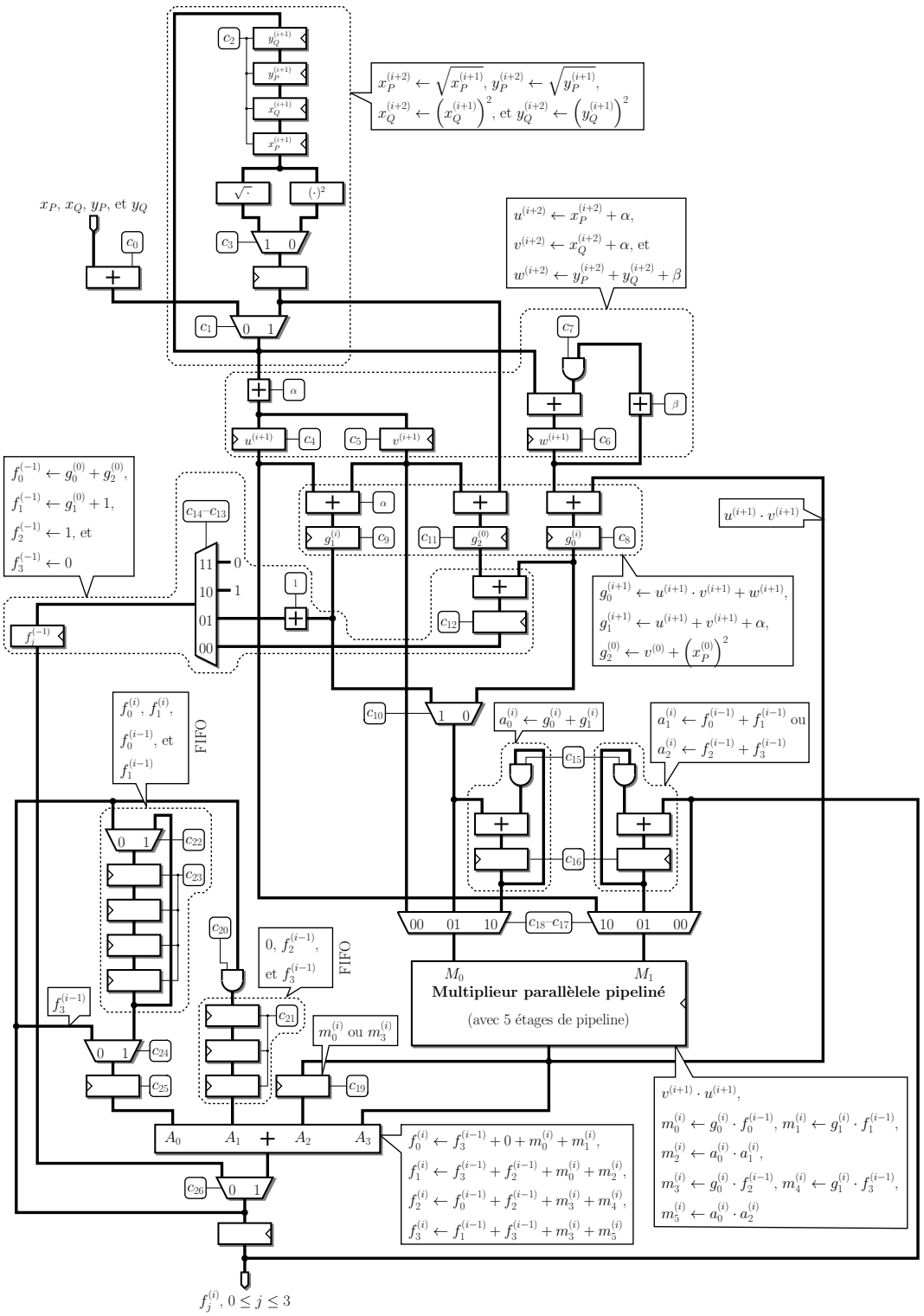


FIGURE 11.12 – Architecture de l'accélérateur pour la caractéristique 2.

Algorithme 11.9) et sert également de point d'entrée au calcul du couplage non réduit. À cette partie se connecte un circuit chargé de calculer et stocker les variables intermédiaires u, v et w qui préparent le calcul de la fonction rationnelle (lignes 19 et 20). Notons que dans ce calcul, les paramètres α et β interviennent ; comme ceux-ci sont connus au moment de la synthèse de l'accélérateur, ces valeurs ne représentent pas des bits de contrôle et sont intégrées au circuit. Le calcul continue ensuite par les coefficients de l'évaluation de la fonction rationnelle (g_0, g_1 et g_2).

À l'exception d'un morceau de circuit destiné à initialiser la valeur de la variable d'accumulation F, le reste du circuit s'articule autour d'un multiplieur parallèle pipeliné à 5 étages et permet, pour l'essentiel, d'effectuer le produit creux présenté dans l'Algorithme 11.10. Ainsi, nous trouvons à chaque entrée du multiplieur un multiplexeur qui permet d'aller chercher les opérandes aux différents emplacements de l'accélérateur ainsi qu'un petit circuit d'accumulation pour calculer à la volée les additions débutant le produit creux (lignes 1 et 2). En sortie du multiplieur, nous trouvons un additionneur à 4 opérandes permettant de reconstituer le résultat du produit creux une fois les sous-produits calculés (lignes 6 à 9). Cet additionneur est accompagné de deux files implémentées par des registres à décalage permettant de stocker f_0 et f_1 d'une part f_2 et f_3 d'autre part. Le contenu de ces files au cours d'une itération est détaillé dans la Figure 11.13.

L'initialisation de l'algorithme de Miller utilisé ici (Algorithme 11.9) demande, entre autres opérations, une multiplication qu'il conviendra de faire le plus tôt possible afin de pouvoir utiliser la latence du multiplieur pour effectuer une partie des autres opérations arithmétiques. L'accélérateur doit recevoir ses opérandes dans l'ordre suivant : x_P, x_Q, y_P et y_Q pour permettre au produit $u^{(0)} \cdot v^{(0)}$ d'être calculé à partir du troisième cycle. Ainsi, il faudra 15 cycles pour compléter l'initialisation, ce qui nous donne $15 + 7\frac{m+1}{2}$ cycles pour le calcul du couplage non réduit. À l'instar de la caractéristique 3, l'accélérateur est contrôlé par un automate qui lit en mémoire les 27 bits de contrôle $c_0, \ldots, c_{26}$ à chaque cycle.

Nous présentons dans la Table 11.14 les performances des différents accélérateurs obtenues pour le calcul du couplage η_T non-réduit avec l'architecture parallèle. Nous constaterons, au Chapitre 13, que cette implémentation permet non seulement d'obtenir des temps de calcul très courts par rapport au reste de la littérature mais aussi d'obtenir de très bon compromis temps-surface. Cette performance s'explique notamment par l'emploi d'algorithmes de multiplication sous-quadratiques.

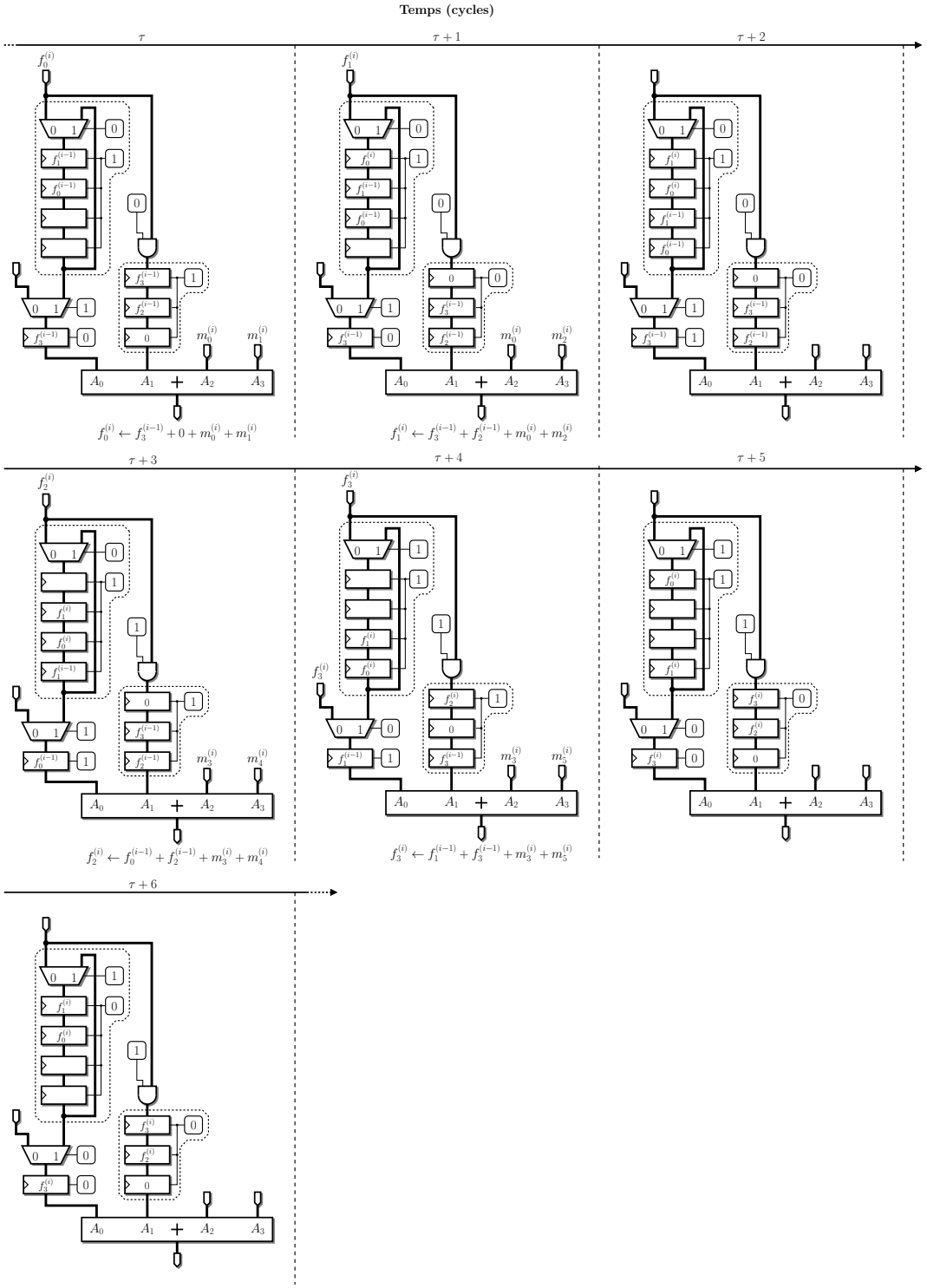


FIGURE 11.13 – Évolution des registres à décalage en entrée de l'additionneur.

TABLE 11.14 – Performances des accélérateurs parallèles pour le calcul du couplage η_T non-réduit.

Courbe	**Securité [bits]**	**FPGA**	**Surface [slices]**	**Fréquence [MHz]**	**Temps de calcul [μs]**	**AT [ms·slices]**
$E_3(\mathbb{F}_{3^{97}})$	66	Virtex-II Pro	13622	182	4.68	63.7
		Virtex-4 LX	13759	222	3.82	52.6
$E_2(\mathbb{F}_{2^{239}})$	67	Virtex-II Pro	12826	196	4.36	55.9
		Virtex-4 LX	13195	299	2.86	37.8
$E_2(\mathbb{F}_{2^{313}})$	75	Virtex-II Pro	18384	154	7.24	133
		Virtex-4 LX	19035	244	4.57	86.9
$E_3(\mathbb{F}_{3^{167}})$	83	Virtex-II Pro	33286	152	9.54	317
		Virtex-4 LX	33292	222	6.50	216
$E_2(\mathbb{F}_{2^{457}})$	88	Virtex-II Pro	37187	147	11.0	409
		Virtex-4 LX	38184	250	6.47	247
$E_3(\mathbb{F}_{3^{193}})$	89	Virtex-II Pro	37713	130	12.8	484
		Virtex-4 LX	38083	182	9.16	349
$E_2(\mathbb{F}_{2^{557}})$	96	Virtex-4 LX	47522	149	13.2	627
$E_3(\mathbb{F}_{3^{239}})$	97	Virtex-4 LX	55042	196	10.5	577
$E_2(\mathbb{F}_{2^{613}})$	100	Virtex-4 LX	53616	143	15.1	812
$E_2(\mathbb{F}_{2^{691}})$	105	Virtex-4 LX	68838	130	18.8	1290
$E_3(\mathbb{F}_{3^{313}})$	109	Virtex-4 LX	81988	159	16.9	1390

Chapitre 12

Coprocesseur arithmétique pour corps fini

Dans ce chapitre, nous décrivons un coprocesseur capable d'effectuer les opérations arithmétiques sur un corps fini de petite caractéristique. Les opérations que nous utilisons dans le cadre du calcul de couplage, et plus généralement pour les calculs sur courbes elliptiques, sont la multiplication et l'addition mais aussi l'application de l'endomorphisme de Frobenius ($x \mapsto x^p$ sur $\mathbb{F}_{p^m}$) et éventuellement de ses itérés ou de son inverse ($x \mapsto \sqrt[p]{x}$). Nous décrivons ici un coprocesseur avec une architecture paramétrable permettant des calculs sur différents corps de type $\mathbb{F}_{2^m}$ et $\mathbb{F}_{3^m}$ et de moduler les différents rapports de performance entre ces opérations arithmétiques afin de s'adapter au mieux au calcul envisagé. Cette architecture est décrite dans la Section 12.1.

Nous avons utilisé cette architecture dans trois implémentations que nous présentons dans ce chapitre :

— calcul de l'exponentiation finale pour l'accélérateur *ad hoc* présenté au chapitre précédent (Section 12.2),
— calcul du couplage η_T sur des courbes définies sur des corps de degré d'extension composé (Section 12.3),
— calcul d'un couplage η optimal sur une courbe supersingulière de genre 2 (Section 12.4).

Ainsi, ce coprocesseur permet un nombre important d'applications grâce à l'adaptabilité de son architecture tout en maintenant des performances élevées grâce aux possibilités de parallélisation des opérations qu'il procure.

12.1 Architecture du coprocesseur

Nous cherchons donc ici à concevoir un coprocesseur qui puisse être utilisé dans différents algorithmes. Nous ne pouvons donc pas reposer sur un multiplieur parallèle de grande surface contrairement à ce qui a été fait pour l'accélérateur *ad hoc* du Chapitre 11. Cependant, la multiplication restant l'opération la plus critique, nous commençons donc par décrire le multiplieur.

12.1.1 Multiplieur parallèle-série

Nous cherchons ici à réaliser un multiplieur plus compact et nous ne pouvons nous permettre d'utiliser des algorithmes sous-quadratiques de multiplication. Nous nous limitons

ainsi à l'algorithme d'accumulation des produits partiels tel qu'appris dans les petites classes. Cependant, l'implémentation de cet algorithme en matériel permet tout de même d'obtenir différents compromis temps-surface pour le calcul d'un produit.

Soit $\mathbb{F}_{p^m}$ le corps fini pour lequel nous construisons le multiplieur et f le polynôme irréductible choisi pour le représenter ($\mathbb{F}_{p^m} \cong \mathbb{F}_p[t]/(f(t))$). Soit $A = a_{m-1}t^{m-1} + \cdots + a_0$ et $B = b_{m-1}t^{m-1} + \cdots + b_0$ deux éléments de $\mathbb{F}_{p^m} \cong \mathbb{F}_p[t]$ (les a_i, b_i sont des éléments de $\mathbb{F}_p$). Le produit de A par B peut alors s'écrire

$$A \cdot B = \sum_{i=0}^{m-1} Ab_i t^i.$$

Les produits partiels $Ab_i t^i$ peuvent alors être réduits séparément avant d'être sommés. Ainsi, une première idée d'architecture est d'utiliser un registre à décalage pour l'opérande B et de générer un produit partiel à chaque cycle. Afin de pouvoir limiter les réductions moudulo $f(t)$ à effectuer, nous pouvons écrire le produit sous forme d'un schéma d'évaluation de Horner :

$$A \cdot B = Ab_0 + t(\cdots + t(Ab_{m-2} + tAb_{m-1})).$$

Ainsi, en commençant à lire B par les poids forts, nous aurons besoin, à chaque cycle, uniquement d'effectuer une réduction correspondant à la multiplication par t. Cette forme de multiplieur est appelé parallèle-série car un opérande (A) est traité en parallèle et l'autre (B) en série.

Cette approche permet d'obtenir un opérateur très compact puisqu'il ne requiert que m multiplieurs et un peu plus de m additionneurs sur $\mathbb{F}_p$ [1]. Cependant, cet opérateur induit une grande latence pour le calcul d'un produit : il faut m cycles pour chaque multiplication et il n'est pas possible de pipeliner les opérations.

Afin d'obtenir un opérateur plus rapide, il est possible de traiter plusieurs coefficients de B à la fois. Si nous traitons D coefficients à la fois, le schéma de Horner s'écrit ainsi :

$$A \cdot B = Ab_0 + Ab_1 \cdot t + \cdots Ab_{D-1}t^{D-1} + t^D(\cdots)$$

Il faudra ainsi seulement $\lceil m/D \rceil$ cycles pour effectuer la multiplication au prix de D fois plus de multiplieurs et environ D fois plus d'additionneurs sur $\mathbb{F}_p$ [2]. C'est ce multiplieur que nous utilisons dans notre coprocesseur et son architecture est décrite dans le cas de la caractéristique 3 dans la partie droite de la Figure 12.1.

12.1.2 Architecture globale

Nous décrivons ici le reste de l'architecture du coprocesseur. Comme nous souhaitons pouvoir exploiter le parallélisme entre les différentes opérations arithmétiques, nous ne souhaitons pas réutiliser certains composants du multiplieur pour effectuer les autres opérations comme cela a pu être fait dans d'autres architectures construisant un opérateur unifié [Beu+07]. Nous avons donc choisi d'implémenter dans un composant différent l'addition et le calcul du Frobenius. Comme ces opérations sont tout de même moins critiques, nous avons choisi de ne pas implémenter un opérateur pour chacune de celles-ci. En effet, les variables sont stockées

1. Si f a d coefficients non nuls, il faudra $m + d - 1$ additionneurs. Il est possible de choisir un polynôme très creux pour f.
2. La réduction du produit par t^x peut demander plus d'additions que celle correspondant au produit par t.

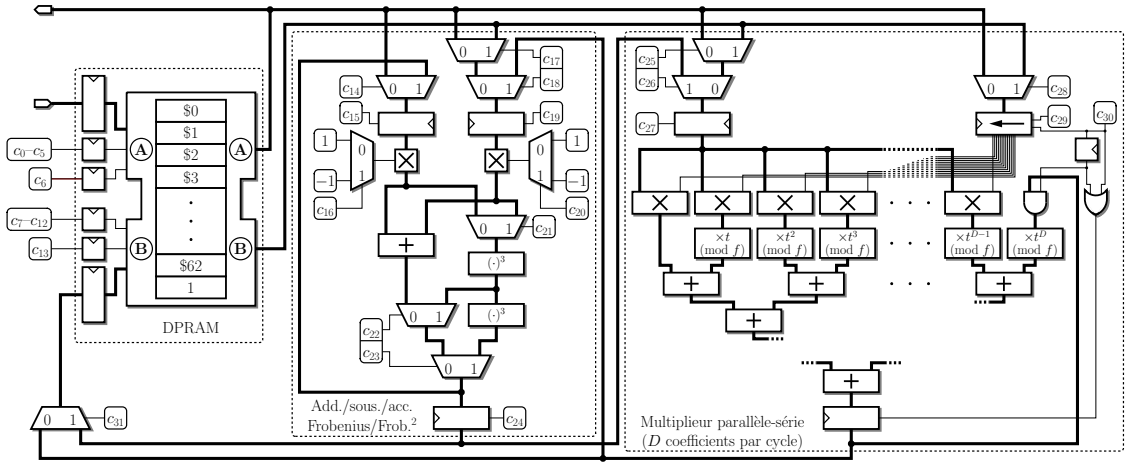


FIGURE 12.1 – Architecture d'un coprocesseur pour un corps fini $\mathbb{F}_{3^m}$.

dans un banc de registre implémenté par une mémoire « double port » (DPRAM) et nous avons constaté que le facteur le plus limitant lors de la programmation du coprocesseur est l'accès aux registres. Ainsi, rajouter un composant risque de ne pas être intéressant car il sera alors difficile de le fournir en opérandes.

Ce composant a été spécialement conçu pour les calculs sur les courbes de petite caractéristique ; en effet, il est courant de devoir calculer l'application d'itérés du Frobenius et nous avons donc choisi de fournir au composant la possibilité de calculer une double — voire une triple — application du Frobenius selon la forme du polynôme irréductible choisi et des calculs à effectuer. En effet, nous avons cherché à équilibrer la longueur des chemins critiques du multiplieur et de ce nouveau composant en lui ajoutant toutes les capacités utiles et réalisables dans le temps imparti par le multiplieur. Dans le cas de la caractéristique 3 nous avons également rajouté la possibilité de prendre les opposés de chacun des opérandes. Ce composant est décrit dans la Figure 12.1.

Ces deux composants stockent ainsi les valeurs sur lesquelles ils travaillent dans une mémoire double port et des chemins depuis ceux-ci ont donc été dessinés. Cependant, le passage par la mémoire induit une latence qui pourrait ralentir le calcul, c'est pourquoi nous avons également rajouté des chemins reliant les sorties de chaque composant avec l'une des entrées de l'autre composant, ou bien encore une boucle pour le composant réalisant les Frobenius directement de sa sortie à son entrée. Ces chemins courts permettent également de diminuer la pression sur les registres et ainsi d'utiliser au mieux les composants arithmétiques.

Ce coprocesseur est contrôlé, selon ses versions et la taille de la mémoire utilisée, par une trentaine de bits, notés c_0 à c_{31} sur la Figure 12.1. Nous avons choisi de laisser le choix de l'implémentation de ce contrôle à chacune des utilisations. En effet, un calcul de quelques milliers de cycles demande déjà une centaine de kilooctets de mémoire pour les bits de contrôle. Cependant, il est possible d'exploiter les régularités de chacune de nos applications pour réduire les besoins en cette mémoire d'instructions.

12.2 Calcul d'exponentiation finale

La première utilisation de notre coprocesseur arithmétique est le calcul d'exponentiation finale en association avec les architectures parallèles et *ad hoc* du chapitre précédent.

Nous avons détaillé au Chapitre 7 les algorithmes permettant le calcul de l'exponentiation finale. Ceux-ci s'expriment comme une suite d'opérations arithmétiques sur $\mathbb{F}_{p^m}$. Il a donc été possible d'implémenter directement ces algorithmes pour notre coprocesseur. Comme le calcul d'exponentiation finale est pipeliné avec celui du couplage non-réduit, nous devons calculer une exponentiation finale au moins aussi vite que le couplage non-réduit pour ne pas ralentir le calcul complet. *A contrario* nous ne gagnons rien à calculer cette exponentiation plus rapidement et nous avons tout intérêt à limiter la surface occupée par le coprocesseur. Nous avons donc choisi D, le nombre de coefficients traités en parallèle par le multiplieur, le plus petit possible permettant de satisfaire les contraintes de latence pour le calcul d'exponentiation.

12.2.1 Utilisation du Frobenius inverse

Nous présentons ici une idée qui nous a permis d'accélérer notre coprocesseur. En effet, les chemins critiques du coprocesseur sont relativement équilibrés mais le plus long passe en général par les opérateurs calculant le Frobenius. Nous avons constaté que si nous remplaçons les utilisations du Frobenius ($x \mapsto x^p$) par le Frobenius inverse ($x \mapsto \sqrt[p]{x}$), cet opérateur est en général plus rapide pour les représentations de corps.

Partant de cette constatation, nous avons réexaminé les algorithmes d'exponentiation finale pour essayer de remplacer les applications du Frobenius par des Frobenius inverses. La première utilisation du Frobenius concerne le calcul d'une inversion sur l'extension $\mathbb{F}_{p^{\frac{km}{2}}}$ et le corps de base $\mathbb{F}_{p^m}$ par deux applications imbriquées de l'algorithme d'Itoh & Tsujii. Nous reprenons ici les notations de la Section 8.1.4 où il est présenté. Nous cherchons ainsi à inverser $a \in \mathbb{F}_{q^n}$. Les utilisations du Frobenius sont toutes dans le calcul de

$$s = a^{q \cdot \frac{q^{n-1}-1}{q-1}} \text{ où } q \cdot \frac{q^{n-1}-1}{q-1} = \overline{\underbrace{1 \ldots 1}_{n-1} 0}^{(q)}.$$

Nous montrons maintenant qu'il est possible de n'utiliser que des racines q-ième pour ce calcul. Pour ce faire, nous posons

$$w'_i = a^{\frac{q^n - q^{n-i}}{q-1}} = a^{\overline{\underbrace{1 \ldots 1}_{i} \underbrace{0 \ldots 0}_{n-i}}^{(q)}}.$$

À l'instar du calcul avec les Frobenius, cette suite est munie d'une relation de récurrence :

$$w'_{i+j} = w'_i \cdot \sqrt[q^i]{w'_j}.$$

Ainsi, en remplaçant les lignes 1 à 6 de l'Algorithme 8.1 par celles qui suivent, nous obtenons un algorithme au coût identique, à ceci près que les Frobenius ont été remplacés par des Frobenius inverses :

1. $w'_1 \leftarrow \sqrt[q]{a}$ (1 Frob. inv. sur $\mathbb{F}_{q^n}$)
2. **for** $i \in 1, \ldots, l-1$ **do**
3. $\quad w'_{B_i} \leftarrow w'_{B_i - B_{i-1}} \cdot \sqrt[q^{B_i - B_{i-1}}]{w'_{B_{i-1}}}$ (($B_i - B_{i-1}$) Frob. inv., 1 mul. sur $\mathbb{F}_{q^n}$)

4. **end for**
5. $s \leftarrow w'_{n-1}$

où B_i est également une chaîne d'addition de Brauer pour $n-1$. En appliquant cet algorithme modifié aux deux inversions, nous éliminons ainsi les utilisations du Frobenius correspondantes.

La deuxième utilisation consiste en l'élévation à la puissance $p^{\frac{m+1}{2}}$ lors de la dernière phase de l'algorithme d'exponentiation finale. Or, nous pouvons constater que, pour tout $a \in \mathbb{F}_{p^m}$,

$$a^{p^{\frac{m+1}{2}}} = \sqrt[p^{\frac{m-1}{2}}]{a^{p^m}} = \sqrt[p^{\frac{m-1}{2}}]{a}$$

grâce au petit théorème de Fermat. Ainsi, chaque application du Frobenius itéré $\frac{m+1}{2}$ fois peut être remplacée par $\frac{m-1}{2}$ applications du Frobenius inverse, nous gagnons donc ici k opérations par rapport à l'algorithme utilisant les Frobenius.

Nous avons ainsi pu remplacer toutes les apparitions du Frobenius par des Frobenius inverses dans le cas de la caractéristique 3. Pour la caractéristique 2 en revanche, il reste encore quelques applications du Frobenius (Algorithme 7.4 lignes 1 et 14) qui apparaissent lors des élévations à la puissance $2^{2m}-1$ et 2^m+1 ; en effet, ce sont des carrés apportés par les multiplications dans $\mathbb{F}_{2^{4m}}$ de valeurs partageant certains coefficients. Comme le Frobenius en caractéristique 2 correspond à la mise au carré, nous transformons ainsi une opération a priori coûteuse (produit) en une opération linéaire (Frobenius). Les 6 carrés concernés devront donc être remplacés par des produits si nous ne munissons notre coprocesseur que du Frobenius inverse. C'est pourquoi nous avons choisi de ne pas appliquer cette optimisation en caractéristique 2.

Nous avons alors synthétisé notre coprocesseur pour les différents corps que nous avons utilisés avec soit le support du Frobenius, soit celui du Frobenius inverse en caractéristique 3 afin de sélectionner les cas où cette optimisation fait sens. Les résultats de placement/routage sur Virtex-4 sont présentés dans la Table 12.2. En général, le coprocesseur peut alors fonctionner à une fréquence plus élevée avec le Frobenius inverse car la profondeur des sommes de coefficients pour le calcul du Frobenius inverse est le plus souvent moins grande. Cependant, la surface augmente alors légèrement car il y a moins de sous-expressions communes dans ce calcul que dans le calcul du Frobenius. Ainsi, tout le calcul d'exponentiation finale est accéléré quand le Frobenius inverse est plus rapide. C'est pourquoi nous avons utilisé cette optimisation pour tous les corps de caractéristique 3 de notre implémentation, excepté $\mathbb{F}_{3^{239}}$.

12.3 Couplage avec des corps de degré d'extension composé

Nous présentons dans cette section une implémentation de couplage sur des courbes elliptiques supersingulières définies sur des corps de degré d'extension non premier. Cette implémentation a fait l'objet de l'article [Est10]. L'idée principale de cette contribution est de définir des courbes sur un corps $\mathbb{F}_{p^{n\cdot m}}$ avec n une petit entier et m premier. L'avantage de cette méthode est arithmétique : comme nous le montrons à la Figure 12.3, cela nous permet d'une part de n'implémenter en matériel que l'arithmétique du corps de base $\mathbb{F}_{p^m}$ et non celle du corps de définition de la courbe $\mathbb{F}_q$ obtenant ainsi un coprocesseur plus compact. D'autre part, l'arithmétique du corps de définition $\mathbb{F}_{p^{n\cdot m}}$ est alors implémenté comme un logiciel pour notre coprocesseur présenté à la Section 12.1, la souplesse de programmation ainsi gagné nous permet alors d'utiliser, entre autres, des algorithmes de multiplication sous-quadratiques. Nous présentons l'implémentation de cette arithmétique d'extension à la Section 12.3.2.

TABLE 12.2 – Calcul du Frobenius et du Frobenius inverse en caractéristique 3.

Représentation du corps	**Op.**	**Nombre d'add.**	**Profondeur**	**Surface [slices]**	**Freq. [MHz]**
$\mathbb{F}_3[x]/(x^{97}+x^{16}-1)$	$(\cdot)^3$	106	4	**4704**	185
	$\sqrt[3]{\cdot}$	96	3	4722	**192**
$\mathbb{F}_3[x]/(x^{167}-x^{71}+1)$	$(\cdot)^3$	229	5	**7607**	160
	$\sqrt[3]{\cdot}$	166	3	7682	**175**
$\mathbb{F}_3[x]/(x^{193}+x^{64}-1)$	$(\cdot)^3$	234	4	9265	**179**
	$\sqrt[3]{\cdot}$	192	3	**9092**	**179**
$\mathbb{F}_3[x]/(x^{239}-x^{5}+1)$	$(\cdot)^3$	242	4	**11589**	**179**
	$\sqrt[3]{\cdot}$	238	3	11848	177
$\mathbb{F}_3[x]/(x^{313}-x^{187}-1)$	$(\cdot)^3$	558	6	**15073**	141
	$\sqrt[3]{\cdot}$	312	3	15117	**172**

Le nombre d'additions correspond aux additions de coefficients — c'est-à-dire d'élément de $\mathbb{F}_3$ — pour calculer le Frobenius ou le Frobenius inverse. Nous donnons également la profondeur qui est le maximum d'addition à effectuer pour obtenir chaque coefficient du résultat. Les surfaces et fréquences données sont celles du coprocesseur complet.

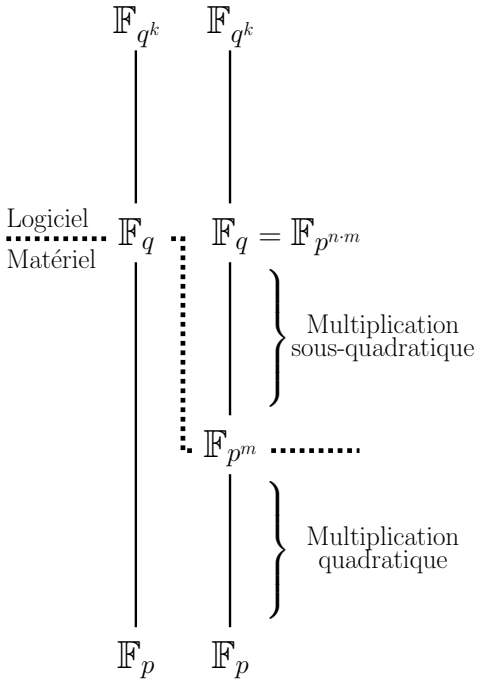


FIGURE 12.3 – Utilisation d'une courbe sur un corps de degré d'extension composé pour le calcul de couplage.

TABLE 12.4 – Courbes elliptiques supersingulières définies sur différents corps $\mathbb{F}_{p^{n \cdot m}}$ de degré d'extension composé et complexités associées aux différentes attaques.

					Coût des attaques (bits)			
	p^m	n	b	$\log_2 \ell$	Pollard's ρ	FFS	GHS	SDHP
E_2	2^{1117}	1	1	1076	531	128	–	–
	2^{367}	3	1	698	342	128	489	275
	2^{227}	5	1	733	359	129	363	189
	$\mathbf{2^{163}}$	**7**	**1**	**753**	**370**	**129**	**279**	**142**
	2^{127}	9	1	487	236	130	225	114
	2^{103}	11	1	922	454	129	187	94
	2^{89}	13	0	1044	515	164	130	82
	2^{73}	15	0	492	239	136	127	68
E_3	3^{503}	1	1	697	342	132	–	–
	$\mathbf{3^{97}}$	**5**	**−1**	**338**	**163**	**130**	**245**	**128**
	3^{67}	7	−1	612	300	129	182	92
	3^{53}	11	−1	672	330	140	152	77
	3^{43}	13	1	764	376	138	125	63

Remarquons que de telles courbes elliptiques définies sur un corps de degré d'extension composé ont été utilisées auparavant en cryptographie sous le nom de variétés de trace nulle — *Trace-Zero Varieties* (TZV) [Fre01]. En effet, la restriction de Weil d'une courbe E définie sur un corps $\mathbb{F}_{q'^n}$ est isomorphe au produit $E(\mathbb{F}_{q'}) \times B(\mathbb{F}_{q'})$ où B est une variété de trace nulle. Celle-ci peut également être représentée par le quotient $E(\mathbb{F}_{q'^n})/E(\mathbb{F}_{q'})$. Dans notre cas ($q' = p^m$) et comme nous travaillons dans un grand sous-groupe $E(\mathbb{F}_{q'^n})[\ell]$ qui ne peut être contenu dans $E(\mathbb{F}_{q'})$, la ℓ-torsion est bien isomorphe à un sous-groupe de la variété de trace nulle correspondante. Les TZV ont été étudiées dans le contexte des couplages, principalement pour réaliser de la compression de points [Ces08 ; RS02 ; RS09].

12.3.1 Choix de courbes

Se placer dans un corps de degré d'extension composé rend néamoins les courbes sujettes à des attaques supplémentaires : celles reposant sur la descente de Weil (Section 4.2.4). Le choix du corps de définition de la courbe demande une cryptanalyse attentive et nous référons le lecteur au Chapitre 4 pour une description de toutes les attaques concernées. Dans la Table 12.4, nous présentons différents choix de corps de définition pour les courbes elliptiques supersingulières E_2 et E_3 avec les complexités en bit de ces différentes attaques. Nous avons cherché ici à atteindre la sécurité standard de 128 bits. [3]

Pour constituer cette table, il est nécessaire de factoriser le cardinal de la courbe afin d'en extraire le plus grand facteur premier. Le cardinal de ces courbes supersingulières est

3. Rappelons que nos travaux sont antérieurs aux récents développements sur le logarithme discret en petite caractéristique et n'en tiennent pas compte (Chapitre 1).

particulier et connu : il s'agit des facteurs aurifeuilliens des nombres de Cunningham en base 2 et 3. Ces nombres ont beaucoup intéressé la communauté autour de la factorisation des entiers et la plupart d'entre eux ont pour cela déjà été factorisés. Nous avons donc utilisé les tables de Sam Wagstaff [Wag] et de Paul Leyland [Ley].

La Table 12.4 confirme l'intuition que plus le degré d'extension est composé, plus les attaques reposant sur la descente de Weil (GHS, SDHP) sont efficaces. Cependant, l'attaque FFS sur le groupe d'arrivée du couplage reste la complexité limitante aussi longtemps que le degré d'extension n est raisonnablement petit.

Il n'est pas nécessaire pour tous les protocoles d'assurer la résistance au problème *static Diffie-Hellman* (SDHP). Cependant, nous avons choisi pour cette implémentation de conserver 128 bits de sécurité pour ce problème. Ainsi, nous avons implémenté le couplage pour les courbes $E_2(\mathbb{F}_{2^{7\cdot 163}})$ et $E_3(\mathbb{F}_{3^{5\cdot 97}})$. La variété de trace nulle correspondant à cette dernière est de genre 4 et fournit un degré de plongement de 30, permettant ainsi d'atteindre la valeur optimale $k/g = 7,5$ donnée à la Table 4.4.

12.3.2 Arithmétique d'extension de corps

Nous décrivons ici l'implémentation de l'arithmétique de $\mathbb{F}_{p^{n\cdot m}}$ pour notre coprocesseur instancié pour le corps $\mathbb{F}_{p^m}$. Nous nous sommes intéressés ici aux deux corps utilisés pour les implémentations sur $\mathbb{F}_{2^{7\cdot 163}}$ et $\mathbb{F}_{3^{5\cdot 97}}$. Étant donné que nous disposons pour ces corps d'opérateurs compacts pour le calcul du Frobenius (élévation à la puissance p) et que nous choisissons des trinômes irréductibles pour la construction des extensions de degré n, nous avons choisi de représenter ces extensions en base polynomiale. Ainsi, nous prendrons :

$$\mathbb{F}_{2^{7\cdot m}} \cong \mathbb{F}_{2^m}[X]/(X^7 + X + 1) \text{ et}$$
$$\mathbb{F}_{3^{5\cdot m}} \cong \mathbb{F}_{3^m}[X]/(X^5 - X + 1).$$

Nous passons alors en revue l'implémentation des différentes primitives arithmétiques dont nous aurons besoin pour le calcul de couplages.

§ 1. Addition. L'addition est l'opération arithmétique la plus simple puisqu'il suffit d'additionner les deux opérandes coefficient par coefficient. Cependant, l'implémentation de cette opération sur notre coprocesseur n'est pas immédiate ; en effet, les accès mémoire sont limitants et nous ne pouvons réaliser toutes ces additions à la suite. Nous montrons à la Figure 12.5 l'ordonnancement d'une addition pour un degré d'extension n de 5. Il faut donc 9 cycles pour réaliser une addition sur une extension de degré 5 et 12 pour une extension de degré 7. Plus généralement, il faut $\frac{3n}{2} + O(1)$ cycles étant donné qu'il faut, pour chaque coefficient, lire deux valeurs et en écrire une.

§ 2. Frobenius. Nous aurons aussi besoin de pouvoir calculer l'élévation à la puissance p et ses itérées sur $\mathbb{F}_{p^{n\cdot m}}$ en n'utilisant que des additions et le Frobenius sur $\mathbb{F}_{p^m}$. Nous utilisons alors la linéarité du Frobenius pour exprimer ceci : appliquer le Frobenius itéré i fois sur un élément de l'extension correspond à l'appliquer séparément sur chacun des coefficients puis à calculer une combinaison linéaire des valeurs obtenues dépendant de la valeur de $i \bmod n$.

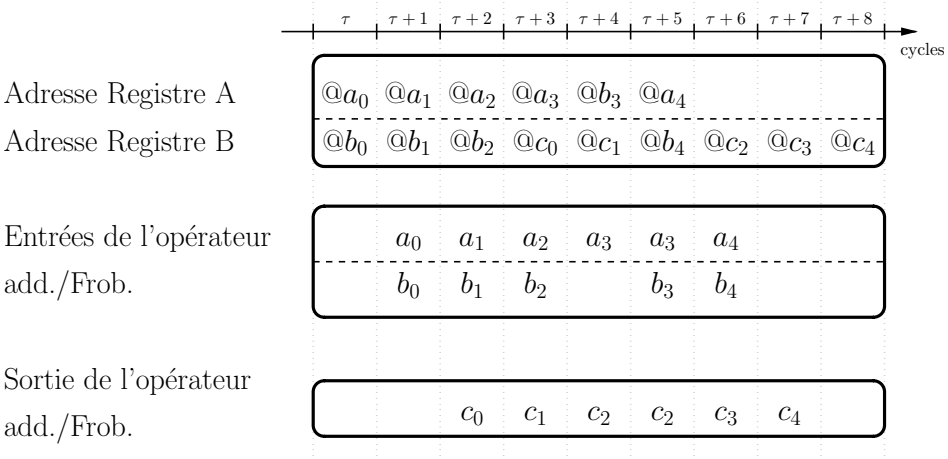


FIGURE 12.5 – Ordonnancement pour notre coprocesseur d'une addition pour un degré d'extension $n = 5$.

Nous utiliserons les itérés du Frobenius de trois manières : le Frobenius, le Frobenius itéré deux fois et, de façon moins fréquente (uniquement lors de l'exponentiation finale), de longues itérations.

§ 3. Multiplication. Nous nous intéressons maintenant à la multiplication. Pour celle-ci, nous dépendons également du corps de base $\mathbb{F}_{p^m}$; en effet, nous devons choisir D le nombre de coefficients par cycle que traitera le multiplieur parallèle-série et par conséquent le nombre de cycles que prend chaque multiplication sur $\mathbb{F}_{p^m}$. Deux éléments interviennent dans ce choix. D'une part, plus nous augmentons D, plus la fréquence du multiplieur diminue et nous cherchons à équilibrer celle-ci avec celle de l'opérateur mixte pour les additions et les Frobenius. D'autre part, m est premier et $\frac{m}{D}$ ne peut donc être entier quand $D > 1$; le dernier cycle de la multiplication ne traite dès lors que $m \bmod D$ coefficients. Nous devons alors maximiser $m \bmod D$ afin d'utiliser au mieux la surface du multiplieur.

Pour notre implémentation sur $\mathbb{F}_{2^{7\cdot 163}}$, nous avons choisi $D = 17$, ce qui permet de réaliser la multiplication sur $\mathbb{F}_{2^{163}}$ en 10 cycles. Nous devons alors choisir un algorithme de multiplication pour l'extension de degré 7 ; nous présentons les coûts de différents algorithmes dans la Table 12.6. Dans cette table, nous indiquons aussi le ratio du nombre d'additions sur le nombre de multiplications car nous chercherons à exploiter au maximum le parallélisme de notre coprocesseur en recouvrant toutes les additions par des multiplications. Ce ratio doit ainsi être bien plus petit que les 10 cycles nécessaires à un produit afin de ne pas être limité non plus par la pression sur le banc de registres. Nous avons alors choisi d'utiliser la formule de Montgomery (Algorithme 12.7, [Mon05]). L'implémentation de cette formule sur notre coprocesseur demande 228 cycles.

Pour notre seconde implémentation sur $\mathbb{F}_{3^{5\cdot 97}}$, nous avons choisi $D = 14$, ce qui permet d'effectuer un produit sur $\mathbb{F}_{3^{97}}$ en 7 cycles. Nous avons de même rassemblé différents choix d'algorithmes pour la multiplication dans l'extension de degré 5 (Table 12.8). Nous avons commencé par utiliser un algorithme reposant sur le théorème des restes chinois (Algorithme 8.4) que nous avions implémenté en 89 cycles. Grâce à la découverte et au développement de notre algorithme pour la recherche de formules de multiplication (Chapitre 9), nous avons trouvé

Algorithme 12.7 Algorithme de multiplication sur $\mathbb{F}_{2^{7\cdot m}} \cong \mathbb{F}_{2^m}[X]$ avec $X^7 + X + 1 = 0$ dérivé de [Mon05, Équation (9)]. (22 mul., 84 add.)

Entrées : $A = a_0 + a_1X + a_2X^2 + a_3X^3 + a_4X^4 + a_5X^5 + a_6X^6$ et $B = b_0 + b_1X + b_2X^2 + b_3X^3 + b_4X^4 + b_5X^5 + b_6X^6$

Sortie : $C = A \cdot B$

1. $a_7 \leftarrow a_0 + a_4$; $b_7 \leftarrow b_0 + b_4$
2. $a_8 \leftarrow a_3 + a_5$; $b_8 \leftarrow b_3 + b_5$
3. $a_9 \leftarrow a_2 + a_6$; $b_9 \leftarrow b_2 + b_6$
4. $a_{10} \leftarrow a_1 + a_7$; $b_{10} \leftarrow b_1 + b_7$
5. $a_{11} \leftarrow a_8 + a_9$; $b_{11} \leftarrow b_8 + b_9$
6. $p_0 \leftarrow a_6 \cdot b_6$; $p_1 \leftarrow a_5 \cdot b_5$; $p_2 \leftarrow a_4 \cdot b_4$
7. $p_3 \leftarrow a_3 \cdot b_3$; $p_4 \leftarrow a_8 \cdot b_8$; $p_5 \leftarrow a_2 \cdot b_2$
8. $p_6 \leftarrow a_9 \cdot b_9$; $p_7 \leftarrow a_1 \cdot b_1$; $p_8 \leftarrow a_0 \cdot b_0$
9. $p_9 \leftarrow a_7 \cdot b_7$
10. $p_{10} \leftarrow (a_5 + a_6) \cdot (b_5 + b_6)$; $p_{11} \leftarrow (a_4 + a_6) \cdot (b_4 + b_6)$
11. $p_{12} \leftarrow (a_1 + a_3) \cdot (b_1 + b_3)$; $p_{13} \leftarrow (a_1 + a_{11}) \cdot (b_1 + b_{11})$
12. $p_{14} \leftarrow (a_0 + a_2) \cdot (b_0 + b_2)$; $p_{15} \leftarrow (a_0 + a_{11}) \cdot (b_0 + b_{11})$
13. $p_{16} \leftarrow (a_0 + a_1) \cdot (b_0 + b_1)$; $p_{17} \leftarrow (a_8 + a_{10}) \cdot (b_8 + b_{10})$
14. $p_{18} \leftarrow (a_{10} + a_{11}) \cdot (b_{10} + b_{11})$
15. $p_{19} \leftarrow (a_3 + a_6 + a_{10}) \cdot (b_3 + b_6 + b_{10})$
16. $p_{20} \leftarrow (a_3 + a_7 + a_9) \cdot (b_3 + b_7 + b_9)$
17. $p_{21} \leftarrow (a_1 + a_2 + a_4 + a_5) \cdot (b_1 + b_2 + b_4 + b_5)$
18. $t_0 \leftarrow p_{11} + p_{13}$; $t_1 \leftarrow p_5 + p_{10}$; $t_2 \leftarrow p_9 + t_0$
19. $t_3 \leftarrow p_0 + p_1$; $t_4 \leftarrow p_4 + p_{15}$; $t_5 \leftarrow p_6 + p_{11}$
20. $t_6 \leftarrow p_2 + p_{21}$; $t_7 \leftarrow p_3 + p_9$; $t_8 \leftarrow p_{13} + p_{20}$
21. $t_9 \leftarrow p_{16} + t_3$; $t_{10} \leftarrow p_{10} + t_7$; $t_{11} \leftarrow p_8 + p_{19}$
22. $t_{12} \leftarrow p_{15} + t_1$; $t_{13} \leftarrow t_4 + t_6$; $t_{14} \leftarrow t_5 + t_{11}$
23. $c_0 \leftarrow p_3 + p_{21} + p_{18} + t_2 + t_9 + t_{12}$
24. $c_1 \leftarrow p_4 + p_7 + p_{12} + p_{15} + t_2 + t_5 + t_6$
25. $c_2 \leftarrow p_{13} + p_{20} + t_9 + t_{10} + t_{14}$
26. $c_3 \leftarrow p_{19} + t_3 + t_4 + t_8 + t_{12}$
27. $c_4 \leftarrow t_1 + t_2 + t_{10}$
28. $c_5 \leftarrow p_1 + t_7 + t_{13} + t_{14}$
29. $c_6 \leftarrow t_0 + t_1 + t_8 + t_{13}$
30. **return** $c_0 + c_1X + c_2X^2 + c_3X^3 + c_4X^4 + c_5X^5 + c_6X^6$

TABLE 12.6 – Algorithmes de multiplication sur $\mathbb{F}_{2^{7\cdot 163}}$.

Algorithme	×	+	Ratio
Naïf	49	48	0.98
Karatsuba $7 \xrightarrow{2} 4$ (avec l'astuce de Montgomery)	40	52	1.30
Karatsuba $7 \xrightarrow{2} 4 \xrightarrow{2} 2 \xrightarrow{2} 1$	25	51	2.04
Karatsuba $7 \xrightarrow{2} 4 \xrightarrow{2} 2 \xrightarrow{2} 1$ (avec l'astuce de Montgomery)	23	76	3.30
Formule de Montgomery (Algorithme 12.7)	**22**	**84**	**3.82**
Reposant sur le CRT (Section 8.2.4)	22	88	4.05

TABLE 12.8 – Algorithmes de multiplication sur $\mathbb{F}_{3^{5\cdot 97}}$.

Algorithme	×	+	Ratio
Naïf	25	24	0.96
Karatsuba $5 \xrightarrow{2} 3$ (avec l'astuce de Montgomery)	21	29	1.38
Karatsuba $5 \xrightarrow{2} 3 \xrightarrow{3} 1$	15	39	2.60
Karatsuba $5 \xrightarrow{2} 3 \xrightarrow{3} 1$ (avec l'astuce de Montgomery)	14	43	3.07
Formule de Montgomery (Section 8.2.5, [Mon05])	13	54	4.15
Reposant sur le CRT (Section 8.2.4, Algorithme 8.4)	12	53	4.42
Recherche exhaustive (Chapitre 9, Algorithme 9.8)	**11**	**47**	**4.27**

un algorithme optimal en nombre de produits décrit à l'Algorithme 9.8. Nous avons alors pu implémenter celui-ci en 82 cycles ce qui représente une amélioration du temps de calcul de 8%.

§ 4. Inverse. Comme notre processeur ne dispose pas d'opérateur spécifique à l'inversion sur le corps de base $\mathbb{F}_{p^m}$, nous appliquons deux fois l'algorithme d'Itoh & Tsujii pour ramener le calcul d'inverse dans $\mathbb{F}_{p^{n\cdot m}}$ à un inverse dans $\mathbb{F}_{p^m}$, puis une seconde fois pour calculer cet inverse dans $\mathbb{F}_{p^m}$. Ces algorithmes sont donnés pour $\mathbb{F}_{3^{5\cdot 97}}$ à l'Exemple 8.1 et pour $\mathbb{F}_{2^{7\cdot 163}}$ à l'Algorithme 12.9.

Nous avons rassemblé dans la Table 12.10 les coûts de ces différentes opérations arithmétiques en nombre d'opérations sur le corps de base $\mathbb{F}_{p^m}$ pour les extensions $\mathbb{F}_{2^{7\cdot 163}}$ et $\mathbb{F}_{3^{5\cdot 97}}$.

TABLE 12.10 – Coût des opérations arithmétiques dans les extensions.

(a) pour l'extension $\mathbb{F}_{2^{7 \cdot 163}}$

	$\times$	$+$	$(.)^2$
Addition	–	7	–
Multiplication	22	84	–
Frobenius itéré, $(.)^{2^i}$ où $i \equiv 0 \pmod 7$	–	–	$7i$
où $i \equiv 1 \pmod 7$	–	3	$7i$
où $i \equiv 2 \pmod 7$	–	6	$7i$
où $i \equiv 3 \pmod 7$	–	9	$7i$
où $i \equiv 4 \pmod 7$	–	9	$7i$
où $i \equiv 5 \pmod 7$	–	6	$7i$
où $i \equiv 6 \pmod 7$	–	3	$7i$
Inverse	89	276	162 –

(b) pour l'extension $\mathbb{F}_{3^{5 \cdot 97}}$

	$\times$	$+$	$(.)^3$
Addition	–	5	–
Multiplication	11	47	–
Frobenius itéré, $(.)^{3^i}$ où $i \equiv 0 \pmod 5$	–	–	$5i$
où $i \equiv 1 \pmod 5$	–	5	$5i$
où $i \equiv 2 \pmod 5$	–	6	$5i$
où $i \equiv 3 \pmod 5$	–	8	$5i$
où $i \equiv 4 \pmod 5$	–	7	$5i$
Inverse	41	117	96

Algorithme 12.9 Algorithme d'inversion sur $\mathbb{F}_{2^{7\cdot 163}}$.

Entrées : $A = a_0 + a_1X + a_2X^2 + a_3X^3 + a_4X^4 + a_5X^5 + a_6X^6 \in \mathbb{F}_{2^{7\cdot 163}} \cong \mathbb{F}_{2^{163}}[X]$
Sortie : A^{-1}

1. $W_1 \leftarrow A$
2. $W_2 \leftarrow W_1 \cdot W_1^{2^{163}}$ $(22m, 87a)$
3. $W_4 \leftarrow W_2 \cdot W_2^{2^{2\cdot 163}}$ $(22m, 90a)$
4. $W_6 \leftarrow W_2 \cdot W_4^{2^{2\cdot 163}}$ $(22m, 90a)$
5. $s_0 + s_1X + s_2X^2 + s_3X^3 + s_4X^4 + s_5X^5 + s_6X^6 \leftarrow W_6^{2^{163}}$ $(3a)$
6. $t \leftarrow s_0 \cdot a_0 + s_1 \cdot a_6 + s_2 \cdot a_5 + s_3 \cdot a_4 + s_4 \cdot a_3 + s_5 \cdot a_2 + s_6 \cdot a_1$ $(7m, 6a)$
7. $w_1 \leftarrow t$
8. $w_2 \leftarrow w_1 \cdot w_1^2$ $(1m, 1f)$
9. $w_4 \leftarrow w_2 \cdot w_2^{2^2}$ $(1m, 2f)$
10. $w_8 \leftarrow w_4 \cdot w_4^{2^4}$ $(1m, 4f)$
11. $w_{16} \leftarrow w_8 \cdot w_8^{2^8}$ $(1m, 8f)$
12. $w_{32} \leftarrow w_{16} \cdot w_{16}^{2^{16}}$ $(1m, 16f)$
13. $w_{64} \leftarrow w_{32} \cdot w_{32}^{2^{32}}$ $(1m, 32f)$
14. $w_{128} \leftarrow w_{64} \cdot w_{64}^{2^{64}}$ $(1m, 64f)$
15. $w_{160} \leftarrow w_{32} \cdot w_{128}^{2^{32}}$ $(1m, 32f)$
16. $w_{162} \leftarrow w_2 \cdot w_{160}^{2^2}$ $(1m, 2f)$
17. $u \leftarrow w_{162}^2$ $(1f)$
18. **return** $s_0 \cdot u + s_1 \cdot uX + s_2 \cdot uX^2 + s_3 \cdot uX^3 + s_4 \cdot uX^4 + s_5 \cdot uX^5 + s_6 \cdot uX^6$ $(7m)$

12.3.3 Calcul du couplage

Comme notre coprocesseur permet la souplesse de la programmation logicielle, nous n'avions pas besoin pour cette implémentation de choisir des algorithmes pour le couplage η_T présentant un corps de boucle compact comme dans le chapitre précédent. Cependant, notre coprocesseur n'offre pas la possibilité d'avoir à la fois le Frobenius et le Frobenius inverse. Nous avons donc choisi les variantes des algorithmes sans Frobenius inverse et demandant le moins d'opérations arithmétiques, c'est-à-dire l'Algorithme 7.3 (boucle renversée sans racine carrée) pour la caractéristique 2 et l'Algorithme 7.10 (boucle déroulée) pour la caractéristique 3. L'étape suivante est d'exprimer ces algorithmes comme une suite d'opérations exécutables sur notre coprocesseur.

Le couplage η_T sur $E_2(\mathbb{F}_{2^{7\cdot 163}})$ requiert ainsi : 88 509 multiplications, 448 361 additions et 52 782 applications du Frobenius π_2 sur le corps de base $\mathbb{F}_{2^{163}}$. En ce qui concerne la courbe $E_3(\mathbb{F}_{3^{5\cdot 97}})$, il faudra 34 185 multiplications, 234 690 additions et 21 099 applications du Frobenius π_3 sur $\mathbb{F}_{3^{97}}$ pour le calcul du couplage η_T.

12.3.4 Contrôle du coprocesseur

En considérant seulement les multiplications sur le corps de base et leur durée en cycles, le calcul du couplage sur $E_2(\mathbb{F}_{2^{7\cdot 163}})$ demandera au moins 880 000 cycles, et celui sur $E_3(\mathbb{F}_{3^{5\cdot 97}})$ en demandera au moins 280 000. Notre coprocesseur pour l'arithmétique dans $\mathbb{F}_{2^{163}}$ est contrôlé par 29 bits et celui sur $\mathbb{F}_{3^{97}}$ a besoin de 36 bits de contrôle. Ainsi, il faudra un minimum de 3.2 Mo pour décrire les instructions du calcul du couplage sur $E_2(\mathbb{F}_{2^{7\cdot 163}})$ et un minimum de 1 Mo pour donner celles du couplage sur $E_3(\mathbb{F}_{3^{5\cdot 97}})$. Nous ne pouvons donc pas stocker

TABLE 12.11 – Résultats de placement/routage pour le coprocesseur sur les deux corps utilisés dans l'implémentation de [Est10].

	FPGA	**Surface [slices]**	**Fréquence [MHz]**	**Temps de calcul [μs]**
$E_2(\mathbb{F}_{2^{7\cdot 163}})$ $(D=17)$	XC3S1000−5	2348	133	8580
	XC4VLX15−11	2313	254	4470
	XC6VLX75T−3	835	308	3730
$E_3(\mathbb{F}_{3^{5\cdot 97}})$ $(D=14)$	XC3S1000−5	4713	104	3910
	XC4VLX25−11	4755	192	2120
	XC6VLX75T−3	1848	250	1630

les instructions directement dans une mémoire : cela demanderait plus de ressources que le coprocesseur lui-même.

Afin de réduire les besoins en mémoire d'instructions, nous avons implémenté deux niveaux de code : le microcode et le macrocode. Dans le niveau inférieur, le microcode, nous avons implémenté l'arithmétique de l'extension $\mathbb{F}_{p^{n\cdot m}}$. Le macrocode consiste alors en une suite d'appels aux différentes opérations stockées dans le microcode. Afin de réduire encore les besoins en mémoire d'instructions, nous avons également implémenté un mécanisme de boucle pour factoriser le code des itérations de Miller. Nous avons ainsi réduit la description du couplage η_T sur $E_2(\mathbb{F}_{2^{7\cdot 163}})$ à 193 macro-opérations et à 464 macro-opérations pour celle du couplage en caractéristique 3. *In fine*, il nous faudra 1 147 131 cycles pour calculer le couplage en caractéristique 2 et 407 112 cycles pour celui en caractéristique 3. Notons que, bien que l'ajout d'un tel mécanisme de micro/macrocode ait induit une perte significative de parallélisme, il a permis de réduire les besoins en mémoire d'instructions.

Nous avons alors synthétisé nos deux architectures pour différents FPGA : les Virtex-4 LX et Virtex-6 LXT, afin de pouvoir nous comparer au reste de la littérature, et le Spartan-3, qui est plus adapté aux systèmes embarqués. Cette implémentation fournit une architecture très compacte comme le montrent les surfaces données dans la Table 12.11.

12.4 Calcul de couplage en genre 2

Nous détaillons dans cette section l'implémentation du couplage Eta optimal pour la courbe hyperelliptique supersingulière H_2 (Section 4.3.1 § 3). Afin qu'elle assure une sécurité de 128 bits, nous choisissons de l'utiliser sur le corps de base $\mathbb{F}_{2^{367}}$. Nous avons déjà décrit les algorithmes permettant de calculer un tel couplage à la Section 7.3. Cette implémentation a été publiée dans [Ara+12].

À la différence de l'implémentation précédente (couplage avec des corps de degré d'extension composé), le corps de base que nous manipulons est de plus grande taille. Cependant, le fait d'être sur une courbe de genre 2 limite la taille du corps de base. Ainsi, l'architecture du coprocesseur de corps fini que nous proposons dans ce chapitre reste adaptée. Le multiplieur a une architecture parallèle-série qui est paramétrée par le nombre de coefficients du second opérande traités à chaque cycle. Afin de conserver une fréquence de fonctionnement élevée

TABLE 12.12 – Résultats de placement/routage pour le coprocesseur sur $\mathbb{F}_{2^{367}}$ avec $D = 16$.

FPGA	**Surface [slices]**	**Fréquence [MHz]**	**Temps de calcul [μs]**
XC2VP30−6	4646	176	4400
XC3S1500−5	4713	114	6800
XC4VLX25−11	4518	220	3520
XC6VLX75T−3	1366	241	3200

pour notre coprocesseur, nous gardons ce paramètre à une valeur similaire au prix d'une plus grande latence pour le coprocesseur. *In fine*, le choix de $D = 16$ semble être optimal pour le corps $\mathbb{F}_{2^{367}}$.

Le coprocesseur a également la possibilité de calculer le Frobenius π_2 et son i-ème itéré π_{2^i}. Comme l'algorithme de calcul du couplage, en particulier le calcul d'exponentiation finale, comprend de longues chaînes de Frobenius, nous avons intérêt à choisir i le plus grand possible tel que le chemin critique du coprocesseur ne soit pas influencé (sans perte de fréquence). Le corps de base que nous avons choisi peut-être défini par le trinôme irréductible $X^{367} + X^{21} + 1$. Ce polynôme permet de définir le Frobenius et ses itérés par des applications linéaires des coefficients suffisamment compactes pour que nous puissions choisir $i = 3$.

Le coprocesseur pour corps fini ainsi instancié a été synthétisé pour quatre familles de FPGA Xilinx : Virtex-2 Pro, Virtex-4 LX, Virtex-6 LXT et la famille de FPGA à faible coût Spartan-3. Nous donnons dans la Table 12.12 les résultats de placement-routage de ces architectures.

Il nous reste alors à implémenter sur notre coprocesseur ainsi construit les formules d'octuplement, les fonctions de Miller associées, l'arithmétique de l'extension $\mathbb{F}_{2^{12\cdot 367}}$ et l'exponentiation finale : c'est-à-dire produire la séquence de bits de contrôle du coprocesseur qui réaliseront ces algorithmes. Comme c'est un travail fastidieux et que les formules des fonctions de Miller changent selon que nous choisissions zéro, un ou deux diviseurs dégénérés en entrée, nous avons choisi d'implémenter le cas le plus général avec deux diviseurs génériques en entrée.

Une fois encore, il n'est pas possible d'implémenter le contrôle du coprocesseur par le biais d'une mémoire d'instructions contenant l'intégralité des bits de contrôle pour chaque cycle : cela demanderait une mémoire trop importante. Nous avons donc également cherché à factoriser le code du calcul de couplage. Cependant, nous n'avons pas ici la même structure d'extension et nous ne pouvons pas reposer sur un système de micro/macrocode. Nous avons donc implémenté un mécanisme plus évolué d'appels de fonctions. Grâce à ce mécanisme, il a suffit de 89 kbit de mémoire d'instructions.

Le coprocesseur demande 773 685 cycles pour exécuter le calcul du couplage Eta optimal sur $H_2(\mathbb{F}_{2^{367}})$, ce qui nous permet de calculer le couplage en 3.55 ms sur Virtex-4 LX (Table 12.12).

Chapitre 13

Synthèse des résultats en matériel et comparaison

Nous venons de décrire, dans les deux chapitres précédents, nos différentes implémentations : accélérateur *ad-hoc* parallèle (Chapitre 11), accélérateur compact avec des corps de degré d'extension composé (Section 12.3) et avec une courbe de genre 2 (Section 12.4). Nous cherchons maintenant à regrouper tous nos résultats d'implémentation matérielle et à comparer ces architectures à celles disponibles dans la littérature.

Nous commençons cette comparaison en nous concentrant sur les implémentations pour FPGA. Celle-ci se révèle délicate pour plusieurs raisons. Premièrement, des implémentations existent pour plusieurs niveaux de sécurité : jusqu'en 2010, il n'y avait pas d'implémentation permettant d'atteindre la sécurité standard de 128 bits[1]. Une fois ce niveau atteint, peu d'implémentations ont exploré les sécurités supérieures. De plus, les plateformes FPGA disponibles sont variées et évoluent avec le temps, elles diffèrent ainsi par leur taille, leur fréquence maximale de fonctionnement (*speedgrade*), la quantité de RAM disponible, etc. comme nous avons pu le voir à la Section 10.2.3. Enfin, chaque architecture de la littérature a pu être dessinée dans différentes optiques : certaines sont optimisées pour la latence, pour la surface (systèmes embarqués) ou bien encore en débit par unité de surface occupée sur le FPGA.

Nous avons séparé les implémentations FPGA de couplages en deux catégories : les implémentations atteignant la sécurité standard de 128 bits et celles, plus anciennes, se limitant aux niveaux de sécurité plus faibles.

Implémentations FPGA en deçà de 128 bits de sécurité

Nous commençons par nous intéresser aux implémentations les plus anciennes. Nous donnons dans la Table 13.1 une liste des implémentations FPGA de couplages dont la sécurité est comprise entre 75 et 110 bits. À l'exception de celles de [Ron+07] et [Bar+08], ces implémentations utilisent toutes des courbes elliptiques supersingulières en caractéristique 2 ou 3. En effet, leur degré de plongement est adapté à ces niveaux de sécurité ainsi que la Table 4.3 le montre.

1. Nous tenons à rappeler que les estimations de sécurité dans ce document ne tiennent pas compte des développements récents sur le logarithme discret en petite caractéristique (*cf.* Chapitre 1).

TABLE 13.1 – Accélérateurs matériels pour le calcul de couplages pour des sécurités allant de 75 à 110 bits.

	Courbe	Securité [bits]	FPGA	Surface [slices]	Fréquence [MHz]	Temps de calcul [μs]	AT [s·slices]
[Ron+07]	$H_2(DD)/\mathbb{F}_{2^{103}}$	75	XC2VP100−6	30464	41	132	4.02
[Bar+08]	$E_p/\mathbb{F}_{p512}$	87	XC2V8000−5	33857	135	1610	54.5
[Beu+08a]	$E_2/\mathbb{F}_{2^{459}}$	89	XC2VP20−6	8153	115	327	2.67
	$E_3/\mathbb{F}_{3^{193}}$	89	XC2VP20−6	8266	90	298	2.46
[SKG09]	$E_2/\mathbb{F}_{2^{457}}$	88	XC4VLX200−10	58956	100	101	5.94
	$E_2/\mathbb{F}_{2^{557}}$	96	XC4VLX200−10	37931	66	676	25.6
Chapitre 11 [Beu+11]	$E_3/\mathbb{F}_{3^{167}}$	**83**	**XC2VP100−6**	**40765**	**118**	**12.3**	**0.501**
			XC4VLX100−11	**40974**	**175**	**8.24**	**0.338**
	$E_2/\mathbb{F}_{2^{457}}$	**88**	**XC2VP100−6**	**42965**	**147**	**11.0**	**0.473**
			XC4VLX100−11	**44223**	**215**	**7.52**	**0.333**
	$E_3/\mathbb{F}_{3^{193}}$	**89**	**XC2VP100−6**	**46135**	**130**	**12.8**	**0.591**
			XC4VLX200−11	**47260**	**179**	**9.33**	**0.441**
	$E_2/\mathbb{F}_{2^{557}}$	**96**	**XC4VLX200−11**	**55156**	**149**	**13.2**	**0.728**
	$E_3/\mathbb{F}_{3^{239}}$	**97**	**XC4VLX200−11**	**66631**	**179**	**11.5**	**0.766**
	$E_2/\mathbb{F}_{2^{613}}$	**100**	**XC4VLX200−11**	**62418**	**143**	**15.1**	**0.943**
	$E_2/\mathbb{F}_{2^{691}}$	**105**	**XC4VLX200−11**	**78874**	**130**	**18.8**	**1.48**
	$E_3/\mathbb{F}_{3^{313}}$	**109**	**XC4VLX200−11**	**97105**	**159**	**16.9**	**1.64**

À l'exception de l'opérateur compact unifié proposé dans [Beu+08a], la plupart des implémentations à faible sécurité mettent principalement l'accent sur la performance : elles ont pour but de minimiser le temps de calcul au prix d'une importante surface occupée sur le FPGA. En effet, ces implémentations utilisent des FPGA haut de gamme dotés de surfaces importantes : à titre de comparaison, un FPGA Xilinx Virtex-4 LX 100 comporte un peu moins de 50 000 slices et coûte quelques milliers de dollars. Cette stratégie, privilégiant la performance sur la compacité des accélérateurs pour le calcul du couplage, explique ainsi le fait que ces implémentations n'atteignent pas la sécurité standard de 128 bits. Par le fait que les FPGA offrent une surface limitée, il n'est pas possible de choisir des corps de plus grande taille pour augmenter la sécurité.

Notre accélérateur parallèle *ad-hoc* présenté au Chapitre 11 montre un saut de performance important par rapport aux architectures précédentes. Nous expliquons principalement ceci par l'utilisation d'algorithmes rapides pour la multiplication à la place d'algorithmes quadratiques. Comme nous avons utilisé ces algorithmes dans un multiplieur parallèle, la surface demandée par l'accélérateur est importante mais l'utilisation de formules rapides le compense largement et permet d'améliorer le compromis temps/surface.

Remarquons que, bien qu'elles aient des degrés de plongement différents, les familles de courbes E_2 et E_3 fournissent des performances très comparables. Nous l'expliquons en partie par le fait que le degré de plongement meilleur en caractéristique 3 compense l'arithmétique plus coûteuse.

Implémentations FPGA à 128 bits de sécurité

Nous commentons maintenant les implémentations FPGA de couplages atteignant 128 bits de sécurité, présentées dans la Table 13.2. Pour atteindre ce niveau de sécurité, différents choix de courbes ont été proposés : courbes Barreto-Naehrig, courbes elliptiques supersingulières, ou bien encore courbes hyperelliptiques de genre 2 supersingulières. Les courbes supersingulières ont l'avantage de fournir des couplages symétriques et de demander une arithmétique de corps de petite caractéristique, *a priori* plus adaptée aux FPGA. Cependant, elles ne fournissent qu'un degré de plongement limité ($k/g \leq 6$) qui n'est pas adapté à ce niveau de sécurité. C'est pourquoi une partie des implémentations récentes se sont concentrées sur l'usage des courbes Barreto-Naehrig qui, elles, donnent $k = 12$.

Par conséquent, la comparaison de ces implémentations en est d'autant plus délicate que certaines implémentations reposent sur une arithmétique modulaire sur $\mathbb{F}_p$ avec p d'environ 256 bits. Ce faisant, celles-ci manipulent des entiers de taille moyenne et peuvent utiliser avantageusement les blocs DSP fournis par les FPGA. Ces blocs DSP permettent notamment d'effectuer des multiplications d'entiers de taille 18 bits × 18 bits [2]. Les unités DSP et les slices des FPGA ne sont pas commensurables ; nous ne pouvons donc comparer ces deux types d'implémentation de façon équitable. Les DSP n'apparaissent notamment pas dans le compromis temps-surface.

Nos implémentations (Section 12.3 [Est10] et Section 12.4 [Ara+12]) ont pour but de fournir des implémentations très compactes de couplages à 128 bits de sécurité. Nous pouvons constater dans la Table 13.2 que cet objectif est atteint puisque nous proposons les architectures aux surfaces les plus faibles. Ces architectures offrent également un compromis temps/surface très compétitif. Elles n'ont été battues que par les meilleures implémen-

2. 25 bits × 18 bits pour les Virtex 6.

TABLE 13.2 – Accélérateurs matériels pour le calcul de couplages à 128 bits de sécurité.

Courbe	Securité [bits]	FPGA	Surface [slices]	Fréquence [MHz]	Temps de calcul [μs]	AT [s·slices]
[GMR10]	$E_{BN}/\mathbb{F}_{p256}$	128	XC4VLX200−12	52000	50	16400	853
Section 12.3 [Est10]	$E_3/\mathbb{F}_{3^{97\cdot5}}$	130	**XC3S1000−5**	**4713**	**104**	**3910**	**18.4**
			XC4VLX25−11	**4755**	**192**	**2120**	**10.1**
			XC6VLX75T−3	**1848**	**250**	**1630**	**3.01**
	$E_2/\mathbb{F}_{2^{163\cdot7}}$	129	**XC3S1000−5**	**2348**	**133**	**8580**	**20.1**
			XC4VLX15−11	**2313**	**254**	**4470**	**10.3**
			XC6VLX75T−3	**835**	**308**	**3730**	**3.11**
[Che+11]	$E_{BN}/\mathbb{F}_{p254}$	126	XC6VLX240T−2	7032 + 32 DSP	250	573	4.03
[GRD11]	$E_2/\mathbb{F}_{2^{1223}}$	128	XC2VP ?- ?	36534	125	381	13.9
			XC4VLX200−11	35458	168	286	10.1
			XC6VLX130T−3	15167	250	190	2.88
Section 12.4 [Ara+12]	$H_2(GG)/\mathbb{F}_{2^{367}}$	128	**XC2VP30−6**	**4646**	**176**	**4400**	**20.5**
			XC3S1500−5	**4713**	**114**	**6800**	**32.0**
			XC4VLX25−11	**4518**	**220**	**3520**	**15.9**
			XC6VLX75T−3	**1366**	**241**	**3200**	**4.38**
[AHN13]	$E_2/\mathbb{F}_{2^{1223}}$	128	XC6VLX365T−3	13596	271	148	2.01
			XC6VLX365T−3	16403	267	102	1.67
[FVV12]	$E_{BN}/\mathbb{F}_{p256}$	128	XC6VLX240T−3	4014 + 42 DSP	210	1170	4.70
[GVR13]	$E_{BN}/\mathbb{F}_{p254}$	128	XC6VLX240T−3	5163 + 144 DSP	166	375	1.94
[Yao+13]	$E_{BN}/\mathbb{F}_{p254}$	128	XC6VLX240T−1	5237 + 64 DSP	230	358	1.87

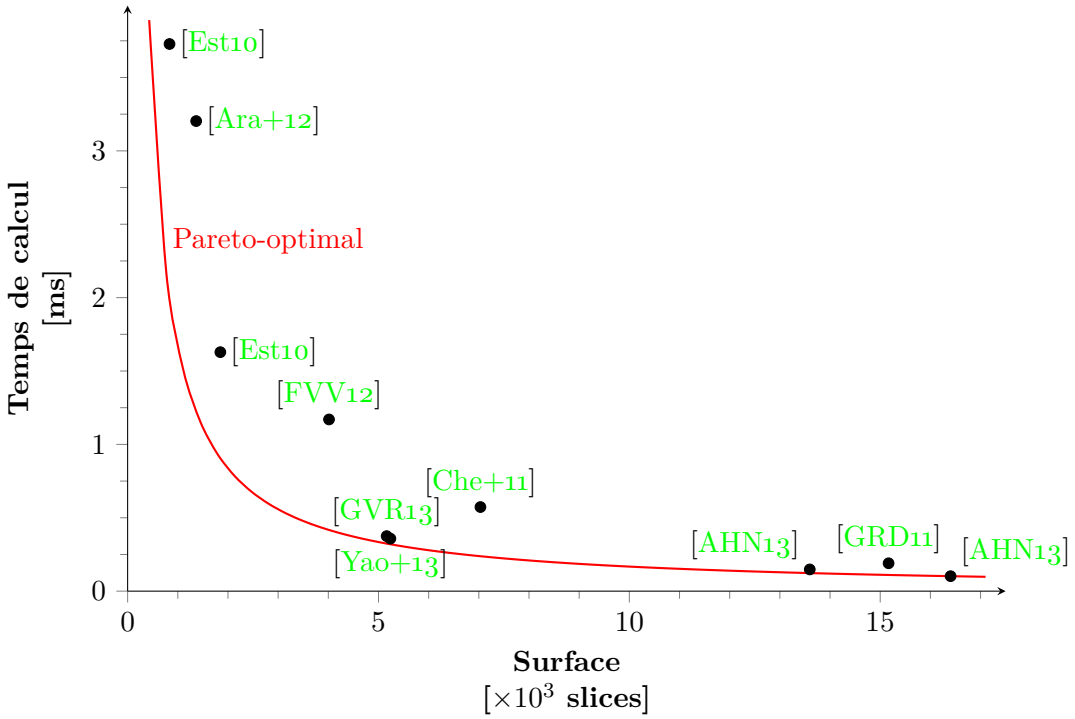


FIGURE 13.3 – Comparaison en temps-surface pour les accélérateurs de couplage sur Virtex-6.

tations pour courbes Barreto-Naehrig [GVR13 ; Yao+13] en omettant de compter le coût des unités DSP, ainsi que des travaux plus récents [GRD11 ; AHN13]. Ces travaux ne fournissent cependant pas des accélérateurs compacts. Remarquons tout de même l'implémentation dans [AHN13] qui améliore significativement les performances obtenues jusqu'alors grâce à un multiplieur reposant sur le produit par une matrice de Toeplitz en exploitant au mieux un compromis surface/parallélisation.

Tout comme pour les sécurités inférieures, les résultats ne montrent pas de différence significative entre les courbes elliptiques supersingulières de caractéristique 2 et 3.

Enfin, en ce qui concerne le genre 2 (*cf.* Section 12.4 et [Ara+12]), nous pouvons remarquer que le travail effectué sur l'algorithmique et l'arithmétique de cette familles de courbes nous a permis de ramener les performances des accélérateurs correspondants au même ordre de grandeur que celui pour les courbes elliptiques. Si les performances présentées ici concernent uniquement le cas où les deux arguments du couplage sont des diviseurs génériques, rappelons toutefois que la plupart des protocoles tolèrent qu'un de leurs arguments soit toujours dégénéré. Dans une telle situation, le temps de calcul sera divisé par 2 environ, donnant ainsi des performances équivalentes à celles de la Section 12.3 [Est10].

En outre, nous présentons dans la Figure 13.3, un graphique permettant d'apprécier la variété des implémentations présentées dans la Table 13.2. Nous avons mis la surface en abscisse et le temps de calcul en ordonnée. Chaque architecture est alors représentée par un point sur ce graphique et correspond à une ligne de la Table 13.2. Nous n'y avons mis que les accélérateurs pour les FPGA Virtex 6 afin d'avoir une comparaison juste. Nous voyons ainsi se détacher les accélérateurs compacts des accélérateurs minimisant le temps de calcul. Nous avons également tracé l'hyperbole qui correspond à l'optimal en termes de compromis

TABLE 13.4 – Accélérateurs matériels (ASIC) pour le calcul de couplages à 128 bits de sécurité.

	Courbe	ASIC	Surface [kGates]	Fréquence [MHz]	Temps de calcul [μs]	AT [s·Gates]
[FVV09]	$E_{BN}/\mathbb{F}_{p256}$	130 nm	183	204	2910	533
[Kam+09]	$E_{BN}/\mathbb{F}_{p256}$	130 nm	97	338	15800	1530
[AHN13]	$E_2/\mathbb{F}_{2^{1223}}$	65 nm	473	500	80.6	38.1
		65 nm	524	500	54.6	28.6

TABLE 13.5 – Implémentations logiciels de couplages à 128 bits de sécurité.

	Courbe	Core 2 [$\times 10^6$ cycles]	Nehalem [$\times 10^6$ cycles]
[Beu+09b]	$E_2/\mathbb{F}_{2^{1223}}$	23.0	–
	$E_3/\mathbb{F}_{3^{359}}$	15.1	–
[ALH10]	$E_2/\mathbb{F}_{2^{1223}}$	18.8	8.28
[NNS10]	$E_{BN}/\mathbb{F}_{p257}$	4.38	–
[Beu+10b]	$E_{BN}/\mathbb{F}_{p254}$	2.95	2.82
[Ara+11]	$E_{BN}/\mathbb{F}_{p254}$	2.19	2.04
[Ara+12]	$\boldsymbol{H_2(DD)/\mathbb{F}_{2^{367}}}$	**4.44**	**2.75**
	$\boldsymbol{H_2(GD)/\mathbb{F}_{2^{367}}}$	**8.37**	**5.04**
	$\boldsymbol{H_2(GG)/\mathbb{F}_{2^{367}}}$	**17.0**	**9.90**

temps/surface ; celui-ci est atteint par [AHN13].

Implémentations ASIC et logicielles

Nous mentionnons également pour référence les implémentations matérielles pour ASIC (*Application Specific Integrated Circuit*) et les implémentations logicielles. Il est extrêmement délicat de comparer ces deux plateformes aux FPGA.

Nous donnons dans la Table 13.4 les performances des trois implémentations ASIC connues de couplages atteignant 128 bits de sécurité. Nous donnons, pour celles-ci, les quatre mêmes grandeurs que pour les implémentations FPGA : surface, fréquence, temps de calcul et compromis temps/surface. Les surfaces sont cependant non commensurables à celles données pour les FPGA et se mesurent en nombre de portes logiques utilisées pour le circuit. Ici encore, l'influence de l'évolution de la technologie est grande et il est difficile de comparer des circuits gravées en 130 nm et en 65 nm. Nous retiendrons cependant que l'architecture présentée dans [AHN13] est très performante sur ASIC également.

Nous avons également rassemblé les meilleures implémentations logicielles de couplages à 128 bits de sécurité connues à ce jour dans la Table 13.5. Notons qu'il existe également des implémentations à des niveaux de sécurité supérieurs en logiciel [Ara+13]. Les travaux que nous avons menés sur la courbe hyperelliptique H_2 ont également été exploités lors d'une collaboration avec Diego Aranha qui a réalisé l'implémentation logicielle correspondante [Ara+12]. Cette implémentation fournit à ce jour le couplage de type I le plus rapide en logiciel ; elle

est également très proche des implémentations des courbes Barreto-Naehrig.

Conclusion et perspectives

Dans ce manuscrit, nous avons présenté le cheminement qui nous a amenés aux diverses implémentations de couplages cryptographiques présentées ici et dans [Beu+09a ; Beu+11 ; Est10 ; Ara+12]. Afin d'obtenir de bonnes performances, nous avons cherché à revenir aux fondements algorithmiques, voire mathématiques, des couplages pour s'adapter au mieux à notre plateforme cible : le FPGA. Parce que la qualité d'une implémentation d'une primitive cryptographique est le fruit de nombreux choix, une approche rigoureuse demande d'avoir une vue d'ensemble détaillée de tous les aspects de la primitive étudiée : mathématiques, algorithmiques, arithmétiques, cryptanalytiques et architecturaux. Nous avons mis cette méthode en pratique dans le cas des couplages sur les courbes supersingulières et nous avons ainsi cherché à apporter une compréhension fine des techniques et « astuces » de la littérature. Ceci nous a conduits à un travail de généralisation et de synthèse donnant une construction générique. L'application de notre construction retrouve alors les résultats connus et nous permet d'envisager des couplages plus adaptés à nos implémentations.

Selon les contextes, les algorithmes et les accélérateurs matériels que nous avons développés sont utilisables directement. Cependant, certaines applications ont des besoins plus spécifiques qui demanderaient d'étudier d'autres aspects : consommation, résistance aux attaques par canaux cachés, niveaux de sécurité supérieurs, *etc.* De ce point de vue, nous pensons qu'il serait particulièrement intéressant de considérer la conception de circuits intégrés spécifiques (ASIC) pour lesquels la métrique change par rapport aux FPGA et de nombreuses optimisations vis-à-vis de la consommation électrique peuvent être mises en place. Nous avons concentré nos efforts sur l'étude d'une primitive cryptographique (les couplages) ; néanmoins, une approche plus intégrée proposant l'implémentation d'un protocole complet attesterait de façon plus convaincante de la praticabilité des outils cryptographiques dans les systèmes embarqués par exemple.

Toutefois, les progrès récents concernant le problème du logarithme discret sur les corps de petite caractéristique [Bar+13b] remettent en cause les estimations de sécurité sur lesquelles nous avons basé nos constructions. Par conséquent, l'utilisation de courbes supersingulières en petite caractéristique pour la cryptographie semble fortement compromise. Jusqu'à maintenant, le logarithme discret sur des corps en grande caractéristique paraît épargné ; les courbes Barreto-Naehrig, par exemple, restent donc de bons candidats pour obtenir des couplages cryptographiques même si elles ne permettent d'obtenir que des couplages de type III et pas de couplages symétriques.

Au delà de l'implémentation de couplages, notre travail est constitué de l'amélioration d'un certain nombre d'éléments pouvant servir dans un domaine plus large. Notre coprocesseur arithmétique (Chapitre 12) par exemple peut trouver de nombreux autres contextes d'application dès que ceux-ci requièrent une implémentation efficace de l'arithmétique de corps fini. Bien que notre accélérateur *ad hoc* (Chapitre 11) soit, comme son nom l'indique,

assez peu réutilisable, il démontre que le seuil à partir duquel les algorithmes de multiplications sous-quadratiques sont meilleurs que les algorithmes quadratiques est très bas pour les implémentations matérielles, bien plus que celui des implémentations logicielles. Enfin, nos travaux sur la complexité bilinéaire (Chapitre 9) sont aussi bien plus généraux que leurs seules applications aux couplages et à la multiplication de polynômes ; l'extension de ces travaux et l'amélioration de l'algorithme de recherche de formules forment une piste dont nous souhaitons continuer l'exploration.

La méthodologie que nous avons employée lors de cette thèse, c'est-à-dire l'étude combinée des aspects mathématiques, arithmétiques et algorithmiques lors du développement d'accélérateurs matériels, peut être utilisée dans d'autres domaines de la cryptographie. Nous pensons en premier lieu à la cryptographie reposant sur les réseaux euclidiens car son développement est en plein essor. En effet, elle offre un potentiel important en termes de fonctionnalités et son implémentation efficace représente un défi important à la vue de la taille des données à manipuler. La communauté produit actuellement les premiers composants pour les briques de base nécessaires comme les transformées de Fourier rapides courtes [Göt+12], le tirage aléatoire suivant une distribution en gaussienne discrète [GD12] et même pour une signature complète [GLP12]. Ces implémentations ouvrent la voie d'un travail de fond à la fois pour optimiser au mieux des accélérateurs pour la cryptographie reposant sur les réseaux euclidiens et pour le choix des paramètres de sécurité.

L'utilisation de circuits spécifiques n'est pas réservée à la partie constructive de la cryptologie. En cryptanalyse, il est important de bénéficier d'implémentations efficaces démontrant la faisabilité d'une attaque. Bien que les algorithmes en jeu pour la factorisation et le logarithme discret dans un corps fini soient en général très complexes, certaines portions de ces algorithmes pourraient bénéficier grandement d'accélérateurs dédiés comme, par exemple, les étapes de cofactorisation et d'algèbre linéaire [IKS07 ; IKS10]. Plus directement, la calcul du logarithme discret sur les courbes profite largement d'accélérateurs matériels, comme cela a par exemple été mis en évidence par Bailey *et al.* dans le cadre du challenge ECC2K-130 [Bai+09].

Bibliographie

[AAG99] Iris Anshel, Michael Anshel et Dorian Goldfeld. « An algebraic method for public-key cryptography ». In : *Mathematical Research Letters* 6.3 (1999), p. 287–292 (cf. p. 3).

[Adl79] Leonard M. Adleman. « A subexponential algorithm for the discrete logarithm problem with applications to cryptography ». In : *Foundations of Computer Science, IEEE Annual Symposium on*. IEEE Computer Society, oct. 1979, p. 55–60. doi : 10.1109/SFCS.1979.2 (cf. p. 48).

[Adl94] Leonard M. Adleman. « The function field sieve ». In : *Algorithmic Number Theory Symposium – ANTS-I*. Éd. : Leonard M. Adleman et Ming-Deh A. Huang. Lecture Notes in Computer Science 877. Springer, 1994, p. 108–121. doi : 10.1007/3-540-58691-1_48 (cf. p. 49).

[AH99] Leonard M. Adleman et Ming-Deh A. Huang. « Function Field Sieve Method for Discrete Logarithms over Finite Fields ». In : *Information and Computation* 151.1–2 (1999), p. 5–16. doi : 10.1006/inco.1998.2761 (cf. p. 49).

[AHN13] Jithra Adikari, M. Anwar Hasan et Christophe Negre. « Towards Faster and Greener Cryptoprocessor for Eta Pairing on Supersingular Elliptic Curve over $\mathbb{F}_{2^{1223}}$ ». In : *Selected Areas in Cryptography – SAC 2012*. Éd. : Lars R. Knudsen et Huapeng Wu. Lecture Notes in Computer Science 7707. Springer Berlin Heidelberg, 2013, p. 166–183. doi : 10.1007/978-3-642-35999-6_12 (cf. p. 188–190).

[Ajt96] Miklós Ajtai. « Generating hard instances of lattice problems ». In : *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. STOC '96. ACM, 1996, p. 99–108. doi : 10.1145/237814.237838 (cf. p. 3).

[Alb11] Martin R. Albrecht. *The M4RIE library for dense linear algebra over small fields with even characteristic*. Oct. 2011. arXiv : 1111.6900 [cs.MS] (cf. p. 122).

[ALH10] Diego F. Aranha, Julio López et Darrel R. Hankerson. « High-Speed Parallel Software Implementation of the η_T Pairing ». In : *Topics in Cryptology – CT-RSA 2010*. Éd. : Josef Pieprzyk. Lecture Notes in Computer Science 5985. Springer Berlin Heidelberg, 2010, p. 89–105. doi : 10.1007/978-3-642-11925-5_7 (cf. p. 190).

[Ara+11] Diego F. Aranha, Koray Karabina, Patrick Longa, Catherine H. Gebotys et Julio López. « Faster Explicit Formulas for Computing Pairings over Ordinary Curves ». In : *Advances in Cryptology – EUROCRYPT 2011*. Éd. : Kenneth G. Paterson. Lecture Notes in Computer Science 6632. Springer Berlin Hei-

delberg, 2011, p. 48–68. DOI : 10.1007/978-3-642-20465-4_5 (cf. p. 54, 190).

[Ara+12] Diego F. ARANHA, Jean-Luc BEUCHAT, Jérémie DETREY et Nicolas ESTIBALS. « Optimal Eta pairing on supersingular genus-2 binary hyperelliptic curves ». In : *Topics in Cryptology – CT-RSA 2012*. Éd. : Orr DUNKELMAN. Lecture Notes in Computer Science 7178. Springer, fév. 2012, p. 99–115. DOI : 10.1007/978-3-642-27954-6_7 (cf. p. 6, 76, 100, 182, 187–190, 193).

[Ara+13] Diego F. ARANHA, Laura FUENTES-CASTAÑEDA, Edward KNAPP, Alfred J. MENEZES et Francisco RODRÍGUEZ-HENRÍQUEZ. « Implementing Pairings at the 192-Bit Security Level ». In : *International Conference on Pairing-Based Cryptography – Pairing 2012*. Éd. : Michel ABDALLA et Tanja LANGE. Lecture Notes in Computer Science 7708. Springer Berlin Heidelberg, 2013, p. 177–195. DOI : 10.1007/978-3-642-36334-4_11 (cf. p. 190).

[Art91] Michael ARTIN. *Algebra*. Prentice-Hall, Inc., 1991 (cf. p. 131).

[Bai+09] Daniel V. BAILEY, Lejla BATINA, Daniel J. BERNSTEIN, Peter BIRKNERN, Joppe W. BOS, Hsieh-Chung CHEN, Chen-Mou CHENG, Gauthier van DAMME, Giacomo de MEULENAER, Luis Julian Dominguez PEREZ, Junfeng FAN, Tim GÜNEYSU, Frank GURKAYNAK, Thorsten KLEINJUNG, Tanja LANGE, Nele MENTENS, Ruben NIEDERHAGEN, Christof PAAR, Francesco REGAZZONI, Peter SCHWABE, Leif UHSADEL, Anthony Van HERREWEGE et Bo-Yin YANG. *Breaking ECC2K-130*. 2009. Cryptology ePrint Archive, Rapport 2009/541 (cf. p. 47, 194).

[Bar+02] Paulo S. L. M. BARRETO, H.Y. KIM, B. LYNN et Michael SCOTT. « Efficient algorithms for pairing-based cryptosystems ». In : *Advances in Cryptology – CRYPTO 2002*. Éd. : M. YUNG. Lecture Notes in Computer Science 2442. Springer, 2002, p. 354–368. DOI : 10.1007/3-540-45708-9_23 (cf. p. 64, 66).

[Bar+07] Paulo S. L. M. BARRETO, Steven D. GALBRAITH, Colm Ó hÉIGEARTAIGH et Michael SCOTT. « Efficient pairing computation on supersingular abelian varieties ». In : *Designs, Codes and Cryptography* 42.3 (2007), p. 239–271. DOI : 10.1007/s10623-006-9033-6 (cf. p. 69, 84, 93, 101, 103).

[Bar+08] Alessandro BARENGHI, Guido Marco BERTONI, Luca BREVEGLIERI et Gerardo PELOSI. « A FPGA Coprocessor for the Cryptographic Tate Pairing over $\mathbb{F}_p$ ». In : *Proceedings of the Fifth International Conference on Information Technology : New Generations (ITNG'08)*. IEEE Computer Society, avr. 2008, p. 112–119. DOI : 10.1109/ITNG.2008.260 (cf. p. 185, 186).

[Bar+12] Razvan BARBULESCU, Jérémie DETREY, Nicolas ESTIBALS et Paul ZIMMERMANN. « Finding optimal formulae for bilinear map ». In : *International Workshop on the Arithmetic of Finite Fields – WAIFI 2012*. Éd. : Ferruh ÖZBUDAK et Francisco RODRÍGUEZ-HENRÍQUEZ. Lecture Notes in Computer Science 7369. Springer, juil. 2012, p. 168–186. DOI : 10.1007/978-3-642-31662-3_12 (cf. p. 6, 121).

[Bar+13a] Razvan Barbulescu, Cyril Bouvier, Jérémie Detrey, Pierrick Gaudry, Hamza Jeljeli, Emmanuel Thomé, Marion Videau et Paul Zimmermann. *Discrete logarithm in GF(2^{809}) with FFS*. 2013. Cryptology ePrint Archive, Rapport 2013/197 (cf. p. 49).

[Bar+13b] Razvan Barbulescu, Pierrick Gaudry, Antoine Joux et Emmanuel Thomé. *A quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic*. 2013. Cryptology ePrint Archive, Rapport 2013/400 (cf. p. 7, 193).

[BCS97] Peter Bürgisser, Michael Clausen et Mohammad A. Shokrollahi. *Algebraic Complexity Theory*. Grundlehren Der Mathematischen Wissenschaften. Springer, 1997. doi : 10.1007/978-3-662-03338-8 (cf. p. 121, 123, 124).

[Beu+07] Jean-Luc Beuchat, Nicolas Brisebarre, Jérémie Detrey et Eiji Okamoto. « Arithmetic Operators for Pairing-Based Cryptography ». In : *Cryptographic Hardware and Embedded Systems - CHES 2007*. Éd. : Pascal Paillier et Ingrid Verbauwhede. Lecture Notes in Computer Science 4727. Springer Berlin / Heidelberg, 2007, p. 239–255. doi : 10.1007/978-3-540-74735-2_17 (cf. p. 170).

[Beu+08a] Jean-Luc Beuchat, Nicolas Brisebarre, Jérémie Detrey, Eiji Okamoto et Francisco Rodríguez-Henríquez. « A Comparison Between Hardware Accelerators for the Modified Tate Pairing over $\mathbb{F}_{2^m}$ and $\mathbb{F}_{3^m}$ ». In : *Pairing 2008*. Éd. : Steven D. Galbraith et Kenneth G. Paterson. Lecture Notes in Computer Science 5209. Springer, 2008, p. 297–315. doi : 10.1007/978-3-540-85538-5_20 (cf. p. 85, 88, 115, 162, 186, 187).

[Beu+08b] Jean-Luc Beuchat, Nicolas Brisebarre, Jérémie Detrey, Eiji Okamoto, M. Shirase et Tsuyoshi Takagi. « Algorithms and Arithmetic Operators for Computing the η_T Pairing in Characteristic Three ». In : *IEEE Transactions in Computers* 57.11 (nov. 2008), p. 1454–1468. doi : 10.1109/TC.2008.103 (cf. p. 94, 96–98).

[Beu+09a] Jean-Luc Beuchat, Jérémie Detrey, Nicolas Estibals, Eiji Okamoto et Francisco Rodríguez-Henríquez. « Hardware accelerator for the Tate pairing in characteristic three based on Karatsuba-Ofman multipliers ». In : *Cryptographic Hardware and Embedded Systems – CHES 2009*. Éd. : Christophe Clavier et Kris Gaj. Lecture Notes in Computer Science 5747. Springer, août 2009, p. 225–239. doi : 10.1007/978-3-642-04138-9_17 (cf. p. 6, 94, 96, 151, 193).

[Beu+09b] Jean-Luc Beuchat, Emmanuel López-Trejo, Luis Martínez-Ramos, Shigeo Mitsunari et Francisco Rodríguez-Henríquez. « Multi-core Implementation of the Tate Pairing over Supersingular Elliptic Curves ». In : *Cryptology and Network Security – CANS 2009*. Éd. : J. A. Garay, Atsuko Miyaji et Akira Otsuka. Lecture Notes in Computer Science 5888. Springer, 2009, p. 413–432. doi : 10.1007/978-3-642-10433-6_28 (cf. p. 190).

[Beu+10a] Jean-Luc Beuchat, H. Doi, K. Fujita, A. Inomata, P. Ith, A. Kanaoka, M. Katouno, M. Mambo, E. Okamoto, T. Okamoto, T. Shiga, M. Shirase, R. Soga, Tsuyoshi Takagi, A. Vithanage et H. Yamamoto. « FPGA and ASIC Implementations of the η_T Pairing in Characteristic Three ». In : *Com-*

puters and Electrical Engineering 36.1 (jan. 2010), p. 73–87. DOI : 10.1016/j.compeleceng.2009.05.001 (cf. p. 157).

[Beu+10b] Jean-Luc BEUCHAT, Jorge Enrique GONZÁLEZ-DÍAZ, Shigeo MITSUNARI, Eiji OKAMOTO, Francisco RODRÍGUEZ-HENRÍQUEZ et Tadanori TERUYA. « High-Speed Software Implementation of the Optimal Ate Pairing over Barreto–Naehrig Curves ». In : *International Conference on Pairing-Based Cryptography – Pairing 2010*. Éd. : Marc JOYE, Atsuko MIYAJI et Akira OTSUKA. Lecture Notes in Computer Science 6487. Springer Berlin Heidelberg, 2010, p. 21–39. DOI : 10.1007/978-3-642-17455-1_2 (cf. p. 190).

[Beu+11] Jean-Luc BEUCHAT, Jérémie DETREY, Nicolas ESTIBALS, Eiji OKAMOTO et Francisco RODRÍGUEZ-HENRÍQUEZ. « Fast architectures for the η_T pairing over small-characteristic supersingular elliptic curves ». In : *IEEE Transactions in Computers* 60.2 (fév. 2011), p. 266–281. DOI : 10.1109/TC.2010.163 (cf. p. 6, 94, 96, 110, 151, 186, 193).

[BF01] Dan BONEH et Matthew K. FRANKLIN. « Identity-based encryption from the Weil pairing ». In : *Advances in Cryptology – CRYPTO 2001*. Lecture Notes in Computer Science 2139. Springer Berlin / Heidelberg, 2001, p. 213–229. DOI : 10.1007/3-540-44647-8_13 (cf. p. 3, 4).

[BF03] Dan BONEH et Matthew K. FRANKLIN. « Identity-based encryption from the Weil pairing ». In : *SIAM Journal on Computing* 32.3 (2003), p. 586–615. DOI : 10.1137/S0097539701398521 (cf. p. 3, 4).

[BG04] Daniel R. L. BROWN et Robert P. GALLANT. *The Static Diffie-Hellman Problem*. 2004. Cryptology ePrint Archive, Rapport 2004/306 (cf. p. 50).

[Blä03] Markus BLÄSER. « On the complexity of the multiplication of matrices of small formats ». In : *Journal of Complexity* 19.1 (fév. 2003), p. 43–60. DOI : 10.1016/S0885-064X(02)00007-9 (cf. p. 126).

[BLS01] Dan BONEH, Ben LYNN et Hovav SHACHAM. « Short Signatures from the Weil Pairing ». In : *Advances in Cryptology - ASIACRYPT 2001*. Éd. : Colin BOYD. Lecture Notes in Computer Science 2248. Springer Berlin / Heidelberg, 2001, p. 514–532. DOI : 10.1007/3-540-45682-1_30 (cf. p. 3, 5).

[BLS03] Paulo S. L. M. BARRETO, Ben LYNN et Michael SCOTT. « Constructing Elliptic Curves with Prescribed Embedding Degrees ». In : *Security in Communication Networks*. Éd. : Stelvio CIMATO, Giuseppe PERSIANO et Clemente GALDI. Lecture Notes in Computer Science 2576. Springer Berlin Heidelberg, 2003, p. 257–267. DOI : 10.1007/3-540-36413-7_19 (cf. p. 55).

[BLS04] Dan BONEH, Ben LYNN et Hovav SHACHAM. « Short Signatures from the Weil Pairing ». In : *Journal of Cryptology* 17.4 (2004), p. 297–319. DOI : 10.1007/s00145-004-0314-9 (cf. p. 3, 5, 105).

[BN06] Paulo S. L. M. BARRETO et Michael NAEHRIG. « Pairing-Friendly Elliptic Curves of Prime Order ». In : *Selected Areas in Cryptography – SAC 2005*. Éd. : Bart PRENEEL et Stafford E. TAVARES. Lecture Notes in Computer Science 3897. Springer, 2006, p. 319–331. DOI : 10.1007/11693383_22 (cf. p. 54, 55).

[BSS05] Ian F. Blake, Gadiel Seroussi et Nigel P. Smart. *Advances in elliptic curve cryptography*. Cambridge University Press, 2005. DOI : 10.1017/CBO9780511546570 (cf. p. 40).

[BW05] Friederike Brezing et Annegret Weng. « Elliptic Curves Suitable for Pairing Based Cryptography ». In : *Designs, Codes and Cryptography* 37 (1 2005), p. 133–141. DOI : 10.1007/s10623-004-3808-4 (cf. p. 54).

[Can87] David G. Cantor. « Computing in the Jacobian of a hyperelliptic curve ». In : *Mathematics of computation* 48 (1987), p. 95–101. DOI : 10.1090/S0025-5718-1987-0866101-0 (cf. p. 61).

[CBH11] Nicolas T. Courtois, Gregory V. Bard et Daniel Hulme. *A New General-Purpose Method to Multiply* 3×3 *Matrices Using Only* 23 *Multiplications*. Mar. 2011. arXiv : 1108.2830 [cs.SC] (cf. p. 126).

[CEP83] E. Rodney Canfield, Paul Erdős et Carl Pomerance. « On a problem of Oppenheim concerning "factorisatio numerorum" ». In : *Journal of Number Theory* 17.1 (août 1983), p. 1–28. DOI : 10.1016/0022-314X(83)90002-1 (cf. p. 48).

[Cer97] Certicom. *Certicom ECC Chalenge*. http://www.certicom.com/images/pdfs/cert_ecc_challenge.pdf. 1997 (cf. p. 46).

[Ces08] Emanuele Cesena. *Pairing with Supersingular Trace Zero Varieties Revisited*. 2008. Cryptology ePrint Archive, Rapport 2008/404 (cf. p. 175).

[CFA06] H. Cohen, G. Frey et R. Avanzi. *Handbook of elliptic and hyperelliptic curve cryptography*. CRC press, 2006. DOI : 10.1201/9781420034981 (cf. p. 61).

[CH07] Jaewook Chung et M. Anwar Hasan. « Asymmetric Squaring Formulae ». In : *Symposium on Computer Arithmetic – ARITH-18*. Éd. : Peter Kornerup et Jean-Michel Muller. IEEE Computer Society, 2007, p. 113–122. DOI : 10.1109/ARITH.2007.11 (cf. p. 121).

[Che+11] Ray C. C. Cheung, Sylvain Duquesne, Junfeng Fan, Nicolas Guillermin, Ingrid Verbauwhede et Gavin Xiaoxu Yao. « FPGA Implementation of Pairings Using Residue Number System and Lazy Reduction ». In : *Cryptographic Hardware and Embedded Systems – CHES 2011*. Éd. : Bart Preneel et Tsuyoshi Takagi. Lecture Notes in Computer Science 6917. Springer, 2011, p. 421–441. DOI : 10.1007/978-3-642-23951-9_28 (cf. p. 188, 189).

[CHM10] Sanjit Chatterjee, Darrel R. Hankerson et Alfred J. Menezes. « On the Efficiency and Security of Pairing-Based Protocols in the Type 1 and Type 4 Settings ». In : *International Workshop on the Arithmetic of Finite Fields – WAIFI 2010*. Éd. : M. Anwar Hasan et Tor Helleseth. Lecture Notes in Computer Science 6087. Springer, 2010, p. 114–134. DOI : 10.1007/978-3-642-13797-6 (cf. p. 105).

[CNP05] Gabriel Cardona, Enric Nart et Jordi Pujolàs. « Curves of genus two over fields of even characteristic ». In : *Mathematische Zeitschrift* 250 (1 2005), p. 177–201. DOI : 10.1007/s00209-004-0750-0 (cf. p. 46).

[CÖ08] Murat Cenk et Ferruh Özbudak. « Efficient multiplication in $\mathbb{F}_{3^{\ell m}}, m \geq 1$ and $5 \leq \ell \leq 18$ ». In : *Advances in Cryptology - AFRICACRYPT 2008*. Éd. : Serge Vaudenay. Lecture Notes in Computer Science 6055. Springer, 2008, p. 406–414. doi : 10.1007/978-3-540-68164-9_27 (cf. p. 118).

[CÖ09] Murat Cenk et Ferruh Özbudak. « Improved Polynomial Multiplication Formulas over $\mathbb{F}_2$ Using Chinese Remainder Theorem ». In : *IEEE Transactions in Computers* 58 (2009), p. 572–576. doi : 10.1109/TC.2008.207 (cf. p. 118).

[CÖ10] Murat Cenk et Ferruh Özbudak. « On multiplication in finite fields ». In : *Journal of Complexity* 26.2 (2010), p. 172–186. doi : 10.1016/j.jco.2009.11.002 (cf. p. 125, 138, 140).

[Com+08] Pierre Comon, Gene Golub, Lek-Heng Lim et Bernard Mourrain. « Symmetric Tensors and Symmetric Tensor Rank ». In : *SIAM Journal on Matrix Analysis and Applications* 30.3 (sept. 2008), p. 1254–1279. doi : 10.1137/060661569 (cf. p. 137).

[Coo66] Stephen A. Cook. « On the Minimum Computation Time of Function ». Thèse de doct. Harvard University, 1966, p. 51–71 (cf. p. 116).

[Cou01] Jean-Marc Couveignes. « Algebraic Groups and Discrete Logarithm ». In : *Public-Key Cryptography and Computational Number Theory*. Éd. : Kazimierz lster, Jerzy Urbanowicz et Hugh C. Williams. De Gruyter Proceedings in Mathematics. de Gruyter, 2001, p. 17–27. doi : 10.1515/9783110881035.17 (cf. p. 47).

[CP01] C. Cocks et R. G. E. Pinch. « Identity-based cryptosystems based on the Weil pairing ». Unpublished manuscript. 2001 (cf. p. 54).

[DGM99] Iwan M. Duursma, Pierrick Gaudry et François Morain. « Speeding up the Discrete Log Computation on Curves with Automorphisms ». In : *Advances in Cryptology - ASIACRYPT'99*. Éd. : Kwok-Yan Lam, Eiji Okamoto et Chaoping Xing. Lecture Notes in Computer Science 1716. Springer, 1999, p. 103–121. doi : 10.1007/978-3-540-48000-6_10 (cf. p. 46).

[DH76] Whitfield Diffie et Martin E. Hellman. « New directions in cryptography ». In : *IEEE Transactions on Information Theory* 22.6 (1976), p. 644–654. doi : 10.1109/TIT.1976.1055638 (cf. p. 2, 3).

[Die11] Claus Diem. « On the discrete logarithm problem in class groups of curves ». In : *Mathematics of computation* 80.273 (2011), p. 443–475. doi : 10.1090/S0025-5718-2010-02281-1 (cf. p. 50).

[DL03] Iwan M. Duursma et Hyang-Sook Lee. « Tate Pairing Implementation for Hyperelliptic Curves $y^2 = x^p - x + d$ ». In : *Advances in Cryptology - ASIACRYPT 2003*. Éd. : Chi-Sung Laih. Lecture Notes in Computer Science 2894. Springer Berlin / Heidelberg, 2003, p. 111–123. doi : 10.1007/978-3-540-40061-5_7 (cf. p. 61, 69, 93).

[DM91] Jean Duprat et Jean-Michel Muller. « Écrire les nombres autrement pour calculer plus vite. » In : *Technique et Science Informatique* 10.3 (1991) (cf. p. 110, 152).

[ElG84] Taher ElGamal. « A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms ». In : *Advances in Cryptology – CRYPTO 1985*. Éd. : G. R. Blakley et David Chaum. Lecture Notes in Computer Science 196. Springer, 1984, p. 10–18. DOI : 10.1007/3-540-39568-7_2 (cf. p. 50).

[Est10] Nicolas Estibals. « Compact hardware for computing the Tate pairing over 128-bit-security supersingular curves ». In : *International Conference on Pairing-Based Cryptography – Pairing 2010*. Éd. : Marc Joye, Atsuko Miyaji et Akira Otsuka. Lecture Notes in Computer Science 6487. Springer, déc. 2010, p. 397–416. DOI : 10.1007/978-3-642-17455-1_25 (cf. p. 6, 94, 96, 118, 140, 173, 182, 187–189, 193).

[Fan+07] Haining Fan, Jiaguang Sun, Ming Gu et Kwok-Yan Lam. *Overlap-free Karatsuba-Ofman polynomial multiplication algorithm*. 2007. Cryptology ePrint Archive, Rapport 2007/393 (cf. p. 118).

[FH07] Haining Fan et M. Anwar Hasan. « Comments on Montgomery's "Five, Six, and Seven-Term Karatsuba-Like Formulae" ». In : *IEEE Transactions in Computers* 56.5 (mai 2007), p. 716–717. DOI : 10.1109/TC.2007.1024 (cf. p. 121).

[FR94] Gerhard Frey et Hans-Georg Rück. « A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves ». In : *Mathematics of computation* 62.206 (avr. 1994), p. 865–874. DOI : 10.2307/2153546 (cf. p. 3, 35, 44).

[Fre01] Gerhard Frey. « Applications of Arithmetical Geometry to Cryptographic Constructions ». In : *Proceedings of the Fifth International Conference on Finite Fields and Applications (Augsburg, 1999)*. Éd. : Dieter Jungnickel et Harald Niederreiter. Springer Berlin Heidelberg, 2001, p. 128–161. DOI : 10.1007/978-3-642-56755-1_13 (cf. p. 3, 175).

[FST10] David Freeman, Michael Scott et Edlyn Teske. « A Taxonomy of Pairing-Friendly Elliptic Curves ». In : *Journal of Cryptology* 23 (2 2010), p. 224–280. DOI : 10.1007/s00145-009-9048-z (cf. p. 54).

[FVV09] Junfeng Fan, Frederik Vercauteren et Ingrid Verbauwhede. « Faster $\mathbb{F}_p$-arithmetic for Cryptographic Pairings on Barreto-Naehrig Curves ». In : *Cryptographic Hardware and Embedded Systems – CHES 2009*. Éd. : Christophe Clavier et Kris Gaj. Lecture Notes in Computer Science 5747. Springer, 2009, p. 240–253. DOI : 10.1007/978-3-642-04138-9_18 (cf. p. 190).

[FVV12] Junfeng Fan, Frederik Vercauteren et Ingrid Verbauwhede. « Efficient Hardware Implementation of Fp-Arithmetic for Pairing-Friendly Curves ». In : *IEEE Transactions in Computers* 61.5 (mai 2012), p. 676–685. DOI : 10.1109/TC.2011.78 (cf. p. 188, 189).

[Gal+09] Steven D. Galbraith, Jordi Pujolàs, Christophe Ritzenthaler et Ben Smith. « Distortion Maps for Genus Two Curves ». In : *Journal of Mathematical Cryptology* 3.1 (juin 2009), p. 1–18. DOI : 10.1515/JMC.2009.001 (cf. p. 41, 46, 100).

[Gal10] Raminder S. Galbraith Steven D.and Ruprai. *Using Equivalence Classes to Accelerate Solving the Discrete Logarithm Problem in a Short Interval*. 2010. Cryptology ePrint Archive, Rapport 2010/615 (cf. p. 45).

[Gal12] Steven D. Galbraith. *The Mathematics of Public Key Cryptography*. Cambridge University Press, 2012 (cf. p. 24, 28, 45).

[Gau+07] Pierrick Gaudry, Emmanuel Thomé, Nicolas Thériault et Claus Diem. « A double large prime variation for small genus hyperelliptic index calculus ». In : *Mathematics of computation* 76.257 (2007), p. 475–492. doi : 10.1090/S0025-5718-06-01900-4 (cf. p. 48).

[Gau08] Pierrick Gaudry. « Algorithmique des courbes algébriques pour la cryptologie ». Thèse d'hab. Université Henri Poincaré - Nancy I, oct. 2008. url : http://hal.inria.fr/tel-00514843 (cf. p. 48).

[Gau09] Pierrick Gaudry. « Index calculus for abelian varieties and the elliptic curve discrete logarithm problem ». In : *Journal of Symbolic Compution* 44.12 (2009), p. 1690–1702. doi : 10.1016/j.jsc.2008.08.005 (cf. p. 50).

[GD12] Steven D. Galbraith et Nagarjun C. Dwarakanath. *Efficient sampling from discrete gaussians for lattice-based cryptography on a constrained device*. 2012. url : http://www.math.auckland.ac.nz/~sgal018/gen-gaussians.pdf (cf. p. 194).

[Gen09] Craig Gentry. « Fully homomorphic encryption using ideal lattices ». In : *Proceedings of the 41st annual ACM symposium on Theory of computing*. STOC '09. ACM, 2009, p. 169–178. doi : 10.1145/1536414.1536440 (cf. p. 3).

[GH12] Steven D. Galbraith et Mark Holmes. « A non-uniform birthday problem with applications to discrete logarithms ». In : *Discrete Applied Mathematics* 160.10–11 (2012), p. 1547–1560. doi : 10.1016/j.dam.2012.02.019 (cf. p. 45).

[GHS02] Pierrick Gaudry, Florian Hess et Nigel P. Smart. « Constructive and destructive facets of Weil descent on elliptic curves ». In : *Journal of Cryptology* 15.1 (2002), p. 19–46. doi : 10.1007/s00145-001-0011-x (cf. p. 50).

[GK86] Shafi Goldwasser et Joe Kilian. « Almost all primes can be quickly certified ». In : *Proceedings of the eighteenth annual ACM symposium on Theory of computing*. STOC '86. ACM, 1986, p. 316–329. doi : 10.1145/12130.12162 (cf. p. 3).

[GLP12] Tim Güneysu, Vadim Lyubashevsky et Thomas Pöppelmann. « Practical Lattice-Based Cryptography : A Signature Scheme for Embedded Systems ». In : *Cryptographic Hardware and Embedded Systems – CHES 2012*. Éd. : Emmanuel Prouff et Patrick Schaumont. Lecture Notes in Computer Science 7428. Springer Berlin Heidelberg, 2012, p. 530–547. doi : 10.1007/978-3-642-33027-8_31 (cf. p. 194).

[GMR10] Santosh Ghosh, Debdeep Mukhopadhyay et Dipanwita Roychowdhury. « High Speed Flexible Pairing Cryptoprocessor on FPGA Platform ». In : *International Conference on Pairing-Based Cryptography – Pairing 2010*. Éd. : Marc Joye, Atsuko Miyaji et Akira Otsuka. Lecture Notes in Computer Science 6487. Springer Berlin Heidelberg, 2010, p. 450–466. doi : 10.1007/978-3-642-17455-1_28 (cf. p. 188).

[Gor93] Daniel M. Gordon. « Discrete Logarithms in $GF(p)$ Using the Number Field Sieve ». In : *SIAM Journal on Discrete Mathematics* 6.1 (1993), p. 124–138. doi : 10.1137/0406010 (cf. p. 49).

[Göt+12] Norman Göttert, Thomas Feller, Michael Schneider, Johannes Buchmann et Sorin Huss. « On the Design of Hardware Building Blocks for Modern Lattice-Based Encryption Schemes ». In : *Cryptographic Hardware and Embedded Systems – CHES 2012*. Éd. : Emmanuel Prouff et Patrick Schaumont. Lecture Notes in Computer Science 7428. Springer Berlin Heidelberg, 2012, p. 512–529. DOI : 10.1007/978-3-642-33027-8_30 (cf. p. 194).

[GP02] Jorge Guajardo et Christof Paar. « Itoh-Tsujii inversion in standard basis and its application in cryptography and codes ». In : *Designs, Codes and Cryptography* 25.2 (2002), p. 207–216. DOI : 10.1023/A:1013860532636 (cf. p. 111).

[GPR13] Steven D. Galbraith, John M. Pollard et Raminder S. Ruprai. « Computing discrete logarithms in an interval ». In : *Mathematics of computation* 82.282 (2013). DOI : 10.1090/S0025-5718-2012-02641-X (cf. p. 45).

[GPS07] Elisa Gorla, Christoph Puttmann et Jamshid Shokrollahi. « Explicit Formulas for Efficient Multiplication in $\mathbb{F}_{36^m}$ ». In : *Selected Areas in Cryptography – SAC 2007*. Éd. : Carlisle Adams, Ali Miri et Michael J. Wiener. Lecture Notes in Computer Science 4876. Springer, 2007, p. 173–183. DOI : 10.1007/978-3-540-77360-3_12 (cf. p. 157).

[GPS08] Steven D. Galbraith, Kenneth G. Paterson et Nigel P. Smart. « Pairings for cryptographers ». In : *Discrete Applied Mathematics* 156.16 (sept. 2008), p. 3113–3121. DOI : 10.1016/j.dam.2007.12.010 (cf. p. 41).

[Gra+07] Robert Granger, Florian Hess, Roger Oyono, Nicolas Thériault et Frederik Vercauteren. « Ate Pairing on Hyperelliptic Curves ». In : *Advances in Cryptology – EUROCRYPT 2007*. Éd. : Moni Naor. Lecture Notes in Computer Science 4515. Springer Berlin / Heidelberg, 2007, p. 430–447. DOI : 10.1007/978-3-540-72540-4_25 (cf. p. 40, 65, 69).

[Gra+13a] Robert Granger, Faruk Gologlu, Gary McGuire et Jens Zumbragel. *Discrete Logarithms in $GF(2^{1971})$*. https://listserv.nodak.edu/cgi-bin/wa.exe?A2=ind1302&L=NMBRTHRY&F=&S=&P=4793. 19 fév. 2013 (cf. p. 7).

[Gra+13b] Robert Granger, Faruk Gologlu, Gary McGuire et Jens Zumbragel. *Discrete Logarithms in $GF(2^{6120})$*. https://listserv.nodak.edu/cgi-bin/wa.exe?A2=ind1304&L=NMBRTHRY&F=&S=&P=9793. 11 avr. 2013 (cf. p. 7).

[Gra10] Robert Granger. « On the Static Diffie-Hellman Problem on Elliptic Curves over Extension Fields ». In : *Advances in Cryptology - ASIACRYPT 2010*. Éd. : Masayuki Abe. Lecture Notes in Computer Science 6477. Springer Berlin / Heidelberg, 2010, p. 283–302. DOI : 10.1007/978-3-642-17373-8_17 (cf. p. 50).

[GRD11] Santosh Ghosh, Dipanwita Roychowdhury et Abhijit Das. « High Speed Cryptoprocessor for η_T Pairing on 128-bit Secure Supersingular Elliptic Curves over Characteristic Two Fields ». In : *Cryptographic Hardware and Embedded Systems – CHES 2011*. Éd. : Bart Preneel et Tsuyoshi Takagi. Lecture Notes in Computer Science 6917. Springer, 2011, p. 442–458. DOI : 10.1007/978-3-642-23951-9_29 (cf. p. 188, 189).

[GV11] Aurore GUILLEVIC et Damien VERGNAUD. *Genus 2 Hyperelliptic Curve Families with Explicit Jacobian Order Evaluation and Pairing-Friendly Constructions*. 2011. Cryptology ePrint Archive, Rapport 2011/604 (cf. p. 55).

[GVR13] Santosh GHOSH, Ingrid VERBAUWHEDE et Dipanwita ROYCHOWDHURY. « Core Based Architecture to Speed Up Optimal Ate Pairing on FPGA Platform ». In : *International Conference on Pairing-Based Cryptography – Pairing 2012*. Éd. : Michel ABDALLA et Tanja LANGE. Lecture Notes in Computer Science 7708. Springer Berlin Heidelberg, 2013, p. 141–159. DOI : 10.1007/978-3-642-36334-4_9 (cf. p. 188, 189).

[Hås90] Johan HÅSTAD. « Tensor rank is NP-complete ». In : *Journal of Algorithms* 11.4 (déc. 1990), p. 644–654. DOI : 10.1016/0196-6774(90)90014-6 (cf. p. 124).

[Hay+10] Takuya HAYASHI, Naoyuki SHINOHARA, Lihua WANG, Shin'ichiro MATSUO, Masaaki SHIRASE et Tsuyoshi TAKAGI. « Solving a 676-Bit Discrete Logarithm Problem in $GF(3^{6n})$ ». In : *Public Key Cryptography – PKC 2010*. Éd. : Phong NGUYEN et David POINTCHEVAL. Lecture Notes in Computer Science 6056. Springer Berlin / Heidelberg, 2010, p. 351–367. DOI : 10.1007/978-3-642-13013-7_21 (cf. p. 49).

[Hay+12] Takuya HAYASHI, Takeshi SHIMOYAMA, Naoyuki SHINOHARA et Tsuyoshi TAKAGI. *Breaking pairing-based cryptosystems using η_T pairing over $GF(3^{97})$*. 2012. Cryptology ePrint Archive, Rapport 2012/345 (cf. p. 49).

[Hes04a] Florian HESS. « A note on the Tate pairing of curves over finite fields ». In : *Archiv der Mathematik* 82 (1 2004), p. 28–32. DOI : 10.1007/s00013-003-4773-2 (cf. p. 35).

[Hes04b] Florian HESS. « Computing relations in divisor class groups of algebraic curves over finite fields ». preprint. 2004. URL : http://www.staff.uni-oldenburg.de/florian.hess/publications/dlog.pdf (cf. p. 47).

[Hes08] Florian HESS. « Pairing Lattices ». In : *International Conference on Pairing-Based Cryptography – Pairing 2008*. Éd. : Steven D. GALBRAITH et Kenneth G. PATERSON. Lecture Notes in Computer Science 5209. Springer, 2008, p. 18–38. DOI : 10.1007/978-3-540-85538-5_2 (cf. p. 69, 76).

[HPS98] Jeffrey HOFFSTEIN, Jill PIPHER et Joseph H. SILVERMAN. « NTRU : A ring-based public key cryptosystem ». In : *Algorithmic Number Theory Symposium – ANTS-III*. Éd. : Joe P. BUHLER. Lecture Notes in Computer Science 1423. Springer Berlin Heidelberg, juin 1998, p. 267–288. DOI : 10.1007/BFb0054868 (cf. p. 3).

[HSV06] Florian HESS, Nigel P. SMART et Frederik VERCAUTEREN. « The Eta Pairing Revisited ». In : *IEEE Transactions on Information Theory* 52.10 (oct. 2006), p. 4595–4602. DOI : 10.1109/TIT.2006.881709 (cf. p. 43, 69).

[HZ04] Guillaume HANROT et Paul ZIMMERMANN. « A long note on Mulders' short product ». In : *Journal of Symbolic Computation* 37.3 (2004), p. 391–401. DOI : 10.1016/j.jsc.2003.03.001 (cf. p. 115, 155).

[IKS07] Tetsuya Izu, Jun Kogure et Takeshi Shimoyama. « CAIRN 2 : An FPGA Implementation of the Sieving Step in the Number Field Sieve Method ». In : *Cryptographic Hardware and Embedded Systems – CHES 2007*. Éd. : Pascal Paillier et Ingrid Verbauwhede. Lecture Notes in Computer Science 4727. Springer Berlin Heidelberg, 2007, p. 364–377. DOI : 10.1007/978-3-540-74735-2_25 (cf. p. 194).

[IKS10] Tetsuya Izu, Jun Kogure et Takeshi Shimoyama. « CAIRN : Dedicated Integer Factoring Devices ». In : *Network-Based Information Systems (NBiS), 2010*. 2010, p. 558–563. DOI : 10.1109/NBiS.2010.60 (cf. p. 194).

[IT88] Toshiya Itoh et Shigeo Tsujii. « A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases ». In : *Information and Computation* 78.3 (1988), p. 171–177. DOI : 10.1016/0890-5401(88)90024-7 (cf. p. 111).

[JL05] Antoine Joux et Reynald Lercier. *Discrete logarithms in $GF(2^{607})$ and $GF(2^{613})$*. https://listserv.nodak.edu/cgi-bin/wa.exe?A2=ind0509&L=NMBRTHRY&P=R1490. 22 sept. 2005 (cf. p. 49).

[Jou+09] Antoine Joux, Reynald Lercier, David Naccache et Emmanuel Thomé. « Oracle-Assisted Static Diffie-Hellman Is Easier Than Discrete Logarithms ». In : *Cryptography and Coding*. Éd. : Matthew G. Parker. Vol. 5921. Lecture Notes in Computer Science. Springer Berlin Heidelberg, déc. 2009, p. 351–367. DOI : 10.1007/978-3-642-10868-6_21 (cf. p. 50).

[Jou00] Antoine Joux. « A One Round Protocol for Tripartite Diffie-–Hellman ». In : *Algorithmic Number Theory Symposium – ANTS-IV*. Éd. : Wieb Bosma. Lecture Notes in Computer Science 1838. Springer Berlin / Heidelberg, 2000, p. 385–393. DOI : 10.1007/10722028_23 (cf. p. 3, 4).

[Jou04] Antoine Joux. « A One Round Protocol for Tripartite Diffie–Hellman ». In : *Journal of Cryptology* 17.4 (2004), p. 263–276. DOI : 10.1007/s00145-004-0312-y (cf. p. 3, 4).

[Jou12] Antoine Joux. *Discrete logarithms in a 1175-bit finite field*. https://listserv.nodak.edu/cgi-bin/wa.exe?A2=ind1212&L=NMBRTHRY&F=&S=&P=13902. 24 déc. 2012 (cf. p. 7).

[Jou13a] Antoine Joux. *Discrete logarithms in a 1425-bit finite field*. https://listserv.nodak.edu/cgi-bin/wa.exe?A2=ind1301&L=NMBRTHRY&F=&S=&P=2214. 6 jan. 2013 (cf. p. 7).

[Jou13b] Antoine Joux. *Discrete Logarithms in $GF(2^{1778})$*. https://listserv.nodak.edu/cgi-bin/wa.exe?A2=ind1302&L=NMBRTHRY&F=&S=&P=2317. 11 fév. 2013 (cf. p. 7).

[Jou13c] Antoine Joux. *Discrete Logarithms in $GF(2^{4080})$*. https://listserv.nodak.edu/cgi-bin/wa.exe?A2=ind1303&L=NMBRTHRY&F=&S=&P=13682. 22 mar. 2013 (cf. p. 7).

[Jou13d] Antoine Joux. *Discrete Logarithms in $GF(2^{6168})[= GF((2^{257})^{24})]$*. https://listserv.nodak.edu/cgi-bin/wa.exe?A2=ind1305&L=NMBRTHRY&F=&S=&P=3034. 21 mai 2013 (cf. p. 7).

[JV11] Antoine Joux et Vanessa Vitse. « Elliptic Curve Discrete Logarithm Problem over Small Degree Extension Fields ». In : *Journal of Cryptology* (2011). doi : 10.1007/s00145-011-9116-z (cf. p. 50).

[Kam+09] David Kammler, Diandian Zhang, Peter Schwabe, Hanno Scharwaechter, Markus Langenberg, Dominik Auras, Gerd Ascheid et Rudolf Mathar. « Designing an ASIP for cryptographic pairings over Barreto-Naehrig curves ». In : *Cryptographic Hardware and Embedded Systems – CHES 2009*. Éd. : C. Clavier et K. Gaj. Lecture Notes in Computer Science 5747. Springer, 2009, p. 254–271. doi : 10.1007/978-3-642-04138-9_19 (cf. p. 190).

[Ker83a] Auguste Kerckhoffs. « La cryptographie militaire ». In : *Journal des sciences militaires* IX (jan. 1883), p. 5–38 (cf. p. 1).

[Ker83b] Auguste Kerckhoffs. « La cryptographie militaire II ». In : *Journal des sciences militaires* IX (fév. 1883), p. 161–191 (cf. p. 1).

[Knu97] Donald E. Knuth. *The art of computer programming. Vol. 2 : Seminumerical algorithms 3rd ed.* Addison-Wesley Series in Computer Science et Information Processing, 1997 (cf. p. 45).

[KO63] Anatolii Alexevich Karatsuba et Yu Ofman. « Multiplication of Multidigit Numbers on Automata ». In : *Soviet Physics Doklady* 7 (1963), p. 595 (cf. p. 114, 121).

[Kob87] Neal I. Koblitz. « Elliptic curve cryptosystems ». In : *Mathematics of computation* (1987), p. 203–209. doi : 10.2307/2007884 (cf. p. 3, 11).

[Kob89] Neal I. Koblitz. « Hyperelliptic cryptosystems ». In : *Journal of Cryptology* 1.3 (1989), p. 139–150. doi : 10.1007/BF02252872 (cf. p. 3, 11).

[KSS08] Ezekiel Kachisa, Edward Schaefer et Michael Scott. « Constructing Brezing-Weng Pairing-Friendly Elliptic Curves Using Elements in the Cyclotomic Field ». In : *International Conference on Pairing-Based Cryptography – Pairing 2008*. Éd. : Steven D. Galbraith et Kenneth G. Paterson. Lecture Notes in Computer Science 5209. Springer Berlin / Heidelberg, 2008, p. 126–135. doi : 10.1007/978-3-540-85538-5_9 (cf. p. 55).

[Len87] Jr. Lenstra Hendrik W. « Factoring Integers with Elliptic Curves ». In : *Annals of Mathematics*. Second Series 126.3 (nov. 1987), p. 649–673. doi : 10.2307/1971363 (cf. p. 3).

[Ley] P. Leyland. *Cunningham numbers*. http://www.leyland.vispa.com/numth/factorization/cunningham/main.htm (cf. p. 53, 176).

[Lor96] Dino Lorenzini. *An Invitation to Arithmetic Geometry*. Graduate Studies in Mathematics Series. American Mathematical Society, 1996 (cf. p. 24).

[LR12] Reynald Lercier et Christophe Ritzenthaler. « Hyperelliptic curves and their invariants : Geometric, arithmetic and algorithmic aspects ». In : *Journal of Algebra* 372 (2012), p. 595–636. doi : 10.1016/j.jalgebra.2012.07.054 (cf. p. 43).

[McE78] Robert J. McEliece. « A Public-Key Cryptosystem Based On Algebraic Coding Theory ». In : *Deep Space Network Progress Report* 44 (jan. 1978), p. 114–116 (cf. p. 3).

[MH78] R. Merkle et M.E. Hellman. « Hiding information and signatures in trapdoor knapsacks ». In : *IEEE Transactions on Information Theory* 24.5 (1978), p. 525–530. DOI : 10.1109/TIT.1978.1055927 (cf. p. 3).

[Mil04] Victor S. Miller. « The Weil Pairing, and Its Efficient Calculation ». In : *Journal of Cryptology* 17.4 (2004), p. 235–261. DOI : 10.1007/s00145-004-0315-8 (cf. p. 31, 59).

[Mil86a] Victor S. Miller. « Short Programs for functions on Curves ». IBM, Thomas J. Watson Research Center. 1986. URL : http://crypto.stanford.edu/miller/miller.pdf (cf. p. 31, 59).

[Mil86b] Victor S. Miller. « Use of elliptic curves in cryptography ». In : *Advances in Cryptology – CRYPTO 1985*. Éd. : Hugh Williams. Lecture Notes in Computer Science 218. Springer Berlin / Heidelberg, 1986, p. 417–426. DOI : 10.1007/3-540-39799-X_31 (cf. p. 3).

[Mil86c] J.S. Milne. « Jacobian Varieties ». In : *Arithmetic Geometry*. Éd. : Gary Cornell et Joseph H. Silverman. Springer New York, 1986, p. 167–212. DOI : 10.1007/978-1-4613-8655-1_7 (cf. p. 18).

[MNT01] Atsuko Miyaji, Masaki Nakabayashi et Shunzou Takano. « New Explicit Conditions of Elliptic Curve Traces for FR-Reduction ». In : *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences* E84-A.5 (mai 2001), p. 1234–1243 (cf. p. 55).

[Mon05] Peter L. Montgomery. « Five, Six, and Seven-Term Karatsuba-Like Formulae ». In : *IEEE Transactions in Computers* 54.3 (mar. 2005), p. 362–369. DOI : 10.1109/TC.2005.49 (cf. p. 116, 119, 121, 129, 138, 177–179).

[MOV93] Alfred J. Menezes, Tatsuaki Okamoto et Scott A. Vanstone. « Reducing elliptic curve logarithms to logarithms in a finite field ». In : *IEEE Transactions on Information Theory* 39.5 (sept. 1993), p. 1639–1646. DOI : 10.1109/18.259647 (cf. p. 3, 44).

[Mul00] Thom Mulders. « On short multiplications and divisions ». In : *Applicable Algebra in Engineering, Communication and Computing* 11.1 (août 2000), p. 69–88. DOI : 10.1007/s002000000037 (cf. p. 126).

[NNS10] Michael Naehrig, Ruben Niederhagen et Peter Schwabe. « New Software Speed Records for Cryptographic Pairings ». In : *Progress in Cryptology – LATINCRYPT 2010*. Éd. : Michel Abdalla et Paulo S. L. M. Barreto. Lecture Notes in Computer Science 6212. Springer Berlin Heidelberg, 2010, p. 109–123. DOI : 10.1007/978-3-642-14712-8_7 (cf. p. 190).

[Ose08] Ivan Oseledets. « Optimal Karatsuba-like formulae for certain bilinear forms in GF(2) ». In : *Linear Algebra and its Applications* 429.8-9 (oct. 2008), p. 2052–2066. DOI : 10.1016/j.laa.2008.06.004 (cf. p. 122, 131, 140).

[OW99] Paul C. van Oorschot et Michael J. Wiener. « Parallel Collision Search with Cryptanalytic Applications ». In : *Journal of Cryptology* 12 (1 1999), p. 1–28. DOI : 10.1007/PL00003816 (cf. p. 45).

[Pat96] Jacques Patarin. « Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP) : Two New Families of Asymmetric Algorithms ». In : *Advances in Cryptology – EUROCRYPT'96*. Éd. : Ueli Maurer. Lecture Notes in Computer Science 1070. Springer Berlin Heidelberg, 1996, p. 33–48. DOI : 10.1007/3-540-68339-9_4 (cf. p. 3).

[PH11] David A. Patterson et John L. Hennessy. *Computer Organization and Design : The Hardware/Software Interface*. 4e éd. Morgan Kaufmann, déc. 2011 (cf. p. 145).

[PH78] S. Pohlig et M. Hellman. « An improved algorithm for computing logarithms over GF(p) and its cryptographic significance (Corresp.) » In : *IEEE Transactions on Information Theory* 24.1 (jan. 1978), p. 106–110. DOI : 10.1109/TIT.1978.1055817 (cf. p. 44).

[Pol78] John M. Pollard. « Monte Carlo methods for index computation (mod p) ». In : *Mathematics of computation* (1978), p. 918–924. DOI : 10.2307/2006496 (cf. p. 44).

[Reg09] Oded Regev. « On lattices, learning with errors, random linear codes, and cryptography ». In : *Journal of the ACM* 56.6 (sept. 2009), 34 :1–34 :40. DOI : 10.1145/1568318.1568324 (cf. p. 3).

[RML08] Francisco Rodríguez-Henríquez, Guillermo Morales-Luna et Julio López. « Low-Complexity Bit-Parallel Square Root Computation over GF(2^m) for All Trinomials ». In : *IEEE Transactions in Computers* 57.4 (2008), p. 472–480. DOI : 10.1109/TC.2007.70822 (cf. p. 110).

[Ron+07] Robert Ronan, Colm Ó hÉigeartaigh, Colin Murphy, Michael Scott et Tim Kerins. « Hardware acceleration of the Tate pairing on a genus 2 hyperelliptic curve ». In : *Journal of Systems Architecture* 53.2-3 (2007), p. 85–98. DOI : 10.1016/j.sysarc.2006.09.003 (cf. p. 185, 186).

[RS02] Karl Rubin et Alice Silverberg. « Supersingular Abelian Varieties in Cryptology ». In : *Advances in Cryptology – CRYPTO 2002*. Éd. : Moti Yung. Lecture Notes in Computer Science 2442. Springer, 2002, p. 336–353. DOI : 10.1007/3-540-45708-9_22 (cf. p. 52, 175).

[RS09] Karl Rubin et Alice Silverberg. « Using Abelian Varieties to Improve Pairing-Based Cryptography ». In : *Journal of Cryptology* 22.3 (2009), p. 330–364. DOI : 10.1007/s00145-008-9022-1 (cf. p. 175).

[RSA78] Ronald L. Rivest, Adi Shamir et Leonard M. Adleman. « A method for obtaining digital signatures and public-key cryptosystems ». In : *Communications of the ACM* 21.2 (fév. 1978), p. 120–126. DOI : 10.1145/359340.359342 (cf. p. 2).

[Sch93] Oliver Schirokauer. « Discrete Logarithms and Local Units ». In : *Philosophical Transactions of the Royal Society of London. Series A : Physical and Engineering Sciences* 345.1676 (1993), p. 409–423. DOI : 10.1098/rsta.1993.0139 (cf. p. 49).

[Sem98] Igor A. Semaev. « Evaluation of Discrete Logarithms in a Group of p-Torsion Points of an Elliptic Curve in Characteristic p ». In : *Mathematics of computation* 67.221 (1998), p. 353–356. DOI : 10.1090/S0025-5718-98-00887-4 (cf. p. 44).

[Sha85] Adi Shamir. « Identity-based cryptosystems and signature schemes ». In : *Advances in Cryptology – CRYPTO 1984*. Éd. : George Blakley et David Chaum. Lecture Notes in Computer Science 196. Springer Berlin / Heidelberg, 1985, p. 47–53. DOI : 10.1007/3-540-39568-7_5 (cf. p. 3).

[Sho97] Victor Shoup. « Lower bounds for discrete logarithms and related problems ». In : Lecture Notes in Computer Science 1233. Konstanz, Germany : Springer-Verlag, 1997, p. 256–266. DOI : 10.1007/3-540-69053-0_18 (cf. p. 43).

[Sil86] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*. Springer Verlag, 1986. DOI : 10.1007/978-0-387-09494-6 (cf. p. 12, 25, 32, 46, 51, 82, 92).

[SKG09] Chang Shu, Soonhak Kwon et Kris Gaj. « Reconfigurable Computing Approach for Tate Pairing Cryptosystems over Binary Fields ». In : *IEEE Transactions in Computers* 58.9 (sept. 2009), p. 1221–1237. DOI : 10.1109/TC.2009.64 (cf. p. 186).

[Sma99] Nigel P. Smart. « The Discrete Logarithm Problem on Elliptic Curves of Trace One ». In : *Journal of Cryptology* 12 (3 1999), p. 193–196. DOI : 10.1007/s001459900052 (cf. p. 44).

[SS71] Arnold Schönhage et Volker Strassen. « Schnelle Multiplikation großer Zahlen ». In : *Computing* 7 (3 1971), p. 281–292. DOI : 10.1007/BF02242355 (cf. p. 114).

[SS99] Nigel P. Smart et Samir Siksek. « A Fast Diffie—Hellman Protocol in Genus 2 ». In : *Journal of Cryptology* 12.1 (1999), p. 67–73. DOI : 10.1007/PL00003818 (cf. p. 3).

[Str69] Volker Strassen. « Gaussian elimination is not optimal ». In : *Numerische Mathematik* 13.4 (1969), p. 354–356. DOI : 10.1007/BF02165411 (cf. p. 126).

[Tat66] John T. Tate. « Endomorphisms of abelian varieties over finite fields ». In : *Inventiones Mathematicae* 2.2 (1966), p. 134–144. DOI : 10.1007/BF01404549 (cf. p. 31, 32).

[Thé03] Nicolas Thériault. « Index Calculus Attack for Hyperelliptic Curves of Small Genus ». In : *Advances in Cryptology - ASIACRYPT 2003*. Éd. : Chi-Sung Laih. Lecture Notes in Computer Science 2894. Springer Berlin Heidelberg, 2003, p. 75–92. DOI : 10.1007/978-3-540-40061-5_5 (cf. p. 47, 48).

[Too63] Andrei L. Toom. « The complexity of a scheme of functional elements realizing the multiplication of integers ». In : *Soviet Mathematics Doklady* 3 (1963), p. 714–716 (cf. p. 116).

[Ver01] Eric R. Verheul. « Evidence that XTR Is More Secure than Supersingular Elliptic Curve Cryptosystems ». In : *Advances in Cryptology – EUROCRYPT 2001*. Éd. : Birgit Pfitzmann. Lecture Notes in Computer Science 2045. Springer Berlin / Heidelberg, 2001, p. 195–210. DOI : 10.1007/3-540-44987-6_13 (cf. p. 41).

[Ver04] Eric R. Verheul. « Evidence that XTR is more secure than supersingular elliptic curve cryptosystems ». In : *Journal of Cryptology* 17.4 (sept. 2004), p. 277–296. DOI : 10.1007/s00145-004-0313-x (cf. p. 42).

[Ver10] Frederik VERCAUTEREN. « Optimal pairings ». In : *IEEE Transactions on Information Theory* 56.1 (jan. 2010), p. 455–461. DOI : 10.1109/TIT.2009.2034881 (cf. p. 69, 76, 80, 101).

[Vit12] Vanessa VITSE. « Attaques algébriques du problème du logarithme discret sur courbe elliptique ». Thèse de doct. Université Versailles Saint-Quentin en Yvelines, oct. 2012. URL : http://tel.archives-ouvertes.fr/tel-00655714 (cf. p. 48).

[Wag] S. WAGSTAFF. *The Cunningham Project*. http://homes.cerias.purdue.edu/~ssw/cun/index.html (cf. p. 53, 176).

[Was08] Lawrence C. WASHINGTON. *Elliptic Curves : Number Theory and Cryptography*. Chapman & Hall/CRC Press, 2008 (cf. p. 24).

[Wei40] André WEIL. « Sur les fonctions algébriques à corps de constantes fini ». In : *Comptes-rendu de l'Académie des Sciences de Paris* 210 (1940), p. 592–594 (cf. p. 31, 32).

[Win71] Shmuel WINOGRAD. « On multiplication of 2×2 matrices ». In : *Linear Algebra and its Applications* 4.4 (1971), p. 381–388. DOI : 10.1016/0024-3795(71)90009-7 (cf. p. 126).

[Win80] Shmuel WINOGRAD. *Arithmetic Complexity of Computations*. SIAM, 1980. DOI : 10.1137/1.9781611970364.fm (cf. p. 117).

[WZ99] Michael J. WIENER et Robert ZUCCHERATO. « Faster Attacks on Elliptic Curve Cryptosystems ». In : *Selected Areas in Cryptography – SAC 1998*. Éd. : Stafford E. TAVARES et Henk MEIJER. Lecture Notes in Computer Science 1556. Springer, 1999, p. 631–631. DOI : 10.1007/3-540-48892-8_15 (cf. p. 46).

[Yao+13] Gavin Xiaoxu YAO, Junfeng FAN, Ray C. C. CHEUNG et Ingrid VERBAUWHEDE. « Faster Pairing Coprocessor Architecture ». In : *International Conference on Pairing-Based Cryptography – Pairing 2012*. Éd. : Michel ABDALLA et Tanja LANGE. Lecture Notes in Computer Science 7708. Springer Berlin Heidelberg, 2013, p. 160–176. DOI : 10.1007/978-3-642-36334-4_10 (cf. p. 54, 188, 189).

Résumé : Les couplages sont des primitives cryptographiques qui interviennent désormais dans de nombreux protocoles. Dès lors, il est nécessaire de s'intéresser à leur calcul et à leur implémentation efficace. Pour ce faire, nous nous reposons sur une étude algorithmique et arithmétique de ces fonctions mathématiques.
Les couplages sont des applications bilinéaires définies sur des courbes algébriques, plus particulièrement, dans le cas qui nous intéresse, des courbes elliptiques et hyperelliptiques. Nous avons choisi de nous concentrer sur une sous-famille de celles-ci : les courbes supersingulières dont les propriétés permettent d'obtenir à la fois des couplages symétriques et des algorithmes efficaces pour leur calcul.
Nous décrivons alors une approche unifiée permettant d'établir une large variété d'algorithmes calculant des couplages. Nous l'appliquons notamment à la construction d'un nouvel algorithme pour le calcul de couplages sur des courbes supersingulières de genre 2 et de caractéristique 2.
Les calculs nécessaires aux couplages que nous décrivons s'appuient sur l'implémentation d'une arithmétique rapide pour les corps finis de petite caractéristique : la multiplication est l'opération critique qu'il convient d'optimiser. Nous présentons donc un algorithme de recherche exhaustive de formules de multiplication.
Enfin, nous appliquons toutes les méthodes précédentes à la conception et l'implémentation de différents accélérateurs matériels pour le calcul de couplages sur différentes courbes dont les architectures ont été optimisées soit pour leur rapidité, soit pour leur compacité.

Abstract: Pairings are cryptographic primitives which are now used in numerous protocols. Computing and implementing them efficiently is then an interesting challenge relying on an algorithmic and arithmetic study of those mathematical functions.
More precisely, pairings are bilinear maps defined over elliptic and hyperelliptic curves. Among those, we restrict our study to supersingular curves, as they allow both symmetric pairings and efficient algorithm for pairing computation.
We propose an unified framework for the construction of algorithms computing pairings and we apply it to the design of a novel algorithm for a pairing over a genus-2 characteristic-2 hyperelliptic curve.
The computations involved in our algorithms require the implementation of rapid arithmetic for finite fields of small characteristic. Since multiplication is the critical operation, we present an algorithm for the exhaustive search of multiplication formulae.
Finally, we apply all the previous methods to the design and implementation of different hardware accelerators for the computation of cryptographic pairings over various curves.