École Normale Supérieure de Lyon

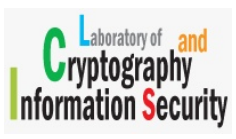
Master d'Informatique 1
Rapport de stage

Cryptographie à base de couplages et multiplieurs parallèles en caractéristiques 2 et 3

Nicolas Estibals

Encadrant : Jean-Luc Beuchat

Juin - août 2008



Laboratory of Cryptography and Information Security
University of Tsukuba

Abstract

This paper report the internship I have made with Jean-Luc Beuchat in the Laboratory of Cryptography and Information Security (LCIS) at the University of Tsukuba, Japan. We studied some parallel algorithms for multiplication over finite fields of characteristic two and three as a part of pairing computation for cryptography. We present an hardware implementation of pipelined multipliers on FPGA using some of those algorithms. Then we concluded that fully parallel designs contrary to more classical and sequential methods give better timings and use not so much area given that we improve the speed-per-area factor.

Keywords : parallel multiplier, finite fields arithmetic, pairing cryptography, hardware implementation, FPGA.

Résumé

Ce rapport rend compte du stage au LCIS (Laboratory of Cryptography and Information Security) à l'université de Tsukuba au Japon que j'ai effectué sous la direction de Jean-Luc Beuchat durant l'été 2008. Dans le cadre de la cryptographie basée sur les couplages, nous y proposons une étude de quelques algorithmes de multiplications parallèles sur les corps finis de caractéristiques deux et trois, ainsi qu'une implémentation matérielle d'opérateur de multiplication pipeliné sur FPGA utilisant certains de ces algorithmes. Ainsi nous pourrons constater que, outre le gain de performance important qu'apporte l'approche parallèle par rapport à la plus classique approche parallèle-série, nous obtenons des opérateurs de taille raisonnable étant donné que nous avons amélioré le produit temps-surface.

Mots-clefs : multilplieur parallèle, arithmétique des corps finis, cryptographie à base de couplage, implémentation matériel, FPGA.

Remerciements

Je tiens à remercier en tout premier lieu Jean-Luc Beuchat, mon encadrant, pour sa disponibilité et les échanges fructueux qui m'ont permis d'avancer dans ce travail. Je le remercie également, ainsi que Thibault Hilaire et Naoko Minobe, pour tous les bons moments passés en leur compagnie au Japon.

Je remercie tous les membres du LCIS pour leur accueil très chaleureux, et plus particulièrement Eiji Okamoto.

Je tiens à remercier Nicolas Brisebarre qui m'a mis en contact avec Jean-Luc, ainsi que tous ceux qui m'ont aidé dans la recherche de ce stage.

Je remercie aussi toutes les personnes qui ont rendu ce stage possible tant à l'ÉNS et à l'université Claude Bernard qu'au LCIS : Jan Matas, Bérénice Gagne, Yévédo Jeanne Goukindji et tous ceux que j'oublie et qui m'ont aidé dans les démarches administratives.

Merci à Jérémie Detrey à qui je dois les figures 2.1 et 2.2.

Merci aussi à Florent De Dinechin qui a mis à disposition toutes les ressources nécessaires à la synthèse des circuits présentés dans ce rapport.

Merci encore à Jean-Luc et à Nicolas pour leur patience et leurs nombreux conseils durant la relecture de ce rapport.

Table des matières

Chapitre 1

Introduction

Ces dernières décennies ont vu apparaître de nouveaux types de cryptosystèmes basés sur les courbes elliptiques. Plus récemment, les couplages sur ces mêmes courbes ont permis le développement de nombreux protocoles cryptographiques. Le calcul de ces couplages est une opération non triviale et exigeante en ressources, de plus les courbes utilisées sont définies sur des corps finis, ce qui nous oblige à faire appel à une arithmétique modulaire. La spécificité de cette arithmétique empêche d'obtenir des implémentations logicielles performantes du fait que les processeurs généraux ne soient pas adaptés à ce type de calculs. Ainsi la conception de matériel approprié à ces calculs se révèle nécessaire et demande des investigations sur les opérateurs matériels pour des corps finis. Plus spécifiquement, la multiplication sur corps finis de caractéristique 2 ou 3 ($\mathbb{F}_{p^m}$ avec $p = 2$ ou 3) se trouve être une opération critique dans les calculs de couplage sur les courbes elliptiques.

1.1 État de l'art

Les courbes elliptiques et leurs applications cryptographiques sont maintenant déjà bien connues, aussi il existe de nombreux livres permettant de se familiariser avec le domaine, citons par exemple celui de L. C. Washington [38]. Néanmoins le lecteur curieux des fondements mathématiques des courbes elliptiques cherchera de plus amples détails dans celui de J. H. Silverman [33].

Les couplages ont été introduits en cryptographie par A. Menezes *et al.* [26] puis dans [18] par G. Frey et H. Rück comme une attaque contre le problème du logarithme discret (DLP) sur les courbes elliptiques ; cela a, par la suite, ouvert un large champ applicatif en cryptologie, avec notamment un protocole de chiffrement basé sur l'identité [7], un protocole d'échange de clefs tripartie [22] ou encore un mécanisme de signature courte [8]. En effet les couplages permettent, en substance, de rassembler deux clefs en une seule sans qu'il soit possible de retrouver les deux premières clefs ; c'est cette propriété essentielle qui permet de créer des protocoles cryptographiques originaux et efficaces à base de couplages. Il est à noter que l'utilisation des courbes elliptiques est largement motivée par la réduction de la taille des clefs, en effet une clef de 224 bits correspond à la sécurité attendue d'une clef RSA de 2048 bits [20, p. 19], [31, p. 27].

Plus formellement, soient deux groupes G_1 et G_2, un couplage est une application bilinéaire de $G_1 \times G_1$ dans G_2. Weil(1940) [39] et Tate(1966) [35] ont proposé deux couplages sur le groupe des points d'une courbe elliptique ; cependant il a fallu attendre Miller(1986) pour connaître un algorithme efficace de calcul d'un couplage [27, 28]. De nombreuses améliorations ont été apportées par la suite à cet algorithme itératif [1, 12, 19]. Celui-ci demande un certain nombre de multiplications sur un corps fini et fait ainsi de la multiplication l'un des goulots d'étranglement du calcul de couplages.

La multiplication sur $\mathbb{F}_{p^m}$ peut être vue comme une multiplication de polynômes suivie d'une réduction modulaire. Mastrovito a tenté une approche combinant ces deux aspects [25, 30] dont nous discuterons à la Section 3.4 (p. 9). Le premier algorithme permettant de calculer un produit en temps subquadratique a été donné par A. Karatsuba en 1962 [23] et sa complexité fut améliorée un an plus tard par A. Toom [36]. L'étude de Bernstein permet d'avoir une bonne vue d'ensemble des différents algorithmes de multiplications [2]. Les récentes applications en cryptographie ont permis de nombreux développements algorithmiques sur le produit dans $\mathbb{F}_{2^m}$ et, dans une moindre mesure, $\mathbb{F}_{3^m}$ [32, 40, 15, 29, 14, 5]. Cependant peu d'implémentations matérielles sont connues. Citons tout de même la thèse de J. Shokrollahi [32] qui explore différentes variations de Karatsuba et l'article de J-L. Beuchat *et al.* [4] (meilleure implémentation connue à ce jour sur le compromis temps-surface).

1.2 Contribution

Dans le contexte de l'implémentation de couplage en matériel, la littérature reporte peu de multiplieurs parallèles au profit des multiplieurs parallèles-séries [34], parce qu'ils utilisent beaucoup moins de surface. Cependant, dans le but d'obtenir des performances accrues, il peut être intéressant d'exploiter au mieux le parallélisme existant dans la multiplication. Une implémentation pour $\mathbb{F}_{3^{97}}$ a été décrite [11]. Le but ici est de décrire un générateur de multiplieurs parallèles et pipelinés optimisant les performances ; de plus nous proposerons une implémentation générique qui ne dépend ni de la caractéristique (2 ou 3), ni de la taille des clefs (typiquement entre 239 et 459 bits en caractéristique 2 et entre 97 et 193 trits en caractéristique 3). Enfin il sera agréable de constater que malgré le surcoût en surface dû à la parallélisation du multiplieur, le compromis temps-surface pour le circuit complet calculant le couplage semble[1] avoir été amélioré avec cette approche.

1.3 Contenu

Nous commencerons par détailler un peu plus la notion de couplage, ses utilisations en cryptographie, l'algorithme et le circuit que nous créons pour calculer un couplage dans le Chapitre 2. Après s'être convaincu de l'utilité de s'intéresser à la multiplication sur corps fini dans ce contexte, nous présenterons des éléments d'arithmétique des corps finis et nous étudierons les différents algorithmes parallèles qui permettent une telle multiplication au Chapitre 3. Enfin nous présenterons notre implémentation d'opérateur matériel de multiplication et ses performances dans le Chapitre 4.

[1] À la date où nous écrivons ce rapport, nous n'avons pas encore de résultats complets concernant le coprocesseur de calcul de couplages.

Chapitre 2

Couplage : calcul et applications

2.1 Définition d'un couplage

La cryptographie utilise les couplages depuis que Menezes *et al.* les ont introduits comme une attaque possible contre le problème de logarithme discret (DLP, *Discrete Logarithm Problem*, Déf 2.1.2) sur les courbes elliptiques en 1993. Les couplages sont une notion mathématique généralisant en quelque sorte les produits scalaire. Nous utilisons en cryptographie une définition plus restreinte.

Définition 2.1.1 (Couplage)**.** Un couplage est une application bilinéaire non dégénérée $\hat{e} : G_1 \times G_1 \to G_2$ où G_1 et G_2 sont des groupes cycliques de même cardinal :

$$\forall P, Q \in G_1, \forall a, b \in \mathbb{Z}^*, \hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$$

$$\forall P \in G_1^*, \hat{e}(P, P) \neq 1$$

Nous notons G_1 de façon additive et G_2 de façon multiplicative.

Dans notre cas, nous utilisons le groupe des points sur une courbe elliptique comme groupe de base G_1. Les deux couplages les plus connus sur ce groupe sont ceux de Weil et de Tate, il est possible de montrer qu'inverser de telles fonctions est un problème *dur*, c'est-à-dire aussi dur que le DLP sur le groupe des points d'une courbe elliptique. Ce problème, considéré comme dur, sert de référence pour éprouver les protocoles cryptographiques au même titre que la factorisation des grands entiers.

Définition 2.1.2 (*Discrete Logarithm Problem*)**.** Le problème du logarithme discret sur un groupe G est de retrouver $a \in \mathbb{Z}$ étant donné seulement $P \in G$ et le produit aP.

Ainsi il est possible de calculer le couplage de deux points *facilement* (Miller [27]) mais il est *difficile* d'inverser ce calcul, *i.e.* de trouver la solution du problème suivant :

$$\text{Calculer } \hat{e}(P, P)^{abc} \text{ étant donnés } P, aP, bP, cP.$$

2.2 Un exemple d'application : l'échange de clef tripartie

La suite de nos travaux consistera à calculer le mieux possible un tel couplage. Nous donnons maintenant l'exemple d'un protocole d'échange de clef tripartie pour illustrer leur intérêt. Afin d'échanger des données cryptées, Alice et Bob doivent convenir d'une clef sans jamais la communiquer, pour cela ils choisissent chacun un nombre au hasard (a et b) qu'ils multiplient à une clef publique (P), s'échangent les résultats et les multiplient par leur nombre aléatoire secret. Ainsi à la fin du protocole, ils obtiennent chacun une clef secrète (abP) sans qu'il soit possible de la retrouver en analysant les communications entre Alice et Bob (Fig. 2.1, p.4).

A. Joux [22] a proposé un protocole d'échange de clefs entre 3 personnes utilisant un couplage basé sur le même principe. Alice, Bob et Charlie tirent chacun un nombre aléatoire (a, b et c), et envoient le produit de ce nombre et de la clef publique (P) aux deux autres. Chacun calcule alors le couplage des deux clefs qu'il reçoit et l'élève à la puissance du nombre qu'il a choisi. *In fine*, chacun obtient la même clef secrète $\hat{e}(P, P)^{abc}$, ceci est assuré par la propriété essentielle de bilinéarité des couplages. La figure 2.2 récapitule ce protocole.

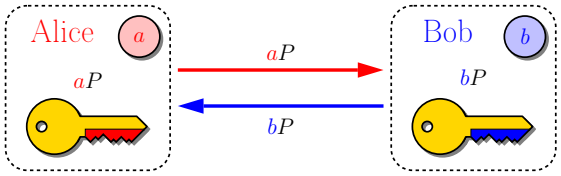


FIG. 2.1 – Échange de clefs avec le protocole de Diffie-Hellman

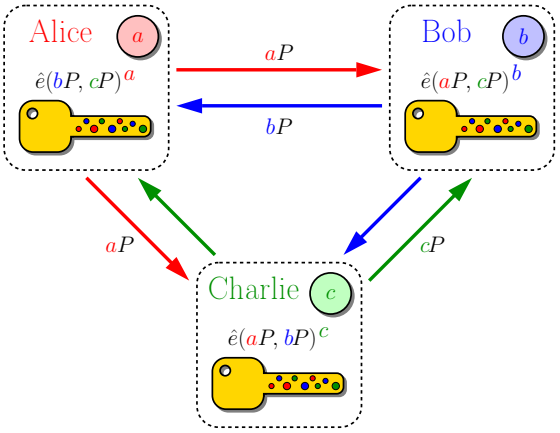


FIG. 2.2 – Échange de clefs tripartie

2.3 Calcul du couplage de Tate modifié

Notre but est ainsi de calculer un couplage, c'est-à-dire de choisir puis de calculer une fonction qui possède les propriétés de bilinéarité et de non-dégénérescence. Nous avons conservé les choix présentés dans l'article [4]. Nous avons ainsi choisi de calculer le couplage de Tate modifié η_T grâce à l'Algorithme 1. Cet algorithme consiste en une boucle où 7 multiplications sur $\mathbb{F}_{2^m}$ sont effectuées par itération ; une multiplication est effectuée à la ligne 11, les six autres étant contenues dans le produit sur $\mathbb{F}_{2^{4m}}$ de la ligne 14. Ce dernier peut se faire avec seulement 6 multiplications et 14 additions sur $\mathbb{F}_{2^m}$ grâce à un contexte particulier dont les détails peuvent être trouvés dans [4]. Ce sont ces multiplications qui nous intéresseront dans la suite. Précisons seulement pour l'instant que $\mathbb{F}_{2^m}$ et $\mathbb{F}_{3^m}$ sont les corps de Galois de caractéristiques 2 et 3, ils servent ici à représenter les points sur une courbe elliptique et nous les décrirons plus précisément à la Section 3.1.

Algorithme 1 Calcul du couplage η_T en caractéristique 2

Entrée: $P = (x_P, y_P)$, $Q = (x_Q, y_Q)$ avec $x_P, y_P, x_Q, y_Q \in \mathbb{F}_{2^m}$, $\alpha, \beta, \bar{\delta} \in \mathbb{F}_2$.
Sortie: $\eta_T(P, Q) \in \mathbb{F}^*_{2^{4m}}$.

$u, v, g_0, g_1, g_2 \in \mathbb{F}_{2^m}$,
$F, L, G \in \mathbb{F}_{2^{4m}}$,
t et s sont des symboles formels qui permettent de construire $F_{2^{4m}}$ à partir de $\mathbb{F}_{2^m}$.

1. $y_P \leftarrow y_P + \bar{\delta}$;
2. $u \leftarrow x_P + \alpha$; $v \leftarrow x_Q + \alpha$;
3. $g_0 \leftarrow \boldsymbol{u \cdot v} + y_P + y_Q + \beta$; (1 multiplication et 2 additions sur $\mathbb{F}_{2^m}$)
4. $g_1 \leftarrow u + x_Q$; $g_2 \leftarrow v + {x_P}^2$;
5. $G \leftarrow g_0 + g_1 s + t$;
6. $L \leftarrow (g_0 + g_2) + (g_1 + 1)s + t$;
7. $F \leftarrow \boldsymbol{L \cdot G}$; (2 multiplications et 5 additions sur $\mathbb{F}_{2^m}$)
8. **for** $j = 1$ to $\frac{m-1}{2}$ **do**
9. $x_P \leftarrow \sqrt{x_P}$; $y_P \leftarrow \sqrt{y_P}$; $x_Q \leftarrow {x_Q}^2$; $y_Q \leftarrow {y_Q}^2$;
10. $u \leftarrow x_P + \alpha$; $v \leftarrow x_Q + \alpha$;
11. $g_0 \leftarrow \boldsymbol{u \cdot v} + y_P + y_Q + \beta$; (1 multiplication et 2 additions sur $\mathbb{F}_{2^m}$)
12. $g_1 \leftarrow u + x_Q$;
13. $G \leftarrow g_0 + g_1 s + t$;
14. $F \leftarrow \boldsymbol{F \cdot G}$; (6 multiplications et 14 additions sur $\mathbb{F}_{2^m}$)
15. **end for**

2.4 Architecture du calcul de couplage

Nous avons conçu un coprocesseur réalisant l'algorithme précédent autour d'un multiplieur parallèle pipeliné. Un opérateur pipeliné se comporte de la manière suivante : il accepte un jeu d'entrée à chaque cycle d'horloge mais renvoie le résultat correspondant après quelques cycles de latence, ainsi il peut y avoir plusieurs jeux de données en cours de calcul dans l'opérateur qui sortiront les uns à la suite des autres à chaque

cycle d'horloge. Pipeliner un opérateur permet de rendre le circuit plus rapide. Nous ne décrivons ici (Fig. 2.3, p.5) que la partie du circuit correspondant aux lignes 11, 13 et 14 de l'Algorithme 1, le reste de l'algorithme se réalisant aisément dans des modules séparés sur le circuit, en parallèle de ce que nous allons décrire maintenant. Il y a sept multiplications à réaliser par itération de la boucle principale, il nous faut donc au moins sept cycles d'horloge par itération. Nous avons trouvé un ordonnancement de ces sept multiplications permettant de proposer un nouveau calcul au multiplieur et d'utiliser sa sortie à chaque cycle d'horloge. Il s'agit là d'une première difficulté, en effet il n'est pas évident que les sept cycles d'horloge permettent de réaliser tous les calculs de la boucle.

Enfin il nous a fallu dessiner un circuit le plus compact possible permettant de calculer les sommes et apportant les bonnes valeurs aux entrées du multiplieur. Ce circuit est présenté à la Figure 2.3. Le circuit se compose : du multiplieur suivi d'un additionneur quatre entrées permettant de réaliser l'essentiel des opérations, de deux petits modules aux entrées du multiplieur permettant d'ajuster celles-ci et de collecter les résultats des modules non-présentés ici.

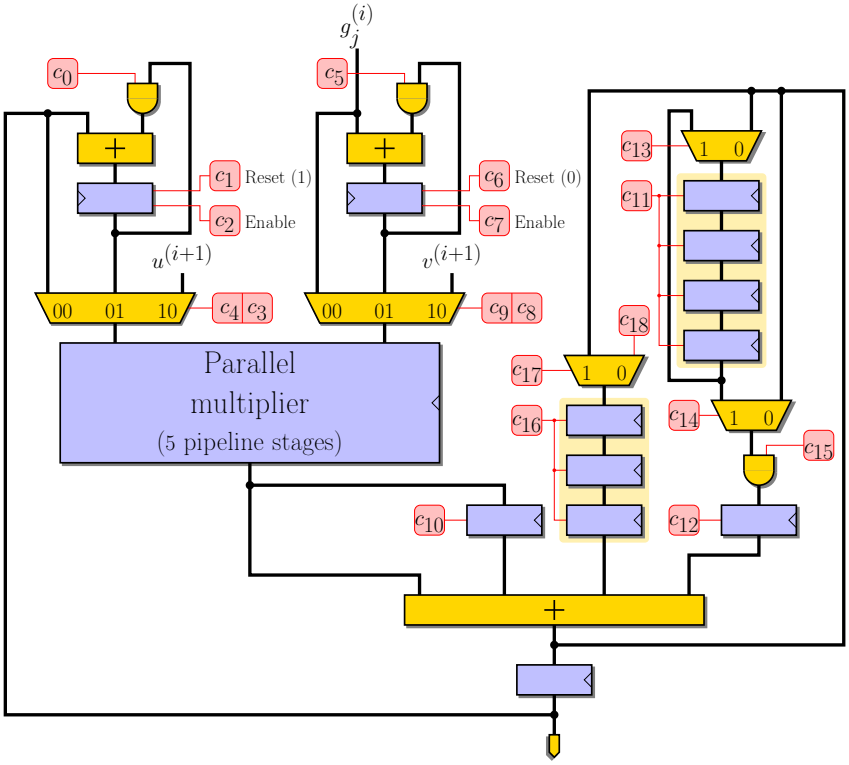


Fig. 2.3 – Architecture du coprocesseur de calcul de couplage en caractéristique 2. Vue partielle : seuls les circuits correspondant aux lignes 11, 13 et 14 de l'algorithme précédent ont été représentés sur le schéma.

Notez que nous avons présenté ici uniquement ce qui concernait la caractéristique 2, un travail similaire a été réalisé en caractéristique 3. Il ne s'agit ici que de se convaincre de l'intérêt d'une architecture parallèle pipelinée pour réaliser ce type d'algorithme itératif.

Chapitre 3

Algorithmes de multiplication sur $\mathbb{F}_{p^m}$

Nous avons maintenant une idée de la place que prend la multiplication sur un corps fini dans un calcul de couplage. Nous allons à présent étudier plus précisément l'arithmétique sur les corps finis et plus particulièrement les algorithmes de multiplication sur ces corps.

3.1 Éléments d'arithmétique des corps finis

3.1.1 Rappels

Commençons ce chapitre par quelques rappels sur les corps finis, ou corps de Galois. Un corps est une structure algébrique où il est possible d'effectuer les quatre opérations élémentaires : $+$, $-$, $\times$, $\div$ (Déf. 3.1.1).

Définition 3.1.1 (Corps)**.** Un corps $(K, +, \times)$ est un ensemble muni de deux lois internes $+$ et $\times$ qui vérifie :
- $(K, +)$ est un groupe commutatif,
- $(K \setminus \{0\}, \times)$ est un groupe,
- $\times$ est distributif sur $+$.

Il est à noter que la définition ne contient pas la commutativité de la multiplication, cependant, un théorème dû à Wedderburn permet d'affirmer que tout corps fini est commutatif. Imposer la finitude d'un corps contraint d'ailleurs fortement la structure de corps (Th. 3.1.4).

Définition 3.1.2 (Caractéristique)**.** La caractéristique d'un corps est le plus petit entier strictement positif k tel que $k * \mathbb{1}$ soit nul s'il existe et 0 sinon.

Lemme 3.1.3. *La caractéristique d'un corps est soit un premier soit* 0*.*

Théorème 3.1.4. *L'ordre ou cardinalité d'un corps fini est un primaire (puissance d'un nombre premier). Ce nombre premier est alors la caractéristique du corps.*

Le principal corollaire du théorème 3.1.4 est que l'ordre d'un corps fini peut toujours s'écrire sous la forme p^m avec p premier et $m \in \mathbb{N}^*$. De plus l'ordre d'un corps fini le définit totalement à un isomorphisme près. Nous procéderons donc à un abus de notation en considérant $\mathbb{F}_q$ comme *le* corps d'ordre q.

3.1.2 Construire $\mathbb{F}_{p^m}$

Les corps finis les plus simples sont les corps premiers $\mathbb{Z}/p\mathbb{Z}$ (avec p premier), les autres sont construits par extension algébrique. Nous cherchons donc à construire les $\mathbb{F}_{p^m}$ à partir du corps des entiers modulo p. Il suffit pour cela de choisir un polynôme irréductible F sur $\mathbb{F}_p$ de degré m et de considérer les classes d'équivalence des polynômes sur $\mathbb{F}_p$ modulo F :

$$\mathbb{F}_{p^m} \approx \mathbb{F}_p[X]/F\mathbb{F}_p[X] \text{ avec } F \in \mathbb{F}_p[X] \text{ irréductible} \tag{3.1}$$

Les extensions $\mathbb{F}_{p^m}$ existent pour tout $m \in \mathbb{N}^*$. En effet il existe au moins un polynôme irréductible sur $\mathbb{F}_p$ de degré m pout tout m positif, c'est une conséquence du lemme 14.38 du livre *Modern Computer Algebra* [37, p. 398] qui borne le nombre de polynômes irréductibles de degré m sur $\mathbb{F}_p$.

Le but ici n'est pas d'exposer les théories de Galois, ni même d'en donner un aperçu mais seulement d'aboutir à la construction donnée en 3.1 des corps finis. En effet, nous utiliserons par la suite des polynômes sur $\mathbb{F}_p$ pour représenter les éléments de $\mathbb{F}_{p^m}$. Une fois F choisi, il y a une bijection entre les polynômes de

degré $m-1$ au plus et $\mathbb{F}_{p^m}$; ainsi nous représenterons $\mathbb{F}_{p^m}$ par ces polynômes, c'est-à-dire une suite finie de m éléments de $\mathbb{F}_p$.

Conscients qu'il n'y a dans ces rappels qu'un aperçu très limité d'algèbre, nous invitons le lecteur à se référer au livre de S. Lang [24] et à celui de J. von Zur Gathen et J. Gerhard [37] pour plus de détails.

3.2 Réduction modulaire

Puisque nous représentons les éléments de $\mathbb{F}_{p^m}$ par des polynômes, les opérations algébriques peuvent toutes être faites sur $\mathbb{F}_p[X]$. Cependant pour garder un représentant unique, il faut, et il suffit, de ne prendre que les polynômes de degré strictement inférieur à m en effectuant au besoin une réduction modulo F (le polynôme irréductible qui nous a permis d'étendre $\mathbb{F}_p$ à $\mathbb{F}_{p^m}$).

Dès lors l'addition ne pose pas de problème, puisque la somme de deux polynômes de degré $m-1$ au plus est un polynôme de degré au plus $m-1$. Cependant, le cas de la multiplication est différent, puisque le produit de deux tels polynômes donne un polynôme de degré $2m-2$. Il faut alors effectuer une réduction modulaire.

3.2.1 Matrice de réduction modulaire

Comme nous travaillons en caractéristique 2 ou 3, nous pouvons écrire notre polynôme irréductible F sous la forme (c'est le cas pour tout polynôme de degré m sur $\mathbb{F}_2$ ou $\mathbb{F}_3$ à une multiplication scalaire près) :

$$F = -X^m + \sum_{i=0}^{m-1} f_i X^i, f_i \in \mathbb{F}_p. \tag{3.2}$$

Nous définissons alors $\tilde{F} = (\tilde{f}_{i,j})_{0 \leqslant i \leqslant m-1, 0 \leqslant j \leqslant m-2}$ la matrice de réduction modulaire telle que :

$$X^{m+j} \mod F = \sum_{i=0}^{m-1} \tilde{f}_{i,j} X^i. \tag{3.3}$$

Cette matrice se calcule facilement par récurrence grâce aux formules suivantes :

$$\begin{cases} \forall i \in [\![0, m-1]\!], & \tilde{f}_{i,0} = f_i, \\ \forall j \in [\![1, m-1]\!], & \tilde{f}_{0,j} = \tilde{f}_{m-1,j-1} f_0, \\ \forall i, j \in [\![1, m-1]\!], & \tilde{f}_{i,j} = \tilde{f}_{i-1,j-1} + \tilde{f}_{m-1,j-1} f_i. \end{cases} \tag{3.4}$$

3.2.2 Réduction du produit

Multiplier deux polynômes A et B de degré $m-1$ donne un polynôme S de degré $2m-2$, qui se réduit essentiellement grâce à un produit matrice-vecteur avec $\tilde{F}$. Il peut être très pratique de noter les polynômes représentant les éléments de $\mathbb{F}_{p^m}$ sous la forme de vecteurs-colonnes ; nous ne nous en priverons pas par la suite afin d'utiliser le calcul matriciel dans un souci de concision des formules.

$$\begin{aligned} A \cdot B \mod F &= \sum_{i=0}^{m-1} s_i X^i + \sum_{i=0}^{m-2} s_{i+m} (X^{i+m} \mod F) \\ &= \sum_{i=0}^{m-1} s_i X^i + \sum_{j=0}^{m-2} s_{j+m} \sum_{i=0}^{m-1} \tilde{f}_{i,j} X^i \\ &= \sum_{i=0}^{m-1} s_i X^i + \sum_{i=0}^{m-1} (\sum_{j=0}^{m-2} \tilde{f}_{i,j} s_{j+m}) X^i \\ &= (s_i)_{0 \leqslant i \leqslant m-1} + \tilde{F} (s_{m+i})_{0 \leqslant i \leqslant m-2}. \end{aligned} \tag{3.5}$$

En choisissant correctement le polynôme F (*id est* en prenant F le plus creux possible), le produit matrice-vecteur se ramène à quelques additions et décalages. Le choix du polynôme irréductible est critique à plusieurs étapes du calcul d'un couplage. Heureusement, il s'agit dans la majorité des cas de trouver le polynôme le plus *simple* possible. Dans le cas de la multiplication, la notion de simplicité correspond à un polynôme le plus creux possible, le cas idéal étant un trinôme, *ie.* de la forme $-X^m \pm X^k \pm 1$ avec k le plus petit possible, quand il existe. Nous ne discuterons pas ici des différents compromis que recouvre le choix d'un tel polynôme.

3.2.3 Exemple de réduction

Pour notre exemple, nous travaillerons sur $\mathbb{F}_{3^6}$ avec $F = -X^6 - X + 1$[1]. Notez que par abus de notation, nous utilisons les entiers représentant les classes de $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ plutôt que les classes d'équivalence elles-mêmes. Nous avons alors :

$$\tilde{F} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 2 \end{pmatrix}.$$

Supposons de plus que nous avons obtenu $S = 2X^{10} + 2X^9 + X^6 + X^4 + 2X^3 + 2X^2 + 2$ et utilisons la formule 3.5 :

$$A \cdot B \mod F = \begin{pmatrix} 2 \\ 0 \\ 2 \\ 2 \\ 1 \\ 0 \end{pmatrix} + \tilde{F} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 2 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \\ 2 \\ 2 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 2 \\ 0 \\ 2 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ 2 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

et finalement :

$$A \cdot B \mod F = X^5 + X^4 + X^3 + 2X^2 + 2X.$$

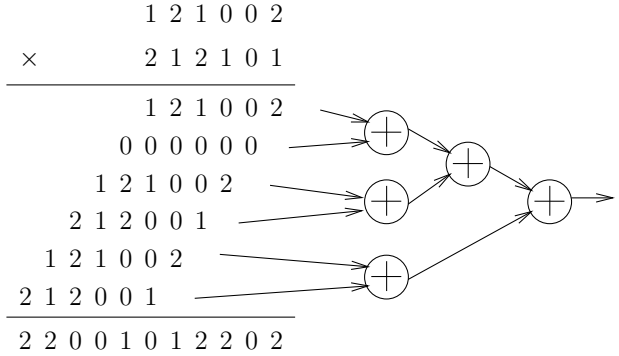
3.3 Multiplieur naïf

Nous allons ici décrire la façon la plus élémentaire d'effectuer la multiplication dans un système de numération à position. Il s'agit de la méthode que nous avons tous apprise à l'école primaire.

En ne retenant que l'essentiel de la méthode apprise sur les bancs de l'école (Fig. 3.1, p.8), multiplier revient à calculer les produits partiels, *ie.* les produits des chiffres deux à deux, et à en faire la somme (en respectant leur position). Notre cas est plus simple : nous travaillons avec des polynômes et non des entiers ce qui implique qu'il n'y a pas de retenue à propager lors de la somme. Nous voulons faire la multiplication sur $\mathbb{F}_{p^m}$, et ainsi il nous faut effectuer m^2 multiplications sur $\mathbb{F}_p$, puis m sommes de m éléments de $\mathbb{F}_p$ au plus. En utilisant un arbre équilibré d'addition il faudra alors $\Theta(m^2)$ additionneurs sur $\mathbb{F}_p$ et chaque produit partiel traverse au plus $O(\log m)$ additionneurs (*cf.* Fig. 3.2, p.8).

	1	2	3
×		1	3
	3	6	9
1	2	3	
1	5	9	9

Fig. 3.1 – Multiplions comme en primaire



$$(X^5 + 2X^4 + X^3 + 2) \times (2X^5 + X^4 + 2X^3 + X^2 + 1)$$
$$= 2X^{10} + 2X^9 + X^6 + X^4 + 2X^3 + 2X^2 + 2$$

Fig. 3.2 – Multiplication naïve sur $\mathbb{F}_{3^6}$

[1] Ce polynôme irréductible a été trouvé sur la page de F. Chabaud [9]

3.4 Produit de Mastrovito

Mastrovito [25] a proposé en 1989 de combiner la multiplication sur $\mathbb{F}_p[X]$ et la réduction. Reprenons l'équation 3.5 de réduction d'un produit et intégrons les produits partiels :

$$\begin{aligned} A \cdot B \mod F &= (s_i)_{0\leqslant i\leqslant m-1} + \tilde{F}(s_{m+i})_{0\leqslant i\leqslant m-2} \\ &= \left(\sum_{k=0}^{i} a_{i-k}b_k\right)_{0\leqslant i\leqslant m-1} + \tilde{F}\left(\sum_{k=i+1}^{m-1} a_k b_{m+i-k}\right)_{0\leqslant i\leqslant m-2} \\ &= (L(A) + \tilde{F}H(A))B \\ &= M(A)B \end{aligned} \tag{3.6}$$

avec :

$$L(A) = \begin{pmatrix} a_0 & 0 & \dots & 0 \\ a_1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ a_{m-1} & \dots & a_1 & a_0 \end{pmatrix}, H(A) = \begin{pmatrix} 0 & a_{m-1} & \dots & a_1 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & a_{m-1} \end{pmatrix}.$$

Nous pouvons alors constater que l'on peut effectuer le produit après la construction de $M(A)$. Construire $M(A)$ correspond à faire quelques additions sur les coefficients de A, une fois encore plus le polynôme irréductible est creux, moins il y a d'additions à faire. À titre d'exemple retenons simplement que si $F = -X^m \pm X^k \pm 1$ avec $k < m/2$ chaque élément de la matrice $M(A)$ est la somme de moins de quatre coefficients de A.

Ensuite il faut faire le produit matrice-vecteur avec B, ce qui va demander m^2 produits sur $\mathbb{F}_p$ puis m sommes de m éléments de $\mathbb{F}_p$, se qui peut se faire avec un arbre d'addition de profondeur $\Theta(\log m)$. Pour plus de détails quant à la complexité d'un tel multiplieur et sa dépendance à la forme de F, le lecteur pourra consulter l'article [30] de Reyhani-Masoleh et Hasan.

Reprenons notre exemple avec cette méthode :

$$\begin{aligned} A \cdot B \mod F &= (L(A) + \tilde{F}H(A))B \\ &= \left(\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 & 0 & 0 \\ 2 & 1 & 0 & 0 & 2 & 0 \\ 1 & 2 & 1 & 0 & 0 & 2 \end{pmatrix} + \tilde{F}\begin{pmatrix} 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}\right) B \\ &= \begin{pmatrix} 2 & 1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 1 & 0 \\ 0 & 0 & 1 & 2 & 1 & 1 \\ 1 & 0 & 0 & 1 & 2 & 1 \\ 1 & 1 & 0 & 0 & 1 & 2 \\ 2 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 2 \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ 2 \\ 1 \\ 1 \\ 1 \end{pmatrix}. \end{aligned}$$

3.5 Diviser pour régner

Les deux algorithmes de multiplications présentés jusqu'ici (3.3 et 3.4) sont des algorithmes avec une complexité quadratique, or il est possible d'effectuer la multiplication avec une complexité subquadratique en appliquant le paradigme *Diviser-pour-régner*.

3.5.1 Algorithme de Karatsuba-Offman

L'algorithme de Karatsuba-Offman [23] est le premier algorithme qui a permis d'avoir une complexité subquadratique pour calculer un produit. Bien qu'il ait été initialement proposé pour la multiplication de grands entiers, il s'adapte très bien aux polynômes. L'algorithme consiste à séparer les entrées en partie haute (A_H, B_H) et en partie basse (A_L, B_L), et à utiliser la formule (3.8) obtenue en développant le produit et en exploitant l'identité suivante :

$$(a + b')(a' + b) = (a + a')(b + b') - ab - a'b' \tag{3.7}$$

$$\begin{aligned} A \cdot B &= (A_L + X^{\lceil m/2 \rceil} A_H)(B_L + X^{\lceil m/2 \rceil} B_H) \\ &= A_L B_L + (A_H + B_L)(A_L + B_H) X^{\lceil m/2 \rceil} B_H) + A_H B_H X^{2\lceil m/2 \rceil} \\ &= A_L B_L + ((A_L + A_H)(B_L + B_H) - A_L B_L - A_H B_H) X^{\lceil m/2 \rceil} B_H) + A_H B_H X^{2\lceil m/2 \rceil} \end{aligned} \tag{3.8}$$

Ainsi pour effectuer le produit de deux polynômes de degré au plus $m-1$, il suffit de faire trois multiplications de taille $\lceil m/2 \rceil$ ($A_L B_L$, $(A_L + A_H)(B_L + B_H)$ et $A_H B_H$) et de faire quelques additions pour décomposer les opérandes et reconstruire le résultat. Ce procédé est généralisable : les facteurs peuvent être séparés en trois parties ou plus et l'identité 3.7 à nouveau utilisée. De même les multiplications de taille inférieure peuvent être calculées de la même façon en appelant récursivement la même méthode. Il est alors aisée de déduire les algorithmes que nous avons écrits dans le Tableau 3.1 sous la forme suivante : séparation des facteurs, appels récursifs et reconstruction du résultat.

Tab. 3.1 – Algorithmes de multiplication selon Karatsuba.

Algorithme	Séparation	Appels récursifs	Reconstruction
$\mathcal{K}_{2,m}$	$A \to A_L + X^{\lceil m/2 \rceil} A_H$ $B \to B_L + X^{\lceil m/2 \rceil} B_H$ $A_M \leftarrow A_H + A_L$ $B_M \leftarrow B_H + B_L$	$p_H \leftarrow A_H * B_H$ $p_M \leftarrow A_M * B_M$ $p_L \leftarrow A_L * B_L$	$S \leftarrow p_H X^{2\lceil m/2 \rceil}$ $+ (p_M - p_H - p_L) X^{\lceil m/2 \rceil}$ $+ p_L$
$\mathcal{K}_{3,m}$	$A \to A_0 + X^{\lceil m/3 \rceil} A_1 + X^{2\lceil m/3 \rceil} A_2$ $B \to B_0 + X^{\lceil m/3 \rceil} B_1 + X^{2\lceil m/3 \rceil} B_2$ $A_{S_0} \leftarrow A_1 + A_2$ $A_{S_1} \leftarrow A_0 + A_2$ $A_{S_2} \leftarrow A_1 + A_0$ $B_{S_0} \leftarrow B_1 + B_2$ $B_{S_1} \leftarrow B_0 + B_2$ $B_{S_2} \leftarrow B_1 + B_0$	$p_0 \leftarrow A_0 * B_0$ $p_1 \leftarrow A_1 * B_1$ $p_2 \leftarrow A_2 * B_2$ $p'_0 \leftarrow A_{S_0} * B_{S_0}$ $p'_1 \leftarrow A_{S_1} * B_{S_1}$ $p'_2 \leftarrow A_{S_2} * B_{S_2}$	$S \leftarrow p_2 X^{4\lceil m/3 \rceil}$ $+ (p'_0 - p_1 - p_2) X^{3\lceil m/3 \rceil}$ $+ (p'_1 - p_0 + p_1 - p_2) X^{2\lceil m/3 \rceil}$ $+ (p'_2 - p_1 - p_0) X^{\lceil m/3 \rceil}$ $+ p_0$

3.5.2 Élimination des recouvrements à la reconstruction du produit dans l'algorithme de Karatsuba

La reconstruction du résultat dans les algorithmes précédents demande un certain nombre d'additions, Ainsi, dans le cas de l'algorithme $\mathcal{K}_{2,m}$, la somme finale contient jusqu'à 5 termes non nuls (quand m est impair, 3 sinon) et demande $6\lceil m/2 \rceil$ sommes sur $\mathbb{F}_2$ Fig. 3.3, p.10).

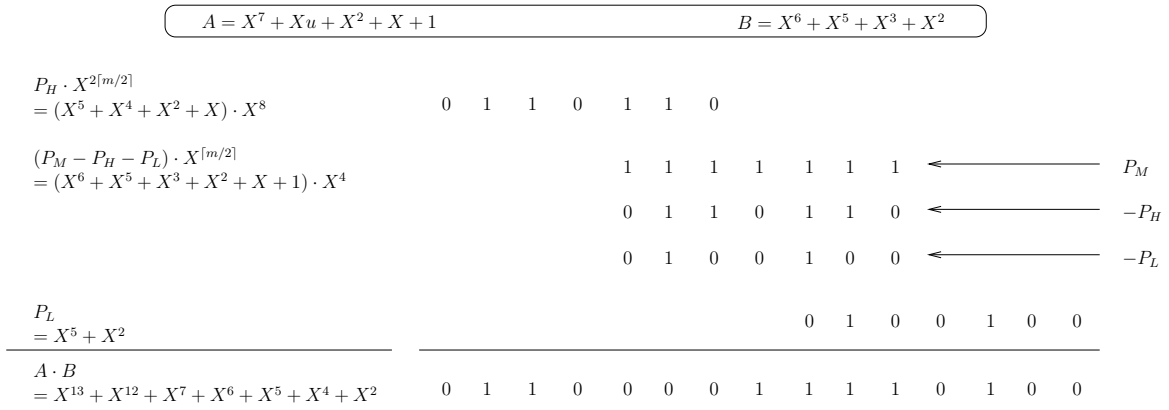


Fig. 3.3 – Illustration du recouvrement à la reconstruction dans l'algorithme $\mathcal{K}_{2,8}$.

Fan *et al.* [16] ont proposé de séparer les polynômes, non pas en parties haute et basse mais en parties paire (A_E, B_E) et impaire (A_O, B_O) dans l'algorithme de Karatsuba. Cette idée vient de G. Hanrot et P. Zimmermann dans un travail sur les produits courts de Mulder [21]. Nous avons étendu le précédent résultat de Fan à la caractéristique 3 et à l'algorithme $\mathcal{K}_{3,m}$ (*cf.* Tab. 3.2). Grâce à ceci, la somme finale a une profondeur inférieure à 3 et demande $5\lceil m/2 \rceil$ sommes sur $\mathbb{F}_p$. Pour $\mathcal{K}_{3,m}$, il faut séparer les termes des

polynômes selon leur index modulo 3, et ceci nous fait passer d'une somme de reconstruction de profondeur 10 et de taille $12\lceil m/3\rceil$ à une somme finale de profondeur 4 et de taille $9\lceil m/3\rceil$.

TAB. 3.2 – Algorithmes modifiées de multiplication selon Karatsuba.

Algorithme	Séparation	Appels récursifs	Reconstruction
$\mathcal{K}'_{2,(m)}$	$A \to A_E(X^2) + XA_O(X^2)$ $B \to B_E(X^2) + XB_O(X^2)$ $A_M \leftarrow A_O + A_E$ $B_M \leftarrow B_O + B_E$	$p_O \leftarrow A_O * B_O$ $p_M \leftarrow A_M * B_M$ $p_E \leftarrow A_E * B_E$	$S \leftarrow (p_E + Xp_O)(X^2)$ $+ X(p_M - p_E - p_O)(X^2)$
$\mathcal{K}'_{3,(m)}$	$A \to A_0(X^3)+XA_1(X^3)+X^2A_2(X^3)$ $B \to B_0(X^3)+XB_1(X^3)+X^2B_2(X^3)$ $A_{S_0} \leftarrow A_1 + A_2$ $A_{S_1} \leftarrow A_0 + A_2$ $A_{S_2} \leftarrow A_1 + A_0$ $B_{S_0} \leftarrow B_1 + B_2$ $B_{S_1} \leftarrow B_0 + B_2$ $B_{S_2} \leftarrow B_1 + B_0$	$p_0 \leftarrow A_0 * B_0$ $p_1 \leftarrow A_1 * B_1$ $p_2 \leftarrow A_2 * B_2$ $p'_0 \leftarrow A_{S_0} * B_{S_0}$ $p'_1 \leftarrow A_{S_1} * B_{S_1}$ $p'_2 \leftarrow A_{S_2} * B_{S_2}$	$S \leftarrow (p_0 + X(p'_0 - p_1 - p_2))(X^3)$ $+ X(p'_2 - p_0 - p_1 + Xp_2)(X^3)$ $+ X^2(p_1 + p'_1 - p_2 - p_0)(X^3)$

3.5.3 Algorithme de Toom-Cook

L'année suivant la découverte de l'algorithme de Karatsuba, A. L. Toom [36] proposa un autre algorithme de multiplication qui fut amélioré par T. Cook dans sa thèse [10]. Le principe de cet algorithme est d'évaluer le polynôme produit en suffisamment de points pour pouvoir le reconstruire par interpolation. Avec cet algorithme, il faut découper les facteurs en trois parties, évaluer le produit en 5 points (ce qui demande 5 multiplications de taille inférieure), et interpoler pour obtenir les coefficients C_i du résultat final.

$$\begin{aligned} A \cdot B &= (A_0 + A_1X^{\lceil m/3\rceil} + A_2X^{2\lceil m/3\rceil})(B_0 + B_1X^{\lceil m/3\rceil} + B_2X^{2\lceil m/3\rceil}) \\ &= C_0 + C_1X^{\lceil m/3\rceil} + C_2X^{2\lceil m/3\rceil} + C_3X^{3\lceil m/3\rceil} + C_4X^{4\lceil m/3\rceil} \end{aligned} \tag{3.9}$$

Il existe des variations qui découpe les polynômes en 5 ou 7 parties mais nous n'avons malheureusement pas la place de les détailler ici. Remarquons tout de suite qu'il suffit de faire ici 5 appels récursifs à des multiplications de taille $\lceil m/3\rceil$ contrairement à $\mathcal{K}_{3,m}$ où il en faut 6. Pour pouvoir appliquer cet algorithme, il nous faut 5 points où évaluer le polynôme de telle sorte que l'interpolation soit simple. Rappelons qu'il existe un théorème de l'unisolvance précisant qu'il n'existe qu'un seul polynôme de degré k au plus défini par un ensemble de $k+1$ points, ainsi le produit comme un polynôme de degré 4 prenant ses coefficients dans les polynômes de degré $\lceil m/3\rceil$ au plus sur $\mathbb{F}_p$ est une démarche correcte. Notez qu'il est donc important de considérer les polynômes A et B comme des polynômes de degré 3 de coefficients A_0, A_1, A_2 et B_0, B_1, B_2, ces coefficients étant eux-mêmes des polynômes. Pour ne pas confondre les deux types de polynômes, nous noterons dès lors Y l'indéterminée des polynômes A, B et C et nous conserverons X comme l'indéterminée pour leurs coefficients :

$$\begin{aligned} A \cdot B &= (A_0(X) + A_1(X) \cdot Y + A_2(X) \cdot Y^2)(B_0(X) + B_1(X)\dot{Y} + B_2(X) \cdot Y^2) \\ &= C_0(X) + C_1(X) \cdot Y + C_2(X) \cdot Y^2 + C_3(X)Y^3 + C_4(X)Y^4 \end{aligned} \tag{3.10}$$

Nous utilisons les points 0, 1 et -1 (si la caractéristique est différente de 2)[2], nous rajoutons aussi un point à l'infini afin d'interpoler immédiatement le coefficient de poids fort (plus formellement nous nous plaçons sur l'extension rationnelle du corps de base des polynômes : ici $\mathbb{F}_2$ ou $\mathbb{F}_3$). Nous n'avons ainsi que 3 ou 4 points selon la caractéristique où évaluer le polynôme, cependant nous pouvons à nouveau étendre le corps de base : considérons maintenant les polynômes de degré 1 au plus sur les extensions rationnelles de $\mathbb{F}_2$ et $\mathbb{F}_3$, cela revient à se permettre d'utiliser X et $X+1$ comme points d'interpolation. Malheureusement, utiliser $X+1$ nous contraint à effectuer une division (exacte) par $X+1$ à la reconstruction, c'est une opération compliquée et fortement séquentielle. Nous avons donc décidé d'abandonner l'idée d'utiliser l'algorithme de Toom-Cook faute de temps devant les difficultés qu'il posait. Cependant il faudrait peut-être reprendre cet algorithme pour la caractéristique 3, en effet les points 0, 1, -1, X et ∞ suffisent pour interpoler et ne demandent pas de division non-triviale à la reconstruction.

[2]Rappelons que $1 = -1$ en caractéristique 2.

M. Bodrato s'est beaucoup intéressé à l'algorithme de Toom-Cook et ses variations (notamment sur les polynômes sur $\mathbb{F}_2$). Ses travaux furent une grande source d'inspiration pour cette section [6].

3.5.4 Formules de Montgomery

Partant du constat que les formules de Karatsuba en séparant les polynômes en 5, 6 ou 7 parties demandent beaucoup de multiplications de taille inférieure, P.L. Mongomery a cherché des formules demandant moins d'appels récursifs [29]. Ses résultats ont été amélioré par H. Fan et M. A. Hasan [14]. Bien que ces formules demandent effectivement moins de multiplications que celles que l'on obtient uniquement en utilisant l'identité (3.7), elles compliquent très fortement la reconstruction. Faute de temps, nous avons choisi de ne pas examiner plus en détails ces formules, ni de les implémenter ; cela reste une piste à explorer.

3.6 Une approche mixte

Les algorithmes récursifs que nous venons de présenter peuvent tous s'écrire sous la forme : une étape de séparation des entrées, des appels récursifs à une multiplication de taille inférieure et une étape de reconstruction du produit. Ainsi le choix reste entier quant à l'algorithme à utiliser pour la multiplication de taille inférieure, nous pouvons donc utiliser différents algorithmes selon les étages de récursion.

Les différents algorithmes que nous avons présentés ne font pas le même nombre d'appels récursifs et n'ont pas le même coût pour les étapes de séparation et reconstruction. Certains permettent de faire peu de multiplications mais sont coûteux pour la reconstruction, ils seront alors avantageux pour les *grandes* tailles de polynômes. D'autres demandent plus de multiplications mais sont moins coûteux, ce sont ceux-ci que nous utiliserons pour les multiplications de petites tailles. Ainsi pour calculer le produit de deux polynômes, nous pourrons par exemple faire appel une fois à $\mathcal{K}_{3,}$ puis trois fois à $\mathcal{K}_{2,}$ et terminer avec l'algorithme de multiplication naïve. Cette façon de choisir plusieurs algorithmes selon la taille de multiplication à effectuer est utilisée dans la bibliothèque de calcul multi-précision GMP [17].

Remarquons finalement qu'utiliser $\mathcal{K}_{2,m}$ avec m impair ou $\mathcal{K}_{3,m}$ avec m non-divisible par 3 demande de rajouter des coefficients nuls dans les calculs intermédiaires, et ainsi nous effectuons des calculs inutiles. À chaque étape de la récursion, il faut donc choisir l'algorithme qui donnera un coût moindre.

Chapitre 4

Implémentation et résultats expérimentaux

Nous exposerons dans ce chapitre l'implémentation matérielle des précédents algorithmes. Nous avons choisi une implémentation sur matériel reconfigurable : les FPGA. Nous les décrirons brièvement avec leur programmation dans une première section 4.1. Puis nous décrirons les architectures associées aux algorithmes précédents (Section 4.2). Enfin nous donnerons quelques analyses de performances afin de comparer les différents algorithmes mis en œuvre (Section 4.3) avant de comparer nos résultats avec la littérature existante (Section 4.4).

4.1 Les FPGA et leur programmation

L'idée de se servir de matériel reconfigurable a d'abord été émise par G. Estrin en 1960 [13], cette idée s'est ensuite concrétisée au milieu des années 1980 lorsque Xilinx a sorti le premier FPGA (*Field Programmable Gates Array*). Les FPGA permettent d'implémenter virtuellement tous les circuits logiques, et de se servir ensuite du composant pour effectuer les calculs. Ainsi le paradigme qui motive l'utilisation des FPGA consiste à associer la malléabilité d'un programme et la rapidité d'un composant.

Les FPGA concurrencent les ASIC (*Appliction-Specific Integrated Circuit*) qui eux ne sont pas reprogrammables et demandent un long processus de conception. Cependant les ASIC permettent d'atteindre de meilleures performances en fonctionnant à des fréquences 5 à 50 fois plus élevées.

4.1.1 Architecture des FPGA

Nous présentons ici l'architecture des FPGA de la marque Xilinx que nous avons utilisés. Les autres FPGA ont des architectures silmilaires mais les termes et les paramètres diffèrent. Afin de pouvoir implémenter tous les circuits (du moins dans la limite d'une certaine taille), l'architecture des FPGA se présente comme une grille de cellules programmables : les CLB (*Configurable Logic Blocks*) ; celles-ci sont reliées par une matrice de routage elle aussi programmable (voir Fig. 4.1, p.14).

Chaque CLB se compose de plusieurs *slices*, qui elles-mêmes se composent de plusieurs LUT (*Look-Up Table*) que nous considérerons ici comme la brique de base nous permettant de réaliser un circuit. Il existe de nombreux modèles de FPGA et le nombre de *slices* et de LUT sont des paramètres qui varient. De plus, nous ne cherchons ici qu'à donner un point de vue très partiel de l'architecture des FPGA pour justifier certains choix d'implémentation. Les LUT (*Look-Up Table*) permettent de réaliser n'importe quelle fonction booléenne à quatre entrées ; elles sont suivies d'un *flip-flop* ce qui permet de se servir de leur sortie en différé ou immédiatement selon le besoin (Fig. 4.2, p.14). Nous avons aussi exploré les possibilités qu'offrent les nouveaux FPGA `Virtex V` dont l'architecture des *slices*, légèrement différente, permet d'avoir des LUT à 6 entrées ou des LUT à 5 entrées et 2 sorties.

4.1.2 VHDL et synthèse du circuit

La programmation des FPGA passe par un langage de description de circuit. Il en existe deux principaux : VHDL et `Verilog`. Nous avons utilisé VHDL. Pour permettre de décrire un circuit générique (c'est-à-dire pour différentes tailles de données par exemple), nous avons choisi de générer les descriptions de circuit grâce à un programme écrit en `Python`.

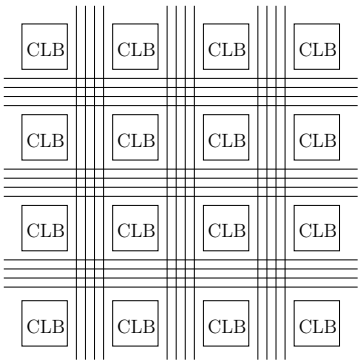


FIG. 4.1 – Agencement des CLB dans un FPGA

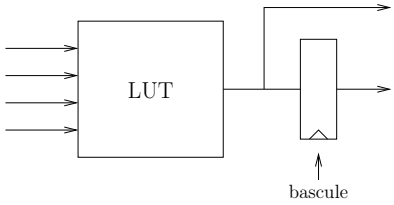


FIG. 4.2 – Schéma simplifié d'une demi-*slice* sur un `Virtex II pro`

Les architectures ainsi décrites peuvent être simulées. Nous avons utilisé `Modelsim` pour ceci. Toutes nos architectures ont pu être testées et validées par simulation.

Enfin le circuit peut être synthétisé (création de la *netlist* qui décrit l'architecture avec de la logique élémentaire uniquement) puis placé et routé (projection de cette *netlist* sur l'architecture d'un FPGA). Ainsi nous obtenons une description de la configuration à appliquer à un FPGA pour qu'il exécute notre circuit ; l'analyse de cette description nous donne alors les performances de notre circuit :

- sa taille (mesurée en nombre de *slices*),
- sa fréquence maximale de fonctionnement.

Nous rappelons que le but de notre multiplieur parallèle est d'être rapide et que notre premier critère fut donc la fréquence, même si le produit temps-surface nous a aussi intéressé.

4.2 Implémentations des algorithmes de multiplication

4.2.1 Représentation de $\mathbb{F}_p$ et opérateurs sur $\mathbb{F}_p$

Tous les algorithmes que nous avons décrits dans le Chapitre 3 se basent sur l'arithmétique sur $\mathbb{F}_p$. Nous allons décrire ici comment elle se réalise sur un circuit. Nous ne travaillons qu'en caractéristique 2 ou 3.

Les opérations sur $\mathbb{F}_2$ sont simples. En effet $\mathbb{F}_2$ peut être vu comme $\{0, 1\}$ muni de l'opérateur d'addition `Xor` et de l'opérateur de multiplication `And`. Ainsi cette arithmétique est très facile à réaliser sur un circuit en représentant $\mathbb{F}_2$ par 0 ou 1 sur un fil et en utilisant les portes logiques habituelles.

Le cas de $\mathbb{F}_3$ est légèrement plus compliqué. Nous avons choisi de représenter les éléments de $\mathbb{F}_3$ sur deux bits à la manière des représentations *borrow-save* : $a \in \mathbb{F}_3$ est représenté par (a^+, 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 les opérations $+$ et $\times$. En caractéristique 3, il faut aussi pouvoir prendre l'opposé d'un nombre ; c'est une opération gratuite en matériel vu la représentation choisie : il suffit d'échanger le bit $+$ et le bit $-$. Nous avons essayé de tirer profit des LUT 5 entrées 2 sorties des `Virtex V` pour réaliser chaque opération en une seul LUT. Cependant nos essais ne furent pas concluants, les outils de synthèse n'étant pas au point pour utiliser ce matériel relativement récent.

4.2.2 Architecture du multiplieur sur $\mathbb{F}_{p^m}$

Nous allons décrire ici l'architecture générale de notre multiplieur sur $\mathbb{F}_{p^m}$ étant donné un multiplieur sur $\mathbb{F}_p[X]$. Un schéma de l'opérateur est donné à la figure 4.3. Il s'agit essentiellement de calculer le produit sur $\mathbb{F}_p[X]$ et de le réduire ensuite.

La réduction consiste en la somme d'un coefficient de poids faible et de *quelques* coefficients de poids forts. Cette somme dépendant du polynôme irréductible, il est difficile d'évaluer *a priori* la taille et le chemin critique de ce composant.

Notre multiplieur est pipeliné : si nous lui donnons un jeu d'entrée 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. Notre générateur permet de choisir la profondeur du pipeline, c'est pourquoi il y a des registres optionnels que l'on peut choisir de générer ou non dans nos architectures. La 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 on le place dans un circuit complet, nous avons placé des registres en entrée et en sortie du multiplieur. Étant donné l'algorithme de couplage que nous avons choisi, nous avons utilisé un pipeline de profondeur 5 en caractéristique 2 et 7 en caractéristique 3.

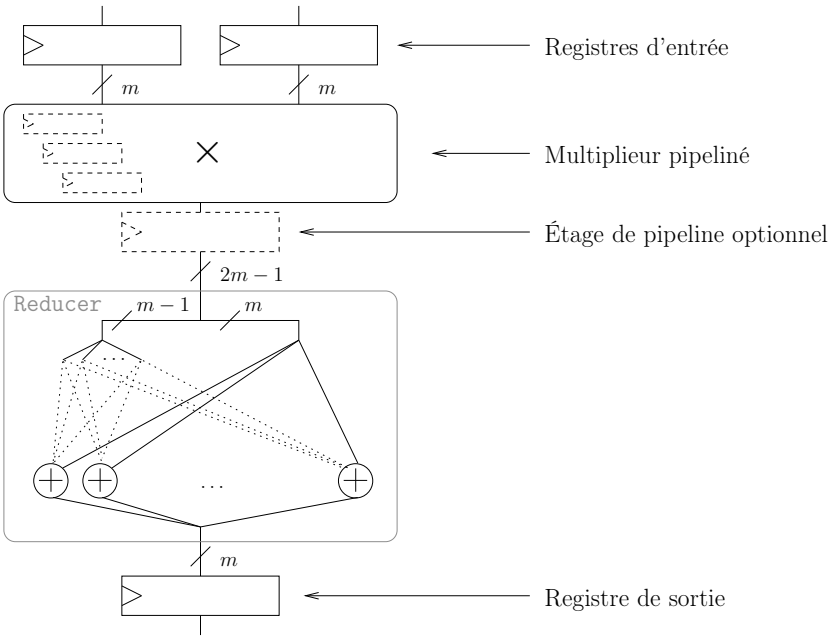


FIG. 4.3 – Architecture générale du multiplieur

4.2.3 Architecture d'un étage de multiplication selon Karatsuba

Nous allons décrire dans cette sous-section qui est commun à tous les algorithmes récursifs que nous avons implémentés (FIG. 4.4, p.16).

Comme nous l'avons vu à la Section 3.6, tous nos algorithmes récursifs font appel à des multiplications de rang inférieur sans se soucier de la façon dont elles sont calculées. Ainsi nous montrons notre architecture avec des *boîtes noires* pour les multiplications de rang inférieur.

De façon générale, chaque algorithme consiste en quelques additions pour découper les entrées en plusieurs jeux d'entrées pour les multiplications de rang inférieur. Les sorties de ces multiplieurs sont alors récupérées par un composant qui les rassemblent, 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 `Splitter` et `Recomposer` 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 jusqu'à 4 (ou 6 sur `Virtex V`) 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, seule la somme de deux (ou trois sur `Virtex V`) termes maximum ne traverse qu'une LUT. C'est ce point d'implémentation qui fait que certains algorithmes (tel que $\mathcal{K}_{3,m}$) ne sont pas aussi efficaces en pratique que théoriquement.

4.3 Résultats expérimentaux

Nous avons implémenté les algorithmes suivant : l'algorithme naïf, le produit de Mastrovito, $\mathcal{K}_{2,m}$, $\mathcal{K}'_{2,m}$ et $\mathcal{K}_{3,m}$ sur un FPGA `Virtex II pro` (`xc2vp50-ff1517-6`). Nous montrons dans cette Section les performances qui leur sont associées. Dans un premier tableau (TAB. 4.1), nous comparons les approches quadratiques (algorithme naïf, Mastrovito) et subquadratiques (Karatsuba). Bien que l'algorithme naïf permette d'obtenir de meilleures performances, il n'est pas utilisable dès que la taille augmente car il demande une trop grande surface.

Dans un second tableau (TAB. 4.2), nous montrons différents choix de récursion pour réaliser la multiplication sur $\mathbb{F}_{p^m}$ avec $\mathbb{F}_{p^m}$ un corps couramment utilisé en cryptographie à base de couplage. Nous montrons ainsi qu'il faut faire un compromis entre le temps et la surface, en effet plus nous utilisons l'algorithme de multiplication naïve tôt dans la récursion, plus la surface augmente et la latence diminue.

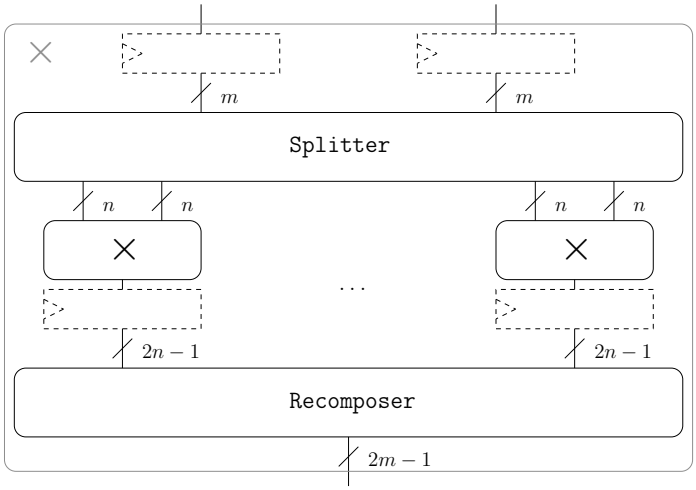


FIG. 4.4 – Architecture d'un multiplieur récursif

Algorithme	$\mathbb{F}_{p^m}$	Temps (ns)	Fréquence (MHz)	Surface (slices)	Produit temps-surface ($\mu s \cdot$ slices)
Naïf	$\mathbb{F}_{2^8}$	3.482	287	28	0.097
$\mathcal{K}_{2,8}$, $\mathcal{K}_{2,4}$, Naïf	$\mathbb{F}_{2^8}$	3.879	258	34	0.132
Naïf	$\mathbb{F}_{2^{31}}$	6.496	154	406	2.64
$\mathcal{K}_{2,31}$, Naïf	$\mathbb{F}_{2^{31}}$	7.007	143	406	2.84
$\mathcal{K}_{2,31}$, $\mathcal{K}_{2,16}$, Naïf	$\mathbb{F}_{2^{31}}$	8.016	125	334	2.68
$\mathcal{K}_{2,31}$, $\mathcal{K}_{2,16}$, $\mathcal{K}_{2,8}$, Naïf	$\mathbb{F}_{2^{31}}$	7.595	131	351	2.67
Mastrovito	$\mathbb{F}_{2^{239}}$	24.786	40	29180	723
$\mathcal{K}_{2,239}$, $\mathcal{K}_{2,120}$, $\mathcal{K}_{2,60}$, $\mathcal{K}_{2,30}$, Naïf	$\mathbb{F}_{2^{239}}$	19.108	52	10417	199
Naïf	$\mathbb{F}_{3^4}$	4.790	209	42	0.201
$\mathcal{K}_{2,4}$, Naïf	$\mathbb{F}_{3^4}$	6.282	159	50	0.334
Naïf	$\mathbb{F}_{3^{16}}$	9.637	104	761	7.33
$\mathcal{K}_{2,16}$, $\mathcal{K}_{2,8}$, $\mathcal{K}_{2,4}$, Naïf	$\mathbb{F}_{3^{16}}$	13.476	74	585	7.88
$\mathcal{K}_{2,16}$, $\mathcal{K}_{2,8}$, Naïf	$\mathbb{F}_{3^{16}}$	11.700	85	499	5.84
$\mathcal{K}_{2,97}$, $\mathcal{K}_{2,49}$, $\mathcal{K}_{2,25}$, $\mathcal{K}_{2,13}$, $\mathcal{K}_{2,7}$, Naïf	$\mathbb{F}_{3^{97}}$	27.000	37	9660	260

Aucun des opérateurs n'a été pipeliné pour cette comparaison.

TAB. 4.1 – Comparaison des approches quadratiques et subquadratiques

L'architecture en LUT à 4 entrées des FPGA influe fortement sur les performances des algorithmes une fois implémentés. Ainsi la modification de l'algorithme $\mathcal{K}_{2,m}$ proposée à la Section 3.5.2 (p. 10), n'apporte rien en caractéristique 2, nous expliquons ceci par le fait que la somme de 2, 3 ou 4 éléments de $\mathbb{F}_2$ n'utilise qu'une LUT, et donc, dans le cas de $\mathcal{K}_{2,m}$, éliminer les recouvrements ne diminue pas la profondeur de la somme à calculer durant le recouvrement, et de plus cela augmente le nombre de termes à additionner. Nous ne retrouvons pas cet effet en caractéristique 3 car la somme de 3 ou 4 éléments de $\mathbb{F}_3$ demande effectivement plus de LUT que la somme de 2 éléments de $\mathbb{F}_3$. Nous n'avons malheureusement pas eu le temps d'implémenter l'algorithme modifié $\mathcal{K}'_{3,m}$ mais nous pensons qu'il apporte des améliorations substantielles à l'algorithme $\mathcal{K}_{3,m}$.

Algorithme	$\mathbb{F}_{p^m}$	Temps (ns)	Fréquence (MHz)	Latence (ns)	Surface (slices)	Produit temps-surface (μs·slices)
$\mathcal{K}_{2,239}$, $\mathcal{K}_{2,120}$, $\mathcal{K}_{2,60}$, $\mathcal{K}_{2,30}$, $\mathcal{K}_{2,15}$, Naïf	$\mathbb{F}_{2^{239}}$	7.477	134	37.39	9028	67.5
$\mathcal{K}'_{2,239}$, $\mathcal{K}'_{2,120}$, $\mathcal{K}'_{2,60}$, $\mathcal{K}'_{2,30}$, $\mathcal{K}'_{2,15}$, Naïf	$\mathbb{F}_{2^{239}}$	7.533	133	37.67	10897	82.1
$\mathcal{K}_{2,239}$, $\mathcal{K}_{2,120}$, $\mathcal{K}_{2,60}$, $\mathcal{K}_{2,30}$, Naïf	$\mathbb{F}_{2^{239}}$	6.783	147	33.92	11792	80.0
$\mathcal{K}'_{2,239}$, $\mathcal{K}'_{2,120}$, $\mathcal{K}'_{2,60}$, $\mathcal{K}'_{2,30}$, Naïf	$\mathbb{F}_{2^{239}}$	6.667	150	33.34	12490	83.2
$\mathcal{K}_{3,239}$, $\mathcal{K}_{2,80}$, $\mathcal{K}_{2,40}$, $\mathcal{K}_{2,20}$, Naïf	$\mathbb{F}_{2^{239}}$	6.801	147	34.01	9237	62.8
$\mathcal{K}'_{3,239}$, $\mathcal{K}_{2,80}$, $\mathcal{K}_{2,40}$, $\mathcal{K}_{2,20}$, Naïf	$\mathbb{F}_{2^{239}}$	6.798	147	33.99	9218	62.7
$\mathcal{K}'_{2,313}$, $\mathcal{K}'_{2,157}$, $\mathcal{K}'_{2,79}$, $\mathcal{K}'_{2,40}$, $\mathcal{K}'_{2,20}$, Naïf	$\mathbb{F}_{2^{313}}$	8.998	111	45,00	15415	139
$\mathcal{K}_{2,97}$, $\mathcal{K}_{2,49}$, $\mathcal{K}_{2,25}$, $\mathcal{K}_{2,13}$, $\mathcal{K}_{2,7}$, Naïf	$\mathbb{F}_{3^{97}}$	7.987	125	55,91	10403	83,1
$\mathcal{K}'_{2,97}$, $\mathcal{K}'_{2,49}$, $\mathcal{K}'_{2,25}$, $\mathcal{K}'_{2,13}$, $\mathcal{K}'_{2,7}$, Naïf	$\mathbb{F}_{3^{97}}$	7.705	130	53.94	10316	79.5

Les multiplieurs ont été pipelinés en 5 étages pour la caractéristique 2 et 7 pour la caractéristique 3.

Tab. 4.2 – Différentes récursions pour la multiplication

4.4 Comparaison avec d'autres approches

Dans le Tableau 4.3, nous comparons notre implémentation avec d'autres approches trouvées dans la littérature. Nous ne pouvons malheureusement pas fournir de comparaison pour différents corps $\mathbb{F}_{p^m}$, étant donné que ce sont généralement les performances relatives au calcul du couplage dans son ensemble qui sont fournies. Cependant nous pouvons constater que nous obtenons de bonnes performances sur $\mathbb{F}_{3^{97}}$. N. Cortez-Duarte *et al.* [11] proposent un multiplieur pipeliné utilisant l'algorithme de Karatsuba-Offman mais séquencialise une partie des appels réccursifs. J.-L. Beuchat *et al.* [3] donnent un opérateur très compact qui calcule le produit en 33 cycles, cette approche séquentielle donnant une latence beaucoup plus grande. Nous améliorons ainsi le produit temps-surface.

Algorithme	$\mathbb{F}_{p^m}$	FPGA	Temps (ns)	Fréquence (MHz)	Surface (slices)	Produit temps-surface (μs·slices)
N. Cortez-Duarte *et al.* [11]	$\mathbb{F}_{3^{97}}$	`Virtex II pro`	17.90	57	9041	162
J.-L. Beuchat *et al.* [3]	$\mathbb{F}_{3^{97}}$	`Cyclone II`	221.5	149	700	155
$\mathcal{K}'_{2,97}$, $\mathcal{K}'_{2,49}$, $\mathcal{K}'_{2,25}$, $\mathcal{K}'_{2,13}$, $\mathcal{K}'_{2,7}$, Naïf	$\mathbb{F}_{3^{97}}$	`Virtex II pro`	7.705	130	10316	79.5

Les temps donnés ici sont les latences entre deux résultats successifs que peut fournir chaque multiplieur.

Tab. 4.3 – Comparaison avec des multiplieurs de la littérature

Chapitre 5

Conclusion

Dans le cadre de cette étude, nous proposons différents algorithmes de multiplication sur corps fini de caractéristiques 2 et 3. De même nous présentons des architectures parallèles les réalisant. Ces multiplieurs ont été conçus pour être intégrés à un coprocesseur permettant de calculer le couplage de Tate modifié [1]. Ainsi nous avons exploré un certain nombre d'algorithmes de multiplication, certains classiques (Karatsuba, Toom) et d'autres plus anecdotiques (formules de Montgomery, produit de Mastrovito). De plus nous avons implémenté une partie de ces algorithmes sur FPGA. Ceci nous a permis d'affiner nos comparaisons entre les différents algorithmes. De plus ces implémentations seront utilisées dans le coprocesseur de couplage.

De nombreuses pistes restent encore à explorer dans l'étude des multiplieurs parallèles. Nous aurions par exemple aimé poursuivre l'étude des algorithmes de multiplication selon Toom-Cook en caractéristique 3. De même les formules de Montgomery ne semblent pas adaptées à un opérateur matériel à cause du nombre élevé d'additions à la reconstruction, cependant il pourrait être intéressant de rechercher des formules selon la méthode de P. L. Montgomery en découpant les opérandes différemment. Nous regrettons enfin de ne pas avoir eu le temps de proposer une implémentation pour chacun des algorithmes présentés dans ce rapport.

D'un point de vue plus personnel, ce stage m'a permis de découvrir la cryptographie basée sur les courbes elliptiques et l'arithmétique modulaire. J'ai ainsi pu approfondir mes connaissances en architecture des ordinateurs et me familiariser avec la programmation les FPGA.

Bibliographie

[1] Paulo S. L. M. Barreto, Hae Y. Kim, Ben Lynn, and Michael Scott. Efficient algorithms for pairing-based cryptosystems. In M. Yung, editor, *Advances in Cryptology – CRYPTO 2002*, number 2442 in Lecture Notes in Computer Science, pages 354–369. Springer, 2002.

[2] Daniel J. Bernstein. Multidigit multiplication for mathematicians. *Advances in Applied Mathematics*, 2001.

[3] Jean-Luc Beuchat, Nicolas Brisebarre, Jérémie Detrey, and Eiji Okamoto. Arithmetic operators for pairing-based cryptography. In *CHES'07*, number 4727 in Lecture Notes in Computer Science, pages 239–255. Springer-Verlag, 2007.

[4] Jean-Luc Beuchat, Nicolas Brisebarre, Jérémie Detrey, Eiji Okamoto, and 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'08*, number 5209 in Lecture Notes in Computer Science. Springer-Verlag, 2008.

[5] Jean-Luc Beuchat, Takanori Miyoshi, Yoshihito Oyama, and Eiji Okamoto. Multiplication over $\mathbb{F}_{p^m}$ on FPGA: A survey. *ARC'07, Lecture Notes in Computer Science. Springer*, 2007.

[6] Marco Bodrato. Towards optimal Toom-Cook multiplication for univariate and multivariate polynomials in characteristic 2 and 0. In Claude Carlet and Berk Sunar, editors, *WAIFI'07 proceedings*, volume 4547 of *LNCS*, pages 116–133. Springer, June 2007. `http://bodrato.it/papers/#WAIFI2007`.

[7] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In *Advances in Cryptology – CRYPTO 2001*, number 2139 in Lecture Notes in Computer Science, pages 213–229. Springer-Verlag London, UK, 2001.

[8] Dan Boneh, Ben Lynn, and Hovav Shacham. Short Signatures from the Weil Pairing. *Journal of Cryptology*, 17(4):297–319, 2004.

[9] Florent Chabaud. Polynomials over galois fields. `http://fchabaud.free.fr/English/default.php?COUNT=1&FILE0=Poly`.

[10] Stephen A. Cook. *On the Minimum Computation Time of Function*. PhD thesis, Harvard University, 1966.

[11] Nidia Cortez-Duarte, Francisco Rodríguez-Henríquez, Jean-Luc Beuchat, and Eiji Okamoto. A Pipelined Karatsuba-Ofman Multiplier over GF(3^{97}) Amenable for Pairing Computation. Cryptology ePrint Archive, Report 2008/127, 2008.

[12] Iwan Duursma and Hyang-Sook Lee. Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$. In C.S. Laih, editor, *Advances in Cryptology – ASIACRYPT 2003*, number 2894 in Lecture Notes in Computer Science, pages 111–123. Springer, 2003.

[13] Gerald Estrin. Organization of computer systems: The fixed-plus variable structure computer. *Proceedings of the Western Joint Computer Conference*, pages 33–40, 1960.

[14] Haining Fan and M. Anwar Hasan. Comments on Montgomery's "Five, Six, and Seven-Term Karatsuba-Like Formulae". 2006.

[15] Haining Fan and M. Anwar Hasan. A new approach to subquadratic space complexity parallel multipliers for extended binary fields. *IEEE Transactions on Computers*, 56(2):224–233, February 2007.

[16] Haining Fan, Jiaguang Sun, Ming Gu, and Kwok-Yan Lam. Overlap-free Karatsuba-Ofman polynomial multiplication algorithm. Cryptology ePrint Archive, Report 2007/393, 2007.

[17] Free Software Foundation, Inc. *GMP: The GNU multiple precision arithmetic library*, 2008. `http://gmplib.org/`.

[18] Gerhard Frey and Hans-Georg Rück. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Mathematics of Computation*, 62(206):865–874, 1994.

[19] Steven D. Galbraith, Keith Harrison, and David Soldera. Implementing the tate pairing. In C. Fieker and D.R. Kohel, editors, *Algorithmic Number Theory – ANTS 2002*, number 2369 in Lecture Notes in Computer Science, pages 324–337. Springer, 2002.

[20] Darrel R. Hankerson, Alfred J. Menezes, and Scott A. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[21] Guillaume Hanrot and Paul Zimmermann. A long note on Mulders'short product. *Journal of Symbolic Computation*, 37(3):391–401, 2004.

[22] Antoine Joux. A one round protocol for tripartite Diffie–Hellman. *Journal of Cryptology*, 17(4):263–276, 2004.

[23] Anatolii Alexevich Karatsuba and Yu Ofman. Multiplication of Multidigit Numbers on Automata. *Soviet Physics Doklady*, 7:595, 1963.

[24] Serge Lang. *Algebra*. Addison-Wesley, 2nd edition, 1984.

[25] Edoardo D. Mastrovito. VLSI design for multiplication over finite fields $GF(2^m)$. *Lecture Notes in Computer Science*, 3(357):297–309, 1989.

[26] Alfred J. Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *Information Theory, IEEE Transactions on*, 39(5):1639–1646, 1993.

[27] Victor S. Miller. Short programs for functions on curves. IBM, Thomas J. Watson Research Center, 1986.

[28] Victor S. Miller. The Weil pairing, and its efficient calculation. *Journal of Cryptology*, 17(4):235–261, 2004.

[29] Peter L. Montgomery. Five, six, and seven-term karatsuba-like formulae. *IEEE Transactions on Computers*, 54(3):362–369, March 2005.

[30] Arash Reyhani-Masoleh and M. Anwar Hasan. Low complexity bit parallel architectures for polynomial basis multiplication over gf(2^m). *IEEE Transactions on Computers*, 53(8):945–959, August 2004.

[31] Francisco Rodríguez-Henríquez, N. A. Saqib, Arturo Díaz-Pérez, and Çetin Kaya Koç. *Cryptographic Algorithms on Reconfigurable Hardware*. Signals and Communication Technology. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[32] Jamshid Shokrollahi. *Efficient Implementation of Elliptic Curve Cryptography on FPGAs*. PhD thesis, Rheinischen Friedrich-Wilhelms-Universitat Bonn, 2006.

[33] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*. Springer Verlag, 1986.

[34] Leilei Song and Keshab K. Parhi. Low energy digit-serial/parallel finite field multipliers. *Journal of VLSI Signal Processing*, 19(2):149–166, July 1998.

[35] John T. Tate. Endomorphisms of abelian varieties over finite fields. *Inventiones Mathematicae*, 2(2):134–144, 1966.

[36] Andrei L. Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. *Soviet Mathematics Doklady*, 3:714–716, 1963.

[37] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2003.

[38] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography*. CRC Press, 2003.

[39] André Weil. Sur les fonctions algébriques à corps de constantes fini. *CR Acad. Sci. Paris*, 210:592–594, 1940.

[40] André Weimerskirch and Christof Paar. Generalizations of the Karatsuba algorithm for efficient implementations. Cryptology ePrint Archive, Report 2006/243, 2006.