MA-004-318-1

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université SAAD DAHLAB Blida
Faculté des Sciences
Département d'Informatique

Mémoire de Fin d'Etudes

Présenté en vue de l'obtention du

Diplôme de Master en Informatique

Spécialité : Génie des Systèmes Informatique

Thème

Implémentation d'un crypto-système ECC sur circuit FPGA

Présenté par :	Encadré par :	Promoteur :
BENKESSIRAT Selma	Dr ISSAD Mohamed	Pr BENBLIDIA Nadjia
BENKESSIRAT Amina	Mr BELLEMOU Mohamed	

président de jury : Mr Benyahia
Examinateur : Mr Nahal

Année universitaire : 2015/2016

MA-004-318-1

"Si tu veux construire un bateau, ne rassemble pas tes hommes et femmes pour leur donner des ordres, pour expliquer chaque détail, pour leur dire où trouver chaque chose... Si tu veux construire un bateau, fais naître dans le cœur de tes hommes et femmes le désir de la mer."

...Antoine de Saint-Exupéry

Resume :

Le but de ce projet est l'implementation du crypto systeme ECC (Elliptic Curve Cryptography) embarque sur circuit FPGA de Xilinx à base du processeur Microblase. La multiplication scalaire est l'opération cœur du chiffrement et du déchiffrement basés sur les courbes elliptiques. Cette dernière est basee sur la multiplication modulaire et l'inverse modulaire. Au cours de notre projet, nous avons choisi l'algorithme de la multiplication modulaire de Montgomery et l'algorithme de Montgomery Power Lader respectivement pour l'implémentation de la multiplication modulaire et l'inverse modulaire. Dans le but d'atteindre un meilleur compromis entre le temps d'exécution et les ressources matérielles requises, nous avons proposé deux approches d'implémentation. Une approche purement logicielle ou l'exécution des opérations du chiffrement et du déchiffrement est assurés en entier par le processeur Microblaze. Une approche basée sur une combinaison logicielle/matérielle. Dans cette implémentation l'exécution des opérations de la multiplication modulaire et l'inverse modulaire est assuré par une IP(intellectual property) matérielle. Les résultats d'implémentation ont montré que : la seconde approche d'implémentation présente le meilleur compromis temps d'exécution/ressources matérielle requises. Les délais du chiffrement et du dechiffrement sont respectivement 480.29977 *ms* et 240.40081 *ms* le nombre de slices occupés est de 2245 *slices*.

Abstract:

The objective of this work is the implementation of crypto system ECC (Elliptic Curve Cryptography) embedded onto Xilinx's FPGA circuit which is based on Microblaze processor. In ECC scalar multiplication is the main operation of the encryption and the decryption. It is based on modular multiplications and modular inverses. In our project, we have used the Montgomery modular multiplication and Montgomery Power Lader algorithms respectively to implement modular multiplication and modular inverse. In order to achieve a better compromise between the execution time and the requirements hardware, two implementations approaches were proposed. The first approach is a pure software implementation where the execution of the encryption and decryption operations are fully provided by the MicroBlaze processor. The seconde is based on a hardware / software co-design. In this implementation the execution of operations of modular multiplication and modular inverse is cariedout by a hardware IP (intellectual property). The implementation results showed that the second implementation approach presents the best compromise execution time / hardware resources. The execution time of the encryption and decryption are respectively 480.29977 ms and 240.40081 ms. The number of occupied slices is 2245 slices.

ملخص

الهدف من هذا المشروع هو تنفيذ نظام التشفير ECC يتمثل على المعالج Microblaze ، محمل على دارة منطقية البرمجة (FPGA) ل Xilinx . الضرب القياسي هي العملية الأساسية للتشفير و فك التشفير في نظام ECC هذه الأخيرة ترتكز على عمليتي الضرب الترديدي و المقلوب الترديدي. في مشروعنا، اخترنا خوارزمية الضرب الترديدي لمونغومري و خوارزمية Montgomery Power Leader لتنفيذ الضرب الترديدي و المقلوب الترديدي على الترتيب لهدف التوفيق بين وقت التنفيذ و الموارد المادية المستعملة، اقترحنا منهجيتين لزرع النظام ECC ، النهج الأول برمجية Software محض حيث يتم تنفيذ عمليتي التشفير و فك التشفير بالكامل من قبل المعالج Microblase. أما الثاني، فيجمع بين ال Hardware و Software ، حيث يتم تنفيذ عمليتي الضرب الترديدي و المقلوب الترديدي من قبل ل IP Materielle أظهرت نتائج التنفيذ أن النهج الثاني يقدم التوفيق الأمثل بين وقت التنفيذ و الموارد المادية المستعملة. في الواقع وقت تنفيذ التشفير و فك التشفير يقدر ب 480.29977 *ms* و 240.40081 *ms* على الترتيب. المساحة المشغولة بداخل الدارة FPGA قدرت ب 2245 slices

Remerciements

Au nom d'Allah, le Tout Miséricordieux, le Très Miséricordieux
Paix et Salut sur le Prophète.

La réalisation de ce travail a été possible grâce à **Allah** le tout puissant et grâce à la contribution de plusieurs personnes à qui nous voudrions témoigner toute notre reconnaissance.

Au terme de ce travail, nous tenons à remercier **Allah** *le tout puissant de nous avoir donné le courage, la volonté et la patience pour achever ce travail.*

Nous voudrions adresser toute notre gratitude au directeur de ce mémoire, Mr.Mohamed ISSAD, pour sa patience, sa disponibilité et surtout ses judicieux conseils, qui nous ont grandement aidés.

Nous tenons à remercier spécialement Mr.Mohamed Bellemou pour son aide précieuse, ses conseils qui ont contribué à alimenter notre réflexion, sa disponibilité et surtout son aide morale.

Nous désirons remercier toute l'équipe du Centre de Développement des Technologies Avancées (CDTA).

Nous tenons aussi à remercier Mme Benblidia pour son aide précieuse et ses conseils.

Nous désirons aussi remercier nos enseignants, qui nous ont fourni les outils nécessaires à la réussite de nos études universitaires.

Nous tenons à témoigner toute notre gratitude à nos chers parents et frères pour leur confiance et leur support inestimable.

Nous tenons à remercier les membres du jury qui nous font l'honneur d'examiner notre travail.

Nous voudrions exprimer notre reconnaissance envers nos cousins, nos amis et nos collègues qui nous ont apporté leur support moral et intellectuel tout au long de notre démarche.

Nous tenons à remercier spécialement notre chère amie et sœur Nesrine pour son soutien moral, ses encouragements et sa disponibilité à notre égard.

Nous tenons à remercier chaleureusement notre cousine et triplet Meriem pour son inquiétude, sa capacité d'écoute et son aide morale.

Merci à tous....

Tables des matières

Introduction générale..... 1

Chapitre 1 : Généralités sur la cryptographie

1.1. Introduction..... 4

1.2. La cryptographie..... 4

1.2.1. Cryptographie symétrique(chiffrement à clé privée)..... 4

1.2.2. Cryptographie asymétrique(chiffrement à clé public)..... 5

1.3. Protocole de cryptographie à base de courbes elliptiques..... 6

1.3.1 Définitions..... 6

1.3.2 Crypto système à base de courbe elliptique ECC..... 8

1.3.3 Opération de chiffrement et de déchiffrement..... 9

1.3.4 Les règles d'addition et dédoublement de points..... 10

1.3.5 Génération des clés publiques et privées..... 10

1.3.6 Multiplication scalaire..... 11

1.4. Conclusion..... 12

Chapitre 2 : Algorithmes de calcul des opérations de base de la multiplication scalaire

2.1 Introduction..... 13

2.2 Arithmétique modulaire..... 13

2.2.1 Addition modulaire..... 13

2.2.2 Soustraction modulaire..... 14

2.2.3 Multiplication modulaire..... 14

2.2.4 Multiplication modulaire de Montgomery (MMM)..... 15

2.2.5 Inverse modulaire..... 18

2.2.6 Exponentiation modulaire..... 18

2.3 Adaptation des opérations arithmétiques de base au processeur Microblaze 19

2.4 Multiplication scalaire de Montgomery 20

2.5 Conclusion..... 23

Chapitre 3 :Systèmes embarqués

3.1 Introduction..... 24
3.2 Circuits intégrés..... 24
3.3 Système embarqué sur puce..... 24
3.4 Plateforme d'implémentation de systèmes embarqués..... 25
3.5 Approche de conception logicielle/matérielle (co-design)..... 26
3.6 Circuits FPGAs..... 27
3.7 Circuits FPGAs de la famille Virtex-5..... 29
3.8 Système embarqué sur FPGA..... 29
3.9 Processeur embarqué sur FPGA..... 29
3.10 Processeur Microblaze..... 30
3.11 Système embarqué à base du processeur Microblaze..... 31
3.12 Coclusion..... 33

Chapitre 4 :Implémentation de la plateforme chiffrement/déchiffrement

4.1 Introduction..... 34
4.2 Description de la plateforme de chiffrement/déchiffrement..... 34
4.3 Architecture du crypto système embarqué..... 36
4.4 Implémentation purement logicielle..... 37
4.4.1 Exécution d'un chiffrement/déchiffrement..... 38
4.4.2 Fonction de la multiplication scalaire $mul\_scal(.)$..... 43
4.4.3 Fonctions d'exécutions des opérations arithmétiques..... 46
4.5 Implémentation basée sur la combinaison logicielle/matérielle..... 49
4.6 Présentation de l'Interface Homme Machine IHM..... 52
4.6.1 Configuration de protocole de communication RS232..... 53
4.6.2 Génération des clés du chiffrement et du déchiffrement..... 54
4.6.3 Conversion du message clair en un point..... 56
4.6.4 Transmission vers la carte FPGA des données necessaires pour le chiffrement/déchiffrement..... 56
4.6.5 Conversion du point résultant du déchiffrement en un message clair.. 58
4.7 Conclusion..... 59

Chapitre 5 : Résultat d'implémentation

5.1 Introduction..... 60
5.2 Méthodologie de conception..... 60
5.3 Performances temporelles et ressources matérielles occupées..... 61
5.3.1 Performances temporelles..... 61
5.3.2 Ressources matérielles requises..... 61
5.4 Conclusion..... 62
Conclusion Générale..... 63
Annexe
Bibliographie

Figure 4.16 Code C de la fonction mongomery(.) 48
Figure 4.17 Code C de la fonction add_mod(.) 48
Figure 4.18 Code C de la fonction sub_mod(.) 49
Figure 4.19 Architecture du composant IP_EXP_MMM 50
Figure 4.20 Page d'accueil de l'IHM 53
Figure 4.21 Configuration du port RS232 54
Figure 4.22 Génération des clés 55
Figure 4.23 Code java de la génération des clés 55
Figure 4.24 Code java de conversion d'un message en point 56
Figure 4.25 Code java de la transmission des données de l'ordinateur vers la carte FPGA 57
Figure 4.26 Code java de la réception des données de la carte FPGA vers l'ordinateur 57
Figure 4.27 Code java de la conversion d'un point en message 58
Figure 4.28 Chiffrement d'un message 58
Figure 4.29 Déchiffrement d'un message 59

Liste des tableaux

Tableau 2.1	Complexité algorithmique de l'algorithme 2.7.....	22
Tableau 4.1	Format des instructions du composant IP_EXP_MMM...	51
Tableau 4.2	Les foncions de communication avec le composant IP_EXP_MMM.....	51
Tableau 5.1	Performances temporelles en fonction de l'approche utilisée.....	61
Tableau 5.2	Ressources matérielles requises pour l'implémentation des approches proposées.....	62

Liste des algorithmes

Algorithme 1.1	Algorithme de génération de la paire de clé.....	11
Algorithme 1.2	Algorithme de la multiplication scalaire.....	12
Algorithme 2.1	Algorithme de l'addition modulaire.....	13
Algorithme 2.2	Algorithme de la soustraction modulaire.....	14
Algorithme 2.3	Algorithme de la multiplication modulaire.....	14
Algorithme 2.4	Algorithme de la multiplication de Montgomery.....	15
Algorithme 2.5	Algorithme de la MMME.....	17
Algorithme 2.6	Algorithme de Montgomery Power Ladder.....	18
Algorithme 2.7	Algorithme de la multiplication scalaire de Montgomery.....	21

Introduction générale

Introduction générale

Au cours de ces dernières années, l'informatique est devenue le cœur de l'exploitation de la technologie numérique. L'évolution de celle-ci offre de nouvelles possibilités de communication. Nous sommes entourés de données numériques qui transitent via des réseaux, par des personnes ou des objets. Une diversité de services est à la disposition des individus étant connectés ; citons entre autres la téléphonie mobile, le paiement et le e-commerce ...etc. La multitude de ces services offre un confort à la société, mais les risques en termes de sécurité engendrés par les transactions numériques ne sont pas négligeables. Les données qui transitent entre les individus sont sujettes aux attaques, qui compromettent leurs sécurités et qui causent l'interruption, l'interception et la modification des données. Pour cela, plusieurs services de sécurité ont été mis en œuvre, pour assurer la confidentialité l'intégrité et la disponibilité des données. Nous nous intéressons à la confidentialité des données qui est assurée par la cryptographie. La cryptographie est composée de la cryptographie symétrique et de la cryptographie asymétrique. Dans la cryptographie symétrique, la clé utilisée pour chiffrer une donnée est la même clé que celle utilisée pour déchiffrer cette dernière. Elle est dite clé secrète. Dans la cryptographie asymétrique, la clé utilisée pour chiffrer une donnée est différente de celle utilisée pour la déchiffrer. La clé du chiffrement est dite clé public, la clé du déchiffrement est dite clé privée.

Dans ce travail, nous nous intéressons à la cryptographie asymétrique. De nos jours, les crypto système asymétriques les plus utilisés sont le RSA (Rivest-Shamir-Adleman) et le ECC (Elliptic Curve Cryptography)[1]. Ces crypto systèmes se basent sur des problèmes mathématiques difficiles à résoudre. Leur sécurité est assurée par l'utilisation des clés de grande tailles pouvant aller jusqu'à des milliers de bits.

Dans le crypto système RSA les opérations de chiffrement et de déchiffrement sont basées sur le calcul de l'exponentiation modulaire, noté $Y = X^z \ mod \ N$, ou X est la donnée à chiffrer ou à déchiffrer, Z est la clé public ou la clé privée, N est le modulo [2].

Dans le crypto système ECC, les opérations de chiffrement et de déchiffrement sont basées sur le calcul de la multiplication scalaire, notée $Q = k \times P$, ou k est un scalaire et P est un point. Les opérations de base de cette opération sont l'addition

modulaire, la soustraction modulaire, la multiplication modulaire et l'inverse modulaire [3].

La conception des crypto systèmes sur une même puce consiste en l'implémentation matérielle de la fonction de base, ainsi que la gestion software par un processeur embarqué. L'implémentation de ces système peut être sur des circuits spécifiques de type ASIC (Application Specific Integrated Circuit), ou sur des circuits programmables de type FPGA (Filed Programmable Gate Array). En effet, l'implémentation de ces systèmes se fait par la combinaison des deux parties matérielle/logicielle sur une même puce. La partie logicielle rend le système plus flexible. La partie matérielle est exploitée pour les performances temporelles [4],[5].

L'objectif de notre travail est l'étude de la nature et la complexité calculatoire d'un crypto système ECC. On s'intéresse plus particulièrement à ses opérations arithmétiques de base et à son implémentation sur circuit FPGA. La plateforme ciblée est du type PSoC (Programmabe System on Chip) à base du processeur embarqué Microblaze de Xilinx.

Afin d'atteindre le meilleur compromis entre la vitesse d'exécution, les ressources matérielles occupées et la flexibilité du crypto système, deux approches d'implémentation ont été proposées. La première approche est purement logicielle, ou tous les algorithmes des opérations de base de chiffrement et de déchiffrement sont exécutées par le processeur Microblaze [6].

La deuxième approche est une combinaison logicielle/matérielle. Cette dernière consiste à intégrer une IP (Intellectuel Property) autour du processeur. Celle-ci est dédiée à l'exécution de la multiplication modulaire et l'inverse modulaire. Le contrôle des algorithmes de la multiplication scalaire, le chiffrement et le déchiffrement est assuré par le processeur Microblaze.

Pour réaliser ce projet de fin d'étude, dont le thème est : « *Implémentation d'un crypto système ECC sur circuit FPGA* », proposé par l'équipe AC2 de la division Architecture des Systèmes et Multimédia (ASM), au Centre de Développement des Technologie Avancées (CDTA), notre mémoire est organisé comme suit :

- Chapitre 1 est consacré aux généralités sur la cryptographie à base de courbes elliptiques.

- Chapitre 2 est dédié à l'étude des besoins en arithmétique modulaire par le crypto-système ECC et leurs algorithmes de calcul.
- Chapitre 3 est réservé à l'étude des systèmes sur puce, plus particulièrement les systèmes implémentés sur circuit FPGA.
- Chapitre 4 présente la méthodologie développée pour la réalisation du crypto-système ECC.
- Chapitre 5 est consacré à la présentation des résultats d'implémentation et à l'étude des performances de notre crypto-système.

Nous terminerons par une conclusion générale qui inclut nos perspectives.

Chapitre 1

Généralités sur la cryptographie à base de courbe elliptiques

1.1 Introduction

Dans le domaine des réseaux informatiques, la sécurité des données confidentielles est une contrainte majeure pour assurer leurs confidentialités. De ce fait, la cryptographie est considérée comme une solution dominante pour atteindre cet objectif. Cette dernière permet de rendre une donnée illisible en se basant sur un ensemble de méthodes mathématiques, qui sont conçues pour réaliser une telle transformation en leur fournissant une clé de cryptage. De nos jours plusieurs protocoles de cryptographie existent. On peut citer entre autres le RSA, l'ECC et l'AES etc [1],[2],[3]. Dans ce projet, nous nous intéressons à une catégorie bien spécifique de cryptographie qui est basée sur les courbes elliptiques. Dans ce chapitre nous allons présenter les concepts généraux de la cryptographie. Ensuite nous décrivons les notions de base qui permettent de manipuler les courbes elliptiques pour le chiffrement et le déchiffrement des données.

1.2 La cryptographie

La cryptographie est un ensemble de techniques permettant de transmettre une donnée chiffrée inintelligible par celui qui n'a pas le droit de la lire [1],[7]. Pour chiffrer une donnée, trois paramètres sont nécessaires:

- Fonction de chiffrement et de déchiffrement.
- Donnée à chiffrer ou à déchiffrer.
- Clé de chiffrement et de déchiffrement.

La figure 1.1 schématise le concept de base du chiffrement et de déchiffrement.

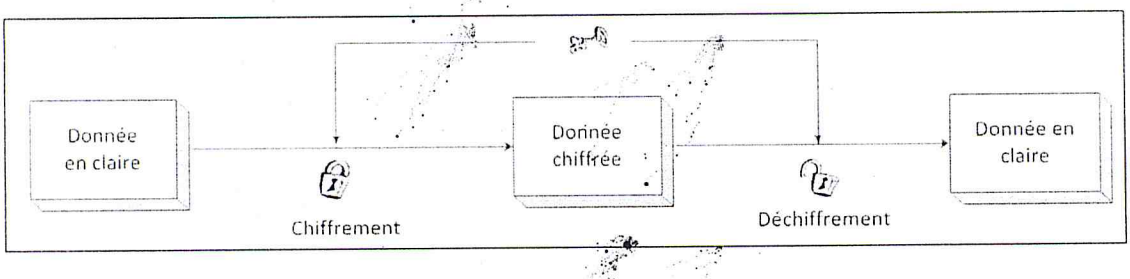

Figure 1.1 : Principe de base du chiffrement et du déchiffrement.

D'une manière générale, nous distinguons deux catégories de protocoles de cryptographie, à savoir symétrique et asymétrique [1].

1.2.1 Cryptographie symétrique (chiffrement à clé privée)

Ce type de cryptographie utilise une seule clé pour le chiffrement et le déchiffrement de la donnée. Il est considéré comme une solution moins couteuse et

facile à implémenter. La clé de chiffrement et moins longue que la clé du chiffrement asymétrique, [1], [8].

On suppose que M est la donnée à chiffrer et k est la clé privée partagée par l'expéditeur et le récepteur de la donnée. L'expéditeur utilise la fonction de chiffrement $E()$ et la clé k pour générer une donnée M' chiffrée.

Après la réception de la donnée chiffrée, le récepteur utilise une fonction de déchiffrement $D()$ et la clé k pour déchiffrer la donnée. Les opérations du chiffrement et du déchiffrement sont représentées respectivement par les équations (1.1) et (1.2).

$$E(M,k) = M' \tag{1.1}$$

$$D(M',k) = M \tag{1.2}$$

Cette méthode exige la distribution de la clé entre les deux communicants. Ce qui cause un problème [1], [8]. Puisque le protocole exige un canal sécurisé pour partager la clé.La figure 1.2 schématise un chiffrement/déchiffrement.

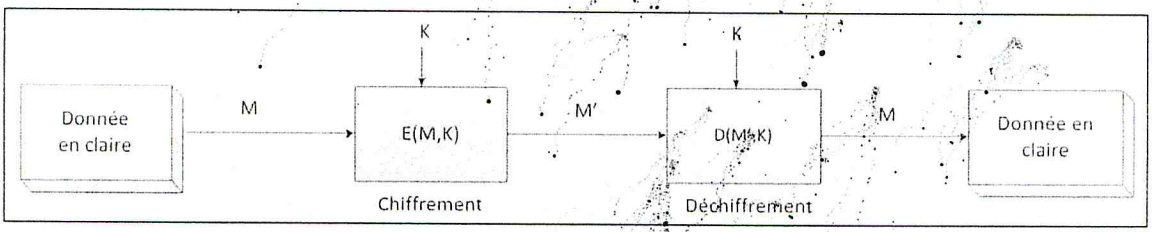

Figure 1.2 : Schéma synoptique du protocole de cryptographie symétrique.

I.2.2 Cryptographie asymétrique (chiffrement à clé public)

Le problème de distributions des clés est résolu par la cryptographie asymétrique. Ce concept a été développé par Wlifield Diffie et Martin Hellman[9]. Ce type de cryptographie utilise deux clés distinctes. Une pour le chiffrement de la donnée. L'autre pour le déchiffrement. Avant l'envoi de la donnée M, celle-ci est chiffrée avec la clé public du destinataire kp. Le résultat est :

$$E(M,kp) = M' \tag{1.3}$$

$E()$ représente la fonction de chiffrement. M' est la donnée chiffrée. Après la réception de M', le destinataire utilise sa clé privée ks, qui est gardée secrète, pour déchiffrer la donnée :

$$M = D(M',ks) \tag{1.4}$$

$D()$ est la fonction de déchiffrement. Ce type de cryptographie offre la possibilité d'envoyer des messages signés [1],[10]. La figure 1.3 schématise le protocole de cryptographie asymétrique.

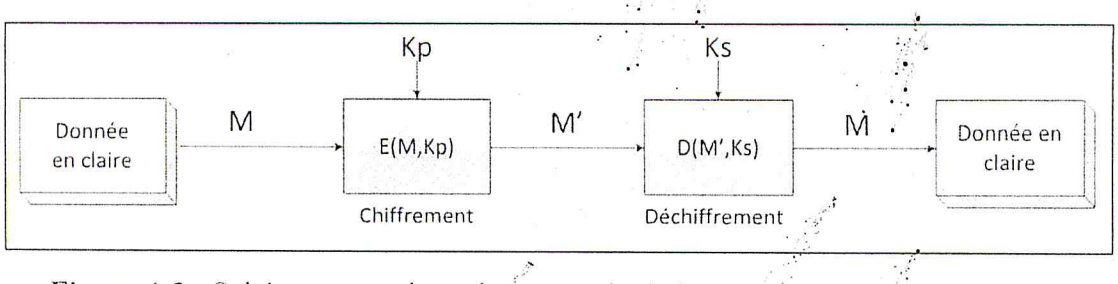

Figure 1.3 : Schéma synoptique du protocole de la cryptographie asymétrique.

Les crypto systèmes à clé public les plus connus de nos jours sont :

- RSA (Rivest-Shamir-Adleman).
- ECC (Elliptic Curve Cryptography).

Dans ce travail, nous nous intéressons au second crypto-système. Pour cela nous allons tout d'abord introduire les courbes elliptiques et toutes les notions de base qui nous permettent de maitriser ce type de crypto système.

1.3 Protocole de cryptographie à base de courbe elliptique

Avant de présenter le protocole de cryptographie basé sur les courbes elliptiques, nous étudions en premier lieu les concepts mathématiques dont nous avons besoin pour comprendre son fonctionnement.

1.3.1 Définitions

Nous commençons par donner quelques définitions nécessaires à la réalisation de ce travail.

a. Groupe

En mathématique, un groupe est un couple $(G,\bullet)$ ou G est un ensemble et $\bullet$ est une loi de composition interne [11]. Le couple $(G,\bullet)$ est un groupe si les conditions suivantes sont satisfaites :

- Fermeture : $\forall(a,b) \in G \mid a \bullet b \in G$
- Associativité : $\forall(a,b) \in G \mid (a \bullet b) \bullet c = a \bullet (b \bullet c)$
- Elément neutre : $\exists e \in G \mid a \bullet e = e \bullet a = a$
- Symétrique : $\forall a \in G, \exists b \in G \mid a \bullet b = b \bullet a = e$

b est considéré comme étant l'élément symétrique de a dans G.

b. Groupe abélien

Un groupe $(G,\bullet)$ est dit abélien (ou commutatif) si [11] :

- $\forall(a,b) \in G \; a \bullet b = b \bullet a$

c. Groupe cyclique

Un groupe fini G est cyclique si tout élément du groupe peut s'exprime sous forme d'une puissance ou d'un multiple d'un élément particulier g, appelé le générateur du groupe, c'est-à-dire $\{G = < g > = g^n \mid n \in Z\}$. Par exemple si : $G = \{g^0, g^1, g^2, g^3, g^4, g^5, g^6\}$ et $g^6 = g^0$, alors G est un groupe cyclique. Tout groupe cyclique est abélien car $g^n g^m = g^{n+m} = g^{m+n} = g^m g^n$.

L'ordre d'un élément e d'un groupe cyclique est le nombre entier n positif tel que $n * e = 0$ (en notation additive) ou $e^n = 1$ (en notation multiplicative). Reprenons le même groupe G du paragraphe précédent, par exemple l'ordre de l'élément g^2 est 3 car l'élément neutre du groupe est $g^0 = 1$ et $(g^2)^3 = g^6$ [11].

d. Anneau unitaire et anneau commutatif

Un anneau unitaire, ou simplement anneau, est un ensemble E muni de deux lois de composition, notées + (addition) et • (multiplication). [1],[11] E est un anneau, si :

- $(E, +)$ est un groupe commutatif.
- La loi • est associative et distributive par rapport à la loi +.
- La loi • possède un élément neutre.

Il existe un élément neutre de la loi de composition + noté 0 tel que :

$$\forall a \in E \mid a + 0 = 0 + a = a.$$

Et

$$\forall a \in E, \exists b \in E \mid a + b = 0.$$

Il existe un autre élément neutre pour la loi de composition •, noté 1 tel que :

$$\forall a \in E \mid a \bullet 1 = 1 \bullet a = a.$$

Un anneau commutatif est un anneau dont la loi de composition • est commutative, c'es à dire :

$$\forall (a, b) \in E \mid a \bullet b = b \bullet a.$$

e. Corps fini

Un corps fini F est un corps dont le nombre d'éléments est fini. Le nombre d'éléments est l'ordre du corps, noté q, qui peut être représenté par la puissance d'un nombre premier $q = p^n$. p est un nombre premier, appelé la caractéristique du corps. $n \in Z$ [11].

f. Corps premier

Un corps est un corps premier, noté Fp lorsque l'ordre du corps $q = p$ et p est un nombre premier. Le corps est constitué des nombres entiers {0, 1, 2, ..., p − 1} et$\forall a \in Z, a \bmod p$ donne le reste unique r qui est compris entre [0, p − 1] [11] .

g. Corps binaire

Un corps fini de l'ordre 2^n est un corps binaire, noté F_2^n , qui peut être construit en utilisant une représentation polynomiale. Les éléments du corps sont des polynômes binaires dont les coefficients $a_i \in \{0,1\}$ et les degrés sont inférieurs à n. C'est-à-dire $F_2^n = a_{n-1}z_{n-1} + a_{n-2}z_{n-2} + \ldots + a_1z_1 + a_0$. $a_i \in \{0,1\}$ et $Z_i = 2^i$ [11] .

1.3.2 Crypto système à base de courbes elliptiques ECC

Après la définition des notions mathématiques nécessaire, nous allons passer dans ce qui suit à la présentation du protocole de cryptographie basé sur les courbes elliptiques.

Une courbe elliptique E est une courbe algébrique représentée par l'équation suivante [10],[12] :

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

Où a_i et e sont des réels [10]. Son discriminent $\Delta \neq 0$. Ce dernier est calculé par les équations suivantes :

- $\Delta = -{d_2}^2 d_8 - 8{d_4}^3 - 27{d_6}^2 + 9d_2d_4d_6$
- $d_2 = {a_1}^2 + 4a_2$
- $d_4 = 2a_4 + a_1a_3$
- $d_6 = {a_3}^2 + 4a_6$
- $d_8 = {a_1}^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2{a_3}^2 - {a_4}^2$

Dans notre travail, nous nous intéressons au cas si la courbe elliptique E est définie dans un champ Gp, ou p est un nombre premier. L'équation de E est définie par :

$$y^2 = x^3 + ax + b \bmod N$$

Son discriminent est de la forme :

$$\Delta = -16(4a^3 + b^2) \bmod N \text{ et } \Delta \neq 0$$

Les paramètres d'une courbe elliptique sont G, p, a, n tel que [10], [12] :

- N : la taille du corps premier fini.
- a, b : les coefficients de l'équation de la courbe E.
- G : point générateur de la courbe.
- n : ordre du point générateur.

1.3.3 Opération de chiffrement et de déchiffrement

La cryptographie à base de courbes elliptique peut être utilisée pour chiffrer un message lisible M, en un message M'. M est transformé en un ensemble de points P_m à partir de l'ensemble de points fini de la courbe elliptique $Ep(a,b)$. La première étape consiste à choisir un point générateur, $G \in Ep(a,b)$, tel que la plus petite valeur de n pour laquelle $nG = O$ est un très grand nombre premier. L'ensemble des points $Ep(a,b)$ et G sont rendus public.

Chaque destinataire génère sa clé privée $nb < n$ et calcule sa clé public $Pb = nbxG$. Pour chiffrer le message en point P_m au destinataire, l'émetteur choisi un nombre aléatoire k et calcule la paire de points P_C en utilisant la clé privée du destinataire.

$$P_C = [(kG), (Pm + kPb)] \tag{1.5}$$

Après la réception de la paire de points P_C, le destinataire multiplie le premier point kG, avec sa clé privée nb. Puis l'additionne avec le deuxième point de P_C $(P_m + kPb)$.

$$P_m = (P_m + kPb) - [nb(kG)] = (P_m + knbG) - [nb(kG)] \tag{1.6}$$

P_m représente l'ensemble des points qui correspondent au message M en clair. Seulement le destinataire connait la clé privée nb, peut retrancher $nb(kG)$ du deuxième point du message chiffré [13]. Les opérations de chiffrement et de déchiffrement sont montrées sur la figure 1.4.

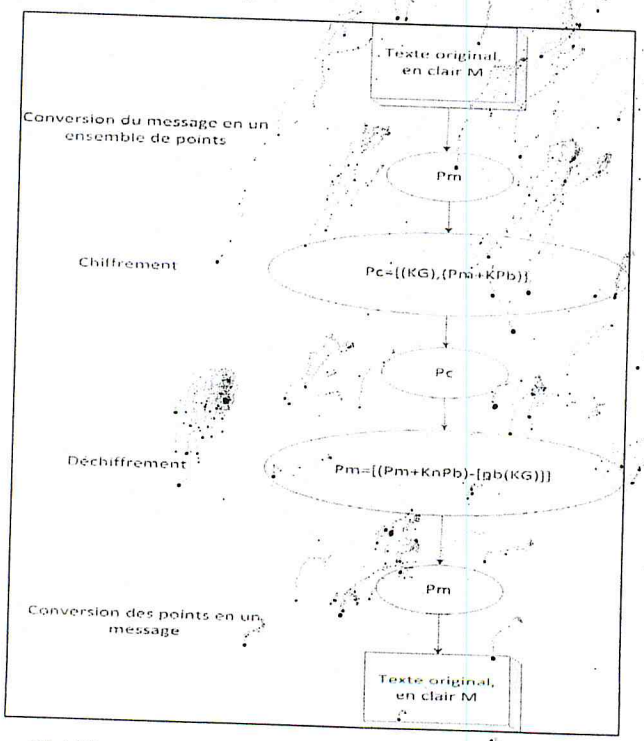

Figure 1.4 : Chiffrement et déchiffrement à base de courbe elliptique.

Nous supposons qu'une courbe elliptique E_p est définie dans un corps premier F_p. L'ensemble de points sur la courbe avec le point à l'infini, noté O, forment un groupe abélien dont la loi de composition est l'addition de points satisfaisant les conditions suivantes [10]:

- Fermeture : $\forall(P1, P2) \in E_p^2 \mid P1 + P2 \in E_p$.
- Commutativité : $\forall(P1, P2) \in E_p^2 \mid P1 + P2 = P2 + P1$.
- Associativité : $\forall(P1, P2, P3) \in E_p^3 \mid (P1 + P2) + P3 = P1 + (P2 + P3)$.
- Elément neutre O: $\forall P \in E_p \mid P + O = O + P = O$.
- Elément symétrique : $\forall P \in E_p, \exists Q \in E_p \mid P + Q = Q + P = O$.

Pour obtenir le point symétrique de P, il suffit de changer le signe de sa coordonnée y. c'est-à-dire :

$$Si\ P = (x, y), alors\ -P = (x, -y)\ et\ P + (-P) = O$$

1.3.4 Les règles d'addition et dédoublement de points

Le calcul de l'addition de deux points P et Q est défini par les équations (1.7),(1.8) [13]. Si on considère que :

$P = (x_1, y_1) \in E, Q = (x_2, y_2) \in E$ et P ≠Q, les coordonnées (x_3, y_3)de P + Q sont définies par les équations suivantes:

$$x_3 = {\lambda_a}^2 - x_1 - x_2\ mod\ N \quad (1.7)$$

$$y_3 = \lambda_a(x_1 - x_3) - y_1\ mod\ N \quad (1.8)$$

Avec $$\lambda_a = \left(\frac{y_2 - y_1}{x_2 - x_1}\right) mod\ N \quad (1.9)$$

Le dédoublement $2P$ du point P est défini comme suit :

$Si\ P = (x_1, y_1) \in E$ alors les coordonnées (x_3, y_3) de $2P$ sont définie par les équations (1.10) et (1.11) .

$$x_3 = {\lambda_d}^2 - 2x_1\ mod\ p \quad (1.10)$$

$$y_3 = {\lambda_d}^2(x_1 - x_3) - y_1\ mod\ N \quad (1.11)$$

Avec $$\lambda_d = \left(\frac{3{x_1}^2 + a}{2y_1}\right) mod\ N \quad (1.12)$$

1.3.5 Génération des clés publiques et privées

Soit une courbe elliptique défini par $E(N, a, b, G, n)$. Une paire de clé (d, Q) est associée à cette courbe. d représente la clé privée. Celle-ci est un entier généré d'une

manière aléatoire dans l'intervalle $[1, n-1]$. Q est la clé public qui est associée à la clé privée d [13].

La génération de la paire de clé (d, Q) se fait comme suit :

- Choisir l'entier d dans l'intervalle $[1, n-1]$
- Calculer $Q = dG$
- Retourner (d, Q)

L'algorithme 1.1 représente l'algorithme de génération de clé [10].

Algorithme 1.1- Algorithme de génération de la paire de clé.

Entrées : $E(p, a, b, G, n)$
Sortie : (d, Q)
Début
1) d = Random(1, n − 1);
2) *Calculer* $Q = d \times G$;
3) *Retourner* (d, Q);
Fin

1.3.6 Multiplication scalaire

En analysant les équations (1.5), (1.6) et l'algorithme de génération de clé, on constate que l'opération cœur est le calcul de la multiplication d'un scalaire k par un point P. Cette dernière est notée par $Q = k \times P$, où $k \in F_p$, $(P, Q) \in E^2_p$. A travers tous les algorithmes développés dans la littérature, l'opération $k \times P$ est souvent considérée comme étant une opération assez complexe en termes d'opérations arithmétiques de base requises. Une multiplication de point peut être considérée comme une suite d'addition de point consécutive :

$$Q = k \times P = \underbrace{P + P + \cdots + P}_{k\ fois}$$

La méthode pour calculer la multiplication scalaire est d'utiliser l'algorithme de doublement et d'addition de point (voir algorithme 1.2). Supposons que P est un point sur une courbe elliptique qui est définie sur un corps premier, notée (F_p) . Pour calculer $k \times P$ où k est un nombre entier positif de longueur l bits, nous représentons k en binaire $\sum_{i=0}^{l-1} k_i\, 2^i$. Ensuite nous parcourons k du bit de poids fort au bit de poids faible.

L'algorithme qui permet de calculer la multiplication scalaire est défini comme suit [14]:

Algorithme 1.2- Algorithme de la multiplication scalaire $Q = k \times P$

Entrées : $k = (1, k_{l-2}, k_{l-3}, \dots k_0)_2$, $P \in E, a$
Sortie : $Q = k \times P$
Début

1) $R_0 \leftarrow P$;
2) $R_1 \leftarrow Dédoublement\ (P, a)$;
3) **Pour** $i = l - 2$ **jusqu'**à 0 **faire**
4) **Si** $k_i = 0$ **alors**
5) $R_1 = Addition(R_0\ , R_1\ )$;
6) $R_0 = Dédoublement(R_0\ , a)$;
7) **Sinon**
8) **Si** $k_i = 1$ **alors**
9) $R_0 = Addition(R_0\ , R_1\ )$;
10) $R_1 = Dédoublement(R_1\ , a)$;
11) **Fin si**
12) **Fin si**
13) **Fait**
14) $Q \leftarrow R_0$;
15) *Retourner* Q;

Fin

1.4 conclusion

Dans ce chapitre, nous avons présenté le protocole de cryptographie à base de courbes elliptiques ainsi que les notions mathématiques de base pour la compréhension des courbes elliptiques ECC. Nous avons constaté que le fonctionnement de ce genre de crypto système est basé sur la multiplication scalaire. Dans le chapitre suivant, nous allons présenter les algorithmes de calculs qui permettent l'exécution de ses opérations de base.

2.1 Introduction

Dans le premier chapitre, nous avons étudié les étapes de bases qui constituent le protocole de cryptographie ECC. Nous avons constaté que la multiplication scalaire $k \times P$ est l'opération cœur de ce protocole. Celle-ci est utilisée non seulement pour la génération des clés, mais aussi lors du chiffrement et du déchiffrement des données. De ce fait, augmenter les performances d'exécution d'un crypto système ECC est fortement lié à l'optimisation de la multiplication scalaire et de ses opérations arithmétiques de base.

Ce chapitre est consacré à l'étude algorithmique des opérations arithmétique requises pour le calcul de la multiplication scalaire.

2.2 Arithmétique modulaire

L'arithmétique modulaire est un ensemble de méthodes permettant la résolution de problèmes sur les nombres entiers. Ces méthodes dérivent de l'étude du reste obtenu par une division euclidienne. L'idée de base de l'arithmétique modulaire est de travailler non pas sur les nombres eux-mêmes, mais sur les restes de leur division par entier. Dans ce qui suit nous allons présenter les différentes opérations arithmétiques utilisées dans le calcul de la multiplication scalaire, en l'occurrence, l'addition, la soustraction, la multiplication et l'inverse modulaires. Chaque opération sera présentée avec son algorithme d'exécution.

2.2.1 Addition modulaire

L'addition modulaire peut être décomposée en une addition suivie d'une réduction. En d'autre terme, elle consiste à additionner deux nombres A et B, puis à prendre le reste T de la division de $(A + B)$ sur un entier N. L'addition modulaire est représentée par l'expression (2.1).

$$T = (A + B) \ mod \ N \tag{2.1}$$

L'algorithme de l'exécution de l'addition modulaire est défini comme suit[15] :

Algorithme 2.1- Algorithme de l'addition modulaire

Entrée: A, B, N

Sortie: $T = (A + B) \ mod \ N$

Variable intermédiaire: C

Début

1. $C = A + B;$
2. ***Si*** $C \geq N$ ***alors***
3. $C = C - N;$

4. ***Sinon***
5. $T = C;$
6. ***Fin si***
7. *Retourner* $T;$

Fin

2.2.2 Soustraction modulaire

La soustraction modulaire est décomposée en une soustraction suivie d'une réduction. Elle consiste à soustraire deux nombres A et B puis à prendre le reste T de la division de $(A - B)$ sur un entier N. La soustraction modulaire est représentée par l'expression (2.2).

$$T = (A - B) mod\ N \tag{2.2}$$

L'algorithme de l'exécution de la soustraction modulaire est défini comme suit [15]:

Algorithme 2.2- Algorithme de la soustraction modulaire

Entrée: A, B, N

Sortie: $T = (A - B)\ mod\ N$

Variable intermédiaire: C

Début

1. $C = A - B;$
2. ***Si*** $C < 0$ ***alors***
3. $C = C + N;$
4. ***Sinon***
5. $T = C;$
6. ***Fin si***
7. *Retourner* $T;$

Fin

2.2.3 Multiplication modulaire

La multiplication modulaire consiste à multiplier deux nombres $(A \times B)$ et à prendre le reste T de la division par un entier N du produit obtenu. Cette opération est représentée par l'expression (2.3)

$$T = (A \times B)\ mod\ N \tag{2.3}$$

L'algorithme de l'exécution de la multiplication modulaire est défini comme suit : [15]

Algorithme 2.3- Algorithme de la multiplication modulaire

Entrée: A, B, N

Sortie: $T = (A \times B)\ mod\ N$

Variable intermédiaire: C

Début

1. $C = A \times B;$
2. $T = C \bmod N;$
3. *Retourner* $T;$

Fin

La division est une opération complexe. Son temps de calcul est beaucoup plus important par rapport au délai d'exécution de la multiplication [16] . Il est alors très couteux d'effectuer la réduction modulaire par la division. Pour se remédier à ce problème plusieurs, méthodes ont été proposées pour éviter la division. Parmi ces dernières la méthode proposée par Montgomery est considérée comme étant la méthode la plus adaptée pour une implémentation matérielle de la multiplication modulaire, puisque la division par la base revient à effectuer un simple décalage.

2.2.4 Multiplication Modulaire de Montgomery (MMM)

La multiplication de Montgomery est un algorithme développé par Montgomery en 1985. Il a pour but de transformer la réduction en modulo N en une division par une puissance de base [17]. Cette méthode calcule $A \times B \times R^{-1} \bmod N$, où N est un entier impair codé sur n chiffres. A et B sont strictement inferieurs à N. R est une constante tel que $R \geq \beta^{n}$ et $PGCD(R,N) = 1$. β est la base de représentation des données. L'algorithme de Montgomery est défini comme suit [18]:

Algorithme 2.4- Algorithme de la multiplication de Montgomery.

Entrées: A, B, N *avec* $0 \leq A, B < N$

Précalculés: N' *et* R *avec* $R \geq \beta^{-n}, (-N) \times N' = 1 \bmod R$ *et* $PGCD(R,N) = 1$

Sortie: $T = (A \times B \times R^{-1}) \bmod N$

Variable intermédiaire: C

Début

1. $C = A \times B;$
2. $q = (C \times N') \bmod R;$
3. $T = (C + q \times B)/R;$
4. ***Si*** $T \geq N$ ***alors***
5. $T \leftarrow T - N;$
6. ***Si non***
7. $T \leftarrow T;$

8. ***Fin si***
9. *Retourner T;*

Fin

La MMM de deux entiers A et B est donnée par l'expression (2.4) [17].

$$T = Montgomery\ (A, B) = (A \times B \times R^{-1}) mod\ N \qquad (2.4)$$

2.2.4.1 Représentation de Montgomery et ses propriétés

Le résultat obtenu par la MMM comporte le facteur R^{-1}. Ce qui pose une difficulté lors d'un calcul intensif de la multiplication modulaire. Pour cela, Montgomery utilise une conversion des données d'entrés vers un autre domaine de représentation, en utilisant la notation de Montgomery. Cette dernière est notée par $mon(X)$. Le principe est de calculer $mon(X) = (X \times R) mod\ N$ où $X < N$ et $R \geq \beta^n$. Cette notation engendre des propriétés de conversion et de stabilité [18]:

a. Conversion :

- De la représentation classique vers la représentation de Montgomery :

$$A \rightarrow mon(A): Montgomery(A, R^2\ mod\ N) = (A \times R^2 \times R^{-1}) mod\ N$$
$$= (A \times R) mod\ N = mon(A)$$

- De la représentation de Montgomery vers la représentation classique:

$$mon(A) \rightarrow A: Montgomery(mon(A), 1) = (mon(A) \times 1) mod\ N$$
$$= (A \times R \times R^{-1}) mod\ N = A\ mod\ N$$

b. Stabilité :

La MMM est stable par rapport à la multiplication. En d'autre terme :

$$Mongomery\big(mon(A), mon(B)\big) = (mon(A) \times mon(B) \times R^{-1}\ mod\ N$$
$$= (A \times R \times B \times R \times R^{-1}) mod\ N = (A \times B \times R) mod\ N = mon(A \times B)$$

2.2.4.2 Variantes de la multiplication modulaire de Montgomery

Dans l'ECC, les tailles des clés utilisées actuellement sont de l'ordre 160 bits et plus. En effet, lorsque la taille de donnée est importante, ceci augmente la complexité de calcul de la MMM. De plus, sa réalisation devient difficile si les ressources matérielles sont limitées sur le support d'implémentation. Dans le but d'atteindre la mise en œuvre de cette opération sur matériel, des modifications ont été apportées sur son algorithme original (Algorithme 2.4).

2.2.4.3 Multiplication Modulaire de Montgomery entrelacée (MMME)

Dans notre travail, nous nous sommes basées sur l'utilisation de la variante dite : Multiplication de Montgomery Entrelacée Sans Soustraction Finale. La description et l'exécution de son algorithme est détaillée dans [19].

a Algorithme de la MMMESF

L'algorithme de la MMMESF est défini comme suit :

Algorithme 2.5- Algorithme de la MMME

Entrée : $A = \sum_{i=0}^{n} a_i \times \beta^i, B = \sum_{i=0}^{n} b_i \times \beta^i, N = \sum_{i=0}^{n} n_i \times \beta^i$

$avec\ 0 \leq A, B < 2 \times N\ et\ N_n = 0$

Précalculés: $N' = -N_0^{-1} mod\ \beta, avec\ \beta = 2^k, PGCD\ (N, \beta) = 1\ et$

$R = 2^{(n+1)\times k}\ avec\ k \geq 2$

Variables intermédiaires: $q_i, T_i = \sum_{j=0}^{n} T_{(i),j} \times \beta^j$

Sortie: $T = \sum_{j=0}^{n} T_{(n),j} \times \beta^i = (A \times B \times R^{-1}) mod\ N$

Début

1. $T_{(0)} = 0;$
2. ***pour*** i ***allant de*** 0 à n ***faire***
3. $q_{(i)} = ((T_{(i),0} + (a_i \times b_0)) \times N') mod\ \beta;$
4. $T_{(i+1)} = (T_{(i)} + (a_i \times B) + q_i \times N) / \beta;$
5. ***Fin pour***
6. $Retourner\ T = T_{(i+1)};$

Fin

b Complexité temporelle de l'algorithme 2.5

Les performances de l'algorithme 2.5 dépendent des deux principaux paramètres, $\beta = 2^k$ la base de représentation des données et la taille du modulo N en terme de nombre de bits.

Pour un modulo de taille m bits, l'algorithme 2.5 aura besoin de $\left(\frac{m}{k}\right) + 1$ itérations pour terminer l'exécution de la multiplication de Montgomery. De ce fait, un temps nécessaire pour calculer la MMM est: $t_m = t_i \times \left[\left(\frac{m}{k}\right) + 1\right]$. t_i représente le délai d'exécution d'une itération(i). Le nombre d'itération est lié à la base β utilisée pour représenter les données. En effet, en base, $\beta = 2^k$ avec $k \geq 2$, les chiffres a_i , q_i et N' sont représentés sur k bits dans le systéme $\{0,1, \ldots, \beta - 1\}$. De ce fait, le paramètre k qui définit la base β influe sur les performances d'exécution de cet algorithme.

L'augmentation de k, réduit d'une part le nombre d'itération nécessaires à l'exécution de l'algorithme 2.5. D'autre part, cette réduction entraine une complexité car l'algorithme 2.5 exige à chaque itération (i):

- Une multiplication pour calculer q_i.
- Deux multiplications et deux additions pour calculer $T_{(i)}$.

La complexité de calcul de l'algorithme 2.5 est définie par l'équation suivante :

$$C_{alg2.5} = (n+1) \times [3 \times Mul + 2 \times Add]$$

Avec $n = {}^{m}/_{k}$

2.2.5 Inverse modulaire

L'inverse modulaire est défini par l'équation suivante :

$$T = x^{-1} mod\ N \quad (2.5)$$

En vertu de la complexité de calcul de cette opération, celle-ci peut être exécutée par un calcul d'une exponentiation modulaire, en utilisant le petit théorème de Fermat [20]. Ce théorème est défini comme suit [24] :

Si N est un nombre premier, nous avons :

$$1 = x^{N-1} \ mod\ N$$

Ainsi :

$$x^{-1} = x^{N-2} \ mod\ N \quad (2.6)$$

D'après l'équation (2.6) nous constatons que le calcul de l'inverse modulaire est transformé au calcul de l'exponentiation modulaire.

2.2.6 Exponentiation modulaire

L'exponentiation modulaire est définie par l'équation :

$$C = X^{y} \ mod\ N \quad (2.7)$$

Dans la littérature plusieurs algorithmes sont proposés pour calculer l'exponentiation modulaire. Dans notre travail nous avons utilisé l'algorithme Montgomery Power Ladder [21]. Ce dernier est défini dans l'algorithme 2.6.

Algorithme 2.6- Algorithme de Montgomery Power Ladder

Entrées: $C, X, N, y = (y_{l-1}, \dots, y_0)_2$

Sortie: $C = X^{y} \ mod\ N$

Début

1. $Z_0 \leftarrow 1$;
2. $Z_1 \leftarrow X$;
3. ***pour*** $j = l-1$ à 0 ***faire***
4. ***si*** $(y_j = 0)$ ***alors***

5. $Z_1 \leftarrow Z_0 Z_1 \ mod \ N;$
6. $Z_0 \leftarrow (Z_0)^2 \ mod \ N;$
7. ***sinon*** $(y_j = 1)$
8. $Z_0 \leftarrow Z_0 Z_1 \ mod \ N;$
9. $Z_1 \leftarrow (Z_1)^2 \ mod \ N;$
10. ***Fin si***
11. ***fin pour***
12. *Retourner* $Z_0;$

Fin

Cet algorithme est basé sur le décalage à gauche de la représentation binaire de l'exposant y. A chaque itération de l'algorithme deux multiplications modulaire et deux élévations au quarré sont effectuées en parallèle. Son exécution utilise deux variables R_0 et R_1. A l'étape initiale, R_0 et R_1 sont initialisées respectivement par 1 et le message X. La complexité de cet algorithme en terme de nombre de multiplication modulaire est déterminée par le nombre de ces opérations à exécuter à chaque itération. En effet, comme à chaque itération deux multiplications modulaires sont calculées, ainsi la complexité de l'algorithme 2.6 peut être évaluée comme suit $C_{algo\ 2.6} = 2 \times n \times MM$.

2.3 Adaptation des opérations arithmétiques de base au processeur Microblaze

Dans les opérations d'addition et du dédoublement définies respectivement par les expressions (1.7, 1.8) et (1.10, 1.11) les opérations arithmétiques de base sont : l'addition modulaire, la soustraction modulaire, la multiplication modulaire et l'inverse modulaire. Les données d'entrés/sorties sont de taille 256-bits. Le bus de données du processeur Microblaze est de taille 32-bits. Cette contrainte nous impose de représenter toutes les données manipulées en base $\beta = 2^{32}$, telles que :

$$D = \sum_{i=0}^{n-1} D[i] \times 2^{i \times 32} \text{ avec } D[i] = \sum_{j=0}^{31} D_j \times 2^j$$

Ces données sont stockées dans des mémoires embarquées sur circuit FPGA, ainsi toutes les opérations arithmétiques des algorithmes 2.1, 2.2 et 2.3 sont effectuées en mode série sur une précision de taille 32-bits. La figure 2.1 résume le déroulement de ces opérations sur un bus de données de taille 32-bits[25][26].

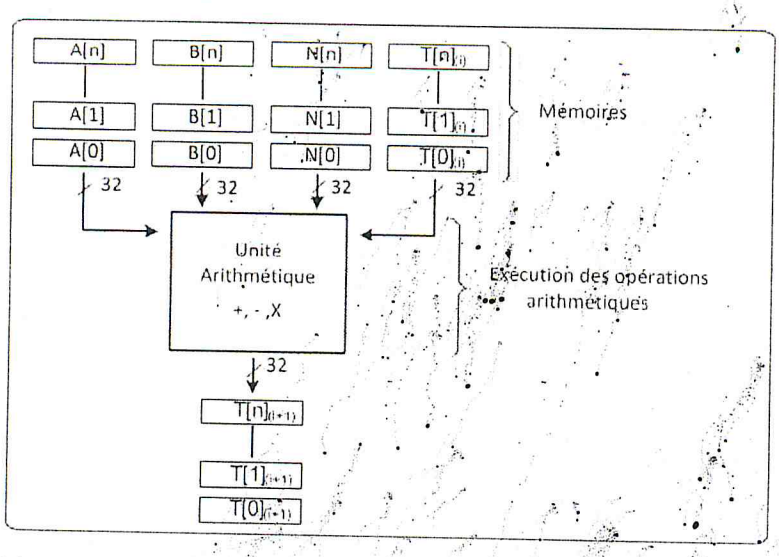

Figure 2.1 : Décomposition des opérandes et des résultats intermédiaires en chiffres de taille 32-bits

Dans notre travail, la taille des données mises en exécution en termes du nombre de bits est 256 bits. Le nombre de chiffres issu du codage de ces données en base 2^{32} est de $n = {}^{256}/_{32} = 8\ chiffres$. La taille en question correspond au niveau de sécurité du crypto système ECC, défini dans [22].

Dans le but d'adapter l'exécution de l'algorithme 2.5 aux ressources interne du processeur, des modifications sur la partie arithmétique de cet algorithme sont nécessaires. Pour éviter l'utilisation de registres de grandes tailles, toutes les données sont supposées stocker dans des mémoires embarquées. Les opérations arithmétique de l'algorithme sont effectuées en mode série sur une précision de taille 32-bits. Le résultat T de la MMM est obtenu à la fin chiffre par chiffre.
L'algorithme final est détaillé dans [19].

2.4 Multiplication scalaire de Montgomery

La stabilité et la représentation de Montgomery sont intéressantes lorsque le calcul intensif de la multiplication scalaire est exigé. Ce qui est le cas pour le calcul de la multiplication scalaire $G = k \times P$. Les étapes de la multiplication scalaire de Montgomery, basée sur la MMM sont résumé sur la figure 2.2.

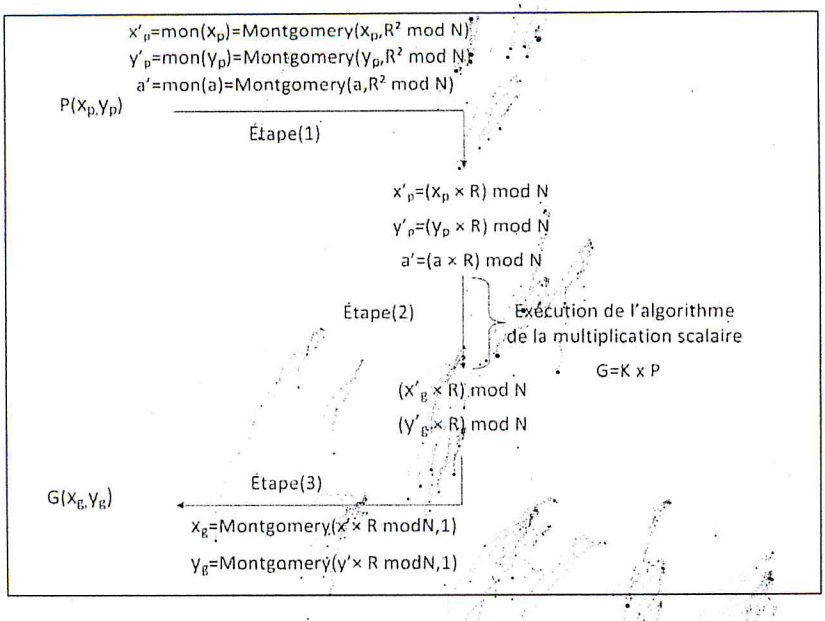

Figure 2.2 : Etapes de calcul de la multiplication scalaire en utilisant la MMM.

Dans une première étape, une conversion des données d'entrées du domaine classique vers le domaine de Montgomery est effectuée. Celle-ci concerne les coordonnées du point $P = (x_p, y_p)$ et le paramètre a de la courbe elliptique. La conversion en question est réalisée par trois MMM, en spécifiant x_p, y_p, a et $(R^2 \, mod \, N)$ comme opérandes d'entrées. La valeur de $R^2 \, mod \, N$ est une constante qui dépend de N et de la base β. Ainsi, les calculs sont exécutés dans le domaine de Montgomery tout au long du processus de calcul de la multiplication scalaire de Montgomery. Finalement, les coordonnées (x_g, y_g) du point résultant G sont obtenues par une conversion du domaine de Montgomery vers le domaine classique. Celle-ci est effectuée par deux MMM, qui prennent comme opérandes d'entrée les résultats x'_g, y'_g issus de l'étape précédente et "1" [23], [17].

L'algorithme 2.7 représente l'algorithme de calcul de la multiplication scalaire $\times G$, basé sur la multiplication de Montgomery.

Algorithme 2.7- Algorithme de la multiplication scalaire de Montgomery

Entrées: $k = (1, k_{t-2}, k_{t-3}, \ldots k_0)_2, G(x_g, y_g) \in E(F(p)), a, N \; et \; R$

Sortie: $Q(x_q, y_q) = k \times G$

Début

1. $x_g = mon(x_g) = Montgomery(x_g, R^2 mod \, N);$
2. $y_g = mon(y_g) = Montgomery(y_g, R^2 mod \, N);$
3. $a = mon(a) = Montgomery(a, R^2 mod \, N);$
4. $R_0 \leftarrow G;$

5. $R_1 \leftarrow DédoublGment\ (P, a, N);$
6. ***Pour*** i ***allant de*** $t-2$ ***jusqu'***à 0 ***faire***
7. ***Si*** $k_i = 0$ ***alors***
8. $R_1 = Addition(R_0\ , R_1\ , N);$
9. $R_0 = Dédoublement(R_0\ , a\ , N);$
10. ***Si non***
11. $R_0 = Addition(R_0\ , R_1\ , N);$
12. $R_1 = Dédoublement(R_1\ , a, N);$
13. ***Fin si***
14. ***Fin pour***
15. $Q = R_0;$
16. $x_q = Montgomery(x_q \times R\ mod\ N, 1);$
17. $y_q = Montgomery(y_q \times R\ mod\ N, 1);$
18. $Q = (x_q, y_q);$
19. $Retourner\ Q;$
20. ***Fin***

L'exécution de cet algorithme est basée sur des décalage à gauche de la représentation binaire du scalaire. Après l'étape d'initialisation, ou toutes les données d'entrées sont converties dans le domaine de Montgomery, le processus itératif consiste en l'exécution d'un dédoublement et d'une addition de points.

La complexité $C_{algo2.7}$ de l'algorithme 2.7 en fonctions des opérations de base est montrée sur le tableau 1.1

Tableau 1.1 : Complexité algorithmique de l'algorithme 2.7.

Opération	Complexité
Exp (Algorithme 2.6)	$2 \times l \times MM$
C_{add2p} : addition de deux points Equations : 1.7, 1.8, 1.9	$(2 \times l + 3)MM + 5sub + add$
C_{ded2p} : dédoublement de point Equations : 1.10, 1.11, 1.12	$(2 \times l + 4)MM + 5add + 3sub$
Multiplication scalaire (algorithme 2.7)	$5MM + (t-1)(C_{add2p}+C_{ded2p})$

2.5 Conclusion

Dans ce chapitre, nous avons étudié les algorithmes d'exécution des opérations de base de la multiplication scalaire. Nous avons aussi présenté une méthode pour le calcul de la multiplication modulaire, en l'occurrence la multiplication de Montgomery et une méthode optimale pour le calcul de l'inverse modulaire. Nous avons aussi étudié l'adaptation de toutes les données manipulées par le crypto système ECC aux ressources internes du processeur Microblaze. Le chapitre suivant est consacré à l'étude des systèmes sur puce, en particulier les systèmes implémentés sur carte FPGA.

Chapitre 3

Systèmes embarqués

3.1 Introduction

L'évolution des technologies de fabrication des circuits intégrés fait que l'on est amené à concevoir des circuits de plus en plus complexes (plusieurs millions de transistors). Durant les dernières décennies, on est passé de la conception des circuits composés de quelques milliers de portes à des systèmes structurés et intégrés comme un réseau sur une même puce. Les puces modernes peuvent contenir plusieurs processeurs, de la mémoire et un réseau de communication complexe. Le principe de la conception reste le même ; il s'agit de générer une réalisation physique sous forme d'une puce en partant d'une spécification du système. Par contre, les outils mis en œuvre et le processus de conception ont beaucoup plus évolué. Partant d'une conception complètement manuelle où l'on dessinait les masques du circuit à réaliser, on est passé à une conception quasi automatique en partant d'une description du comportement du circuit sous forme d'un programme décrit dans un langage de haut niveau. Dans ce chapitre nous allons introduire les systèmes embarqués sur puce et les différentes plateformes d'implémentations ainsi que le partitionnement matériel/logiciel.

3.2 Circuits intégrés

Le circuit intégré (CI), aussi appelé puce électronique, est un composant électronique reproduisant une ou plusieurs fonctions électroniques plus ou moins complexes. Il intègre souvent plusieurs types de composants électroniques de base sur une surface réduite, rendant le circuit facile à mettre en œuvre [27],[28].

3.3 Système embarqué sur puce

Un système sur une puce ou SoC pour (System On Chip), constitue un circuit complexe qui intègre tous les éléments fonctionnels d'un produit sur une même puce, pouvant comprendre de la mémoire, un ou plusieurs processeurs, des périphériques d'interface, ou tout autre composant nécessaire à la réalisation de la fonction attendue. Un système embarqué est autonome il ne possède pas d'entrés/sorties standards [28][4][36]. Une architecture simplifiée d'un système embarqué est montré sur la figure 3.1.

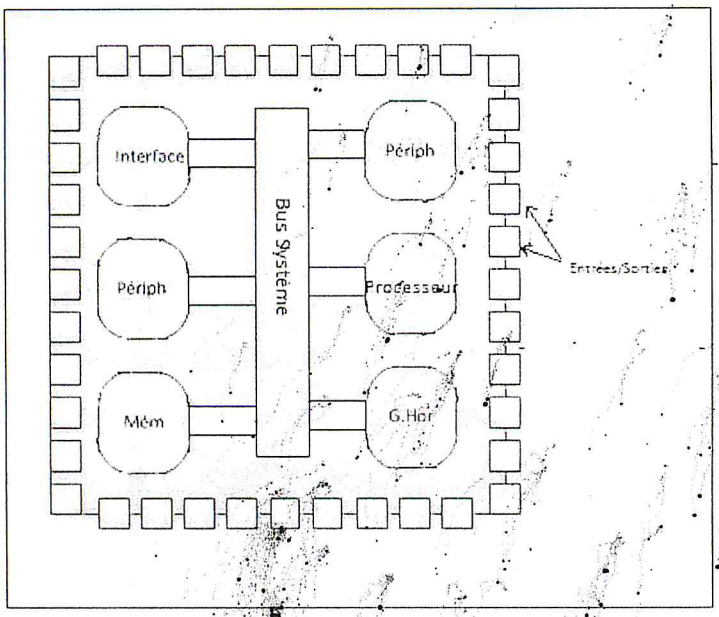

Figure 3.1 : Architecture d'un système embarqué.

Pour la réalisation d'un tel système, il est primordial de choisir une plateforme pour son implémentation.

3.4 Plateforme d'implémentation de systèmes embarqués

Le choix de la plateforme qui convient au mieux au système attendu est basé sur la réponse de la question suivante : « comment atteindre le fonctionnement correcte d'un système avec un faible cout de réalisation et en respectant des contraintes supplémentaires ? » [4].

Il existe plusieurs types de plateformes matérielles d'implémentation, à savoir :

- Micro contrôleur.
- DSP
- Circuit ASIC
- Circuit FPGA

En effet, le microcontrôleur est un dispositif à usage général pour le traitement et le contrôle des informations qui peut être adapté à une grande variété de système. Il est utilisé généralement pour les systèmes ne nécessitant pas de grandes capacités de traitement de données. Les DSPs (Digital Signal Processor) sont recommandés si les besoins en calculs sont plus importants. Dans le cas ou on vise un système avec des performances élevées, il est nécessaire de choisir des circuits spécifiques tel que ASICs (Application Specifie Inetegrated Circuit) et les FPGAs (Field Programmable Gate

Array). Un ASIC est conçu sur mesure pour une application particulière. Il intègre un ou plusieurs microcontrôleurs ou DSP. Il est recommandé pour une large série de fabrication. Un circuit FPGA est reconfigurable. Ce dernier ne devient pas figée après l'implémentation du système. Cette solution est recommandée pour le prototypage. Dans ces deux derniers types de plateformes, on parle souvent de SoC et de système sur circuit programmable PSoC (Programmable System On Chip) [4],[29].

3.5 Approche de conception logicielle/matérielle (co-design)

La conception des systèmes contenants une partie logicielle et une partie matérielle n'est pas un nouveau problème mais qui date depuis des années. La méthodologie classique de la conception de ces systèmes impose de séparer la partie logicielle de la partie matérielle très tôt dans le cycle de développement. L'interaction entre ces deux parties ne se fait que lors d'une phase bien précise qui est la phase d'intégration. Ce qui pose souvent des problèmes d'incompatibilité. Cette méthode de conception possède de nombreuses faiblesses qui la rendent incapable de supporter la complexité croissante des systèmes embarqués actuel. Parmi ces faiblesses un manque de description globale qui accompagnerait la conception du système (logiciel et matériel) du début à la fin. Cette description permettrait aux concepteurs (de la partie matérielle et de la partie logicielle) de bien connaître l'ensemble du système. Ce qui éviterait beaucoup de problème. L'intégration des différentes parties est effectuée à la fin du cycle : il est souvent trop tard pour lever les incompatibilités. Les points faibles de cette méthodologie font d'elle une méthodologie inutile pour la conception des systèmes embarqués actuel. De ce fait une nouvelle approche de conception a eu naissance, le développement logiciel/matériel ou le co-design, qui consiste à développer conjointement les différentes parties d'un système hétérogène, dédiée à des applications bien spécifiques. Cette approche est recommandée notamment à la réalisation des systèmes embarqués de type SoC ou PSoC. Le système est conçu à partir d'une spécification unique décrivant sont architecture et son comportement. Cette phase de spécification est suivie d'une phase de partitionnement du système en parties. Une partie matérielle implantée sous forme de circuit matériel. Cette partie est utilisée pour satisfaire au mieux les contraintes temporelles. Une partie logicielle implantée sous forme d'un programme exécutable sur un processeur à usage général

embarqué sur la puce. Cette partie est exploitée pour sa flexibilité qui offre la possibilité de reprogrammation.

La partie logicielle est modélisée par un langage de description machine, le langage C ou C++ ou encor en assembleur. Cette partie est exécutée par le processeur. La partie matérielle est décrite à l'aide d'un langage de description matérielle tel que le VHDL ou le Verilog [4],[30].

La figure 3.2 schématise le concept d'implémentation logicielle/matérielle.

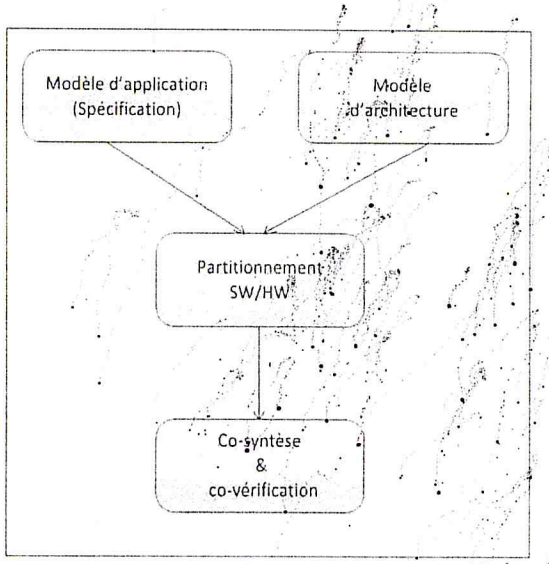

Figure 3.2 : Représentation de l'approche de conception co-design

Parmi les avantages de cette approche nous citons [4][30]:

- Possibilité de rajouter des fonctionnalités à l'application de base.
- Facilité de modifier une fonctionnalité en cas de besoin, en modifiant que la partie logicielle concernée. En conséquent les couts de modification diminuent.
- Les deux ressources matérielle et logicielle sont complémentaires et coopératives. En cas de besoin d'augmentation de la flexibilité du système, il est question d'améliorer la partie logicielle uniquement. Par contre l'amélioration des performances temporelles consiste à rajouter du matérielle uniquement.

3.6 Circuits FPGAs

Un circuit FPGA est un composant électronique composé de millions de transistors pour réaliser des fonctions plus ou moins complexes. La structure interne d'un FPGA peut être changée sans avoir à modifier sa structure globale. Ce qui permet de faire le

prototypage rapide et de moindre coût. L'avantage aussi est la possibilité de la mise à jour des composants implémentés.

Xilinx, Altera et Quicklogic et plusieurs autres compagnies produisent les FPGAs. Dans ce travail, nous nous intéressons aux circuis FPGAs de Xilinix. Toutes les compagnies partagent le même concept architectural, qui se divise en trois parties [28],[4],[31],[32]:

- Interfaces d'entrées/sorties.
- Les blocs logiques de base.
- Les interconnexions.

La figure 3.3, présente l'architecture globale d'un circuit FPGAs.

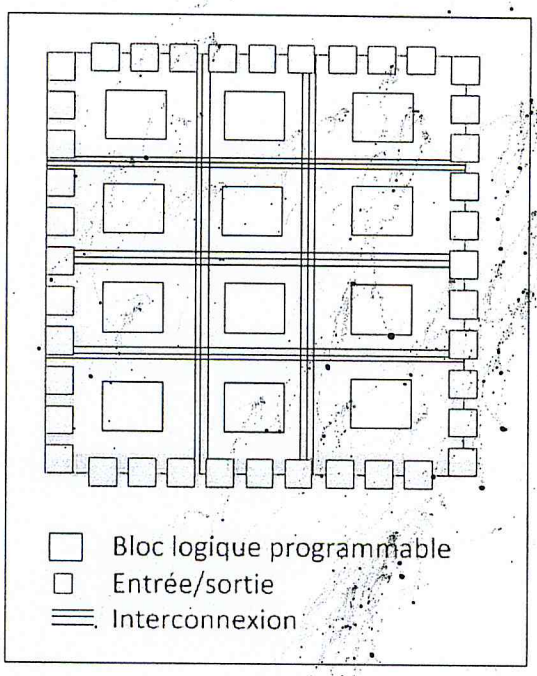

Figure 3.3 : Architecture globale d'un circuit FPGA

a. Les interfaces d'entrées/sorties

Les interfaces d'entrés/sortie sont l'intermédiaire entre les blocs logiques internes et les ressources externe. L'interfaçage des signaux peut être bidirectionnel ou unidirectionnel et peut avoir trois états (0, 1 et haut impédance).

b. Les blocs logiques

Les blocs logiques sont des blocs logiques programmables ou configurables pour réaliser une fonction logique qui peut être simple ou complexe.

Xilinx, nomme les blocs logiques de base CLB (Configurable Logic Blocks). Chaque CLB contient des slices, et chaque slice a des LUTs (lookup Tables)[35].

c. Les interconnexions

Les interconnexions transmettent les signaux d'un point à un autre.

Xilinx fournit plusieurs familles de circuits FPGA. Dans notre travail, nous nous intéressons à la famille Virtex-5.

3.7 Circuits FPGA de la famille Virtex-5

La famille Virtex-5 offre une densité supérieure à 12 M de portes logiques. Ils peuvent atteindre une fréquence de fonctionnement de 550 MHz. Ils sont fabriqués avec la technologie 65nm. Chaque CLB est constitué de 6 slices. Chaque slice contient un générateur de fonctions, une logique de traitement de la retenue (carry logic), des portes Xor dédiés à l'implémentation des additionneurs, des multiplexeurs, quatre bascules D et deux éléments de stockages [4],[31],[32].

Un circuit Virtex-5 est constitué aussi de blocs RAM de taille 32-Kbits et DSPs (Degital Signal Processor). Chacun de ces derniers peut être configuré en multiplieur ou multiplieur accumulateur ou encore un multiplieur suivi d'additionneur.

3.8 Système embarqué sur FPGA

L'évolution de la technologie de fabrication de circuits FPGAs, permet l'intégration de fonctionnalités de traitement embarqué sur ces derniers. Puisque, il n'y a aucune implémentation d'architecture fixe. Autrement dit aucune obligation que les fonctions soient réalisées par matériel plutôt que par logiciel. De plus les FPGAs sont reconfigurable, donc une fois l'application conçue de façon qui respecte toutes les contraintes, il n y a aucune raison de la garder sur le circuit. De ce fait, les FPGAs sont utilisés comme des plateformes de prototypage [4],[32],[33].

3.9 Processeur embarqué sur FPGAs

La conception de systèmes numériques complexes nécessite la mise en œuvre d'un processeur. Ce processeur embarqué, peut être soit de type « *softcore* » ou de type « *hardcore* ».

- Processeur « *Hardcore* » : déjà implanté sur le circuit électronique. Il est généralement choisit pour ses performances, que pour sa flexibilité.
- Processeur « *Softcore* » : ce type de processeurs est fourni comme une IP configurable. Il est généralement choisit pour sa flexibilité que ses performances. La portabilité vers n'importe quel circuit FPGA.

La description du processeur « *Soficore* » se fait à l'aide du langage de description materielle (VHDL, Verilog) qui est lié à un fondateur. De ce fait le code source ne peut être implémenté que dans les FPGAs du même fondateur [4],[9],[34].

Dans notre travail, nous nous intéressons au processeur Microblaze, propriétaire de Xilinx.

3.10 Processeur Microblaze

Le processeur Microblaze de Xilinx est un processeur-32 bits à jeux d'instruction réduit RISC (Reduced Instruction Set Computer). Son architecture est présentée dans la figure 3.4 [19] .

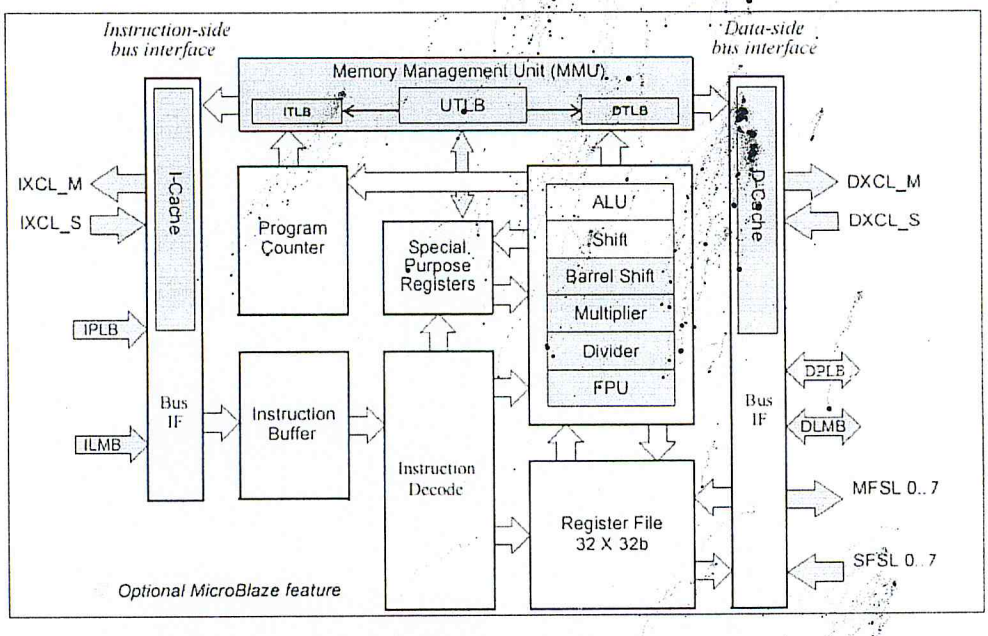

Figure 3.4 : Architecture interne du processeur Microblaze[19]

Les caractéristiques du processeur Microblaze peuvent se résumer dans les points suivants :

- Sa conception est basée sur une architecture Harvard avec des bus d'instruction et de données séparés.
- Il possède 32 registres à usage général.
- Les instructions sont exécutées en pipeline avec 5 étages de profondeur.

Les parties grises sont optionnelles et montrent à quel point ce processeur est configurable. Ses options, en l'occurrence le Barrel Shift, le Multiplier, le Divider, l'unité FPU (Floating Point Unit) et les mémoires caches sont des composants implémentés sur materiel. Leurs utilisations permettent d'accélérer les traitements de

données. Les fréquences maximales du processeur Microblaze varient selon la famille du circuit FPGA utilisée. La fréquence maximale de la famille Virtex-5 à laquelle nous nous intéressons est de 235 Mhz [19][6].

3.11 Système embarqué à base du processeur Microblaze

La conception et la réalisation des systèmes embarqués sur puce nécessitent un cycle de développement assez long. Donc, en premier lieu il est préférable de procéder au développement d'un prototype dans lequel les circuits FPGAs sont souvent utilisés. Les fondeurs de ce type de circuit mettent à la disposition des concepteurs, des cartes de prototypage à base de ces circuits, afin de réaliser des implémentations en un temps relativement faible. Dans notre travail la carte de prototypage mise à l'étude est la carte Genesy de Digilent qui comporte l'ensemble de composant nécessaire pour le développement et la vérification du fonctionnement d'une application PSoC. Elle est équipée du circuit FPGA XC5VLX50T de famille Virtex-5 [35].

Le circuit FPGA est entourée d'une panoplie de composants, citons un port RS232, une mémoire RAM de taille 256-MBytes, deux ports USBs, des interrupteurs, des LEds, un port Ethernet, un afficheur LCD, un générateur d'horloge programmable qui peur atteindre une fréquence de 100 Mhz...etc. Tous ces composant permettent à la carte FPGA de communiquer avec son environnement externe [4],[34],[35]. La carte de prototypage Gesenys est montrée dans la figure 3.5.

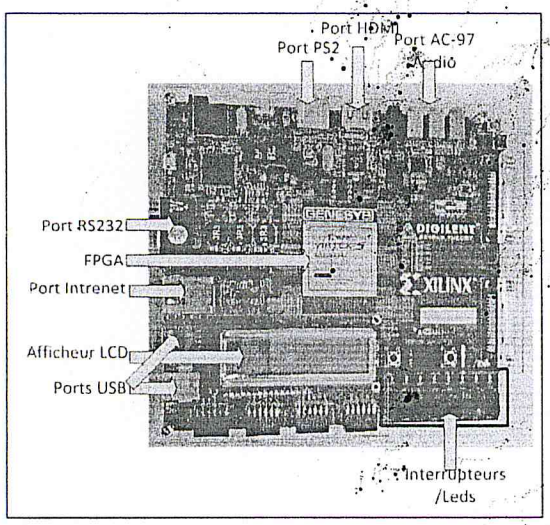

Figure 3.5 : carte de prototypage Gesenys

L'implémentation du processeur Microblaze en tant que contrôleur principale du PSoC, nécessite, selon le besoin du concepteur, l'ajout de périphériques sur le même circuit. Ces derniers servent d'interface entre le processeur et les différents composants

de la carte. Ils sont connectés autour de Microblaze par l'utilisation du bus système PLB (Processor Local Bus), comme ceci est montré dans la figure 3.6

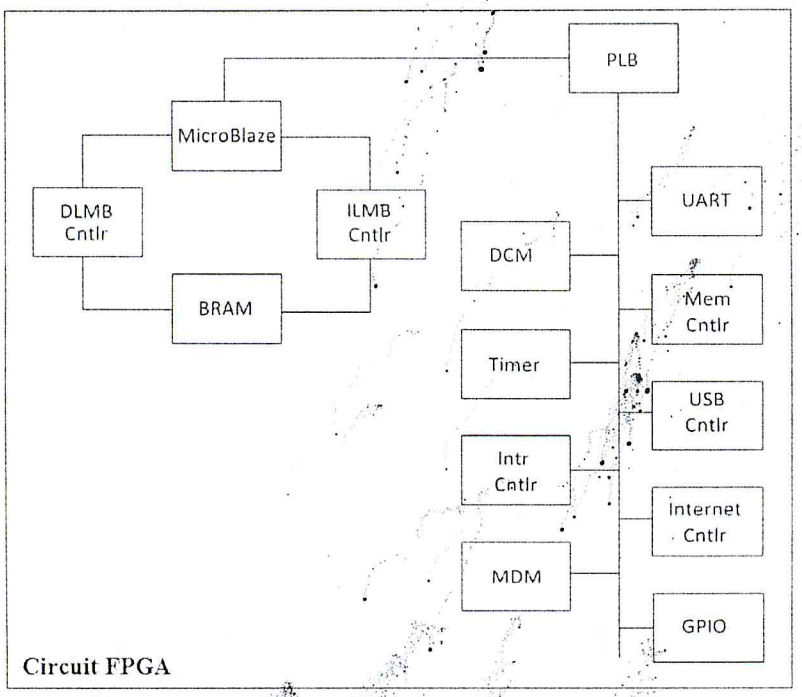

Figure 3.6 : Architecture d'un PSoC à base du processeur Microblaze

Parmi ces périphériques, on peut retrouver un UART (Universal Asynchronous Receiver/Transmitter) qui sert à la communication avec le port RS232 de la carte, un contrôleur de mémoire externe (Mem cntlr), des contrôleurs de ports USB et Ethernet (USB et Ethernet Cntlr) et des GPIOs (General Purpose Input Output). Ces derniers permettent de communiquer avec des LEDs, les interrupteurs et l'afficheur LCD.

D'autres périphériques de base sont indispensables au fonctionnement du systéme. Les périphériques en question sont constitué de[4],[35] :

- Une mémoire BRAM et ses contrôleurs (DLMB Cntlr et ILMB Cntlr).
- Une DCM (Degital Clock Manager) dont le rôle est la gestion du signal horloge dans le circuit FPGA.
- Un timer pour calculer les performances temporelles du système.
- Un contrôleur d'interruption (Intr Cntlr).
- Un MDM (Microblaze Debug Module) pour vérifier le fonctionnement correct des applications.

Chapitre 4

Implémentation de la plateforme de chiffrement/déchiffrement

3.12 Conclusion

Dans ce chapitre, nous avons présenté les systèmes embarqués sur puce et les différentes plateformes d'implémentations. En particulier, nous nous sommes focalisés sur les circuits FPGAs. Nous avons également présenté la famille du circuit FPGA ciblée, en l'occurrence la famille Virtex-5. Nous avons montré que ce type de circuit est très recommandé pour la réalisation des PSoCs à base du processeur Microblaze. En effet, en vu d'atteindre l'objectif de ce travail qui consiste en l'implémentation du crypto système ECC à base du processeur Microblaze, nous avons étudié son architecture de base et les ressources qui le constituent. Dans le chapitre suivant, nous allons détailler toute la méthodologie développée pour la réalisation du crypto système ECC.

4.1. Introduction

Le but de ce travail est la réalisation d'un crypto système ECC. En effet, les deux expressions (1.5), (1.6) présentées dans le chapitre 1, sont utilisées respectivement pour l'implémentation des opérations du chiffrement et du déchiffrement de ce crypto système. L'implémentation de ce crypto système est répartie sur deux niveaux d'abstraction. Le premier niveau est une Interface Homme/Machine (IHM), développée en langage Java. Cette dernière est exécutée sur un ordinateur. Le deuxième niveau d'abstraction est constitué par des fonctions décrites en langage C. Ces fonctions sont implémentées sur circuit FPGA et utilisées par le processeur Microblaze pour contrôler l'exécution des opérations du chiffrement et du déchiffrement.

Dans ce travail, notre objectif ne se limite pas uniquement à l'implémentation de ce crypto système, mais d'étudier ses performances en terme de délai d'exécution et de ressources matérielles requises. Pour ce faire nous avons proposé deux approches d'implémentation.

- La première approche est une implémentation purement logicielle. Celle-ci est basée sur l'exploitation du processeur Microblaze pour exécuter les opérations du chiffrement et du déchiffrement.
- La deuxième approche est une combinaison logicielle/matérielle. Cette dernière consiste à intégrer une IP autour du processeur. Celle-ci est dédiée à l'exécution de la MMM ainsi que l'exponentiation modulaire. Le contrôle de ces dernières est assuré par le processeur Microblaze.

Dans ce chapitre nous allons décrire de façon globale la plateforme proposée pour l'implémentation du crypto système ECC. Nous présentons par la suite la première approche et la deuxième approche d'implémentation. Enfin nous terminons par une description de l'IHM.

4.2. Description de la plateforme de chiffrement/déchiffrement

La plateforme de chiffrement/déchiffrement à base du crypto système ECC est composée de deux parties fondamentales liées par un canal de communication. Les parties en question sont une carte de prototypage à base de circuit FPGA et une interface Homme/Machine (IHM). La figure 4.1 schématise cette plateforme.

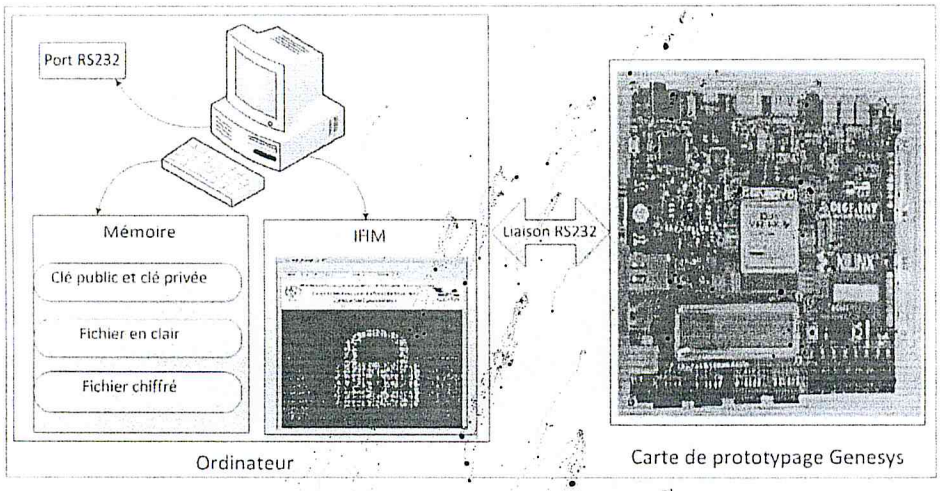

Figure 4.1: Plateforme du crypto système ECC.

a. La carte de prototypage

La carte de prototypage utilisée est la carte Genesys de Digilent. Elle est constituée d'un circuit FPGA de la famille Virtex-5, en l'occurrence le circuit XC5VLX50T [37]. Ce circuit a pour fonction l'exécution des opérations du chiffrement et du déchiffrement.

b. Le canal de communication

Ce canal est basé sur le protocole RS232, chargé du transfert des données de l'ordinateur vers la carte FPGA ou dans le sens inverse.

c. L'Interface Homme/Machine

C'est une interface réalisée en java, exécutée sur un ordinateur. Elle permet une meilleure flexibilité de l'utilisation du système embarqué sur circuit FPGA en se chargeant de la préparation de l'environnement du chiffrement et du déchiffrement. Parmi ses fonctions :

- Configuration des paramètres du protocole RS232
- Génération des clés de chiffrement et de déchiffrement.
- Représentation des données sous forme de suite d'octet.
- Transmission des données vers le circuit FPGA.
- Réception et affichage des résultats du chiffrement/ déchiffrement.

La plateforme réalisée permet aussi de traiter des fichiers stockés dans la mémoire interne de l'ordinateur. Son fonctionnement est résumé comme suit :

Après la configuration du protocole RS232 et la génération des clés, l'IHM intervient par la récupération de la clé publique ou privé du destinataire. L'implémentation de cette étape est réalisée en langage de programmation Java. Après la récupération de la clé, les données nécessaires pour le chiffrement/déchiffrement sont décomposées en des chiffres de taille 8-bits et transmises vers la carte de prototypage Genesys. Dans le cas du chiffrement, ces données sont constituées par la valeur du scalaire k, les coordonnés du point générateur G, de la clé public Pb et du message M en clair. Si l'opération considérée est un déchiffrement, ces données comportent la clé privé nb et les coordonnés de la multiplication scalaire $k \times G$ et du message chiffré. La sélection entre les deux opérations est effectuée grâce à une entrée supplémentaire introduite à partir de l'IHM.

A noter que les paramètres de la courbe utilisée correspondent au standard secp256r1 [22]. Ce dernier nous fournit un niveau de sécurité de 256 bits. Les valeurs des paramètres de la courbe à savoir a, b, n, N et G sont définit dans le chapitre suivant.

Après la réception des données, l'algorithme du chiffrement ou du déchiffrement sera exécuté. Une fois cette dernière est terminée sur le circuit FPGA, le résultat sera transmis vers l'ordinateur.

La complexité de l'exécution du crypto-système ECC est fortement liée au calcul de la multiplication scalaire, réalisée grâce à la partie embarquée sur FPGA. Cette partie est considérée comme le point clé de la plateforme réalisée.

4.3. Architecture du crypto système embarqué

L'architecture matérielle de l'approche utilisée est montrée sur la figure 4.2.

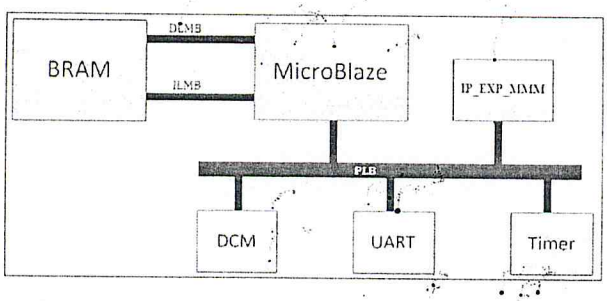

Figure 4.2 : Architecture matérielle du système embarqué.

Le système embarqué réalisé est composé de :

- Un processeur Microblaze.
- Une DCM (Digital Clock Manager).
- Une mémoire local BRAM (blockRAM).
- Un bus de données DLMB (Data Local Memory Bus).
- Un bus d'instructions ILMB (Instruction Local Memory Bus).
- Un bus système PLB (Processor Local Bus).
- Un UART .
- Un timer.
- Une IP matérielle.

Le processeur communique avec les périphériques qui l'entourent à travers le bus PLB qui est de 32-bits.

L'UART permet la communication avec le port RS232 de la carte FPGA. La taille de la mémoire BRAM a été configurée à 32 k-bits. Le débit de l'UART à 115200 bps.

Le rôle du Timer est de calculer le nombre de top d'horloge nécessaire aux opérations du chiffrement et de déchiffrement.

L'IP matérielle permet d'accélérer le calcul la multiplication modulaire de Montgomery et de l'exponentiation modulaire nécessaires pour l'exécution de l'opération de base du crypto système en l'occurrence la multiplication scalaire.

4.4. Implémentation purement logicielle

La partie logicielle du système embarqué réalisé a été implémentée en utilisant l'outil EDK (Embeded Desigein Kit) de Xilinx. Cette partie est constituée par un certain nombre de fonctions. Elles sont stockées dans la mémoire BRAM du processeur, puis exécutées par ce dernier.

Les taches attribuées au processeur Microblaze sont les suivantes :

- Réception des données d'entrées.
- Exécution de l'opération du chiffrement/déchiffrement, définie respectivement par les équations (1.5), (1.6) selon une entrée de sélection.
- Exécution des algorithmes des opérations de base à savoir : la multiplication scalaire (incluant l'addition de deux points et le dédoublement d'un point), l'addition, la soustraction, la multiplication et l'exponentiation modulaires.

- Transmission du résultat vers l'extérieur.

Les fonctions développées pour réaliser le chiffrement ou le déchiffrement sont hiérarchisées selon l'ordre décroissant du niveau d'abstraction. L'organisation de ces fonctions est montrée dans la figure 4.3.

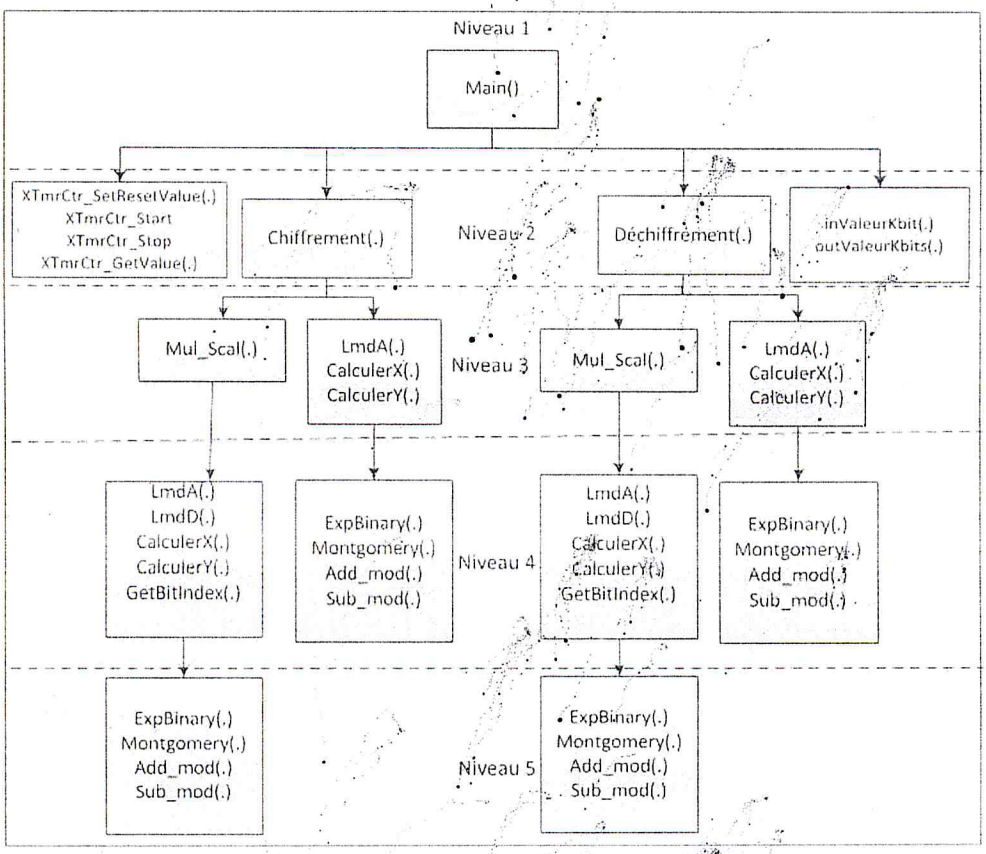

Figure 4.3 : Organisation des fonctions des parties logicielles du crypto système embarqué.

Le programme principale Main(.) est décrit en langage C. Il contient la fonction de communication avec l'UART et les fonctions du chiffrement et du déchiffrement, définit respectivement par $Chiffrement\ (.)$ et $Déchiffrement\ (.)$. Le programme C du programme principal $Main(.)$ est représenté sur la figure 1 de l'annexe.

4.4.1. Exécution d'un chiffrement/déchiffrement

Le fonctionnement du crypto système sur FPGA est réparti en trois étapes. Comme ceci comme est montré sur la figure 4.4

Ces étapes sont définies telles que :

- Réception des données d'entrées.
- Exécution des fonctions du chiffrement/déchiffrement.

- Transmission du résultat.

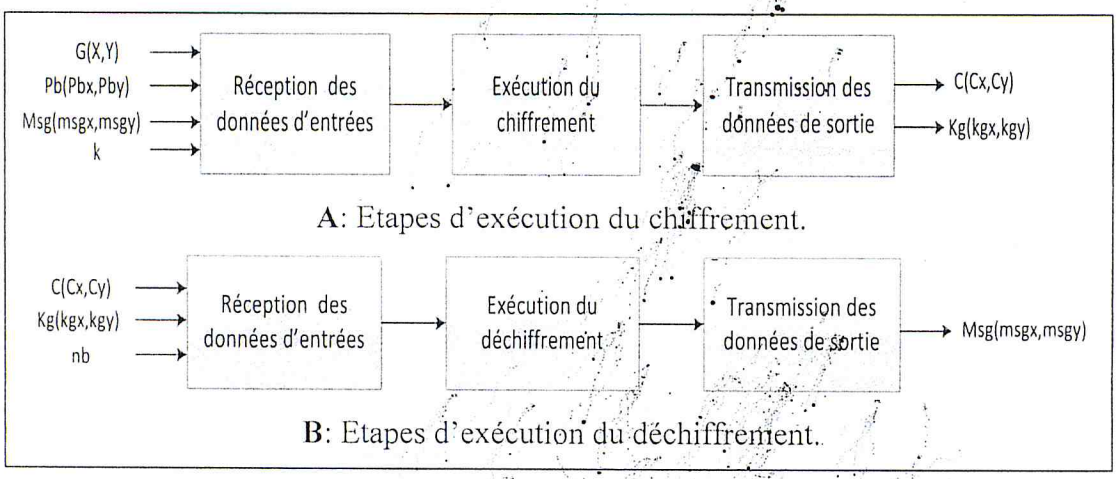

Figure 4.4 : Etapes d'exécution du chiffrement/déchiffrement.

4.4.1.1. Réception des données d'entrées

Après l'initialisation de toutes les variables intermédiaire à zéro, les données transmises par l'IHM sont stockées dans la mémoire locale du processeur sur des chiffres de taille 8 bits. La réception des données est basée sur la fonction $inValeur32Bit$ (). Son code C est représenté dans la figure 4.5.

```
inline unsigned long inValeur32Bit(){
      unsigned long a=0,tmp;int i;
      for(i=0;i<=3;i++){
                  tmp=XUartLite_RecvByte(0x40600000)<<(8*i);
                  a|=tmp; }
      return a; }
```

Figure 4.5 : Code C de la fonction $inValeur32Bit()$

En effet, la liaison RS232 entre la carte et l'IHM impose que à chaque trame de donnée transmise par l'ordinateur seulement 8 bits sont actifs. De ce fait, la fonction inValeur32Bit() permet de reconstruire des données codées sur 32 bits, à partir des données d'entrées codées sur 8 bits, son implémentation est basée sur :

- Une fonction assurant la transmission vers le processeur Microblaze des 8 bits reçus à travers l'UART. La fonction en question est fournie par Xilinx dans EDK [38]. Cette dernière est définie par : XUartLite_RecvByte().
- Des décalages à gauche de 8 bits, suivant un indice i.

4.4.1.2. Exécution des fonctions du chiffrement/déchiffrement

Ces fonctions sont basées sur la multiplication scalaire et l'addition de deux points. Selon une entrée de sélection, le processeur Microblaze exécute l'une des opérations de chiffrement ou de déchiffrement. Cette entrée est notée par b dans le programme principal $main(.)$ (Voir figure 1 de l'annexe).

a. **Chiffrement** : la fonction $Chiffrement(.)$ permet d'exécuter un chiffrement sur les cordonnées $(msgx,mesgy)$ du message M. L'exécution de cette opération correspond au calcul de l'opération (1.5). Le résultat de cette dernière est un couple de point $P_c = [(kG, (P_m + kP_b)]$ tel que $kG = (kgx, kgy)$ et $(P_m + kP_b) = (cx, cy)$. La fonction $Chiffrement(.)$ comprend les fonctions $mul\_scal(.)$, $LmdA(.)$, $CalculerX(.)$ et $CalculerY(.)$. La fonction $mul\_scal(.)$ permet de calculer les deux multiplications scalaires $k \times G$ et $k \times P_b$ de l'équation (1.5). Elle prend en entrée les cordonnées (X, Y) du point G, les cordonnées(Pbx, Pby) du point P_b qui représente la clé publique, le scalaire k et les cordonnées $(msgx, msgy)$ du message M. La figure 4.6 représente l'organigramme de l'opération du chiffrement.

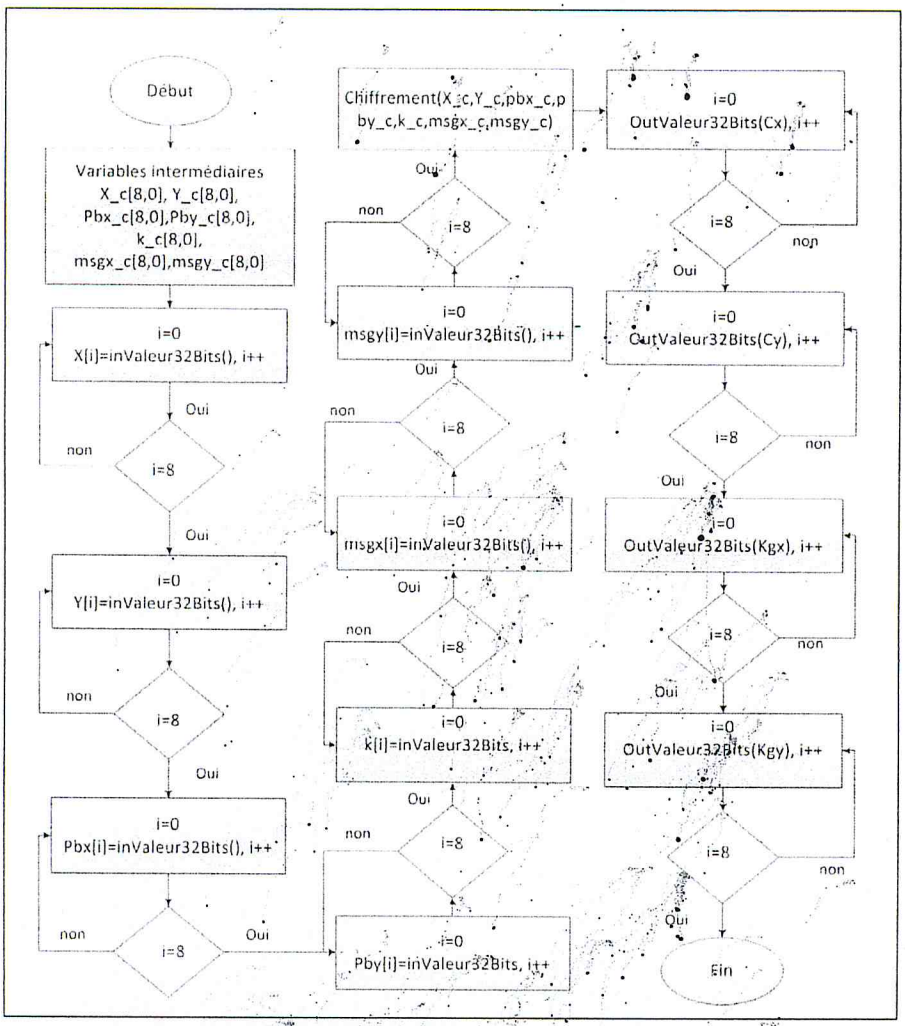

Figure 4.6 : L'organigramme de l'opération du chiffrement

b. Déchiffrement : La fonction $Dechiffrement(.)$ permet de restituer les cordonnées $(msgx, msgy)$ du message M en clair à partir des cordonnées (cx, cy) du message chiffré. Elle reçoit en entrée (cx, cy) , (kgx, kgy) et la clé privée nb. La fonction $Dechiffrement(.)$ comprend les fonctions $mul\_scal(.), LmdA(.)$, $CalculerX(.)$ et $CalculerY(.)$. La fonction $mul\_scal(.)$ permet de calculer la multiplication scalaire $nb \times (k \times G)$ de l'équation (1.6). Les fonctions $LmdA(.)$, $CalculerX(.)$ et $CalculerY(.)$ permettent de calculer l'addition $(P_m + k.P_b) - n_b(k.G)$ de l'équation (1.6). La figure 4.7 représente l'organigramme de l'opération du déchiffrement.

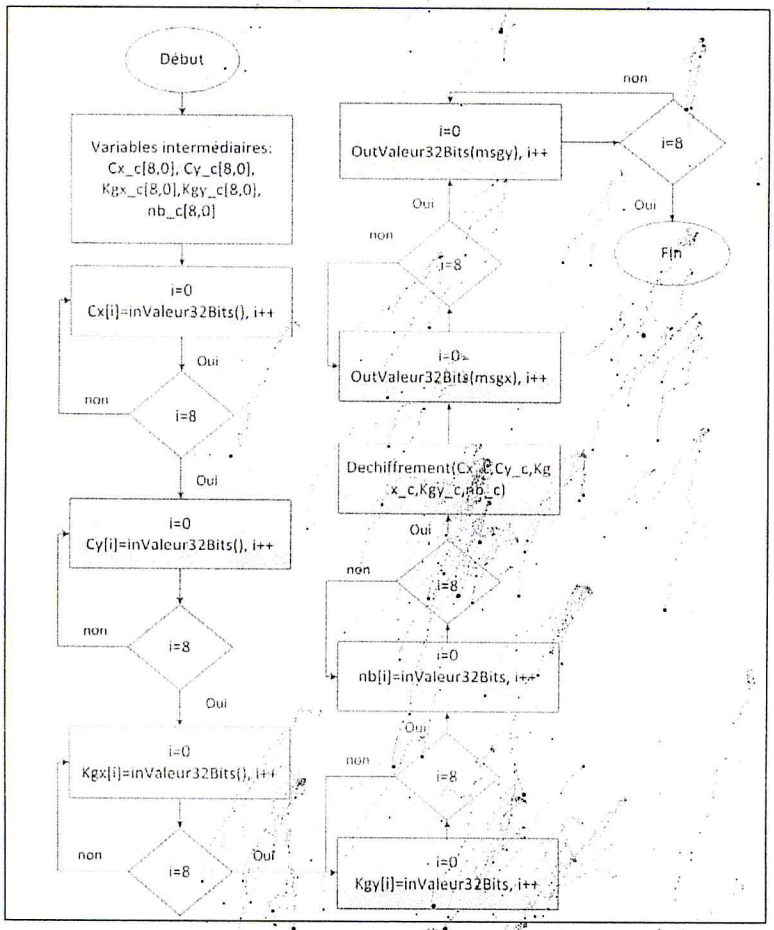

Figure 4.7 : Organigramme de l'opération du déchiffrement.

4.4.1.3. Transmission du résultat

Durant cette étape, le résultat du chiffrement ou du déchiffrement est transmit vers l'IHM. Elle est basée sur la fonction $OutValeur32bits(.)$. Celle-ci assure la décomposision du résultat sur des mots de taille 8 bits. Le code C de cette fonction est donné sur la figure 4.8.

```
inline void outValeur32Bit(unsigned long valeur){
      int i;
   for(i=0;i<=3;i++)
         XUartLite_SendByte(0x40600000,valeur>>(8*i));}
```

Figure 4.8: Code C de la fonction $outValeur32Bit(.)$

L'implémentation de cette dernière est basée sur :

- Des décalages à droite, suivant l'indice j.
- La fonction $XUartL.ite\_SendByte()$, fournie par Xilinx[38].

4.4.2. Fonction de la multiplication scalaire *mul_scal(.)*

La fonction $mul\_scal(.)$ est utilisée dans le chiffrement ainsi que dans le déchiffrement. Elle permet de calculer l'opération de base $Q = k \times P$, tel que P est un point. k est un scalaire. Cette fonction est exécutée par Microblaze selon les étapes de l'algorithme 2.6, à savoir :

- Décalage à droite du $i^{éme}$ chiffre du scalaire k.
- Tester le $j^{éme}$ bit de $k[i]$ selon la ligne 7 de l'algorithme 2.7.
- Effectuer l'opération d'addition et de découblement de points..
- Stocker le résultat dans la variable de sortie Q..

Le code C de la fonction $mul\_scal(.)$ est montré sur la figure 2 de l'annexe et son organigramme est montré sur la figure 4.9.

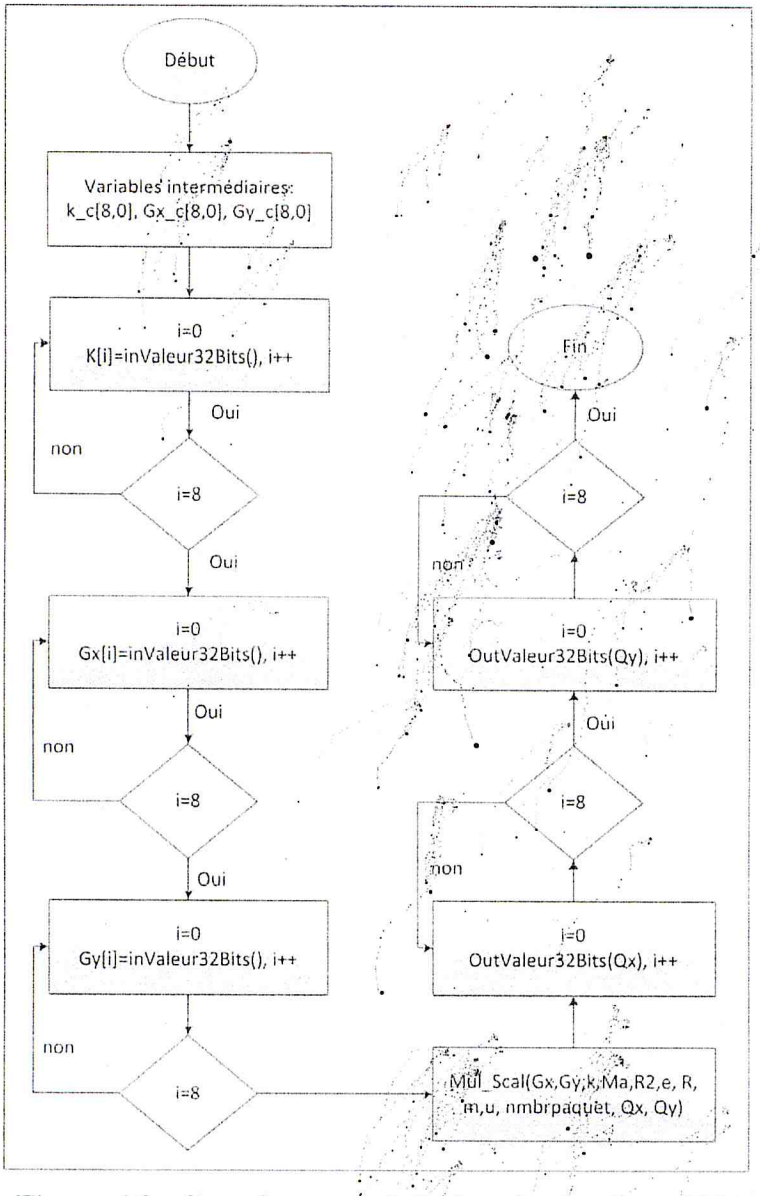

Figure 4.9 : Organigramme de la fonction $mul\_scal(.)$

$mul\_scal(.)$ est constituée principalement par les fonctions: $LamdA(.)$, $LmdD(.)$, $CalculerX(.)$ et $CalculerY(.)$. Ces dernières assurent le calcul d'une addition et d'un dédoublement de points. En plus à ces fonctions, le code C de $mul\_scal(.)$ comprend aussi une fonction qui permet d'effectuer les décalages du scalaire k. Celle-ci est nommée $getBitIndex1(.)$.

- $\boldsymbol{LmdA}(.)$ correspond à l'implémentation de l'équation (1.9). Elle est constituée de deux soustractions modulaire, une exponentiation modulaire qui n'est rien d'autre qu'une transformation de l'inverse modulaire selon le petit théorème de Fermat et d'une multiplication de Montgomery. Ces opérations sont calculées respectivement par

les fonctions $sub\_mod(.)$, $expBinnary(.)$ et $montgomery(.)$. Le code C de la fonction $LmdA(.)$ est montré sur la figure 3 de l'annexe et son organisation est montré sur la figure 4.10.

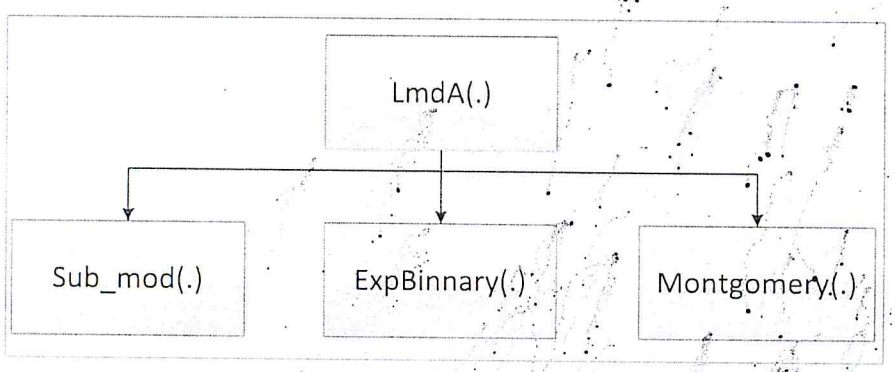

Figure 4.10 : Organisation de la fonction $LmdA(.)$

- $\boldsymbol{LmdD}(.)$ ccorrespond à l'implémentation de l'équation (1.12). Elle comprend trois additions modulaires, une exponentiation modulaire et deux multiplications de Montgomery. Ces opérations sont calculées respectivement par les fonctions $add\_mod(.)$, $expBinnary(.)$ et $montgomery(.)$. Le code C de la fonction $LmdD(.)$ est présenté sur la figure 4 de l'annexe et son organisation est montré sur la figure 4.11.

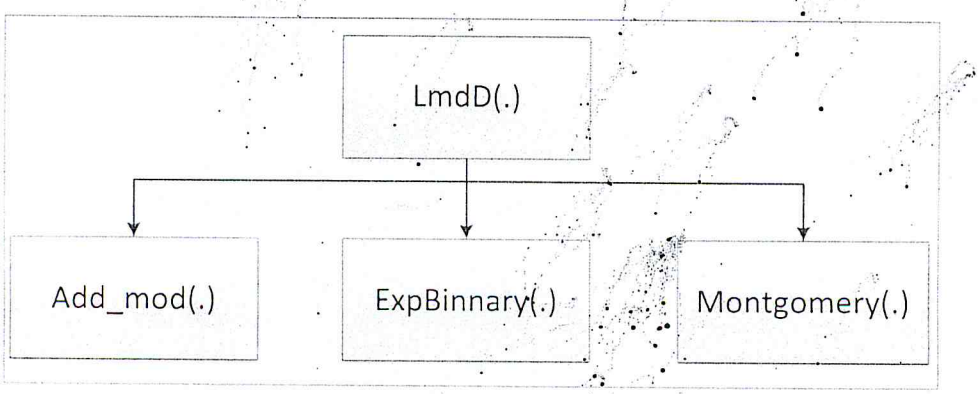

Figure 4.11. : Organisation de la fonction $LmdD(.)$

- $\boldsymbol{CalculerX}(.)$ permet le calcul de l'opération donnée par l'expression (1.7). Cette fonction est constituée d'une soustraction modulaire effectuée par la fonction $sub\_mod(.)$, une addition modulaire effectuée par la fonction $add\_mod(.)$ ainsi qu'une multiplication modulaire exécutée par la fonction $montgomery(.)$. Le code C de la fonction $CalculerX(.)$ est donné sur la figure 5 de l'annexe et son organisation est donné sur 4.12.

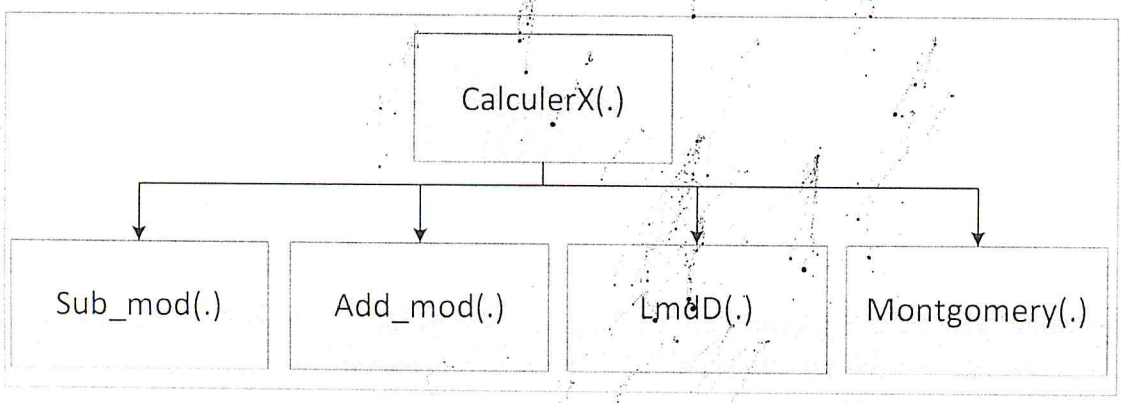

Figure 4.12 : Organisation de la fonction $CalculerX(.)$

- $CalculerY(.)$ permet le calcul de l'opération donnée par l'expression (I.8). Cette fonction est constituée de deux soustractions modulaire effectuées par la fonction $sub\_mod(.)$ et une multiplication modulaire, exécutée par la fonction $montgomery(.)$. Le code C de la fonction $CalculerY()$ est présenté sur la figure 6 de l'annexe et son organisation est montré sur la figure 4.13.

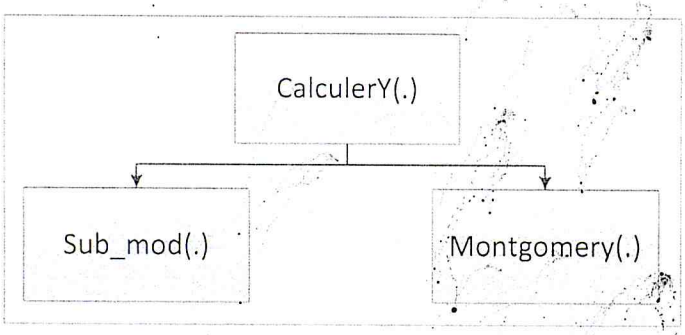

Figure 4.13 : Organisation de la fonction $CalculerY(.)$

- $GetBitIndex1(.)$ assure les décalages des i^{eme} chiffres du scalaire k. Le code C de cette fonction est montré sur la figure 4.14.

```
Xuint32 getBitIndex1(Xuint32 num, int index){
Xuint32 bit=(num>>index)&1;
return bit;
}
```

Figure 4.14. :Le Code C de la fonction GetBitIndex1(.)

4.4.3. Fonctions d'exécution des opérations arithmétiques

Dans la partie précédente, nous avons présenté l'implémentation en langage C de l'opération cœur du chiffrement et du déchiffrement, à savoir la multiplication scalaire. Dans ce qui suit, nous allons présenter les fonctions permettant l'exécution des

opérations arithmétiques du crypto système ECC ; en l'occurrence, $expBinary(.)$, $montgomery(.)$, $add\_mod(.)$ et $sub\_mod(.)$.

- ***expBinary***$(.)$ correspond à l'implémentation en langage C de l'algorithme 2.6. Elle permet de calculer l'exponentiation modulaire $C = X^Y \ mod \ N$. Cette fonction est constituée par deux fonctions en l'occurrence, $getBitIndex(.)$ et $mongomery(.)$. la première permet d'effectuer des décalages à droite sur l'exposant Y. la deuxième assure l'exécution itérative de la MMM de manière logicielle. Le code C de la fonction $expBinary(.)$ est défini sur la figure 7 de l'annexe et son organisation est montré sur 4.15.

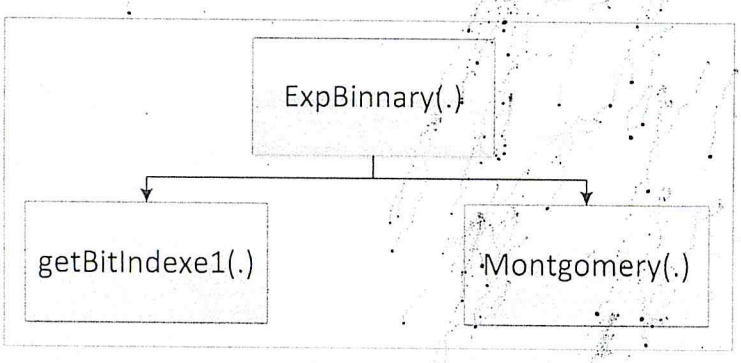

Figure 4.15: Organisation de la fonction $expBinnary(.)$

- ***montgomery***$(.)$ permet d'effectuer la multiplication modulaire de deux entier A et B. Elle correspond à la description en langage C de l'algorithme 2.7. Cette dernière comprend les fonctions $multiply(.)$ et $add3Op(.)$. Le code C de la fonction $montgomery(.)$ est définit sur la figure 4.16.

```
void montgomery(unsigned long A[9],unsigned long B[9],unsigned long
N[9],unsigned long m,unsigned long Resultat[9]) {
int Z=0,P=0,C1=0,C2=0,L=0,i,j,ii;
      int q=0,retenue1=0,retenue2=0,tmp=0;
      for(i=0;i<=8;i++)S[i]=0;
for(i=0;i<=8;i++){// debut boucle i
    C1=0;C2=0;retenue1=0;retenue2=0;tmp=0;//initialisation des
retenu dans chaque iteration  P=A[i]*B[0];
  Z=P+S[0];
  q=(Z*m);
    //*****calcul de S=A[i]*B +q*N /R
            for(j=0;j<=8;j++)
      {// debut boucle j
              tmp=C1; // save la valeur de C1
        multiply(A[i],B[j],&C1,&P); //calcul P=A[i]*B[j]de 32 bits
et C1=retenu de 32bit
          add3Op(P,S[j],tmp,&retenue1,&Z);
        tmp=C2;// save la valeur de C2
              multiply(q,N[j],&C2,&P);
        add3Op(Z,P,tmp,&retenue2,&P);
            if(j!=0)S[j-1]=P;}// fin boucle j
        S[j-1]=C1+C2+retenue1+retenue2;}
for(i=0;i<=8;i++)Resultat[i]=S[i];}
```

Figure 4.16: Code C de la fonction $montgomery(.)$

- $add\_mod(.)$ permet de calculer l'addition modulaire des deux entiers A et B. elle correspond à l'implémentation en langage C de l'algorithme (2.1). Cette dernière comprend les sous fonctions élémentaires $add2Op(.), sous2Op(.)$ et $getBitIndex1(.)$. Le code source de la fonction $add\_mod(.)$ est présenté sur la figure figure 4.17.

```
void add_mod(Xuint32 A[9],Xuint32 B[9],Xuint32
N[9],Xuint32 *SZ[9]){
      Xuint32 SZ1[9],SZ2[9];Xuint32 i,C1=0,C2=1,sign;
      for(i=0;i<9;i++){
            add2Op(A[i],B[i],&C1,&SZ1[i]);
            sous2Op(SZ1[i],N[i],&C2,&SZ2[i]);}
      sign=getBitIndex1((C1+C2+3),1);
      if(sign==1)setIN(SZ1,SZ);
      else setIN(SZ2,SZ);}
```

Figure 4.17 : Code C de la fonction add_mod(.)

- $sub\_mod(.)$ permet d'effectuer la soustraction modulaire des deux entiers A et B. Cette dernière comprend les sous fonctions élémentaires $sous2Op(.)$, $add2Op(.)$ et $getBitIndex1()$. Le code source de la fonction $sub\_mod(.)$ est montré sur la figure figure 4.18.

```
void sub_mod(Xuint32 A[9],Xuint32 B[9],Xuint32
N[9],Xuint32 *SZ[9]){
      Xuint32 SZ1[9],SZ2[9];
      Xuint32 i,C1=1,C2=0,sign;
      for(i=0;i<9;i++){
            sous2Op(A[i],B[i],&C1,&SZ1[i]);
            add2Op(SZ1[i],N[i],&C2,&SZ2[i]);
            }
      sign =getBitIndex1((C1+3),0);
      if(sign==1)setIN(SZ2,SZ);
      else setIN(SZ1,SZ); }
```

Figure 4.18 : Code C de la fonction *sub_mod*(.)

4.5. Implémentation basée sur la combinaison Logiciellee/Materielle

L'objectif principal de cette approche consiste en l'optimisation des performances temporelle de l'exécution de la multiplication scalaire $k \times p$. En effet, en analysant sa complexité algorithmique, nous constatons que l'exponentiation modulaire et la multiplication modulaire sont des opérations critiques. Autrement dit, elles ralentissent l'exécution de l'algorithme 2.7. Pour ce faire le partitionnement logicielle/matérielle proposé dans cette approche est d'implémenter dans un composant matériel nommé IP_EXP_MMM. Ce dernier est dédié au calcul de la multiplication modulaire de Montgomery et de l'exponentiation modulaire. Le contrôle de l'algorithme de la multiplication scalaire est exécuté de manière logicielle par Microblaze. Les taches attribuées a ce dernier sont :

- Décalage à droite du i^{eme} chiffre du scalaire k.
- Tester la valeur du j^{eme} bit de k[i] selon la ligne 7 de l'algorithme 2.7.
- Exécution des opérations d'addition et de soustraction modulaires.
- Transmission des données nécessaires au calcul de la multiplication modulaire et de l'exponentiation modulaire.

L'implémentation de la multiplication modulaire et de l'exponentiation modulaire sur matériel nécessite la conception de leurs architectures et une interface matérielle permettant la communication entre le composant et le processeur. En effet Xilinx fournit dans ce système une interface avec ses pilotes logicielle nommée IPIF(IP Interface). Celle-ci est constituée d'un ensemble de composant qui nous permet d'avoir l'accès à un autre IP à partir de la partie logicielle. Les composants en question sont des registres et des mémoires FIFO.

L'architecture de la partie matérielle du composant IP_EXP/MMM est montrée sur la figure 4.19. Celle-ci est constituée de :Interface IPIF et d'un module nommé EXP_MMM.

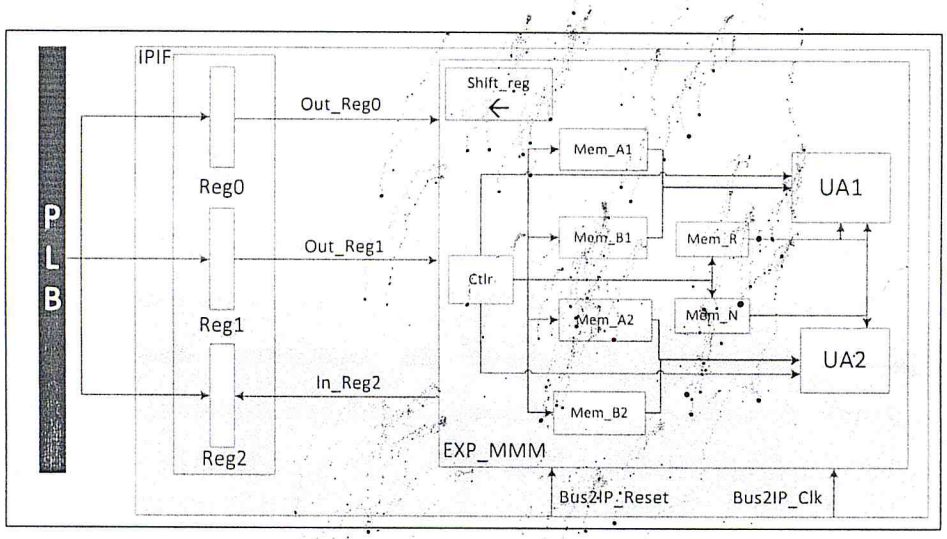

Figure 4.19 : Architecture du composant IP_EXP_MMM

L'interface IPIF permet de décoder le protocole du bus PLB. Le module EXP_MMM assure l'exécution des opérations de la multiplication modulaire et de l'exponentiation modulaire $C = X^Y \ mod \ N$.

L'interface IPIF est constituée de 3 registres, Reg0, Reg1 et Reg2. Le premier est utilisé par le processeur Microblaze pour transmettre les instructions de contrôle. Le second permet de transférer des données vers l'architecture interne du module EXP_MMM. Le troisième assure la transmission du résultat de l'opération effectuée.

EXP_MMM comporte les composants suivant :

- Six mémoires Mem_N, Mem_R, Mem_A1, Mem_A2, Mem_B1 et Mem_B2. Celles-ci sont utilisées pour stocker le modulo, la constante R, les opérandes des algorithmes de l'exponentiation modulaire ou de la multiplication modulaire.
- Un registre à décalage shifft_reg. Ce registre a pour rôle de stocker la valeur de l'exposant et d'effectuer des décalages à gauche, selon l'algorithme 2.6 de l'exponentiation modulaire.
- Deux unités arithmétiques UA1 et UA2. Ces unités permettent de calculer deux MMM en parallèle. Le choix d'implémenter deux unités en parallèle est

justifié par l'exploitation du parallélisme qui caractérise l'exécution de l'algorithme 2.7 .L'unité arithmétique utilisée dans notre travail est détaillée dans [19].

- Un registre pour stocker la valeur de la constante de Mongomery Reg_m.
- Un circuit de contrôle cltr. Son role est de générer :
  - Les adresse mémoires qui correspondent aux phases d'écriture et de lecture. Des mémoires intégrées dans le module Exp_MMM.
  - Les signaux de contrôles des mémoires et des registres.
  - Les signaux de contrôle des deux unités arithmétiques.

Pour controler les étapes d'éxecution des algorihtmes de la MMM et de l'exponentiation modualaire, le processeur Microblaze utilise des instructions. Les codes hexadécimales de ces instructions sont représentées dans le tableau 4.1

Tableau 4.1 : Formats des instructions du composant IP_EXP_MMM.

Instruction	Code en hexadécimale	Description
Reset_Ip	0x80000003	Intialisation du module EXP_MMM.
Run_writeDataCts	0x00000001	Transfert du modulo, la valeur de R et de la constante de Mongomery.
Run_writeA_B	0x00000002	Transfert de quatres opérande pour l'éxécution des deux MMM en parallèle.
Run_MMM	0x00000007	Exécution parallèle des deux MMM.
Run_write_exp	0x4001003	Transfert de l'exposant Z
Run_writeX	0x40000002	Transfert de la valeur de X
Run_exp	0x40040003	Exécution de l'exponentiation modulaire

L'accès au registre configuré dans l'interface IPIF est effectué en utilisant des fonctions ecrites en langage C. Ces fonctions sont fournies par Xilinx. Elles sont définies dans le tableau 4.2

Tableau 4.2 : les fonctions de communication avec le composant IP_EXP_MMM.

Fonction langage C	Description
IP_EXP_MMM_mWriteSlvRegi(baseaddr,0,data)	Ecriture de la donnée « data » dans le registre d'indice i.
IP_EXP_MMM_mReadSlvReg2(baseaddr,0)	Lecture d'une donnée à partir du registre 2.

4.6. Présentation de l'Interface Homme/Machine IHM

Cette interface a été développée non seulement pour permettre une utilisation de la plateforme du chiffrement et du déchiffrement, mais aussi pour la vérification de la partie embarquée sur circuit FPGA. L'IHM a été développée sous java, en utilisant l'environnement de développement NetBeans. Les taches attribuées à cette interface se résument dans :

- La configuration du protocole RS232.
- Génération des clés du chiffrement et du déchiffrement.
- La conversion du message clair en un point.
- Transmission des données nécessaires pour le chiffrement/déchiffrement vers la carte FPGA.
- Réception des résultats du chiffrement/déchiffrement de la carte FPGA et leurs affichages.
- La conversion du point résultant du déchiffrement en un message clair.

La page d'accueil de l'IHM est représentée sur la figure 4.20.

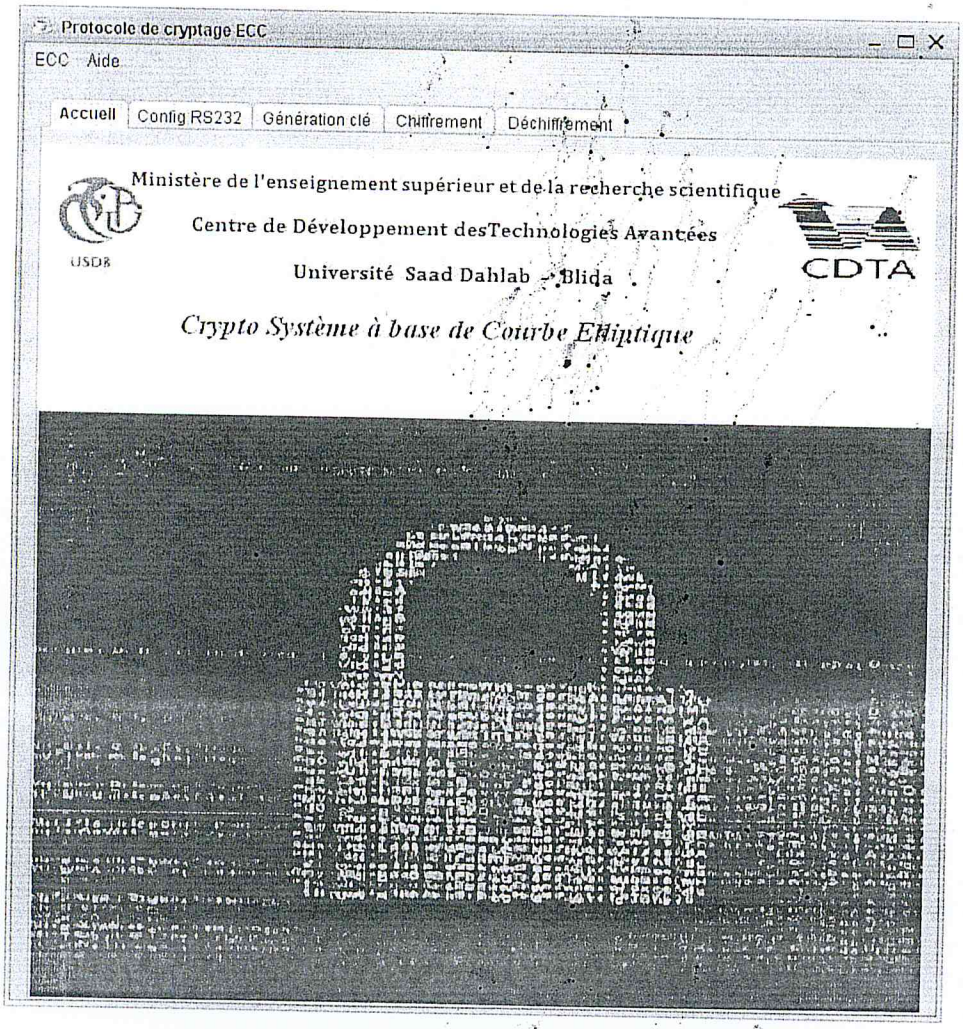

Figure 4.20 : Page d'accueil de l'IHM

4.6.1. Configuration du protocole de communication RS232

Cette partie est nécessaire pour l'établissement de la liaison de communication entre l'ordinateur et la carte de prototypage Genesys. Cette configuration est basée sur la définition des paramètres suivants :

- Débit de transmission des données.
- La taille des données à transmettre.
- Le bit de parité.
- Le bit d'arrêt.
- Le control de flux.

Ces paramètres doivent être configurés de la même manière que la configuration de l'UART du système embarqué. La configuration des paramètres du protocole de communication RS232 est montrée dans la figure 4.21

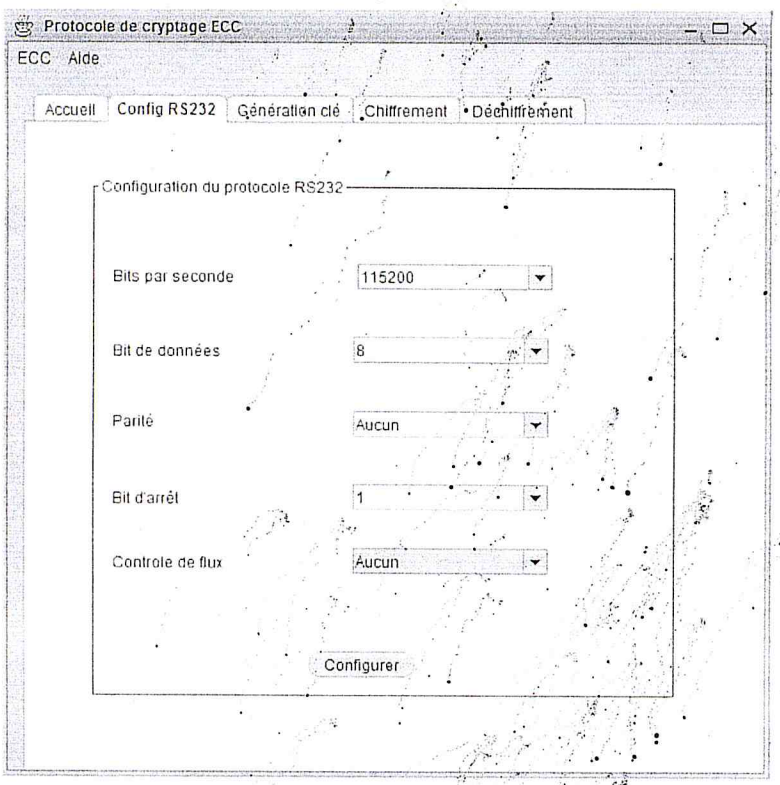

Figure 4.21 : Configuration du port RS232

4.6.2. Génération des clés du chiffrement et du déchiffrement

Cette partie est consacrée pour :

- La génération de la clé publique *Pb* et de la clé privée *nb*.
- L'enregistrement dans la mémoire interne de l'ordinateur des valeurs en hexadécimales des deux clés dans des fichiers .txt.

La partie de l'IHM qui permet la génération des clés est montrée sur la figure 4.22.

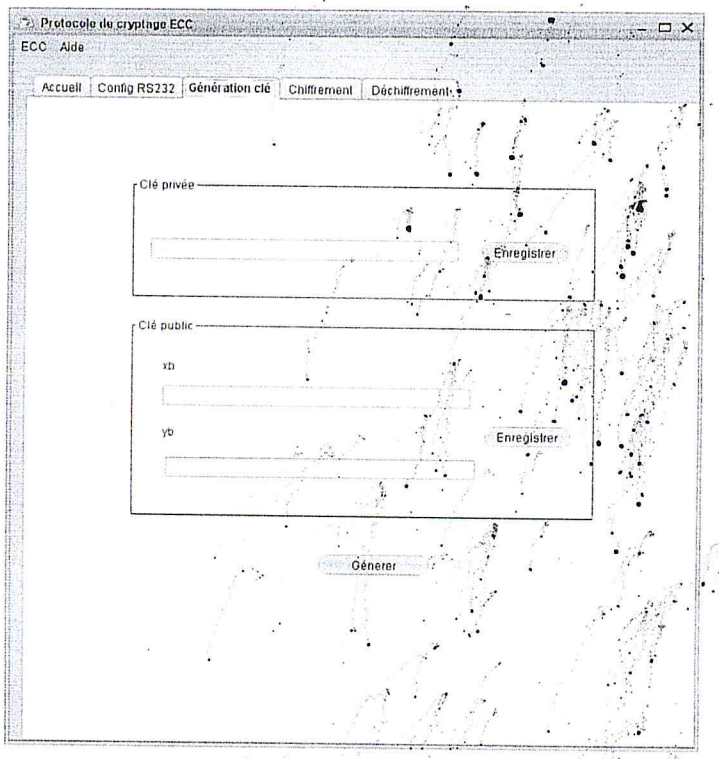

Figure 4.22 : Génération des clés

La génération des clés est utilisée selon le protocole du crypto système ECC. La taille est choisie suivant le standard sec256 [22] sur 256 bits. Le programme Java de cette partie est basé sur la fonction *Génération*(.). Celle-ci permet de générer la clé privée nb, telle que $nb < n$, où n est l'ordre du point générateur G. La clé public Pb est calculée par la multiplication scalaire : $P_b = nb \times G$. La génération de la clé privée est basée sur l'instance de la class Random qui permet de générer un nombre aléatoire. Le résultat est de type BigInteger muni d'un paramètre indiquant la taille exacte du résultat. Le programme java de cette partie est montré sur la figure 4.23

```
public  BigInteger getRandomBigInteger(){
  Random rand=new Random();
  BigInteger result= new BigInteger(256, rand);
  return result; }
Cle Generation(){
   BigInteger nb=getRandomBigInteger();
    while(nb.max(n)==nb){
   nb=getRandomBigInteger();}
    ReglesAddition rr=new ReglesAddition();
    Point p;
    p=rr.multiScalairee(G, a, nb, modulo);
   Cle c=new Cle(nb,p);
 return c; }
```

Figure 4.23 : Code java de la génération des clés

4.6.3. Conversion du message clair en un point

C'est l'étape qui précède la transmission des données vers le circuit FPGA avant le chiffrement. Elle est effectuée en deux étapes essentielles, la représentation du message en sa forme ASCII, est stocker le résultat dans un BigInteger. Ceci est basé sur la fonction $msgtobig(.)$. Et la génération du point représentant le message. Cette génération prend comme paramètre le résultat de la fonction $msgtobig(.)$, puis le multiplier par un facteur k. Cette opération fournit l'abscisse X du message. L'ordonnée Y est obtenue en utilisant les paramètres de la courbe (a, b, N) et l'abscisse X. Ceci est basé sur la fonction $MtoP(.)$ qui fait appel à la fonction $GetPoint(.)$. Le code java de cette partie est montré dans la figure 4.24.

```
public BigInteger MsgToBig(String message) throws
UnsupportedEncodingException {
                return new BigInteger(message.getBytes("us-ascii"));
        }
                public BigInteger GetPoint(BigInteger x){
                BigInteger
y2=(x.pow(3).add(a.multiply(x)).add(b)).mod(p);

                return y2; }
public Point MtoP(BigInteger msg,BigInteger k){
                BigInteger j=BigInteger.ZERO;
                BigInteger msgx=BigInteger.ZERO,
                   Zi=BigInteger.ZERO;
                while(j.min(k)!=k) {
                        msgx=(msg.multiply(k)).add(j);
                        Zi=GetPoint(msgx);

                        if(IsSqrtRoot(Zi))
                                j=k;
                        else{
                                j=j.add(BigInteger.ONE); }}
                                 msgy= sqrt(Zi)
                return new Point(msgx,msgy); }
```

Figure 4.24 : Code java de conversion d'un message en point

4.6.4. Transmission vers la carte FPGA des données nécessaires pour le chiffrement/déchiffrement

La première donnée à envoyer vers la carte FPGA est l'entrée de sélection. Selon cette dernière, la partie embarquée sur circuit FPGA est configurée soit pour effectuer un chiffrement ou un déchiffrement.

Après la génération du point représentant le message, l'étape de la transmission des données nécessaires au chiffrement est exécutée. Les données en question sont :

La clé public Pb, le point du message Pm $(Msgx, Msgy)$, le point générateur de la courbe $G(X, Y)$ et un bigInteger k généré aléatoirement. La fonction qui se charge de la transmission des données vers le circuit FPGA est la fonction outValeurbig(.). Son code java est montré sur la figure 4.25.

Pour déchiffrer les données, nous procédons par la sélection de la clé privée et du résultat du chiffrement à partir de l'IHM. Ces derniers seront transmis vers le circuit FPGA. Les données en question sont constituées de $nb, (cx, cy), (kgx, kgy)$, où nb est la clé privée, (cx, cy) sont les cordonnées du message chiffré. (kgx, kgy) représentent les cordonnés du point résultant de la multiplication scalaire $k \times G$.

```
public void outValeurbig(BigInteger valeur,int taille) {
                BigInteger big=(new
BigInteger("2",10)).pow(taille+1);
               BigInteger v=valeur.add(big);
               byte[] valeurS=v.toByteArray();
               byte b=0;
               int t=taille/8;
               int a,c;
               if(valeurS.length==t+1){a=t;c=1;}
               else {a=t-1;c=0;}
               try {
                   (valeur>>(8*i));
                      for(int i=a;i>=c;i--){
                           if(i<valeurS.length)
                           out.write(valeurS[i]);
                           else
                           out.write(b); }
                     out.flush();
                                } catch (IOException e) {
                   e.printStackTrace();      }}
```

Figure 4.25 : Code java de la transmission des données de l'ordinateur vers la carte FPGA

La réception des données vers l'IHM est basée sur la fonction inValeur32bits(). Son code java est montré sur la figure 4.26.

```
public int inValeur32Bit(){
              int valeur=0;//byte b;
               for(int i=0;i<=3;i++)
       try {valeur |=in.read()<<(i*8);}
        catch (IOException e) {
       e.printStackTrace();}
          return valeur;}
```

Figure 4.26 : Code java de la réception des données de la carte FPGA vers l'ordinateur

4.6.5. Conversion du point résultant du déchiffrement en un message clair

Le résultat du déchiffrement est transmis de la carte, sous forme d'un point $Pm(Msgx, Msgy)$. Sa conversion en un message en clair est nécessaire pour qu'il soit visualisé.

L'abscisse $Msgx$ du point est divisée par le facteur mentionné dans le paragraphe 4.6.3 puis convertie en un message, par la fonction $BigToMsg(.)$. Le code java de cette partie est montré sur la figure 4.27.

```
public static String BigToMsg(BigInteger big){
return new String(big.toByteArray());
}
```

Figure 4.27 : Code java de la conversion d'un point en message.

Les interfaces du chiffrement et du déchiffrement sont montrées respectivement sur les figures 4.28, 4.29

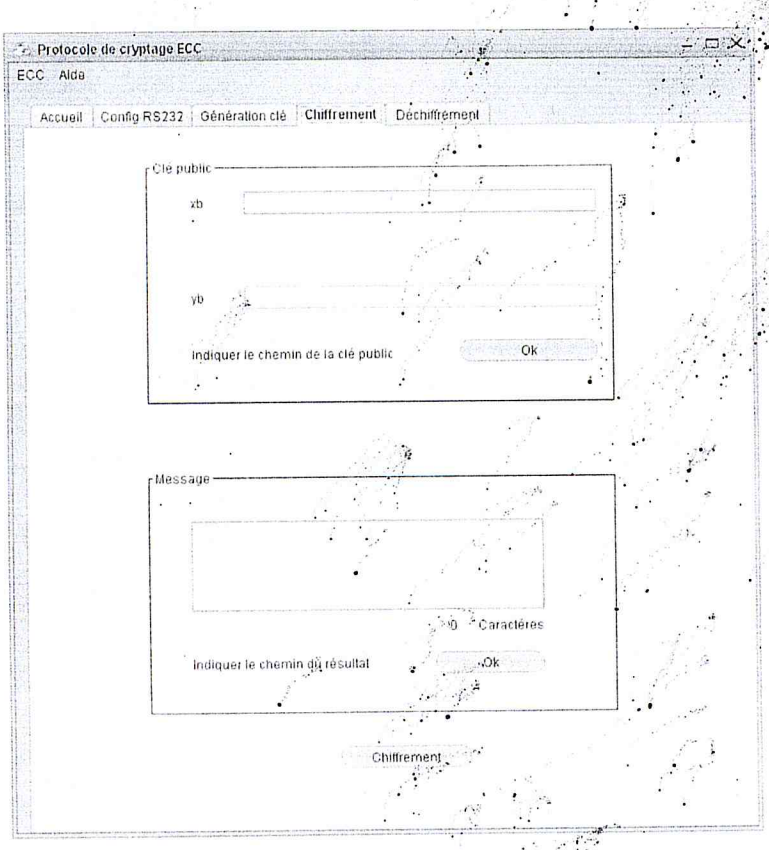

Figure 4.28 : Chiffrement d'un message.

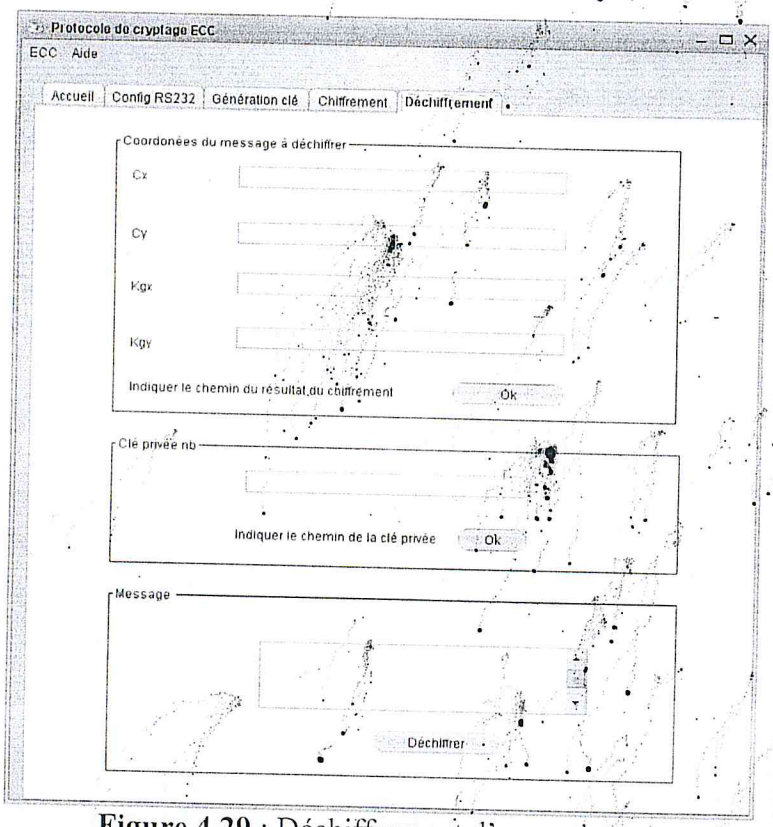

Figure 4.29 : Déchiffrement d'un message.

4.7. Conclusion

Dans ce chapitre nous avons présenté la plateforme du crypto système ECC implémentée. Nous avons ensuite présenté les deux approches proposées pour l'implémentation de ce crypto système. Nous avons terminé le chapitre par la présentation de l'Interface Homme/Machine qui est exécutée sur ordinateur. Le chapitre suivant sera consacré à la présentation des résultats d'implémentation des deux approches proposées pour la réalisation du crypto système ECC.

Chapitre 5

Résultats d'implémentation

5.1. Introduction

Dans notre travail, nous avons proposé deux approches d'implémentation de la multiplication scalaire, dans un environnement PSoC, afin de réaliser un crypto-système ECC. La première est une implémentation purement logicielle. La seconde est basée sur une conception conjointe des deux ressources : logicielle/matérielle, où les opérations de multiplications et de l'exponentiations modulaires sont implémentées sur matériel. Le control de l'algorithme de la multiplication scalaire est exécuté de manière logicielle par le processeur Microblaze. Dans le but de réaliser un crypto-système ECC qui présente le meilleur compromis temps d'exécution/ressources matérielles occupées par le système/flexibilité, notre analyse sera portée sur les performances temporelles et les ressources matérielles requises pour l'implémentation de la partie embarquée du système.

Ce chapitre est consacré à la présentation des résultats d'implémentation, des approches proposées pour la réalisation du crypto-système ECC. Nous allons entamer par décrire la méthodologie élaborée pour le développement de ce crypto-système. Ensuite, nous présenterons les performances issues de l'implémentation des deux méthodes réalisées.

5.2. Méthodologie de conception

Dans ce travail, la méthodologie élaborée pour concevoir les approches d'implémentation proposées est répartie en cinq étapes :

1. Choix des paramètres d'une courbe elliptique.
2. Spécification de la partie matérielle du crypto-système embarqué.
3. Développement de la partie logicielle en langage C.
4. Réalisation des parties matérielle et logicielle du composant IP_EXP_MMM.
5. Développement de l'interface homme/machine.
6. Génération du fichier de configuration du circuit FPGA et implémentation.

Cette méthodologie a été développée en utilisant les outils logiciels de Xilinx :

- EDK pour la spécification du système embarqué et le développement de sa partie logicielle.
- ModelSim pour la vérification fonctionnelle du composant IP_EXP_MMM.

Les valeurs des paramètres N, a, b, G(X, Y) et n de la courbe elliptique utilisée sont choisis du standard secp256r1 pour une sécurité de 256 bits [22]. Elles sont définies comme suit :

N=0xFFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFFFFFFFFFFFF

a=0xFFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFFFFFFFFFFFC

b=0x5AC635D8AA3A93E7B3EBBD55769886BC651D06B0CC53B0F63BCE3C3E27D2604B

X=0x6B17D1F2E12C4247F8BCE6E563A440F277037D812DEB33A0F4A13945D898C296

Y=0x4FE342E2FE1A7F9B8EE7EB4A7C0F9E162BCE33576B315ECECBB6406837BF51F5

n=0xFFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551

5.3. Performances temporelles et ressources matérielles occupées

L'étude des performances des approches proposées pour la réalisation du système embarqué est basée sur l'analyse des performances en termes de temps d'exécution et des ressources matérielles occupées.

5.3.1. Performances temporelles

Pour déterminer le nombre de tops d'horloge nécessaire à l'exécution de la multiplication modulaire, de l'exponentiation modulaire, la multiplication scalaire et des opérations du chiffrement et de déchiffrement de chaque approche d'implémentation, les fonctions en langage C du Timer $XTmrCtr\_Start(\&deley, 1)$ et $XTmrCtr\_Stop(\&deley, 1)$ ont été activées respectivement au début et à la fin de la fonction correspondante à l'exécution de l'opération concernée. Le temps d'exécution est calculé par la multiplication du nombre de top d'horloge par la période de l'horloge. Les performances temporelles de ces opérations

Tableau 5.1 Performances temporelles en fonction de l'approche utilisée

Approche	Performance	Multiplication modulaire	Exponentiation modulaire	Multiplication scalaire	Chiffrement	Déchiffrement
Première approche Purement logicielle	f (Mhz)	100	100	100	100	100
	NbTH	13135	5012200	2585749024	5176629533	2590880509
	t(ms)	0.13135	50.12200	25857.49024	51766.29533	25908.80509
Deuxième approche Implémentation logicielle/matérielle	f (Mhz)	100	100	100	100	100
	NbTH	903	41683	23990662	48029977	24040081
	t(ms)	0.00903	0.41683	239.90662	480.29977	240.40081

Dans ce tableau NbTH représente le nombre de tops d'horloge nécessaire à l'exécution des opérations étudiées. A partir de ce tableau, on constate que les performances temporelles de la deuxième approche sont meilleures par rapport aux performances de la première approche d'implémentation. En effet, le délai obtenu avec la deuxième implémentation pour le chiffrement d'une donnée de valeur numérique inférieure à la valeur du modulo, représente 0,93% du délai de la première approche d'implémentation.

5.3.2. Ressources matérielles requises

Les ressources matérielles requises pour l'implémentation du crypto-système ECC, en fonction des deux approches utilisées sont montrés sur le tableau Tableau 5.2. Ces résultats

sont présentés en termes de nombre de slices, de blocs RAM(36-kbits et 18-kbits) et de blocs DSP.

Tableau 5.1 Ressources matérielles requises pour l'implémentation des approches proposées.

Approche	Slices	Slices (%)	Bloc RAM 36-kbits	Bloc RAM 18-kbits	DSP48E Core
Première approche Purement logicielle	1038	14	8	-	3
Deuxième approche Implémentation logicielle/matérielle	2245	31	8	8	25

Ces résultats montrent que le système embarqué de la première approche nécessite 1038 slices, 8 blocs RAM de 36-kbits et 3 DSP48E Core. Ces ressources matérielles peuvent être considérées comme étant faibles car le système embarqué implémenté dans cette approche est un système de base. Comparé à la deuxième approche d'implémentation, où un composant matériel est intégrée autour du processeur Microblaze, cette approche nécessite plus de ressources matérielles, la différence est de 1207 slices, 8 blocs RAM de 18-kbits et 22 DSP48E Core. Ces ressources sont consommées principalement par le composant IP_EXP_MMM.

5.4. Conclusion

Ce chapitre a été consacré à la présentation comparative des résultats, des deux approches implémentées, en termes de délai d'exécution et de ressources matérielles occupées par ces deux dernières. Nous avons montré que la deuxième approche implémentée est plus gourmande en termes de ressources matérielles, en revanche le délai d'exécution du chiffrement et du déchiffrement engendré par cette dernière représente 0.93% du délai engendré par la première approche. D'après ces résultats, nous constatons que l'implémentation de notre système embarqué basé sur une combinaison logicielle/matérielle engendre un meilleur compromis entre les ressources matérielles occupées et le délai d'exécution.

Conclusion générale

Conclusion générale

L'objectif de notre projet consiste en l'implémentation du crypto système ECC embarqué sur une plateforme PSoC à base du processeur Microblaze de Xilinx. L'opération cœur de ce crypto système est la multiplication scalaire. Celle-ci est nécessaire non seulement pour le chiffrement et le déchiffrement, mais aussi pour la génération des clés. Pour ce faire, nous avons entamé notre travail par deux principales études, à savoir, l'étude du protocole de cryptographie ECC et l'étude des systèmes embarqués à base du processeur Microblaze.

En deuxième étape, nous nous sommes focalisées aux opérations arithmétiques du crypto système ECC. Cette étude est primordiale car toute sa complexité de réalisation relève de ses opérations de base.

Après l'étude des algorithmes d'exécution des opérations de base du crypto système ECC, nous avons constaté que ce dernier est basé sur le calcul de la multiplication scalaire. Ses performances sont fortement liées à la rapidité d'exécution de cette opération qui en sa part repose sur la multiplication modulaire et l'exponentiation modulaire. Ces dernières sont considérées comme étant des opérations critiques et gourmandes en termes de ressource et de temps d'exécution.

L'étude détaillée des deux opérations de base de la multiplication scalaire ont mené aux choix des algorithmes les plus adaptés pour notre crypto système, l'algorithme de la multiplication modulaire de Montgomery a été choisi pour l'implémentation de la multiplication modulaire et l'algorithme de Montgomery Power Ladder pour implémenter l'exponentiation modulaire.

Deux approches d'implémentation ont été proposées:

- Une approche d'implémentation purement logicielle, où l'algorithme de la MMM et l'algorithme de l'exponentiation modulaire sont exécutés par le processeur Microblaze.
- Une approche basée sur la combinaison matérielle/logicielle. Cette approche consiste en l'implémentation matérielle de la MMM et de l'exponentiation modulaire. Le contrôle de l'algorithme de la multiplication scalaire, du chiffrement et du déchiffrement sont exécutés de manière logicielle par le

processeur Microblaze. Le but de cette implémentation est d'augmenter les performances temporelles du crypto système ECC.

Après la conception et la réalisation de la partie embarquée sur circuit FPGA, une plateforme du chiffrement et de déchiffrement basé sur une interface Homme/Machine a été implémentée en langage Java. Celle-ci est réalisée afin de rendre le système plus flexible et de vérifier le fonctionnement correcte de nos implémentations.

Les résultats d'implémentation ont montré que la deuxième approche nécessite plus de matérielles, par rapport à la première approche. En revanche le délai d'exécution du chiffrement et du déchiffrement obtenu par cette dernière représente 0.93% du délai qui correspond à la première approche d'implémentation. De ce fait, nous avons constaté que la deuxième approche aboutit à un meilleur compromis entre les ressources matérielles occupées et le délai d'exécution.

En termes de perspective de notre travail, les performances d'exécution du crypto système ECC peuvent encore être améliorées par l'exploitation des architectures parallèles. Ces dernières seront implémentées non pas par l'utilisation d'un seul processeur embarqué mais par l'intégration de plusieurs processeurs à la fois.

Annexe

Annexe

```
int main(){
while (1) {
b=XUartLite_RecvByte(0x40600000);
        if(b==0){/////*chiffrement donnée
        for(i=0;i<8;i++){
              X[i]=inValeur32Bit();}
               for(i=0;i<8;i++){
              Y[i]=inValeur32Bit();}
   for(i=0;i<8;i++){
             Pbx[i]=inValeur32Bit();}
   for(i=0;i<8;i++){
              Pby[i]=inValeur32Bit();}
   for(i=0;i<8;i++){
              K[i]=inValeur32Bit();}
      for(i=0;i<8;i++){
         msgx[i]=inValeur32Bit();}
   for(i=0;i<8;i++) {
              msgy[i]=inValeur32Bit();}
chiffrement(X,Y,K,msgx,msgy,Pbx,Pby,N,A,R2,E,R,m,u,nbrPaquet,kgx,kgy,cx,cy);

   for(i=0;i<8;i++){
              outValeur32Bit(cx[i]);}
   for(i=0;i<8;i++){
              outValeur32Bit(cy[i]); }
   for(i=0;i<8;i++){
              outValeur32Bit(kgx[i]);
              }
   for(i=0;i<8;i++){
              outValeur32Bit(kgy[i]);          }}

else{/////* dechiffrement donnée
for(i=0;i<8;i++){
nb[i]=inValeur32Bit();    }
for(i=0;i<8;i++){
cx1[i]=inValeur32Bit();}
for(i=0;i<8;i++){
cy1[i]=inValeur32Bit();}
for(i=0;i<8;i++){
kgx1[i]=inValeur32Bit();          }
for(i=0;i<8;i++){
kgy1[i]=inValeur32Bit();  }
dechiffrement(kgx1,kgy1,cx1,cy1,nb,N,A,R2,E,R,m,u,nbrPaquet,mx,my);
for(i=0;i<8;i++){
outValeur32Bit(mx[i]); }
for(i=0;i<8;i++){
outValeur32Bit(my[i]);       }
}}
return 0;}
```

Figure 1 : Code C du programme principal Main

```
void mul_scal(Xuint32 X[9],Xuint32 Y[9],Xuint32 K[9],Xuint32
N[9],Xuint32 A[9],Xuint32 R2[9],Xuint32 E[9],Xuint32 R[9],Xuint32
m,Xuint32 u[9], int nbrPaquet,Xuint32 Xq[9],Xuint32 Yq[9]){
Xuint32 x0[9],y0[9],x1[9],y1[9],xx0[9],yy0[9],xx1[9],yy1[9];
int i,j,ii,init=30;
montgomery(X,R2,N,m,x0);
montgomery(Y,R2,N,m,y0);
LmdD(x0,y0,A,N,E,R,m,nbrPaquet,Lmdd);
CalculerX(x0,x0,Lmdd,N,m,x1);
CalculerY(x0,x1,y0,Lmdd,N,m,y1);
for(i=nbrPaquet-1;i>=0;i--){
for(j=init;j>=0;j--)       {
if(getBitIndex1(K[i],j)==0{
LmdD(x0,y0,A,N,E,R,m,nbrPaquet,Lmdd);
CalculerX(x0,x0,Lmdd,N,m,xx0);
CalculerY(x0,xx0,y0,Lmdd,N,m,yy0);
LmdA( x0, y0, x1, y1, N, E, R, m, nbrPaquet,Lmda);
CalculerX(x0,x1,Lmda,N,m,xx1);
CalculerY(x1,xx1,y1,Lmda,N,m,yy1);
for(ii=0;ii<9;ii++){
x0[ii]=xx0[ii]; y0[ii]=yy0[ii]; }
for(ii=0;ii<9;ii++){
x1[ii]=xx1[ii];
y1[ii]=yy1[ii];}}
else{
LmdD(x1,y1,A,N,E,R,m,nbrPaquet,Lmdd);
CalculerX(x1,x1,Lmdd,N,m,xx1);
CalculerY(x1,xx1,y1,Lmdd,N,m,yy1);
LmdA( x0, y0, x1, y1, N, E, R, m, nbrPaquet,Lmda);
CalculerX(x0,x1,Lmda,N,m,xx0);
CalculerY(x1,xx0,y1,Lmda,N,m,yy0);
for(ii=0;ii<9;ii++)  {
x1[ii]=xx1[ii];
y1[ii]=yy1[ii];  }
for(ii=0;ii<9;ii++){
x0[ii]=xx0[ii];
y0[ii]=yy0[ii];}     }}init=31;}
montgomery(x0,u,N,m,Xq); montgomery(y0,u,N,m,Yq); }
```

Figure 2 : Code C de la fonction *mul_scal*(.)

```
Xuint32 i1[9],i2[9],i3[9];
void LmdA(Xuint32 X0[9],Xuint32 Y0[9],Xuint32 X1[9],Xuint32
Y1[9],Xuint32 N[9],Xuint32 E[9],Xuint32 R[9],Xuint32 m,int
nbrPaquet,Xuint32 Lmda){
sub_mod(Y0,Y1,N,i1);
sub_mod(X0,X1,N,i2);
expBinnary(i2,E,N,R,m,i3,nbrPaquet);
montgomery(i3,i1,N,m,Lmda);}
```

Figure 3 : Code C de la fonction *LmdA*(.)

```
Xuint32 S5[9],S6[9],SS1[9],S2[9],S3[9],S4[9];
void LmdD(Xuint32 X[9],Xuint32 Y[9],Xuint32 A[9],Xuint32 M[9],
Xuint32 E[9], Xuint32 R[9],Xuint32 m,int nbrPaquet,Xuint32 Lmdd){
add(Y,Y,M,S5);
expBinnary(S5,E,M,R,m,S6,nbrPaquet);
montgomery(X,X,M,m,SS1);
add_mod(SS1,SS1,M,S2);
add_mod(SS1,S2,M,S3);
add_mod(A,S3,M,S4);
montgomery(S4,S6,M,m,Lmdd);}
```

Figure 4. :Le Code C de la fonction *LmdD*(.)

```
void CalculerX(Xuint32 Xp0[9],Xuint32 Xp1[9],Xuint32
Lmdd[9],Xuint32 M[9], Xuint32 m,Xuint32 X){
montgomery(Lmdd,Lmdd,N,m,S7);
add_mod(Xp0,Xp1,N,S8);
sub_mod(S7,S8,N,X);}
```

Figure 5 : Code C de la fonction *CalculerX*(.)

```
void CalculerY(Xuint32 xp[9],Xuint32 X[9],Xuint32 yp[9],Xuint32 Lmdd[9] ,Xuint32
M[9],Xuint32 m,Xuint32 Y){
 sub_mod(xp,X,M,S9);
 montgomery(S9,Lmdd,M,m,S10);
 sub_mod(S10,yp,M,Y);}
```

Figure 6 : Code C de la fonction *CalculerY*(.)

```
void expBinnary(unsigned long Y2[],unsigned long E[
],unsigned long N[ ],unsigned long R[ ],unsigned
long m,unsigned long Resultat[ ],int nbrPaquet) {
int i,j;
for (i=0;i<9;i++)   {
              S1[i]=Y2[i];
              C[i]=R[i]; }
for(i=0;i<nbrPaquet;i++){
      for(j=0;j<32;j++)  {
              if(getBitIndex(E[i],j)==1)
      {
              montgomery(C,S1,N,m,C);
              montgomery(S1,S1,N,m,S1);  }}
for(i=0;i<nbrPaquet;i++)Resultat[i]=C[i]; }
```

Figure 7: Code C de la fonction *expBinnary*(.)

Bibliographies

[1] C.Paar and J.Pelzl, "Understanding Cryptography", Springer-Verlag, 2010.

[2] R.L.Rivest, A.Shamir, and L.Adleman , « A Method for Obtaining Degital Signatures and Public-Key Cryptosystems", Communication. ACM, vol 21, No.2, pp.120-126, 1978.

[3] N.Koblitz, « Introduction to Elliptic Curves and Modular Forms », Graduate Text in Mmathematics, Vol.97, Springer, 1984.

[4] J.P.Deschamps, G.J.A.Bioul and G.D.Sutter, "Synthesis of Arithmetic Circuits : FPGA, ASIC, and Embedded Systems", wiley interscience, 2006.

[5] C.Maxfield, "FPGAs World Class Design", Elsivier, 2009

[6] "Microblaze Processor Reference Guide ", UG081 (v13.2)
http://china.xilinx.com/support/documentation/sw_manuals/xilinx13_2_mb_9_guide.pdf

[7] D.G.Mesquita, « Architectures Reconfigurables et Cryptographie : une analyse de Robustesse et Contremesures Face aux Attaques par Canaux Cachés », thèse de Doctorat de l'université de Montpellier II (2004).

[8] A.J.Menezes, P.C.V.Oorschot ans S.A.Vanstone, "Handbook Of Applied Cryptography", CRC Press, 1996

[9] W.Diffie and M.E.Hellman, "New Directions in Cryptography", IEE Transaction on Information Theory, vol.IT22, No.6, pp.644-654, 1976.

[10] D.Hankerson, A.Menezes, S.Vanstone, "Guide to Elliptic Curve Cryptography", Springer, 2004.

[11] J.J.Risel and P.Boyer, "Algébre pour la licence 3",27 fevrier 2006.

[12] J.S.Miline , « Elliptic Curves »,BookSurge Publishers,pp 238-viii,2006.

[13] J.Y.Chouinard, "Design of Secure Computer Systems CSI4138/CEG4394 : Notes on Elliptic Curve Cryptography, September 24, 2002.

[14] M.Joye and S.M.Yen, "The Montgomery Powering Ladder", Springer-Verlag, 2003.

[15] M.H.Ahmed, S.W.Alam and I.Baig, "Architecture and Implementation of Modular Arithmetic Unit for SOC", IEEE computer society, Frontiers of Information Technology 2011

[16] M.D.Ercegovac and T.Lang, "Digital Arithmetic" ,Morgan Kaufman Publishers,2003.

[17] P.L.Montgomery, " Modular Multiplication Without Trial Division", pMathematics of Computation ,vol 44 (170) , 519--521 .

[18] T.Plantard, "Arithmétique Modulaire pour la Cryptographie", Thése PhD, Université Montpelier II, Laboratoire d'Informatique, de Robotique et de Microélectronique, 2005.

[19] M.Issad, "Cryptosystème Embarqué sur FPGA",Thése de Doctorat, Département Télécommunications, Faculté d'Electronique et d'Informatique, Université des Sciences et de la Technologie Houari Boumediene, 2014.

[20] M.Gouy,G.Huvent and A.Ladureau , "Fermat revisité", 10 mars 2003.

[21] Joye, M., and Yen, S.-M. The Montgomery Powering Ladder. In Workshop on Cryptographic Hardware and Embedded Systems (CHES) (2002), vol. 2523 of Lecture Notes in Computer Science, pp. 291–302.

[22] Certicom Research, "Recommended Elliptic Curve Domain Parameters",Standards for Efficient Cryptography Version 1.0, September 2000

[23] H.Kim and S.Lee, "Design and Implementation of a Private and Public Key Crypto Processor and Its Application to a Security System", IEE Transaction on Consumer Electronics,vol.50 ,No.1 , 2004.

[24] M.Gouy,G.Huvent and A.Ladureau , "Fermat revisité", 10 mars 2003.

[25] M.Issad, B.Boudraa and M.Anane, "Efficient Hardware Implementation of Montgomery Multiplication for Embedded Cryptosystems", International Congress on Telecommunication and Application 14 University of A.Mira, Bejaia, Algeria, 2014.

[26] M.Issad, B.Boudraa, M.Anane and S.Sddiki "FPGA Implementation of Modular Exponentiation Using Single Modular Multiplier", International Conference on.Circuits, Systems, Signal Processing, Communications and Computers, Venice, Italy, 2014.

[27]E.A.Lee and S.A.Seshia , "Introduction Of Embedded Systems A Cyber-Physical Systems Approch, LeeSeshia.org, 2011.

[28] R.Saleh, S.Wilton, S.Mirabbasi, A.Hu, M.Greenstreet, G.Lemieux, O.P.Paid, C.Grecu, and A.Ivanoc,"System-on-chip : Reuse and Integration", Proceedings of THE IEE , vol 94, No.6, June 2006 .

[29] S.C.L.Thesis, "Rapid Prototyping Of Embedded Systems Using Field Programmable Gate Arrays", 2009.

[30] W.Wolf, "Hardware-Software Co-Design of Embedded Systems", Proceedings of THE IEE , vol 82, No.7, July 1994.

[31] C.Maxfield, " FPGAs world class designs", 2009

[32] J.P.Deschamp, G.Jean et G.D.Sutter, "Synthesis of arithmetic circuit", 2006

[33] T.Nachef , "Implémentation d'une instrumentation sur FPGA" , université Mouloud Mammeri de Tizi-Ouzou, 201

[34] B.Daya, "Rapid Prototyping of Embedded Systems Using Fiels Programmable Gate Arrays", Bachelor of Science in Electrical Engeineerin, Spring ,2009.

[35] "Virtex-5 User Guide", UG190 (v 1.2), 2006.
http://www.ece.iastate.edu/~zambreno/classes/cpre583/2006/documents/xilinx/ug190.pdf

[36] J.P Ddeschamps, G.J.A Bioul and G.D., "Synthesis of Applied Crypography", CRC Press, 1996

[37] "Genesys Board, Reference Manual", Revision,2012.
http://www.digilentinc.com/Data/Products/GENESYS/Genesys_rm.pdf

[38] "Xilinx Device Drivers Documentation", Generated on 24 Jun 2004 for Xilinx Device Drivers.
http://www.xilinx.com/ise/embedded/edk6_2docs/xilinx_drivers.pdf