Implémentation efficace de primitive cryptographique pour le couplage sur carte FPGA

Amine Mrabet

Université Paris 8
&
École Nationale d'Ingénieurs de Tunis

THÈSE

pour obtenir le grade de

Docteur de l'Université Paris 8

Et de l'École Nationale d'Ingénieurs de Tunis

Spécialité : **Informatique**

préparée au laboratoire **LAGA et EμE**

dans le cadre de l'École Doctorale **CLI et SC**

présentée et soutenue publiquement

par

Amine MRABET

le 08/11/2017

Titre:

Implémentation efficace de primitive cryptographique pour le couplage sur carte FPGA

Directrice de thèse: **Sihem MESNAGER**

Co-directeur de thèse: **Mohsen MACHHOUT**

Jury

Jean Claude BAJARD	Examinateur
Rached TOURKI	Examinateur
Hichem TRABELSI	Examinateur
Ridha BOUALLEGUE	Rapporteur
Christophe NÈGRE	Rapporteur
Nadia EL-MRABET	Encadrante
Sihem MESNAGER	Directrice de thèse
Mohsen MACHHOUT	Co-directeur de thèse

Résumé

Le défi primaire dans le développement matériel de la cryptographie moderne est de faire des implémentations optimales en ressources, et rapide, en garantissant une résistance contre les attaques. Cette recherche porte sur les implémentations pratiques des opérations de cryptographie basées sur la cryptographie à clé publique dans les corps finis. Durant cette thèse nous avons proposé des composants matériels de base. L'arithmétique des corps finis constitue le noyau de la cryptographie à clé publique comme RSA, ECC ou une cryptographie basée sur le couplage.

Nous avons proposé dans cette thèse des architectures du calcul arithmétique haute performance pour implémenter les primitives de cryptographie asymétrique. Les composants décrits dans notre travail ont été implémentés dans des Field Programmable Gate Array platforms (FPGA) de Xlinx. Nous avons utilisé le VHDL pour dévolopper nos composants et nos architectures. Nos résultats présentent des performances en ressources et en vitesse jamais égalées auparavant dans la littérature publique sur ce type de technologie. La particularité de ces architectures est l'utilisation de l'architecture systolique pour développer une multiplication modulaire.

Cette thèse traite la mise en oeuvre matérielle efficace de la méthode CIOS (Coarsely Integrated Operand Scanning) de la multiplication modulaire de Montgomery combinée avec une architecture systolique efficace. D'après nos connaissances, c'est la première implémentation d'une telle conception. Nos architectures visaient à réduire le nombre de cycles d'horloge de la multiplication modulaire. Les résultats d'implémentation des algorithmes CIOS se concentrent sur différents niveaux de sécurité utiles en cryptographie. Cette architecture a été conçue pour utiliser le DSP48 flexible sur les FPGA de Xilinx. Nos architectures sont évolutives et dépendent uniquement du nombre et de la taille des mots. Par exemple, nous fournissons des résultats d'implémentation pour des longs mots de 8, 16, 32 et 64 bits en 33, 66, 132 et 264 cycles d'horloge. Nous décrivons également un design pour calculer une inversion et/ou une division dans $\mathbb{F}_p$. L'inversion peut être utilisée dans les systèmes

de la cryptographie de courbe elliptique et de la cryptographie basée sur le couplage.

Abstract

The primary challenge in the hardware development of the modern cryptography is to make an optimal implementations in resources and speed, with guaranteeing a resistance against attacks. This research focuses on practical implementations of cryptographic operations based on public key cryptography in finite fields. During this thesis we proposed basic hardware components. Finite field arithmetic is the core of public key cryptography such as RSA, ECC, or pairing-based cryptography.

We proposed in this thesis a high-performance architectures of arithmetic calculation to implement asymmetric cryptographic primitives. The components described in this thesis have been implemented in Xlinx Field Programmable Gate Array Platforms (FPGAs). We used the VHDL to devolve our components and architectures. Our results show a performance and speed never presented before in the literature on this type of technology. The particularity of these architectures is the use of systolic architecture to develop a modular multiplication.

This thesis deals with the effective physical implementation of the Coarsely Integrated Operand Scanning (CIOS) method of Montgomery's modular multiplication combined with an effective systolic architecture. According to our knowledge, this is the first implementation of such a design. Our architectures were aimed at reducing the number of clock cycles of modular multiplication. The implementation results of the CIOS algorithms focus on different levels of security useful in cryptography. This architecture was designed to use the flexible DSP48 on Xilinx FPGAs. Our architectures are scalable and depend only on the number and size of the words. For instance, we provide implementation results for 8, 16, 32, and 64 bit long words in 33, 66, 132, and 264 clock cycles. We describe also a design to compute an inversion in Fp as well as division. Inversion can be used in Elliptic Curve Cryptography systems and pairing-based cryptography.

Remerciements

Je tiens tout d'abord à remercier mes directeurs de thèse, Sihem Mesnager et Mohsen Machhout, et mes encadrants, Nadia El Mrabet et Belgacem Bouallegue qui m'ont permis d'effectuer ce travail sur une problématique passionnante. Ils m'ont notamment mis le pied à l'étrier sur les aspects cryptographiques. Je leur suis reconnaissant de m'avoir fait bénéficier tout au long de ce travail de leur grandes compétences, de leur rigueurs intellectuelles, de leur dynamismes, et de leur efficacités certaines que je n'oublierai jamais. Soyez assuré de mon attachement et de ma profonde gratitude.

Je remercie aussi les autres membres du jury : Jean-Claude Bajard, Rached Tourki et Hichem Trabelsi en tant qu'examinateurs, et enfin les rapporteurs Christophe Nègre et Ridha Bouallegue.

Un grand merci à tous les membres de l'équipe de département TIC et du laboratoire Xlim de la faculté des sciences et techniques à Limoges pour leur sympathie, leur amitié. J'ai eu beaucoup de plaisir à travailler avec eux. Le cadre de travail était idéal.

Je voudrais aussi exprimer ma gratitude pour Mr Sylvain Duquesne, Professeur au laboratoire IRMAR pour ses aides au début de ma thèse dans la partie étude bibliographique. Je voudrais remercier également Ronan Lashermes, Jacques Fournier et Jean Baptiste Rigaud pour leurs contributions scientifiques.

Je tiens à remercier tous les membres des équipes du : Laboratoire d'électronique et micro-électronique à Monastir, laboratoire d'informatique avancé de Saint Denis, et l'équipe SAS dans le centre micro-électronique de Provence à Gardanne, que j'ai pu côtoyer durant ma thèse. Ces équipes possèdent une formidable ambiance de travail, ce qui est très motivant et très encourageant dans les périodes difficiles qui

juchent le chemin jusqu'à la soutenance de thèse. Je remercie plus particulièrement mes anciens co-bureaux, et les autres doctorants des groupes.

Je remercie mes parents Mhamed et Hédia, mon frère Ilyes, mes soeurs et leur maris : Asma, Abdelkrim et Soumaya, Brahim pour tout leur soutien tout au long de ma thèse.
Je remercie mes cousins Sabri et Wissem pour tout leur aides durant ma thèse.
Je remercie mon beau père Lassaad, ma belle-mère Kawther et mes beaux frères Arwa et Mohamed pour leur soutien durant ma thèse.

A titre plus personnelle, Je remercie ma femme, Mariem, pour la grande patience, l'encouragement et la confiance qu'elle m'a témoigné dont elle a fait preuve à la relecture de mon manuscrit. Elle a été mon soutien le plus fondamental durant toute la durée de ces travaux, et l'est encore aujourd'hui.

Glossaire

AEE : Algorithme d'Euclide étendu.
AES : Advanced Encryption Standard.
BRAM : Mémoire externe.
CLB : Blocs logiques configurables.
CIOS : Coarsely Integrated Operand Scanning.
CIHS : Coarsely Integrated Hybrid Scanning.
Coube BN : Courbes Barreto-Naehrig.
DL : Logarithme discret.
DLP : Problème de logarithme discret.
DSP : Digital signal processing.
ECC : Elliptic curve cryptography.
ECDLP : Elliptic Curve Discrete Logarithm Problem.
ECDH : Courbe elliptique Diffie-Hellman.
ECDSA : Algorithme de signature électronique de courbe elliptique.
FIOS : Finely Integrated Operand Scanning.
FIPS : Finely Integrated Product Scanning.
FPGA : Field-Programmable Gate Array.
FSM : Finit state machine (machine à états finis).
LB : Logic blocks.
LSW : Least Significant Word.
MMM : Multiplication Modulaire de Montgomery.
MSI : Medium Scale Integration.
MSW : Most Significant Word.
NIST : National Institute of Standards and Technology.
PDH : Problème de Diffie-Hellman.
PGCD : Le plus grand commun diviseur.
PLL : Phase-Locked Loop PKC : Cryptographie à clé publique.
SOS : Separated Operand Scanning.
SSI : Small Scale Integration.

RSA : Nommé par les initiales de ses trois inventeurs (Ronald Rivest, Adi Shamir et Leonard Adleman).
VHDL : VHSIC1 Hardware Description Language.

Table des matières

Table des figures

Liste des Algorithmes

Liste des tableaux

Chapitre 1

Introduction Générale

1 Contexte général

Depuis 1976, la cryptographie à clé publique (en anglais : public key cryptography PKC) [12,20,65] est apparue grâce à W.Diffie et M.Hellman [20]. Elle a révolutionné les systèmes de communication utilisés par plusieurs entreprises (tel que Thalès, Airbus, etc.), gouvernements et banques. La cryptographie à clé publique est appelée aussi la cryptographie asymétrique.

Pour communiquer confidentiellement, nous avons besoin de sécuriser nos messages. C'est pour cette raison et depuis l'antiquité, que la thématique de la cryptographie est utilisée. Dans [75] les auteurs ont introduit la cryptographie. Le chiffrement est un moyen de protection des messages, il consiste à rendre un message clair non compréhensible par quiconque n'est pas doté de la clé secrète de déchiffrement. Nous avons besoin de chiffrer le message lors de son transfert à travers un réseau non sécurisé d'une source vers une destination. Dans plusieurs réseaux de communication il y a des pirates qui cherchent à pirater nos messages. Un des objectifs de la cryptographie est alors de garantir la confidentialité des données. Nous commençons ce chapitre par introduire la définition de la cryptographie asymétrique et identifier ses bases mathématiques, nous présenterons ensuite l'arithmétique sur laquelle nous avons travaillé.

L'asymétrie pour la cryptographie à clé publique est basée sur des problèmes mathématiques difficiles. La construction des instances est calculatoirement facile. Par contre la résolution est très difficile ce qui montre l'intérêt de la cryptographie asymétrique. Un des problèmes les plus connus est la factorisation des entiers. Le cryptosystème à clé publique RSA [71] repose sur ce dernier problème. RSA est le protocole le plus utilisé dans les cryptosystèmes asymétriques. Il est proposé par

Rivest, Shamir et Adleman [71]. Ce type de chiffrement utilise une paire de clés composée d'une clé publique pour chiffrer et d'une clé privée pour déchiffrer des données confidentielles.

Pour générer les clés avec RSA nous devons faire des calculs, il est simple de calculer $n = pq$ si nous avons les opérandes p et q. Mais dans l'autre sens il est difficile de retrouver ces derniers opérandes si nous connaissons seulement le résultat n. RSA est très demandé dans le secteur du commerce électronique, et plus généralement pour toute communication via le net. Nous vous proposons le fonctionnement de l'algorithme de chiffrement RSA :

1. Choisir deux grands nombres premiers p et q aléatoirement.
2. Calculer le produit $n = p \times q$.
3. Calculer $\phi(n) = (p-1) \times (q-1)$.
4. Choisir e tel que $gcd(e, \phi(n)) = 1$ et $e < \phi(n)$
5. Calculer d tel que $e \times d \equiv 1 \bmod \phi(n)$.

Après ce calcul, nous obtenons la paire de clés (clé publique et clé privée). Avec la clé publique est la paire (n, e) et la clé privée est le triplet (d, p, q). Une fois que nous avons généré les clés, nous pouvons chiffrer et déchiffrer avec RSA. Afin de chiffrer un message clair M, nous devons calculer C de la manière suivante : $C \equiv M^e \bmod n$ en utilisant la clé publique (n, e) déjà générée du destinataire. Le récepteur peut déchiffrer le message qu'il a reçu en utilisant sa clé privée. Il doit calculer $M \equiv C^d \bmod n$. Sachant que la clé publique est connue par tout le monde alors que la clé privée est connue que par le destinataire.

Ces dernières années avec l'augmentation rapide des performances des ordinateurs, un grand progrès dans le domaine de la factorisation est apparu, ce qui influe sur la taille des clés utilisées par RSA. Par conséquence la dominance de RSA a diminué par rapport à d'autres cryptosystèmes à clé publique.

Aujourd'hui nous parlons de la dominance dans le domaine de la cryptographie à clé publique par ECC et même aussi par le couplage. Cet intérêt augmente grâce aux petites tailles utilisées pour les clés. La cryptographie basée sur les courbes elliptiques requière pour un niveau de sécurité équivalent, des clés bien plus petites que RSA. Si nous parlons de la cryptographie ECC et/ou du couplage, nous parlons de calculs arithmétiques complexes qui nécessitent plusieurs structures mathématiques. Pour ces nouveaux protocoles de la cryptographie à clé publique, plusieurs implémentations efficaces de ces arithmétiques complexes ont été proposées. C'est l'une des raisons pour lesquelles les cryptosystèmes basés sur les courbes elliptiques

connaissent une grande importance depuis leur proposition par Miller et Koblitz en 1985 [40, 55].

Comme RSA repose sur le problème de la factorisation, la cryptographie basée sur les courbes elliptiques repose sur le problème du logarithme discret. L'un des moyens d'obtenir une cryptographie asymétrique est l'utilisation des groupes sur lesquels le problème du logarithme discret est difficile à résoudre.

Pour une introduction plus détaillée pour la PKC nous vous proposons ces références [12, 65]. La PKC est basée sur des structures mathématiques difficiles qui seront détaillées dans la partie état de l'art du manuscrit. Aussi les systèmes à clé publique améliorent la gestion des clés en diminuant le problème difficile du partage des clés, et proposent des méthodes plus efficaces que les signatures numériques [14].

2 Motivation

Les cryptographes étudient des cryptosystèmes relativement nouveaux, ce sont les cryptosystèmes à courbes elliptiques (ECC) [40, 55]. Ces derniers ont un grand intérêt dans les applications commerciales et ils sont devenus les plus compacts dans le domaine de la cryptographie à clé publique par rapport à RSA. La Table 1.1 détaille une comparaison des clés entre RSA et ECC. Dernièrement la cryptographie basée sur le couplage est devenue importante grâce à l'apparition des nouveaux protocoles comme la cryptographie basée sur l'identité [10], en conséquence, son implémentation matérielle est devenue intéressante. Les implémentations doivent être rentables, en termes de vitesse et de ressources. Cette thèse se concentre sur l'implémentation des primitives cryptographiques à clé publique (ECC et couplage) basée sur des composants matériels.

Niveau de sécurité	80	112	128	192	256
RSA	1024	2048	3072	8192	15360
ECC	160	224	256	384	512
RSA/ECC	6.4	9.1	12	21.3	30

TABLE 1.1 – Comparaison RSA et ECC

2.1 Définition des Courbes elliptiques dans la cryptographie

Dans la littérature dans plusieurs travaux les auteurs ont utilisé les courbes elliptiques [45]. Par exemple Le travail de Lenstra présenté dans [45] a utilisé les courbes elliptiques pour la factorisation, il a amené à une grande découverte, c'est la cryptographie basée sur les courbes elliptiques (Elliptic Curve Cryptography ECC). Cette dernière a été proposée par Miller [55] et Koblitz [40] en 1985.

La cryptographie basée sur les courbes elliptiques a permit de réduire la taille des clés, elle nécessite des clés plus courtes en la comparant avec les techniques traditionnelles RSA pour avoir le même niveau de sécurité [59]. Si nous comparons la taille des clés, pour atteindre le niveau de sécurité équivalent à un AES de 256 bits, il est recommandé d'utiliser des clés de 512 bits pour ECC. Sachant que le cryptosystème RSA nécessite des clés avec la taille qui dépasse les 15000 bits. Dans [59] vous trouvez plus de détails pour les recommandations proposées par l'Institut national des normes et de la technologie (NIST). Ces différents avantages et recommandations proposés pour la cryptographie basée sur les courbes elliptiques ont amené à plusieurs implémentations efficaces et rapides. Par conséquence ces cryptosystèmes sont devenus plus demandés dans plusieurs domaines, tels que le militaire, les banques, les nouvelles technologies (smartphones) et d'autres.

ECC a gagné plus d'importance pour le futur des systèmes et des informations de sécurité américains classifiés et non classifiés [77], il est devenu bien positionné parmi les systèmes à clé publique, et il est prévu qu'il garde cette position dans le futur. Nous pouvons montrer la dominance de ECC dans plusieurs secteurs par des algorithmes [77], comme l'algorithme du protocole de la signature électronique sur courbe elliptique (ECDSA) et l'échange de clés sur courbe elliptique Diffie-Hellman (ECDH) qui sont recommandés pour assurer un bon niveau de sécurité [20].

Plusieurs chercheurs dans le monde ont parlé de l'importance de ECC en matérielle pour proposer une cryptographie efficace [12, 65]. Cette thèse entre dans le cadre des implémentations matérielles efficaces et rapides des composantes pour ECC. L'opération primordiale de ECC est la multiplication scalaire [12] qui est notée k*P, avec k est un entier et P est un point de la courbe elliptique. Cette multiplication scalaire présente l'opération principale et la plus couteuse dans ECC, plus de détails de calcul dans le Chapitre 2.

Pour calculer efficacement k*P, plusieurs méthodes dans la littérature ont été proposées dans les dernières années [12,65]. Mais elles restent toujours un sujet très abordé dans les recherches d'aujourd'hui, car l'amélioration de ces performances représentent un défi intéressant. Présentant les différents niveaux arithmétiques d'im-

plémentations d'une multiplication scalaire, nous trouvons l'arithmétique des scalaires, l'arithmétique des points et l'arithmétique des corps finis. Pour améliorer les performances de ces trois niveaux de calcul, nous proposons dans cette thèse des solutions et des architectures pour l'arithmétique des corps finis. La figure 1.1 présente l'hiérarchie du calcul pour l'implémentation de ECC.

2.2 Définition de la cryptographie à base de couplage

Dans cette section nous allons définir la cryptographie à base du couplage [49], qui est aussi un cryptosystème à clé publique basé sur les courbes elliptiques. Nous pouvons considérer le couplage comme une extension de la courbe elliptique. Plusieurs chercheurs [37, 66, 79], depuis Boneh et Franklin [6], ont proposé des travaux permettant d'utiliser le couplage en cryptographie. Depuis 2001, l'intérêt de la cryptographie basée sur l'identité a commencé d'augmenter dans les récents travaux de recherche [6]. La cause principale de l'augmentation de cet intérêt est due à la puissance du couplage pour résoudre des problèmes en cryptographie. Plusieurs autres protocoles à base du couplage sont proposés dans les recherches récentes, par exemple les signatures courtes [5] et l'échange de Diffie Hellman à trois [37]. Dans [49] vous trouvez une introduction détaillée sur la cryptographie à base du couplage.

Le couplage est une opération à base de calculs mathématiques très compliqués, c'est pour cette raison ce calcul est plus couteux en le comparant avec les calculs dans ECC. Une multiplication scalaire de courbe elliptique est environ dix fois plus rapide qu'un calcul de couplage au niveau de sécurité de 128 bits sur des processeurs x86-64 [33, 74]. Donc l'optimisation du calcul pour le couplage est d'une grande importance. Suite à ce besoin, plusieurs efforts ont été fait pour implémenter des couplages plus rapides et efficaces [10, 33, 46]. Différentes techniques et méthodes (traitées dans la section 4 du Chapitre 2) permettant cette optimisation du couplage. Donc ces derniers temps la cryptographie basée sur le couplage a atteint beaucoup d'importance ainsi son implémentation matérielle qui doit être rentable, en terme de vitesse et de ressource.

3 Contribution

Cette thèse se concentre sur l'implémentation matérielle de plusieurs primitives cryptographiques, qui sont utilisés dans la cryptographie du couplage et de ECC. Pour développer nos architectures, nous utilisons des plateformes appropriées qui sont les cartes Field Programmable Gate Array (FPGA). Les architectures de base

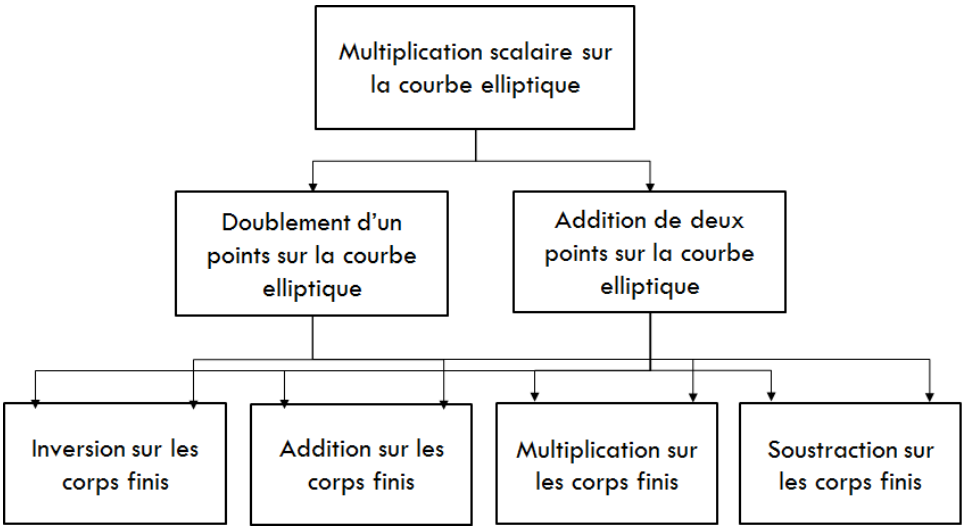

FIGURE 1.1 – Hiérrchie de l'implémentation du calcul pour la cryptographie à base de la courbe elliptique

les plus importantes de ECC et du couplage sont les architectures des arithmétiques de corps finis. La cryptographie à clé publique nécessite toutes les opérations des corps finis telles que l'addition, la soustraction, la multiplication, l'inversion et la division.

La multiplication dans le corps fini est l'opération la plus importante en cryptographie. Afin d'obtenir une conception efficace et performante, notre travail met l'accent sur la proposition d'une architecture efficace pour la multiplication modulaire, nous avons utilisé la multiplication modulaire de Montgomery (MMM) [57]. Grâce à des fonctionnalités intégrées offertes par des FPGA, nous avons pu proposer et réaliser des circuits optimisés, ainsi notre travail accélère les primitives de corps fini. Une de nos contributions est de proposer une architecture systolique, qui est supposée être le meilleur choix pour les implémentations matérielles, pour la méthode CIOS (Coarsely Integrated Operand Scanning) de la multiplication modulaire de Montgomery. Ce travail a été publié dans une conférence internationale avec actes et comité de sélection [52]. En utilisant notre implémentation matérielle MMM efficace, nous proposons une conception efficace pour les opérations ECC : l'addition de deux points et le doublement d'un point. Ce travail a été publié dans une autre conférence internationale avec actes et comité de sélection [51] Nous avons utilisé nos architectures sur les corps finis afin d'optimiser l'implémentation du couplage. Nos architectures proposées pour la multiplication modulaire, permettent d'avoir une implémentation efficace de l'arithmétique des extensions des corps finis. Dans ce travail nous proposons aussi une implémentation optimisée pour l'inversion et/ou

la division modulaire. Cette architecture matérielle de l'inversion permet de gagner en terme de ressources par rapport à des architectures proposées dans la littérature. Notre travail de l'inversion a été publié dans une conférence internationale avec actes et comité de sélection [50]. Nous avons proposé des simulations mathématiques pour les algorithmes du couplage de Ate optimale ainsi que les détails du calculs pour l'addition et le doublement des points sur la courbe elliptique. Ces algorithmes ont été le sujet d'une contribution dans [58].

4 Organisation de la thèse

Nous avons structuré le travail comme suit :

Le Chapitre 1 est une introduction générale pour cette thèse, dans ce chapitre nous présentons des définitions pour la cryptographie à clé publique, une motivation, nos contributions et nous terminons par l'organisation de la thèse.

Le Chapitre 2 donne un bref aperçu des techniques et des algorithmes associés aux opérations de la cryptographie à clé publique et des corps finis. Il inclut également des idées de base de la courbe elliptique et du couplage. Nous proposons à la fin de ce chapitre quelques travaux de la littérature pour les implémentations du couplage.

Le Chapitre 3 explore les fonctionnalités intégrées dans les FPGAs pour développer des primitives à grande vitesse pour les corps finis. Nous proposons dans ce chapitre une introduction pour les architectures systoliques. Nous proposons à la fin de ce chapitre une étude de l'existant qui traite les travaux réalisés avec une architecture systolique.

Le Chapitre 4 prend en détails l'algorithme de la multiplication de Montgomery qui a été modifié pour l'adapter à une architecture systolique, nous avons proposé une architecture sur $\mathbb{F}_p$ pour les grands nombres. Dans ce chapitre nous avons détaillé nos différentes versions proposées pour implémenter une multiplication modulaire. Ce chapitre présente les architectures internes pour chaque version proposée. Les résultats de ce chapitre sont publiés dans une conférence en cryptographie.

Le Chapitre 5 présente un cryptosystème à base de courbe elliptique et un crypto-

système couplage exploitant le concept de la multiplication que j'ai développé. Nous présentons aussi dans ce chapitre d'autres composants matériels nécessaires pour le couplage, telles que l'inversion modulaire, et l'arithmétique sur les extensions des corps finis.

Le chapitre 6 conclut la thèse et propose quelques orientations possibles pour les futures travaux.

Chapitre 2

La cryptographie à clé publique

1 Introduction

Nous présentons dans ce chapitre les bases mathématiques et les éléments théoriques dont nous nous sommes servis dans cette thèse. Nous commencerons ce chapitre par des définitions sur les corps finis, ensuite nous présenterons l'arithmétique des courbes elliptiques et nous terminerons par quelques définitions sur le couplage.

2 Les corps finis

Pour bien optimiser un cryptosystème, peu importe le système de coordonnées, nous devons faire le bon choix du corps fini ainsi que la manière d'implémenter les opérations arithmétiques dans ce corps. Un corps fini est une structure algébrique notée $\mathbb{F}_p$, où p est un nombre premier, nous proposons cette référence [70] pour plus de détails. Le corps fini premier $\mathbb{F}_p$ [18] est isomorphe à :

$$\mathbb{Z}/p\mathbb{Z} = \{0, 1, 2, ..., p-1\} \tag{2.1}$$

La caractéristique de $\mathbb{F}_p$ est l'entier premier p. Sachant que la caractéristique d'un corps est un entier n, avec n est l'entier non nul tel que le produit d'un élément du corps et n est nul [18]. Nous présenterons dans la section suivante l'arithmétique que nous avons utilisé dans la cryptographie asymétrique.

Définition 1. *Extension du corps $\mathbb{F}_p$,*
Pour que $\mathbb{F}_q$ soit un corps, il est nécessaire que q soit une puissance de p ($q = p^k$). Nous pouvons construire $\mathbb{F}_q$ avec le quotient $\mathbb{F}_p[X]/(P(X)\mathbb{F}_p[X])$. $\mathbb{F}_p[X]$ est un ensemble des polynômes avec les coefficients dans $\mathbb{F}_p$. $P(X)$ est le polynôme irréductible

sur $\mathbb{F}_p$ de degré k. Une extension du corps $\mathbb{F}_p$ de degré k est l'ensemble des polynômes de coefficients dans $\mathbb{F}_p$ et de degré inférieur strictement à k [18].

L'arithmétique des extensions de corps peut se construire via des tours d'extension. Nous présenterons la construction des tours d'extensions dans la section 5.2

2.1 Arithmétique pour la cryptographie

Généralement, tout calcul d'opérations arithmétiques pour des cryptosystèmes à clé publique est effectué dans des structures mathématiques finies. Ces derniers sont les corps.

Dans les implémentations cryptographiques asymétriques nous utilisons l'arithmétique modulaire sur les corps finis. Des exemples de cette arithmétique sont la multiplication modulaire, l'addition modulaire, la soustraction modulaire et l'inversion modulaire [18, 25]. Il est indispensable que ces opérations soient définies dans un corps premier $\mathbb{F}_p$.

Pour garantir que le résultat de chaque opération modulaire appartient à l'intervalle $[0, p-1]$, nous effectuons une opération modulo p.
autrement dit, quelque soit l'opération effectuée de :

$\otimes : [0, p-1] \times [0, p-1] \longrightarrow [0, p-1]$

$a, b \longmapsto a \otimes b - qp$

avec $q = \lfloor \frac{a \otimes b}{p} \rfloor$

Dans le cas de l'addition modulaire, et puisque une addition $a + b < 2p$, la division est inutile. Alors nous pouvons déterminer la réduction avec une simple soustraction, ce qui conduit à un résultat dans l'intervalle $[0, p-1]$.

La multiplication modulaire est définie comme l'opération la plus coûteuse en ressources et en temps de calcul pour la cryptographie à clé publique après l'inversion, car $a \times b < p^2$, et q est en général de la taille de p.

Nous présenterons l'exemple suivant pour expliquer la manière de fonctionnement du calcul arithmétique dans les corps finis :

Exemple :

Soient les éléments de $\mathbb{F}_7$ qui sont {0, 1, 2, 3, 4, 5, 6}.
Voici quelques exemples d'opérations sur le corps $\mathbb{F}_7$:
Une multiplication modulaire : $4 \times 5 \bmod 7 = 20 \bmod 7 = 6$.
Une addition modulaire : $4 + 5 \bmod 7 = 9 \bmod 7 = 2$.

Les calculs cryptographiques nécessitent des opérations arithmétiques de ce genre. Nous présenterons par la suite quelques algorithmes de la littérature de l'arithmétique des corps finis.

2.2 Multiplication modulaire

– **Multiplication Modulaire de Montgomery (MMM)** : En 1985, Montgomery a proposé une méthode de multiplication modulaire très efficace avec un modulo p [57]. Cette multiplication est détaillée dans l'algorithme 1, en proposant un nouveau système de représentation des entiers. Pour le calcul dans les grandes caractéristiques sur les corps finis, l'algorithme de la multiplication de Montgomery consiste à calculer une multiplication modulaire sans nécessité de faire une division par le modulo p. En cryptographie, la multiplication de Montgomery est la plus utilisée afin d'effectuer $a \times b \bmod p$. Cette multiplication permet de transformer une division par plusieurs multiplications de puissance de la base de la numérotation. Une division par puissance de la base de numérotation est un simple décalage pour les implémentations matérielles et aussi pour les implémentations softwares.

La multiplication de Montgomery est réalisée avec des nombres représentés dans une autre base de représentation appelée la représentation de Montgomery. La conversion du domaine ordinaire au domaine de Montgomery est détaillée dans la Table 2.1

Soit p un entier premier et impaire et soit $n = \lceil log_2(p) \rceil$, nous avons choisi comme base de numérotation $R = 2^n$ avec $p < R$. Comme p et R sont premiers entre eux, nous pouvons calculer $p' = p^{-1} \bmod R$. Le choix de R est tel que $gcd(R, p) = 1$. La MMM est implémentée avec des nombres représentés dans la base de Montgomery.
La multiplication de Montgomery calcule M (a)× M (b) et donne M (ab) comme résultat.

Définition 2. *La représentation de Montgomery de tout entier a est M(a) = a.R mod p.*
$M : a \in \mathbb{F}_p \rightarrow aR \in \mathbb{F}_p$ est une bijection dans $\mathbb{F}_p$.
$M(a) \pm M(b)$ mod $p = aR \pm bR = (a \pm b)R^n$ mod p.
La Table 2.1 présente la conversion du domaine ordinaire au domaine de Mont-

Algorithme 1 : Multiplication Modulaire de Montgomery

Input : p entier premier et impair, $n = \lceil log_2(p) \rceil$, $R = 2^n$, $p' = -p^{-1} mod$ R, $M(a)$, $M(b) \in \mathbb{F}_p$

Output : $M(ab) \mod p$

1 $\gamma \leftarrow M(a) \times M(b)$
2 $\delta \leftarrow \gamma \times p' \mod R$
3 $T \leftarrow \frac{\gamma + \delta \times p}{R}$
4 **If** $T \geq p$ **then** $T \leftarrow T - p$
5 **return** T

gomery.

Domaine ordinaire	$\Longleftrightarrow$	Domaine de Montgomery
a	$\longleftrightarrow$	M(a)=a·R mod p
b	$\longleftrightarrow$	M(b)=b·R mod p
a·b	$\longleftrightarrow$	M(a·b)=a·b·R mod p

TABLE 2.1 – Conversion du domaine ordinaire au domaine de Montgomery

Koç et al. dans [41] ont proposé plusieurs méthodes pour implémenter la MMM :

1. **Méthode SOS : Separated Operand Scanning** :
   C'est le premier algorithme présenté dans leur travail pour calculer la multiplication de Montgomery. Avec cette méthode ils commencent leur algorithme avec la multiplication de deux opérandes $a \times b$ en utilisant cet algorithme

```
1 for i=0 to s-1
2     C:=0
3     for j=0 to s-1
4           (C,S):= t[i+j] + a[j]*b[i] + C
5           t[i+j]:=S
6       t[i+s]:=C
```

   La valeur finale obtenue est sur 2s-1 mots, t est présenté par les mots : t[0], t[1], ... t[2s-1]. Après le calcul de la multiplication ils effectuent cet algorithme pour commencer la réduction :

```
1   for i=0 to s-1
2       C:=0
3       m:= t[i]*n'[0] mod W
4       for j=0 to s-1
5           (C,S):= t[i+j] + m*n[j] + C
6           t[i+j]:=S
7       ADD(t[i+s],C)
```

La fonction ADD est nécessaire pour le dernier mot du tableau t, afin de garantir que la taille du tableau ne dépasse pas le 2s mots. Par la suite cette méthode nécessite une division par r, en effectuant le simple décalage.

```
1   for j=0 to s
2       u[j]:= t[j+s]
```

Donc nous obtenons le résultat u de s+1 mots. La méthode SOS nécessite 2s+2 mots pour traiter les résultats intermédiaire de l'algorithme. Une partie de multi-précision est ajoutée à la fin si nécessaire. Pour plus de détails pour cette méthode un exemple est présenté dans [41].

2. **Méthode CIOS : Coarsely Integrated Operand Scanning :**
   Nous avons présenté l'algorithme de la méthode CIOS dans le chapitre 4, car cet algorithme présente la base de nos implémentations dans ce chapitre. Cette méthode permet une intégration entre les étapes de la multiplication et les étapes de la réduction. C'est à dire au lieu de faire la multiplication $a \times b$ suivi d'une réduction, cette méthode permet une alternance entre les itérations de la multiplication et les itérations de la réduction.

```
1   for i=0 to s-1
2       C:=0
3       for j=0 to s-1
4          (C,S):= t[j] + a[j]*b[i] + C
5          t[j]:=S
6       (C,S):= t[s] + c
7       t[s]:=S
8       t[s+1]:=C
9       C:= 0
```

```
10      m:=t[0]*n'[0] mod W
11      for j=0 to s-1
12         (C,S):=t[j] + m*n[j] + C
13         t[j]:=S
14      (C,S):=t[s] + C
15      t[s]:= S
16      t[s+1]:=t[s+1] + C
17      for j=0 to s
18         t[j]:=t[j+1]
```

La dernière boucle de j est utilisée pour faire un décalage à droite du résultat afin d'effectuer la réduction. Une amélioration pour cette partie est proposée de la manière suivante :

```
1      m:=t[0]*n'[0] mod W
2      (C,S):=t[0] + m*n[0]
3      for j=1 to s-1
4        (C,S):=t[j] + m*n[j] + C
5        t[j]:=S
6      (C,S):=t[s] + C
7     t[s-1]:= S
8     t[s]:=t[s+1] + C
```

Dans cette méthode nous avons utilisé que s+2 mots, car le décalage se fait d'un mot au lieu de s mots. Par conséquence le résultat final est présenté par les s+1 premiers mots.

3. **Méthode FIOS : Finely Integrated Operand Scanning :**
   Cette méthode est une modification de la méthode CIOS. Cette méthode présente une intégration de deux bloucles internes de la méthode CIOS en une seule boucle.

```
1   for i=0 to s-1
2       (C,S):= t[0] + a[0]*b[i]
3       ADD(t[1],C)
4       m:= S*n'[0] mod W
5       (C,S):= S + m*n[0]
6       for j=1 to s-1
7          (C,S):= t[j] + a[j]*b[i] + C
8          ADD(t[j+1],C)
```

```
9           (C,S):= S + m*n[j]
10          t[j-1]:=S
11      (C,S):=t[s] +C
12      t[s-1]:= S
13      t[S]:=t[s+1] + C
14      t[s+1]:=0
```

La seule différence de la méthode FIOS par rapport à CIOS est que dans la méthode FIOS il y a une seule boucle interne. D'après les auteurs du [41] cette méthode nécessite : $2s^2$+s multiplications, $5s^2$+3s+2 addition.

4. **Méthode FIPS : Finely Integrated Product Scanning :**
   Comme dans la méthode précédente, cette méthode permet une intégration dans le calcul de la multiplication et le calcul de la réduction. Voici la première boucle de l'algorithme qui permet d'effectuer cette méthode :

```
1   for i=0 to s-1
2       for j=0 to i-1
2           (C,S):= t[0] + a[j]*b[i-j]
3           ADD(t[1],C)
4           (C,S):= S + m[j]*n[i-j]
5           t[0]:=S
6           ADD(t[1],C)
7       (C,S):= t[0] + a[i]*b[0]
8       ADD(t[1],C)
9       m[i]:=S*n'[0] mod W
10      (C,S):= S + m[i]*n[0]
11      ADD(t[1],C)
12      t[0]:=t[1]
13      t[1]:=t[2]
14      t[2]:=0
```

La seconde boucle de cette méthode présentée ci-dessous, permet de compléter le calcul en fournissant le résultat final mot par mot.

```
1   for i=s to 2s-1
2       for j=i-s+1 to s-1
2           (C,S):= t[0] + a[j]*b[i-j]
```

```
3            ADD(t[1],C)
4            (C,S):= S + m[j]*n[i-j]
5            t[0]:=S
6            ADD(t[1],C)
7        m[i-s]:=t[0]
8        t[0]:=t[1]
9        t[1]:=t[2]
10       t[2]:=0
```

Cette méthode nécessite $2s^2$+s multiplications et $6s^2$+2s+2 additions. Et nous avons besoin de s+3 mots pour effectuer cette méthode.

5. **Méthode CIHS : Coarsely Integrated Hybrid Scanning :**
   Cette méthode est une modification de la méthode SOS. La méthode SOS nécessite 2s+2 mots pour effectuer le calcul intermédiaire. Mais cette méthode montre qu'il est possible d'utiliser que s+3 mots.

Ils ont recommandé, d'après leur article, la méthode CIOS comme la meilleure solution pour la multiplication sur le processeur choisi (Pentium 60), CIOS fonctionne plus rapidement que les autres algorithmes de multiplication Montgomery surtout en langage assembleur et que cependant, sur d'autres types de processeurs, un algorithme différent pourrait être préférable. Par exemple, sur un DSP, ils ont souvent trouvé que la méthode FIPS est meilleure car elle exploite l'architecture MAC (Multiply- Accumulate). En se basant sur leurs travaux nous avons choisi de travailler quand même avec la méthode CIOS, car à moyen terme on pense transposer notre VHDL sur ASIC en proposant une architecture orientée matériel embarqué pour laquelle il est plus indiqué d'utiliser des cellules élémentaires que d'y implanter des coeurs de DSP comportant des MACs.
Nos architectures sont présentées en détails dans le chapitre 4. La méthode CIOS permet d'améliorer l'algorithme de Montgomery en intégrant la multiplication et la réduction. Plus précisément au lieu de calculer la multiplication puis la réduction, cette méthode permet une alternance entre la multiplication et la réduction. Les entiers a, b et p sont présentés par s mots de taille w. Afin d'effectuer cet algorithme, nous avons utilisé un tableau de taille $s + 2$. Les résultats intermédiaires sont stockés dans le tableau T, et le résultat final est présenté par les $s + 1$ premières cases du même tableau.

– **Multiplication de Blakely** En 1983, Blakely a introduit un des algorithmes de la multiplication modulaire les plus simples. Blakley a adapté la méthode doublement-et-addition classique. Cette méthode consiste à intégrer des réductions après chaque étape de calcul. Cet algorithme utilise une représentation classique des nombres et fonctionne pour tous les modulos. L'algorithme 2 est adapté au fonctionnement du schéma de multiplication de type doublement-et-addition. Nous avons $(a_{N-1}, ..., a_0)$ la représentation binaire de l'opérande a. Calculer la multiplication $a \times b$ revient à calculer :

$$a \times b = 2(...(2(_{aN-1} \times b) + a_{N-2}) + ...) + a_0 \times b. \tag{2.2}$$

Le calcul présenté dans 2.2 est un parcours des poids forts et des poids faibles. Pour effectuer la multiplication de Blakely, nous multiplions chaque résultat intermédiaire par 2 pour tout a_i, et nous ajoutons un b si a_i= 1. Tout au long de l'exécution de l'algorithme, le résultat intermédiaire est toujours inférieur à $2p$. Car cet algorithme effectue les réductions à chaque étape. Chaque réduction est effectuée avec une simple soustraction.

Algorithme 2 : Multiplication de Blakely

Input : p < 2n, a, b < p avec a = $(a_{N-1}, ..., a_0)_2$
Output : $a \cdot b$ mod p

```
1 r ← Null ;
2 for i ← N − 1 to 0 do
3 |   r ← 2 · r ;
4 |   if r ≥ p then
5 |   |   r ← r − p ;
6 |   r ← r + a_i · b ;
7 |   if r ≥ p then
8 |   |   r ← r − p ;
9 return r ;
```

2.3 Inversion modulaire

L'opération d'inversion modulaire a^{-1} mod p d'un entier a existe si et seulement si les deux entiers a et p sont premiers, c'est à dire leur $pgcd(a, p) = 1$. Il y a deux méthodes d'inversion qui sont toujours utilisées dans la littérature : Le petit théorème de Fermat et une variante de l'algorithme d'Euclide étendu (AEE). Il existe plusieurs variantes de ces algorithmes révélées dans des recherches précédentes, la plupart d'entre eux sont discutées et énumérées dans [12]. L'une des

variantes puissante de l'AEE pour l'inversion $\mathbb{F}_p$ basée sur la méthode binaire, qui est connue comme l'algorithme d'inversion binaire est présenté dans l'algorithme 3.

Algorithme 3 : Inversion/Division Binaire sur $\mathbb{F}_p$ [26]

Input : $a \in \mathbb{F}_p$

Output : a^{-1} mod p

```
1  u ← a, v ← p, x1 ← b, x2 ← 0.
2  while u ≠ 1 et v ≠ 1 do
3      while u est pair do
4          u ← u/2.
5          if x1 est pair then
6              x1 ← x1/2.
7          else
8              x1 ← (x1 + p)/2.
9      while v est pair do
10         v ← v/2.
11         if x2 est pair then
12             x2 ← x2/2.
13         else
14             x2 ← (x2 + p)/2.
15     if u ≥ v then
16         u ← u + v.
17         x1 ← x1 + x2.
18     else
19         v ← v + u.
20         x2 ← x2 + x1.
21 if u = 1 then
22     return x1 mod p;
23 else
24     return x2 mod p;
```

Nous présentons ici une idée brève sur l'utilisation d'une inversion dans la cryptographie à clé publique. Nous utilisons l'inversion soit dans le calcul de la multiplication scalaire soit dans le calcul du couplage. Ces deux cryptosystèmes serons présentés respectivement par la suite dans la section 3 et la section 4 .

Inversion dans ECC : Différents systèmes de coordonnées peuvent être utilisés pour représenter des points sur une courbe elliptique, et il est possible d'éliminer l'inversion en représentant les points en coordonnées projectives ou jacobiennes. Cependant, cette élimination présente l'inconvénient d'ajouter des opérations supplémentaires comme la multiplication et le carré. On considère souvent que l'inversion est beaucoup plus coûteuse que les opérations supplémentaires. En conséquence, des multiplications scalaires efficaces sur une courbe elliptique sont effectuées en coordonnées projectives ou Jacobiennes. Donc une optimisation des implémentations de l'inversion modulaire, permet d'éviter le changement de système de coordonnées. Nous pouvons donc utiliser les coordonnées affines pour ECC.

Inversion dans le couplage : Le calcul d'un couplage est composé de deux grandes parties, l'algorithme de Miller et l'exponentiation finale. Le calcul de l'exponentiation finale détaillé après dans la section 12 est aussi coûteux comme l'algorithme de Miller. Ainsi, plusieurs méthodes ont été proposées pour améliorer cette étape. Scott et all. dans [53] proposent de subdiviser l'exponentiation finale en 3 étapes, en utilisant la formule suivante :

$$e = \frac{p^{12}-1}{r} = (p^6-1)\cdot(p^2+1)\cdot\frac{p^4-p^2+1}{r} \tag{2.3}$$

Une inversion sur $\mathbb{F}_{p^{12}}$ est nécessaire pour calculer f^{p^6-1}. En fait, cette inversion est effectuée dans $\mathbb{F}_p$ une fois que nous transformons l'équation considérant que $f^6 = \bar{f}$ est un conjugué de f. En conséquence, nous pouvons calculer $f^{(p^6-1)} = \bar{f}\cdot f^{-1}$. Alors pour effectuer f^{-1} qui coûte une inversion modulaire sur $\mathbb{F}_{p^{12}}$ nous avons besoin d'une inversion modulaire sur $\mathbb{F}_p$.

3 Définition et propriétés des courbes elliptiques

3.1 Généralités sur les courbes elliptiques

De nos jours les courbes elliptiques sont très utilisées pour obtenir des primitives pour la cryptographie à clé publique. Comme nous l'avons dit dans l'introduction générale, ces courbes ont été proposées par Miller [55] et Koblitz [40] en 1985.

Définition 3. *Une courbe elliptique sur $\mathbb{K}$ notée $E(\mathbb{K})$, où $\mathbb{K}$ désigne un corps commutatif, est une courbe algébrique non-singulière qui est définie par l'équation*

cubique suivante :

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \tag{2.4}$$

avec $a_1, a_2, a_3, a_4, a_6 \in \mathbb{K}$ *et* $\Delta \neq 0$, *où* Δ *est le discriminant de la courbe calculée par les équations suivantes :*

$\Delta = -b_2^2b_8 - 8b_4^3 - 27b_6^2 + 9b_2b_4b_6$

$b_2 = a_1^2 + 4a_2$

$b_4 = 2a_4 + a_1a_3$

$b_6 = a_3^2 + 4a_6$

$b8 = a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2$

Nous distinguons deux types de courbes elliptiques : les courbes supersingulières et les courbes non-supersingulières [36].

Définition 4. *Soit la courbe elliptique E définie sur* $\mathbb{F}_q$ *de caractéristique p. Une courbe elliptique est dite surpersingulière s'il existe une des conditions suivantes [18] :*

1. *#E(*$\mathbb{F}_q$*)* $\equiv$ *1 mod p, ou #E(*$\mathbb{F}_q$*)=q+1 mod (p), ou t=0.*
2. *E n'admet pas de point d'ordre p sur* $\bar{\mathbb{F}}_q$

L'équation (2.4) peut être simplifiée [12, 36, 81]. Il y a plusieurs cas possibles de simplification. Nous pouvons simplifier cette équation selon la caractéristique du corps utilisé. Dans la suite nous présentons trois cas de simplification qui existe :

- Le cas de simplification si la caractéristique du corps notée $Car(\mathbb{K}) > 3$ (par exemple soit $\mathbb{K} = \mathbb{F}_{23}$ avec $p = 23$ qui est normalement un grand nombre premier), alors $E(\mathbb{K})$ est isomorphe à une courbe $E^1(\mathbb{K})$ donnée par :

$$E^1 : y^2 = x^3 + ax + b, \tag{2.5}$$

  Avec $\Delta = -16(4a^3 + 27b^2) \neq 0$ et $j(E) = 1728a^3/4\Delta$, $j(E)$ est le j-invariant de E.
- Si la caractéristique du corps $Car(\mathbb{K}) = 3$ donc $E(\mathbb{K})$ est isomorphe à une courbe elliptique $E^2(\mathbb{K})$ donnée par :
  - Cas non supersingulier

$$E^2 : y^2 = x^3 + ax^2 + b, \tag{2.6}$$

avec $ab \neq 0$.

- Cas supersingulier

$$E^2 : y^2 = x^3 + ax + b, \tag{2.7}$$

avec $a \neq 0$.

- Si la caractéristique du corps $Car(\mathbb{K}) = 2$, alors $E(\mathbb{K})$ est isomorphe à une courbe $E^3(\mathbb{K})$ donnée par :
  - Cas non supersingulier

$$E^3 : y^2 + xy = x^3 + ax^2 + b \tag{2.8}$$

avec $b \neq 0$.

  - Cas supersingulier

$$E^3 : y^2 + cy = x^3 + ax + b \tag{2.9}$$

avec $c \neq 0$.

Remarque 1 : Les courbes elliptiques [12, 65] définies sur des corps de caractéristiques 2 et 3 ne sont pas étudiées dans cette thèse. Sachant que ces courbes sont cryptographiquement peu sûres car elles sont vulnérables aux attaques par couplage [37]. Depuis 2013 et suite aux attaques proposées par Antoine Joux, ces corps de petites caractéristiques ne sont plus considérés sûrs, même pour les couplages. Les attaques sont proposées sur le corps binaire. Pour plus de détails nous vous proposons cette référence [67].

Les points de la courbe sur un corps $\mathbb{K}$ ont pour coordonnées (x, y) qui sont les solutions de l'équation de la courbe, ainsi nous y ajoutons le point à l'infini noté O. L'équation de la courbe elliptique définie est mise sous une forme plus simple c'est l'équation de Weierstrass de [36] :

$$y^2 = x^3 + ax + b \tag{2.10}$$

Avec $\Delta = -16(4a^3 + 27b^2) \neq 0$. La figure 2.1 représente la courbe elliptique définie par l'équation $y^2 = x^3 - 18x + 20$ sur $\mathbb{R}$ et la figure 2.2 illustre une courbe définie sur le corps fini $\mathbb{F}_{23}$ de la même équation.

Proposition 3.1. *Soit la courbe elliptique $E(\mathbb{K})$ avec $Car(\mathbb{K}) \geq 3$. $E(\mathbb{K})$ défini par l'équation :$y^2 = x^3 + ax + b$. $E(\mathbb{K})$ représente un groupe pour la loi de cette composition interne :*

1. *$Q + O = O + Q = Q$ pour tout point $Q \in E(\mathbb{K})$.*

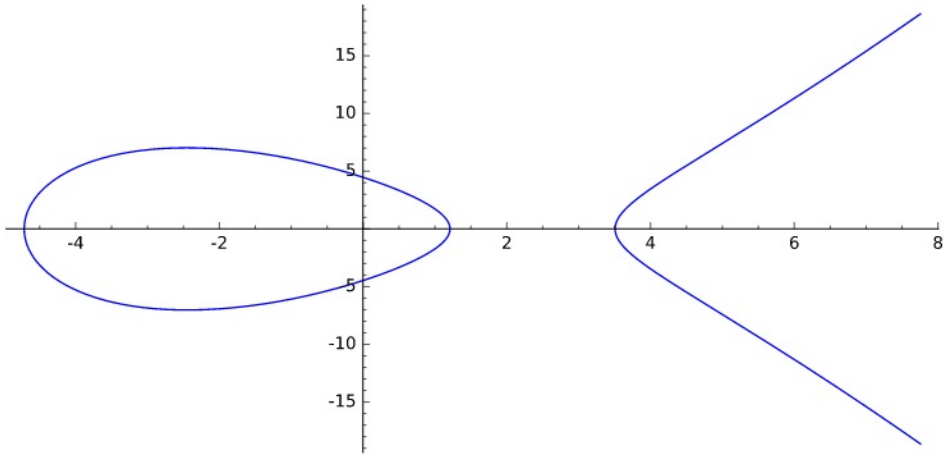

FIGURE 2.1 – La courbe elliptique définie par l'équation $y^2 = x^3 - 18x + 20$ sur $\mathbb{R}$

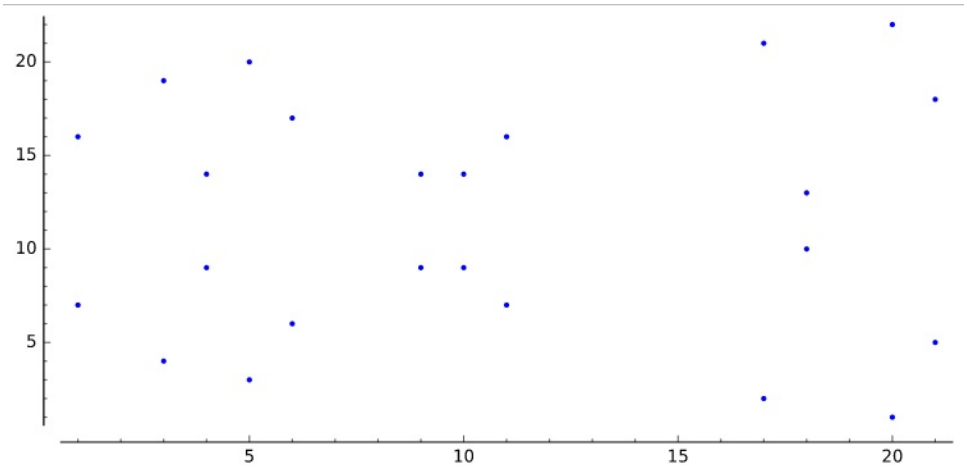

FIGURE 2.2 – La courbe elliptique définie par l'équation $y^2 = x^3 - 18x + 20$ sur $\mathbb{F}_{23}$

2. *Soient Q_1 et Q_2 deux points sur la courbe elliptique, avec $Q_1 = (x_1, y_1)$ et $Q_2 = (x_2, y_2)$. Tel que $Q_1 \neq -Q_2$, alors $Q_1 + Q_2 = Q_3$ avec $Q_3 = (x_3, y_3)$. Les coordonnées de Q_3 x_3 et y_3 sont définies de la manière suivante :*

$$x_3 = \lambda_2 - x_1 - x_2 \tag{2.11}$$

$$y_3 = \lambda(x_1 - x_3) - y_1 \tag{2.12}$$

où $\lambda = \dfrac{y_2 - y_1}{x_2 - x_1}$ *si* $Q_1 \neq Q_2$,
et
$\lambda = \dfrac{3x_1^2 + a}{2y_1}$ *sinon.*
Les calculs des équations sont en coordonnées affines.

3. *Soit $Q \in E(\mathbb{K})$ avec Q=(x,y), nous définissons l'inverse de Q par -Q =(x,-y).*

Pour évaluer le coût du calcul dans ECC, nous devons évaluer le calcul des opérations de base, qui sont l'addition et le doublement. Le coût d'une addition de deux points est donc : une inversion, deux multiplications, et un carré sur le corps $\mathbb{K}$ (I+2M+C), avec, l'inversion (I), la multiplication (M) et le carré (C). Le doublement nécessite (I+2M+2C) [48].

3.2 Multiplication scalaire

Pour définir la multiplication scalaire [8, 12] d'un point P par un entier $k \in \mathbb{N}$, nous devons définir la loi de l'addition sur les courbes. La multiplication scalaire est définie par :

$$\begin{aligned} &\mu : E \longrightarrow E \\ &P \longmapsto kP = \underbrace{(P + P + ... + P)}_{kfois}. \end{aligned} \tag{2.13}$$

Une multiplication scalaire présente l'opération essentielle pour tous les protocoles de la cryptographie qui utilisent les courbes elliptiques ou les couplages [37]. Peu importe la méthode, nous utilisons toujours les formules présentées dans 2.13. Un inconvénient très connu dans chaque calcul de la multiplication, c'est l'utilisation des inversions modulaires sur les corps finis. Cette inconvénient apparait si nous travaillons avec les coordonnées affines.

Dans les implémentations matérielles [1] et logicielles une inversion est beaucoup plus coûteuse qu'une multiplication modulaire ou bien une addition modulaire. Pour éviter l'utilisation des inversions dans les protocoles cryptographiques, plusieurs chercheurs ont proposé d'autres systèmes de coordonnées [48], comme le système de coordonnées projectives. Ces dernières permettent de remplacer une inversion dans le calcul avec plusieurs multiplications. Nous détaillerons le changement des coordonnées dans la section 3.3.

Dans ce qui suit, l'ensemble des points d'une courbe elliptique E défini sur $\mathbb{F}_p$ est noté $E(\mathbb{F}_p)$.

Soit la courbe elliptique $E(\mathbb{F}_p)$ définie sur $\mathbb{F}_p$, avec $\mathbb{F}_p$ est un corps fini et p est un entier premier. Soit un point P d'ordre r avec $P \in E(\mathbb{F}_p)$. Le sous groupe cyclique de $E(\mathbb{F}_p)$ de générateur P est $O, P, 2P, ...(r-1)P$. Avec un scalaire k défini comme entier dans l'intervalle [1, r-1], nous effectuons l'opération principale du ECC, c'est la multiplication scalaire. Elle est notée de la manière suivante : Q = k P. sachant que le résultat Q est aussi un point du sous groupe de générateur P.

La difficulté des systèmes basés sur la multiplication scalaire repose sur le problème du logarithme discret (Elliptic Curve Discrete Logarithm Problem (ECDLP)).

3.3 Choix du système des coordonnées

Nous détaillons ici les différentes coordonnées utilisées pour les courbes elliptiques définies sur des corps finis.

Système des coordonnées Affines

Dans la représentation en coordonnées affines, un point $P \in E(\mathbb{F}_p)$ est présenté par deux coordonnées x et y $\in \mathbb{F}_p$ satisfaisant l'équation (2.4)

Système des coordonnées Projectives

Un point Q de la courbe elliptique, en coordonnées projectives, est représenté par le triplet (X :Y :Z). Ce dernier correspond au point présenté en coordonnées affines représenté par le couple (X/Z :Y/Z) pour $Z \neq 0$. L'équation de la courbe elliptique sera de la manière suivante :

$$Y^2Z = X^3 + aXZ^2 + bZ^3 \tag{2.14}$$

Avec ce système des coordonnées l'inverse du point P= (X :Y :Z) est présenté par $-P$= (X :-Y :Z). Le point à l'infini O est représenté par ce triplet (0 :1 :0).

Voir les références [8, 12] pour plus de détails sur les formules de composition des coordonnés d'un point avec ce système. Comme nous l'avons signalé précédemment, avec ce système de coordonnées projectives nous n'utilisons pas des inversions modulaires sur les corps finis. Donc une addition de deux points sur la courbe nécessite 12M+2C, et un doublement nécessite 7M+5C. Par contre nous avons besoin d'une inversion pour le résultat de la multiplication scalaire, pour passer des coordonnées projectives aux coordonnées affines. Voici les formules pour calculer les coordonnées projectives dans $\mathbb{F}_p$:
Cas du doublement d'un point :

$$X_3 = 2Y_1Z_1((aZ_1^2 + 3X_1^2)^2 - 8X_1Y_1^2Z_1). \tag{2.15}$$

$$Y_3 = (aZ_1^2 + 3X_1^2)(4X_1Y_1^2Z_1 - ((aZ_1^2 + 3X_1^2)^2 - 8X_1Y_1^2Z_1)) - 8Y_1^4Z_1^2. \tag{2.16}$$

$$Z_3 = 8Y_1^3Z_1^3. \tag{2.17}$$

Cas de l'addition de deux points :

$$C = ((Y_2Z_1 - Y_1Z_2)^2Z_1Z_2 - (X_2Z_1 - X_1Z_2)^3 - 2(X_2Z_1 - X_1Z_2)^2X_1Z_2). \quad (2.18)$$

$$X_3 = (X_2Z_1 - X_1Z_2)C. \quad (2.19)$$

$$Y_3 = (Y_2Z_1 - Y_1Z_2)((X_2Z_1 - X_1Z_2)^2X_1Z_2 - C) - (X_2Z_1 - X_1Z_2)^3Y_1Z_2. \quad (2.20)$$

$$Z_3 = (X_2Z_1 - X_1Z_2)^3Z_1Z_2. \quad (2.21)$$

Système des coordonnées Jacobiennes

Un point P de la courbe elliptique en coordonnées jacobiennes est représenté par le triplet (X, Y, Z) [48]. Ce dernier correspond au point présenté en coordonnées affines de la manière suivante :$(X/Z^2 : Y/Z^3)$ avec $Z \neq 0$. L'équation de la courbe elliptique devient sous cette forme :

$$Y^2 = X^3 + aXZ^4 + bZ^6. \quad (2.22)$$

L'inverse d'un point P présenté par le triplet (X,Y,Z) est le point présenté par le triplet (X,-Y,Z), et le point O à l'infini est représenté par le triplet (1,1,0).

Le calcul qui suit présente l'addition de deux points en coordonnées Jacobiennes : Soient $P_1 = (X_1, Y_1, Z_1)$, $P_2 = (X_2, Y_2, Z_2)$ tels que $P_1 \neq \pm P_2$ et $P_1 + P_2 = (X_3, Y_3, Z_3)$.
Alors soient :

$$A = X_1Z_2^2, B = X_2Z_1^2, C = Y_1Z_3^2, D = Y_2Z_1^3, E = B - A, F = D - C. \quad (2.23)$$

nous avons

$$X_3 = -E^3 - 2AE^3 + F^2. \quad (2.24)$$

$$Y3 = -CE^3 + F(AE^2 - X_3). \quad (2.25)$$

$$Z3 = Z_1Z_2E. \quad (2.26)$$

Le calcul qui suit présente le doublement d'un point en coordonnées Jacobiennes : Soient $P_1 = (X_1 : Y_1 : Z_1)$ et $P_3 = 2P_1 = (X_3 : Y_3 : Z_3)$.
alors soient :

$$A = 4X_1Y_1^2, B = 3X_1^2 + aZ_1^4. \quad (2.27)$$

nous avons

$$X_3 = -2A + B^2. \quad (2.28)$$

$$Y_3 = -8Y_1^4 + B(A - X_3). \tag{2.29}$$

$$Z3 = 2Y_1Z_1. \tag{2.30}$$

Ce calcul montre que le coût d'une addition dans le système des coordonnées Jacobiennes est de 12 multiplications et de 4 carrés, alors que le coût d'un doublement est de 4 multiplications et de 6 carrés. Donc une addition dans le système de coordonnées jacobiennes nécessite plus d'opérations q'une addition dans le système à coordonnées projectives, mais inversement pour le doublement.

4 Calcul du couplage

4.1 Généralités sur les couplages

Dans cette section nous présentons les définitions importantes du couplage [49].

Définition 5. *Soient G_1 et G_2 deux groupes abéliens et soit G_3 un groupe multiplicatif commutatif. Les groupes doivent être du même ordre. Un couplage est une application*

$$e : G_1 \times G_2 \to G_3 \tag{2.31}$$

munie des propriétés suivantes :

- *Non-dégénérescence : soit $P \in G_1$ et $Q \in G_2$,*
  si $\forall P \in G_1$, $e(P,Q) = 1$ alors $P = 0$
  et
  si $\forall Q \in G_2$, $e(P,Q) = 1$ alors $Q = 0$.
- *Bilinéarité : soit $P \in G_1$ et $Q_1, Q_2 \in G_2$ alors*
  $e(P, Q_1 + Q_2) = e(P, Q_1) \times e(P, Q_2)$
  Plus généralement,
  $\forall\ x,\ y$ *et* $z \in \mathbb{Z}$
  $e(xP, yQ) = e(P, Q)^{xy}$,
  $e(xP, yQ)^z = e(yP, zQ)^x = e(zP, xQ)^y = e(P, Q)^{xyz}$

Définition 6. *Degré de plongement de la courbe elliptique [18, 24] :*
Soit la courbe elliptique $E(\mathbb{F}_p)$, soit r un diviseur premier de la cardinalité du courbe ($\#E(\mathbb{F}_p)$). Le degré de plongement est le plus petit entier k tel que r divise (p^k-1).

Dans ce qui suit nous présenterons quelques exemples du couplage.

4.2 Couplage de Weil

Le Couplage de Weil est calculé de cette manière :

Proposition 4.1. *Si D est un diviseur de degré zéro sur la courbe E, donc il existe un point P sur la courbe avec $D \sim (P)$ - (0_∞).*
$E[r]$ est la r-torsion de la courbe elliptique E, k est le degré de plongement de la courbe E définie sur le corps fini $\mathbb{F}_p$ noté $E(\mathbb{F}_p)$. Soient $D1$ et $D2$ deux diviseurs de degré 0, $supp(D1) \cap supp(D2) \neq \varnothing$. $rD1 \sim 0$ et $rD2 \sim 0$. Pour plus de détail nous proposons [19].

Le couplage de Weil peut être présenté avec l'application suivante :

$$ew : E(\mathbb{F}_p)[r] \times E(\mathbb{F}_{p^k})/rE(\mathbb{F}_{p^k}) \longrightarrow {\mathbb{F}^*}_{p^k}$$

$$(P, Q) \longmapsto (-1)^r f_{r,p}(Q)/f_{r,Q}(P) \tag{2.32}$$

Pour effectuer un couplage de Weil, nous devons calculer un Miller Lite $f_{r,p}(Q)$ et un Miller Full $f_{r,Q}(P)$.

4.3 Couplage de Tate

Le couplage de Tate est défini avec les mêmes paramètres que Weil, E, $\mathbb{F}_p$,r et k. Soit $\mathbb{F}_{p^k}$ l'extension du corps $\mathbb{F}_p$ tel que $E[r] \subset \mathbb{F}_{p^k}$. Le couplage de Tate est défini de la manière suivante :

$$ew : E(\mathbb{F}_p)[r] \times E(\mathbb{F}_{p^k})/rE(\mathbb{F}_{p^k}) \longrightarrow {\mathbb{F}^*}_{p^k}$$

$$(P, Q) \longmapsto [f_{r,p}(Q)]^{((p^k-1)/r)} \tag{2.33}$$

Afin de calculer le couplage de Tate nous faisons $log2(r)$ itérations pendant l'algorithme de Miller, où r est l'ordre des sous groupes utilisés.

4.4 Couplage de Ate

Le couplage de Ate proposé dans [22] est présenté avec l'application suivante :

$$\mathbb{G}_1 = E[r] \cap Ker(\pi p - [1]) = E(\mathbb{F}_p)[r],$$

$$\mathbb{G}_2 = E[r] \cap Ker(\pi p - [p])$$

$$eA : \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow F^*_{p^k};$$

$$(P,Q) \longmapsto [f_{T,Q}(P)]^{((p^k-1)/r)} \tag{2.34}$$

Ce couplage est détaillé dans [18].

Définition 7. *Soit la fonction f :P $\longrightarrow$ I*

Ker(f)=p∈P tel que f(p)=0

L'avantage principal par rapport au couplage de Tate est la réduction du nombre d'itérations faite lors de l'algorithme de Miller.

log2(T) où T = t - 1, et t est la trace du Frobenius sur E($\mathbb{F}_p$).

L'inconvénient du couplage Ate c'est qu'il nécessite une application de Miller Full.

4.5 Couplage de Ate-Optimale

Le couplage Ate-Optimal, présenté dans cette référence [78], est une amélioration du couplage de Ate en réduisant le nombre d'itérations dans la boucle de l'algorithme de Miller qui calcule la fonction $f_{\lambda,Q}$(P), avec $\lambda = 6x + 2$. L'algorithme 4 présente des différentes étapes pour calculer le couplages de Ate-Optimle. Ce couplage est défini par :

$$\begin{aligned} &e : \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_3 \\ &(Q,P) \longrightarrow [f_{6x+2,Q}(P) \cdot F(P)]^{\frac{P^{12}-1}{r}} \end{aligned} \tag{2.35}$$

avec $F(P) = l_{[6x+2]Q,\pi_p(Q)}(P) \cdot l_{[6x+2]Q+\pi_p(Q),-\pi_p^2(Q)}(P)$.
Sachant que $l_{[6x+2]Q,\pi_p(Q)}(P)$ et $l_{[6x+2]Q+\pi_p(Q),-\pi_p^2(Q)}(P)$ deux équations détaillées dans l'algorithme 4 et le calcul est effectué dans les parties suivantes.

Tel que π_p est le Frobenius sur la courbe elliptique, c'est l'endomorphisme :

$$\begin{aligned} &\pi_p : \mathbb{E} \longrightarrow \mathbb{E} \\ &\pi_p(x;y) = (x^p;y^p). \end{aligned} \tag{2.36}$$

$$\begin{aligned} \mathbb{G}_1 &= E[r] \cap Ker(\pi p - [1]) = E(\mathbb{F}_p)[r] \\ \mathbb{G}_2 &= E[r] \cap Ker(\pi p - [p]) \\ \mathbb{G}_3 &= F_{p^k}^* \end{aligned} \tag{2.37}$$

Nous allons présenté le calcul du doublement d'un point de la courbe et le calcul de l'addition de deux points de la courbe, qui sont les calculs principaux dans un algorithme du couplage de Ate-Optimal. Ces deux calculs sont la base de l'algorithme de Miller dans un couplage.

Algorithme 4 : Couplage de Ate-Optimal sur les courbes BN

Input : $P \in G_1 \in E(\mathbb{F}_q)$ and $Q \in G_2 \in E'(\mathbb{F}_p^{k/6})$

Output : Couplage (Q,P)

$f \leftarrow 1\,; T \leftarrow Q\,;$
for $i \leftarrow L-2$ ***downto*** 0 **do**
  $f \leftarrow f^2 \cdot l_{T,T}(P)\,; T \leftarrow 2T\,;$
  if $s_i = -1$ **then**
    $f \leftarrow f \cdot l_{T,-Q}(P)\,; T \leftarrow T - Q\,;$
  else if $s_i = 1$ **then**
    $f \leftarrow f \cdot l_{T,Q}(P)\,; T \leftarrow T + Q\,;$
$Q_1 \leftarrow \pi_p(Q)\,; Q_2 \leftarrow \pi_p(Q_1)\,;$
$f \leftarrow f \cdot l_{T,Q_1}(P)\,; T \leftarrow T + Q_1\,;$
$f \leftarrow f \cdot l_{T,Q_2}(P)\,; T \leftarrow T - Q_2\,;$
$f \leftarrow f^{(p^{12}-1)/r}\,;$
return $f\,;$

Afin d'implémenter un couplage, nous devons implémenter les deux opérations de base (Addition et doublement). Dans ce qui suit nous donnerons le calcul nécessaire pour effectuer ces deux opérations.

Doublement d'un Point de la courbe et évaluation en un point P

Soit un point $Q = (X_Q, Y_Q, Z_Q)$ représenté par les coordonnées Jacobiennes qui appartient au twiste de la courbe, $Q \in E'(\mathbb{F}_{q^2})$. Donc le doublement de T=2Q $= (X_T, Y_T, Z_T)$ est défini par les équations suivantes :

$$X_T = 9X_Q^4 - 8X_Q Y_Q^2.$$

$$Y_T = 3X_Q^2(4X_Q Y_Q - X_T) - 8Y_Q^4.$$

$$Z_T = 2Y_Q Z_Q.$$

Soit le point $P \in E(\mathbb{F}_q)$ représenté dans les coordonnées affines tel que $P = (x_P, y_P)$, donc l'évaluation en P de l'équation du tangente dans le point T est la suivante :

$$l_{T,T}(P) = 4Y_T Z_T^3 y_P - (6X_T^2 Z_T^2 x_P)w + (6X_T^3 - 4Y_T^2)w^2 \in \mathbb{F}_{q^{12}}.$$

Addition des deux Points de la courbe et évaluation en un point P

Soit les points $Q = (X_Q, Y_Q, Z_Q)$ et $T = (X_T, Y_T, Z_T)$ qui appartiennent au twiste de la courbe, Q et T $\in E'(\mathbb{F}_{p^2})$ représentés en coordonnées Jacobiennes. D'où, l'addition R de T et Q est $R = (X_R, Y_R, Z_R)$ et est définie par les équations comme suit :

$$X_R = (2Y_Q Z_T^3 - 2Y_T)^2 - 4(X_Q Z_T^2 - X_T)^3 - 8(X_Q Z_T^2 - X_T)^2 X_T.$$

$$Y_R = (2Y_Q Z_T^3 - 2Y_T)(4(X_Q Z_T^2 - X_T)2X_T - X_R) - 8Y_T(X_Q Z_T^2 - XT)3.$$

$$Z_R = 2Z_T(X_Q Z_T^2 - X_T).$$

Soit le point $P \in E(\mathbb{F}_q)$ représenté dans les coordonnées affines tel que $P = (x_P, y_P)$, donc l'évaluation en P de l'équation du droite passant par les points T et Q est la suivante :

$l_{T,Q}(P) = 4Z_T(X_Q Z_T^2 - X_T)y_P - 4x_P(Y_Q Z_T^3 + Y_T)w + (4X_Q(Y_Q Z_T^3 X_Q - Y_T) - 4Y_Q Z_T(X_Q Z_T^2 - X_T))w^2 \in \mathbb{F}_{p^{12}}$.

Exponentiation finale

La ligne 11 de l'algorithme 4 effectue l'exponentiation finale en élevant en puissance le résultat f de l'algorithme de Miller nous utilisons la même exponentiation utilisée dans ce travail [35]. La puissance est $e = (p^{12} - 1)/r$. Nous avons calculé cette exponentiation de la manière suivante :

$$e = \frac{p^{12} - 1}{r} = (p^6 - 1) \cdot (p^2 + 1) \cdot \frac{p^4 - p^2 + 1}{r}$$

Ce calcul est décrit par Scott et al dans [47]. L'élévation de f en puissance p^6 est équivalente à un simple conjugué, c'est à dire $f^6 = \bar{f}$ c'est le conjugué de f.

$f^{(p^6-1)} = \bar{f} * f^{-1}$ coûte une inversion et une multiplication sur $\mathbb{F}_{q^{12}}$ qui est un élément du sous groupe cyclotomique $\mathbb{G}_6(\mathbb{F}_{q^2})$ ce qui implique que toute inversion d'un élément dans ce groupe est équivalente à un simple conjugué.

5 Choix de la Courbe Elliptique

Les paramètres les plus importants pour la cryptographie à base du couplage sont les corps, l'ordre de la courbe, le degré de plongement, et l'ordre des groupes G_1, G_2 et G_3. Ces paramètres doivent être choisis de telle sorte que les meilleurs algorithmes de temps exponentiel sont aptes à résoudre le DLP dans G_1 et G_2 mais ils prennent plus de temps qu'un niveau de sécurité choisi. Les algorithmes de temps sous-exponentiel sont aptes à résoudre le DLP dans G_3 mais aussi ils prennent plus de temps qu'un niveau de sécurité choisi. Pour nos implémentations nous avons choisi la sécurité de clé symétrique de 128 bits. Pour ce dernier niveau de sécurité l'Institut national des normes et de la technologie (NIST) recommande un groupe d'ordre premier de 256 bits pour $E(\mathbb{F}_p)$ et de 3072 bits pour le corps fini $\mathbb{F}_{p^k}$, pour plus de détails nous vous proposons la référence suivante [16]. Nous avons choisi le paramètre $x = 2^{62} - 2^{54} + 2^{44}$. Sachant qu'il est très important de bien choisir le paramètre x en raison de son importance pour la sécurité des cryptosystèmes. Les courbes de Barreto-Naehrig, introduites dans [21].

5.1 Courbes adaptées aux couplages (pairing friendly)

Nous appelons une courbe elliptique dans $\mathbb{F}_p$ comme courbe adaptée aux couplages s'il y a un sous-groupe d'ordre r et dont le degré de plongement n'est pas trop grand [18]. Avec ces courbes nous pouvons considérer que les calculs dans les extensions du corps $\mathbb{F}_p$ ($\mathbb{F}_{p^k}$) sont plus simples. Ces courbes "pairing-friendly" qui sont d'ordre premier sont parfaitement importantes pour quelques protocoles à base de couplage, comme les signatures courtes. Ces références [7, 63] présentent plus de détails sur les courbes adaptées aux couplages. Parmi eux nous citons les courbes de Barreto-Naehrig (BN). Dans la début de ma thèse, ces courbes sont considérées les plus performantes pour implémenter du couplage.

Courbes Barreto-Naehrig

Pour implémenter le couplage en matériel ou en logiciel, il est difficile de choisir des courbes elliptiques. Donc, Il vaut mieux utiliser des courbes qui utilisent un degré de plongement adapté. Quand nous avons effectué ce travail, pour un niveau de sécurité de 128 bits, les courbes Barreto-Naehrig sont les courbes les mieux adaptées. Les courbes BN sont une famille de courbes définies par l'équation suivante :

$$y^2 = x^3 + b \tag{2.38}$$

avec $b \neq 0$.
L'équation de la courbe est définie par un paramètre $x \in \mathbb{Z}$ de cette manière :
$p = 36x^4 + 36x^3 + 24x^2 + 6x + 1$.
$l = 36x^4 + 36x^3 + 18x^2 + 6x + 1$.
$t = 6x^2 + 1$.
$k = 12$.

La sélection du paramètre x a une grande importance et une grande influence sur les performances du couplage. En effet, sa taille définit le niveau de sécurité du cryptosystème. Une condition nécessaire pour que cette courbe BN existe, est que les deux entiers p et l soient premiers. Le degré de plongement idéal pour un niveau de sécurité à 128 bits est égale à 12, ce qui correspond aux recommandations du NIST depuis 3 ans.

5.2 Tour d'extension

Pour travailler avec les courbes BN, le degré de plongement adapté pour ces courbes comme nous avons dit est k=12=2*3*2. $x^k - b$ qui est irréductible sur $\mathbb{F}_p$, avec b est ni carré ni cube sur $\mathbb{F}_p$. Vous trouverez plus de détails de la construction de la tour dans [39]. La tour d'extension est créée en utilisant des polynômes irréductibles. Elle est construite comme une tour quadratique et cubique. Cette tour d'extension est adoptée, puisqu'un élément f de $\mathbb{F}_{p^{12}}$ est construit à travers d'une extension quadratique de $\mathbb{F}_{p^6}$. Un élément de $\mathbb{F}_{p^{12}}$ est présenté de cette manière :$f = f_1 + f_2 w$ avec f_1 et f_2 sont des éléments de $\mathbb{F}_{p^6}$. Cette tour d'extension aide dans le calcul difficile de l'exponentiation finale. Soit la tour d'extension suivante :

$$\mathbb{F}_p \longrightarrow^2 \mathbb{F}_{p^2} \longrightarrow^3 \mathbb{F}_{p^6} \longrightarrow^2 \mathbb{F}_{p^{12}}$$

.

$$\mathbb{F}_{p^2} = \mathbb{F}_p[u]/(u^2 - b)$$

tel que $u^2 - b$ est polynôme irréductible sur $\mathbb{F}_p$ avec $b = -5$. $b \in \mathbb{F}_p$ est ni carré ni cube dans $\mathbb{F}_p$. L'extension du corps $\mathbb{F}_{p^2}$ est construit à partir de $\mathbb{F}_p$.

$$\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[v]/(v^3 - u),$$

tel que $v^3 - u$ est polynôme irréductible sur $\mathbb{F}_{p^2}$

$$\mathbb{F}_{p^{12}} = \mathbb{F}_{p^6}[w]/(w^2 - v).$$

tel que $w^2 - v$ est polynôme irréductible sur $\mathbb{F}_{p^6}$

Donc nous avons décidé de présenter l'extension $\mathbb{F}_{p^{12}}$ en utilisant la tour d'extension précédente. C'est la même tour utilisée par [11]. Nous avons construit tout d'abord l'extension quadratique, suivi d'une extension cubique et à la fin une extension quadratique, en utilisant des polynômes irréductibles dans chaque étape.

6 Travaux antérieurs

Plusieurs résultats dans la littérature pour les implémentations des couplages sur la courbe BN sont présentés dans [2, 10, 46, 61]. Parmi les implémentations les plus efficaces nous citons l'implémentation logicielle de [46]. Cette dernière présente un record de vitesse pour le couplage de Ate-Optimale avec les courbes BN. Sa vitesse est calculée par 4470408 cycles sur un processeur Intel Core 2 Quad. Une autre implémentation performante est l'application specific instruction-set processor (ASIP). Elle a été proposée dans [13]. Cette application est conçue en étendant un noyau RISC avec des unités fonctionnelles supplémentaires évolutives. Elle nécessite un environnement spécial de programmation pour exécuter des couplages. Les chercheurs ont développé un compilateur C spécial pour cette application. Un autre spécial cryptosystème à base du couplage qui utilise les courbes BN a été proposé dans [34].

Chapitre 3

Implémentations FPGA

1 Introduction

Une introduction aux Field Programmable Gate Array (FPGA), et une introduction sur les implémentations matérielles des cryptosystèmes sont présentées dans cette partie. Pour nos implémentations matérielles, nous avons utilisé les FPGAs de Xilinx .

2 Les circuits FPGA

Depuis l'année 1985, les FPGA ont été présentés au marché par la compagnie Xilinx. Différents FPGA à partir de cette date, ont été développés par d'autres compagnies comme Atmel, Actel, Altera,etc. Pour la logique programmable le besoin initial a été l'intégration d'un nombre de parties SSI (Small Scale Integration) ou MSI (Medium Scale Integration) sur un seul "chip" avec un temps de conception court et un coût faible. Par rapport aux formes antérieures de la logique programmable, la différence majeure entre des FPGAs est la dimension du circuit qui peut être implantée dans le composant.

Les FPGAs permettent l'implémentation de toute équation booléenne séquentielle ou combinatoire. Ceci se fait grâce à la structure des éléments logiques utilisés que nous appelons blocs logiques (en anglais : logic blocks (LB)).

Les FPGAs sont définis comme des dispositifs logiques d'usage général pouvant être programmés afin d'exécuter des différents traitements. Ils comportent des registres, des portes logiques et généralement des cases mémoire. En comparant avec l'architecture des différents circuits programmables, les FPGAs sont composés des blocs logiques configurables plus simples. Ils sont constitués d'un réseau interne de

blocs logiques entourés par une rangé de blocs d'entrée/sortie, ils sont connectés entre eux à travers des ressources d'interconnexions aussi programmables. Toute cellule élémentaire du réseau programmable est elle-même aussi programmable. Les FPGAs se distinguent les unes des autres par différents critères, qui sont principalement liés à la structure des architectures internes et aux caractéristiques. Nous citons parmi ces critères de distinction : la structure des interconnexions, la taille, la technologie de programmation, la structure des blocs logiques, etc.

3 Structure de base des circuits FPGA

Un FPGA consiste en une matrice de Blocs Logiques reliés par un réseau d'interconnexions. Trois composants fondamentaux composent un FPGA : les Blocs Logiques, les Blocs d'Entrée/Sortie et les interconnexions programmables comme montre la figure 3.1.

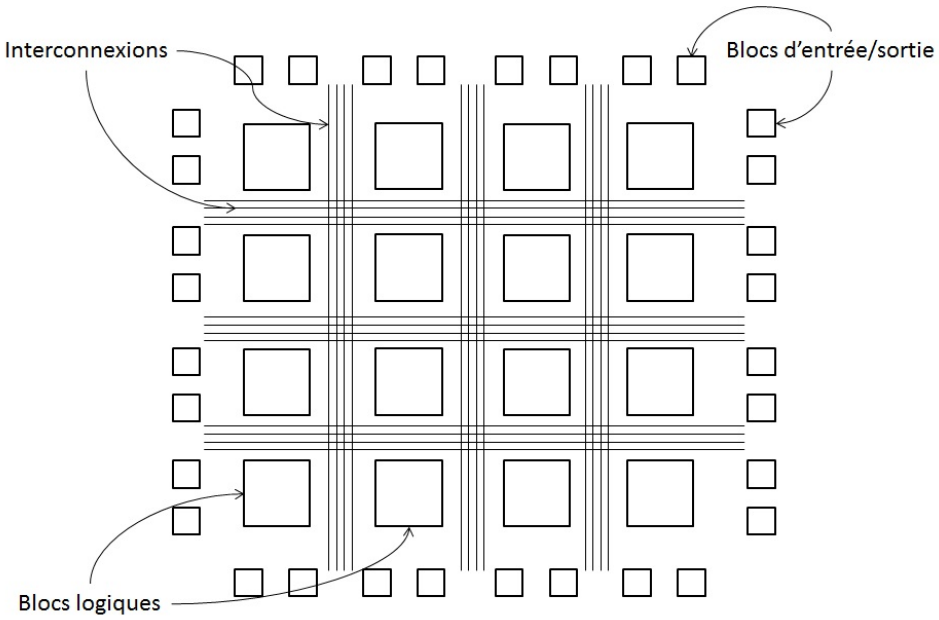

FIGURE 3.1 – Structure interne d'un circuit FPGA

Les FPGAs récents peuvent encore contenir d'autres éléments tels que : les mémoires intégrées et des PLL (Phase-Locked Loop : Boucle à verrouillage de phase) etc. Des organes de communication programmable assurent les liaisons des blocs logiques et des blocs d'entrée/sortie au réseau d'interconnexions.

3.1 Blocs Logiques

Nous pouvons appeler les blocs logiques configurables, cellules logiques (CLB). Ce sont les éléments de base dans un FPGA, qui déterminent leurs performances.

3.2 Interconnexions programmables

Le réseau d'interconnexion est considéré comme une matrice qui est composée de plusieurs segments métalliques courant verticalement et horizontalement entre les blocs logiques et les blocs d'entrée/sortie. Alors, ce réseau permet de connecter, en programmation, soit 2 blocs logiques programmables, soit 2 segments du réseau d'interconnexion, soit un bloc logique programmable et un bloc d'entrée/sortie.

3.3 Blocs d'entrée/sortie

Une liaison entre la logique interne du circuit FPGA et l'extérieur est garantie par ces Blocs. Nous trouvons les Blocs d'entée/sortie présents dans l'ensemble de la périphérie du FPGA. Ils sont placés sur le périmètre du circuit. Chaque bloc d'entrée/sortie contrôle une broche du composant, un bloc peut être configuré soit en sortie, soit en entrée, soit en entrée/sortie, ou bien peut être non utilisé. Un autre rôle des blocs entrée/sortie est de permettre la synchronisation des signaux avec l'horloge du circuit. Cette synchronisation est assurée grâce à des bascules (flipflop).

4 Architecture Systolique

Les architectures systoliques ont été proposées en 1978 par Kung et Leiseron dans [43]. Ce modèle a été proposé pour répondre à la demande croissante de puissance de calcul. Il s'est révélé être un moyen très efficace pour la conception des architectures matérielles.

En un mot, une architecture systolique est un réseau composé d'un grand nombre de cellules élémentaires identiques et localement inter-connectées.

Chaque cellule reçoit les données en provenance des cellules voisines, effectue un calcul arithmétique simple, puis envoie les résultats aux cellules voisines. Toutes ces étapes se faites dans un seul cycle d'horloge. Les cellules dans une architecture évoluent en parallèle, sous le contrôle d'une horloge globale.

La dénomination du mot systolique provient de l'analogie entre la circulation des flots de données dans un réseau et celle du sang dans le corps humain. C'est l'horloge qui assure la synchronisation du système constituant le cœur.

Nous avons besoin de cette architecture systolique dans notre travail pour effectuer une implémentation matérielle de la multiplication modulaire de Montgomery.

Une architecture systolique [44,80] est un outil d'implémentation de la multiplication de Montgomery en matérielle. Les auteurs dans [28, 60, 62, 76] ont présenté leurs implémentations matérielles de la multiplication de Montgomery en utilisant une architecture systolique. Ces architectures manipulent des cellules élémentaires, et chaque cellule effectue un simple calcul arithmétique, une addition et une multiplication. Conformément aux nombres des mots utilisés, l'architecture peut manipuler un nombre variant des cellules. L'architecture systolique utilise des cellules élémentaires très simples. Par conséquent, une architecture systolique diminue les besoins en ressources pour les implémentations matérielles.

5 Étude de l'existant

Les premiers chercheurs, d'après nos connaissances, qui ont proposé une architecture systolique pour implémenter une multiplication modulaire sont Iwamura, Matsumoto et Imai [72, 73]. Ils ont présenté une architecture qui permet d'exécuter une exponentiation modulaire en utilisant la multiplication modulaire de Montgomery. Dans [76] Tenca et Koç ont introduit une multiplication modulaire de Montgomery pipelinée. Par la suite Harris et al. dans [29] ont amélioré les résultats de [76] en utilisant l'architecture systolique pour Montgomery. Siddika Berna Örs, Lejla Batina, Bart Preneel et Joos Vandewalle ont présenté dans [60] une exponentiation modulaire basée sur la multiplication de Montgomery. Dans [62] Guilherme Perin, Daniel Gomes Mesquita et Joao Baptista Martins ont proposé une comparaison entre deux architectures de multiplication modulaire : une architecture systolique et une architecture multiplexée. Dans leurs ils approches utilisent un radix-16 et un radix-32 de décomposition. Les deux architectures sont implémentées sur deux versions de FPGA de Xilinx, un Virtex-4 et un Virtex-5. Les implémentations de leurs travaux qui utilisent une architecture systolique sont les plus récentes et les plus efficaces. L'architecture systolique proposée dans le travail [62] est composée en s cellules élémentaires distribuées dans un tableau unidimensionnel, avec s est le nombre des mots manipulés. A chaque itération de l'algorithme de Montgomery les mots sont lus d'une mémoire externe (BRAM) et envoyés aux composants nécessaires de l'architecture. Pour évaluer le nombre de cycles d'horloge de la multiplication de Montgomery, ils ont considéré les s premiers cycles pour effectuer la lecture des données à partir de la mémoire, de plus la première itération de l'algorithme

nécessite s cycles. Finalement les itérations restantes de l'algorithme nécessitent $4s$ cycles d'horloge. Par conséquent leur architecture nécessite $6s$(=s+s+$4s$) cycles d'horloge. Concernant l'architecture multiplexée, la première étape est identique à celle de l'architecture systolique $2s$, et le nombre de cycles nécessaire pour effectuer le reste de l'algorithme est $6s$. Donc afin d'effectuer une multiplication modulaire avec l'architecture multiplexée, il ont utilisé $8s$(=$2s$ + $6s$) cycles d'horloge.

Chapitre 4

Architecture systolique pour la multiplication de Montgomery en grande caractéristique

1 Introduction

La recherche de l'architecture la plus optimisée pour l'arithmétique a toujours fasciné le monde des systèmes embarqués. Ces dernières années, cela a été particulièrement le cas dans des domaines des corps finis pour la cryptographie, grâce à l'invention des systèmes de cryptage asymétriques basés sur des opérations arithmétiques modulaires. Tout au long de l'histoire de la cryptographie pour les systèmes embarqués, il y a un grand besoin pour des architectures efficaces pour ces opérations. Les implémentations doivent être performantes, en termes de ressources ainsi que de latence. L'arithmétique des corps finis est la primitive la plus importante de ECC, couplage et RSA. Depuis 1976, de nombreux Cryptosystèmes à clé publique (PKC) ont été proposés et tous ces cryptosystèmes basent leur sécurité sur la difficulté d'un problème mathématique. La dureté de ce problème mathématique sous-jacent est essentielle pour la sécurité. Les Cryptosystèmes basés sur les courbes elliptiques sont proposés par Koblitz [40] et Miller [81], RSA [69] et la cryptographie basée sur le couplage [38] est un exemple de PKCs. Tous ces systèmes nécessitent une multiplication modulaire efficace. En conséquence, le développement d'une architecture efficace pour la multiplication modulaire a été un sujet de recherche très populaire. En 1985, Montgomery a présenté une nouvelle méthode pour la multiplication modulaire [57]. C'est l'un des algorithmes les plus appropriés pour effectuer des multiplications modulaires dans les implémentations matérielles et logicielles.

L'implémentation efficace de la multiplication modulaire Montgomery (MMM) en matériel a été traitée par de nombreux auteurs [9,28,60,62,73,76]. Il existe une variété de façons d'effectuer le MMM, en considérant si la multiplication et la réduction sont séparées ou intégrées. L'architecture systolique [44, 80] est une solution pour l'implémentation de l'algorithme de Mongomery en matérielle. Avec un design à la fois en parallèle et en pipeline. [28, 60, 62, 68, 76]. Un travail similaire présenté dans [68] a été fait sur des corps binaires (la caractéristique du corps est une puissance de 2). Ces architectures utilisent des cellules élémentaires de traitement (Processing Elements :PE) où chaque élément de traitement effectue des additions et des multiplications arithmétiques. En fonction du nombre de mots utilisés, l'architecture peut employer un nombre variable de PE. L'architecture systolique utilise des éléments de traitement très simples (comme dans une architecture pipeline). En conséquence, l'architecture systolique diminue les besoins en ressources dans les implémentations matérielles. Notre contribution dans ce chapitre est de combiner une architecture systolique, qui est supposée être le meilleur choix pour les implémentations matérielles, avec la méthode CIOS de multiplication modulaire de Montgomery. Nous optimisons le nombre de cycles d'horloge requis pour calculer un MMM à n-bit et nous réduisons l'utilisation des ressources FPGA. Nous avons implémenté la multiplication modulaire dans un nombre fixe de cycles d'horloge. Selon nos connaissances, c'est la première fois qu'une implémentation en matériel ou en logiciel de multiplication modulaire de Montgomery, adapté à divers niveaux de sécurité, est réalisée en seulement 33 cycles d'horloge. De plus, à notre connaissance, notre travail est le premier qui traite l'architecture systolique et la méthode CIOS sur les corps finis pour les grandes caractéristiques. En utilisant notre implémentation matérielle MMM efficace, nous proposons une conception efficace pour les opérations ECC : l'ajout de points et le doublement. Dans ce chapitre, nous démontrons l'intérêt de l'implémentation de la multiplication de Montgomery présentée dans l'algorithme 5, plus précisément la méthode Coarsely Integrated Operand Scanning(CIOS), avec une architecture systolique. Le grand intérêt de cette architecture est son arithmétique qui permet de travailler parallèlement sur des grands nombres. Nous allons présenter dans ce chapitre nos différentes architectures. Nous avons optimisé le nombre de cycles d'horloge nécessaire pour calculer une multiplication modulaire. Nous avons optimisé aussi les ressources sur la carte FPGA. Notre idée est d'utiliser le maximum des Digital signal processing (DSP). Nous avons implémenté toutes nos architectures avec un nombre de cycles d'horloges fixe. Ce design permet d'optimiser en ressources dans les implémentations matérielles. Nos architectures sont présentées sous forme des cellules élémentaires simples. Chaque cellule effectue un calcul simple, une mul-

tiplication et/ou une addition. Le nombre des cellules pour chaque design dépend de la taille des mots. Notre contribution est de combiner une architecture systolique avec Montgomery. Cette idée est supposée le meilleur choix pour les implémentations sur FPGA. Un autre avantage de cette architecture qui est adaptable à tous niveaux de sécurité. Nous pouvons effectuer une multiplication modulaire dans 33 cycles d'horloge. Nous commençons ce chapitre avec une définition sur les FPGS et les DSP. Après nous démontrons notre idée de combiner l'architecture systolique avec la méthode CIOS, Nous présenterons ensuite nos architectures proposées et nous terminerons par une comparaison des résultats obtenus avec l'existant.

Algorithme 5 : Multiplication Modulaire de Montgomery

Input : p entier premier et impair, $n = \lceil log_2(p) \rceil$, $R = 2^n$, $p' = -p^{-1} mod$ R, $M(a)$, $M(b) \in \mathbb{F}_p$

Output : $M(ab) \mod p$

1 $\gamma \leftarrow M(a) \times M(b)$
2 $\delta \leftarrow \gamma \times p' \mod R$
3 $T \leftarrow \frac{\gamma + \delta \times p}{R}$
4 **If** $T \geq p$ **then** $T \leftarrow T - p$
5 **return** T

2 Introduction aux problématiques de l'implémentation FPGA

Nous souhaitons donner dans cette partie une introduction aux problématiques de l'implémentation matérielle et plus spécifiquement pour la plateforme FPGA.

Blocs DSPs des FPGAs d'Xilinx

Les FPGA modernes, comme le Virtex-4, Virtex-5, Artix 7 et d'autre d'Xilinx, ainsi que les FPGAs d'Altera comme Startix, sont équipés par des extensions matérielles pour le calcul arithmétique. Afin d'accélérer les applications de traitement numérique du signal (en anglais :Digital signal processing (DSP)). Ces blocs peuvent être utilisés pour construire des implémentations plus efficaces en terme de performance et permettre en même temps de réduire la surface utilisée sur FPGAs. Les blocs DSP sont programmés pour effectuer les calculs arithmétiques de base comme la multiplication, l'addition et la soustraction des entiers non-signés. La figure 4.1 illustre la structure générique des DSPs dans les FPGAs modernes. Ces

blocs peuvent fonctionner via des entrées externes A, B et C ainsi qu'une valeur de retour P.

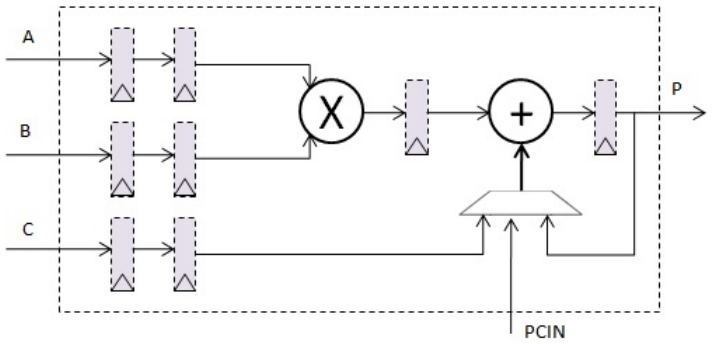

FIGURE 4.1 – Structure des DSPs dans les FPGAs modernes.

3 Implémentation Matérielle

3.1 Architectures proposées

L'idée pour notre architecture est de combiner la méthode CIOS de la multiplication modulaire de Montgomery, présentée dans l'algorithme 6 et proposée par [41], avec une architecture systolique à deux dimensions introduite dans [31,80]. Comme nous avons vu dans la partie 2.2 du chapitre 2 , la méthode CIOS présente une alternance entre les itérations de la multiplication et de la réduction durant toute l'exécution de l'algorithme de Montgomery. Le concept d'une architecture systolique à deux dimensions, présenté dans le chapitre 2, combine des cellules élémentaires identiques qui sont connectées localement. Chaque cellule reçoit des données en provenance des cellules voisines, effectue un calcul simple, puis transmet les résultats, toujours aux cellules voisines.

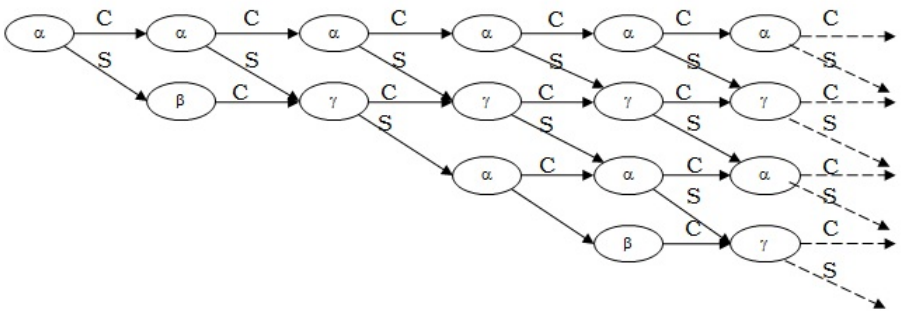

FIGURE 4.2 – Flot de données dans une architecture systolique.

Algorithme 6 : Méthode CIOS [41]

Input : $p < 2^n$, $p' = -p^{-1} mod\ 2^w$, w, s , $K = s \cdot w$:bit length, $R = 2^K$, $a, b < p$
Output : $a \cdot b \cdot R^{-1}$ mod p

```
1  T ← Null ;
2  for i ← 0 to s − 1 do
3      C ← 0 ;
4      for j ← 0 to s − 1 do
5          (C, S) ← T[j] + a[i] · b[j] + C
6          T[j] ← S
7      (C, S) ← T[s] + C
8      T[s] ← S
9      T[s + 1] ← C
10     C ← 0 ;
11     m ← T[0] · p' mod 2^w
12     (C, S) ← T[0] + m · p[0]
13     for j ← 1 to s − 1 do
14         (C, S) ← T[j] + m · p[j] + C
15         T[j] ← S
16     (C, S) ← T[s] + C
17     T[s − 1] ← S
18     T[s] ← T[s + 1] + C
19 return T ;
```

Nos architectures sont basées directement sur les opérations arithmétiques de l'algorithme de Montgomery (CIOS). Les différentes opérations sont effectuées dans un radix-w de base (2^w). C'est à dire que toutes nos entrées sont divisées sur des mots de w bits. En se basant sur cette idée nous avons proposé plusieurs versions pour cette architecture. Nous détaillerons dans ce chapitre les architectures pour $s = 8$, $s = 16$, $s = 32$ et $s = 64$ qui sont notées respectivement NW-8 (Number of Words), NW-16, NW-32 et NW-64. Sachant que $s = K/w$. Avant d'entrer dans les détails de chacune de nos architectures, nous présentons la description générique de notre design systolique. Nous décrivons ainsi le comportement des différents cellules élémentaires. Afin d'avoir une architecture optimisée, nous avons divisé l'algorithme 6 de la multiplication de Montgomery en cinq types de cellules :

- cellule alpha notée α ;
- cellule beta notée β ;
- cellule gamma notée γ ;
- cellule alpha final notée α_f ;
- cellule gamma final notée γ_f.

Nous présentons dans l'algorithme 7, le découpage de cette méthode en cellules élémentaires.

Dans la figure 4.2 nous présentons la dépendance entre les différentes cellules de l'architecture. dans la suite nous décrivons par détails chaque cellule. MSW est le mots de poids fort (en anglais : Most Significant Word), et LSW est le mot de poids faible (en anglais : Least Significant Word). Dans ce chapitre nous avons utilisé comme notation C pour MSW et S pour LSW.

3.2 Cellule α

Cette cellule est présentée par les lignes 4 et 5 dans l'algorithme 6 et détaillée dans l'algorithme 8. Les cellules élémentaires alpha sont évolutives selon le paramètre NW dans notre design. Nous utilisons cette cellule pour effectuer les étapes de la multiplication dans l'algorithme. Les entrées de cette cellule alpha sont :

1. S_In fourni par l'étape précédente,
2. C_In fourni par l'étape précédente,
3. $a[i]$: les mots de l'opérande a,
4. $b[j]$: les mots de l'opérande b.

Les sorties de la cellule alpha sont

1. S Fourni pour l'étape suivante.

Algorithme 7 : Algorithme de la méthode CIOS

Input : $p < 2^n$, $p' = -p^{-1} mod\ 2^w$, w, s , $K = s \cdot w$:bit length, $R = 2^K$, $a, b < p$

Output : $a \cdot b \cdot R^{-1}$ mod p

```
1  T ← 0 ;
2  for i ← 0 to s − 1 do
3      C ← 0 ;
4      for j ← 0 to s − 1 do                          } α cell
5          (C, S) ← T[j] + a[i] · b[j] + C
6          T[j] ← S
7      (C, S) ← T[s] + C                              } α_f cell
8      T[s] ← S
9      T[s + 1] ← C
10     m ← T[0] · p' mod 2^w                          } β cell
11     (C, S) ← T[0] + m · p[0]
12     for j ← 1 to s − 1 do                          } γ cell
13         (C, S) ← T[j] + m · p[j] + C
14         T[j − 1] ← S
15     (C, S) ← T[s] + C                              } γ_f cell
16     T[s − 1] ← S
17     T[s] ← T[s + 1] + C
18 return T ;
```

Algorithme 8 : Cellule alpha

Input : $a[i]$, $b[j]$, C_In, S_In
Output : C, S
1 $tmp1 \leftarrow S\_In + C\_In$
2 $tmp2 \leftarrow a[i] \cdot b[j]$
3 $tmp2 \leftarrow tmp2 + tmp1$
4 $C \leftarrow$MSW($tmp2$)
5 $S \leftarrow$LSW($tmp2$)
6 **return** C, S;

2. C Fourni pour l'étape suivante.

Description de la cellule α

Comme montre la figure 4.3, la multiplication des mots $a[i]$ et $b[j]$ fournit un résultat sur $2w$ bits. Par la suite ce résultat est ajouté à S_α_In. Ce dernier est le mot de poids faible du résultat fourni par la cellule gamma, cette entrée est choisie par un multiplexeur. Le dernier résultat est aussi ajouté à C_α_In. Sachant que C_α_In est le mot de poids fort du résultat fourni par la cellule alpha précédente, qui est fourni aussi à travers un deuxième multiplexeur. Les différentes entrées/sorties de alpha sont détaillées dans la figure 4.10. Les mots de poids fort du résultat des cellules alpha sont envoyés à un autre multiplexeur pour initialiser une autre cellule alpha.Tandis que les mots de poids faible sont fournis à un multiplexeur pour une cellule gamma de l'étape suivante. Après chaque calcul d'une cellule alpha un décalage de l'opérande B se déclenche.

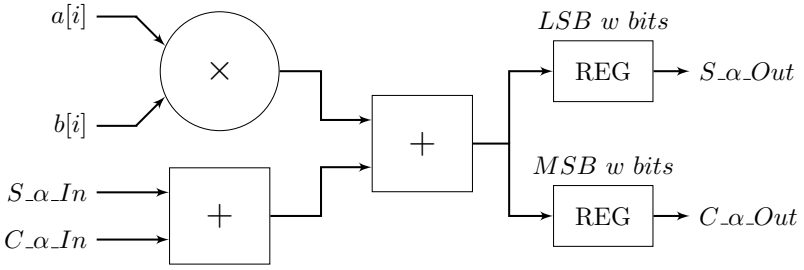

FIGURE 4.3 – Cellule élémentaire Alpha-architecture interne.

3.3 Cellule β

La cellule beta est présentée par les lignes 9, 10 et 11 dans l'algorithme 6 et détaillée dans l'algorithme 9

Les entrées de cette cellule bêta sont :

Algorithme 9 : Cellule beta

Input : S_in, $p[0]$, $p' = -p^{-1} mod\ 2^w$
Output : C, m
1 $tmp1 \leftarrow S\_in \cdot p'$
2 $m \leftarrow LSW(tmp1)$
3 $tmp1 \leftarrow p[0] \cdot m$
4 $tmp1 \leftarrow S\_in + tmp1$
5 $C \leftarrow$MSW$(tmp1)$
6 **return** C, m;

1. S_In fourni par l'étape précédente,
2. $p[0]$ le premier mot du modulo p,
3. p' : entrée pré-calculée.

Les sorties de la cellule alpha sont

1. m Fourni pour l'étape suivante.
2. C Fourni pour l'étape suivante.

Description de la cellule β

Selon notre algorithme 9 et comme illustré dans la figure 4.4, le mot d'index zéro du modulo p ($p[0]$) ainsi que p' sont les entrées de la cellule bêta. L'entier p' correspond à l'inverse du p modulo 2^w. Le résultat de la multiplication entre p' et S_β_In est sur $2w$ bits. Le mot de poids faible de ce résultat est multiplié par le premier mot du modulo p, et on obtient un résultat sur $2w$ bits. Pour terminer le calcul du bêta, on ajoute à la fin l'entrée S_β_In. Seulement le mot de poids fort de ce dernier résultat est utilisé pour les étapes suivantes. Les différentes entrées/sorties du bêta sont présentées dans la figure 4.10.

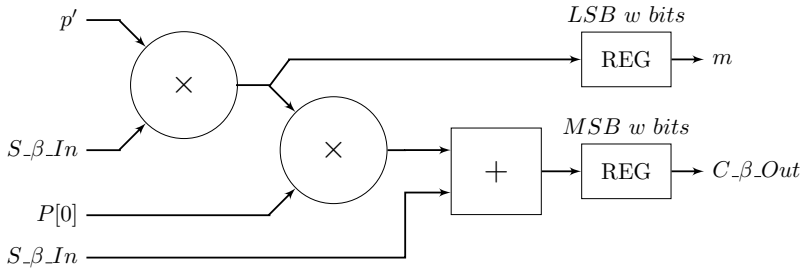

FIGURE 4.4 – Cellule élémentaire Bêta-architecture interne.

3.4 Cellule γ

La cellule gamma fonctionne de la même manière que la cellule alpha, mais avec des différentes entrées et sorties. Gamma est présentée par les lignes 13 et 14 toujours dans le même algorithme 6, est détaillée dans l'algorithme 10. Les PEs gamma sont scalables selon le paramètre NW dans notre design. Nous utilisons cette cellule pour effectuer les étapes de la réduction dans l'algorithme. Les entrées

Algorithme 10 : Cellule gamma

Input : $p[i]$, m, C_in, S_in
Output : C, S
1 $tmp1 \leftarrow S\_in + C\_in$
2 $tmp2 \leftarrow p[i] \cdot m$
3 $tmp2 \leftarrow tmp2 + tmp1$
4 $C \leftarrow$MSW($tmp2$)
5 $S \leftarrow$LSW($tmp2$)
6 **return** C, S;

de cette cellule gamma sont :

1. S_In fourni par l'étape précédente,
2. C_In fourni par l'étape précédente,
3. $p[j]$: les mots du modulo p,
4. m : entrée fourni par la cellule bêta.

Les sorties de la cellule gamma sont :

1. S Fourni pour l'étape suivante.
2. C Fourni pour l'étape suivante.

Description de la cellule γ

Comme illustré dans la figure 4.5, le résultat de la multiplication entre les mots m et $p[j]$ est présenté sur $2w$ bits. Par la suite ce résultat est ajouté à S_γ_In. Ce dernier est le mot de poids faible du résultat fourni par la cellule alpha, cette entrée est choisie par un multiplexeur. Le dernier résultat est aussi ajouté à C_γ_In. Sachant que C_γ_In est le mot de poids fort du résultat fourni par la cellule gamma précédente, qui est choisie aussi à travers un deuxième multiplexeur. Les différentes entrées/sorties de gamma sont détaillées dans la figure 4.10. Le mot de poids fort du résultat de la cellule gamma est envoyé à un autre multiplexeur pour fixer la cellule gamma suivante.Tandis que le mot de poids faible est fourni à un multiplexeur pour la cellule alpha de l'étape suivante.

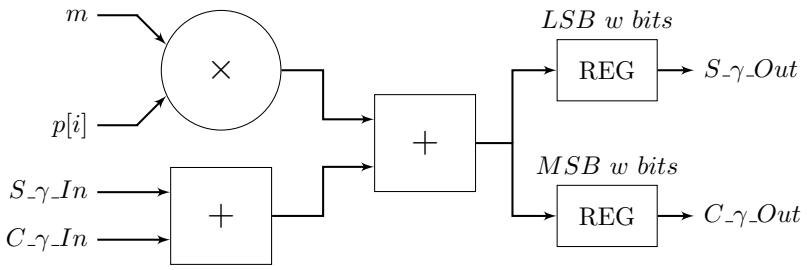

FIGURE 4.5 – Cellule élémentaire Gamma-architecture interne.

3.5 Cellule α_final

Cellule α_final est présentée dans les lignes 6,7 et 8 dans l'algorithme 6 et elle est détaillée dans l'algorithme 11. Les entrées de cette cellule alpha_final sont :

Algorithme 11 : Cellule alpha_final

Input : C_in, S_in
Output : $S1$, $S2$
1 $tmp1 \leftarrow S\_in + C\_in$
2 $S1 \leftarrow$LSW($tmp1$)
3 $S2 \leftarrow$MSW($tmp1$)
4 **return** C, S ;

1. S_In fourni par l'étape précédente,
2. C_In fourni par l'étape précédente,

Les sorties de la cellule alpha_final sont :

1. $S1$ Fourni pour l'étape suivante.
2. $S2$ Fourni pour l'étape suivante.

Description de la cellule α_final

La cellule α_f présente la dernière cellule des lignes des α. Ces lignes correspondent au calcul de l'étape de la multiplication. Dans α_f on a une addition entre $S\_\alpha\_f\_In$ et C_α_f qui retourne un résultat sur $2w$ bits comme présenté dans la figure 4.6.

3.6 Cellule γ_final

: Cellule γ_final est présentée dans les lignes 15,16 et 17 dans l'algorithme 6, et elle est détaillée dans l'algorithme 12. Les entrées de cette cellule gamma_final sont :

1. $S1\_In$ fourni par l'étape précédente,

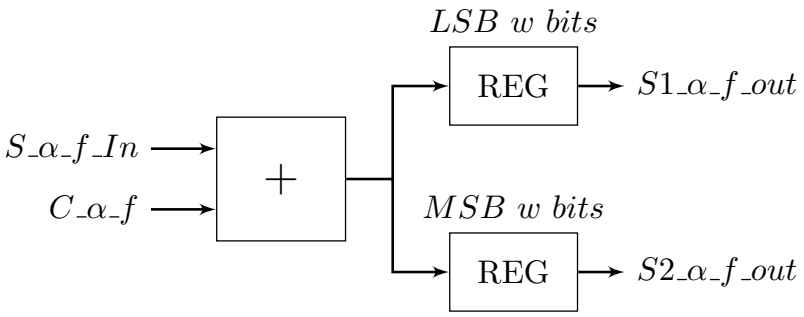

FIGURE 4.6 – Cellule élémentaire Alpha_f-architecture interne.

Algorithme 12 : Cellule gamma_final

Input : C_in, $S1\_in$, $S2\_in$
Output : $S1$, $S2$
1 $tmp1 \leftarrow S1\_in + C\_in$
2 $S1 \leftarrow$LSW($tmp1$)
3 $S2 \leftarrow$MSW($tmp1$)
4 $S2 \leftarrow S2\_in + S2$
5 **return** $S1$, $S2$;

2. $S2\_In$ fourni par l'étape précédente,
3. C_In fourni par l'étape précédente,

Les sorties de la cellule gamma_final sont :

1. $S1$ Fourni pour l'étape suivante.
2. $S2$ Fourni pour l'étape suivante.

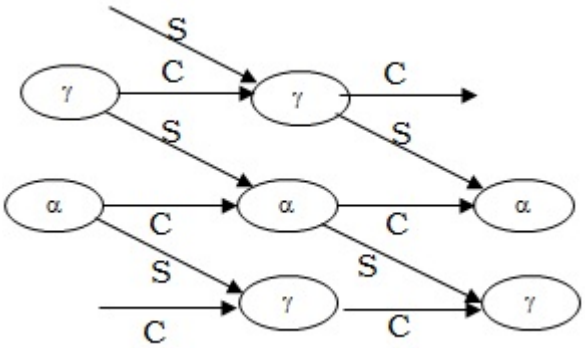

FIGURE 4.7 – PEs de l'Architecture systolique dans two-dimensional array.

Description de la cellule γ_f

La cellule γ_f présente la dernière cellule des lignes des γ, ces lignes correspondent au calcul de l'étape de la réduction. Pour cette cellule $S1\_\gamma\_f\_In$ est ajouté à

C_γ_f ce qui donne un résultat sur $2w$ bits. Par la suite le mot de poids faible de ce résultat est ajouté à $S2\_\gamma\_f\_In$. L'architecture interne de gamma_final est présentée dans la figure 4.8.

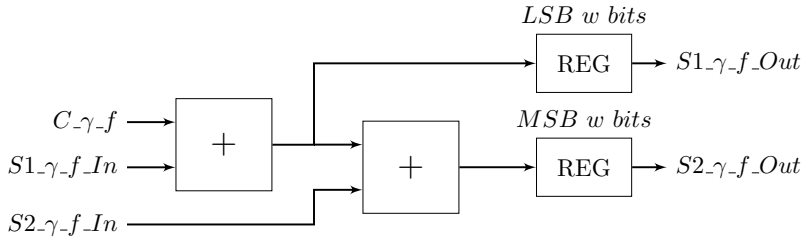

FIGURE 4.8 – Cellule élémentaire Gamma_f-architecture interne.

Cette organisation nous permet d'optimiser le nombre de cycles d'horloge. Chaque cellule dans la figure 4.11 effectue un simple calcul (des opérations arithmétiques). Comme illustré dans la figure 4.10 toutes nos cellules sont gérées par un bloc de contrôle (la machine à états finis ou FSM en anglais), en recevant des signaux pour commencer le calcul à chaque état dans la multiplication de Montgomery. Après la fin du calcul chaque cellule envoie, toujours à la FSM, un signal pour déclarer que ce calcul est terminé. Le résultat final d'une multiplication de Montgomery est la concaténation des dernières sorties des cellules gamma et gamma_final.

4 Nos architectures

Nous allons détailler dans cette partie nos différentes architectures proposées pour implémenter la multiplication modulaire de Montgomery. Nous commencerons par l'architecture NW-8, cette version contient 3 cellules de type α et 3 de type γ. Avec ce design nous pouvons implémenter toute la multiplication de Montgomery en juste 33 cycles d'horloge. Nous passerons par la suite à la deuxième version : l'architecture NW-16. Cette version est composée de 6 cellules alpha et 6 cellules gamma. Nous pouvons effectuer la multiplication de Montgomery avec cette architecture en 66 cycles d'horloge. De la même façon, pour implémenter les architectures NW-32 et NW-64, nous doublons à chaque fois le nombre des cellules alpha et gamma. Grâce à l'évolutivité de notre design, on peut passer d'une version à une autre, en utilisant la même logique de modélisation. A la fin de cette section nous proposons une comparaison entre nos différentes architectures.

4.1 Architecture NW-8

Dans cette architecture, les opérandes ainsi que le modulo sont divisés sur 8 mots comme illustré dans la figure 4.11. L'architecture NW-8 est composée de 9 processeurs élémentaires (ou cellules) distribués dans une matrice bi-dimensionnelle. Chaque cellule est responsable d'un calcul simple prenant en entrées des mots de w bits, les différentes cellules sont présentées dans la figure 4.9. Par exemple, pour le niveau de sécurité 128 bits pour ECC ou le couplage, nous avons besoin d'effectuer des opérations de multiplication à 256 bits, en utilisant notre architecture de NW-8 les opérandes sont divisés à 8 mots de 32 bits, par contre si on travaille dans un niveau de sécurité 256 bits la taille des mots passe à 64 bits. Cette méthode nous permet de modéliser notre architecture avec 9 cellules élémentaires pour un design systolique à deux dimensions. Les 9 cellules sont les suivantes :
3 cellules alpha,
1 cellule alpha_final,
1 cellule bêta,
3 cellules gamma,
1 cellule gamma_final.

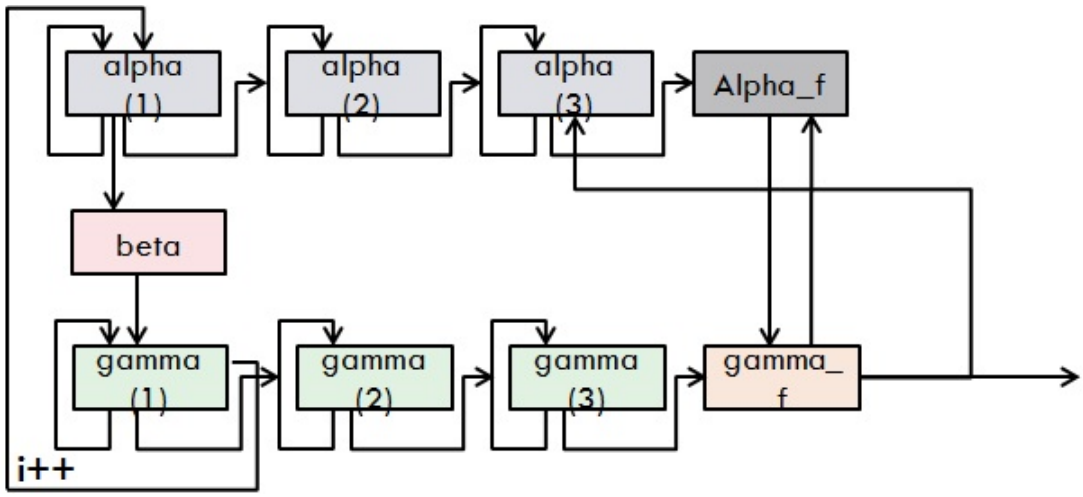

FIGURE 4.9 – CIOS NW-8 Architecture.

Notre choix de modélisation est fait afin de diminuer les étapes dans la machine à état fini(finite state machines (FSM)). Comme détaillé avant dans ce manuscrit chaque cellule élémentaire dans N-dimension est connectée à 2N données d'entrées/sorties, pour communiquer avec 2N cellules élémentaires. Dans notre cas nous travaillons avec une architecture à deux-dimensions. Chaque cellule dans notre design est connectée par 4 chemins de données : 2 entrées et 2 sorties, comme présenté dans la figure 4.7

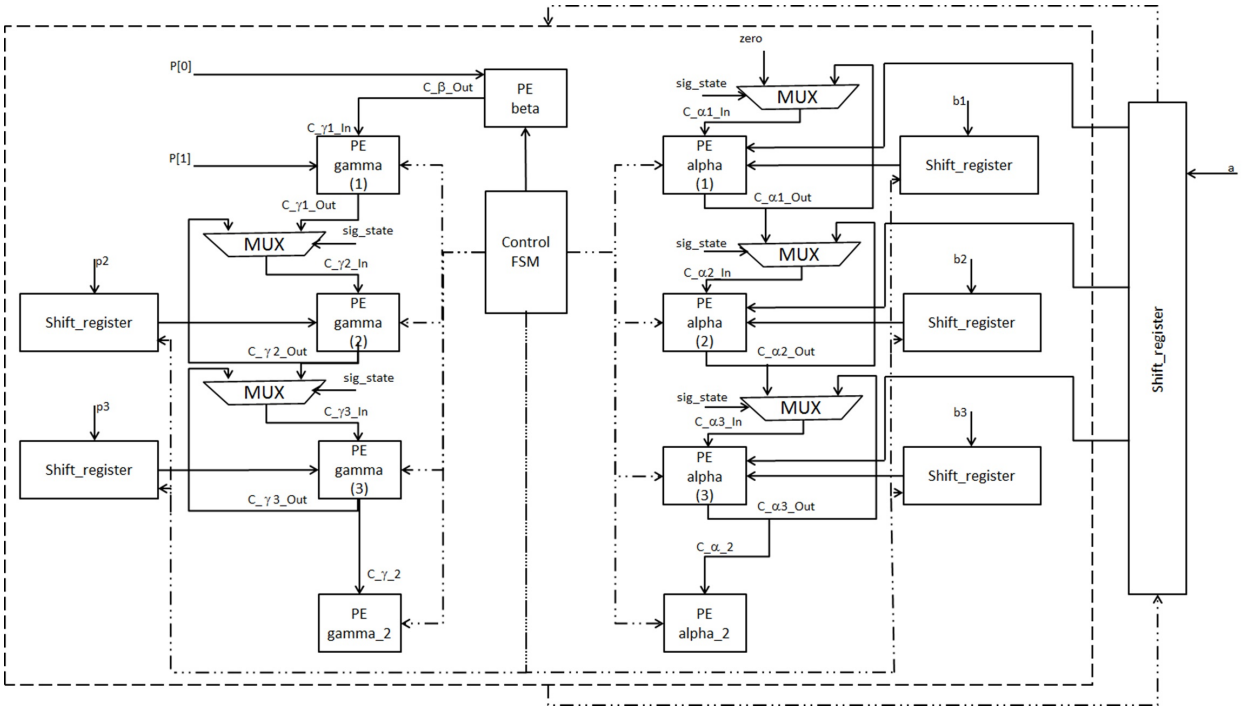

FIGURE 4.10 – Architecture proposée de la multiplication modulaire de Montgomery.

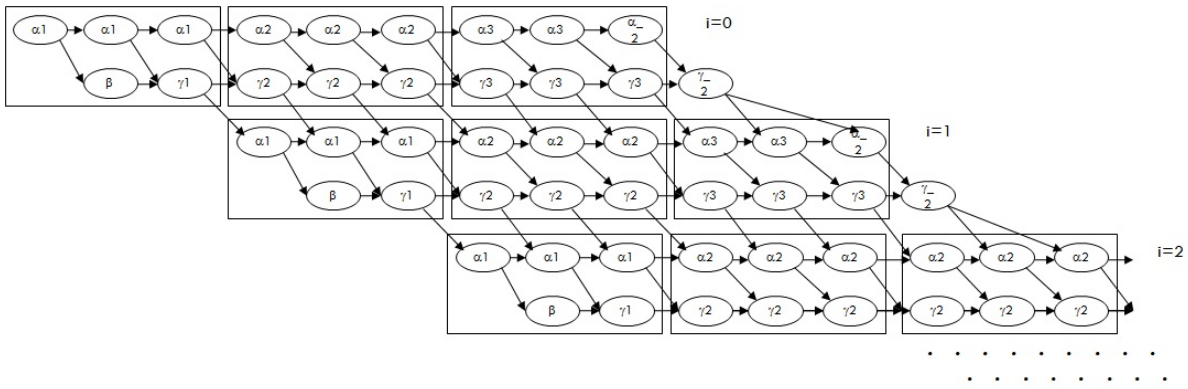

FIGURE 4.11 – Le graphique de dépendance de données de la nouvelle architecture systolique bidimensionnel proposée (NW-8).

Dans cette architecture, les cellules élémentaires sont commandées par une FSM. Ce bloc de contrôle communique avec les différentes cellules ainsi les registres à décalage avec des signaux. Nous avons utilisé des signaux pour déclencher les débuts d'exécution de chaque cellule et à chaque étape. La figure 4.10 présente un aperçu sur toute l'architecture de la multiplication de Montgomery avec NW-8. Pour plus des détails techniques la figure 4.13 présente les différentes cellules avec tous les entrées/sorties. Les registres à décalage sont conçus pour fournir les mots requis pour chaque cellule à chaque étape d'exécution. La cellule élémentaire alpha nécessite les mots $a[i]$ de l'opérande a et $b[j]$ de l'opérande b. Par contre la cellule gamma nécessite les mots $p[j]$ du modulo p avec $1 \leq j \leq s$. La figure 4.12 présente les rotations qu'on a utilisé pour cette architecture. Pour effectuer une multiplication

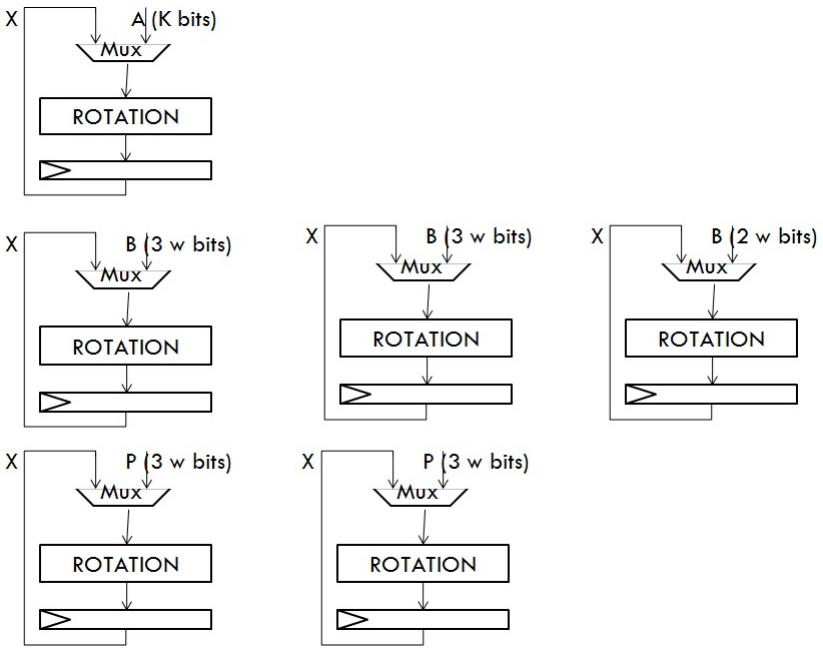

FIGURE 4.12 – Architectures internes - Rotation.

modulaire dans le domaine de Montgomery, avec cette première architecture, on fait $s + 3$ itérations. Et le résultat final de la multiplication est présenté par une concaténation des sorties des cellules gamma de la dernière itération. Le résultat final fourni par cette multiplication est $a \cdot b \cdot R^{-1} \mod p$. Pour évaluer le nombre des cycles d'horloge pour ce design de la multiplication modulaire dans NW-8, le premier paramètre est max{nombre des cellules alpha, nombre des cellules gamma}=3. C'est à dire notre architecture peut manipuler trois itérations de l'algorithme en parallèle comme illustré dans la figure 4.11. Ce qui implique que notre architecture doit boucler $s + 3$ fois. Et par conséquence on peut effectuer toute la multiplication de Montgomery dans 33 cycles d'horloge, puisque notre design est modélisé avec trois

états ($33 = 3 \times (s + 3)$). Quelques résultats pour cette architecture et avec des données de taille 256 bits sont présentés dans la Table 4.1.

Artix-7	DSP	Frequency (MHz)	Clock cycle
MMM (s=8/K=256)	31	105.275	33
Alpha	4	291.023	1
Gamma	4	291.023	1
Beta	4	388.350	1
Alpha_final	1	459.918	1
Gamma_final	2	442.811	1

TABLE 4.1 – Implémentations des cellules et MMM (NW-8).

4.2 Architecture NW-16

L'architecture NW-16 est conçue de la même manière que l'architecture NW-8. Dans cette version les opérandes et le modulo sont divisés sur 16 mots. Ce design est composé de 15 cellules élémentaires distribuées toujours dans une matrice à deux dimensions. Sachant que les cellules utilisées sont les mêmes dans nos différents designs, chaque élément du design est responsable d'un calcul simple prenant des entrées à w bits, les différentes cellules sont présentées dans la figure 4.15. Ces 15 cellules utilisées sont les suivantes :
6 cellules α,
1 cellule α_final,
1 cellule β,
6 cellules γ
1 cellule γ_final.

Passant de NW-8 à NW-16 on peut remarquer que le nombre des cellules α et γ a doublé, avec toujours une β, une α_final et une γ_final. Comme nous avons dit précédemment, le nombre des (alpha_final, beta, gamma_final) reste inchangé peu importe le nombre des mots manipulés dans le design. Pour évaluer le nombre de cycles d'horloge pour ce design de la multiplication modulaire dans NW-16, le premier paramètre est max{nombre des cellules α, nombre des cellules γ}=6. C'est à dire notre architecture peut manipuler six itérations de l'algorithme en parallèle

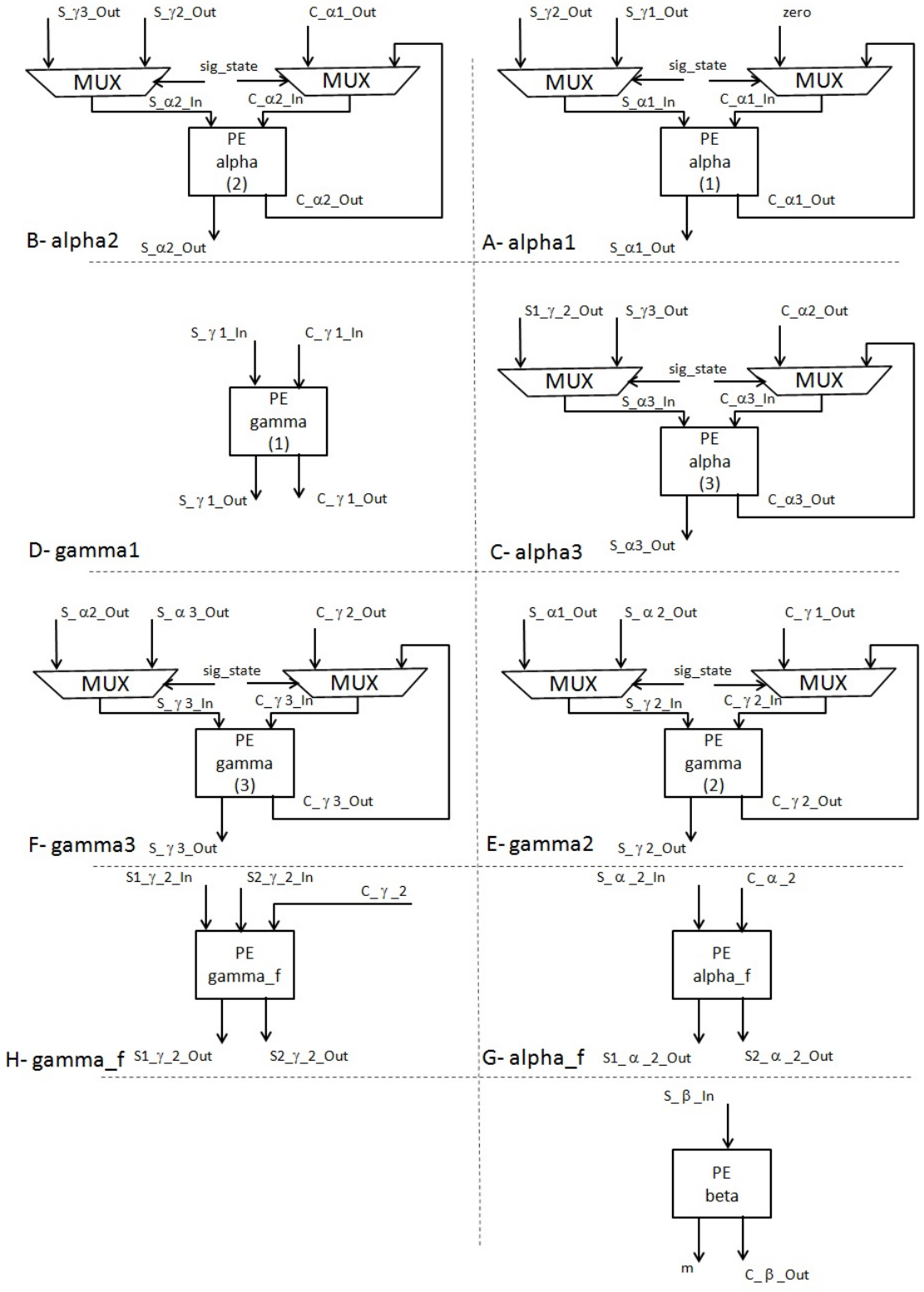

FIGURE 4.13 – Toutes nos cellules élémentaires.

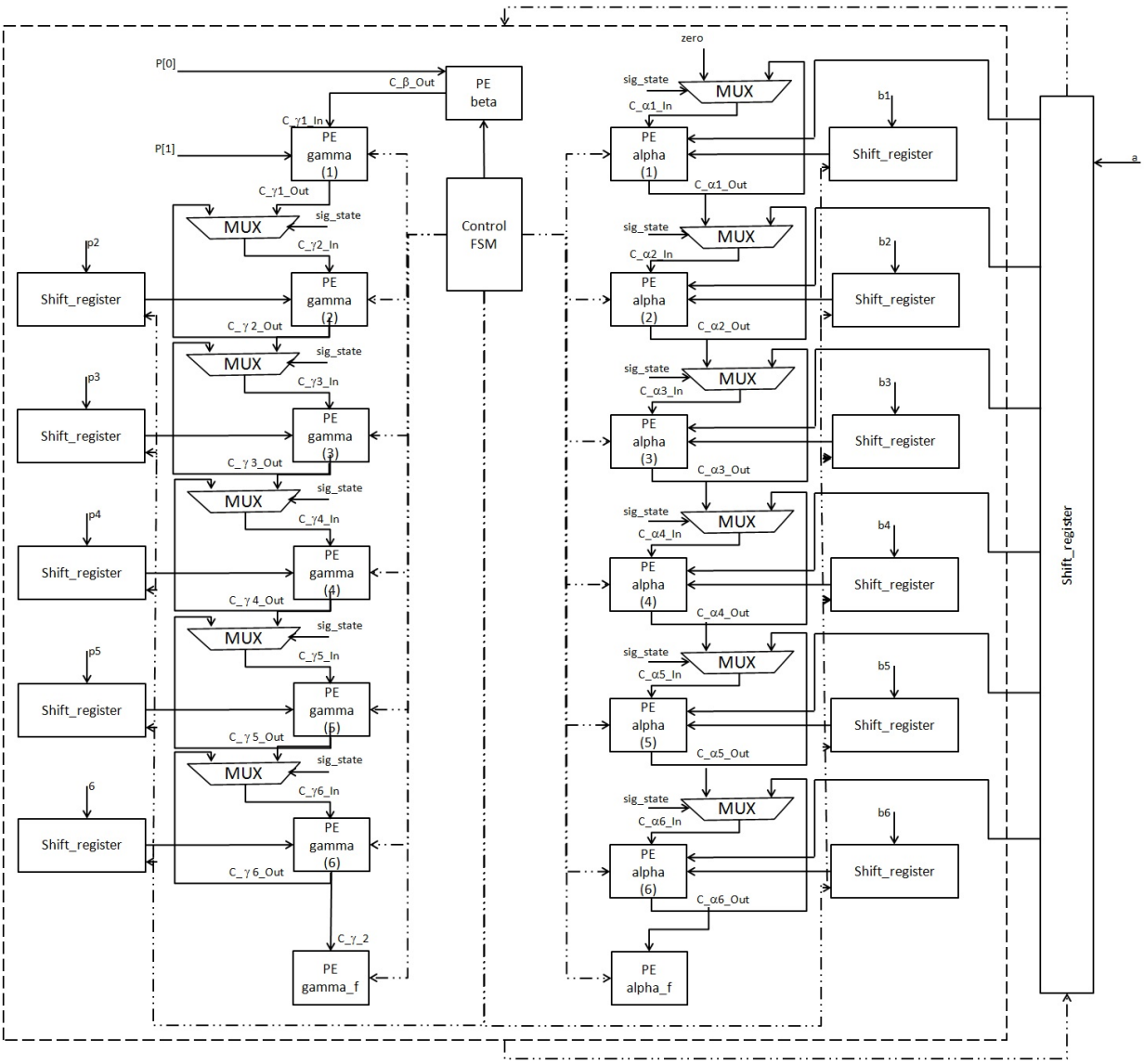

FIGURE 4.14 – Architecture proposée de multiplication modulaire Montgomery NW-16.

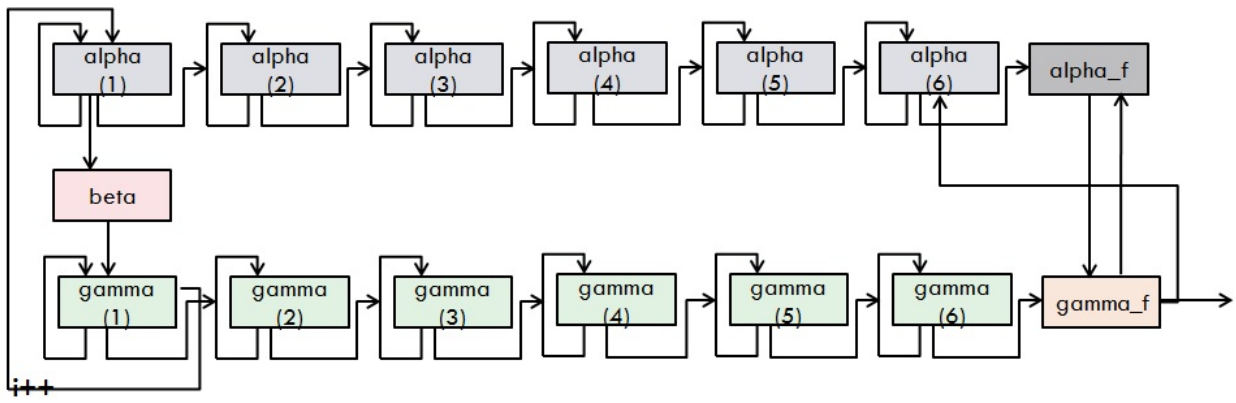

FIGURE 4.15 – CIOS NW-16 Architecture.

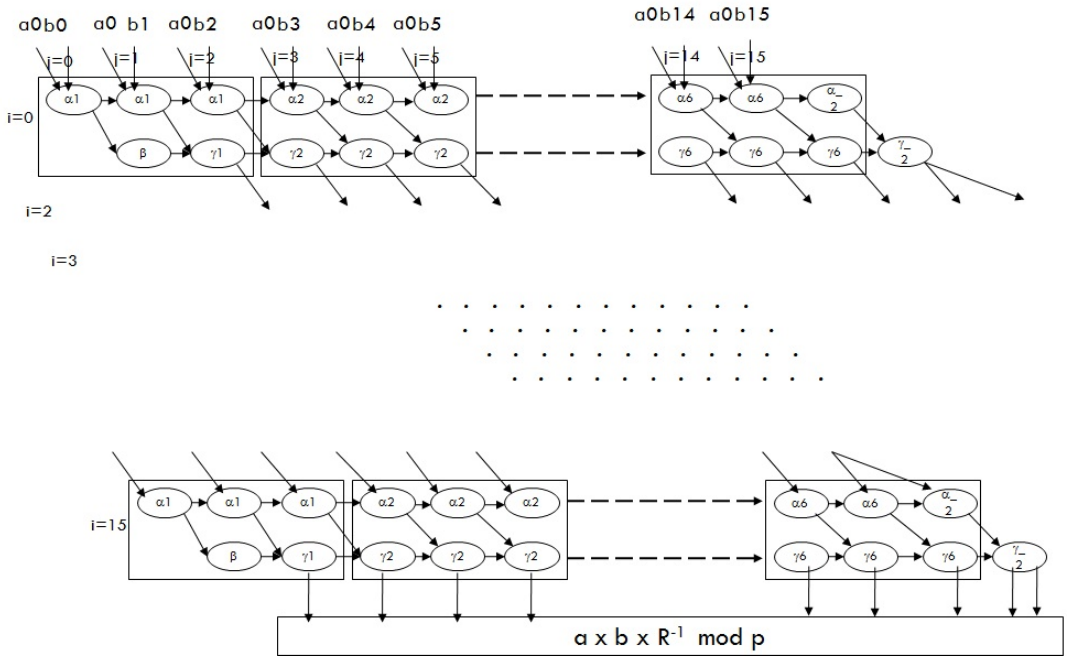

FIGURE 4.16 – Le graphique de dépendance de données de la nouvelle architecture systolique bidimensionnelle proposée (NW-16).

comme illustré dans la figure 4.16. Ce qui implique que notre architecture doit boucler $s + 6$ fois. Par conséquence on peut effectuer toute la multiplication de Montgomery en 66 cycles d'horloge, puisque notre design est modélisé avec trois états ($66 = 3 \times (s+6)$). Quelques résultats pour cette architecture avec des données de taille 256 bits sont présentés dans la table 4.2.

Artix-7	DSP	Fréquence (MHz)	Cycles d'horloge
MMM (s=16/K=256)	29	145.892	66
Alpha	2	379.341	1
Gamma	2	379.341	1
Beta	2	453.104	1
Alpha_final	1	459.918	1
Gamma_final	2	442.811	1

TABLE 4.2 – Implémentations des cellules et MMM (NW-16).

4.3 Architecture NW-32

Dans cette architecture, les opérandes et le modulo sont divisés en 32 mots. Et avec la même logique de conception le NW-32 nécessite 27 cellules élémentaires. Les cellules sont les suivantes :
12 cellules alpha,
1 cellule alpha_final,
1 cellule bêta,
12 cellules gamma
1 cellule gamma_final.
Pour évaluer le nombre de cycles d'horloge pour ce design de la multiplication modulaire dans NW-32, le premier paramètre est max{nombre des cellules α, nombre des cellules γ}=12. C'est à dire notre architecture peut manipuler douze itérations de l'algorithme en parallèle. Ce qui implique que notre architecture doit boucler $s + 12$ fois. Par conséquence on peut effectuer toute la multiplication de Montgomery en 132 cycles d'horloge, puisque notre design est modélisé avec trois états ($132 = 3 \times (s + 12)$).

4.4 Architecture NW-64

Dans cette architecture, les opérandes et le modulo sont divisés en 64 mots. Et avec la même logique de conception le NW-64 nécessite 51 cellules élémentaires. Les cellules sont les suivantes :
24 cellules alpha, 1 cellule α_final,
1 cellule β,
24 cellules γ
1cellule γ_final.
Pour évaluer le nombre de cycles d'horloge pour ce design de la multiplication modulaire dans NW-64, le premier paramètre est max{nombre des cellules α, nombre des cellules γ}=24. C'est à dire notre architecture peut manipuler vingt quatre itérations de l'algorithme en parallèle. Ce qui implique que notre architecture doit boucler s+24 fois. Donc on peut effectuer toute la multiplication de Montgomery en 264 cycles d'horloge, puisque notre design est modélisé avec trois états ($264 = 3 \times (s + 24)$).

Architecture NW-s

Dans cette partie nous généralisons nos différentes architectures, de la même manière que les autres architectures, nous pouvons proposer une version générique.

Les opérandes ainsi que le modulo dans cette version sont divisés sur s-mots. Cette architecture est composée de nbc cellules élémentaires avec nbc=$(s - s/4) + 3$. Les cellules sont distribuées dans une architecture systolique à deux dimensions. Chaque cellule responsable d'un calcul sur des mots de tailles w. Le nombre des cycles d'horloge est $3 \times (s + nb)$ avec nb=max{nombre des cellules alpha , nombre des cellules gamma}=$\frac{(s - s/4)}{2}$. Notre algorithme nécessite $s + nb$ itérations. Avec cette architecture nous pouvons effectuer la multiplication dans $3\times(s+nb)$ cycles d'horloge puisque notre FSM est de trois états.

4.5 Comparaison entre les Architectures

Le tableau 4.3 détaille une comparaison entre nos différentes architectures. Le nombre des cycles d'horloge pour chaque architecture égale à 3 × (s+nb). Avec nb=max{number of cells alpha, number of cells gamma}. Ce qui implique toute architecture nécessite $s + nb$ boucles pour effectuer une multiplication modulaire complète. Il est important de noter que toutes nos architectures sont scalables pour tous les niveaux de sécurité qui sont utiles dans la cryptographie.

CIOS	s=8	s=16	s=32	s=64
K=256	32	16	8	4
K=512	64	32	16	8
K=1024	128	64	32	16
K=2048	256	128	64	32
Clock cycles= 3 × (s+nb)	33	66	132	264
Number of cells	6 +3	12 +3	24 +3	48 +3

TABLE 4.3 – comparaison des architectures

5 Résultats

La Table 4.4 résume les résultats d'implémentations sur FPGA des architectures proposées dans ce chapitre. Nous présentons les résultats pour nos deux architectures NW-8 et NW-16. Tous nos designs sont décrits en langages matérielles (VHDL) et

synthétisés avec les FPGAs d'Xilinx (Artix7 et Virtex5). Afin de vérifier l'exactitude des résultats, nous les avons comparés avec des codes en SageMath. Nous présentons dans cette partie des résultats après implémentation, avec des différentes tailles. Le grand avantage de ces architectures est la pertinence pour plusieurs applications avec différents niveaux de sécurité, comme le couplage, ECC et RSA. Comme présenté dans la Table 4.4 nous avons une propriété intéressante pour ces designs : le nombre de cycles d'horloge ne dépend pas de la taille des données. Les résultats proposés dans ce chapitre sont comparés avec les meilleurs résultats de la littérature [29, 30, 60, 62]. Cette comparaison est détaillée dans la Table 4.5 et dans Table 4.6 . Afin d'implémenter la multiplication modulaire de Montgomery pour un niveau de sécurité, nous devons choisir l'architecture la plus adéquate.

Nous avons remarqué que nos résultats sont meilleurs par rapport à [62] considérant plusieurs points de comparaison comme le nombre de slices et le nombre de cycles d'horloge. Toutefois en comparant l'architecture Systolique par rapport à l'architecture multiplexée, la première est plus rapide pour le décryptage RSA mais avec une fréquence de +40 Mhz et une surface +1637 slices. Notre VHDL étant transposable sur ASIC, ces résultats ont une incidence directe sur la puissance statique de +33% et dynamique de +45% à laquelle il faudrait rajouter la puissance due à la capacité de charge des interconnexions proportionnelle à la longueur de celles-ci. Dans ces conditions, l'architecture multiplexée serait plus intéressante que l'architecture Systolique.

Concernant le nombre de slices, nous rappelons que [62] ont utilisé une mémoire externe afin d'optimiser le nombre de slices utilisées par leurs algorithmes. En comparant avec [60], notre design nécessite moins de nombre de slices, avec une meilleure fréquence et nous améliorons vraiment le nombre de cycles d'horloge.

Notre design a effectué la multiplication de Montgomery en 33 et 66 cycles d'horloge pour la longueur de 512 et 1024 bits pour les niveaux de sécurité AES-256 et AES-512, tandis que [60] ont effectué la multiplication en 1540 cycles d'horloge pour le niveau de sécurité AES-256 et 3076 pour le niveau de sécurité AES-512 car il a été tenu compte dans leurs mesures expérimentales des temps d'accès à la BRAM externe. Nous n'avons pas tenu compte de ces temps, d'après Koç et al, le total des temps de lecture pour la CIOS est $6s^2 + 7s + 2$ et pour l'écriture il est $2s^2 + 5s + 1$. En plus, leurs expérimentations ont été menées sur VIRTEX E (Techno 128nm, obsolète) et les nôtres sur Artix 7 (Technologie 28nm) beaucoup plus rapide

Artix 7- Nexys 4

	NW-8			NW-16		
	128	256	512	256	512	1024
Freq MHz	198	106	65	146	106	65
cycles	33	33	33	66	66	66
Slice Registers	487	870	1614	1123	2164	4208
Slice LUTs	355	809	2650	846	1789	5242
Slices	206	352	878	402	798	2072
DSP	19	31	87	29	57	161

TABLE 4.4 – illustration de la scalabilité de nos architectures.

Xilinx FPGAs

	A7 our	**V5 our**	[62] V5	[60] VE	[42] V	[4] K7 and V5	
	512	**512**	512	512	512	512 K7	512 V5
Freq MHz	**106**	**97**	95	95	72	176	123
Cycles	**66**	**66**	96	1540	–	66	66
vitesse μs	**0.6**	**0.6**	1.010	16	–	0.3	0.5
Slice Reg	**2164**	**3046**	3876	–	–	5076	4960
Slice LUTs	**1789**	**1781**	–	2972	3125	8757	10877
BRAM	0	0	128	–	–	0	0

TABLE 4.5 – Comparaison avec la littérature pour K=512 bits.

Xilinx FPGAs

	A7 our	**V5 our**	[62] V5	[60] VE	[30] VII	[29] VII	[42] V
	1024	**1024**	1024	1024	1024	1024	1024
Freq MHz	**65**	**65**	130	95	116	119	79
Cycles	**66**	**66**	384	3076	1088	1167	–
vitesse μs	**1.01**	**1.01**	2.9	32	9	9	–
Slice Reg	**4208**	**6072**	6642	–	–	–	–
Slice LUTs	**5242**	**5824**	–	5706	9319	9271	6243
BRAM	0	0	256	–	–	–	–

TABLE 4.6 – Comparaison avec la littérature pour K= 1024 bits.

Chapitre 5

Implémentation de l'arithmétique pour la construction des cryptosystèmes

1 Introduction

Nous proposons dans ce chapitre, des méthodes efficaces et des formules explicites optimisées qui accélèrent considérablement le calcul des couplages à base des courbes elliptiques sur les corps finis. Nos contributions peuvent être résumées comme suit : En utilisant notre implémentation matérielle MMM efficace, nous proposons une conception efficace pour les opérations ECC : l'addition de deux points et le doublement d'un point. Nous avons utilisé nos architectures sur les corps finis pour optimiser l'implémentation du couplage Ate-Optimal. Nos architectures proposées pour la multiplication modulaire, permet d'avoir une implémentation efficace de l'arithmétique des extensions des corps finis. Dans ce chapitre nous proposons aussi une implémentation optimisée pour l'inversion et/ou la division modulaire. Cette architecture matérielle de l'inversion permet de gagner en terme de ressources par rapport à des architectures proposées dans la littérature. Nous avons proposé dans ce chapitre une comparaison d'efficacité entre notre inversion/division et une multiplication modulaire publiée récemment dans la littérature. Nous proposons une comparaison de la performance (ressources×latences). Cette comparaison peut aider à faire le choix entre l'implémentation d'un crypto-système avec des coordonnées affines ou bien avec des coordonnées Jacobiennes. Une inversion efficace peut nous aider à éviter les changements du système des coordonnées. Par exemple dans le cas normal pour passer du système de coordonnées affines à un système de coor-

données projectives, une inversion est remplacée par 10 multiplications. Dans une implémentation matérielle, plus nous avons des opérations plus nous utilisons des multiplexeurs. Sachant que les multiplexeurs sont chers de point de vue ressources matérielles dans une carte FPGA.

2 Cryptographie basée sur les Courbes Elliptiques : ECC

Nous rappelons dans cette section l'arithmétique de ECC que nous avons utilisé pour nos implémentations. Plus de détails sur cette partie présenté dans le chapitre 2. L'utilisation des courbes elliptiques en cryptographie,pendant les années 80, a été proposée dans les travaux des chercheurs Victor S. Miller [54] et N. Koblitz [40]. Comme nous avons annoncé dans le chapitre 2, l'avantage de la cryptographie basée sur les courbes elliptiques est l'utilisation des clés de taille réduite en comparant avec le cryptosystème RSA [71] pour le même niveau de sécurité. Par exemple, la clé pour ECC est de 256 bits pour le niveau de sécurité AES 128 bits, tandis que la clé pour RSA est de 3072 bits.

Nous nous sommes concentré sur l'opération principale de ECC. C'est l'opération de la multiplication scalaire sur la courbe elliptique. Cette multiplication scalaire consiste à calculer $\alpha \times P$, pour α un entier et P un point d'une courbe elliptique. L'implémentation d'une telle opération sur des plate-formes matérielles tels que les FPGAs est soumise à des différentes contraintes. Les principales contraintes sont les ressources, le nombre des cycles d'horloge et la fréquence. Donc une arithmétique efficace de la multiplication scalaire est d'importance majeure.

2.1 preliminaries ECC

Nous présentons dans cette section l'équation de la courbe elliptique utilisée pour nos implémentations matérielles. Soit la courbe elliptique E définie sur $\mathbb{F}_p$ avec $p > 3$ selon l'équation de Weierstrass suivante :

$$E : y^2 = x^3 + ax + b \tag{5.1}$$

Où $a, b \in \mathbb{F}_p$ tel que $4a^3 + 27b^2 \neq 0$. La courbe elliptique $E(\mathbb{F}_p)$ est l'ensemble des points $(x, y) \in \mathbb{F}_p^2$. Les coordonnées de E satisfont l'équation (5.1). Les points rationnels de E avec un élément neutre O appelé point à l'infini, forment une structure de groupe abélien. Le calcul de la somme de deux points en coordonnées affines

$P = (x_1, y_1)$ et $Q = (x_2, y_2)$ est $P + Q = (x_3, y_3)$ où :
$x_3 = \lambda^2 - x_1 - x_2$
et
$y_3 = \lambda(x_1 - x_3) - y_1$
avec

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} \text{ if } P \neq Q, \\ \dfrac{3x_1^2 + a}{2y_1} \text{ if } P = Q. \end{cases} \tag{5.2}$$

La multiplication scalaire d'un point P par un entier naturel α est notée αP. Le problème du logarithme discret est de trouver la valeur de α, avec P et αP sont connus. La sécurité de l'ECC est basée sur la dureté du problème logarithmique discret. Les formules d'addition de deux points comme dans l'équation (5.2) sont basées sur plusieurs opérations sur $\mathbb{F}_p$ (par exemple la multiplication, l'inversion, l'addition et la soustraction) qui ont des coûts de calcul différents.

2.2 ECC en coordonnées Jacobiennes

Dans les coordonnées jacobiennes, nous utilisons $(x : y : z)$ pour représenter le point en coordonnées affines $(x/z^2; y/z^3)$. L'équation de la courbe elliptique devient :

$$Y^2 = X^3 + aXZ^4 + bZ^6.$$

Étape de doublement : nous représentons le point $Q \in E(\mathbb{F}_p)$ en coordonnées jacobiennes comme $Q = (X_Q, Y_Q, Z_Q)$. La formule pour le doublement $T = 2Q = (X_T, Y_T, Z_T)$ peut être calculée comme suit :

$$X_T = 9X_Q^4 - 8X_QY_Q^2. \quad Y_T = 3X_Q^2(4X_QY_Q - X_T) - 8Y_Q^4. \quad Z_T = 2Y_QZ_Q.$$

Étape addition : soit $Q = (X_Q, Y_Q, Z_Q)$ et $T = (X_T, Y_T, Z_T) \in E(\mathbb{F}_p)$. Alors, le point $R = T + Q = (X_R, Y_R, Z_R)$, peut être calculé comme suit :

$$X_R = (2Y_QZ_T^3 - 2Y_T)^2 - 4(X_QZ_T^2 - X_T)^3 - 8(X_QZ_T^2 - X_T)^2X_T.$$

$$Y_R = (2Y_QZ_T^3 - 2Y_T)(4(X_QZ_T^2 - X_T)^2X_T - X_R) - 8Y_T(X_QZ_T^2 - XT)^3.$$

$$Z_R = 2Z_T(X_QZ_T^2 - X_T).$$

Algorithme 13 : Multiplication scalaire

Input : $\alpha = (\alpha_n \alpha_{n-1} \dots \alpha_1 \alpha_0)_2$ radix 2 decomposition $\in \mathbb{F}_p$, $P \in E(\mathbb{F}_p)$
Output : αP

```
T ← P
for i = n − 1 to 0 do
    T ← 2T
    if α_i = 1 then
        T ← T + P
return T
```

2.3 Affine VS projective et Jacobienne

Le systèmes des coordonnées affines est le système le plus simple pour effectuer le calcul de ECC ou le calcul du couplage. En outre, Les coordonnées affines exigent une division dans chaque addition et dans chaque doublement, mais exige moins de multiplication que les coordonnées projectives [15]. La Table 5.1 explique une comparaison entre les différents systèmes de coordonnées.

Dans la Table 5.2, nous détaillons les performances ($latence \times surface$) d'une multiplication modulaire publiée dans [4] et de notre inversion modulaire. Pour comparer, notre inversion modulaire (I) égale à 4.5 multiplication modulaire (M) (I=4,5M). Avec ces résultats, nous pouvons dire que pour effectuer la multiplication scalaire, nous pouvons utiliser le système de coordonnées affines pour obtenir une implémentation plus efficace que la multiplication scalaire avec d'autre systèmes de coordonnées en utilisant la multiplication de [4]. Aussi ce résultat nous permet de réduire le nombre d'opérations sur l'algorithme de Miller lors de l'exécution des additions et des doublements. Notre nouvelle architecture matérielle peut également améliorer l'implémentation de l'exponentiation finale et par conséquent nous pouvons améliorer l'implémentation de couplage.

Coordonnée	Addition	Doublement
Affine	I+2M+S	I+2M+2S
Projective	12M+2S	7M+5S
Jacobienne	12M+4S	4M+6S

TABLE 5.1 – Opérations arithmétiques sur les corps finis par Type des Coordonnées.

	Multiplication Modulaire [4]	Inversion Modulaire (notre)	
$latency \times area$	2.293 10^6	10.394 10^6	I$\simeq$ 4.5 M

TABLE 5.2 – comparaison multiplication et inversion sur $latency \times area$ pour une efficacité de 384 bits

2.4 Implémentation de ECC

Lorsque les algorithmes ECC sont implémentés d'une manière séquentielle, présentent l'avantage que le nombre de modules arithmétiques de corps fini peut être réduit au minimum. Par exemple, un seul additionneur, un multiplicateur et une unité de soustraction (peut être un additionneur) sont nécessaires pour l'addition et le doublement des points. La parallélisation entre la multiplication, l'addition et la soustraction a été obtenue.

Nous avons proposé dans la figure 5.1 et la figure 5.2 les graphiques de flux de données pour l'addition de deux points et le doublement d'un point. Dans cette conception, nous utilisons notre architecture systolique efficace de MMM pour effectuer un carré ou une multiplication. La Table 5.3 résume les résultats d'implémentation de la multiplication scalaire. Nous présentons des résultats avec les architectures NW-8 et NW-16. Pour vérifier l'exactitude des résultats matériels, nous comparons les résultats donnés par le FPGA avec l'implémentation du logiciel sage.

	Slice	DSPs	BRAM	Freq	Slice FF	Slice LUT
NW-8 (256)	3745	33	12	98	8281	9722
NW-16 (256)	3770	34	12	130	8313	9255
NW-8 (512)	7066	92	23	59	16500	20394
NW-16 (512)	7116	60	23	74	16501	19199

TABLE 5.3 – Implémentations de ECC (Jacobienne) sur FPGA de Xilinx.

3 Inversion Modulaire

L'inversion modulaire a^{-1} mod p d'un entier existe si et seulement si a et p sont premiers, et par conséquent leur $pgcd(a, p) = 1$. Deux méthodes d'inversion sont

souvent utilisées : le petit théorème de Fermat et une variante de l'algorithme d'Euclide étendu (AEE). Tout d'abord nous effectuons l'inversion par exponentiation. Il existe plusieurs variantes de ces algorithmes rapportées dans la littérature, la plupart d'entre eux sont énumérées et discutées dans [12]. Mais la plus récente inversion modulaire a été présentée dans [3], c'est un algorithme d'inversion modulaire RNS basé sur l'algorithme d'Euclide étendu et le plus-minus trick.

3.1 Algorithme de l'inversion implémentée

L'une des variantes efficaces de l'AEE pour l'inversion $\mathbb{F}_p$ basée sur la méthode binaire, qui est connue comme l'algorithme d'inversion binaire illustré dans l'algorithme 3. Dans cette partie nous proposons une légère modification sur cette algorithme pour l'adapter à une implémentation matérielle optimisée. les modifications sont présentées dans l'algorithme 14

Cet algorithme fonctionne d'une manière itérative, et se poursuit vers l'objectif. À chaque itération, soit u ou v réduite d'au moins de bit, ce qui assure que le nombre total d'itérations est d'au plus 4N, où N est la longueur maximale de p et a. L'algorithme 14 d'inversion binaire peut facilement être modifié pour effectuer la division modulaire $b/a = b * a^{-1}$. Pour obtenir $b = a \bmod p$ en utilisant cet algorithme, il est nécessaire d'initialiser la variable $x1$ à l'étape 1 par b au lieu de 1. Nous suivons cet algorithme pour effectuer des opérations d'inversion et de division sur $\mathbb{F}_p$ dans la courbe et le couplage matériel elliptique.

3.2 Implémentation Matérielle

Notre conception proposée calcule une inversion sur $\mathbb{F}_p$ ainsi que la division modulaire en utilisant l'algorithme binaire qui est décrit dans l'algorithme 14. La conception dans les figures 5.3 et 5.4 effectue les mêmes opérations sur différentes entrées $u, x1$ et $v, x2$. Les figures 5.5 et 5.6 effectuent les mêmes opérations sur différentes entrées $u, x1$ et $v, x2$. Les valeurs mises à jour de $u, v, x1$ et $x2$ après chaque cycle d'horloge sortent comme suit $u, v, x1$ et $x2$. Le multiplexage des résultats intermédiaires se fait sur la base des valeurs du premier bit de u et du premier bit de v, le premier bit indiquant si les valeurs actuelles de u et v sont paires ou impaires.

Les blocs illustrés dans la figure 5.4, la figure 5.5 et la figure 5.6, sont présentés dans l'algorithme 14 de la ligne 3 à la ligne 12 et détaillés dans l'algorithme 15. Les entrées sont u et $x1$. Ce bloc contient un seul additionneur, deux décalages et trois multiplexeurs. Le bloc illustré dans la figure 5.4, la figure 5.6 et la figure 5.5

Algorithme 14 : Inversion/Division Binaire sur $\mathbb{F}_p$

Input : $a \in \mathbb{F}_p$
Output : a^{-1} mod p

```
1  u ← a, v ← p, x1 ← b, x2 ← 0.
2  repeat
3      if u is even then
4          u ← u/2.
5          if x1 is even then
6              x1 ← x1/2.
7          else
8              x1 ← (x1 + p)/2.
9      else
10         if u > v then
11             u ← u + v.
12             x1 ← x1 + x2.
13     if v is even then
14         v ← v/2.
15         if x2 is even then
16             x2 ← x2/2.
17         else
18             x2 ← (x2 + p)/2.
19     else
20         if v > u then
21             v ← u + v.
22             x2 ← x1 + x2.
23 until u = 1 or v = 1;
24 if u = 1 then
25     return x1;
26 else
27     return x2;
```

est présenté dans l'algorithme 14 de la ligne 13 à la ligne 22 et détaillé dans l'algorithme 16. Les entrées sont v et $x2$. Ce bloc contient un seul additionneur, deux décalages et trois multiplexeurs. Il est le même bloc réutilisé dans l'algorithme 15. Une itération de l'algorithme 14 est en un seul cycle d'horloge et à chaque itération, nous avons u ou v réduite d'une taille de bit. Par conséquent, le temps de latence de notre inversion est au maximum 4 N cycles d'horloge.

Algorithme 15 : Bloc 1

```
   Input : u, x1
   Output : u, x1
 1 if u is even then
 2     u ← u/2.
 3     if x1 is even then
 4         x1 ← x1/2.
 5     else
 6         x1 ← (x1 + p)/2.
 7 else
 8     if u > v then
 9         u ← u + v.
10         x1 ← x1 + x2.
11 return u, x1;
```

Algorithme 16 : Bloc 2

```
   Input : v, x2
   Output : v, x2
 1 if v is even then
 2     v ← v/2.
 3     if x2 is even then
 4         x2 ← x2/2.
 5     else
 6         x2 ← (x2 + p)/2.
 7 else
 8     if v >= u then
 9         v ← u + v.
10         x2 ← x1 + x2.
11 return v, x2;
```

	Slice FF	Slice LUT	slice	Frequency (MHz)	Clock cycles
[27] 256 bits (Spartan3)			1490	41	3N+5
Our 256 bits (Spartan3)	1061	2624	1505	51	4N
Our 256 bits (Artix7)	1061	2043	602	128	4N
Our 256 bits (Virtex5)	1061	2286	592	129	4N
[3] 384 bits (Virtex5)	8113	30619	9119	127	3518
Our 384 bits (Virtex5)	1573	2286	873	129	4N
Our 384 bits (Artix7)	1573	3029	850	93	4N
Our 384 bits (Spartan3)	1573	3962	2224	33	4N

TABLE 5.4 – Implementation de l'Inversion modulaire sur FPGAs d'Xilinx avec N est la taille des opérandes

3.3 Résultats d'implémentation

La Table 5.4 résume les résultats postimplementation sur FPGA de la version proposée de l'inversion modulaire ainsi que l'architecture de la division. Le design a été décrit dans un langage matériel (VHDL) et a été implémenté sur les FPGAs Xilinx. Afin de vérifier l'exactitude de nos résultats, nous comparons le résultat donné par FPGA avec le code Sagemath à l'annexe, nous présentons les différents résultats après l'implémentation. Ce design a l'avantage d'aptitude à diverses applications comme le ECC et le couplage (multiplication scalaire et l'exponentiation finale). Aussi notre architecture est évolutive aux différents niveaux de sécurité. Les résultats présentés dans ce travail sont comparés avec les travaux les plus récents dans l'état de l'art [3] dans la Table 5.4. Notre design exige moins de ressources en slice, avec une même fréquence et nous améliorons vraiment le nombre de cycles d'horloge par rapport à [3].

4 Cryptosystème couplage

Comme le calcul arithmétique de la courbe elliptique, le calcul du couplage est aussi un arithmétique compliqué dans la cryptographie à clé publique. Comme nous avons déjà dit dans le chapitre 2 la performance de la sécurité du couplage bilinéaire

en cryptographie dépend des courbes algébriques ou les courbes elliptiques. Le niveau de sécurité AES 128 bits, avec la recommandation du NIST, restera utilisable [16] dans le future de la cryptographie.

Parmi les courbes elliptiques proposées dans la littérature, quand j'ai commencé ma thèse, qui fournissent le niveau de sécurité 128 bits et qui sont adaptées aux couplages nous trouvons les courbes Barreto-Naehrig (ou courbes BN). Ces dernières sont définies sur un corps fini de 256 bits. Les courbes BN nécessitent un degré de plongement pas trop grand qui est égale à 12. Il existe d'autres courbes qui sont adaptées au couplage, il y a par exemple les courbes supersingulaires. Parmi ces courbes supersingulaires, il y a les courbes de caractéristique 2, c'est à dire des courbes qui sont définies sur un corps binaire de 1223 bits et le degré de plongement est égale à 4. Il y a aussi les courbes supersingulaires de caractéristique-3 qui sont définies sur un corps de 509 bits avec le degré de plongement est égale à 6 [10]. Comme les courbes de petites caractéristiques ne sont plus considérées sûres, les courbes BN sont restées plus populaires dans nos jours.

Dans cette section nous présentons un cryptosystème du couplage sur les courbes Barreto-Naehrig. Nous avons préparé les composants nécessaires afin d'implémenter le couplage. L'objectif de cette section est l'utilisation de l'implémentation efficace des primitives de corps fini, de la multiplication et de l'inversion, qui ont été décrits au chapitre précédent pour développer le cryptosystème de couplage. Nous avons les composants suivantes :

1. Des composants arithmétiques sur $\mathbb{F}_{p^k}$ ont été développés pour effectuer l'arithmétique sur $\mathbb{F}_{p^2}$, $\mathbb{F}_{p^6}$ et $\mathbb{F}_{p^{12}}$ pour tout p.
2. L'algorithme de Miller et l'exponentiation finale ont été développés en utilisant ces unités arithmétiques.

Nous avons proposé ces architectures afin de développer un couplage qui nécessite moins de cycles d'horloge que les conceptions existantes.

Dans la définition suivante nous rappelons les informations et les paramètres du couplage que nous avons utilisé dans ce travail (plus de détails présentés dans le chapitre 2 section 4) :

Définition 8. *La cryptographie basée sur le couplage [32] est une application bilinéaire.*
Le couplage est une application de $G_1 \times G_2 \longrightarrow G_3$ où G_1 et G_2 sont des groupes additifs et G_3 est un groupe multiplicatif.
E est une courbe elliptique définie sur $\mathbb{F}_p$.
Le degré de plongement de E est k.

L'entier r un diviseur premier de $\#E(\mathbb{F}_p)$.
Le degré de plongement k de E est le plus petit entier tel que $k \geq 1$ *et* $q^k \equiv 1$ *(mod r).*

Plusieurs couplages utilisent les courbes BN pour la cryptographie : le couplage de Tate [64], le couplage de Ate [22], le couplage de R-Ate [17] et le couplage de Ate optimal [23] qui sont les plus populaires. Nous avons choisi dans ce travail d'utiliser le couplage de Ate optimale. Ce dernier choisit G_1 le sous-groupe cyclique d'ordre r de $E(\mathbb{F}_p)$ et G_2 le sous-groupe cyclique d'ordre r de $E(\mathbb{F}_{p^k})$ et G_3 le sous-groupe de$\mathbb{F}^*_{p^k}$ d'ordre r.

Ce couplage est de type asymétrique car les deux sous groupes G_1 et G_2 sont différents. Nous avons choisi dans cette section le couplage de Ate optimal CAO : $G1 \times G2 \longrightarrow G3$, car le choix des groupes, les types de corps ainsi que d'autres paramètres ont un impact important sur la sécurité et le coût de calcul matériel et logiciel du couplage. Dans nos travaux, nous avons choisi les courbes Barreto-Naehrig (ou courbes BN) [21] pour calculer le couplage Ate optimal, puisque ces courbes sont les plus adaptées aux couplages. Elles sont particulièrement adaptées pour le calcul du couplage de Ate optimal avec un niveau de sécurité de 128 bits en choisissant un entier p premier de taille 256 bits. Ce chapitre propose plusieurs composants du cryptoprocesseur pour le calcul des couplages sur les courbes BN. Le Field programmable gate array (FPGA) sont les plate-formes appropriées pour nos implémentations matérielles des algorithmes des primitives du couplage. Dans ce chapitre, nous développons des composants configurables pour calculer la multiplication et l'inversion sur $\mathbb{F}_{p^2}$, $\mathbb{F}_{p^6}$ et $\mathbb{F}_{p^{12}}$. Les composants arithmétiques configurables $\mathbb{F}_{p^K}$ proposées donne lieu à une amélioration significative de la performance du couplage de Ate-Optimal sur les courbes BN.

Calcul du couplage

L'algorithme 17 montre le calcul du couplage de Ate optimal. Il se compose de deux étapes principales : le calcul de la fonction de Miller et l'exponentiation finale. La première partie est calculée par l'une des versions optimisées de l'algorithme de Miller présenté dans [56].

Les courbes BN admettent une twist de degré 6, ces références [10, 21, 22] présentent plus des définitions pour les twists. Ce qui signifie que le point Q est remplacé sur un point Q' qui est défini sur $\mathbb{F}_{p^{k/6}} = \mathbb{F}_{p^2}$. Les fonctions de ligne $l_{T,T}(P)$ et $l_{T,Q}(P)$ sont calculées sur $\mathbb{F}_{p^2}$ au lieu de $\mathbb{F}_{p^{12}}$. Nous obtenons la courbe $E' = \mathbb{F}_{p^2} : y2 = x3 + b/\xi$ qui est définie sur $\mathbb{F}_{p^2}$. Le paramètre $\xi \in \mathbb{F}_{p^2}$ est un élément

Algorithme 17 : Couplage de Ate-Optimal sur les courbes BN

Input : $P \in G_1 \in E(\mathbb{F}_q)$ and $Q \in G_2 \in E'(\mathbb{F}_{q^{k/6}})$
Output : Couplage (Q,P)

1 $f \leftarrow 1\,;\, T \leftarrow Q\,;$
2 **for** $i \leftarrow L-2$ ***downto*** 0 **do**
3 $\quad f \leftarrow f^2 \cdot l_{T,T}(P)\,;\, T \leftarrow 2T\,;$
4 $\quad$ **if** $s_i = -1$ **then**
5 $\quad\quad f \leftarrow f \cdot l_{T,-Q}(P)\,;\, T \leftarrow T - Q\,;$
6 $\quad$ **else if** $s_i = 1$ **then**
7 $\quad\quad f \leftarrow f \cdot l_{T,Q}(P)\,;\, T \leftarrow T + Q\,;$
8 $Q_1 \leftarrow \pi_p(Q)\,;\, Q_2 \leftarrow \pi_p(Q_1)\,;$
9 $f \leftarrow f \cdot l_{T,Q_1}(P)\,;\, T \leftarrow T + Q_1\,;$
10 $f \leftarrow f \cdot l_{T,Q_2}(P)\,;\, T \leftarrow T - Q_2\,;$
11 $f \leftarrow f^{(p^{12}-1)/r}\,;$
12 **return** $f\,;$

qui n'est ni un carré ni un cube dans $\mathbb{F}_{p^2}$. Et ξ doit être choisi de telles sorte que r divise $\#E'(\mathbb{F}_{p^2})$. C'est à dire les calculs du couplage Ate-Optimal sont limités aux deux points P et Q' respectivement de $E(\mathbb{F}_p)$ et $E'(\mathbb{F}_{p^2})$.

Dans la ligne 8 de l'algorithme du couplage, nous utilisons l'opération de Frobenius. Soit $E[r]$ le sous-groupe r-torsion de E et π_p l'endomorphisme de Frobenius $\pi_p : E \longrightarrow E$ est défini par :

$$\pi_p(x; y) = (x^p; y^p)$$

.

Soit $G_1 = E[r] \cap Ker(\pi_p - 1) = E(\mathbb{F}_p)[r]$
Soit $G_2 = E[r] \cap Ker(\pi_p - p) \subset E(\mathbb{F}_{p^{12}})[r]$
Soit $G_3 = \mu_r \subset \mathbb{F}^*_{p^{12}}$.
Comme nous travaillons avec une courbe BN, r est un entier premier et $G_1 = E(\mathbb{F}_p)[r] = E(\mathbb{F}_p)$. Le couplage de Ate Optimal sur la courbe BN est défini par les deux propriétés : non dégénérescence et bilinéaire. L'équation de ce couplage est donnée par la fonction suivante :

$$AO : G_2 \times G_1 \longrightarrow G_3$$

$$(Q, P) \longmapsto (f_{6t+2,Q}(P).l_{[6t+2]Q,\pi(Q)}(P).l_{[6t+2]Q,\pi^2(Q)}(P))^{\frac{p^{12}-1}{r}}$$

L'algorithme 17 présente les étapes du calcul pour le couplage de Ate-Optimal dans nos travaux. Cet algorithme est composé sur deux grands sous algorithmes, qui

sont l'algorithme de Miller et l'algorithme de l'exponentiation finale. Dans l'algorithme de Miller nous avons utilisé les opérations d'addition et de soustractions des points sur la courbe. Dans notre cas le sous algorithme de Miller est présenté par les lignes de 2 à 7. Le rôle de la boucle de Miller est de calculer la fonction rationnelle $f_{6t+2,Q}$ avec une évaluation dans le point P. Ensuite le produit des fonctions droites $l_{[6t+2]Q,\pi(Q)}(P).l_{[6t+2]Q,\pi^2(Q)}(P)$ est multiplié par $f_{6t+2,Q}(P)$ et est calculé dans les lignes 8 à 10. Et nous finissons le calcul du couplage par l'exponentiation finale dans la lignes 11.

Boucle Miller

Dans l'algorithme 17, la boucle de Miller nécessite un paramètre $6t+2$. Ce dernier a une longueur $L = 65$ bits, avec 7 est le poids de Hamming. Cela signifie que l'implémentation de la boucle de Miller nécessite 64 étapes de calcul de doublement et 6 étapes d'addition/soustraction. Présentons dans ce qui suit le détail de calcul pour effectuer un doublement et/ou une addition avec l'évaluation du résultat dans un point P.

Doublement d'un Point de la courbe et évaluation en un point P

Pour effectuer l'opération du doublement d'un point, nous choisissons un point $Q = (X_Q, Y_Q, Z_Q)$ du twiste de la courbe $E'(\mathbb{F}_{p^2})$. Donc les coordonnées (X_Q, Y_Q, Z_Q) du point Q appartiennent $\mathbb{F}_{p^2}$. Dans notre travail nous avons choisi d'utiliser le système de coordonnées Jacobiennes. Donc le doublement du point Q est le point $T = 2Q = (X_T, Y_T, Z_T)$ qui est défini par les équations suivantes :

$$X_T = 9X_Q^4 - 8X_QY_Q^2.$$

$$Y_T = 3X_Q^2(4X_QY_Q - X_T) - 8Y_Q^4.$$

$$Z_T = 2Y_QZ_Q.$$

L'équation du tangente calculée pour le doublement doit être évaluée avec le point P. Soit le point P de la courbe $E(\mathbb{F}_p)$, P est représenté avec le système de coordonnées affines donc $P = (x_P, y_P)$. Les coordonnées (x_P, y_P) du point P appartiennent à $\mathbb{F}_p$. L'évaluation en P de l'équation du tangente dans le point T est la suivante :

$l_{T,T}(P) = 4Y_T Z_T^3 y_P - (6X_T^2 Z_T^2 x_P)w + (6X_T^3 - 4Y_T^2)w^2 \in \mathbb{F}_{p^{12}}$.

Addition de deux Points de la courbe et évaluation en un point P

Pour effectuer l'opération d'addition de deux points Q et T, nous choisissons deux points du twiste de la courbe $E'(\mathbb{F}_{p^2})$. Comme dans notre travail nous utilisons les coordonnées Jcobiennes, les points sont présentés de la manière suivante : $Q = (X_Q, Y_Q, Z_Q)$ et $T = (X_T, Y_T, Z_T)$. Les coordonnées de Q et T appartiennent à $\mathbb{F}_{p^2}$. Donc le résultat de l'addition est $R = T + Q$ avec $R = (X_R, Y_R, Z_R)$ qui est définie par les équations suivantes :

$X_R = (2Y_Q Z_T^3 - 2Y_T)^2 - 4(X_Q Z_T^2 - X_T)^3 - 8(X_Q Z_T^2 - X_T)^2 X_T$.

$Y_R = (2Y_Q Z_T^3 - 2Y_T)(4(X_Q Z_T^2 - X_T)2X_T - X_R) - 8Y_T(X_Q Z_T^2 - XT)3$.

$Z_R = 2Z_T(X_Q Z_T^2 - X_T)$.

L'équation de droite calculée pour l'addition doit être évaluée avec le point P. Soit le point P de la courbe $\mathbb{F}_p$, p est représenté avec le système de coordonnées affines donc $P = (x_P, y_P)$. Les coordonnées (x_P, y_P) du point P appartiennent à $\mathbb{F}_p$. L'évaluation en P de l'équation du droite passant par les points T et Q est la suivante :

$l_{T,Q}(P) = 4Z_T(X_Q Z_T^2 - X_T)y_P - 4x_P(Y_Q Z_T^3 + Y_T)w + (4X_Q(Y_Q Z_T^3 X_Q - Y_T) - 4Y_Q Z_T(X_Q Z_T^2 - X_T))w^2 \in \mathbb{F}_{p^{12}}$.

Application Frobenius et étape d'addition finale

Une autre étape nécessaire pour effectuer le couplage de Ate Optimale c'est l'application de Frobenius avec une étape d'addition, nous devons ajouter à f le produit des lignes à travers les points Q_1 et $Q_2 \in E'(\mathbb{F}_{p^2})$. Les équations des lignes sont $l_{[6t+2]Q,Q_1}(P)$ et $l_{[6t+2]Q,Q_2}(P)$. Les points Q_1 et Q_2 peuvent être trouvés en appliquant l'opérateur de Frobenius de la manière suivante : $Q_1 = \pi(Q)$ et $Q_2 = \pi^2(Q)$

5 Arithmétique sur les extensions du corps

5.1 Multiplication sur $\mathbb{F}_{p^2}$

Nous avons fait l'implémentation hardware de la multiplication sur $\mathbb{F}_{p^2}$ dans l'algorithme 18 en utilisant une multiplication modulaire. Soit $P(u) = u^2 + 5$ le polynôme irréductible définissant l'extension quadratique.
Soit $A = (a_0 + a_1 u)$ et $B = (b_0 + b_1 u)$ tel que a_0, a_1, b_0 et $b_1 \in \mathbb{F}_p$.
$C = A \cdot B = (a_0 + a_1 u) \cdot (b_0 + b_1 u) = (a_0 b_0 - 5a_1 b_1) + ((a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1)u$;

Algorithme 18 : Multiplication sur $\mathbb{F}_{p^2} = \dfrac{\mathbb{F}_p[u]}{(u^2+5)}$

Input : A and $B \in \mathbb{F}_{p^2}$ with $A = a_0 + a_1 \cdot u$ et $B = b_0 + b_1 \cdot u$
Output : $C = A \cdot B \in \mathbb{F}_{p^2}$ with $C = c_0 + c_1 \cdot u$

1 $s \leftarrow \text{addmod}(a_0, a_1)$;
2 $t \leftarrow \text{addmod}(b_0, b_1)$;
3 $d_0 \leftarrow \text{mul(s,t)}$;
4 $d_1 \leftarrow \text{mul } (a_0, b_0)$;
5 $d_2 \leftarrow \text{mul}(a_1, b_1)$;
6 $d_3 \leftarrow \text{submod}(d_0, d_1)$;
7 $c_1 \leftarrow \text{submod}(d_3, d_2)$;
8 $d_4 \leftarrow 5 \cdot d_2$;
9 $c_0 \leftarrow \text{submod } (d_1, d_4)$;
10 **return** $C = c_0 + c_1 \cdot u$;

5.2 Multiplication sur $\mathbb{F}_{p^6}$

Soit $P(v) = v^3 - u$ le polynôme irréductible définissant l'extension cubique de $\mathbb{F}_{p^2}$.
Soit $A = (a_0 + a_1 v + a_2 v^2)$ et $B = (b_0 + b_1 v + b_2 v^2)$ tel que a_0, a_1, a_2, b_0, b_1 et $b_2 \in \mathbb{F}_{p^2}$.
$C = A \cdot B = (a_0 + a_1 v + a_2 v^2) \cdot (b_0 + b_1 v + b_2 v^2) = c_0 + c_1 v + c_2 v^2$; avec :
$c_0 = [(a_1 + a_2)(b_1 + b_2) - a_1 b_1 - a_2 b_2] \cdot u + a_0 b_0$;
$c_1 = (a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1 + a_2 b_2 \cdot u$;
$c_2 = (a_0 + a_2)(b_0 + b_2) - a_0 b_0 - a_2 b_2 + a_1 b_1 1$;

5.3 Multiplication sur $\mathbb{F}_{p^{12}}$

Soit $P(w) = w^2 - v$ le polynôme irréductible définissant l'extension cubique de $\mathbb{F}_{p^6}$.

Algorithme 19 : Multiplication sur $\mathbb{F}_{p^6} = \dfrac{\mathbb{F}_{p^2}[v]}{(v^3 - u)}$

Input : A and $B \in \mathbb{F}_{p^6}$
with $A = a_0 + a_1 \cdot v + a_2 \cdot v^2$ et $B = b_0 + b_1 \cdot v + b_2 \cdot v^2$
Output : $C = A \cdot B \in \mathbb{F}_p^6$ with $C = c_0 + c_1 \cdot v + c_2 \cdot v^2$

1 $t_0 \leftarrow$ Multiplication sur $F_{p^2}(a_0, b_0)$;
2 $t_1 \leftarrow$ Multiplication sur $F_{p^2}(a_1, b_1)$;
3 $t_2 \leftarrow$ Multiplication sur $F_{p^2}(a_2, b_2)$;
4 $t_3 \leftarrow$ Addition sur $F_{p^2}(a_1, a_2)$;
5 $t_4 \leftarrow$ Addition sur $F_{p^2}(b_1, b_2)$;
6 $t_5 \leftarrow$ Multiplication sur $F_{p^2}(t_3, t_4)$;
7 $t_6 \leftarrow$ soustraction sur $F_{p^2}(t_5, t_1)$;
8 $t_7 \leftarrow$ soustraction sur $F_{p^2}(t_6, t_2)$;
9 $t_8 \leftarrow$ Multiplication sur $F_{p^2}(t_7, u)$;
10 $c_0 \leftarrow$ Addition sur $F_{p^2}(t_8, t_0)$;
11 $t_9 \leftarrow$ Addition sur $F_{p^2}(a_0, a_1)$;
12 $t_{10} \leftarrow$ Addition sur $F_{p^2}(b_0, b_1)$;
13 $t_{11} \leftarrow$ Multiplication sur $F_{p^2}(t_9, t_{10})$;
14 $t_{12} \leftarrow$ soustraction sur $F_{p^2}(t_{11}, t_0)$;
15 $t_{13} \leftarrow$ soustraction sur $F_{p^2}(t_{12}, t_1)$;
16 $t_{14} \leftarrow$ Multiplication sur $F_{p^2}(t_2, u)$;
17 $c_1 \leftarrow$ Addition sur $F_{p^2}(t_{13}, t_{14})$;
18 $t_{15} \leftarrow$ Addition sur $F_{p^2}(a_0, a_2)$;
19 $t_{16} \leftarrow$ Addition sur $F_{p^2}(b_0, b_2)$;
20 $t_{17} \leftarrow$ Multiplication sur $F_{p^2}(t_{15}, t_{16})$;
21 $t_{18} \leftarrow$ soustraction sur $F_{p^2}(t_{17}, t_0)$;
22 $t_{19} \leftarrow$ soustraction sur $F_{p^2}(t_{18}, t_2)$;
23 $c_2 \leftarrow$ Addition sur $F_{p^2}(t_{19}, t_1)$;
24 **return** $C = c_0 + c_1 \cdot v + c_2 \cdot v^2$;

Soit $A = (a_0 + a_1 w$ et $B = (b_0 + b_1 w$ tel que a_0, a_1, b_0 et $b_1 \in \mathbb{F}_{p^6}$.
$C = A \cdot B = (a_0 + a_1 w) \cdot (b_0 + b_1 w) = c_0 + c_1 w$; avec :
$c_0 = a_0 b_0 + a_1 b_1 \cdot v$.
$c_1 = (a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1$.

Algorithme 20 : Multiplication sur $\mathbb{F}_{p^{12}} = \dfrac{\mathbb{F}_{p^6}[w]}{(w^2 - v)}$

Input : A and $B \in \mathbb{F}_{p^{12}}$
with $A = a_0 + a_1 \cdot w$ et $B = b_0 + b_1 \cdot w$
Output : $C = A \cdot B \in \mathbb{F}_{p^{12}}$ with $C = c_0 + c_1 \cdot w$

1 $t_0 \leftarrow$ Multiplication sur $F_{p^6}(a_0, b_0)$;
2 $t_1 \leftarrow$ Multiplication sur $F_{p^6}(a_1, b_1)$;
3 $t_2 \leftarrow$ Multiplication sur $F_{p^6}(t_1, v)$;
4 $t_3 \leftarrow$ Addition sur $F_{p^6}(t_0, t_2)$;
5 $t_4 \leftarrow$ Addition sur $F_{p^6}(a_0, a_1)$;
6 $t_5 \leftarrow$ Addition sur $F_{p^6}(b_0, b_1)$;
7 $t_6 \leftarrow$ Multiplication sur $F_{p^6}(t_4, t_5)$;
8 $t_7 \leftarrow$ soustraction sur $F_{p^6}(t_6, t_0)$;
9 $t_8 \leftarrow$ soustraction sur $F_{p^6}(t_7, t_1)$;
10 **return** $C = c_0 + c_1 \cdot w$;

5.4 Inversion sur $\mathbb{F}_{p^2}$

Pour calculer une inversion dans l'extension quadratique nous avons besoin de 2 multiplications modulaires, une inversion modulaire, 2 carrés modulaires et une addition. L'algorithme 21 présente la méthode pour l'inversion modulaire sur $\mathbb{F}_{p^2}$,

$$(a_0 + a_1 u)^{-1} = \frac{a_0 - a_1 u}{a_0^2 + 5a_1^2}$$

5.5 Inversion sur $\mathbb{F}_{p^6}$

Soit $A = (a_0 + a_1 v + a_2 v^2)$ tel que a_0, a_1 et $a_2 \in \mathbb{F}_{p^2}$.
$C = c_0 + c_1 v + c_2 v^2 = A^{-1} = (a_0 + a_1 v + a_2 v^2)^{-1} = \dfrac{M + Nv + Ov^2}{a_1 Ou + a_0 M + a_2 Nu}$ avec :
$M = a_0^2 - a_1 a_2 u$.
$N = a_2^2 u - a_0 a_1$.
$O = a_1^2 - a_0 a_2$.

Algorithme 21 : Inversion sur $\mathbb{F}_{P^2} = \dfrac{\mathbb{F}_P[u]}{(u^2+5)}$

Input : A with $A = a_0 + a_1 \cdot u$
Output : $C = A^{-1} \in \mathbb{F}_{p^2}$ with $C = c_0 + c_1 \cdot u$

1 $t_0 \leftarrow$ Carré sur $\mathbb{F}_p(a_0)$;
2 $t_1 \leftarrow$ Carré sur $\mathbb{F}_p(a_1)$;
3 $t_2 \leftarrow -5 \cdot t_1$;
4 $t_3 \leftarrow$ sub sur $\mathbb{F}_p(t_0, t_2)$;
5 $t_4 \leftarrow$ inversion sur $\mathbb{F}_p(t_3)$;
6 $c_0 \leftarrow$ multiplication sur $\mathbb{F}_p(a_0, t_4)$;
7 $t_5 \leftarrow$ multiplication sur $\mathbb{F}_p(a_1, t_4)$;
8 $c_1 \leftarrow -t_5$;
9 **return** $C = c_0 + c_1 \cdot u$;

L'algorithme 22 présente la méthode d'inversion modulaire sur $\mathbb{F}_{P^6}$.

Algorithme 22 : Inversion sur $\mathbb{F}_{p^6} = \dfrac{\mathbb{F}_{p^2}[v]}{(v^3-u)}$

Input : $A \in \mathbb{F}_{p^6}$ with $A = a_0 + a_1 \cdot v + a_2 \cdot v^2$
Output : $C = A^{-1} \in \mathbb{F}_{p^6}$ with $C = c_0 + c_1 \cdot v + c_2 \cdot v^2$

1 $t_0 \leftarrow$ Carré sur $F_{p^2}(a_0)$;
2 $t_1 \leftarrow$ Carré sur $F_{p^2}(a_1)$;
3 $t_2 \leftarrow$ Carré sur $F_{p^2}(a_2)$;
4 $t_3 \leftarrow$ Multiplication sur $F_{p^2}(a_0, a_1)$;
5 $t_4 \leftarrow$ Multiplication sur $F_{p^2}(a_0, a_2)$;
6 $t_5 \leftarrow$ Multiplication sur $F_{p^2}(a_1, a_2)$;
7 $t_7 \leftarrow$ Multiplication sur $F_{p^2}(t_5, u)$;
8 $t_8 \leftarrow$ soustraction sur $F_{p^2}(t_0, t_7)$;
9 $t_9 \leftarrow$ Multiplication sur $F_{p^2}(u, t_2)$;
10 $t_{10} \leftarrow$ soustraction sur $F_{p^2}(t_9, t_3)$;
11 $t_{11} \leftarrow$ soustraction sur $F_{p^2}(t_1, t_4)$;
12 $t_{12} \leftarrow$ Multiplication sur $F_{p^2}(a_0, t_8)$;
13 $t_{13} \leftarrow$ Multiplication sur $F_{p^2}(u, a_2)$;
14 $t_{14} \leftarrow$ Multiplication sur $F_{p^2}(t_{13}, t_{10})$;
15 $t_{15} \leftarrow$ Multiplication sur $F_{p^2}(t_{12}, t_{14})$;
16 $t_{16} \leftarrow$ Multiplication sur $F_{p^2}(u, a_1)$;
17 $t_{17} \leftarrow$ Multiplication sur $F_{p^2}(t_{16}, t_{11})$;
18 $t_{18} \leftarrow$ addition sur $F_{p^2}(t_{15}, t_{17})$;
19 $t_{19} \leftarrow$ Inversion sur $F_{p^2}(t_{18})$;
20 $c_0 \leftarrow$ Multiplication sur $F_{p^2}(t_8, t_{19})$;
21 $c_1 \leftarrow$ Multiplication sur $F_{p^2}(t_{10}, t_{19})$;
22 $c_2 \leftarrow$ Multiplication sur $F_{p^2}(t_{11}, t_{19})$;
23 **return** $C = c_0 + c_1 \cdot v + c_2 \cdot v^2$;

5.6 Inversion sur $\mathbb{F}_{p^{12}}$

L'algorithme 23 présente la méthode d'inversion modulaire sur $\mathbb{F}_{p^{12}}$. Pour effectuer une inversion sur $\mathbb{F}_{p^{12}}$ nous avons utilisé deux carrés, trois multiplications, une soustraction et une inversion sur $\mathbb{F}_{p^{12}}$

Algorithme 23 : Multiplication sur $\mathbb{F}_{p^{12}} = \dfrac{\mathbb{F}_{p^6}[w]}{(w^2 - v)}$

Input : $A \in \mathbb{F}_{p^{12}}$
with $A = a_0 + a_1 \cdot w$
Output : $C = A^{-1} \in \mathbb{F}_{p^{12}}$ with $C = c_0 + c_1 \cdot w$

1 $t_0 \leftarrow$ Carré sur $\mathbb{F}_{p^6}(a_0)$;
2 $t_1 \leftarrow$ Carré sur $\mathbb{F}_{p^6}(a_1)$;
3 $t_2 \leftarrow$ Multiplication sur $\mathbb{F}_{p^6}(t_1, v)$;
4 $t_3 \leftarrow$ Soustraction sur $\mathbb{F}_{p^6}(t_0, t_2)$;
5 $t_4 \leftarrow$ Inversion sur $\mathbb{F}_{p^6}(t_3)$;
6 $c_0 \leftarrow$ Multiplication sur $\mathbb{F}_{p^6}(a_0, t_4)$;
7 $t_5 \leftarrow$ Multiplication sur $\mathbb{F}_{p^6}(a_1, t_4)$;
8 $c_1 \leftarrow -t_5$;
9 **return** $C = c_0 + c_1 \cdot w$;

6 Conclusion

Dans ce chapitre, nous avons présenté une implémentation basée sur FPGA pour le calcul des cryptosystèmes à clé publique. Nous avons implémenté une inversion modulaire, cette inversion est nécessaire pour le couplage ou bien ECC si nous travaillons en coordonnées affines. Dans ce travail nous avons traité le sujet de comparaison des nombres des multiplications utilisées pour remplacer une inversion. Ainsi, Nous avons utilisé cette même inversion pour effectuer le calcul de l'exponentiation finale dans le calcul du couplage de Ate-optimal. Pour calculer l'exponentiation finale nous avons passé par les différentes étapes de la tour d'extension. Nous avons exploité le concept de la multiplication que nous avons développé pour présenter une implémentation matérielle efficace des inversions et des multiplications sur les extensions du corps.

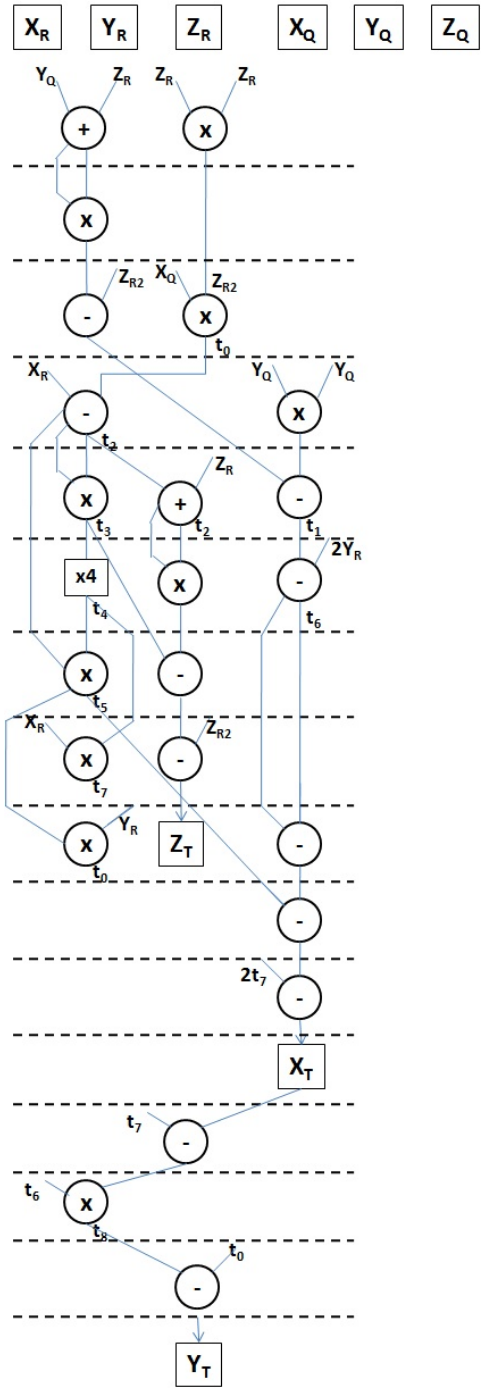

FIGURE 5.1 – flot de données pour l'algorithme d'addition .

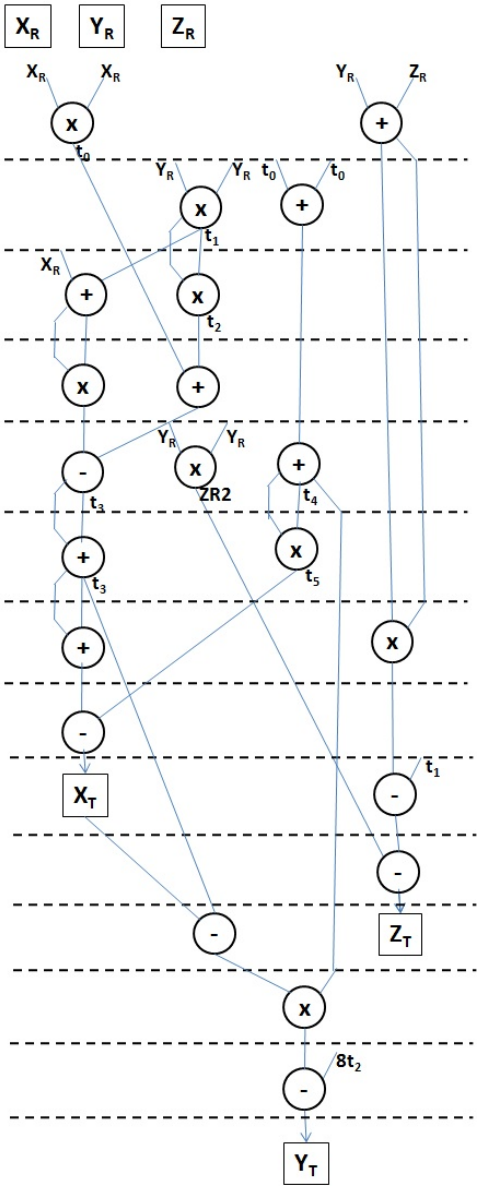

FIGURE 5.2 – flot de données pour l'algorithme de doublement.

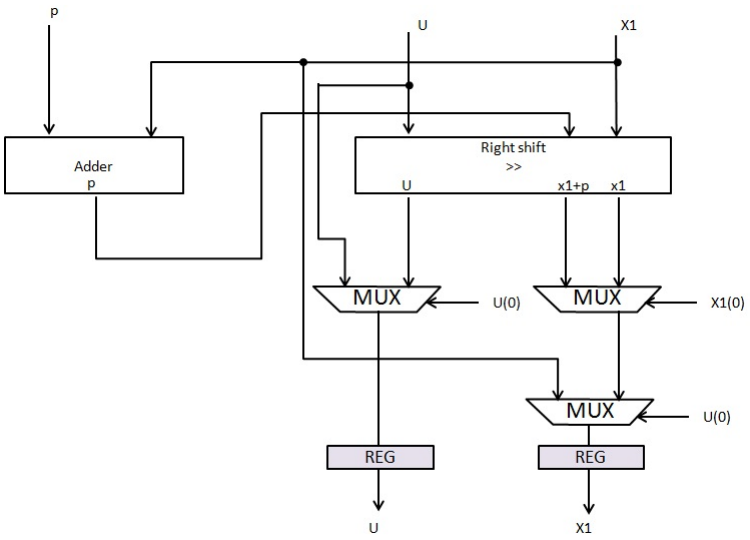

FIGURE 5.3 – Chemin de données du Bloc 1.

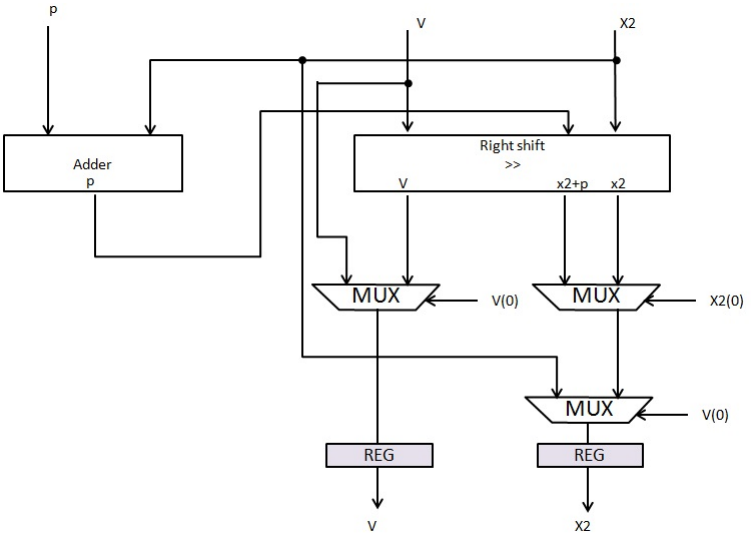

FIGURE 5.4 – Chemin de données du Bloc 2.

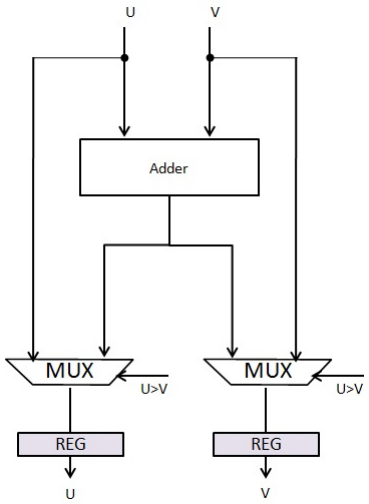

FIGURE 5.5 – Addition u et v.

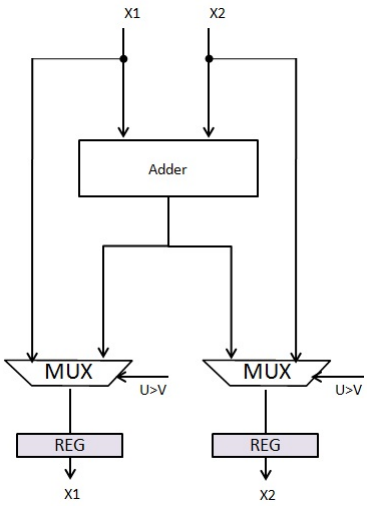

FIGURE 5.6 – Addition x1 et x2.

Chapitre 6

Conclusion Générale

Dans ce chapitre nous concluons la thèse en soulignant les principales implémentations proposées et les principales contributions. Nous discutons également les orientations possibles pour les perspectives.

1 Conclusion

Dès la fin des années 90 du dernier siècle, il y a eu une grande croissance dans les méthodes d'implémentation efficace et sécurisé de la cryptographie asymétrique. C'est le début des époques de la courbe elliptique et des couplages. La croissance, principalement dans l'implémentation efficace en matérielle et logicielle, a permis d'utiliser des algorithmes et des techniques de conception de plus en plus efficaces pour calculer les opérations dans les corps finis.

Parallèlement à cela, des implémentations de plus en plus robustes contre des puissantes attaques physiques, comme les attaques par injection des fautes et les attaques par canaux cachés, ont été développées. Cette thèse vise à offrir les meilleures techniques utiles pour la réduction du temps et des ressources pour la cryptographie basée sur le couplage et le ECC pour des implémentations sur la plate-forme FPGA. Les contributions de ce travail sont conclues comme suit :

1. Dans le chapitre 4 : L'objectif était de proposer une conception efficace pour la multiplication modulaire de Montgomery. L'objectif de cette conception est d'optimiser les ressources et la latence.

   Nous avons présenter dans ce chapitre nos différentes architectures. Nous avons optimisé le nombre de cycles d'horloge nécessaire pour calculer une multiplication modulaire, ainsi que les ressources sur la carte FPGA. Nous avons implémenté toutes nos architectures avec un nombre de cycles d'horloge fixe.

Ce design permet d'optimiser en ressources les implémentations matérielles. Nos architectures sont présentées sous forme des cellules élémentaires simples. Chaque cellule effectue un calcul simple, une multiplication et/ou une addition. Dans cette thèse nous avons utilisé la méthode CIOS de la multiplication de Montgomery. Ce travail a mis l'accent sur l'utilisation des composants FPGA intégrés (les Digital signal processing (DSP)) pour le développement de cette architecture. Notre idée est d'utiliser le maximum des DSPs.

L'architecture systolique que nous avons utilisé dans ce travail fait diminuer les besoins en ressources dans les implémentations matérielles. Notre contribution dans ce chapitre est de combiner une architecture systolique, qui est supposée être le meilleur choix pour les implémentations matérielles, avec la méthode CIOS de la multiplication modulaire de Montgomery. Nous avons optimisé le nombre de cycles d'horloge requis pour calculer un MMM à n-bit et nous avons réduit l'utilisation des ressources FPGA. Nous avons implémenté aussi la multiplication modulaire dans un nombre fixe de cycles d'horloge. Selon nos connaissances, c'est la première fois qu'une implémentation en matériel ou en logiciel de la multiplication modulaire de Montgomery, qui est adaptée à divers niveaux de sécurité, est réalisé en seulement 33 cycles d'horloge.

Afin d'implémenter la multiplication modulaire de Montgomery pour le niveau de la sécurité fixe, nous devons choisir l'architecture la plus adéquate. Les résultats présentés dans ce travail sont comparés avec les travaux précédents [29,30,60,62] dans la Table 4.6 et dans la Table 4.5. Nous avons remarqué que nos résultats sont meilleurs que [62] considérant chaque point de comparaison, à savoir le nombre de slices et le nombre de cycles d'horloge. Compte tenu du nombre de slices, nous rappelons que [62] ont utilisé une mémoire externe afin d'optimiser le nombre de slices utilisé par leurs algorithmes. Compte tenu de la comparaison avec [60], notre design nécessite moins de nombre de slices, et une meilleure fréquence. Nous avons amélioré vraiment le nombre de cycles d'horloge.

Notre design a effectué la multiplication de Montgomery en 33 et 66 cycles d'horloge pour la longueur de 512 et 1024 bits pour les niveaux de sécurité AES-256 et AES-512, tandis que [60] ont effectué la multiplication en 1540 cycles d'horloge pour le niveau de sécurité AES-256 et 3076 pour le niveau de sécurité AES-512 car il a été tenu compte dans leurs mesures expérimentales des temps d'accès à la BRAM externe. Nous n'avons pas tenu compte de ces temps, d'après Koç et al, le total des temps de lecture pour la CIOS est

$6s^2+7s+2$ et pour l'écriture il est $2s^2+5s+1$. En plus, leurs expérimentations ont été menées sur VIRTEX E et les nôtres sur Artix 7.

2. Dans le chapitre 5 : Nous nous sommes concentrés sur les implémentations des cryptosystèmes asymétriques tels que le ECC et les couplages. Nous avons utilisé les courbes BN, qui sont des courbes "pairing friendly". Nous avons utilisé aussi notre architecture systolique efficace de MMM pour effectuer un carré ou une multiplication dans l'implémentation du ECC ou des couplages. Dans ce chapitre nous avons détaillé l'implémentation d'une inversion modulaire. Une inversion modulaire est nécessaire pour implémenter une exponentiation finale dans un couplage. Nous avons desservi de cette architecture d'inversion pour parler d'un cryptosystème ECC en coordonnées affines. Nous avons présenté une implémentation matérielle efficace de l'inversion modulaire ainsi que de l'algorithme de division sur les corps finis de grands nombres. Notre architecture a été adaptée avec les caractéristiques des FPGA. Elle a présenté une bonne performance en considérant l'efficacité $latence \times ressources$.

2 Futurs Travaux

Le travail proposé dans chaque chapitre de cette thèse peut être étendu à d'autres recherches qui sont en cours d'implémentation.

1. Dans le chapitre 4 nous avons détaillé l'implémentation matérielle de la méthode CIOS. Cette méthode manipule des mots de la même taille. Par exemple si nous travaillons dans le niveau de sécurité 128-bit, nous utilisons des opérandes de taille 256 bits, donc notre architecture NW-8 manipule des mots de taille 32 bits. Le but de ce travail est de modifier l'algorithme de base de la méthode CIOS afin de manipuler des mots de tailles différentes. L'idée de ce travail est d'adapter les mots utilisés dans notre architecture avec les DSP48 intégrés sur les cartes FPGA proposées pas Xilinx ou Altéra. Les dispositifs FPGA modernes offrent plusieurs composants intégrés, tels que les DSPs pour effectuer une multiplication de 25 bits $\times$ 18 bits.
2. Dans le chapitre 5 nous avons présenté les cryptosystèmes ECC et le couplage. Les cryptoprocesseurs ont été proposés pour calculer les ECC et le couplage sur les courbes BN. Les implémentations dans cette thèse sont tous effectuées sur les cartes FPGA en matérielle. Cependant, il existe des powerPc ou bien des processeurs ARM intégrés sur FPGA. Par conséquent et à l'avenir, ces propriétés pourraient être exploitées pour proposer des implémentations mixtes.

Ces approches de la conception mixte logicielle/matérielle (ou co-design) ont été proposées afin d'aider les concepteurs à rechercher une adéquation application architecture (AAA) satisfaisant les différentes contraintes de conception comme le coût, la performance, la surface, la consommation, la flexibilité etc ...

Bibliographie

[1] P. van Oorschot A. Menezes and S. Vanstone. Handbook of applied cryptography. *CRC Press, www.cacr.math.uwaterloo.ca/hac*, 1996.

[2] M. Scott A.J. Devegili and R. Dahab. Implementing cryptographic pairings over barreto-naehrig curves. pages 197–207. Pairing 2007. LNCS 4575, 2007.

[3] Karim Bigou and Arnaud Tisserand. Improving modular inversion in rns using the plus-minus method. http ://eprint.iacr.org/2015/193.pdf, 2015.

[4] Karim Bigou and Arnaud Tisserand. Single base modular multiplication for efficient hardware rns implementations of ecc. In *Conference on Cryptographic Hardware and Embedded Systems*, pages 123–140, September 2015.

[5] B. Lynn Boneh and H. Shacham. Short signatures from the weil pairing. *Journal of Cryptology*, 17 :297–319, 2004.

[6] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. volume 2139, pages 213–229. In Joe Kilian, editor, Advances in Cryptology - CRYPTO 200, 2001.

[7] F. Brezing and Z. Weng. Elliptic curves suitable for pairing based cryptography. *Designs, Codes and Cryptography*, 37(1) :133–141, 2005.

[8] H. Cohen and G. Freyr. Handbook of elliptic and hyperelliptic cryptography. 2006.

[9] Ronald Cramer. An optimized hardware architecture for the montgomery multiplication algorithm. In *Public Key Cryptography - PKC 2008*, volume 4939 of *Lecture Notes in Computer Science*, pages 214–228. Springer Berlin Heidelberg, 2008.

[10] A. Menezes D. Hankerson and M. Scott. Software implementation of pairings. *In : Joye, M., Neven, G. (eds.) Identity-Based Cryptography*, 2008.

[11] A. Menezes D. Hankerson and M. Scott. Software implementation of pairings. page 188. M. Joye and G. Neven, editors, Identity-based Cryptography, Cryptology and Information Security Series, chapter 12, 2009.

[12] A. Menezes D. Hankerson and S. Vanstone. Guide to elliptic curve cryptography. In *Springer*.

[13] P. Schwabe H. Scharwaechter M. Langenberg D. Auras G. Ascheid D. Kammler, D. Zhang and R. Mathar. Designing an asip for cryptographic pairings over barreto-naehrig curves. pages 254–271. CHES 2009, LNCS 5747, 2009.

[14] Ben Lynn et Hovav Shacham Dan Boneh. Short signatures from the weil pairing. *Advances in Cryptology - ASIACRYPT*, 4223 :736–754, 2001.

[15] A. Durand. Efficient ways to implement elliptic curve exponentiation on a smart card. volume 1820, pages 357–365. LNCS, 2000.

[16] W. Burr W. Polk et M. Smid E. Barke, W. Barker. Recommendation for key management. *NIST special publication*, page 57, 2007.

[17] H.S. Lee E. Lee and C. M. Park. Efficient and generalized pairing computation on abelian varieties. *Cryptology ePrint Archive, Report 2008, http ://eprint.iacr.org/*, 2008.

[18] Nadia El-Mrabet. Arithmétique des couplages, performance et résistance aux attaques par canaux cachés. 2009.

[19] Andreas Enge. Bilinear pairings on elliptic curves. 2013.

[20] W. Diffie et M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22 :644–654, 1976.

[21] P.S.L.M. Barreto et M. Naehrig. Pairing-friendly elliptic curves of prime order. pages 319–331. SAC 2005. LNCS 3897, 2006.

[22] N.P. Smart F. Hess and F. Vercauteren. The eta pairing revisited. *IEEE Transactions on Information Theory*, 52 :4595–4602, 2006.

[23] N.P. Smart F. Hess and F. Vercauteren. The eta pairing revisited. volume 52(10). InformationTheory, IEEE Transactions on, 2006.

[24] G. Seroussi F.Blake and N. Smart. Advances in elliptic curve cryptography. *London Mathematical Society Lecture Note Series, Cambridge University Press*, 317, 2005.

[25] Julien FRANCQ. Conception et sécurisation d'unités arithmétiques hautes performances pour courbes elliptiques. 2009.

[26] Santosh Ghosh. Design and analysis of pairing based cryptographic hardware for prime fields. 2011.

[27] Philippe Bulens Guerric Meurice de Dormale and Jean-Jacques Quisquater. An improved montgomery modular inversion targeted for efficient implementa-

tion on fpga,an improved montgomery modular inversion targeted for efficient implementation on fpga.

[28] Arash Hariri and Arash Reyhani-Masoleh. Bit-serial and bit-parallel montgomery multiplication and squaring over gf. *IEEE Transactions on Computers*, 58(10) :1332–1345, 2009.

[29] D. Harris, R. Krishnamurthy, M. Anders, S. Mathew, and S. Hsu. An improved unified scalable radix-2 montgomery multiplier. In *Computer Arithmetic, 2005. ARITH-17 2005. 17th IEEE Symposium on*, pages 172–178, June 2005.

[30] Miaoqing Huang, K. Gaj, and T. El-Ghazawi. New hardware architectures for montgomery modular multiplication algorithm. *Computers, IEEE Transactions on*, 60(7) :923–936, July 2011.

[31] Kuan i Lee. *Algorithm and VLSI architecture design for H.264/AVC Inter Frame Coding*. PhD thesis, National Cheng Kung University, Tainan, Taiwan, 2007.

[32] et N.P. Smart I.F. Blake, G. Seroussi. Advances in elliptic curve cryptography. *London Mathematical Society Lecture Note Series, chapter IX. Cambridge University Press*, 2005.

[33] S. Mitsunari E. Okamoto F. Rodríguez-Henríquez J. Beuchat, J.E. Gonz àlez-Díaz and T. Teruya. High-speed software implementation of the optimal ate pairing over barreto-naehrig curves. *International Conference on Pairing-Based Cryptography (Pairing 2010), LNCS*, 6487 :21–39, 2010.

[34] F. Vercauteren J. Fan and I. Verbauwhede. Faster fp-arithmetic for cryptographic pairings on barreto-naehrig curves. volume 5747, pages 240–253. CHES 2009, LNCS, 2009.

[35] Shigeo Mitsunari Eiji Okamoto Francisco Rodriguez-Henriquez Jean-Luc Beuchat, Jorge E. Gonzalez-Diaz and Tadanori Teruya. High-speed software implementation of the optimal ate pairing over barreto-naehrig curves. 2010.

[36] J.H.Silverman. The arithmetic of elliptic curves. *Texts in Mathematics. Springer-Verlag, New York*, 106, 1992.

[37] Antoine Joux. A one round protocol for tripartite diffie-hellman. volume 1838, pages 385–393. In Wieb Bosma, editor, Algorithmic Number Theory, of Lecture Notes in Computer Science, 2000.

[38] Antoine Joux. A one round protocol for tripartite diffie-hellman. *Journal of Cryptology*, 17(4) :263–276, 2004.

[39] N. Koblitz and A. Menezes. Pairing-based cryptography at high security levels. Cryptology ePrint Archive, Available at http ://eprint.iacr.org/2005/076.pdf., 2005.

[40] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177) :203–209, January 1987.

[41] C.K. Koç, Tolga Acar, and Jr. Kaliski, B.S. Analyzing and comparing montgomery multiplication algorithms. *Micro, IEEE*, 16(3) :26–33, Jun 1996.

[42] Saadat Pourmozafari Kooroush Manochehri and Babak Sadeghiyan. Montgomery and rns for rsa hardware implementation. pages 849–880, December 2010.

[43] H. T. KUNG and C. E. LEISERSON. Systolic arrays for (vlsi), proc. of the symposium on sparse matrices computations. pages 256–282, 1978.

[44] H.T. Kung. Why systolic architectures ? *Computer*, 15(1) :37–46, Jan 1982.

[45] H. Lenstra. Factoring integers with elliptic curves. *Annals of Mathematics*, 126 :649–673, 1987.

[46] R. Niederhagen M. Naehrig and P. Schwabe. New software speed records for cryptographic pairings. Cryptology ePrint Archive, Report /186. http ://eprint.iacr.org/., 2010.

[47] M. Charlemagne-L.J. Dominguez Perez M. Scott, N. Benger and E.J. Kachisa. On the final exponentiation for calculating pairings on ordinary elliptic curves. Cryptology ePrint Archive, Report 2008/490, 2008. Available at http ://eprint.iacr.org/2008/490.pdf, 2008.

[48] Nicolas Méloni. Arithmétique pour la cryptographie basée sur les courbes elliptiques. 2007.

[49] A. Menezes. An introduction to pairing-based cryptography. *Recent Trends in Cryptography, Vol. 477 of Contemporary Mathematics*, 477 :47–65, 2009.

[50] Amine MRABET Nadia EL-MRABET Belgacem BOUALLEGUE Sihem MESNAGER and Mohsen MACHHOUT. An efficient and scalable modular inversion/division for public key cryptosystems. *IEEE International Conference on Engineering and Management Information Systems (ICEMIS 2017)*, 2017.

[51] Amine MRABET Nadia EL-MRABET Ronan LASHERMES Jean-Baptiste RIGAUD Belgacem BOUALLEGUE Sihem MESNAGER and Mohsen MACHHOUT. High-performance elliptic curve cryptography by using the cios method for modular multiplication. *CRISIS 2016 : 11th International Conference on Risks and Security of Internet and Systems*, 2016.

[52] Amine MRABET Nadia EL-MRABET Ronan LASHERMES Jean-Baptiste RIGAUD Belgacem BOUALLEGUE Sihem MESNAGER and Mohsen MACHHOUT. A scalable and systolic architectures of montgomery modular multiplication for public key cryptosystems based on dsps. *The Sixth International Conference on Security, Privacy and Applied Cryptographic Engineering (SPACE 2016)*, 2016.

[53] Manuel Charlemagne Luis J. Dominguez Perez Ezekiel J. Kachisa Michael Scott, Naomi Benger. On the final exponentiation for calculating pairings on ordinary elliptic curves. volume 5671, pages 78–88. Pairing 2009, LNCS, 2009.

[54] Victor S Miller. Use of elliptic curves in cryptography. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 417–426. Springer, 1985.

[55] V.S. Miller. Uses of elliptic curves in cryptography. In *In Advaces in Cryptology-CRYPTO*.

[56] V.S. Miller. Short programs for functions on curves. http ://crypto. stanford.edu/miller, 1983.

[57] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170) :519–521, 1985.

[58] Nadia El Mrabet and Marc Joye. Handbook of pairing based cryptography. *CRC Press*, 2016.

[59] National Institute of Standards and Technology (NIST). Recommendation for key management - part 1 : General (revised). *NIST Special Publication Available online at : http ://csrc.nist.gov/publications/PubsSPs.html*, pages 800–57, 2007.

[60] S.B. Ors, L. Batina, B. Preneel, and J. Vandewalle. Hardware implementation of a montgomery modular multiplier in a systolic array. page 8, April 2003.

[61] J. Grobschadl P. Grabher and D. Page. On software parallel implementation of cryptographic pairings. pages 35–50. SAC 2008. LNCS 5381, 2008.

[62] Guilherme Perin, Daniel Gomes Mesquita, and João Baptista Martins. Montgomery modular multiplication on reconfigurable hardware : Systolic versus multiplexed implementation. *Int. J. Reconfig. Comput.*, 2011 :61–610, January 2011.

[63] B. Lynn P.S.L.M. Barreto and M. Scott. Constructing elliptic curves with prescribed embedding degrees. *Security in Communication Networks, LNCS*, 2576 :257–267, 2003.

[64] B. Lynn P.S.L.M. Barreto, H. Kim and M. Scott. Efficient algorithms for pairingbased cryptosystems. *Crypto 2002, LNCS 2442*, pages 354–368, 2002.

[65] D. Doche G. Frey T. Lange K. Nguyen et F. Vercauteren R. Avanzi, H. Cohen. Handbook of elliptic and hyperelliptic curve cryptography. *CRC Press*, 2005.

[66] K. Ohgishi R. Sakai and M. Kasahara. Cryptosystems based on pairings. *The 2000 Symposium on Cryptography and Information Security*, 2000.

[67] Antoine Joux Razvan Barbulescu, Pierrick Gaudry and Emmanuel Thomé. A quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. https ://eprint.iacr.org/2013/400.pdf, 2013.

[68] G. Reymond and V. Murillo. A hardware pipelined architecture of a scalable montgomery modular multiplier over gf(2m). In *2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pages 1–6, Dec 2013.

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

[70] R.Lidl and H.Niederreiter. Finite fields. *Cambridge University Press, Cambridge,second edition*, 1994.

[71] Adi Shamir et Leonard M. Adleman Ronald L. Rivest. A method for obtaining digital signatures and public-key cryptosystems. In *Communications of the ACM 21.2*, pages 120–126, Février 1978.

[72] RainerA. Rueppel. High-speed implementation methods for rsa scheme. In *Advances in Cryptology- EUROCRYPT 92*, volume 658 of *Lecture Notes in Computer Science*, pages 221–238. Springer Berlin Heidelberg, 1993.

[73] RainerA. Rueppel. Systolic-arrays for modular exponentiation using montgomery method. In *Advances in Cryptology - EUROCRYPT 92*, volume 658 of *Lecture Notes in Computer Science*, pages 477–481. Springer Berlin Heidelberg, 1993.

[74] X. Lin S. Galbraith and M. Scott. International conference on pairing-based cryptography (pairing 2010), lncs. *Advances in Cryptology - Eurocrypt*, 5479 :518–535, 2009.

[75] S. Singh. Histoire des codes secrets. In *JC Lattés*, 1999.

[76] Alexandre F. Tenca and Çetin Kaya Koç. A scalable architecture for montgomery multiplication. In *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999,*

Proceedings, volume 1717 of *Lecture Notes in Computer Science*, pages 94–108. Springer, 1999.

[77] "NSA Suite B Cryptography U.S. National Security Agency (NSA). Speeding up secure web transactions using elliptic curve cryptography. *Fact Sheet NSA Suite B Cryptography.*, 2009.

[78] F. Vercauteren. Optimal pairings. volume 56(1), pages 455–461. Information-Theory, IEEE Transactions on, 2010.

[79] E. Verheul. Self-blindable credential certificates from the weil pairing. *Advances in Cryptology - Asiacrypt 2001, LNCS*, 2248 :533–551, 2002.

[80] Mahendra Vucha and Arvind Rajawat. Design and fpga implementation of systolic array architecture for matrix multiplication. *International Journal of Computer Applications*, 26(3) :0975–8887, july 2011.

[81] HughC. Williams. Use of elliptic curves in cryptography. In *Advances in Cryptology-CRYPTO 85 Proceedings*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer Berlin Heidelberg, 1986.

1 Appendix

1.1 Code Sage NW-8

```
#NW-8 Algoritm
s=8
p'
p=[p0,p1,p2,p3,p4,p5,p6,p7]
b=[b0,b1,b2,b3,b4,b5,b6,b7]
a=[a0,a1,a2,a3,a4,a5,a6,a7]
T=[0,0,0,0,0,0,0,0,0,0]
for i in range (s):
C_S=0
for j in range (0,s):
C_S=T[j]+a[i]*b[j]+(C_S>>32)
T[j]=C_S%(2^32)
C_S=T[s]+(C_S>>32)
T[s]=C_S%(2^32)
T[s+1]=C_S>>32
m=(T[0]*p')%(2^32)
C_S=T[0]+m*p0
for j in range (1,s):
C_S=T[j]+m*p[j]+(C_S>>32)
T[j-1]=C_S%(2^32)
C_S=T[s]+(C_S>>32)
T[s-1]=C_S%(2^32)
T[s]=T[s+1]+(C_S>>32)
```

1.2 Code Sage NW-16

```
#NW-16 Algoritm
s=16
p'
p=[p0,p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11,p12,p13,p14,p15]
b=[b0,b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12,b13,b14,b15]
a=[a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15]
T=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
```

```
for i in range (s):
C_S=0
for j in range (0,s):
C_S=T[j]+a[i]*b[j]+(C_S>>16)
T[j]=C_S%(2^16)
C_S=T[s]+(C_S>>16)
T[s]=C_S%(2^16)
T[s+1]=C_S>>16
m=(T[0]*p')%(2^16)
C_S=T[0]+m*p0
for j in range (1,s):
C_S=T[j]+m*p[j]+(C_S>>16)
T[j-1]=C_S%(2^16)
C_S=T[s]+(C_S>>16)
T[s-1]=C_S%(2^32)
T[s]=T[s+1]+(C_S>>16)
```

1.3 Inversion/Division Modulaire

```
A=569983903836785675256830915609662112382515821811106401606598594307888906867 1
AA=192883315053954786886336857648647835083689863887288926196632452539433032298 9
p=160305690344031282777566882874986495156368381011843374997783929801162222469 13
u=A;v=p;x1=AA;x2=0
i=0
while u<>1 and v<>1:
if u%2==0:
u=u/2
if x1%2==0:
x1=x1/2
else:
x1=(x1+p)/2
elif v%2==1:
if u>v:
u=u+v
x1=x1+x2

if v%2==0:
```

```
v=v/2
if x2%2==0:
x2=x2/2
else:
x2=(x2+p)/2
elif u%2==1:
if u<v:
v=v+u
x2=x2+x1
if u==1:
z=x1
else:
z=x2
```

1.4 Algorithme de Miller

```
def miller_BN(P,Q):
#T=Q
xp = P[0]
yp = P[1]
zp = P[2]
xq = Q[0]
yq = Q[1]
zq = Q[2]
xt=xq
yt=yq
zt=zq
yq_= -yq
f=1
#s=6*t+2
s=2^64+2^63-2^56-2^55+2^46+2^45+2
L=[1,0,0,0,0,0,0,-1,-1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0]

for i in L:
R1 = doublement_evaluation(xt,yt,zt,xp,yp)
f = f*f*R1[3]
```

```
xt = R1[0]
yt = R1[1]
zt = R1[2]
if (i==-1):
R2 = addition_evaluation(xt,yt,zt,xq,yq_,zq,xp,yp)
f = f*R2[3]
xt = R2[0]
yt = R2[1]
zt = R2[2]
elif (i==1):
Res3 = addition_evaluation(xt,yt,zt,xq,yq,zq,xp,yp)
f = f*R3[3]
xt = R3[0]
yt = R3[1]
zt = R3[2]

xq1,yq1,zq1=frob_point(xq,yq,zq)
xq2,yq2,zq2=frob_point(xq1,yq1,zq1)
R4 = addition_evaluation(xt,yt,zt,xq1,yq1,zq1,xp,yp)
f = f*R4[3]
xt = R4[0]
yt = R4[1]
zt = R4[2]
yq_2=-yq2

R5 = addition_evaluation(xt,yt,zt,xq2,yq_2,zq2,xp,yp)
f = f*R5[3]
xt = R5[0]
yt = R5[1]
zt = R5[2]
return f
```

2 architecture

2.1 Execution

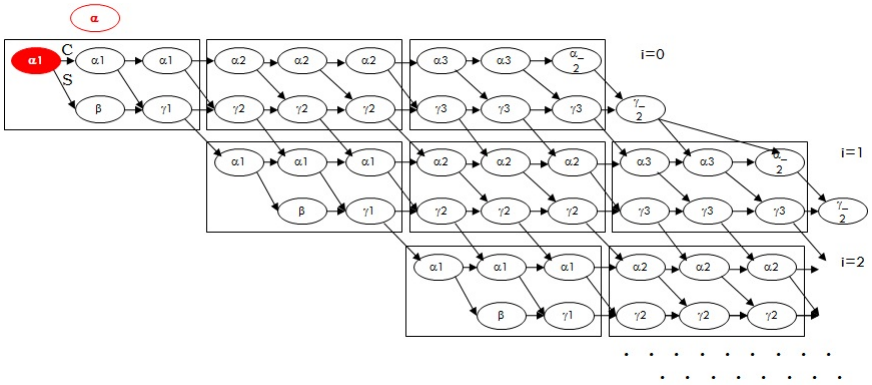

FIGURE 1 – Étape 1.

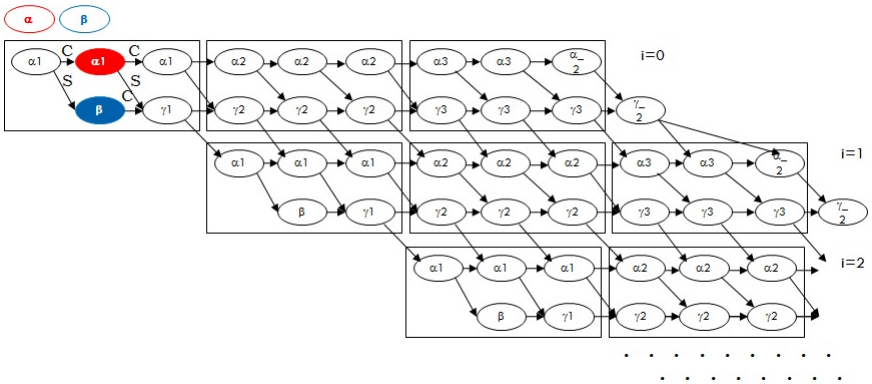

FIGURE 2 – Étape 2.

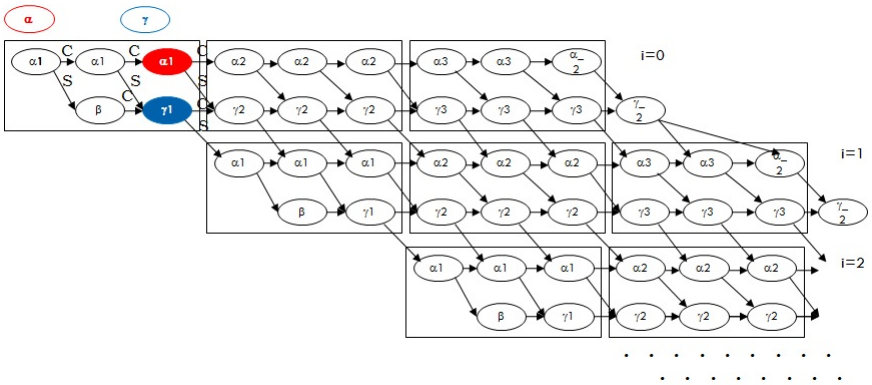

FIGURE 3 – Étape 3.

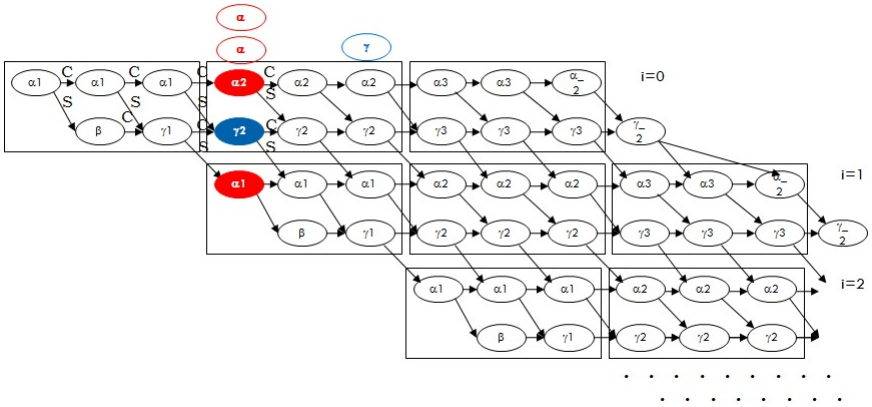

FIGURE 4 – Étape 4.

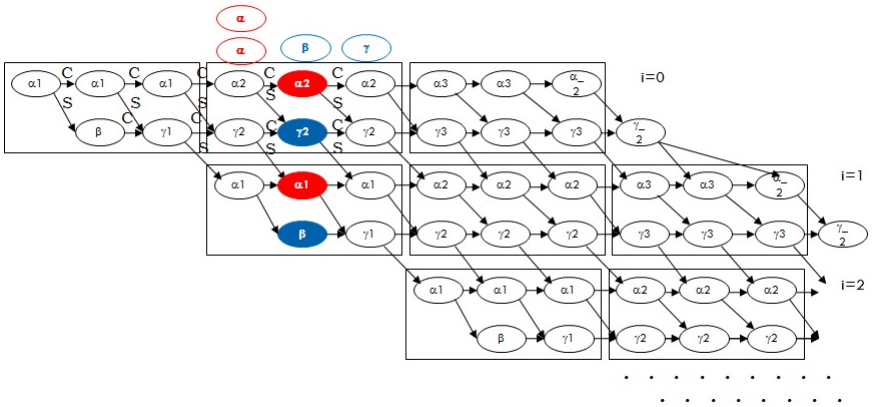

FIGURE 5 – Étape 5.

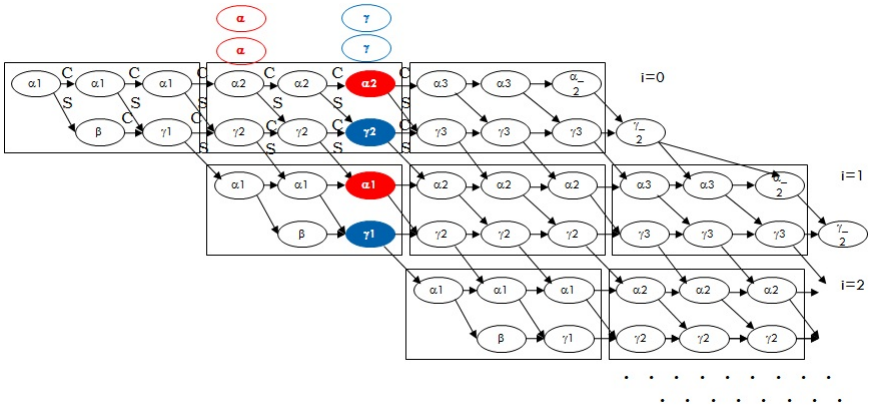

FIGURE 6 – Étape 6.

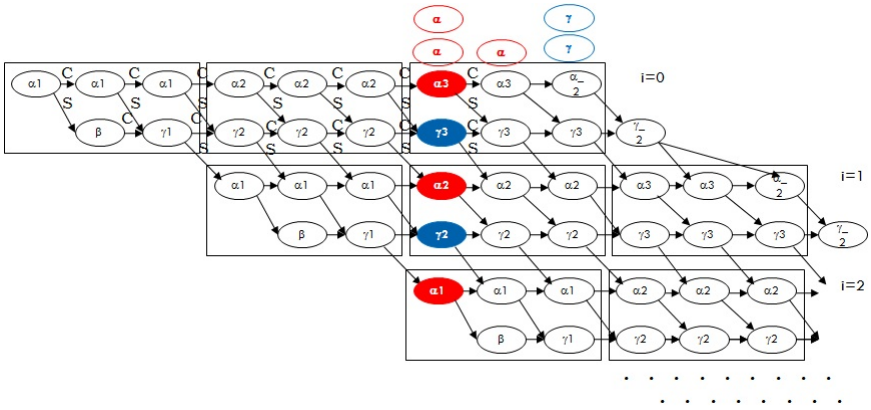

FIGURE 7 – Étape 7.

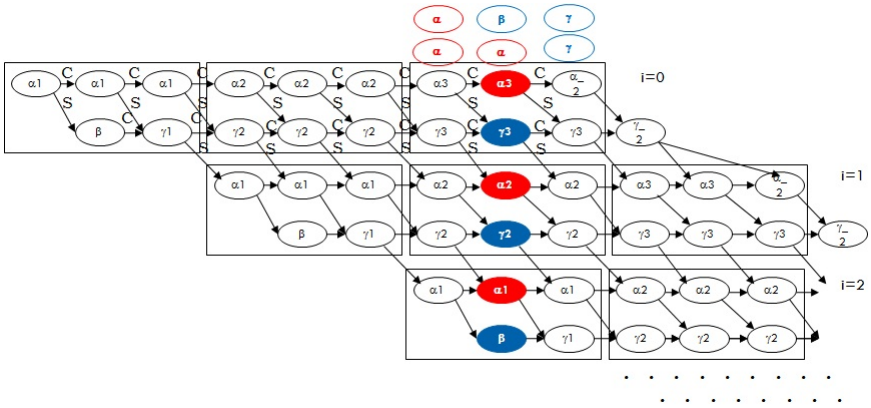

FIGURE 8 – Étape 8.

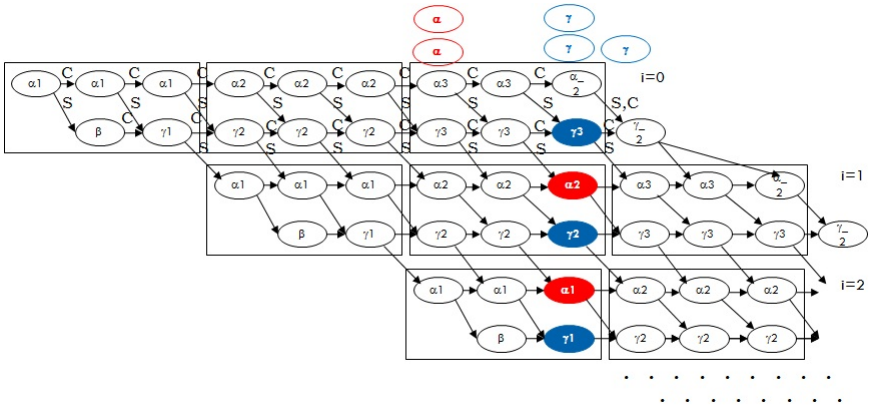

FIGURE 9 – Étape 9.