# UNIVERSITE BLIDA 1

**Institut d'Aéronautique et des Etudes Spatiales**


## THESE DE DOCTORAT EN SCIENCE

Spécialité : Aéronautique

# CRYPTOGRAPHIE A BASE DE COURBE ELLIPTIQUE

# ET

# IMPLEMENTATION


**Par**

**AZINE Houria**


**Devant le jury composé de :**

| | | |
|---|---|---|
| **S.Boukraa** | **Professeur, Université de Blida 1** | **Président** |
| **M.Hamadouche** | **Professeur, Université de Boumerdés** | **Examinateur** |
| **Boukaba** | **MCA, ESTA Dar El Beida** | **Examinateur** |
| **B.Beladjine** | **Professeur, USTO-Oran** | **Examinateur** |
| **M. Lagha** | **Professeur, Université de Blida 1** | **Examinateur** |
| **A. Guessoum** | **Professeur, Université de Blida 1** | **Directeur de thèse** |
| **A.Bencharif** | **MCA, Université d'Arabie saoudite** | **Invité** |

**Blida, le 15 octobre 2020**

**RESUME**

Cette thèse porte principalement sur la contribution au domaine de cryptographie à courbe elliptique afin de réaliser le protocole d'accord Diffie-Hellmann (ECDH). Celle-ci, a permis le développement d'une nouvelle boîte à outils Matlab appelée ECC_NIST qui est une extension à la bibliothèque Matlab déjà existante, qui peut gérer de champs Galois du type GF ($2^m$) sur les courbes elliptiques. Les corps binaires GF($2^m$) présentent l'avantage de faciliter l'implémentation sur nos ordinateurs pour d'évidentes raisons de cohérence et de compatibilité entre leur représentation binaire et la technologie informatique.

L'opération principale du protocole d'accord ECDH est la multiplication scalaire qui consiste à additionner et à doubler un certain nombre de fois un point de courbe elliptique avec lui-même. Donc optimiser cette opération, faire un bon choix de paramètres de la courbe ainsi que le choix des coordonnées sont des critères crucials. Nous avons choisi la méthode de la réduction de Montgomery, qui est une méthode de réduction permettant d'éviter l'utilisation de divisions coûteuses en les remplaçant par des opérations très efficaces comme des décalages. Pour pouvoir utiliser ces décalages, la contrainte est d'autoriser la sortie à être un peu supérieure à p pour une réduction modulo p, et ainsi une optimisation au niveau du multiplieur modulaire qui est basé sur l'algorithme de Karatsuba-Ofman pour l'accélération de l'opération la plus lourde et la plus complexe, car le produit de polynômes et d'entiers est une opération élémentaire, qui intervient dans le calcul de KP . L'efficacité de cet algorithme repose donc sur celle du produit. Pour multiplier deux polynômes de degré n à coefficients dans un anneau A, la méthode classique requiert O($n^2$) opérations dans A. De même, l'algorithme scalaire de multiplication de deux entiers à n chiffres nécessite un nombre d'opérations binaires en O($n^2$) par contre l'algorithme de Karatsuba, réduit la complexité à O($n^{1,59}$).

Il existe de nombreuses références dans l'état de l'art qui présentent des algorithmes d'inversion modulaire d'un opérande. Pour notre protocole d'échange, nous avons utilisé l'algorithme inverse de Itoh-Tsujii multiplicatif sur le champ GF ($2^m$) . Ainsi on réduit le nombre au minimum d'opération car elle est très coûteuse vis à vis des autres opérations comme la multiplication et la règle de la chaîne d'addition sur les courbes elliptiques est donnée par NIST.

**Mots clés :** Cryptographie par les courbes elliptiques, cryptanalyse, Générateur de nombres pseudo-aléatoires, algoritme de Montgomry , l'algorithme inverse de Itoh-Tsujii, l'algorithme de Karatsuba.

**ملخص**

تركز هذه الرسالة بشكل أساسي على المساهمة في مجال تشفير المنحنى الإهليلجي من أجل تحقيق بروتوكول اتفاق Diffie-Hellmann (ECDH). لهذا ، قمنا بتطوير مربع أدوات Matlab جديد يسمى ECC_NIST وهو امتداد لمكتبة Matlab الموجودة بالفعل ، والتي يمكن أن تدير حقول غالوا من نوع GF ($2^m$) على المنحنيات الإهليلجية. تتمتع هيئات GF($2^m$) الثنائية بميزة تسهيل التنفيذ على أجهزة الكمبيوتر الخاصة بنا لأسباب واضحة من الاتساق والتوافق بين تمثيلها الثنائي وتكنولوجيا الكمبيوتر.

العملية الرئيسية لمذكرة التفاهم ECDH هي الضرب العددي ، والذي يتكون من إضافة ومضاعفة نقطة منحنى إهليلجي مع نفسه عدد معين من المرات. لذا ، لتحسين هذه العملية ، لجعل اختيارًا جيدًا لمعلمات المنحنى ، وكذلك اختيار الإحداثيات ، معايير حاسمة. لقد اخترنا طريقة تخفيض Montgomery ، وهي طريقة تخفيض تتجنب استخدام الأقسام المكلفة وتستبدلها بعمليات فعالة للغاية مثل الإزاحة. لتكون قادرة على استخدام هذه التحولات ، فإن القيد هو السماح للإخراج أن يكون أكبر بقليل من p بالنسبة لمعامل التخفيض p ، وبالتالي تحسين على مستوى المضاعف المعياري الذي يستند إلى خوارزمية Karatsuba-Ofman ل أثقل وأسرع عملية تسريع ، لأن ناتج متعددو الحدود والأعداد الصحيحة هو عملية أولية ، والتي تتدخل في حساب KP.
تعتمد فعالية هذه الخوارزمية على المنتج. لمضاعفة متعددو الحدود من الدرجة n مع معاملات في الحلقة A ، تتطلب الطريقة الكلاسيكية عمليات( $n^2$) O في A. وبالمثل ، تتطلب خوارزمية المدرسة لمضاعفة عدد صحيحين بأرقام n عددًا من العمليات الثنائية في $O(n^2)$ من ناحية أخرى ، تقلل خوارزمية Karatsuba من التعقيد إلى ($n^{1.59}$) O.‘
هناك العديد من المراجع في أحدث التقنيات التي تقدم خوارزميات انعكاس معيارية لمعامل ، لبروتوكول التبادل الخاص بنا ، استخدمنا الخوارزمية المضاعفة Itoh-Tsujii المضاعفة في حقل ($2^m$) GF ، وبالتالي يتم تقليل عدد العمليات إلى الحد الأدنى لأنه مكلف للغاية فيما يتعلق بالعمليات الأخرى مثل الضرب وقاعدة سلسلة الإضافة على المنحنيات الإهليلجية التي قدمها NIST .

**الكلمات المفتاحية**: تشفير المنحنى الإهليلجي ، تحليل الشفرات ، مولد الأرقام العشوائية المزيفة ، خوارزمية مونتغمري ، خوارزمية عكس إيتوه تسوجي ، خوارزمية كاراتسوبا.

## Abstract

This thesis focuses mainly on the contribution to the field of elliptic curve cryptography in order to achieve the Diffie-Hellmann agreement protocol (ECDH).
For this, we have developed a new Matlab toolbox called ECC_NIST which is an extension to the already existing Matlab library, which can manage Galois fields of the GF type (2m) on elliptic curves. GF binary bodies (2m) have the advantage of facilitating implementation on our computers for obvious reasons of consistency and compatibility between their binary representation and computer technology.

The main operation of the ECDH Memorandum of Understanding is scalar multiplication, which consists in adding and doubling a point of elliptic curve with itself a certain number of times. So to optimize this operation, to make a good choice of parameters of the curve as well as the choice of coordinates are crucial criteria. We chose the Montgomery reduction method, which is a reduction method that avoids the use of costly divisions and replaces them with very efficient operations such as offsets. To be able to use these shifts, the constraint is to allow the output to be a little greater than p for a reduction modulo p, and thus an optimization at the level of the modular multiplier which is based on the Karatsuba-Ofman algorithm for the heaviest and most complex operation acceleration, because the product of polynomials and integers is an elementary operation, which intervenes in the calculation of KP. The effectiveness of this algorithm therefore rests on that of the product. To multiply two polynomials of degree n with coefficients in a ring A, the classic method requires O ($n^2$) operations in A. Likewise, the school algorithm for multiplying two integers with n digits requires a number of binary operations in O ($n^2$) on the other hand the algorithm of Karatsuba, reduces the complexity to O ($n^{1,59}$).

There are many references in the state of the art which present modular inversion algorithms of an operand, for our exchange protocol, we used the inverse Itoh-Tsujii

multiplicative algorithm on the GF field ($2^m$), thus the number of operations is reduced to a minimum since it is very costly with respect to other operations such as multiplication and the rule of the addition chain on the elliptical curves is given by NIST.

**Keywords:** Cryptography by elliptic curves, cryptanalysis, Pseudo-random number generator, Montgomery algorithm, Itoh-Tsujii inverse algorithm.

REMERCIEMENTS

*« L'effort est pénible mais il est aussi précieux, plus précieux encore que l'œuvre où il aboutit, parce que, grâce à lui, on a tiré de soi plus qu'il n'y avait, on s'est haussé au-dessus de soi-même ».*

Henri Bergson, *L'Energie spirituelle*

La fin d'une thèse est l'une des étapes qui marque une vie. Dans ces rares occasions, il est bon, je crois, de s'arrêter, de regarder en arrière et de se souvenir. Et sachant que les remerciements sont les pages les plus lues d'une thèse, j'en profite pour en faire un peu de publicité.

Mes premières pensées vont naturellement à Allah, le bon dieu, pour toutes les chances qu'il m'a offert dans la vie, vers mon directeur de recherche, le Professeur GUESSOUM pour son soutien et sa disponibilité et aussi mon co-directeur Mr BENCHARIF , qui a cru en moi plus que moi-même, et bien avant que je ne m'y autorise. En plus de sa confiance aveugle, la liberté qu'il m'a tout du long accordée a été indispensable à mon épanouissement. Il m'a appris à construire des raisonnements complexes sans crayon ni papier. Je ne le remercierai jamais assez de m'avoir confié ce sujet. Les logarithmes, les courbes elliptiques plutôt la cryptologie est rentrée dans mon quotidien. Mille mercis, pour cette agitation que vous m'avez transmise Mr Bencharif . Pour vous le désespoir et le pessimisme n'existent pas.

Si ce manuscrit est entre vos mains c'est qu'il a reçu l'aval précieux et indispensable de mes rapporteurs. Donc, je tiens à remercier l'ensemble du jury d'avoir accepté l'invitation et d'avoir fait le déplacement.

Je tiens également à remercier le Pr Benselama, pour sa bravoure et son sens de la responsabilité et surtout d'avoir mis Mr Bencharif sur mon chemin.

Je souhaite remercier sincèrement ma raison d'être, mon oxygène, ma fille Chiraz et mon rêve c'est qu'elle brille comme une étoile.

À ma mère, mes sœurs et leurs enfants « Ghizlane, Rami, Almas, Anaïs et Racim » et mes frères et leurs enfants. J'aurais tant aimé que ma sœur Djamila et mon Papily soient présents mais Allah a décidé autrement.

Ainsi que mes amis qui m'ont apporté un soutien sans faille.

TABLES DES MATIERES

LISTE DES ILLUSTRATIONS, GRAPHIQUES ET TABLEAUX

# INTRODUCTION GENERALE

*« La sécurité du chiffre ne doit pas dépendre de ce qui ne peut pas être facilement changé ». Kerckhoff*

## Contexte historique

La cryptographie est une branche à la frontière des mathématiques et de l'informatique qui prend une importance grandissante à notre époque où les transactions « immatérielles », via les réseaux informatiques, sont en passe de devenir une réalité tangible et inévitable dans la vie de l'individu lambda.

La sécurité informatique est un marché en pleine expansion, en témoignent les chiffres d'une étude réalisée en 2015 [1]. Il va, d'après certaines estimations, croitre d'un montant de 77 milliards de dollars en 2015 à plus de 170 en 2020, soit une croissance annuelle d'environ 17%. Alors qu'elle n'était qu'un sujet secondaire au début des années 2000, la sécurité est devenue un point névralgique important, notamment avec l'émergence du Cloud, la monnaie numérique et de fait, de l'apparition de la problématique de la vie privée.

Longtemps cantonnée aux sphères militaires et politiques, la cryptographie est devenue donc un véritable enjeu de société. Le développement rapide des réseaux de communication numérique, ainsi que l'augmentation du nombre de ses utilisateurs, ont posé de façon de plus en plus critique le problème de l'échange des clés de chiffrement. Le problème d'échange de clés prend une toute autre dimension à l'heure d'internet et de son milliard d'usagers.

La technologie actuelle, malgré, toutes les avancées faramineuses, se voit défier par divers paramètres essentiels, tels que la faible consommation des circuits électroniques, l'autonomie en puissance, la fréquence de leurs horloges, l'espace ou la densité des transistors intégrés, ainsi que les coûts de conception et de mise sur le marché. Ces paramètres définissent la stratégie de conception des systèmes embarqués, tels que ceux utilisés dans les satellites, les portables, les robots mobiles, les nouveaux engins intelligents autonomes, les monnaies numériques « Bitcoin»[2] etc.…

En effet, la complexité réside dans l'optimisation de ce problème multidimensionnel, par des solutions partielles, car actuellement, nul ne dispose d'un circuit à consommation nulle ou presque avec une fréquence d'horloge infini, dans un espace infiniment petit, qui peut être réalisé dans un laps de temps infine,…

Les systèmes les plus complexes, contiennent des fonctions très évoluées, que beaucoup de constructeurs essaient de divulguer, afin de pallier au reverse engineering. Actuellement, une panoplie de constructeurs n'inventent pas, mais attendent qu'un produit soit sur le marché, et il est répliqué dans les jours qui suivent. Dans cet axe de recherche, les vrais constructeurs cachent les secrets de conception, les méthodes utilisées, les algorithmes implantés, le choix des équations normalisées ou non, les codes exécutables, même aux employés de la même entreprise, cette procédure nécessite alors des moyens très fiables d'encapsulation et de chiffrement.

Au cours des dernières années, un sujet sur la théorie des nombres et la géométrie algébrique appelé « courbes elliptiques » a trouvé un champ d'application en cryptographie. Dans cette approche, la complexité des algorithmes de chiffrement/déchiffrement ne cesse d'augmenter, et les mathématiques impliquées deviennent de plus en plus complexes, cas du chiffrement à base des courbes elliptiques qui se base sur des courbes d'une forme très spéciale avec des opérandes de l'ordre des centaines appartenant à un ensemble très particulier dit, champ de Galois.

Les courbes elliptiques sont devenues au fil des années un concept incontournable de la cryptographie asymétrique. Mais au final, elles restent un concept assez obscur pour beaucoup.

Avant tout, ces courbes sont des outils mathématiques intéressants. Bien que récentes en cryptographie (introduites en 1985), elles ont déjà été largement étudiées et ont notamment été utilisées par « Andrew Wiles » dans sa démonstration du grand Théorème de Fermat. Leurs propriétés leur donnent un intérêt particulier pour la cryptographie asymétrique.

L'exécution de ces algorithmes sur des ordinateurs personnels, ne satisfait plus le besoin en vitesse de génération des résultats, comme par exemple le cas de la fonction de hachage SH256 dans le protocole de Bitcoin. Alors des architectures dédiées doivent être réalisées.

Pour assurer la sécurité de nombreuses applications, il est nécessaire d'utiliser des primitives de cryptographie asymétrique, dans les protocoles comme l'échange de clé secrète, la signature numérique ou certains chiffrements spécifiques. La cryptographie sur courbes elliptiques est devenue le Standard en cryptographie asymétrique dans de nombreux pays. Elle supplante RSA du fait de ses bien meilleures performances et de son moindre coût.

Qui fabrique et "Standardise" la cryptographie ?

- Coté théorique :
  - Universités ;
  - Entreprises privées ;
  - Agences gouvernementales.
- Coté Recommandations/Standards :
  - NIST (USA), GOST (Russie), KISA (Corée du sud) ;
  - IETF/CFRG ;
  - Industriels (PKCS, etc).

L'invention et la mise en œuvre commerciale éventuelle de la CCE ont suivi un chemin différent. Puisque ni Koblitz ni Miller n'ont demandé un brevet, l'idée sous-jacente fondamentale de CCE est librement disponible pour tous. Cela a conduit Certicom et d'autres entreprises à postuler à des brevets améliorant l'idée de base de la CCE.

Deux organismes sont alors connus pour breveter des courbes elliptiques aux propriétés intéressantes : le NIST [3] et CERTICOM [4]. Ils proposent tous les deux d'utiliser des courbes obéissant à l'équation de Weierstrass qui utilisent les paramètres « a, b et p » bien définis.

**Contexte de la thèse**

Il existe plusieurs types de cryptographie. Actuellement, dans les protocoles de sécurité, on utilise généralement deux types de cryptographie : la cryptographie asymétrique (ou à clé publique) et la cryptographie symétrique (ou à clé secrète). Ces deux catégories ont des caractéristiques différentes et sont utilisées généralement de façon complémentaire. La cryptographie asymétrique requiert des calculs compliqués sur des grands nombres, ce qui rend ces opérations coûteuses en temps. Les travaux de cette thèse s'intéressent exclusivement à ce type de cryptographie. Plus précisément, on va s'intéresser à l'accélération des calculs pour la cryptographie sur courbes elliptiques pour un protocole ECDH.

On trouve deux grandes catégories de courbes elliptiques dans l'état de l'art de CCE : les courbes ayant comme corps de base $GF(2^m)$ et celles définies sur Fp. Les courbes définies sur $GF(2^m)$proposent une arithmétique généralement plus efficace, car le corps de base $GF(2^m)$ possède plus de structure permettant l'accélération des calculs.

Alors, lors de l'étude de la cryptographie à courbe elliptique (CCE), nous sommes directement référés à CERTICOM pour leur introduction didacticiel; mais leur présentation donne seulement des explications élémentaires sur les opérations basiques

sur des réels et des corps finis avec un petit nombre premier m = 4bits d'où l'idée de concevoir une contribution sur Matlab pour le protocole ECDH.

Dans ce travail, nous allons donc présenter une boîte à outil développée pour Matlab qui traite une valeur principale de m =113 bits. Le Matlab est un logiciel de calcul numérique disponible sur plusieurs plateformes, on y trouve un toolboox et un tutoriel ,mais quelques fonctions de Galois seulement, le nombre « m » du nombre premier est très petit , est limité à 16 bits et aussi quelques opérations du champ de Galois avec des polynômes générateurs, mais pas de courbe elliptique d'où l'idée d'enrichir la bibliothèque Matlab en ajoutant une boite à outils (toolbox) avec des fonctions supplémentaires c'est-à-dire une extension à la boite à outils « bibliothèque » déjà existante sur Matlab. L'implémentation d'un système CCE nécessite un nombre de décision à différents niveaux.

L'enjeu de cette thèse est double : il faut d'un côté, l'accélération de certaines opérations sur des nombres de plusieurs centaines à plusieurs milliers de bits pour les calculs nécessaires à la cryptographie asymétrique CCE et d'un autre coté, implémenter ces opérateurs arithmétiques rapides et peu gourmands qui sont définies sur GF($2^m$) bien sure en justifiant le choix des courbes elliptiques ainsi que les paramètres dans une boite à outil Matlab. Donc une implantation software de l'algorithme de chiffrement à base des courbes elliptiques (CCE) normalisée de NIST sur des courbes prédéfinies (courbes nommés) par des standards de sécurité comme NIST et SEC, ces courbes sont vérifiés et validés par ces standards.

**Plan du rapport de thèse**

Afin de bien organiser le travail, le manuscrit de ce mémoire a été organisé en quatre chapitres principaux. Les chapitres 1, 2 et 3 introduisent les éléments nécessaires à l'étude et à la conception de notre boite à outils, menées au chapitre 4.

Le premier chapitre introduit les concepts fondamentaux liés à la cryptographie et à la sécurité. Nous présenterons en premier temps, les concepts de la cryptographie moderne comme ils sont définis par les standards internationaux de la sécurité. Et dans en deuxième temps, on décrit les différentes attaques existantes et leurs contre-mesures.

Le deuxième chapitre présente les concepts mathématiques liés aux courbes elliptiques, et l'application de ces dernières dans la cryptographie, notamment dans l'échange de clé entre individus et aussi nous avons fait est un tour d'horizon sur les principales propriétés des courbes elliptiques et des différentes formules d'addition de points selon le système de coordonnées choisi , sur le choix de l'algorithme , ainsi que sur

les mathématiques nécessaires à son exécution. Aussi nous avons défini le fameux problème du logarithme discret sur les courbes elliptiques et sa complexité qui a contribué à l'évolution de la cryptographie par la naissance de la cryptographie par les courbes elliptiques (CCE). Et en particulier, ce chapitre présente aussi l'arithmétique modulaire, car la solution architecturale proposée dans cette thèse est basée sur cette catégorie d'opérations donc c'est la colonne vertébrale de notre conception. Le choix de l'algorithme d'accélération de l'opération de la multiplication scalaire est crucial.

Le troisième chapitre a pour objet l'étude et analyse de deux courbes, une appartenant à NIST et la deuxième non normalisée, pour cela nous avons pris comme exemple la courbe SEC256r1 appartenant à NIST et la courbe SEC256 k1 que Satoshi a utilisé dans son protocole de Bitcoin et justifier le choix de ce dernier. Comme énoncé précédemment les choix arithmétiques influent aussi sur les performances de l'opérateur tels que les paramètres (p, a, b, G, n, h), le corps de base GF ($2^m$) ou Fp ainsi que le type d'équation, tous ces paramètres sont les clés de sécurité. Car Malgré les bonnes propriétés mathématiques de CCE, d'autres types d'attaques existent en utilisant les fuites d'information des implantations, comme la consommation d'énergie ou le rayonnement électromagnétique et surtout de l'existence de porte dérobée dans les courbes standards. C'est d'autant plus vrai lorsque nous travaillons sur des implantations sur systèmes embarqués, où des protections spéciales doivent être mises en œuvre.

Le quatrième chapitre, expose les étapes de la conception et de l'extension de la boite à outils Matlab du protocole ECDH, et cela en expliquant chaque script de notre conception, ainsi que les algorithmes choisis pour optimiser et accélérer l'opération de la multiplication scalaire ce qui permet d'accélérer certains calculs, et donner les résultats partiels de chaque étape ainsi que le résultat final de l'échange de clés.

Pour finir, la conclusion propose une synthèse des idées principales qui ont mené à la conception et l'implémentation de la courbe sec113r1. Les perspectives envisagées pour cette conception sont nombreuses pour une amélioration éventuelle de ce travail.

# CHAPITRE 1
# CRYPTOGRAPHIE ET CALCUL INTENSIF ET PRECIS

*« Il existe deux types de cryptographie dans le monde : la cryptographie qui protège vos documents de la curiosité de votre petite sœur et celle qui empêche les gouvernements les plus puissants de lire vos fichiers. Cet ouvrage s'adresse au dernier cas».*

*Bruce Schneider,*

## 1.1 Introduction

La cryptographie est un domaine dont l'objectif principal est de protéger l'information, de la rendre inintelligible à celles ou ceux à qui elle n'est pas destinée. Elle repose sur des algorithmes solides qui s'appuient eux-mêmes sur des problèmes mathématiques réputés difficiles (logarithme discret, factorisation des grands nombres, etc.).

L'histoire de la cryptographie est passionnante, sa pratique remonterait à l'Antiquité sous des formes évidemment plus rudimentaires qu'aujourd'hui. Sans retracer les méandres de son évolution, notons tout de même que son utilisation a connu un véritable boom peu avant la seconde guerre mondiale par le développement d'une machine à chiffrer « Enigma » [5].Le récit a été d'ailleurs popularisé en 2015 par le film Americo-Britannique « Imitation Game » dans lequel Alain Turing, aujourd'hui considéré comme l'un des pères fondateurs de l'informatique, parvient à casser le chiffrement allemand à travers un processus d'automatisation des calculs.

Les cryptosystémes à clefs publiques sont construits sur des problèmes mathématiques difficiles à résoudre.

- Le premier problème utilisé fut le problème de la factorisation d'un entier ;
- Un second problème important est le problème du logarithme discret.

Dans cette thèse, on utilise plus particulièrement la cryptographie basée sur les courbes elliptiques (CCE). Son principal avantage est que l'on ne connait pas d'algorithme sous-exponentiel pour résoudre le problème du logarithme discret sur ce groupe lorsque la courbe est bien choisie.

La principale conséquence est que pour le système CCE des clés plus petites permettent d'atteindre un niveau de sécurité donné (taille proportionnelle d'un facteur 2 à la taille d'une clé symétrique). Ceci est d'autant plus intéressant que le niveau de sécurité est élevé. L'utilisation de la cryptographie basée sur les courbes elliptiques a aussi d'autres différences majeures.

La génération d'une clé secrète RSA [6] est un problème difficile car il faut choisir deux grands nombres premiers de manière aléatoire. A contrario la génération d'une clé CCE est bien plus simple car il s'agit simplement de choisir un nombre au hasard entre 1 et une borne supérieure fixée sans test de primalité.

## 1.2 Enjeux et problématiques de la cryptographie

Parmi les domaines les plus porteurs en termes de sauvegarde, protection, authenticité des données est la cryptographie. En effet, et cela depuis la nuit des temps, différents rois et hommes politiques codaient leur messages, en vue de les transmettre le plus sûrement possible; ainsi apparu le dilemme cryptographie-cryptanalyse, le premier concept concerne l'encapsulation du message dans un autre message, le deuxième concept concerne la divulgation du message caché en utilisant des moyens controversés pour craquer la manière d'encapsulation.

Les deux notions sont complémentaires, à l'inverse de ce que l'on croirait, car la cryptanalyse pousse la cryptographie à ses limites et la force à être un pas en avant.

La relation cryptographie-précision est si adjointe que toute cryptographie non précise impliquerait deux problématiques, la première est la perte du message d'origine, la seconde est le tort que peut compromettre une telle transaction.

## 1.3 Historique de la cryptographie

Les méthodes de cryptographie ont commencées depuis plusieurs siècles, les premiers parchemins encryptés, les symboles des peuples primaires par l'utilisation de la fumée comme message codé, etc.…

Actuellement, la cryptographie est une science bien distinguée [7], elle consiste à transformer un message apparent, clair en un autre message incompréhensible à un tiers parti, en vue de le transmettre ou de le sauvegarder. Un procédé réciproque ou plutôt un procédé inverse est utilisé chez le destinataire du message pour reconnaître le message codé, donc l'opération se résume à cacher et à divulguer, d'une manière non évidente.

A l'ère de l'internet, les domaines d'application de la cryptographie sont devenus plus important que l'ancien message caché dans le revers de la poche du chevalier ou le parchemin à lire à l'envers.

La figure 1.1, présente le modèle des six couches hiérarchiques, pour la sécurité des applications du système d'information.

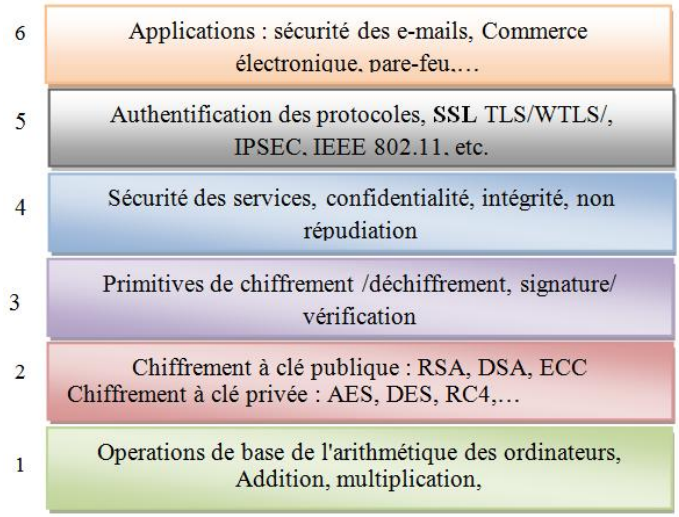



Figure 1.1 : Modèle des six couches hiérarchiques, pour la sécurité des applications du système d'information [7].

Dans le contexte de notre conception, la première couche (1) sera notre brique de base, en vue de l'implantation d'un algorithme de la seconde couche (2), notamment la cryptographie à base des courbes elliptiques (CCE).

## 1.4 La cryptographie classique

Dans le schéma ci-dessous figurent les différentes branches de la cryptographie classique.

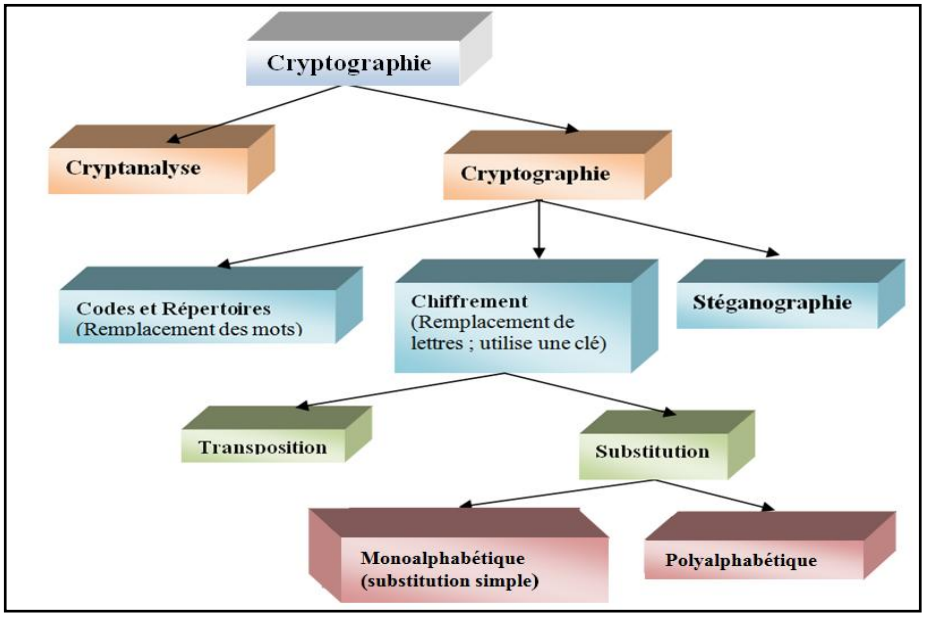



Figure 1.2 : Domaine inclus dans la cryptologie.

## 1.5 Services de la cryptographie

Dans un cas très général, la cryptographie concerne toute situation où l'on veut limiter l'information à un espace d'utilisateurs restreint. Par l'étude de bases mathématiques reliées à la sécurité de l'information. Quatre services nécessaires tels que la confidentialité, l'intégrité des données, l'authentification et la non-répudiation, doivent être mis en place [8] comme le montre les figures ci-dessous.

### 1.5.1 Confidentialité

Elle est amenée par le chiffrement du message. Dans le cas de systèmes à clés symétriques, la même clé est utilisée pour $E_K(M)$ et $D_K(C)$. Ce type de chiffrement nécessite un échange sûr préalable de la clé K entre les entités A et B. Les figures 1.3 et 1.4 présentent la confidentialité d'un système symétrique et un système hybride.

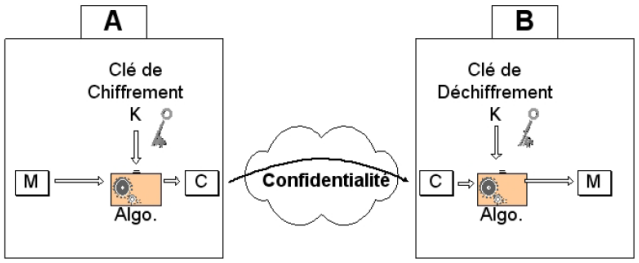



Figure 1.3 : Confidentialité d'un système symétrique.

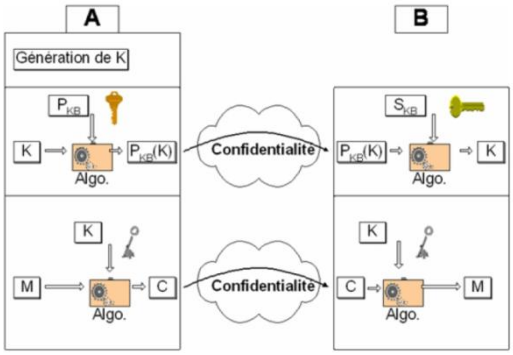



Figure1.4 : Confidentialité d'un système hybride.

### 1.5.2 Intégrité des données

Ce service s'occupe de la non-altération des données, par tout changement, destruction ou par perte accidentelle ou autre.

Il faut ici vérifier si le message n'a pas subi de modification durant la communication. C'est ici qu'interviennent les fonctions de hachage comme le montre la figure 1.5.

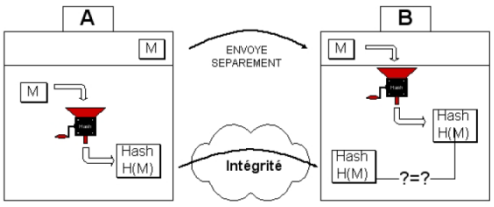



Figure 1.5 : Vérification de l'intégrité par fonction de hachage.

### 1.5.3 Authentification

Ce service est relié à l'identification. Sa fonction s'applique à l'entité (émetteur ou récepteur) et à l'information. Par exemple, si deux parties entrent en communication, l'information transmise à travers les canaux de transmission doit être authentifiée comme originale, date de l'envoi, lieu de l'envoi, contenu, etc.… ; Pour ces raisons, cette notion de cryptographie est divisée en deux classes majeures, authentification de l'entité et authentification de l'origine des données.

### 1.5.4 Non-répudiation

Ce service empêche qu'une entité nie toute transaction réalisée auparavant. Par exemple, qu'un parti nie avoir vendu une propriété à un deuxième parti, dans ce cas un troisième parti de confiance entre en jeu pour résoudre tout conflit d'intérêt.

## 1.6 La Stéganographie

La stéganographie est une solution très simple qui peut être mise en place aisément. Au lieu de rendre le message illisible comme la cryptographie, la stéganographie cherche plutôt à cacher les informations sensibles dans les paquets de contrôle ou les données.
Généralement, dans les paquets, notamment la partie d'entête, il existe toujours quelques octets qui ne sont pas utilisés, on peut donc les utiliser pour stocker les données [9].
L'attaquant ne peut pas détecter leur existence tant qu'il ne sait pas la technique exacte utilisée.

La fiabilité de la stéganographie est en effet un pari sur le fait que l'attaquant ne sait pas où se sont cachées les données. Cependant, une fois la technique utilisée dévoilée, l'attaquant pourra accéder à toutes les données cachées. D'ailleurs, le nombre d'octets non utilisés dans des paquets est très limité, on ne peut pas y stocker des informations très volumineuses.

## 1.7 Histoire de clé

La cryptographie agit sur un texte clair, en vue de le rendre inaccessible à un tiers parti, en remplaçant ce texte clair par un texte chiffré en utilisant des méthodes mathématiques assurant le recouvrement du texte clair par des méthodes de cryptographie inverses.

La cryptographie se divise en deux catégories, la cryptographie à clé publique et la cryptographie à clé symétrique. Donc à la base, il ya une clé de cryptographie que nous allons détailler dans ce qui suit.

Le principe fondamental d'un algorithme de cryptographie est basé sur deux notions essentielles, énoncées par Shannon:

- La confusion, qui vise à rendre le texte aussi peu lisible que possible. Ceci peut se faire par une substitution systématique de symboles, ou par un algorithme de codage aussi complexe que l'on veut.
- La diffusion, qui vise à rendre chaque élément d'information du ciphertext dépendant d'un nombre aussi grand que possible d'éléments d'information du plaintext

### 1.7.1 La clé secrète

Mathématiquement, un crypto système à clé symétrique/ publique, est défini comme un système permettant de faire deux fonctions [10].

Conversion du texte clair ou message $\boldsymbol{M}$ en un texte chiffrée $\boldsymbol{M'}$ de la manière suivante : $\boldsymbol{E_K(M) = M'}$

Et d'assurer le processus inverse de recouvrement du texte clair ayant le texte chiffré : $\boldsymbol{D_K(M') = D_K(E_K(M)) = M}$ . La figure 1.6 illustre ce procédé.

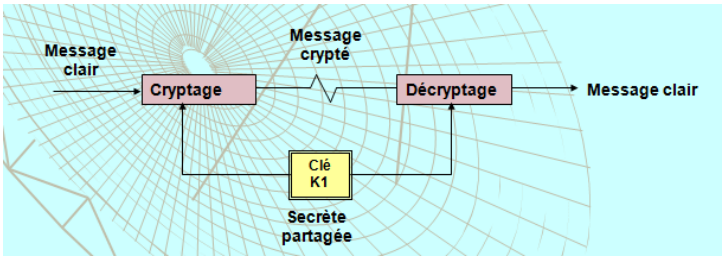



Figure 1.6 : Chiffrement et déchiffrement à l'aide d'une clé symétrique.

Pour le cas d'une clé asymétrique, le chiffrement s'effectue avec la clé $K_1$ (clé publique du destinataire) tandis que le déchiffrement s'effectue avec la clé $K_2$ (clé privée du destinataire) comme suit :

L'émetteur chiffre : $\boldsymbol{E_{K_1}(M) = M'}$

Le récepteur déchiffre : $\boldsymbol{D_{K_2}(M') = D_{K_2}(E_{K_1}(M)) = M}$

La figure 1.7, illustre ce procédé.

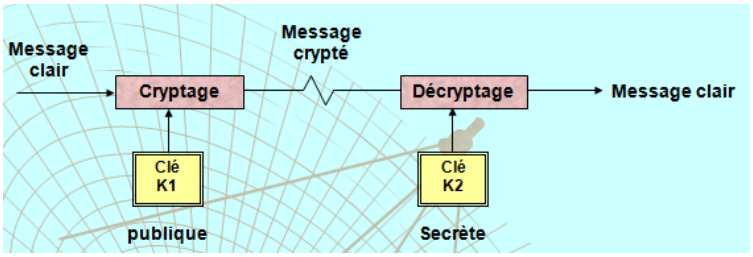



Figure 1.7 : Chiffrement et déchiffrement à l'aide de clés différentes.

Toute la sécurité du système de cryptographie repose sur la connaissance des clés de chiffrement [11]. Les différents algorithmes utilisés lors de la cryptographie peuvent être divulgués, voire même normalisés.

### 1.7.2 Algorithmes à clés symétriques

Ce type d'algorithme [12] appelé aussi algorithme conventionnel ou algorithme à une clé, (one-Key), [7] permet de calculer la clé de déchiffrement en utilisant la clé de chiffrement. Toutefois, il est impératif que les parties (émetteur et récepteur) se mettent d'accord sur une clé commune. Si la clé tombe dans les mains d'une tierce personne, tout le système de cryptographie devient obsolète.

Les algorithmes basés sur les clés symétriques se divisent, par leur manière de chiffrement, en deux catégories :

- Algorithmes à flux : cette catégorie chiffre le texte clair au niveau bit ou octet. Le chiffrement du prochain bit dépend du bit actuel, un peu a l'image d'une machine d'état, prenons l'exemple des algorithmes SEAL, TWOPRIME, WAKE, RC4, A5,…
- Algorithmes à blocs : cette catégorie chiffre le texte clair, en général par blocs de 56,64 ou 128 bits en des blocs de même taille. Les algorithmes les plus connus sont le DES, AES, Serpent, RC6, MARS, IDEA, Twofish, …

La notion de clé symétrique a connu bien des évolutions, depuis les années 70's, lorsque la taille de la clé était de 56 bits, pour le DEA, défini dans le DES, en ces temps cette clé était une clé très sûre, ces jours-ci, quelques heures suffisent pour la casser.

Même si les systèmes à clés secrètes offrent une haute sécurité et un faible coût de calcul, ils présentent les inconvénients cités dans le tableau 1.1.

Table 1.1 : Inconvénients de la cryptographie à clés symétriques.

| Taches à réaliser | Inconvénients |
|---|---|
| Distribution et échange des clés | La clé commune doit être connue seulement par l'émetteur et le récepteur, ce qui incombe une responsabilité des deux parties, et toute fuite compromettra tout le système. |
| Gestion des clés | Les systèmes ayant plusieurs utilisateurs utilisent plusieurs clés. Pour des raisons de sécurité les clés doivent être changées à chaque session. |
| Incomplétude | L'authentification et la non-répudiation sont impossibles à implémenter complètement [7] |

### 1.7.3 Les limites de la cryptographie Symétrique

Le principal problème de cette méthode de chiffrement concerne la distribution des clés. En effet, si la même clé est utilisée par plus de 2 personnes, elle doit être abandonnée lorsqu'une copie est interceptée. Elle ne peut pas être authentifiée car elle est connue de plus d'une personne.

En plus du nombre de clés importants, cette technique de chiffrement est aussi limitée par la confidentialité de distribution de ses clés. En effet, la clé de chiffrement est identique à la clé de déchiffrement. Ainsi c'est la même clé qui va nous permettre à la fois de chiffrer le message et aux destinataires de le déchiffrer.

La multiplication des clés Pour établir un canal de communication entre deux individus :

- Il faut qu'il soit chiffré avec une clé partagée entre les deux individus ;
- Il est ainsi confidentiel pour ceux qui ne possèdent pas la clé de chiffrement.

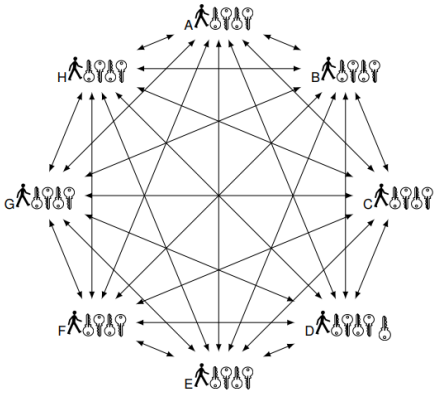



Figure 1.8 : Multiplication des clés pour une communication maillée [13].

Pour que deux canaux de communications soient indépendants l'un de l'autre, c'est à dire qu'une personne accède à l'un mais pas à l'autre, il faut que ces deux canaux utilisent des clés différentes.

Il est possible qu'un des interlocuteurs connaissent plusieurs clés utilisées dans différents canaux le reliant à des utilisateurs différents.

Exemple l'utilisateur D possède une clé pour chaque lien (avec H, G, F, E et C). Problème comment échanger toutes ces clés ?

Pas d'intégrité et d'identification de l'auteur, si Alice, Bob et Cédric partage le même lien de communication alors ils partagent la même clé de chiffrement symétrique.

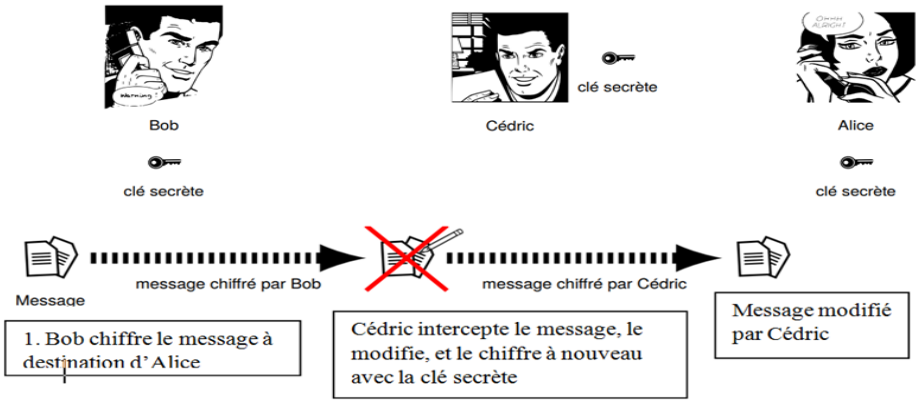



Figure 1.9 : Partage du même lien de communication [13].

Problème Chacun peut intercepter et modifier les messages qui s'échangent.

Ainsi, les limites du chiffrement par clé secrète sont principalement la sécurité incertaine lors du transfert de cette clé et la nécessité de générer autant de clés que de couples de correspondants

### 1.7.4 Algorithmes à clés asymétriques ou clés publiques

En vue d'éviter que la clé partagée, par les deux parties, ne soit divulguée, une technique basée sur l'utilisation d'une clé divulguée ou publique est introduite, celle-ci est basée sur des algorithmes qui séparent la clé de cryptage de la clé à transmettre. Les deux parties gardent une deuxième clé à leur niveaux, c'est-à-dire qu'ils utilisant une clé publique pour chiffrer et déchiffrer en introduisant une deuxième clé secrète à leur niveaux respectifs. Il faut noter que la clé publique ne permet pas de générer la clé secrète, donc aucune corrélation n'existe entre les différentes clés, à l'inverse de la clé symétrique.

Les deux clés (publique et secrète) sont utilises communément en vue de chiffrer et déchiffrer le message comme suit :

L'émetteur chiffre : $E_{K,K_1}(M) = C$

Le récepteur déchiffre : $D_{K,K_2}(C) = D_{K,K_2}(E_{K,K_1}(M)) = M$

1.8 Limites de la cryptographie à clé publique

Le principal problème du chiffrement à clé publique réside dans la lenteur à laquelle se font les opérations de chiffrement et de déchiffrement. C'est pour cela que dans la pratique, les chiffrements symétriques et asymétriques sont souvent combinés. Et on essaie d'exploiter leurs avantages du mieux possible pour rendre les systèmes performants. L'autre problème de cette méthode de chiffrement est qu'elle nécessite un nombre très important de calculs. En effet, des clés différentes sont utilisées pour chiffrer et déchiffrer. Il n'y a ainsi pas de problème de transfert de clé comme pour le chiffrement à clé secrète mais par contre les temps de calcul deviennent très longs.

1.9 Problèmes communs entre les cryptosystèmes Symétriques et Asymétriques

Deux problèmes persistent dans ces 2 types de cryptosystèmes.

- En premier lieu, la gestion des clés « secrètes », la confidentialité reposant exclusivement sur ces dernières, si elles sont égarées, divulguées ou oubliées, toute la sécurité s'effondre.
- Second point, la sécurité se base sur des arguments empiriques plutôt que théoriques.

La cryptographie symétrique et asymétrique ont des avantages et des inconvénients. Le tableau 1.2 dresse les avantages et Inconvénients de la cryptographie symétrique et asymétriques.

Table 1.2 : Avantage et Inconvénients de la cryptographie à clés symétriques et asymétrique.

| | Symétrique | Asymétrique |
|---|---|---|
| **Avantages** | - Rapidité (jusqu'à 1000 fois plus rapide) ;<br>- capacité de calculs moins intensive ;<br>- Facilité d'implantation sur hardware ;<br>- Taille de clé : 128 bits (⇒ 16 caractères : mémorisable). | - Distributions des clés facilitées, pas d'authentification ;<br>- Permet de signer des messages facilement ;<br>- Nombre de clés à distribuer est réduit par rapport aux clés symétriques. |
| **Inconvenient** | - Nombre de clés à gérer - Distribution des clés (authentification, confidentialité<br>- Certaines propriétés (p.ex. signatures) sont difficiles à réaliser | - Taille des clés<br>- Vitesse de chiffrement très lente |

1.10 Sécurité des algorithmes

La sécurité d'un algorithme est directement proportionnelle aux types de données à chiffrer, si le temps utilisé pour casser un algorithme est supérieur à la valeur des données,

ou si le temps utilise pour casser l'algorithme est supérieur au temps d'utilisation de cet algorithme, il n'y'a pas lieu de s'alarmer. Lars Knuden [8] a classé les méthodes de cassement des algorithmes par degré de sévérité.

- **Cassement total** : Le cryptoanalyste trouve la clé tel que $D_K(C) = M$
- **Déduction globale** : Le cryptoanalyste trouve un algorithme similaire à $D_K(C)$ sans avoir la clé K.
- **Déduction locale** : Le cryptoanalyste trouve le texte clair à partir d'un texte chiffré
- **Déduction de l'information** : Le cryptoanalyste trouve quelques informations sur la clé (quelques bits…)

Un algorithme est considéré inconditionnellement sécurisé, si quel que soit la quantité de texte chiffré dont dispose un cryptoanalyste, il ne lui est pas possible de recouvrir le texte clair. Au fait, seulement les algorithmes "à une seule utilisation" sont incassables, tous les autres, avec des ressources infinies, seront cassés en essayant toutes les clés possibles, l'une après l'autre et en vérifiant si le texte clair déchiffré a un sens. Cette méthode est appelée attaque en brute force.

La cryptographie, par contre traite des cryptosystèmes, dont la cryptanalyse est impossible. Un algorithme est à calcul sécurisé ou non tractable, s'il ne peut pas être cassé avec les ressources actuelles ou futures.

Remarquons que pour qu'un cryptosystème soit efficace, il doit palier aux trois problèmes majeurs suivants :

- Les clés devront être transmises, distribuées le plus secrètement possible, toutefois ceci ne devient plus évident, pour une compagnie dont les données véhiculent toute la planète, via internet, par exemple.
- Si une clé est compromise (volée, trouvée, cassée, …), Ève sera membre des émetteurs de messagerie.
- Si une clé unique est utilisée par chaque couple d'émetteur-destinataire, pour un réseau de n utilisateurs, le nombre de clés devient égal à « $n.\frac{n-1}{2}$ ». Donc de l'ordre de $O(n^2)$.

Le compromis réside dans la réduction de la taille de la clé en assurant une plus grande sécurité, la réduction de la taille assurera l'optimisation de la bande passante, donc des débits plus élevés ; et la sécurité assurera la minimisation des possibilités de casser les clés en un temps raisonnable.

Nous verrons dans le chapitre suivant que cette alternative est bien possible avec la cryptographie à base de courbes elliptiques, qui dans le même contexte résout le problème des clés privées transmises, en utilisant deux types de clés, une clé qui ne voyage nulle part dite clé privée, et une deuxième clé qui est transmise publiquement.

Pour que les protocoles fonctionnent, des fonctions de base sont implémentées, celles-ci sont appelées lors des étapes d'exécution du protocole, un peu à l'image des méthodes lors de l'utilisation des objets en programmation orientée objet.

## 1.11 Mesures de complexité

L'objet de la théorie de la complexité est :

- pour les algorithmes d'évaluer le nombre d'opérations élémentaires (complexité temporelle) et l'espace mémoire nécessaire (complexité spatiale) pour leur résolution ;
- pour les problèmes (de décision), de les classifier suivants leur niveau de difficulté.

La principale préoccupation, c'est le temps de calcul en fonction de la taille des données d'entrée dans les algorithmes car le problème de l'espace mémoire se pose de moins en moins avec le développement de la technologie.

La complexité d'un algorithme [14] se mesure par différents facteurs :

- Complexité des données : Le nombre de données nécessaires pou commencer une attaque,
- Complexité des calculs : Le temps nécessaire pour réaliser une attaque, appelé aussi facteur de travail,
- Espace de stockage : L'espace mémoire requis pour compléter une attaque.

En général, la complexité d'une attaque est calculée sur la base du plus petit facteur. Quelques attaques requièrent un compromis des trois facteurs, par exemple, une attaque rapide nécessite un espace de stockage très grand.

L'ordre de grandeur d'une attaque se mesure en magnitude d'opérations, donc l'ordre d'une attaque $=2^{128}$, nécessite $2^{128}$ opérations, donc si le cryptoanalyste dispose d'un million de processeurs effectuant un million d'opérations par seconde, il faudrait $10^{19}$ années pour trouver une seule clé (à peu près un billion de fois l'âge de l'univers).

Toutefois, il ne faut pas être confiant sur ses durées, car la technologie et les mathématiques font des avancées faramineuses, ce qui tendrait à limiter la confiance sur tel ou tel algorithme.

## 1.12 Fonction à un seul-sens

Ce sont des fonctions utilisées en cryptographie à clé publique, elles ont la particularité de contenir de très simples operateurs, et la spécificité de ne pas avoir de fonction inverse simple à estimer. C'est-à-dire que la fonction $f(x)$ est simple à calculer mais étant donné la fonction $f(x)$ , il est très difficile d'estimer $x$.

## 1.13 Fonction à un seul-sens avec trappe

Cette fonction est un cas particulier de la fonction précédente, sauf qu'elle dispose d'une trappe secrète, qui permet de déduire, étant donne $f(x)$. C'est l'exemple de démanteler une montre, qui est relativement facile, mais pour la remonter, c'est une vraie problématique si l'on ne dispose pas de la notice de montage.

## 1.14 Fonction de hachage à un seul sens

Cette fonction à plusieurs appellations telles que, fonction de compression, de contraction, signature digitale, checksum, intégrité, etc., donc c'est une fonction au cœur même des systèmes de cryptographie.

Une fonction de hachage est une fonction mathématique qui transforme une entrée (de type texte) de taille variable en une variable de taille fixe, en général plus petite, appelée valeur de hachage. Une simple fonction de hachage peut être un registre XOR, et qui donne en sortie une octet XOR de tous les octets de l'entrée.

Il existe aussi un autre code appelé aussi code d'authentification des messages (MAC), qui est basé sur la fonction de hachage en plus de l'inclusion de la clé.

Parmi les fonctions de hachage les plus répandues sont le MD5, le SHA-1 et le SHA-256 utilisé dans le protocole de Bitcoin. Le MD5, convertit un message de n bits, vers 128 bits, tandis que le SHA-1 produit 160 bits et le SHA-256 produit 256bits. Un autre domaine d'application des fonctions de hachage concerne la génération des séquences pseudo-aléatoires [14], comme illustré dans la figure 1.11 utilisée dans la génération des paramètres de la courbe elliptique SEC256r1 du NIST.

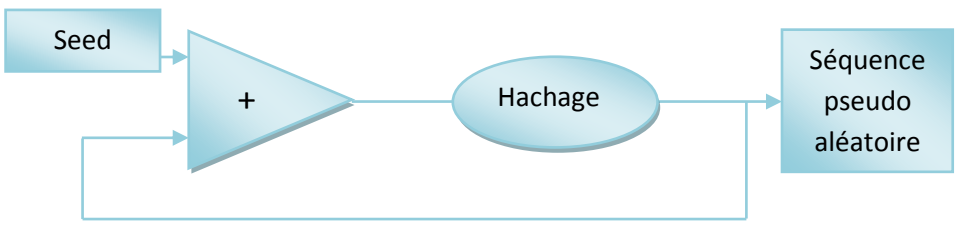



Figure 1.10 : Génération d'une séquence pseudo-aléatoire par une fonction de hachage.

## 1.15 Signatures numériques ou digitales

Cette théorie est similaire aux signatures manuelles, qui furent pendant une très longue durée de l'histoire humaine un acte de confiance, croyant que la signature ne pouvait être contrefaite.

Au niveau numérique, divers points émergent :

- La signature est authentique [15]. La signature doit convaincre le destinataire que le document a été signe délibérément.
- La signature est inoubliable. La signature est une preuve que le document n'a pas été signé par une tierce personne.
- La signature n'est pas réutilisable, la signature est une partie intégrante du document en question, qui ne peut être transféré à un autre document.
- Le document signé est inaltérable, il ne peut être altéré après signature.
- La signature est indéniable. Le document et la signature sont physiques. Le signataire ne peut nier avoir signé le document.

Certes le défi est de taille, car un fichier informatique peut être altéré, modifié, copié, sans laisser aucune évidence de la personne qui a fait les changements.

## 1.16 Pseudo-générateur de séquences aléatoires

Une séquence aléatoire est une séquence qui ne peut se répéter après une période de temps. Cette notion serait fantastique à réaliser pour la cryptographie, hélas, l'utilisation des ordinateurs, qui sont des machines à états finis, certainement, un très large nombre d'états, mais finis, contraint les possibilités de non périodicité à une période très grande. Le problème réside sur le fait que si les nombres générés présentent la simple corrélation, des résultats bizarres ou imprévisibles commencent à être obtenus et tout le système de cryptographie devient fragile.

Donc, au lieu d'avoir des générateurs aléatoires, la cryptographie se contente de générateurs pseudo-aléatoires.

## 1.17 Propriétés des générateurs

Pour qu'un générateur soit adopté en cryptographie, il doit posséder les propriétés suivantes :

- Il doit passer tous les tests statistiques, avec succès.
- Il doit être imprédictible, i.e., nul humain / machine ne doit prédire le prochain nombre à générer.

- Il ne doit pas être reproduit efficacement. C'est-à-dire si le même générateur est exécuté deux fois de suite, avec les mêmes entrées, il ne doit pas y avoir de corrélation entre les deux séquences générées.

## 1.18 Cryptosystéme à clé publique

Les différents problèmes de distribution des clés sont solutionnés par la cryptographie à clé publique. Ce concept a été introduit par Whitfield Diffie et Martin Hellman en 1975. Il est maintenant prouvé que les services secrets britanniques avaient fait cette même découverte plusieurs années avant Diffie et Hellman et avaient protégé ce secret militaire sans en faire aucune utilisation [16].

Le chiffrement / déchiffrement à clé publique est un procède asymétrique, se basent sur deux clés totalement différentes, la première clé dite « clé publique » chiffre, tandis que la deuxième clé dite « clé secrète » déchiffre.

La force des algorithmes de cryptographie à clé asymétrique est qu'aucune des deux clés utilisées, ne peut se déduire de la deuxième clé. Le revers de la médaille est que si une personne crypte ces données avec sa clé de chiffrement publique, il ne pourra déchiffrer ce même message chiffré avec sa deuxième clé secrète, car cette dernière n'est utilisée que pour déchiffrer des messages issus du chiffrement réalisée avec une autre clé publique.

Il existe actuellement différents algorithmes se basant sur les clés asymétriques, tels que :

- Algorithme d'El GamalTaher [17] ;
- Algorithme RSA de par ses inventeurs Ron Rivest, Adi Shamir et Leonard Adelman [18];
- Algorithme se basant sur les courbes elliptiques, de par son inventeur Neil Koblitz[19].

### 1.18.1 Crypto système d'El Gamal

Ce crypto système est une application ingénieuse de Taher El Gamal [17] qui a utilisé la fonction à un sens avec trappe de Diffie-Hellman, la rendant ainsi un crypto système a clé publique. Le travail d'ElGamal a inspiré un grand intérêt de la communauté scientifique dans le domaine de la recherche et de l'appliqué.

Une des raisons les plus prometteuses qui ont donne de l'élan aux travaux d'El Gamal, concernent l'utilisation du problème de Diffie-Hellman, qui est considéré comme un problème aussi difficile que le problème du logarithme discret [20].

**Algorithme 1.1 : Algorithme de chiffrement à clef publique d'El Gamal [17].**

**Génération de la clef de Bob :**

1) Bob choisit un groupe cyclique G (typiquement (GF(p), ×) ou bien un groupe défini sur une courbe elliptique), d'élément générateur g.
2) Bob choisit un nombre aléatoire x compris entre 0 et l'ordre du groupe G noté |G|.
3) Bob calcule $h = g^x$ .
4) Bob publie sa clef publique qui est le triplet (G, g, h).

**Alice chiffre son message m :**

1) Alice choisit un nombre aléatoire y et calcule $c1 = g^y$ .
2) Alice calcule le secret commun $s = h^y$ .
3) Alice attribue à son message m une valeur m0 appartenant à G.
4) Alice calcule $c_2 = m' \times s$.
5) Alice envoie à Bob le couple ($c_1$, $c_2$).

**Bob déchiffre le message d'Alice ($c_1$, $c_2$) :**

1) Bob calcule le secret commun $s = c^x_1$ .
2) Bob calcul $m0 = c_2 \times s^{-1}$ , le résultat est correct puisque $c_2 \times s^{-1} = m' \times s \times s^{-1} = m'$ .

1.18.2 Crypto système R.S.A[20]

Le RSA est considéré comme l'un des meilleurs systèmes de la cryptologie à clé asymétriques, de par ses inventeurs Rivest, Shamir et Adelman [20], le RSA a été initialement introduit une comme réalisation publique dans les fonctions à sens unique avec trappe. Il est toujours considéré comme le cryptosystème asymétrique le plus utilisé, notamment pour échanger des données confidentielles sur internet.

L'algorithme est basé sur l'hypothèse qu'il est extrêmement difficile d'effectuer la factorisation d'un nombre entier en facteurs premiers. Certes, la factorisation n'est pas la seule méthode pour casser RSA, mais pour l'instant, il n'existe pas d'autre attaque qui soit suffisamment efficace. RSA est proposé avec 3 algorithmes qui sont conçus respectivement pour :

- la génération de clés ;
- la signature numérique ;
- la vérification de signature.

1.18.2.1 Génération des clés

Le RSA fonctionne à partir de deux nombres premiers. Ces deux nombres doivent être très grands et calculer la clé privée et la clé publique comme dans l'algorithme suivant:

**Algorithme 1.2 : Génération des clés [7]**

Entrées : Deux nombres premiers p et q.
Sorties : Une clé privée d et une clé publique e.
Prendre un nombre aléatoire p et q.
Si p = q alors Prendre un nombre aléatoire p et q.
Sinon N = p q.
Fin Si
Calcul $\varphi_N = (p-1)(q-1)$ .
Prendre un nombre aléatoire e dans l'intervalle [1, $\varphi_N$].
Si pgcd(e, $\varphi_N$) ! = 1 alors
Prendre un nombre aléatoire e dans l'intervalle [1, $\varphi_N$].
Sinon Calcule $d \equiv e^{-1} \bmod (\varphi_N)$.
Fin Si

### 1.18.2.2 Chiffrement du message [7]

Supposons maintenant que les intervenants A et B possèdent chacun son module RSA et ces clés NA, $e_A$ ,$d_A$ pour A et NB, $e_B$, $d_B$ pour B. Si A veut envoyer un message M à B, il peut procéder comme dans l'algorithme suivant:

**Algorithme 1.3 : Chiffrement d'un message [7]**

Entrées : Un message clair M et la clé publique **($N_B$., $e_B$).**
Sorties : Un message chiffré C.
Transformer le message en un nombre entier M de l'intervalle **[0, $N_B$. −1]**
Calculer **$C \equiv M^{e_B} \bmod N_B$. et $C < N_B$**
Envoyer le message C.

### 1.18.2.3 Déchiffrement du message

B a reçu un message chiffré C de la part de A. Alors B peut le déchiffrer en utilisant sa clé secrète dB comme dans l'algorithme suivant :

**Algorithme 1.4 : de Déchiffrement d'un message [7]**

Entrées : Un message chiffré C et la clé privée **($N_B$., $d_B$.).**
Sorties : Un message clair M.
Calculer **$M = C^{d_B} \bmod N_B$**
Retourner le message M.

### 1.18.3 Signature d'un message

Supposons que A veut envoyer un message M à B. Comment B peut-il être sûr que le message reçu a bien été envoyé par A. Pour résoudre ce problème, RSA peut être utilisé aussi bien pour transmettre un message que pour le signer. Tout d'abord, A utilise la clé publique ($N_B$, $e_B$) de B pour transmettre le message chiffré C et produire une signature S du message à l'aide de sa propre clé privée ($N_A$, $d_A$) **.**

**Algorithme 1.5 : Signature d'un message [7]**

Entrées : Un message clair M et clé privée (**$N_A$, $d_A$**) , fonction de hachage h .
Sorties : Un message chiffré C et sa signature S.
A transforme le message en un nombre entier M de l'intervalle [**0 , $N_B$ − 1**].
Calculer **D = h(M)**

Calculer $\mathbf{S \equiv D^{dA} \bmod N_A}$
Retourner le message C et la signature S.

**1.18.3.1** Vérification d'une signature **[7]**

Pour vérifier la signature, nous utilisons alors l'algorithme suivant :

**Algorithme 1.6 : Vérification d'une signature[7]**

Entrées : Un message chiffre C, une signature S et la clé publique $\mathbf{(N_A,e_A)}$, et la clé Privée $\mathbf{(N_B, e_B)}$,
Sorties : Vérification d'une signature.
B déchiffre le message C : Calculer $\mathbf{M \equiv C^{dB} \bmod N_B}$
B Calcule $\mathbf{D' \equiv S^{eA} \bmod N_A}$ et **D=h(M)**
Si $\mathbf{D^{'} = D}$ Alors la signature est vérifiée.

1.18.4 Considérations de sécurité pour l'utilisation de RSA [21]

- Il est fondamental d'utiliser CRT (Chinese Remainder Theorem[21] dans RSA pour augmenter la vitesse de déchiffrement.
- Il est aussi fondamental d'utiliser le schéma de cryptage RSAES-OAEP au lieu de RSA seul pour les opérations de chiffrement et déchiffrement. Cette méthode a eté introduite par Bellare et Rogaway in 1994 [22] .
- RSA–REACT: An Alternative to RSA–OAEP[23]
- Il est fondamental d'utiliser une clé RSA 2048-bit comme un minimum pour l'utilisation de la cryptographie RSA dans les nouvelles applications (à partir de 2010).

1.19 Problème Logarithme discret [24]

Le premier système du logarithme discret (DL) était le protocole d'échange de clés Diffie et Hellman en 1976. En 1984, ElGamal décrit DL chiffrement à clé publique et schémas de signature. Nous allons maintenant décrire l'algorithme qui permet la génération de paramètres de DL.

**Algorithme 1.7 : génération de paramètres de DL**[24]

Entrées: paramètres t, l.
Sorties: paramètres **(p,q,g)**
1. Sélectionner t un nombre premier q de t-bit et un nombre premier **p** de l-bit tel que **q** divise p − 1.
2. choisir un élément g en ordre q.
3. Sélectionnez arbitraire $\mathbf{h \in [1, p-1]}$ et Calcule $g = h^{(p-1)/q} \bmod p$
4. Si g = 1 alors aller à la phase 3.
5. Retourner **(p,q,g)**

Nous allons maintenant décrire l'algorithme qui permet la génération des clés DL comme le montre la figure ci -dessous.

| **Algorithme 1.8 : génération des clés DL**[24] |
| --- |
| Entrées: paramètres **(p,q,g)**<br>Sorties: clé publique **y**, clé privée x.<br>1. Sélectionner x ∈ R [1, q − 1].<br>2. Calcule $\boldsymbol{y = g^x \bmod p}$<br>3. Retourner **(x, y).** |

1.20 Les crypto systèmes à bases de courbes elliptiques (CCE)

Il s'agissait de concevoir un crypto-système asymétrique basée sur des courbes elliptiques définies sur des corps finis. Le problème mathématique qui a motivé leur utilisation est leur résistance au problème du logarithme discret.

La cryptographie à base de courbes elliptiques (ECC) [25,26] dépend de l'arithmétique impliquée dans les opérations effectuées sur les points de la courbe elliptique. La figure 1.11 illustre les modules mathématiques nécessaires dans la signature digitale basée sur les courbes elliptiques (Elliptic Curve Digital Signature Algorithm : ECDSA), par exemple.

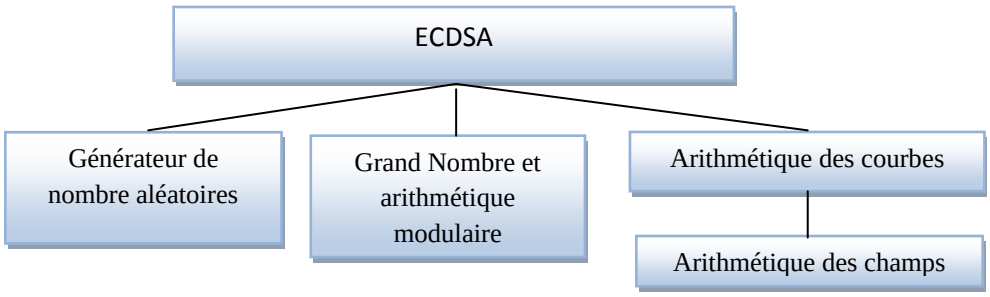



Figure 1.11 : Différents axes des mathématiques impliqués dans l'ECDSA.

Avant d'entamer toute définition du cryptosystème basé sur les ECC, il y'a lieu de positionner le problème de la difficulté du problème du logarithme discret dans ces courbes elliptiques qui est essentielle pour la sécurité de ces systèmes de cryptographie.

1.21 La comparaison de performance entre ECC et RSA

Ces deux types de clés principaux partagent la même propriété importante d'être des algorithmes asymétriques (une clé pour chiffrer et une clé pour déchiffrer). Cependant, selon Global Sign, l'ECC peut offrir le même niveau de puissance de chiffrement pour des clés beaucoup plus courtes, offrant ainsi une meilleure sécurité tout en réduisant les besoins en calculs.

Le facteur de différenciation clé entre l'ECC et RSA est la taille de la clé comparé à la force de chiffrement. Comme le montre le tableau 1.3 et le graphique 1.12.

Tableau 1.3 : avantages et Inconvénients de la cryptographie à clés asymétriques dans le cas du RSA et ECC.

| | Asymétrique (RSA) | Asymétrique (ECC) |
|---|---|---|
| **Avantages** | - Cryptosystème largement répandu<br>- Nombreuses études au sujet de sa sécurité | - Taille de clé inférieure pour une sécurité égale<br>- La taille des clés croit moins vite que le RSA si on souhaite une meilleure sécurité<br>- Utilisation pour systèmes embarqués<br>- Calculs moins lourds que l'exponentiation<br>- Utilisation mémoire moindre<br>- Cryptanalyse par algorithme exponentiel |
| **Inconvenients** | - Opérations de dé/chiffrement très inégales en termes de temps de calcul<br>- Cryptanalyse par algorithme sous exponentiel | - Complexe<br>- Peu de développement sur des systèmes à grande échelle (mais tend à changer)<br>- Travaux d'optimisation essentiellement destinés aux systèmes mobiles |

ECC peut avoir le même niveau de sécurité que RSA avec une clé beaucoup plus courte. Partant de ce constat, différents protocoles ont été adaptés aux courbes elliptiques car en plus de leur résistance au DLP, les tailles de clés utilisées sont moindres pour un même niveau de sécurité que celles dans le cas du problème de factorisation .Le tableau ci-dessous dresse la relation entre taille des clés et la sécurité.

Tableau 1.4 : Comparaison des tailles de clés entre ECC et RSA.

| **Bits de sécurité** | **RSA** | **ECC** |
|---|---|---|
| 80 | 1024 | 160 |
| 112 | 2048 | 224 |
| 128 | 3072 | 256 |
| 192 | 7680 | 384 |
| 256 | 15360 | 512 |

La figure 1.12 permet de comparer facilement les tailles de clés entre ECC et RSA (128 bits).

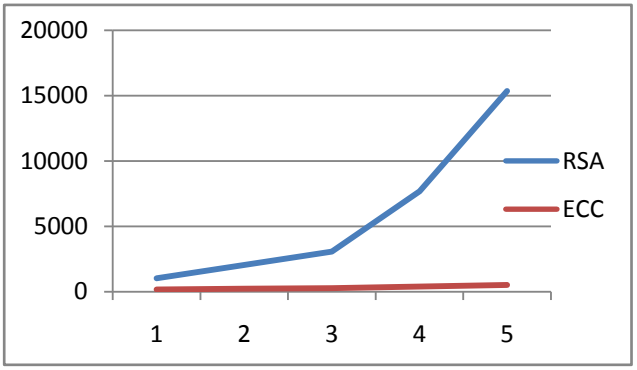



Figure 1.12 : Comparaison des tailles de clés entre ECC et RSA.

### 1.21.1 Problème du logarithme discret pour les courbes elliptiques

Le problème du calcul de logarithmes discrets sur courbes elliptiques est à la base de la sécurité de nombreux protocoles cryptographiques.

Le problème du logarithme discret [27] dans les courbes elliptiques (ECDLP), est défini comme suit :

Étant donne une courbe $E$ définie sur $GF(q)$, et un point $P \in E\big(GF(q)\big)$ d'ordre $n$, et un point apparetenant à $\langle P \rangle$ , trouver l'entier $l \in [0, n-1]$ tel que : $Q = l \times P$ .

L'entier $l$ est appelé logarithme discret de $Q$dans la base$P$, dénoté $l = log_p Q$ .

Les paramètres de la courbe elliptique en tant que support cryptographique doivent être choisis avec soin, en vue de résister à toutes les attaques connues sur l'ECDLP. L'algorithme naïve connue pour solutionner ce problème appelé « recherche exhaustive » par laquelle on calcule la séquence de points $P, 2P, 3P, 4P, ..$, jusqu'à atteindre Q. La complexité temporelle est égale à $n$ étapes, en moyenne égale à $n/2$ étapes. Toutefois, cette méthode devient obsolète, lorsque $n$ est suffisamment large ($n \geq 2^{80}$).

La meilleure attaque connue sur l'ECDLP est la combinaison de l'algorithme de Pohlig-Hellman et celui de Pollard rho [28], qui présentent une complexité de $O(\sqrt{p})$, $p$ étant un large diviseur de $n$. Pour pallier à cette attaque, les paramètres de la courbe elliptique doivent être choisies de telle façon que $n$ soit divisible par un très grand nombre premier $p$, tel que $\sqrt{p}$ soit très large c'est-à-dire $p > 2^{160}$, ceci rend l'attaque non faisable en un temps de calcul raisonnable.

Si d'autres considérations sur le choix des paramètres sont aussi prises en charge, il s'avérerait que l'ECDLP, présente un réel défi à la cryptanalyse actuelle [8].

L'on doit noter qu'il n'ya aucune preuve que le ECDLP soit intraitable, c'est-à-dire que personne ne peut prouver qu'il n'existe pas d'algorithme efficace pour solutionner ce problème.

Depuis la proposition de l'utilisation des courbes elliptiques comme support de cryptographie, en 1985 par Neil Koblitz [19 ,29,30], aucun algorithme sous-exponentiel n'a pu être découvert en vue de solutionner le ECDLP.

Une liste non exhaustive des attaques les plus connues contre l'ECDLP sont :

- Attaque de Pollig-Hellman,
- Attaque de Pollard rho,
- Attaque parallélisée de Pollard rho,
- Attaque de Pollard rho par l'utilisation des automorphismes,

- Attaque par index de calcul (Index-calculus),
- Attaques isomorphismes,
- Attaques sur les courbes anormales à base des champs premiers,
- Attaques couplées de Weil & Tate.

Les différentes attaques sont exposées dans [3,8, 20, 30] avec plus de détails.

### 1.21.2 Génération des paramètres d'une courbe elliptique

L'implémentation d'un système ECC nécessite un nombre de décisions à différents niveaux du système de chiffrement, ces choix dépendent de la plateforme sur laquelle sera implémenté le système et des buts que l'on veut atteindre. Pour réaliser un ECC il faut donc implanter l'ensemble de ces opérateurs.

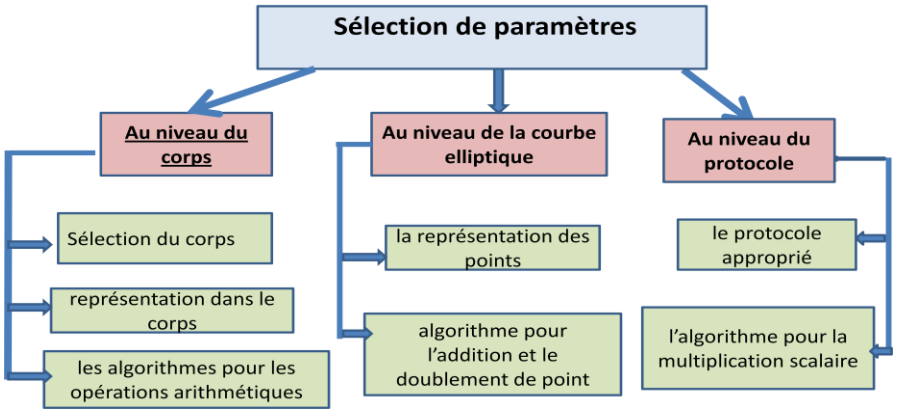



Figure 1.13 : L'architecture des différents niveaux de sélection de paramètre de la courbe.

Pour éviter de choisir une courbe malicieuse, ou facilement attaquable, deux organismes sont connus pour breveter des courbes elliptiques aux propriétés intéressantes : le NIST et CERTICOM, tous les deux ont établi une liste de courbes assurées, du moins pour le moment, pour établir les différentes procédures de chiffrement / déchiffrement au niveau des instances gouvernementales [3,4]. Ils proposent tous les deux d'utiliser des courbes obéissant à l'équation de Weierstrass qui utilisent les paramètres a, b et p , par contre Satoshi a utilisé une courbe de Koblitz un peu spéciale qui n'appartient pas à ces deux organismes, la chapitre 4 essaye d'enlever ce mystère ou plutôt à trouver une explication à son choix.

### 1.21.3 Génération de la paire de clés asymétriques

Une paire de clés à base des courbes elliptiques est définie par un domaine de paramètres définis comme suit :

D= (q, FR, a, b,{s: domain_parameter_seed}, G, n, h) [7]

Le tableau 1.5 nous illustre le domaine des paramétres.

Tableau 1.5 : domaine des paramètres.

| | |
|---|---|
| q | la taille du corps de base |
| FR | indique le type de base utilisée ; |
| a et b | deux éléments du corps qui définissent l'équation de la courbe ; |
| domain_parameter seed | la graine utilisée pour définir D, c'est un paramètre optionnel qui est utilisé quand la courbe a été générée de manière aléatoire et que cela peut être vérifié ; |
| G | un point de base de la courbe d'ordre premier n |

Il faut également spécifier la fonction de hachage H qui devra être utilisée.

- ✓ La clé publique Q est choisie dans l'ensemble du groupe $\langle P \rangle$ généré par $P$.
- ✓ La clé prive $d$ correspond à la valeur $d = log_p Q$.

L'algorithme générant la paire de clé doit s'assurer que les paramètres du domaine $D$ soient valides.

**Algorithme 1.9 : génération de la paire de clé [7]**

Entrée : $D = (q, FR, S, a, b, P, n, h)$
Sortie : Clé publique$Q$, clé privée $d$

- Sélectionner $d \in_R [1, n-1]$
- Calculer $Q = dP$
- Retourner $(Q, d)$

Notons que calculer la clé privée $d$ de la clé publique $Q$ est le problème de l'ECDLP, donc il est impératif de choisir les paramètres de la courbe judicieusement.

### 1.21.4 Validation de la clé publique

Cette propriété est nécessaire, surtout, lors du protocole d'établissement des clés de Diffie-Hellman, ou une entité A dérive le secret k en combinant sa clé prive et la clé publique reçu d'une autre entité B, et utilisant k dans un protocole de clé symétrique (chiffrement et authentification de message). B peut tricher et sélectionner une clé publique invalide de telle façon que l'utilisation de $k$ donne des informations sur la clé privé de A.

**Algorithme 1.10 : validation de la clé publique [7]**

Entrée : Domaine des paramètres$D = (q, FR, S, a, b, P, n, h)$, et la clé publique$Q$.
Sortie : Accepter or rejeter la validité de $Q$

- Verifier que $Q \neq \infty$.
- Vérifier que $x_Q$ et $y_Q$ sont bien des éléments représentatifs de $GF(q)$ (i.e. : entiers dans l'intervalle $[0, q-1]$, si GF(Q) est un champ premier, et une chaine binaire de taille m si GF(q) est un champ binaire d'ordre$2^m$)
- Vérifier que $Q$ satisfait la courbe elliptique définie par a et b.
- Vérifier que $n.Q = \infty$
  - Si une des étapes est fausse, retourner (clé non valide), sinon retourner (clé valide)

Parfois la vérification de $n.Q = \infty$, devient trop lente, et peut être omise dans certains protocoles.

## 1.22 Longueur des clés de chiffrement

La taille des clés de chiffrement représente la nouvelle stratégie qui définit le choix de l'algorithme de chiffrement / déchiffrement ; dans cette orientation les nouveaux algorithmes se basent sur des fondements mathématiques, en vue de réduire la clé de chiffrement en augmentant la difficulté de la cryptanalyse, comme illustré dans le tableau 1.6.

Tableau 1.6. : Comparatif des tailles des clés de divers algorithmes de cryptographie [7].

| Bits de sécurité | Algorithmes Symétriques | IFC (e.g., RSA) | ECC (e.g., ECDSA) |
|---|---|---|---|
| 80 | 2TDEA | k = 1024 | f = 160-223 |
| 112 | 3TDEA | k = 2048 | f = 224-255 |
| 128 | AES-128 | k = 3072 | f = 256-383 |
| 192 | AES-192 | k = 7680 | f = 384-511 |
| 256 | AES-256 | k = 15360 | f = 512+ |

- F et k représentent les tailles de clés équivalentes

Le chiffrement / déchiffrement actuel est basé sur deux types de chiffrement, le chiffrement symétrique ou les intervenants utilisent la même clé pour chiffrer et déchiffre et le chiffrement a clés asymétriques, ou chacun des deux parties possède une clé privé et une clé publique qui peut être envoyé publiquement par les canaux de communication.

## 1.23 La cryptanalyse

L'objectif de la cryptographie est de garder au secret la clé et le message, la cryptanalyse à un objectif tout à fait inverse.

Donc Il s'agit de l'étude des mécanismes théoriques ou techniques visant à briser (casser) un algorithme de chiffrement, c'est-à-dire le fait de retrouver le message M à partir de C, sans connaître la clé K a priori. Dans certains cas, il s'agira également de retrouver cette clé K. Le must serait de déchiffrer le message sans la clé, cette notion est appelée 'attaque' du système de cryptographie induisant un 'compromis' : perte de la clé.

La première hypothèse émise par A. Kerckhoffs au XIX$^{ème}$ siècle [32], que la sécurité doit résider entièrement dans la clé. En supposant que le cryptoanalyste possède tous les autres détails utilisés lors de la cryptographie (algorithme, implantation, etc...).

La cryptanalyse se divise en 4 types d'attaques principales, détaillées dans ce qui suit.

1.23.1 Les 4 attaques cryptanalytiques

Il en existe 4 grands types, chacun pouvant utiliser différentes techniques comme le montre la figure 1.14.

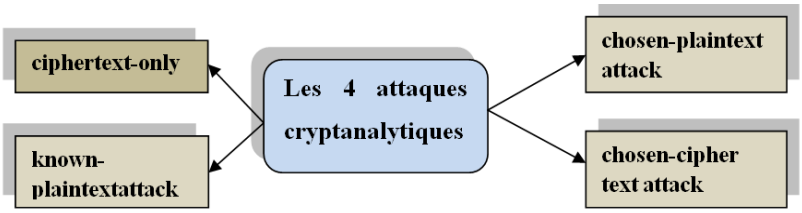



Figure 1.14 : schéma des 4 attaques cryptanalytiques.

**1.23.1.1** Attaques à l'aide de textes chiffrés seulement (ciphertext-only) (COA)

On suppose que le cryptoanalyste dispose de différents plusieurs textes chiffrés par la même clé. Ce type d'attaque concerne le possible recouvrement du texte original, ou de la clé, ce qui serait meilleur, en vue de déchiffrer les textes chiffres futurs.

Étant donné : $C_1 = E_k(M_1), C_2 = E_k(M_2), \dots C_i = Ek_K(M_i)$

Déduire : $M_1, M_2, \dots M_i;\ K$ ;

Avec l'algorithme permettant de générer : $C_{i+1} = E_K(M_{i+1})$

Cette technique ne fonctionne que pour la plupart des chiffrements classiques basiques, les seuls permettant l'utilisation de l'analyse de fréquence. On peut aussi également procéder par force brute pour briser de tels chiffrements.

1.23.1.2 Attaques à base de textes clairs-connus (known-plaintext attack) (KPA)

Le cryptoanalyste a accès, non seulement au texte chiffré [8] mais aussi au texte clair des messages. Son travail se réside à déduire la ou les clés utilisées pour le chiffrement ou l'algorithme utilisé pour le chiffrement ou déchiffrement utilisant la/les mêmes clés.

Etant donne : $M_1, C_1 = E_k(M_1), M_2, C_2 = E_K(M_2), \dots M_i, C_i = E_K(M_i)$

Déduire : $K$

Avec l'algorithme permettant de générer : $M_{i+1}$

A partir de : $C_{i+1} = E_K(M_{i+1})$

Etant donné un texte chiffré et un fragment de texte clair associé, on recherche le texte clair restant et/ou la clé. On utilise la technique dite de la cryptanalyse linéaire (1993).

1.23.1.3 Attaques à base de textes clairs choisis (chosen-plaintext attack) (CPA)

Le cryptoanalyste a accès, non seulement au texte chiffré mais aussi au texte clair des messages, et dispose du choix du texte à chiffrer, ce qui diffère de la section suivante, car

le cryptoanalyste peut être spécifique sur le bloc à chiffrer, par exemple les segments donnant plus d'information sur la clé.

Son « travail » se réside à déduire la ou les clés utilisées pour le chiffrement ou l'algorithme utilisé pour le chiffrement ou déchiffrement de nouveaux messages utilisant la/les mêmes clés.

Étant donné : $M_1, C_1 = E_k(M_1), M_2, C_2 = E_K(M_2), \dots M_i, C_i = E_K(M_i)$

Le cryptoanalyste a le choix d'utiliser « $M_1, M_2, \dots M_i$ » , pour déduire « $K$ ».

Avec l'algorithme permettant de générer : $M_{i+1}$

A partir de : $C_{i+1} = E_K(M_{i+1})$

1.23.1.4 Attaques à base textes chiffrés choisis (chosen-ciphertextattack) CCA

Le cryptoanalyste peut choisir différents textes chiffrés, en vue de déchiffrer, et dispose de la boite noire qui chiffre automatiquement. Son rôle se réside à déduire la clé,

Étant donné : $C1, M1 = D\text{k}(C1), C2, M2 = D\text{k}(C2), \dots C\text{i}, M\text{i} = D\text{k}(C\text{i})$

Déduire : $K$

Cette attaque concerne la cryptographie à clés publiques, mais peut efficace contre un algorithme à clé symétrique.

Etant donné la capacité de déchiffrer un fragment de texte chiffré choisi arbitrairement, on recherche la clé.

1.23.2 Quelques autres techniques

De cette manière, il est possible de retrouver des implantations d'algorithmes ou des données stockées en mémoire. On obtient ainsi une image de circuit qu'on peut utiliser afin de le rétro-concevoir. En général, ces attaques sont destructives et le composant n'est donc plus utilisable par la suite. Cependant, si un attaquant ou une attaquante a sa disposition plusieurs exemplaires d'un même composant, il peut réaliser une attaque active sur un exemplaire afin de récupérer par exemple un algorithme, puis utiliser les autres afin de réaliser une attaque passive vérifiant les résultats obtenus.

Il paraît très complexe de se prémunir face aux attaques actives. Dans certaines situations, il est possible d'implanter un détecteur de changement de température ou d'une autre perturbation, qui provoquera l'arrêt du composant en cas d'attaque. Une détection d'injection de fautes peut également être ajoutée au composant.

Afin de se protéger des attaques passives, il est nécessaire de comprendre les fuites d'information que peut provoquer l'exécution d'un algorithme sur un composant. Par

exemple, il est possible de privilégier des opérations en temps constant qui ne dépendent pas d'une valeur secrète.

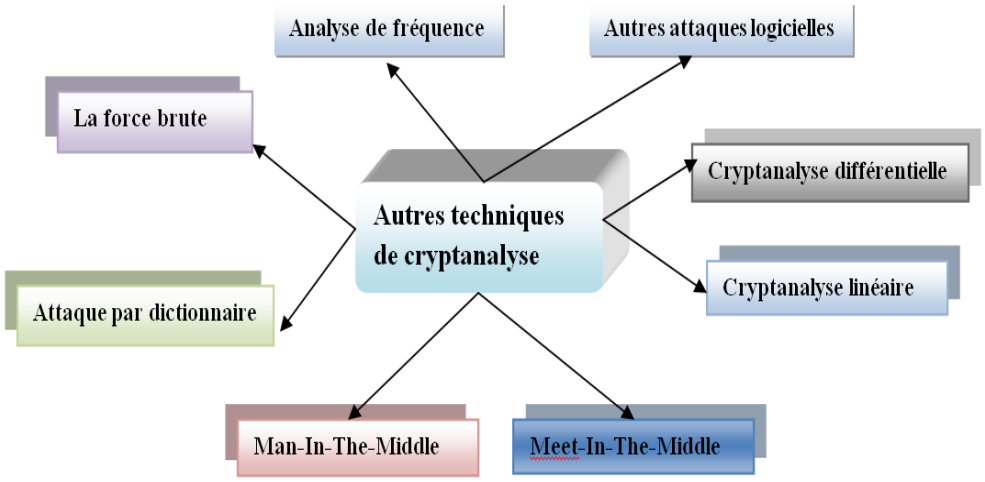



Figure 1.15 : autre techniques d'attaques.

1.23.2.1 La force brute

Le principe est ici de tester toutes les clés possibles de manière exhaustive. La limite maximale est donnée par $T^N$ avec T représentant la taille de l'alphabet, N est la taille de la clé.

Par exemple, pour une clé de 128 bits, il y a $2^{128}$ clés possibles. Cette technique n'est "efficace" que pour des textes chiffrés avec une clé relativement courte. Une clé de 56 bits n'est plus à utiliser aujourd'hui si l'objectif premier est la confidentialité. Les algorithmes symétriques actuels utilisent en standard des clés variant entre 112 (3DES) et 256 bits (AES). *Le tableau 1.7 nous donne* Temps de calcul pour une taille de clé donnée.

Tableau 1.7 : Temps de calcul pour une taille de clé donnée.

| Taille de la clé (bits) | nombre de clés alternatives | Temps requis au chiffrement/µs | Temps requis à $10^6$ au chiffrement/µs |
|---|---|---|---|
| 32 | $2^{32}=43.10^9$ | $2^{31}$ µs=35.8 mn | 2.14 ms |
| 56 | $2^{56}=72.10^{16}$ | $2^{55}$ µs =1142 ans | 10.01 heures |
| 128 | $2^{128}=3,4.10^{38}$ | $2^{127}$ µs=5,4.$10^{24}$ années | 5,4.$10^{24}$années |
| 168 | $2^{168}=3,7.10^{50}$ | $2^{167}$ µs=5,9.$10^{36}$ années | 5,9.$10^{36}$années |
| 26 caractères | 26!=4.$10^{26}$ | 2.$10^{26}$ µs=6,4.$10^{12}$ années | 6,4.$10^{12}$années |

Le tableau ci-dessous donne la comparaison entre les différents systémes cryptographiques.

Tableau 1.8 : Sécurité fournie selon la taille de la clé.

| Niveau | Protection | Symétrique | Asymétrique | Logarithme Discret Clé | Logarithme Discret groupe | ECC | Hash |
|---|---|---|---|---|---|---|---|
| 1 | -Attaques en temps réel par des individus. -Uniquement acceptable pour la taille de l'étiquette d'authentification | 32 | - | - | - | - | - |
| 2 | -Protection à très court terme contre les petites organisations. - Ne doit pas être utilisé pour la confidentialité dans les nouveaux systèmes | 64 | 816 | 128 | 816 | 128 | 128 |
| 3 | -protection à court terme contre les organisations de taille moyenne. -Protection à moyen terme contre les petites organisations | 72 | 1008 | 144 | 1008 | 144 | 144 |
| 4 | - Protection à très court terme contre les agences. -Protection à long terme contre les petites organisations. -Plus petit niveau d'usage général, 3DES à 2 clés limité à $10^6$ textes en clair / cryptés, protection de 2009 à 2020. | 80 | 1248 | 160 | 1248 | 160 | 160 |
| 5 | Niveau standard hérité. 3DES à 2 clés limité à $10^6$ textes en clair / chiffrés, protection de 2009 à 2020. | 96 | 1776 | 192 | 1776 | 192 | 192 |
| 6 | Protection à moyen terme. Protection 3DES à 3 touches de 2009 à 2030. | 112 | 2432 | 224 | 2432 | 224 | 224 |
| 7 | protection à long terme. Application générique -recommandation indépendante, protection de 2009 à 2040. | 128 | 3248 | 256 | 3248 | 256 | 256 |
| 8 | Avenir prévisible. bonne protection contre les ordinateurs quantiques. | 256 | 15424 | 512 | 15424 | 512 | 512 |

**Remarque :**

Un algorithme est dit cassé quand il est possible de retrouver la clé en effectuant moins d'opérations qu'en utilisant la force brute.

Un algorithme cassé est “moins sûr”, mais pas inutile pour autant. Il faudra veiller à son utilisation si on souhaite assurer la confidentialité des données, mais rien n’empêche de l’utiliser à d’autres fins.

1.23.2.2 Attaque par dictionnaire

Lorsque la clé est un mot (Par exemple un mot de passe), on peut tenter de court-circuiter la Force Brute. Le principe est ici d’utiliser un recueil de mots possibles (le dictionnaire), et de tester tous les mots de ce dictionnaire.

1.23.2.3 Analyse de fréquence

Cette analyse repose sur l’obtention d’indices précieux : quelle est la langue utilisée ? Quel est le thème du texte ? Il faut toutefois que la taille de K soit inférieure à la taille de C, au risque de ne pas permettre l’unicité de la solution. Il faut aussi que le texte C soit suffisamment long pour être représentatif. Et enfin que l’algorithme utilisé soit une substitution simple (mono- ou poly-alphabétique).

1.23.2.4 Cryptanalyse différentielle

Il s’agit de l’étude (modélisation) des transformations subies par le message durant son passage dans l’algorithme de chiffrement. Le principe est de modéliser ce qu’une modification en entrée induira sur le résultat de l’algorithme comme le montre la figure ci-dessous.

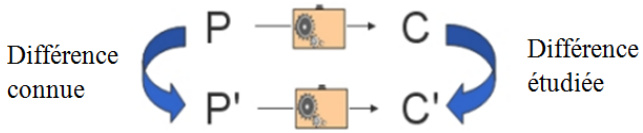



Figure 1.16 – Cryptanalyse différentielle.

1.23.2.5 Cryptanalyse linéaire

Le but est d’effectuer une approximation linéaire de l’algorithme de chiffrement. Il n’y a ici aucune possibilité de choisir le texte clair à chiffrer, on dispose tout au plus d’un ensemble de couples (M, C).

On tente alors de découvrir une expression de la forme :

$M_{i1} \oplus M_{i2} \oplus \ldots \oplus M_{iu} \oplus C_{j1} \oplus C_{j2} \oplus \ldots \oplus C_{jv} = 0$

avec Mi représentant le i $^{eme}$bit de M = [$M_1$, $M_2$, . . . $M_u$] et $C_j$ représentant le j $^{eme}$ bit de C = [$C_1$, $C_2$, . . . $C_v$]. Si c’est le cas, et selon la probabilité d’occurrence de l’expression, on peut déduire des faiblesses de l’algorithme (en termes de transformations aléatoires).

**Remarque :**

Cryptanalyses différentielle et linéaire sont des outils parfaits pour comparer la résistance de divers chiffrements. Aucun algorithme cryptographique n’est dit valable s’il ne résiste pas à ce type de cryptanalyse. La résistance à ces attaques ne signifie pas

résistance contre toutes les autres méthodes inconnues. Sans une connaissance approfondie des techniques de cryptanalyse, il n'est pas possible de créer un algorithme de chiffrement performant, sûr et robuste.

1.23.2.6 Meet-In-The-Middle (MITM)

Cette attaque est souvent illustrée par l'attaque ayant démontré les faiblesses du DES. Pour rappel, le 2DES fonctionne de la manière suivante :

$C = E_{K2}(E_{K1}(M))$

On suppose que l'attaquant dispose d'un couple (M,C). Il chiffre M avec les $2^{56}$ clés f d'un coté et déchiffre C avec les $2^{56}$ clés g de l'autre. Lorsque $E_f(M) = D_g(C)$, il sait qu'il a découvert les 2 clés utilisées.

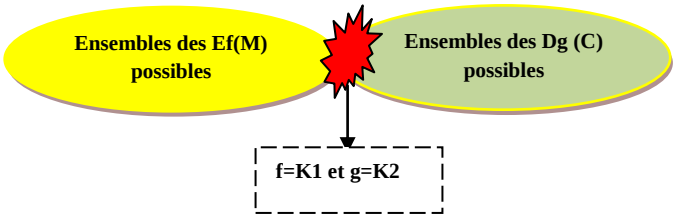



Figure 1.17: Meet-In-The-Middle.

Un nombre de clés possibles réduit rend cette attaque utilisable.

La rencontre au milieu est une attaque classique. Cette attaque rend inutile le sur-chiffrement, c'est-à-dire l'emploi de deux circuits de chiffrement par bloc montés en série avec deux clés. C'est pour cela qu'on utilise le 3-DES et non pas le 2-DES

1.23.2.7 Man-In-The-Middle

Souvent confondue avec l'attaque précédente en raison de son acronyme (MITM), elle est pourtant très différente dans les faits. Son déroulement est illustré à la figure ci-dessous : le pirate se fait passer pour B auprès de A et pour A auprès de B.

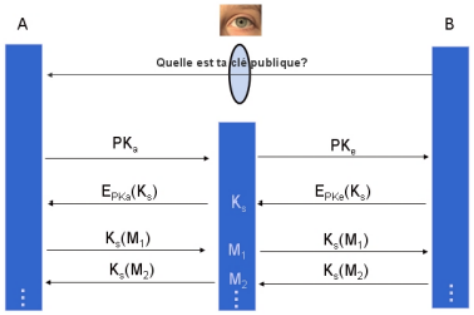



Figure 1.18: Man-In-The-Middle.

### 1.23.3 Autres attaques logicielles

Quelques autres attaques logicielles répandues sont fréquemment citées :

- Mascarade/Déguisement : attaquer le service d'authentification d'un système en se faisant passer pour une personne digne de confiance.
- Attaque par Re-jeux : répéter un envoi de données ou une séquence d'actions sans les modifier (répéter un ordre de crédit par exemple).

La liste établie précédemment n'est bien évidemment pas exhaustive. Il existe un très grand nombre d'autres attaques, souvent plus spécifiques à un algorithme particulier.

### 1.23.4 Attaques sur les fonctions de hachage

Il existe de type d'attaque sur les fonctions de hachage comme le montre la figure 1.19.

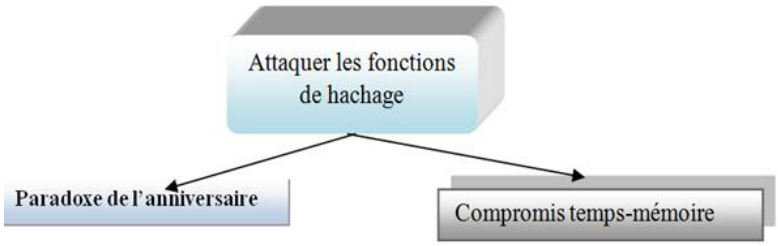



Figure 1.19 : Attaque sur les fonctions de hachage.

#### 1.23.4.1 Utilisation du paradoxe de l'anniversaire

Nous avons déjà traité le cas des attaques par le paradoxe de l'anniversaire (Birthday Attack). Celui-ci consistait à traiter les variations d'un texte clair et d'un texte frauduleux pour déterminer un texte frauduleux de substitution.

#### 1.23.4.2 Le compromis temps-mémoire

L'approche est ici radicalement différente. Il ne s'agit plus de substituer une chaine par une autre tout en conservant un même haché, mais bel et bien de retrouver la chaine initiale, ayant donc servi à créer un haché donné. En d'autre termes, retrouver le x dans $H(x) = h$, H et h étant connus.

Ce type d'attaque est principalement utilisé dans le contexte du cassage de mots de passe : on voudrait retrouver le mot de passe correspondant à un haché donné, sachant que la plupart des systèmes de protection par mot de passe (notamment les OS) fonctionnent par comparaison du mot de passe entré par l'utilisateur avec le haché du mot de passe attendu (haché créé lors de la définition de ce mot de passe et stocké en clair dans l'OS).

A priori, il y a deux solutions :

- Tester tous les textes clairs possibles (mais selon la taille, cela demanderait trop de temps) ;

❖ Stocker tous les résultats (mais demanderait trop d'espace mémoire) .

La solution intermédiaire porte ainsi le nom de compromis temps-mémoire. Une nouvelle notion entre en ligne de compte : la Fonction de Réduction (R). Elle consiste en le processus inverse d'une fonction de hachage : elle fournit un texte clair à partir d'un haché donné. Il s'agira d'une fonction spécifique mais dont le résultat devra être pseudo-aléatoire (par exemple, le fait de prendre les premiers caractères du haché).

### 1.23.5 Les attaques par canaux auxiliaires

Une famille d'attaques n'est pas basée sur les données elles-mêmes, mais plutôt sur leur environnement. On les appelle les attaques par canaux auxiliaires, ou Side Channel Attacks. Plus précisément, le principe de telles attaques est d'étudier l'aspect physique d'un cryptosystème sous différents angles au lieu d'étudier son aspect logique (algorithmique, mathématique).

Depuis quelques années, il existe de nombreuses propositions d'attaques par canaux Cachés. Suivant le scenario considérée et la puissance supposée d'un attaquant, ces attaques peuvent être classifiées suivant différents critères comme le montre la figure 1.20 . Les attaques par canaux auxiliaires sont presque exclusivement réservées au monde de l'embarqué [33].

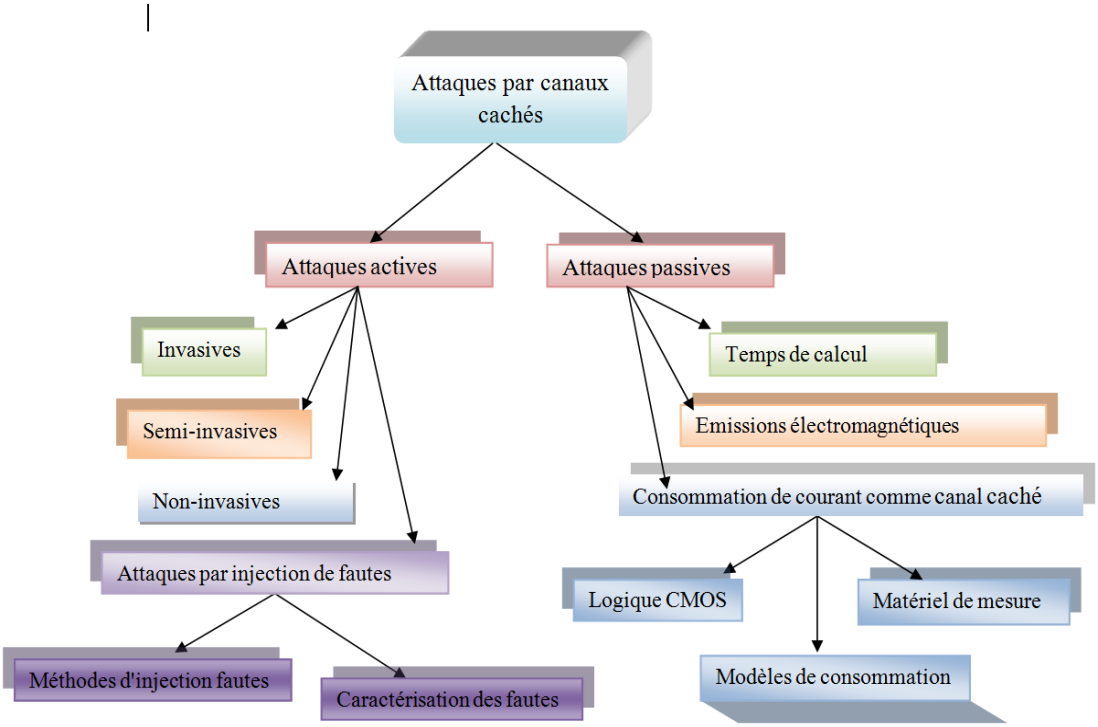



Figure 1.20 : classification des attaques par canaux auxiliaires.

On se protège contre un attaquant borné : en temps de calcul, en mémoire et en nombre de mesures. On peut donc augmenter la complexité des attaques et diminuer le nombre de mesures disponibles.

## 1.24 Conclusion

On a présenté dans ce chapitre une vue d'ensemble sur la cryptographie moderne en mettant le point sur les terminologies, les notations essentielles dans le domaine de Cryptologie ainsi que les différentes techniques d'attaques. Car L'usage ancien du chiffrement et l'usage actuel en informatique ont conduit aux contraintes suivantes :

- Réalisation simple et rapide du chiffrement et du déchiffrement.
- Eviter un encombrement important des clés.
- Un crypto systèmes dépend de paramètres (clés).

Des nouveaux services de sécurité ont été ajoutés grâce aux primitives de hachage, mac et signature numérique, la crypto moderne ajoute donc des services d'authentification, d'intégrité et de non-répudiation en plus de la confidentialité. Ces nouveaux services permettent une meilleure sécurité dans les communications et les transactions utilisant les infrastructures réseaux et Internet.

Lorsqu'un mécanisme cryptographique est implanté sur une cible matérielle, que ce soit une carte à puce ou un FGPA il faut prendre en compte son environnement d'exécution afin de définir les possibilités d'attaques et les protections nécessaires.

On a aussi expliqué les différents modes de cryptologie et les fonctions de hachages, ce qui nous donne une base pour passer aux autres chapitres où on va s'intéresser aux courbes elliptiques qui sont utilisées pour la cryptographie asymétrique. Enfin la sécurité est un compromis entre efficacité et convivialité

# CHAPITRE 2
# CHAMPS DE GALOIS ET COURBES ELLIPTIQUES

*« L'homme raisonnable s'adapte au monde ; l'homme déraisonnable s'obstine à essayer d'adapter le monde à lui-même. Tout progrès dépend donc de l'homme déraisonnable ».*
***Bernard Shaw***

## 2.1 Introduction

Quels que soient les moyens mis en œuvre, du fait de l'ouverture vers l'extérieur, les réseaux seront de plus en plus vulnérables. Dans ses conditions, il appartient à chaque entreprise de mesurer le risque et le coût d'une indisponibilité du système, de la divulgation d'information... et déterminer une politique dite de sécurité, d'en mesurer les coûts et d'assurer en permanence une veille technologique pour que les moyens employés restent efficaces devant l'évolution des menaces.

Les courbes elliptiques sont très utilisées dans la cryptographie dans plusieurs applications et protocoles de nos jours. Dans ce chapitre, nous allons en premier temps donner les notions mathématiques des ECC (Elliptic Curves), mettre le point sur la théorie des courbes elliptiques et leurs applications dans la cryptographie et en deuxième temps dresser un portrait de l'état de l'art des implémentations cryptographiques matérielles basées sur les courbes elliptiques et définir de manière succincte les courbes elliptiques et en dégager les principales propriétés.

## 2.2 Notions mathématiques élémentaires

Avant de présenter les courbes elliptiques, nous étudions d'abord les notions mathématiques dont nous avons besoin pour comprendre son fonctionnement.

Étant donné un ensemble F muni de deux opérations, l'addition (notée par « + ») et la multiplication (notée par « . »), qui satisfont les propriétés suivantes :

(i) (**F**, +) est un groupe abélien avec l'élément identité (additif) représentée par 0.

(ii) (**F** \ {0}, .) est un groupe abélien avec l'élément identité (multiplicatif) représentée par 0.

(iii) Une loi de distribution : $(a + b).c = a.c + b.c$ pour tout ***(a,b,c)*** $\epsilon$ ***F***.

Si l'ensemble F est dénombrable ou fini, F est dit champ fini.

### 2.2.1 Corps

Un corps est un anneau commutatif dans lequel tout élément non nul est inversible. Une telle structure est en général notée (K,+,.) se résumant à un ensemble muni de deux opérations .

### 2.2.2 Corps fini

Un corps est un ensemble F muni des opérations de multiplication et d'addition, et qui satisfont les règles familières suivantes :

– l'associativité et la commutativité de l'addition et de la multiplication,

– la loi est distributive,

– l'existence d'un élément neutre additive 0 et d'un élément neutre multiplicative 1,

– l'existence des inverses additifs et des inverses multiplicatives pour Tout élément sauf le 0.

Les exemples suivants de corps sont fondamentaux dans de nombreux domaines de la mathématique :

– le corps Q constitué de tous les nombres rationnels ;

– le corps R des nombres réels ;

– le corps C des nombres complexes ;

– le corps (Z/nZ) des entiers modulo un nombre premier p.

### 2.2.3 Corps fini binaire

Un corps fini binaire est un corps fini de caractéristique 2, notée $F2^m$ ou $GF(2^m)$.

#### 2.2.3.1 Champs Finis ou champs de Galois

Une mise en œuvre efficace de l'arithmétique des corps fini est un concept fondamental à l'étude des systèmes cryptographiques à base de courbes elliptiques, car toutes les opérations sont effectuées sur la courbe elliptique avec des opérandes appartenant au même champ [29].

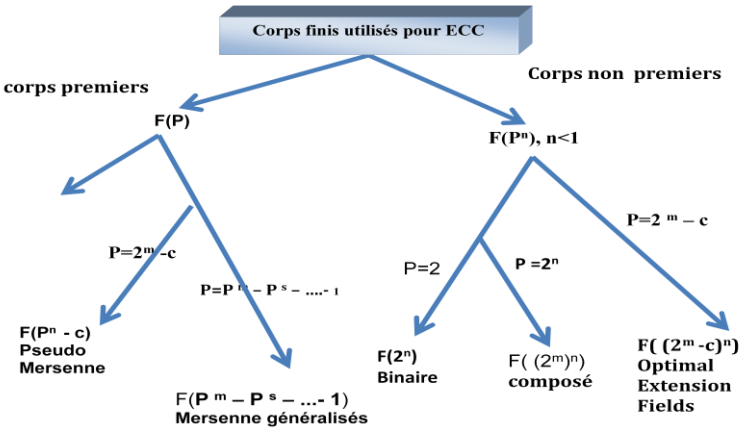



Figure 2.1 : Classification de quelques types de corps finis utilisés dans le cadre des courbes elliptiques.

2.2.3.2 Operations dans les champs finis

Un champ F est muni de deux opérations, l'addition et de multiplication. La soustraction des éléments du champ est défini en termes de l'addition de $a, b \in F$, tel que :

$$a - a + b = (-b) \quad (2.1)$$

Lorsque $-b$ est l'unique élément de **F** tel que :

$$b + (-b) = 0 \quad (-b \text{ est appelé négatif de } b). \quad (2.2)$$

De même, la répartition des éléments du champ est défini en termes de multiplication de $a, b \in F$ avec $b = 0$, $a / b = a \bullet b^{-1}$ ou $b^{-1}$est l'unique élément dans F tel que :

$$b \bullet b^{-1} = 1 \quad (b^{-1} \text{ est appelé l'inverse de } b) \quad (2.3)$$

2.2.3.3 Existence et unicité

L'ordre d'un corps fini est le nombre d'éléments du champ. Il existe un ensemble fini F d'ordre q si et seulement si q est une puissance a base d'un nombre premier tel que, $q = p^m$ où p est premier appelé caractéristique de F, et m est un entier positif.

- Si m =1, alors **F** est dit champ premier.
- Si m ≥ 2, **F** est dit champ étendu.

Pour toute puissance première de $q$, il existe un seul et seulement un seul champ fini d'ordre $q$; Ce qui veut dire que si deux champs sont d'ordre q, ils sont structuralement les mêmes, à faire l'exception de la notation des éléments qui diffère. Mathématiquement les deux champs sont isomorphes, chaque champ est noté $\boldsymbol{GF}_q$ ou $GF(q)$.

Un corps fini est schématiquement un ensemble fini dans lequel on peut :

- additionner
- Multiplier
- ou diviser les éléments.

Pour étudier la cryptographie sur les courbes elliptiques, il faut que nous comprenions les deux types de corps ci-dessous. On peut distinguer deux types de corps finis :

- les corps dit premiers qui ont un nombre premier p d'éléments,
- et les corps non premiers qui ont $P^n$ éléments.

2.2.4 Champs premiers

Soit p un nombre premier, les entiers modulo p sont les éléments de l'ensemble {0,1,2,…p-1} avec l'addition et la multiplication performées modulo p, représentent un champ premier d'ordre p, dénoté $GF_p$. p est appelé module de $GF_p$.

Pour chaque entier a, alors « a mod p » est appelé le reste unique r, tel que $0 \leq r \leq p-1$, obtenu par la division de a par p. cette opération est appelé réduction modulo p.

Exemple du champ premier [7] $\mathbf{GF}_2{}^9$, dont les éléments sont {0,1,2, …,28}. Les opérations suivantes montrent les propriétés de ce champ :

(i) Addition 17+20 = 8, car 37 mod29 =8.

(ii) Soustraction 17-20 =26, car -3 mod29 =26.

(iii) Multiplication 17.20 =21, car 310 mod29= 21.

(iv) Inversion $17^{-1}$ =12, car 17.12 mod29 =1.

### 2.2.5 Champs binaires

Les champs d'ordre $2^m$ sont appelés champs binaires ou champs de caractéristique 2. Un champ binaire $GF(2^m)$ est construit à partir d'une base polynomiale. Donc les éléments de $GF(2^m)$ sont des polynômes binaire. i.e : des polynômes dont les coefficients appartiennent à {0,1}, de degré au plus égal à $"m-1"$ :

$$F_{2^m} = \{a_{m-1}z^{m-1} + a_{m-2}z^{m-2} + \cdots + a_1 z^1 + a_0 : a_i \in \{0,1\}\}. \quad (2.4)$$

Un polynôme irréductible $f(z)$ de degré m est choisi. L'irréductibilité de $f(z)$ veut dire qu'il ne peut être factorisé en un produit de polynômes de degré inferieur à « m ».

- L'addition dans $GF(2^m)$ est une addition polynomiale, c'est l'addition des coefficients modulo 2.
- La multiplication est une multiplication polynomiale avec une réduction modulo $f(z)$.

Pour tout polynôme $a(z)$, $a(z)\, mod\, f(z)$ dénote l'unique reste $r(z)$ de degré inferieur à m, obtenu par « longue division » de $a(z)$ par $f(z)$, cette opération est appelée réduction polynomiale.

Exemple du champ binaire GF($2^4$), formé de 16 éléments polynomiaux de degré maximum égal à 3, représenté dans la Tableau 2.1.

Tableau 2.1. : Éléments du champ $GF_{2^4}$ générés par le polynôme $f(z) = z^4 + z + 1$.

| | | | |
|---|---|---|---|
| 0 | $z^2$ | $z^3$ | $z^3 + z^2$ |
| 1 | $z^2 + 1$ | $z^3 + 1$ | $z^3 + z^2 + 1$ |
| $z$ | $z^2 + z$ | $z^3 + z$ | $z^3 + z^2 + z$ |
| $z + 1$ | $z^2 + z + 1$ | $z^3 + z + 1$ | $z^3 + z^2 + z + 1$ |

Quelques exemples d'opérations dans le champ$F_{2^4}$ , avec $f(z) = z^4 + z + 1$

(i) Addition : $(z^3 + z^2 + 1) + (z^2 + z + 1) = z^3 + z$

(ii) Soustraction : $(z^3 + z^2 + 1) - (z^2 + z + 1) = z^3 + z$, car $-1 = 1$ dans le champ $F_2$ , avec $a = -a$ pour tout $a \in F_{2^m}$.

*(iii)* Multiplication : $(Z^3+Z^2+1).(Z^2+Z+1)=Z^2+1$

puisque $(Z^3+Z^2+1).(Z^2+Z+1)=Z^5+Z+1$

et $(Z^5+Z+1) \bmod (Z^5+Z+1)=Z^2+1$

(iv) Inversion: $(Z^3+Z^2+1)^{-1}=Z^2$ puisque $(Z^3+Z^2+1).Z^2 \bmod (Z^4+Z+1)=1$

2.2.5.1 Champs étendus

Un champ étendu $GF_{p^m}$ , puise des mêmes caractéristiques, que le champ $\mathrm{F_p}$ , ces éléments sont de la forme :

$$GF_{p^m} = \{a_{m-1}z^{m-1} + a_{m-2}z^{m-2} + \cdots + a_1z^1 + a_0 : a_i \in \mathrm{F_p}\}. \qquad (2.5)$$

Même le polynôme $f(z)$ reste le même pour générer les éléments de $GF_{p^m}$, ainsi que lors de la réduction polynomiale.

**Exemple :** Etant donne p = 251 et m = 5, voici quelques exemples d'opérations dans le champ $G\mathrm{F}_{2^{251}}$ , avec $f(z) = z^5 + z^4 + 12z^3 + 9z^2 + 7$.

Si $a = 123z^4 + 76z^2 + 4$ et $b = 196z^4 + 12z^3 + 225z^2 + 76$

(v) Addition : $a + b = 68z^4 + 12z^3 + 50z^2 + 80$

(vi) Soustraction : $a - b = 178z^4 + 239z^3 + 102z^2 + 7z + 179$

(vii) Multiplication : $a.b = 117z^4 + 151z^3 + 117z^2 + 182z + 217$

(viii) Inversion : $(a)^{-1} = 109z^4 + 111z^3 + 250z^2 + 98z + 85$

2.2.5.2 Sous-champ d'un champ fini

Un sous-ensemble $k$ d'un champ $K$ est un champ par rapport aux opérations de K. dans ce cas $K$ est un dit champ étendu de $k$, n'importe quel champ fini $GF_{p^m}$ a précisément un sous-champ d'ordre $p^l$ , pour tout diviseur $l$ de $m$. Les éléments de ce sous-champ sont les éléments $a \in GF_{p^m}$ , tel que :$a^{p^l} = a$ , à l'inverse tout sous-champ de $GF_{p^m}$ a un ordre égal à $p^l$ pour chaque diviseur $l$ de $m$.

2.2.5.3 Bases d'un champ fini

Le champ fini $GF_{q^m}$ peut être comparé à un espace vectoriel sur son sous-champ $GF_q$. Donc les vecteurs sont les éléments de $GF_{q^m}$, et les scalaires sont les éléments de $GF_q$, l'addition vectorielle est une l'addition dans$GF_{q^m}$ , la multiplication scalaire est une multiplication dans $GF_{q^m}$ des éléments de $GF_q$ avec les éléments de $GF_{q^m}$. L'espace vectoriel a une dimension égal à$n$ , et possède plusieurs bases.

Si $B = \{b_1, b_2, \ldots, b_n\}$ est une base, donc $a \in GF_{q^m}$ peut être représenté uniquement par le n-tuple $\{a_1, a_2, \ldots, a_n\}$ éléments de$GF_q$ , de la façon suivante :

$$a = a_1.b_1 + a_2.b_2 + \cdots + a_n.b_n \qquad (2.6)$$

Par exemple, dans la représentation polynomiale du champ $GF_{p^m}$ décrit comme précédemment, $F_{p^m}$ est un espace vectoriel sur $GF_p$et $\{z^{m-1}, z^{m-2}, \cdots, z^1, 1\}$ est une base $GF_{p^m}$ sur $GF_p$.

2.2.5.4 Groupe multiplicatif d'un champ fini

Les éléments non-nuls d'un champ $GF_p$ , noté $GF_q^*$, forment un group cyclique sous la multiplication. Donc, il existe des éléments $b \in GF_q^*$, appelés générateurs tel que : $GF_q^* = \{b_i : 0 \leq i \leq q-2\}$ (2.7)

L'ordre de $a \in GF_q^*$, est le plus petit entier positif, $t$ tel que $a^t = 1$ , car $GF_q^*$ est un groupe cyclique, ce qui conduit à dire que $t$ est un diviseur de $q-1$.

Dans ce qui suit, nous allons traiter des champs binaires, car notre implantation est dans ce cadre, les champs premiers sont expliqués avec plus de détails dans [7,30,34].

## 2.3 Arithmétique des champs binaires

En vue de l'implantation hardware des courbes elliptiques, il est nécessaire d'introduire les briques utiles, relatives aux opérations élémentaires représentées en bas de la pyramide en vue de la réalisation de la multiplication scalaire.

Nous assumons que les registres sont de taille $W$, qui peut être un registre de taille multiple de 8 bits, cette contrainte sera levée lors de notre implantation hardware. Les bits d'un mot de $W$, noté $U$, sont dénombrés de 0 à $W-1$. Avec le bit le plus à droite de $U$ désigné par bit 0. Ces notations seront utilisées tout au long de ce chapitre.

Soit $f(z)$ un polynôme binaire irréductible de degré m, tel que $f(z) = z_m + r(z)$.

Les éléments de $GF_{2^m}$ sont les polynômes binaires de degré au plus égal à "$m-1$", et la multiplication est performée modulo $f(z)$.

Soit un élément du champ $GF_{2^m}$:

$$a(z) = a_{m-1}z^{m-1} + a_{m-2}z^{m-2} + \cdots + a_1 z^1 + a_0 \quad (2.8)$$

Associé au vecteur binaire de taille m :

$$\{a_{m-1}, a_{m-2}, \ldots, a_1, a_0\} \quad (2.9)$$

Si l'on considère que $t = \lceil m/W \rceil$ (2.10)

Et $s = W.t - m.$ (2.11)

Donc le registre $W$, peut être représenté de la façon suivante.

| | $A[t-1]$ | | A[1] | A[0] |
|---|---|---|---|---|
| | $(a_{m-1}, a_{m-2}, \ldots, a_{(t-1)W})$ | $\cdots$ | $(a_{2.W-1}, \ldots, a_{W+1}\ a_W)$ | $(a_{W-1}, \ldots, a_1\ a_0)$ |

b1 bits

Représentation d'un élément $a \in GF_{p^m}$ , comme une matrice $A$ de mots de taille $W$ chacun Les bits « b1 » représentent les plus significatifs sont nuls.

### 2.3.1 L'Addition

L'addition dans champs binaires est une opération XOR au niveau bit. Cette opération dépend des ressources allouées, nous supposerons dans ce qui suit que le registre de taille m est disponible, toutefois, en vue de paramétrer notre travail, nous considèrerons le registre de taille s.

Algorithme 2.1 nous donne l'addition binaire dans un champ binaire.

**Algorithme 2.1 d'addition binaire**

Entrées : Polynômes binaires $a(z), b(z)$ de degré au plus $m-1$
Sortie : $c(z) = a(z) + b(z)$

- Boucle $i = 0 : t-1$
  Faire : $C[i] = A[i] \oplus B[i]$
- Fin boucle

Retourner ( $c$ ).

### 2.3.2 La multiplication modulaire

La multiplication modulaire est largement utilisée dans les applications cryptographiques à clé publique spécialement là où l'exponentiation modulaire est essentielle comme le cas du cryptosystème CCE.

La multiplication modulaire consiste à effectuer la multiplication de deux nombre A et B et de prendre le reste S obtenu de la division du produit (A.B) par un troisième nombre N, appelé modulo. Cette opération est donnée par l'expression :

S= (A.B) modulo N

Cet operateur est parmi les opérateurs les plus complexes en termes d'espace et de temps. Même si les nouveaux circuits de FPGA [35] , fournissent des multiplieurs très rapides dédiés. Hélas, ces composants ne peuvent être utilisés dans les champs finis, car ces multiplieurs intégrés se basent sur la retenue lors de la multiplication, ce qui ne correspond pas au type d'opération des multiplications à base de portes XOR n'ayant pas de retenue.

L'orientation des concepteurs travaillant dans le domaine de la cryptographie se dirigent à étudier de nouveaux algorithmes rapides, efficaces et facilement implantables.

En plus pour retrouver le reste S, cette technique nécessité une division qui est une opération de calcul multi-précision [36] la plus compliqué et le plus

Le schéma synoptique ci-dessous présente les différentes méthodes de multiplication modulaire.

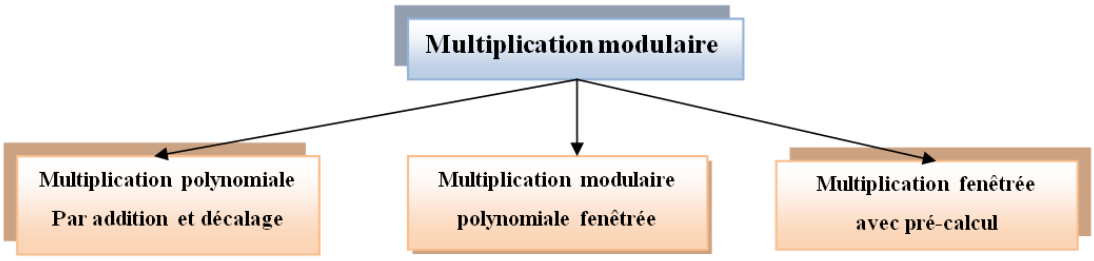



Figure 2.2 : multiplication modulaire.

2.3.2.1 Multiplication polynomiale par addition et décalage

Étant donné deux polynômes binaires $a(z), b(z)$ de degré au plus $m-1$, le produit $a(z).b(z)$ est défini comme suit :

$$a(z).b(z) = a_{m-1}z^{m-1}.\mathrm{b(z)} + a_{m-2}z^{m-2}.\mathrm{b(z)} + \cdots + a_1 z^1.\mathrm{b(z)} + a_0.\mathrm{b(z)} \quad (2.12)$$

A l'itération $i$, l'algorithme calcule $z^i.\mathrm{b(z)}\ mod\ f(z)$ et additionne le résultat à un accumulateur d'addition $c$ si $a_i = 1$.

$$\text{Si } b(z) = b_{m-1}z^{m-1} + b_{m-2}z^{m-2} + \cdots + b_1 z^1 + b_0 \quad (2.13)$$

$$\text{Donc, } b(z).z = b_{m-1}z^m + b_{m-2}z^{m-1} + \cdots + b_1 z^2 + b_0 z \quad (2.14)$$

$$\equiv b_{m-1}r(z) + (b_{m-2}z^{m-1} + \cdots + b_1 z^2 + b_0 z)\ mod\ f(z) \quad (2.15)$$

Alors, $b(z).z\ mod\ f(z)$ peut être calcule par un décalage à gauche de la représentation vectorielle de $b(z)$, suivi d'une addition de $r(z)$ a $b(z)$ si le bit $b_{m-1}$ =1.

**Algorithme 2.2 : la multiplication dans $GF_{2^m}$ par décalage et addition :**

Entrées : Polynômes binaires $a(z), b(z)$ de degré au plus $m-1$
Sortie : $c(z) = a(z).b(z)$
- Si $a_0 = 1,\ alors\ c \leftarrow b,\ sinon\ c \leftarrow 0.$
- Boucle $i = 1 : m-1$
  - Faire : $b \leftarrow b.z\ mod f(z)$
  - Si $a_i = 1,\ alors\ c \leftarrow c + b$
- Fin boucle

Retourner ( $c$ )

Cet algorithme a l'avantage d'éviter la réduction polynomiale en bloc, car avec cette méthode, elle s'effectue étape par étape. Mais l'inconvénient de durer au moins $m$ étapes, donc avec les tailles des champs de l'ordre des centaines, cette opération représente un réel handicap pour les architectures dédiées à la cryptographie, par exemple.

2.3.2.2 Multiplication modulaire polynomiale fenêtrée

Nous pouvons observer que lors du calcul de $z^k.\mathrm{b(z)}$, pour $k \in [0, W-1]$, alors le calcul de $z^{W_{j+k}}.\mathrm{b(z)}$ peut être calculé en ajutant de $j$ zéros à la droite de la représentation

vectorielle de $z^k.\mathrm{b}(\mathrm{z})$. L'algorithme suivant traite les bits du mot A de la droite à la gauche.

On utilise la notation suivante tel que : $C = (C[n], \dots, C[2], C[1], C[0])$ est une matrice, et $C\{j\}$ est la forme tronquée de la matrice $(C[n], \dots, C[2], C[j+1], C[j])$.

**Algorithme 2.3 de la multiplication polynomiale, (Combinaison de Droite à Gauche)**

Entrées : Polynômes binaires $a(z), b(z)$ de degré au plus $m-1$
Sortie : $c(z) = a(z).b(z)$

- $C \leftarrow 0$
- Boucle $k = 0 : W-1$
  - Boucle $j = 0 : t-1$
  - Si le bit de position $k$de $A[j]$ est égal a 1, faire addition $B$et $C\{j\}$
  - Fin boucle
  - si $k \neq (W-1)$ , $alors\ B \leftarrow B.z$
- Fin boucle

Retourner ( $C$)

Comme illustré dans la figure 2. 3. cas de m=113, et W=32

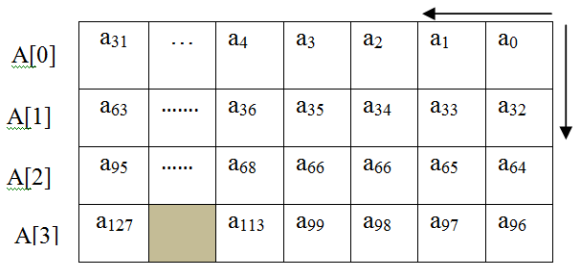



Figure 2.3 : Méthode de multiplication polynomiale par traitement des lignes de droite à gauche et des colonnes de haut en bas.

La même procédure peut être réalisée de gauche à droite, i.e. du bit le plus significatif au bit le moins significatif.

Cet algorithme est plus rapide que le premier (3.2.2.1), car il contient moins de décalages, toutefois, il peut été encore accéléré par un pré-calcul des polynômes $u(z).b(z)$ pour tous les polynômes $u(z)$ de degré inférieur à$w$, comme illustré dans la section suivante.

2.3.2.3 Multiplication fenêtrée avec pré-calcul

**Algorithme 2.4 : la multiplication polynomiale (Combinaison de Droite à Gauche) avec une fenêtre de taille $w$:**

Entrées : Polynômes binaires $a(z), b(z)$ de degré au plus $m-1$
Sortie : $c(z) = a(z).b(z)$

- Calculer $B_u = u(z).b(z)$ pour tous les polynômes de degré au plus égal a $w-1$
- $C \leftarrow 0$
- Boucle $k = 0 : (\frac{W}{w}) - 1$
  - Boucle $j = 0 : t-1$

- Poser $u = (u_{w-1}, u_{w-2}, \dots, u_1)$, ou $u_i$ est le bit $(wk + i)$ de $A[j]$
- Additionner $B_u$ à $C[j]$

▪ Fin boucle
▪ si $k \neq 0$ , $alors\ C \leftarrow C.z^w$

o Fin boucle

Retourner ( $C$)

2.3.2.4 Le multiplieur de Karatsuba-Ofman

Il s'agit d'un algorithme qui a été proposé par Karatsuba dès 1963, à l'origine pour effectuer des multiplications sur des grands entiers. Cette méthode est l'une des méthodes les plus performantes, vu que différents concepteurs se réfèrent à cette méthode avec certaines modifications [37,38], en vue de l'accélérer et l'implanter dans un espace bien optimisé.

L'algorithme de Karatsuba calcule le produit de deux polynômes dans R[X] et de degré ≤ n en $O\left(n^{log_2(3)}\right)$ multiplications dans R. L'algorithme de Karatsuba calcule le produit de deux entiers de taille $2^{wn}$ en $O\left(n^{log_2(3)}\right)$ multiplications entre des entiers de w bit.

Si l'on note $K(n)$ le nombre de multiplications nécessaires pour calculer le produit de 2 nombres à $n$ chiffres avec cette méthode, on obtient la relation de récurrence suivante : $K(n) \leq 3K\left(\left\lceil \frac{n}{2} \right\rceil\right) + 4n$ , où $\lceil n/2 \rceil$ est la partie entière par excès de m/2 (l'entier suivant immédiatement n/2) où le terme 4n vient estimer le nombre d'additions.

Cette méthode utilise la technique de diviser pour conquérir, en plus d'une technique de minimisation des multiplications utilisées, en regroupant les termes croisés. La complexité est réduite à $O\left(n^{log_2(3)}\right) \approx O(n^{1,585})$. [30]. Ceci est plus rapide que l'algorithme standard ; pour donner un exemple, pour $n$ = 1000, $n^{\log_2(3)}$ est de l'ordre de 50 000 alors que $n^2$ = 1 000 000.

La technique repose sur ce qui suit :

Étant donne deux polynômes binaires $a(z)$ , $b(z)$, le produit :

$$a(z).b(z) = (A_1 z^l + A_0).(B_1 z^l + B_0) \quad (2.16)$$

$$= A_1.B_1 z^{2l} + (A_1.B_0 + A_0.B_1)z^l + A_0.B_0 \quad (2.17)$$

Avec $l = m/2$ et $A_1, A_0, B_1, B_0$ des polynômes de taille $l$.

A cette étape, 4 multiplieurs de taille $m/2$ sont utilisés, cette opération peut se répéter jusqu' atteindre un multiplieur au niveau bit.

Par l'utilisation d'un multiplieur de 4 bits comme brique de base, par exemple, on obtient ce qui suit :

Posons $a(z)$ et $b(z)$, deux opérandes de 4 bits chacun, tels que :

$$a(z) = a_3z^3 + a_2z^2 + a_1z + a_0 \quad (2.18)$$

$$b(z) = b_3z^3 + b_2z^2 + b_1z + b_0 \quad (2.19)$$

$$c(z)=a(z).b(z) = (a_3z^3 + a_2z^2 + a_1z + a_0).(b_3z^3 + b_2z^2 + b_1z + b_0)$$

$$c(z) = a(z).b(z) = a_3.b_3z^6 + (a_3.b_2 + b_2.a_3).z^5 + (a_2.b_2 + a_3.b_1 + b_1.a_3).z^4 + (a_3.b_0 + a_0.b_3 + a_2.b_1 + a_1.b_2).z^3 + (a_2b_0 + b_0a_2 + a_1b_1).z^2 + (a_1b_0 + b_1a_0).z + a_0b_0 \quad (2.20)$$

Donc d'un point de vue hardware :

$$\left.\begin{aligned} c(0) &= a_0.b_0 \\ c(1) &= (a_1.b_0 \oplus b_1.a_0) \\ c(2) &= (a_2.b_0 \oplus a_1.b_1 \oplus b_0.a_2) \\ c(3) &= (a_3.b_0 \oplus a_2.b_1 \oplus a_1.b_2 \oplus a_0.b_3) \\ c(4) &= (a_3.b_1 \oplus a_2.b_2 \oplus b_1.a_3) \\ c(5) &= (a_3.b_2 \oplus b_2.a_3) \\ c(6) &= a_3.b_3 \end{aligned}\right\} \quad (2.21)$$

Notons qu'il n'y a pas de propagation de retenue, dans l'arithmétique des champs de Galois.

Le besoin du multiplieur en termes de portes logiques est illustré dans le tableau 2.2.

Tableau 2.2 : Complexité du multiplieur 4 bits en portes logiques.

| **Portes** | **Complexité** |
|---|---|
| ET | 16 |
| XOR | 9 |

L'autre amélioration introduite par Karatsuba-Ofman, est que le terme de milieu de l'équation 2.17, peut s'écrire de la façon suivante :

$$(A_1.B_0 + A_0.B_1).z^l = [(A_1 + A_0).(B_1 + B_0) + A_1.B_1 + A_0.B_0].z^l \quad (2.22)$$

L'équation 2.17 devient alors :

$$= A_1.B_1z^{2l} + [(A_1 + A_0).(B_1 + B_0) + A_1.B_1 + A_0.B_0].z^l + A_0.B_0 \quad (2.23)$$

Ou l'on remarque que les produits : $A_1.B_0$ et $A_0.B_1$ ont disparus, laissant place aux termes, déjà existants $A_0.B_0$ et $A_1.B_1$.

Le coût ou complexité matérielle du multiplieur est illustré dans le tableau 2.3.

Tableau 2.3. : Ressources nécessaires au multiplieur Karatsuba-Ofman de $m$ bits.

| **Composants** | **Implantation Parallèle** | **Implantation Série** |
|---|---|---|
| Multiplieur de taille $m/2$ bits | 3 | 1 |
| Additionneur 4 Portes (XOR) $(2l-2)$bits | 2 | 1 |
| Additionneur 2 Portes (XOR) (l)bits | 2 | 1 |
| Nombre de cycles d'exécution | 3 | 5 |

Ce type de multiplieurs peut être modifié, en vue de son amélioration, en segmentant les opérandes initiaux $a(z)\ et\ b(z)$, en trois segments de taille égale, cette modification a la particularité de permettre la réduction de la taille des multiplicandes plus rapidement, réalisant ainsi des multiplications de moindre taille, augmentant relativement la complexité de l'addition décalée des résultats intermédiaires.

Notons l'adjonction d'un registre de valeurs nulles, à la partie supérieure des opérandes, dont la taille est égale à :

$$Regsitre\ de\ valeurs\ nulles = 3 * \left[partie\ entière\ de\left(\frac{m}{3}\right) + 1\right] - m \qquad (2.24)$$

Ce registre contribue initialement à la segmentation tri-symétrique des opérandes, et devra être pris en compte, lors des additions décalées.

- Multiplieur de Karatsuba-Ofman, pour le cas des variables de taille 224 bits :

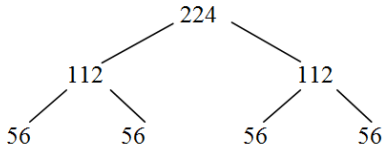



Figure 2.4: Segmentation en multiplieurs de taille de $n/2$.

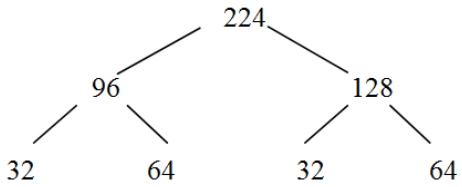



Figure 2.5: Segmentation en multiplieurs de taille proportionnelle à $(W = 32)$.

Le choix est alors obtempéré par les deux critères suivants :

- ✓ Réduction du décalage lors de rassemblement des résultats des multiplieurs, le plus les multiplieurs sont de taille identique, le plus simple est la juxtaposition.

- ✓ Réutilisation possible des multiplieurs (composants) à d'autres implantations.

Multiplieur de Karatsuba-Ofman, pour le cas des opérandes de taille 192 bits

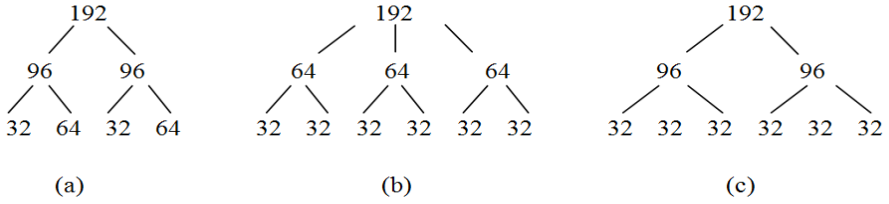



Figure 2.6 : Différentes configurations (a), (b), (c) du multiplieur de Karatsuba-Ofman, cas de opérandes de taille 192 bits.

L'algorithme Karatsuba-Ofman se trouve dans le chapitre 4 , lorsque différentes configurations sont en ligne de course, la sélection est aussi basée sur les critères précédents, ainsi que le critère de complexité hardware, comme illustré dans le tableau 2.4.

Tableau 2.4 : Nombre de multiplieurs par configuration, cas des opérandes de taille 192 bits.

| Configuration | Multiplieurs 32 x 32 |
|---|---|
| (a) | 21 |
| (b) | 18 |
| (c) | 18 |

### 2.3.3 Le carré polynomial modulaire

Élever au carrée un polynôme binaire dans $GF(2^m)$ est une opération très simple, car elle consiste à insérer des coefficients nuls aux puissances impaires du polynôme résultat [39], en gardant les mêmes coefficients du polynôme de base aux puissances paires, comme mentionnée dans la figure 3.6.

| | | $a_{m-1}$ | $a_{m-2}$ | …. | $a_1$ | $a_0$ | | |
|---|---|---|---|---|---|---|---|---|
| | ↙ | | ↓ | | | ↘ | ↘ | |
| 0 | $a_{m-1}$ | 0 | $a_{m-2}$ | 0 | … | $a_1$ | 0 | $a_0$ |

Figure 2.7: Carré d'un polynôme de taille $m-1$.

Prenons l'exemple de la section 2.2.3, équation 2.20,

$c(z) = a(z).b(z) = a_3.b_3z^6 + (\mathrm{a_3}.\mathrm{b_2} + \mathrm{b_2}.\mathrm{a_3}).\mathrm{z}^5 + (\mathrm{a_2}.\mathrm{b_2} + \mathrm{a_3}.\mathrm{b_1} + \mathrm{b_1}.\mathrm{a_3}).\mathrm{z}^4 + (\mathrm{a_3}.\mathrm{b_0} + \mathrm{a_0}.\mathrm{b_3} + \mathrm{a_2}.\mathrm{b_1} + \mathrm{a_1}.\mathrm{b_2}).\mathrm{z}^3 + (\mathrm{a_2b_0} + \mathrm{b_0a_2} + \mathrm{a_1b_1}).\mathrm{z}^2 + (\mathrm{a_1b_0} + \mathrm{b_1a_0}).\mathrm{z} + a_0b_0$ (2.25)

En posant $a(z) = b(z)$,

On obtient :

$$c(z) = a(z)^2 = \mathrm{a_3}^2z^6 + (\mathrm{a_3}.\mathrm{a_2} + \mathrm{a_2}.\mathrm{a_3}).\mathrm{z}^5 + (\mathrm{a_2}^2 + \mathrm{a_3}.\mathrm{a_1} + \mathrm{a_1}.\mathrm{a_3}).\mathrm{z}^4 + (\mathrm{a_3}.\mathrm{a_0} + \mathrm{a_0}.\mathrm{a_3} + \mathrm{a_2}.\mathrm{a_1} + \mathrm{a_1}.\mathrm{a_2}).\mathrm{z}^3 + (\mathrm{a_2a_0} + \mathrm{a_0a_2} + \mathrm{a_1}^2).\mathrm{z}^2 + (\mathrm{a_1a_0} + \mathrm{a_1a_0}).\mathrm{z} + \mathrm{a_0}^2$$

$$c(z) = a(z)^2 = \mathrm{a_3}^2z^6 + \mathrm{a_2}^2.\mathrm{z}^4 + \mathrm{a_1}^2.\mathrm{z}^2 + \mathrm{a_0}^2$$

$c(z) = a(z)^2 = \mathrm{a_3}z^6 + \mathrm{a_2}.\mathrm{z}^4 + \mathrm{a_1}.\mathrm{z}^2 + \mathrm{a_0}$ (2.26)

D'un point de vue implantation hardware :

$c(0) = a_0, \quad c(1) = 0, \quad c(2) = \mathrm{a_1}$

$c(3) = 0, \ c(4) = \mathrm{a_2}, \quad c(5) = 0, \quad c(6) = \mathrm{a_3}$ (2.27)

Dans le cas général : $a(z)^2 = a_{m-1}z^{2m-2} + a_{m-2}z^{2m-4} + \cdots + a_1z^2 + a_0$ (2.28)

Pour une implantation avec un registre de taille fixe $W$, il y'a lieu de faire un pré-calcul de plusieurs carrés de polynômes de taille $W/4$, par exemple.

**Algorithme 2.5 du carré d'un opérande de taille m (avec $W = 32$ )**

Entrées : Polynômes binaires $a(z)$de degré au plus $m - 1$
Sortie : $c(z) = a(z)^2$

- Pré-calculer pour chaque octet $d = (d_7, d_6, \dots, d_1, d_0)$ la quantité $T(d) = (d_7, 0, d_6, 0 \dots 0, d_1, 0, d_0)$
- Boucle $de\ i = 0 : t - 1$
  - Faire $A[i] = (u_3, u_2, u_1, u_0)$ pour chaque octet$u_j$ ,$\{j = 1..4\}$
  - $C[2i] \leftarrow (T(u_1), T(u_0))$
  - $C[2i + 1] \leftarrow (T(u_3), T(u_2))$
- Fin boucle
  Retourner ( $c$)

Cette méthode est plutôt dédiée aux machines dont la taille des opérandes est standard (32,64,..).

Dans la section 4.2.4, nous présenterons une méthode plus simple à réaliser algorithmiquement, avec un coût supplémentaire au niveau des ressources matérielles.

### 2.3.4 La réduction polynomiale ou modulaire

Étant donne un polynôme de réduction modulaire écrit sous la forme suivante : $f(z) = z^m + r(z)$, Où le degré de $r(z) \leq m - 1$ Avec $(z) = r_{m-1}z^{m-1} + \cdots + r_2z^2 + r_1z^1 + r_0$ , qui est représenté par une notation binaire $r = (r_{m-1}, \dots, r_2, r_1, r_0)$.

La réduction polynomiale est réalisée lorsque la taille du résultat, issu de la multiplication et/ou du carré polynomial dans le champ $GF(2^m)$, dépasse $m-1$ .

Pour chaque bit $\in [m\,..\,2.m-2]$, s'il est égal à 1, une addition polynomiale avec $f(z)$ est effectuée.

2.3.4.1 Cas de la multiplication

En général, le résultat $c(z)$ d'une multiplication polynomiale a une taille de « 2.m - 2 » éléments, donc une réduction modulaire s'avère impérative en vue d'obtenir un opérande de taille $m$ bits.

$$c(z) = c_{2m-1}z^{2m-1} + \cdots + c_{m-1}z^{m-1} + \cdots + c_1 z^1 + c_0 \qquad (2.29)$$

$$\equiv c_{2m-1}z^{2m-1} + \cdots + c_{m-1}z^{m-1} + \cdots + c_1 z^1 + c_0 \ \ (mod\ f(z)) \qquad (2.30)$$

La réduction polynomiale ou modulaire peut se réaliser, soit bit par bit, ou d'une façon parallèle, toujours dépendamment des tailles des bus disponibles du support hardware utilisé.

La figure 2.8 illustre les différentes méthodes de réduction polynomiale.

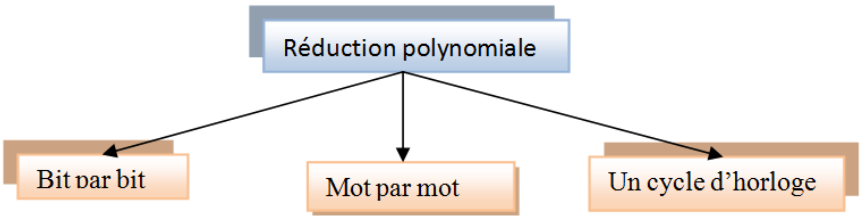



Figure 2.8 : réduction polynomiale.

2.3.4.2 Réduction polynomiale bit par bit

En scannant le polynôme $c(z)$, du bit de position $m$ au bit de position « $2.m-2$ », l'algorithme suivant montre les étapes, à suivre pour réaliser la réduction polynomiale.

**Algorithme 2.6 de la réduction polynomiale (bit par bit)**

Entrées : Polynômes binaires $c(z)$ de degré au plus $2m-2$
Sortie : $c(z) = c(z) mod f(z)$

- Pré-calculer $u_{k(z)} = z^k.r(z)$, 0
- Boucle $i = 2m-2 : m$
  - Si $c_i = 1$ faire
    - Poser $j = \lfloor (i-m)/W \rfloor$ et $k = (i-m) - W.j$
    - Additionner $u_k(z)$ à $C\{j\}$
- Fin Si
- Fin Boucle

Retourner $(C[t-1], \ldots C[1], C[0])$.

Les polynômes irréductibles des champs $GF(2^m)$ sont des trinômes ou penta nômes, dont une partie des termes est assez proche en puissance, permettant parfois la

réduction sur une plage d'un octet, ce qui est très intéressant pour une implantation hardware.

Considérons le polynôme $f(z) = z^{163} + z^7 + z^6 + z^3 + 1$, [29,40]

Avec $W = 32$, et $t = 6$.

Le mot $C[9]$, représenté par le polynôme :

$$c_{319}z^{319} + \cdots + c_{289}z^{289} + c_{288}z^{288} \quad (2.31)$$

Alors la réduction polynomiale s'effectue de la manière suivante :

Sachant que :

$$z^{163} = z^7 + z^6 + z^3 + 1 \; mod \; f(z) \quad (2.32)$$

Alors toute multiplication par le terme $z^i$, est aussi valide

$$z^{163+125} = z^{7+125} + z^{6+125} + z^{3+125} + z^{125} \; mod \; f(z) \quad (2.33)$$

Ainsi que :

$$z^{319} = z^{163} + z^{162} + z^{159} + z^{156} \; mod \; f(z) \quad (2.34)$$

En remplaçant les puissances modulaires des équations, 2.33 à 2.34, la réduction polynomiale s'effectue comme l'illustre la figure 3.8.

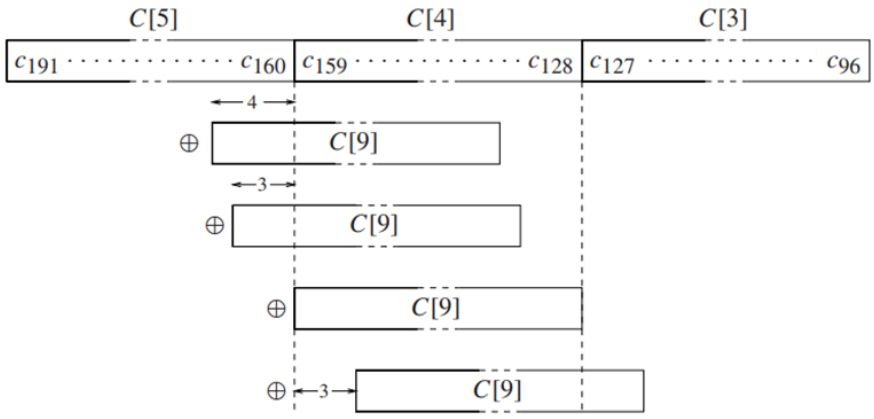



Figure 2.9. : Étapes de réduction modulaire du mot $C[9]$ par le Polynôme :

$$f(z) = z^{163} + z^7 + z^6 + z^3 + 1 \; ((W = 32)$$ [30].

Le même procédé est effectué aux autres mots : $C[10], C[8], C[7], C[6]$ et $C[5]$.

2.3.4.3 Réduction polynomiale mot par mot

L'algorithme suivant montre les étapes à suivre lors de la réduction modulaire, par le polynôme $f(z) = z^{163} + z^7 + z^6 + z^3 + 1$, cas de $m = 163$, en utilisant des mots de taille $W = 32$.

**Algorithme 2.7 de la réduction polynomiale (Mot par Mot : ($W = 32$))**

f (z) = $z^{233} + z^{74} + 1$ (with W = 32)

Entrées : Polynômes binaires $c(z)$ de degré au plus 324
Sortie : $c(z) = c(z) mod f(z)$

- Boucle $i = 10 : 6$
  - $T \leftarrow C[i\ ].$
  - $C[i\ -6] \leftarrow C[i\ -6] \oplus (T \ll 29)$
  - $C[i\ -5] \leftarrow C[i\ -5] \oplus (T \ll 4) \oplus (T \ll 3) \oplus T\ \oplus (T \ll 3) C[i-4] \leftarrow C[i\ -4] \oplus (T \ll 28) \oplus (T \ll 29)$
- Fin Boucle
- $T \leftarrow C[5] \gg 3$
- $C[0] \leftarrow C[0] \oplus (T \ll 7) \oplus (T \ll 6) \oplus (T \ll 3) \oplus T$
- $C[1] \leftarrow C[1] \oplus (T \gg 25) \oplus (T \gg 26)$
- $C[5] \leftarrow C[5] \& 0x7$

  Retourner $(C[5], C[4], C[3], C[2], C[1], C[0])$

Les symboles :

$\ll$ et $\gg$, représentent des décalages à gauche et à droite respectivement, $\oplus$ représente un XOR , & représente un ET logique et $\leftarrow$ représente une affectation de variable.

Les différents polynômes recommandés par le NIST, pour les champs $GF(2^m)$, sont décrits avec plus de détails dans [30].

2.3.4.4 Réduction polynomiale en un cycle d'horloge

Cette méthode est basée sur la structure fixe du polynôme irréductible, ainsi que l'addition décalée de ce dernier avec le résultat après chaque réduction polynomiale bit par bit.

Les figures 2.10 et 2.11., illustrent la réduction modulaire pour le résultat de la multiplication ou le carre polynomial, pour des opérandes appartenant à $GF(2^8)$, ayant comme polynôme de réduction $f(z) = z^8 + z^4 + z^3 + z^2 + 1$.

La même procédure reste valable pour les champs $GF(2^m)$.

Étant donné un opérande $c(z)$ issu de la multiplication où le carre polynomial, appartenant au champ $GF(2^8)$, dont la taille est $2 \times 8 - 2 + 1 = 15\ bits$, de 0 à 14 $bits$.

| Position Coefficients | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | $c_0$ | | | | | | | |
| 1 | | $c_1$ | | | | | | |
| 2 | | | $c_2$ | | | | | |
| 3 | | | | $c_3$ | | | | |
| 4 | | | | | $c_4$ | | | |
| 5 | | | | | | $c_5$ | | |
| 6 | | | | | | | $c_6$ | |
| 7 | | | | | | | | $c_7$ |

| Position Mappage | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | | | | | | | |
| 1 | | 1 | | | | | | |
| 2 | | | 1 | | | | | |
| 3 | | | | 1 | | | | |
| 4 | | | | | 1 | | | |
| 5 | | | | | | 1 | | |
| 6 | | | | | | | 1 | |
| 7 | | | | | | | | 1 |

Figure 2.10. : Mappage des 8 premiers coefficients de $c(z)$ sans réduction polynomiale.

Les premiers termes de $c_0$ à $c_7$, ne sont jamais réduits, du fait que leur degré est inferieur à celui du polynôme de réduction.

Le tableau 2.5 montre les valeurs des puissances des coefficients $c_8$ à $c_{14}$de c(z) .

Tableau 2.5. : Éléments $c_8$ à $c_{15}$ de $c(z)$ dans G$F(2^8)$.

| | | |
|---|---|---|
| $z^8 = z^4 + z^3 + z^2 + 1$ | $z^9 = z^5 + z^4 + z^3 + z$ | $z^{10} = z^6 + z^5 + z^4 + z^2$ |
| $z^{11} = z^7 + z^6 + z^5 + z^3$ | $z^{12} = z^8 + z^7 + z^6 + z^4$ | $z^{13} = z^9 + z^8 + z^7 + z^5$ |
| $z^{14} = z^{10} + z^9 + z^8 + z^6$ | | |

Après réduction polynomiale des éléments du tableau 2.5, on obtient le mappage de la figure 2.11 [41].

| Position Coefficients | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | Position Mappage | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 8 | $c_8$ | | $c_8$ | $c_8$ | $c_8$ | | | | | 8 | 1 | | 1 | 1 | 1 | | | |
| 9 | | $c_9$ | | $c_9$ | $c_9$ | $c_9$ | | | | 9 | | 1 | | 1 | 1 | 1 | | |
| 10 | | | $c_{10}$ | | $c_{10}$ | $c_{10}$ | $c_{10}$ | | | 10 | | | 1 | | 1 | 1 | 1 | |
| 11 | | | | $c_{11}$ | | $c_{11}$ | $c_{11}$ | $c_{11}$ | | 11 | | | | 1 | | 1 | 1 | 1 |
| 12 | $c_{12}$ | | $c_{12}$ | $c_{12}$ | | | $c_{12}$ | $c_{12}$ | | 12 | 1 | | 1 | 1 | | | 1 | 1 |
| 13 | $c_{13}$ | $c_{13}$ | $c_{13}$ | | | | | $c_{13}$ | | 13 | 1 | 1 | 1 | | | | | 1 |
| 14 | $c_{14}$ | $c_{14}$ | | | $c_{14}$ | | | | | 14 | 1 | 1 | | | 1 | | | |

Figure 2.11 : Mappage des 6 seconds coefficients de $c(z)$ après réduction polynomiale.

Les équations logiques issues de chaque colonne (à droite des figures 2.10) génèrent les valeurs des nouveaux coefficients de $c(z)$ comme suit :

- $c(0) = \mathrm{c}(0) \oplus \mathrm{c}(8) \oplus \mathrm{c}(12) \oplus \mathrm{c}(13) \oplus \mathrm{c}(14)$
- $c(1) = \mathrm{c}(1) \oplus \mathrm{c}(9) \oplus \mathrm{c}(13) \oplus \mathrm{c}(14)$
- $c(2) = \mathrm{c}(2) \oplus \mathrm{c}(8) \oplus \mathrm{c}(10) \oplus \mathrm{c}(12) \oplus \mathrm{c}(13)$
- $c(3) = \mathrm{c}(3) \oplus \mathrm{c}(8) \oplus \mathrm{c}(9) \oplus \mathrm{c}(11) \oplus \mathrm{c}(12)$
- $c(4) = \mathrm{c}(4) \oplus \mathrm{c}(8) \oplus \mathrm{c}(9) \oplus \mathrm{c}(10) \oplus \mathrm{c}(14)$
- $c(5) = \mathrm{c}(5) \oplus \mathrm{c}(9) \oplus \mathrm{c}(10) \oplus \mathrm{c}(11)$
- $c(6) = \mathrm{c}(6) \oplus \mathrm{c}(10) \oplus \mathrm{c}(11) \oplus \mathrm{c}(12)$
- $c(7) = \mathrm{c}(7) \oplus \mathrm{c}(11) \oplus \mathrm{c}(12) \oplus \mathrm{c}(13)$ (2.35)

La complexité matérielle est certainement plus importante que les approches précédentes. Pour notre implantation dans G$F(2^{163})$, cette approche a permis de gagner des centaines de cycles d'horloge.

2.3.4.5 Réduction du carré polynomial

Le résultat issu du carré polynomial d'un nombre appartenant à $GFGF(2^m)$, contient $m-1$ valeurs nulles, ce qui simplifie encore plus les équations de la section précédente.

En reprenant l'exemple précédent dans $GF(2^8)$, les tableaux 2.10 et 2.11., se compressent dans le tableau 2.12.

| Position Coeffi. | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | $c_0$ | | | | | | | |
| 1 | | | $c_2$ | | | | | |
| 2 | | | | | $c_4$ | | | |
| 3 | | | | | | | $c_6$ | |
| 4 | $c_8$ | | $c_8$ | $c_8$ | $c_8$ | | | |
| 5 | | | $c_{10}$ | | $c_{10}$ | $c_{10}$ | $c_{10}$ | |
| 6 | $c_{12}$ | | $c_{12}$ | $c_{12}$ | | | $c_{12}$ | $c_{12}$ |
| 7 | $c_{14}$ | $c_{14}$ | | | $c_{14}$ | | | |

| Position Mappage | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | | | | | | | |
| 2 | | | 1 | | | | | |
| 4 | | | | | 1 | | | |
| 6 | | | | | | | 1 | |
| 8 | 1 | | 1 | 1 | 1 | | | |
| 10 | | | 1 | | 1 | 1 | 1 | |
| 12 | 1 | | 1 | 1 | | | 1 | 1 |
| 14 | 1 | 1 | | | 1 | | | |

Figure 2.12. : Mappage des coefficients de $c(z)$ après réduction polynomiale du résultat du carré polynomial.

Pour une implantation hardware, les coefficients de c(z) deviennent, alors :

- $c(0) = \mathrm{c}(0) \oplus \mathrm{c}(8) \oplus \mathrm{c}(12) \oplus \mathrm{c}(14)$
- $c(1) = \mathrm{c}(14)$
- $c(2) = \mathrm{c}(2) \oplus \mathrm{c}(8) \oplus \mathrm{c}(10) \oplus \mathrm{c}(12)$
- $c(3) = \mathrm{c}(8) \oplus \mathrm{c}(12)$
- $c(4) = \mathrm{c}(4) \oplus \mathrm{c}(8) \oplus \mathrm{c}(10) \oplus \mathrm{c}(14)$
- $c(5) = \mathrm{c}(10)$
- $c(6) = \mathrm{c}(6) \oplus \mathrm{c}(10) \oplus \mathrm{c}(12)$
- $c(7) = \mathrm{c}(12)$

2.3.4.6 La méthode arbre de puissance

La méthode arbre de puissance est due à Knuth [42] , cet algorithme construit un arbre dont la racine est 1, représenté sur la figure 2.13.

La construction du (k+1) nième niveau se fait à partir du n$^{ième}$ niveau. En effet supposons que le n$^{ième}$ niveau construit. Considérons le nœud 0 du n$^{ième}$ niveau, de gauche vers la droite.

Le (k+1)$^{nième}$ niveau se construit en attachant les nœuds précédents.

Dans cette construction, on enlève les nœuds qui sont déjà apparus.

Pour calculer « **M$^e$ »,** on localise « **e** » dans l'arbre des puissances, la séquence est construite par les nœuds rencontrés sur le chemin pour arriver à « e ».

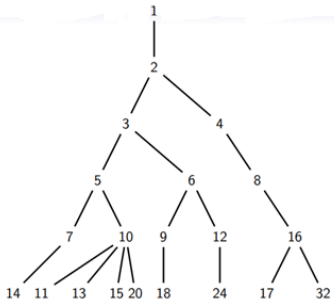



Figure 2.13 : méthode d'arbre de puissance à 5 niveaux.

**Remarque :**

Différents arbres peuvent être construits et plusieurs chemins peuvent être suivis pour arriver à la valeur désirée et il faut trouver le chemin le plus court.

Cette méthode s'adapte bien pour les petits exposants, car dans le cas des grands exposants les branches de l'arbre sont grandes et toutes les valeurs se trouvant sur ces branches doivent être stockées et cela exige un espace mémoire important.

### 2.3.5 L'inversion modulaire

L'inversion modulaire est l'une des opérations les plus complexes en arithmétique modulaire, principalement lorsque la taille du champ augmente, cas des $GF(2^m)$, en cryptographie. La problématique réside dans le nombre d'opérations nécessaires à l'exécution d'une seule inversion.

Il existe divers algorithmes permettant l'opération de l'inverse modulaire, nous citons les quatre plus connus comme la montre la figure ci-dessous.

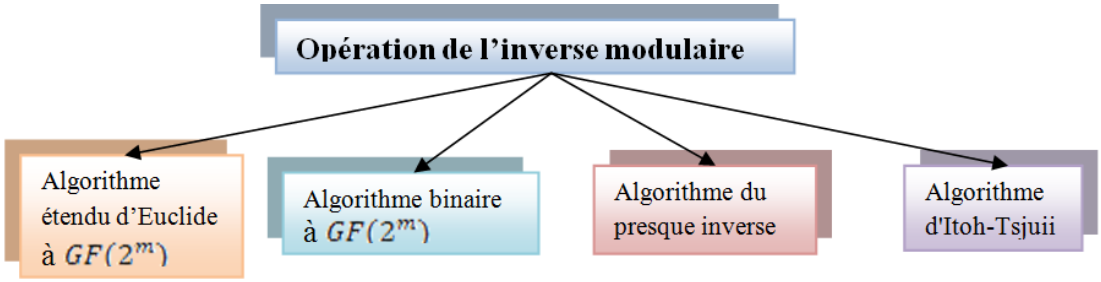



Figure 2.14 : les différents algorithmes de l'inverse modulaire.

La notion d'inverse modulaire dans les champs $GF(2^m)$, est définie comme suit :

Étant donné un polynôme non nul $a(z)$ dénoté par $a$, son inverse est l'élément unique appartenant à $GF(2^m)$ tel que $a.g = 1$ dans $GF(2^m)$ soit $a.g = 1 \, mod \, f$.

L'inverse est noté $a^{-1} mod \, f$ .

2.3.5.1 Algorithme étendu d'Euclide

Étant donné deux polynômes a et b, au moins l'un d'eux est non nul. Le plus grand diviseur commun appelé $pgdc$, note par $pdgc(a,b)$ est le polynôme de degré $d$ qui divise $a$ et $b$ en même temps.

2.3.5.2 Théorème

Étant donne deux polynômes binaires $a$ et $b$, alors le :

$pgdc(a,b) = pgdc(b - c.a, a)$ Pour tous les polynômes binaires $c$.

Dans la méthode Euclidienne classique, pour calculer le $pgdc$ de a et b, ou $deg(a) >= deg(b)$, $b$ est divisé par $a$ en vue d'obtenir le quotient $q$ et le reste $r$ satisfaisant :

$b = q.a + r$, Avec $deg(r) < deg(a)$, (2.36)

En utilisant le théorème précédent, $pgdc(a,b) = pgdc(r,a)$, alors le problème se restreint à trouver le $pgdc(a,r)$, avec $(a,r)$ comme polynômes ayant des degrés moindres comparés à $(a,b)$.

Ce procédé est répété jusqu'à ce que l'un des arguments est zéro, alors le résultat est directement obtenu car $pgdc(0,d) = d$. L'algorithme doit converger car les degrés des polynômes résultats décroissent à chaque itération.

Notons que le nombre de divisions est au plus égal a $k$ tel que $k = deg(a)$.

L'algorithme d'Euclide peut être étendu aux polynômes binaires g et h satisfaisant : $a.g + b.h = d$, Avec $d = pgdc(a,b)$.

L'algorithme de la section 2.4.2.1 suivant maintient les invariants :

$ag1 + bh1 = u$

$ag2 + bh2 = v$

L'algorithme termine lorsque $u = 0$,

Dans ce cas v= $pgdc(a,b)$ et $ag2 + bh2 = d$.

2.3.5.3 Algorithme d'Euclide Étendu

L'algorithme 2.8 détaille l'algorithme d'Euclide Étendu dans les champs GF($2^m$).

**Algorithme 2.8 : Euclide Étendu**

Entrée : Deux polynômes non binaires $a$and $b$tel que $deg(a) \leq deg(b)$.
Sortie : $d = gcd(a,b)$ et les polynômes binaires $g, h$satisfaisant$ag + bh = d$.

- $u \leftarrow a, v \leftarrow b$.
- $g1 \leftarrow 1, g2 \leftarrow 0, h1 \leftarrow 0, h2 \leftarrow 1$.
- $Tant\ que\ u = 0\ faire$
  - $j \leftarrow deg(u) - deg(v)$
  - $Si\ j < 0\ then: u \leftrightarrow v, g1 \leftrightarrow g2, h1 \leftrightarrow h2, j \leftarrow -j$ .
  - $u \leftarrow u + z\,j\,v$.

- $g1 \leftarrow g1 + z\,j\,g2, h1 \leftarrow h1 + z\,jh2.$
- $d \leftarrow v, g \leftarrow g2, h \leftarrow h2.$
- $Retourner\ (d, g, h).$

2.3.5.4 Algorithme d'Euclide Étendu pour l'inverse dans les champs $GF(2^m)$

L'algorithme 2.9 détaille l'algorithme d'Euclide Étendu pour l'inverse dans les champs GF($2^m$).

**Algorithme 2.9 : Euclide Étendu pour l'inverse dans les champs $GF(2^m)$**

Entrée : Polynôme $a$ non nul de degré au plus $m - 1$.
Sortie: $a^{-1} mod\ f$

- $1. u \leftarrow a, v \leftarrow f.$
- $g1 \leftarrow 1, g2 \leftarrow 0.$
- $Tant\ que\ u = 1\ faire$
  - $j \leftarrow deg(u) - deg(v).$
  - $Si\ j < 0\ alors: u \leftrightarrow v, g1 \leftrightarrow g2, j \leftarrow -j.$
  - $u \leftarrow u + z^j v.$
  - $g1 \leftarrow g1 + z^j\ g2.$

$Retourner(g1).$

2.3.5.5 L'algorithme binaire pour l'inversion dans $GF(2^m)$

L'algorithme ci dessous nous donne l'algorithme de, l'operateur d'inversion dans $\boldsymbol{GF(2^m)}$.

**L'algorithme 2.10 : algorithme binaire pour l'inversion dans $GF(2^m)$**

Entrée : Polynôme $a$ non nul de degré au plus $m - 1$.
Sortie: $a^{-1} mod\ f$

- $u \leftarrow a, v \leftarrow f$
- $g1 \leftarrow 1, g2 \leftarrow 0.$
- $.Tant\ que\ (u = 1\ et\ v = 1)\ faire$
  - $tant\ que\ z\ divize\ u\ faire$
    $u \leftarrow u/z.$
    $Si\ z\ divize\ g1\ alors\ g1 \leftarrow g1/z;\ sinon\ g1 \leftarrow (g1 + f)/z.$
  - $Tant\ que\ z\ divize\ v\ faire$
    $v \leftarrow v/z.$
    $If\ z\ divise\ g2\ alors\ g2 \leftarrow g2/z;\ sinon\ g2 \leftarrow (g2 + f)/z$
  - $Si\ deg(u) > deg(v)\ alors: u \leftarrow u + v, g1 \leftarrow g1 + g2;$
    $Sinon: v \leftarrow v + u, g2 \leftarrow g2 + g1.$
- $Si\ u = 1\ alors\ retourner(g1);\ sinon\ retourner(g2).$

2.3.5.6 Algorithme du presque inverse

Cet algorithme est une modification de l'algorithme binaire de l'inversion, dans lequel un polynôme g et un entier k sont d'abord calculés tel que : $a.g == z^k\ (mod\ f)$

La réduction est alors réalisée ainsi : $a^{-1} = z^k\ mod\ f$

Les invariants maintenus sont :

$$ag1 + bh1 = u$$

$$ag2 + bh2 = v.$$

Pour un certain couple (h1, h2) qui ne sont pas explicitement calculés.

- Détail de l'algorithme

**Algorithme 2.11 du Presque inverse**

Entrée : Un polynôme $a$ non nul de degré au plus$m-1$.
Sortie: $a^{-1}\ mod\ f$

- $u \leftarrow a, v \leftarrow f$ .
- $g1 \leftarrow 1, g2 \leftarrow 0, k \leftarrow 0.$
- $Tant\ que\ (u = 1\ et\ v = 1)\ faire$
  - $Tant\ que\ z\ divize\ u\ faire$
    $u \leftarrow u/z, g2 \leftarrow z \cdot g2, k \leftarrow k + 1.$
  - $Tant\ que\ z\ divize\ v\ do$
    $v \leftarrow v/z, g1 \leftarrow z \cdot g1, k \leftarrow k + 1.$
  - $Si\ deg(u) > deg(v)\ alors: u \leftarrow u + v, g1 \leftarrow g1 + g2.$
    $Sinon: v \leftarrow v + u, g2 \leftarrow g2 + g1.$
- $Si\ u = 1\ alors\ g \leftarrow g1;\ sinon\ g \leftarrow g2.$
- $Retourner\ (z^{-k} g\ mod\ f\ ).$

2.3.5.7 Algorithme d'Itoh-Tsjuii [7,43]

Le dernier petit théorème de Fermat stipule que l'égalité $A^{2^m-1} = 1$ est toujours vraie, alors $A^{2^{m-1}-1} = A^{-1}$, est une alternative pour calculer l'inverse modulaire, celle-ci est sujette à une très forte optimisation des ressources matérielles.

L'algorithme d'inversion d'Itoh–Tsujii est utilisé pour inverser des éléments dans un champ fini . Il a été introduit en 1988 et utilisé pour la première fois sur GF ($2^m$) en utilisant la représentation des éléments sur une base normale . Cependant, l'algorithme est générique et peut être utilisé pour d'autres bases, telles que la base polynomiale . Il peut également être utilisé dans n'importe quel corps fini GF($p^m$ ).

**Algorithme 2.12 : Algorithme d'Itoh-Tsjuii**

Entrée : A ∈ GF ( p m )
Sortie : $A^{-1}$

1. r ← ( $p^{m-1}$) / ( $p^{-1}$)
2. calculer $A^{r-1}$ dans GF ( p m )
3. calculer A r = $A^{r-1} \cdot A$
4. calculer ( A r ) $^{-1}$ dans GF ( p )
5. calculer $A^{-1} = (A\ r)^{-1} \cdot A^{r-1}$

retourne $A^{-1}$

Cet algorithme est rapide car les étapes 3 et 5 impliquent toutes deux des opérations dans le sous-champ GF(p). De même, si une petite valeur de p est utilisée, une table de correspondance peut être utilisée pour l'inversion à l'étape 4. La majeure partie du temps passé dans cet algorithme se situe à l'étape 2, la première exponentiation. C'est l'une des

raisons pour lesquelles cet algorithme est bien adapté à la base normale, car la quadrature et l'exponentiation sont relativement faciles dans cette base.

D'après l'état de l'art [43, 44], il s'avère que cet algorithme représente un bon compromis pour une implantation hardware.

Il existe aussi une autre alternative pour l'inversion par la méthode de Montgomery, [45,46].

Il existe en effet, dans la séquence d'Itoh-Tsujii un certain nombre de multiplications pouvant être lancées simultanément. L'algorithme est décrit dans Algorithme 2.13.

**Algorithme 2.13 : Algorithme d'Itoh-Tsujii paralléle**

**Données :** A non zero element $A \in GF(2^m)$
**Résultat** : $\alpha^{-1} \in GF(2^m)$
**1. Initialisation** : Reg1 ← A ; Reg2 ← 1 ; Pow ← 0 ;
2. pour i de 1 à $[\log_2(m-1)]$ **faire**
**si** $m_{i-1} = 1$ **alors**
2 Reg3 ← Reg1 ;
3 Reg2 ← $(Reg3)2^{Pow} * Reg1$ ;
4 Pow ← $Pow + 2^{i-1}$
5 Reg1 ← $Reg11^{2^{2^{i-1}}} * Reg1$ (en parallèle avec la ligne 3)
6 retourner $(Reg1\ 2^{Pow} * Reg2)^2$

Il existe en effet, dans la séquence d'Itoh-Tsujii un certain nombre de multiplications pouvant être lancées simultanément.

## 2.4 Courbes elliptiques

Les sections précédentes ont traitées des notions fondamentales nécessaires à la multiplication scalaire sur les courbes elliptiques. Nous allons tout au long des sections suivantes détailler des équations de ces formes très particulières [30, 47, 48, 49,50], comme illustre dans la figure 2.15.

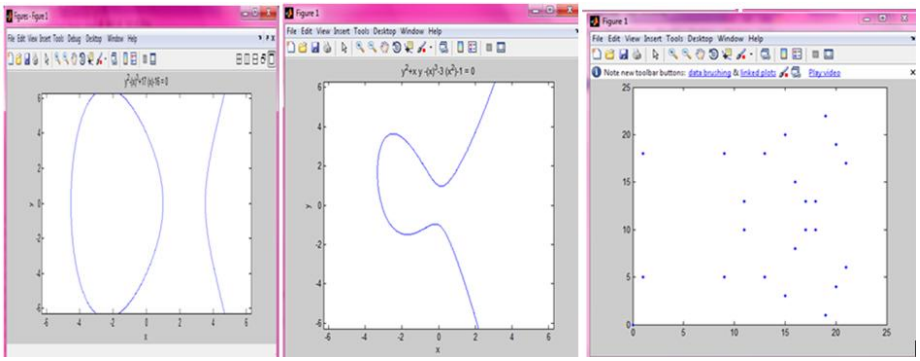


Figure 2.15 : Différentes formes de courbes elliptiques dans **R.**

La cryptographie basée sur les courbes elliptiques fait appel à différentes opérations. La figure 2.16 montre que ces opérations peuvent être hiérarchisées et met en avant les liens de dépendance. Les protocoles cryptographiques sont basés sur l'exponentiation. Ces deux opérations basiques sur les points sont des calculs sur leurs coordonnées, qui sont des éléments d'un corps.

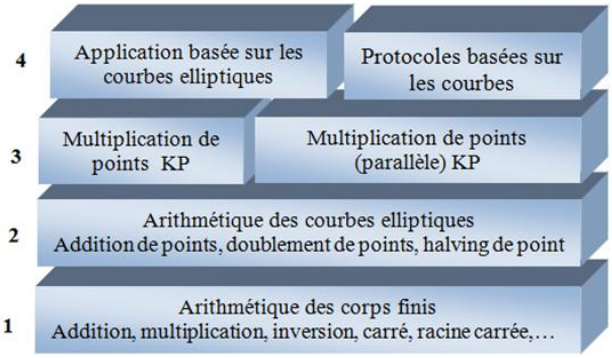



Figure 2.16 : Hiérarchie des opérations sur les courbes elliptiques en cryptographie [7].

2.4.1 Définition

Une courbe elliptique E définie sur un champ K est définie par l'équation :

$$E: y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \quad (2.37)$$

Avec $a_1, a_2, a_3, a_4, a_6 \in K$ et $\Delta \neq 0$, tel que : $\Delta$ est le déterminant de E, défini comme suit :

$$\Delta = -{d_2}^2 d_8 - 8{d_4}^3 - 27{d_6}^2 + 9d_2d_4d_6 \quad (2.38)$$

Tels que :

$$\begin{cases} d_2 = {a_1}^2 + 4a_2 \\ d_4 = 2a_4 + a_1a_3 \\ d_6 = {a_3}^2 + 4a_6 \\ d_8 = {a_1}^2a_6 + 4a_2a_6 - a_1a_3a_4 - a_2{a_3}^2 - {a_4}^2 \end{cases} \quad (2.39)$$

Si $L$ est un champ étendu de $K$ , alors l'ensemble des points $L$ -rationnel sur $E$ est :

$E(L) = \{(x, y) \in LxL : y^2 + a_1xy + a_3y - x^3 - a_2x^2 - +a_4x - a_6 = 0\} U \{\infty\}$ (2.40).

Avec $\infty$ étant le point à l' infini.

Notons que :

- L'équation (3.50) est appelée équation de Weierstrass, [7]
- On dit que E est défini sur K, car les éléments $a_1, a_2, a_3, a_4, a_6$ appartiennent a K, on écrit communément $E(K)$ ou $E/K$,
- La condition $\Delta \neq 0$ assure que la courbe est lisse, et que tout point de la courbe E, n'a qu'une seule tangente distincte.

2.4.2 Forme simplifiée de l'équation de Weierstrass [ 51]

2.4.1.1 Définition

Deux courbes elliptiques E1 et E2 définies sur K, représentées par les équations de Weierstrass :

$$\begin{cases} E1: y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \\ E2: y^2 + \overline{a_1}xy + \overline{a_3}y = x^3 + \overline{a_2}x^2 + \overline{a_4}x + \overline{a_6} \end{cases} \quad (2.41)$$

Sont dites iso-morphiques sur K, s'il existe $u, r, s$ et $t$ appartenant à K, avec $u \neq 0$, tel que le changement de variables :

$$(x, y) \rightarrow (u^2x + r, y + u^2sx + t) \quad (2.42)$$

Transforme l'équation $E1$ en $E2$. Cette transformation est appelée transformation admissible de variables.

2.4.2.1 Transformations isomorphes

L'équation (2.41) définie sur K, peut être simplifiée considérablement en appliquant un changement de variables admissibles. La nouvelle forme sera utilisée dans notre étude, et sera la base de notre implantation hardware.

Différentes formes iso-morphique à l'équation (2.41), dépendamment des valeurs de la caractéristique de K.

- Si la caractéristique de K est différente de 2 ou de 3, le changement de variables admissibles se fait comme suit :

$$(x, y) \rightarrow \left(\frac{x - 3{a_1}^2 - 12a_2}{36}, \frac{y - 3a_{1x}}{216} - \frac{{a_1}^2 + 4a_1a_2 - 12a_3}{24}\right) \quad (2.43)$$

Transformant ainsi la courbe E à la forme suivante :

$$y^2 = x^3 + ax + b \quad (2.44)$$

Avec $a, b \in K$ et $\Delta = -16(4a^3 + 27b^2)$ (2.45)

- Si la caractéristique de K est égale à 2, si $a_1 \neq 0$ , le changement de variables admissibles se présente comme suit :

$$(x, y) \rightarrow \left({a_1}^2x + \frac{a_3}{a_1}\ , {a_1}^3y + \frac{{a_1}^2a_4 + {a_3}^2}{{a_1}^3}\right) \quad (2.46)$$

Transformant la courbe E comme suit :

$$y^2 + xy = x^3 + ax^2 + b \quad (2.47)$$

Avec $a, b \in K$ et $\Delta = b$ . (2.48)

Ce type de courbe est appelé courbe non-super singulière

- Si la caractéristique de K est égale à 2, et si $a_1 = 0$ , le changement de variables admissibles se présente comme suit :

$$(x, y) \rightarrow (x + a_2\ , y) \quad (2.49)$$

Transformant la courbe E comme suit :

$$y^2 + cy = x^3 + ax + b \quad (2.50)$$

Avec $a, b, c \in K$ et $\Delta = c^4$ (2.51)

Si la caractéristique de K est égale à 3, et si ${a_1}^2 \neq -a_2$ , le changement de variables admissibles se présente comme suit :

$$(x, y) \rightarrow (x + \frac{d_4}{d_2}, y + a_1 x + a_1 \frac{d_1}{d_4} + a_3) \quad (2.52)$$

Avec $d_2 = {a_1}^2 + a_2$ et $d_4 = a_4 - a_1 a_3$ (2.53)

Transformant la courbe E à :

$$y^2 = x^3 + ax^2 + b \quad (2.54)$$

Tels que : $a, b \in K$ et $\Delta = -a^3 b$ (2.55)

Ce type de courbe est appelée non-super singulière.

- Si la caractéristique de K est égale à 3, et si ${a_1}^2 = -a_2$ , le changement de variables admissibles se présente comme suit :

$$(x, y) \rightarrow (x, y + a_1 x + a_3) \quad (2.56)$$

Transformant la courbe E à :

$$y^2 = x^3 + ax + b \quad (2.57)$$

Tels que : $a, b \in K$ et $\Delta = -a^3$ (2.58)

Ce type de courbe est appelée courbe super-singulière

On remarque qu'il y'a plusieurs transformations, chaque type est orientée vers des champs premiers [49,50], nous allons nous concentrer sur la forme la plus adaptée aux applications de cryptographie binaires [52, 53, 54,55], représentée par la courbe $E: y^2 + xy = x^3 + ax^2 + b$ .

### 2.4.3 Loi de groupes des courbes elliptiques

Le tableau 2.6 nous donne les différentes interprétations des différentes opérations sur les courbes elliptiques.

Tableau 2.6; Interprétation géométrique et une interprétation algébrique.

| **Sommes sur une Courbe Elliptique** | |
|---|---|
| **Interprétation géométrique** | **Interprétation algébrique** |
| • Soit un point P, alors – P est le point sur la courbe à l'opposé à l'axe x<br>• Soit deux points P et Q , P+Q = -R Où R est le point sur la courbe à l'intersection de la ligne PQ. | • Pour $R^2$, on peut déduire des formules explicites en fonctions des coordonnées x, y des points de points de P et Q ainsi que des paramètres de la courbe.<br>• Ces formules sont directement |

| | |
|---|---|
| • Doublement , si P = Q<br>alors P+P = P+P= 2P = R, où R est l'intersection de la droite tangente à la courbe au point P . | généralisables dans $Zp^2$. |
| Dans les deux cas (E,+)forme un groupe , si a et b sont bien choisis | |
| • L'ensemble des points d'une courbe elliptique sur un corps *K* muni de l'opération d'addition ⊕ constitue un groupe avec le point à l'infini comme élément neutre.<br>➢ L'élément neutre : pour tout point *P* de la courbe, $P+\infty=P$<br>➢ La commutativité : P1+P2=P2+P1<br>➢ L'inverse de P est -P = (x1, - y1) on a alors : (x,y)+(x,-y)=∞<br>➢ Associativité : $(P_1+P_2)+P_3=P_1+(P_2+P_3)$. | |

Soit une courbe E définie sur un champ K. il existe une règle géométrique qui relie l'addition de deux points appartenant à la courbe E, à un troisième point. L'ensemble des points de E forme avec l'addition un groupe abélien, avec le point à l'infini comme élément neutre. C'est ce type de groupe qui est utilisé actuellement dans la construction du système de cryptographie à base de courbes elliptiques.

2.4.4 Propriétés du groupe

- Élément identité : $P+\infty=\infty+P=P$ pout tout point $P \in E(GF(2^m))$
- Négatif de P : si $P=(x,y) \in E(GF(2^m))$ alors$(x,y)+(x,x+y)=\infty$, alors le point $(x,x+y)=-P$ est appelé négatif de $P$ ; notons que si –P est un point de $E(GF(2^m))$ et $-\infty=\infty$.

2.4.1.2 Addition de deux points elliptiques

Étant donné deux points $P=(x_1,y_1)$ et $Q=(x_2,y_2) \in E(GF(2^m))$, Tels que : $P \neq \pm Q$

Alors : $P+Q=R=(x_3,y_3)$ (2.59)

Tel que : $\begin{cases} x_3=\lambda^2+\lambda+x_1+x_2+a \\ y_3=\lambda(x_1+x_3)+x_3+y_1 \end{cases}$ (2.60)

Avec : $\lambda=\frac{y_1+y_2}{x_1+x_2}$ (2.61)

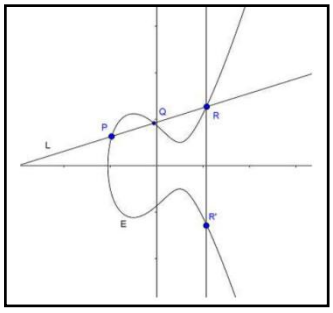



Figure 2.17. : Addition géométrique de deux points appartenant à la même courbe elliptique (P+Q=R).

Il existe quelques subtilités concernant l'addition des points sur les courbes elliptiques qui doivent être abordées. D'abord, que se passe-t-il si nous voulons additionner un point P à lui-même ? Imaginez ce qui arrive à la ligne L reliant P et Q si le point Q glisse le long de la courbe et se rapproche de P. Dans la limite, comme Q approche P, la ligne L devient la ligne tangente de E à P.

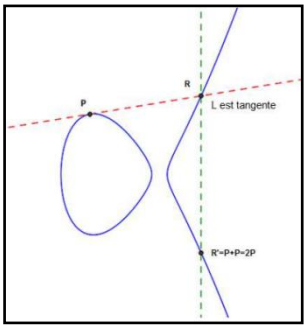



Figure 2.18. : L'addition de P à lui-même.

Ainsi, pour ajouter P à lui-même, prenons simplement la ligne L pour tangente à E en P, comme illustré dans la figure 2.3. Ensuite, L coupe E en P et en un autre point R. Dans un certain sens, L coupe toujours E en trois points, mais P compte deux fois d'entre eux. Un deuxième problème potentiel se pose avec la «loi d'addition», est que si nous essayons d'additionner un point P = (a, b) à son point symétrique P ′ = (a, −b) sur l'axe des X [56]. La ligne L qui traverse P et P ′ est la ligne verticale x = a, et cette ligne coupe E en deux points seulement P et P′.

2.4.4.1 Doublement d'un point elliptique

Étant donné un point $P = (x_1, y_1) \in E(GF(2^m))$

Tels que : $P \neq -P$

Alors : $2P = R = (x_3, y_3)$ (2.62)

Tel que : $\begin{cases} x_3 = \lambda^2 + \lambda + a = {x_1}^2 + \frac{b}{{x_1}^2} \\ y_3 = {x_1}^2 + \lambda x_3 + x_3 \end{cases}$ (2.63)

Avec : $\lambda = x_1 + \frac{y_1}{x_1}$ (2.64)

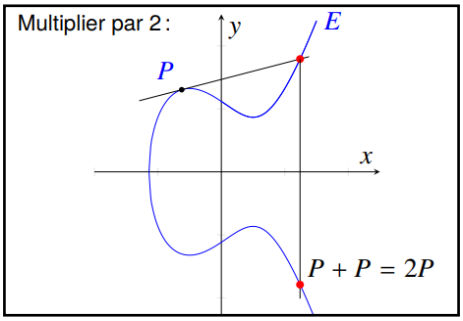


Figure 2.19 : Addition d'un point à lui-même, géométriquement, sur une courbe elliptique (doublement elliptique $P + P = 2P = R$).

**Algorithme 2.14 :** Algorithme doublement-et-addition pour calculer Q = kP

**Données** : k = $\sum_{i=0}^{l-1} k_i \ 2^i$ et P ∈ E(Fp)
**Résultat** : Q = kP
**1** Q ← ∞;
**2 pour** i de 0 à l − 1 **faire**
**3** **si** ki = 1 **alors**
**4** Q ← Q + P
**5** **fin**
**6** P ← 2P
**7 fin**
**reourner Q**

La figure 2.20 représente des courbes elliptiques de doublement de point.

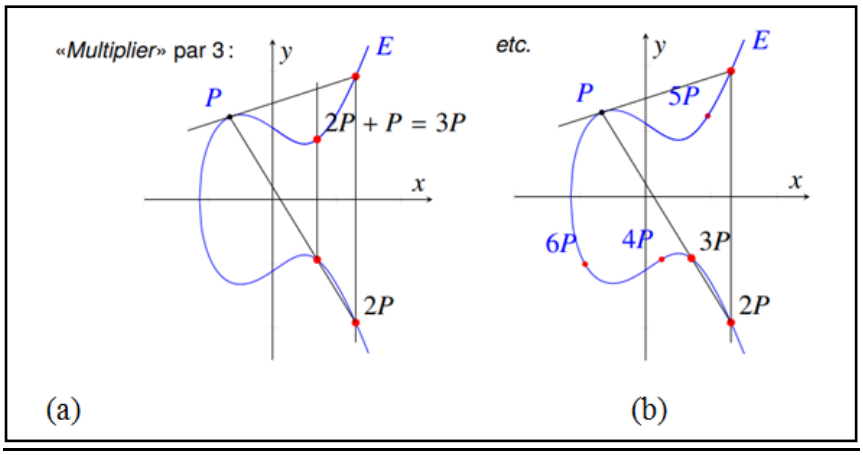



Figure 2.20 : Courbe elliptique et loi de composition (a) :2P+P =3P=R et (b) : R= 6P).

2.4.4.2 Exemple d'une courbe E définie sur $GF(2^4)$

Comme exemple, prenons la courbe non super-singulière $E: y^2 + xy = x^3 + ax^2 + b$, ayant comme polynôme de réduction $f(z) = z^4 + z + 1$.

Chaque élément $a_3z^3 + a_2z^2 + a_1z^1 + a_0$ représenté par le nombre binaire $(a_3a_2a_1a_0) \in GF(2^4)$ de taille 4 bits.

Cas de $a = z^3$ et $b = z^3 + 1$, la courbe $E$ devient alors :

$$E: y^2 + xy = x^3 + z^3x^2 + (z^3 + 1) \qquad (2.65)$$

Dont l'ensemble des points d $E\big(GF(2^4)\big)$ est représenté dans la Tableau 2.7.

Tableau 2.7. : Ensemble des points de la courbe $E: y^2 + xy = x^3 + z^3x^2 + z^3 + 1$ définie sur $GF(2^4)$.

| | | | | |
|---|---|---|---|---|
| ∞ | (0011,1100) | (1000,0001) | (1000,0001) | (1100,0000) |
| (0000,1011) | (0011,1111) | (1000,1001) | (1000,1001) | (1100,1100) |

| | | | | |
|---|---|---|---|---|
| (0001,0000) | (0101,0000) | (1001,0110) | (1001,0110) | (1111,0100) |
| (0001,0001) | (0101,0101) | (1001,1111) | (1001,1111) | (1111,1011) |
| (0010,1101) | (0111,1011) | (1011,0010) | (1011,0010) | |
| (0010,1111) | (0111,1100) | (1011,1001) | (1011,1001) | |

En additionnant les points (0010,1111) + (1100,1100), on obtient le point (0001,0001) appartenant aussi à la courbe $E(GF(2^4))$.

2.4.4.3 Ordre d'une courbe elliptique

Étant donné une courbe elliptique E définie sur $GF(q)$. Le nombre de points dans $E(GF(q))$ est noté : $\#E(GF(q))$, car l'équation de Weierstrass admet au plus deux solutions pour chaque point $x \in GF(q)$[8].

2.4.5 Théorème de Hasse [57]

Soit une courbe elliptique E définie sur $GF(q)$, alors :

$$q + 1 - 2\sqrt{q} \leq \#E\big(GF(q)\big) \leq q + 1 + 2\sqrt{q} \qquad (2.66)$$

L'intervalle $[q + 1 - 2\sqrt{q}\,, q + 1 + 2\sqrt{q}]$ est appelé intervalle de Hasse, une alternative à ce théorème considère que :

$$\#E\big(GF(q)\big) = q + 1 - t \qquad (2.67)$$

tel que $|t| \leq 2\sqrt{q}$, (2.68)

« t » est appelé trace of de $E$ sur $GF(q)$

Sachant que $2\sqrt{q}$est considérablement petit comparé à $q$ , alors :

$$\#E\big(GF(q)\big) \approx q \qquad (2.69)$$

2.4.5.1 Théorème des ordres admissibles des courbes elliptiques

Soit $q = p^m$, et $p$ la caractéristique de $GF(q)$, il existe une courbe elliptique $E$ définie sur $GF(q)$, avec $\#E\big(GF(q)\big) = q + 1 - t$, si et seulement si l'une des conditions suivantes est conservée :

- $t \not\equiv 0 \pmod p$ et $t^2 \leq 4q$
- m est impair et l'une des conditions suivantes est satisfaite :
  - cas ou $t = 0$
  - cas $t^2 = 2q$ et $p = 2$
  - ou $t^2 = 3q$ et $p = 3$
- m est pair et l'une des conditions suivantes est satisfaite :
  - cas ou $t^2 = 4q$
  - cas ou $t = q$ et $p \not\equiv 1 \pmod 3$
  - ou $t = 0$ et $p \not\equiv 1 \pmod 4$

Comme conséquence à ce théorème: Étant donne un nombre premier $p$ et un entier $t$ satisfaisant la condition : $t \leq 2\sqrt{q}$ , alors il existe une courbe elliptique $E\big(GF(q)\big)$ tel que $\#E\big(GF(q)\big) = q + 1 - t$.

L'exemple suivant illustre ce théorème,

- Soit la courbe $E: y^2 = x^3 + ax + b$ définie sur $GF(37)$, le tableau 2.8., liste pour chaque entier $n \in [37 + 1 - 2\sqrt{37}, 37 + 1 + 2\sqrt{37}]$ les différents coefficients $(a, b)$ avec $\#E(GF(37)) = n$

Tableau 2.8. : Les ordres admissibles $n = \#E(GF(37))$ de la courbe elliptique $E: y^2 = x^3 + ax + b$ définie sur $GF(37)$.

| $n$ | $(a,b)$ | $n$ | $(a,b)$ | $n$ | $(a,b)$ | $n$ | $(a,b)$ | $n$ | $(a,b)$ |
|---|---|---|---|---|---|---|---|---|---|
| 26 | (5,0) | 31 | (2,8) | 36 | (1,0) | 41 | (1,16) | 46 | (1,11) |
| 27 | (0,9) | 32 | (3,6) | 37 | (0,5) | 42 | (1,9) | 47 | (3,15) |
| 28 | (0,6) | 33 | (1,13) | 38 | (1,5) | 43 | (2,9) | 48 | (0,1) |
| 29 | (1,12) | 34 | (1,18) | 39 | (0,3) | 44 | (1,7) | 49 | (0,2) |
| 30 | (2,2) | 35 | (1,8) | 40 | (1,2) | 45 | (2,14) | 50 | (2,0) |

Notons que l'utilité de l'ordre $\#E\big(GF(q)\big)$ concerne la définition de la super singularité de la courbe $E$.

2.4.5.2 Définition de la super-singularité

Étant donné $p$ la caractéristique de $GF(q)$, une courbe elliptique $E$ définie sur $GF(q)$ est dite :

- Super-singulière si $p$ divise $t$, $t$ étant la trace,
- Non super-singulière si $p$ ne divise pas $t$.

Si E est une courbe elliptique définie sur $GF(q)$ , alors $E$ est aussi définie sur toute extension $GF(q^n)$ du champ $GF(q)$.

Le groupe $GF(q)$ des points $GF(q)$ −rationnels est un sous-groupe du groupe $GF(q^n)$ des points $GF(q^n)$ rationnels, et donc :

$\#E\big(GF(q)\big)$Divise $\#E\big(GF(q^n)\big)$

Si $\#E\big(GF(q)\big)$ est connu, alors $\#E\big(GF(q^n)\big)$ peut être déterminé par le théorème suivant.

2.4.5.3 Théorème de l'ordre récursif

Soit E une courbe elliptique définie sur $GF(q)$, et $\#E\big(GF(q)\big) = q + 1 - t$, alors :

$\#E\big(GF(q^n)\big) = q^n + 1 - V_n$ , pour tout $n \geq 2$ (2.70)

Avec $\{V_n\}$ la séquence définie récursivement par :

$V_0 = 2, \qquad V_1 = t$

Et $V_n = V_1 V_{n-1} - qV_{n-2}$ pout tout $n \geq 2$. (2.71)

2.4.6 Structure de groupe

Le théorème suivant décrit la structure de groupe de $E(GF(q))$. $\boldsymbol{Z}_n$ Sera utilise pour décrire la cyclicité d'ordre $n$.

2.4.6.1 Cyclicité

Soit E une courbe elliptique définie sur $GF(q)$, alors $E(GF(q))$ est isomorphe à $\boldsymbol{Z}_{n1} \oplus \boldsymbol{Z}_{n2}$, ou $n_1, n_2$ sont des entiers positifs unique tels que $n_2$ divise $n_1$ et $q-1$.

Notons que : $\#\, E(GF(q)) = n_1 . n_2$ (2.72)

- Si $n_2 = 1$, alors $E(GF(q))$ est un groupe cyclique.
- Si $n_2 > 1$, alors $E(GF(q))$ est dit avoir un rang égal a deux.
- Si $n_2$ est un entier suffisamment petit (i.e. : 2,3 ou 4), on dit parfois que $E(GF(q))$ est presque cyclique.

Sachant que $n_2$ divise $n_1$ et $q-1$, alors $E\big(GF(q)\big)$ est cyclique ou presque cyclique pour la plupart des courbes elliptiques E définies sur $GF(q)$.

2.5 Courbes elliptiques sur domaine fini

On n'utilise que des valeurs entières positives pour les coordonnées des points sur la courbe, on utilise le modulo pour «replier» le domaine de ces points dans un espace réduit cela implique qu'on obtient un ensemble fini de points.

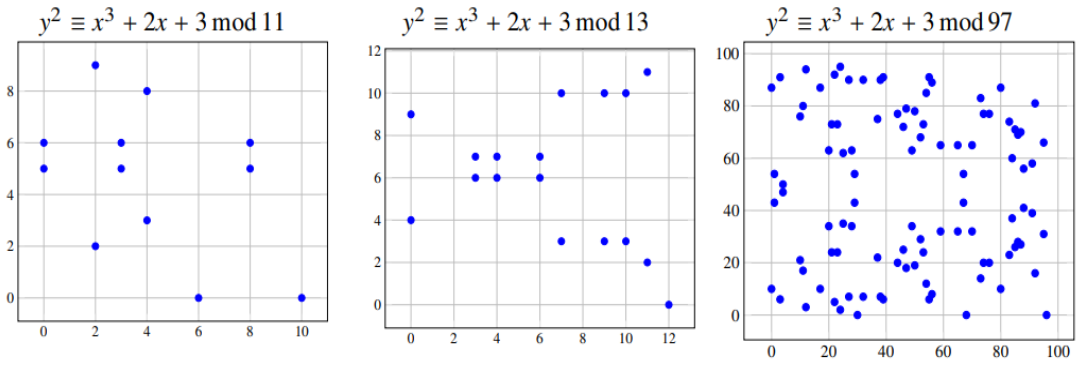



Figure 2.21 : Représentation des points sur domaine fini.

On observe que :

- plus la taille de p est grande, plus le domaine $\boldsymbol{F_P}$ est important ;
- la distribution des points semble aléatoire et recouvre la surface de manière régulière ;
    - Notion de groupe :

- les points appartenant à la courbe elliptique définie sur Fp définissent un groupe :
  - l'addition d'un point avec un autre et la multiplication scalaire donnent chacune un point appartenant à cet ensemble ;
  - le nombre de points appartenant à ce groupe définit l'ordre du groupe ; Exemple : pour $y^2 \equiv x^3 + 2x + 3 \bmod 11$ on dénombre 12 points, et 17 points pour $y^2 \equiv x^3 + 2x + 3 \bmod 13$.

### 2.5.1 Exemple de la courbe $E: y^2 = x^3 + 4x + 20$

La courbe est définie sur $GF(29)$, avec $\#E\big(GF(29)\big) = 37$

Puisque 37 est premier, $E(GF(29))$ est un groupe cyclique et n'importe quel point de $E(GF(29))$, à part le point à l'infini ∞ est un point générateur de $E(GF(29))$.

Le tableau 2.9 montre que le point P=(1,5) appartenant à la courbe E peut générer tous les points de $E(GF(29))$.

Tableau 2.9. : Ensemble des valeurs générées par le point P=(1,5), de la courbe E : $E: y^2 = x^3 + 4x + 20$ définie sur $GF(29)$.

| | | | | |
|---|---|---|---|---|
| 0P=∞ | 8P=(8,10) | 16P=(0,22) | 24P=(16,2) | 32P=(6,17) |
| 1P=(1,5) | 9P=(14,23) | 17P=(27,2) | 25P=(19,16) | 33P=(15,2) |
| 2P=(4,19) | 10P=(13,23) | 18P=(2,23) | 26P=(10,4) | 34P=(20,26) |
| 3P=(20,3) | 11P=(10,25) | 19P=(2,6) | 27P=(13,6) | 35P=(4,10) |
| 4P=(15,27) | 12P=(19,13) | 20P=(2,23) | 28P=(14,6) | 36P=(1,24) |
| 5P=(6,12) | 13P=(16,27) | 21P=(2,6) | 29P=(8,19) | |
| 6P=(17,19) | 14P=(5,22) | 22P=(27,27) | 30P=(24,7) | |
| 7P=(24,22) | 15P=(3,1) | 23P=(0,7) | 31P=(17,10) | |

### 2.5.2 Exemple du groupe $GF(2^4)$

La courbe est représentée par le polynôme de réduction suivant:

$f(z) = z^4 + z + 1$, cas de la courbe : $E: y^2 + xy = x^3 + z^3x^2 + (z^3 + 1)$ avec : $\#E(GF(2^4)) = 22$. Où $E(GF(2^4))$ est cyclique, et le point $P = (z^3, 1) = (1000{,}0001)$ a un ordre égal a 11, ces multiples sont mentionnes dans le tableau 2.10.

Tableau 2.10. : Multiples du point $P = (z^3, 1) = (1000{,}0001)$ d'ordre égal à 11 dans $E(GF(2^4))$

| | | | |
|---|---|---|---|
| 0P=∞ | 3P=(1100,1001) | 6P=(1011,1001) | 9P=(1001,0110) |
| 1P=(1000,0001) | 4P=(1111,1011) | 7P=(1111,0100) | 10P=(1000,1001) |
| 2P=(1001,1111) | 5P=(1011,0010) | 8P=(1100,1100) | |

## 2.6 Représentation des points et lois de groupes

Lors des calculs dans les courbes elliptiques, il s'avère que la division modulaire est répétitive, cela tend à alourdir la procédure de calcul, alors par des projections mathématiques de conversion des coordonnées, les équations de la courbe elliptique prennent une forme plus linéaire.

### 2.6.1 Les différentes coordonnées

Comme nous l'avons vu précédemment, l'ensemble des points d'une courbe elliptique et un point à l'infini forment un groupe abélien. L'efficacité du calcul des opérations de groupe, l'addition et le doublement de points, est primordiale. La complexité de ces calculs varie grandement suivant la représentation choisie des points de la courbe. Dans cette section, nous traitons à la fois le cas de courbes elliptiques définies sur des corps finis de caractéristique strictement supérieure à 3 mais aussi les corps de caractéristique 2.

La division est l'opération la plus couteuse dans un corps fini, on estime généralement qu'elle vaut entre 10 et 100 multiplications (au bas mot, selon l'algorithme employé, la caractéristique du corps mais aussi la taille même des éléments, en bits). Jusqu'ici, lorsque nous souhaitions ajouter un point à un autre, ou bien le doubler, il était nécessaire de calculer une inversion. Il existe différentes représentations des points des courbes elliptiques afin de réduire le nombre d'inversions à sa plus simple expression. Les plus populaires sont sans doute les coordonnées projectives [58] et les coordonnées Jacobiennes [59].

#### 2.6.1.1 Coordonnées affines

Soit E une courbe elliptique définie sur un corps fini Fq d'´equation (affine) :

$y^2 = x^3 + ax + b$.

Soit P1 = (x1, y1), P2 = (x2, y2) ∈ E(Fq) deux points de la courbe. Les formules d'addition et de doublement sont données sur la figure 2.7. On remarque que ces opérations nécessitent le calcul d'une inversion dans Fq qui est une opération complexe et très coûteuse. Ces coordonnées sont donc très peu utilisées pour une implémentation. On note ce système de coordonnées A.

#### 2.6.1.2 Coordonnées projectives

Concernant les coordonnées projectives, le principe est de rajouter à un point P=(x,y) une troisième coordonnée z qui va accumuler les inversions, comme nous pourrions le faire dans les nombres rationnels.

Soit K un champ et soit c et d deux entiers positifs [7,29],

La notion d'équivalence "$\sim$" du groupe $K^3\backslash\{(0,0,0)\}$ d'éléments non nuls, sur K est définie comme suit :

$$(X_1, Y_1, Z_1) \sim (X_2, Y_2, Z_2) \quad (2.73)$$

Si :

$$X_1 = \lambda^c, \qquad Y_1 = \lambda^d$$

$$Z_1 = \lambda Z_2 \text{ Pour certains } \lambda \in K^* \quad (2.74)$$

Alors la classe d'équivalence (X, Y, Z) $\in K^3\backslash\{(0,0,0)\}$ est :

$$(X:Y:Z) = \{(\lambda^c X, \lambda^d Y, \lambda Z): \lambda \in K^*\} \quad (2.75)$$

- $(X:Y:Z)$ est appelé coordonnée projective de $(X, Y, Z)$
- $(X, Y, Z)$ est appelé coordonnée représentative de $(X:Y:Z)$

Notons que si $Z \neq 0$, alors $(\lambda^c X/Z\,, \lambda^d Y/Z\,, 1)$ est aussi un représentatif de (X : Y : Z), et est le seul représentatif avec la coordonnée Z=1.

Donc il y'a une correspondance 1:1 entre l'ensemble des coordonnées projectives :

$$P(K)^* = \{(X, Y, Z): X, Y, Z \in K, Z \neq 0\} \quad (2.76)$$

Et l'ensemble des coordonnées affines :

$$A(K) = \{(x, y): x, y \in K\} \quad (2.77)$$

L'ensemble des points :

$$P(K)^0 = \{(X, Y, Z): X, Y, Z \in K, Z = 0\} \quad (2.78)$$

Est appelée ligne à l'infini, car ses points ne correspondent à aucune coordonnées affines.

La forme projective de l'équation de Weierstrass, est obtenue en remplaçant les coordonnées (x, y) par $(X/Z^c\,, Y/Z^d)$.

Le tableau ci-dessous nous résume les équations pour différentes représentations des coordonnées affines et projectives standard.

Tableau 2.11 : les équations des différentes coordonnées.

| **Représentation** | **Addition** | **Doublement** |
|---|---|---|
| AFFINE | $x_3 = ((y_1+y_2)/(x_1+x_2))^2+((y_1+y_2)/(x_1+x_2)) +x_1+x_2+a$<br>$y_3 = ((y_1+y_2)/(x_1+x_2))^3+(x_2+a+1) *((y_1+y_2)/(x_1+x_2))+x_1+x_2+a+y_1$ | $x_3 = (x_1+y_1/x_1)^2+(x_1+y_1/x_1) +a$<br>$y_3 = (x_1+y_1/x_1)^3+ (x_1+a+1)*(x_1+y_1/x_1) +a+y_1$ |
| Projective Standard | $X_3=BE$; $Y_3= C(AX_1+Y_1B)Z_2+(A+B)E$<br>$A= Y_1Z_2+Z_1Y_2$ ; $B= X_1Z_2+Z_1X_2$ ;<br>$C= B^2$ ; $D=Z_1Z_2$<br>$E= (A^2+AB+AC)\,D+BC$ | $X_3=CE$;<br>$Y_3= (B+C)E+A^2C$ ; $Z_3=CD$<br>Avec : $A =X_1^2$ ;<br>$B= A+Y_1Z_1$ $C=X_1Z_1$ |

2.7 Méthode de Montgomery

Étant donne deux points $R = (x_1, y_1)$ et $Q = (x_2, y_2)$ deux points appartenant à la courbe elliptique de l'équation (3.59), alors $R + Q = (x_3, y_3)$ et $R - Q = (x_4, y_4)$, appartiennent aussi a la courbe, et l'on peut démontrer que :

$$x_3 = x_4 + \frac{x_1}{x_1 + x_2} + \left(\frac{x_1}{x_1 + x_2}\right)^2 \quad (2.79)$$

Montgomery remarqua que l'équation (3.91) ne requiert que les coordonnées d'abscisse de $R, Q, R - Q$ pour déterminer $R + Q$.

En remplaçant la coordonnée $x$ du point $R$ par $\frac{X}{Z}$ , alors le point $2R = (X_2, -, Z_2)$ converti en coordonnées projectives devient alors :

$$\begin{cases} X_2 = X_1^4 + b \times Z_1^4 \\ Z_2 = X_1^2 \times Z_1^2 \end{cases} \quad (2.80)$$

Par la même méthode, les coordonnées projectives de $R + Q$ peuvent être calculées comme une fraction de $\frac{X_3}{Z_3}$ comme suit :

$$\begin{cases} Z_3 = (X_1 \times Z_2 + X_2 \times Z_1)^2 \\ X_3 = x \times Z_3 + (X_1 \times Z_2)(X_2 \times Z_1) \end{cases} \quad (2.81)$$

Pour revenir aux coordonnées affines, les transformations suivantes sont utilisées :

$$x_3 = \frac{X_3}{X_3} \quad (2.82)$$

$$y_3 = (x + x_3)[(X_1 + x.Z_1)(X_2 + x.Z_2) + (x^2 + y)(Z_1.Z_2)](x.Z_1.Z_2)^{-1} + y \quad (2.83)$$

Les coordonnées $(x, y)$ dans l'équation (2.95) appartiennent au premier point initial.

**Algorithm 2.15 : Montgomery point multiplication (for elliptic curves over FG($2^m$)**

INPUT: k = ($k_{t-1}$,..., $k_1$, $k_0$)$_2$ with kt−1 = 1, P = (x, y) ∈ E($F_{2^m}$ ).
OUTPUT: k P.

- $X_1 \leftarrow x$, $Z_1 \leftarrow 1$, $X_2 \leftarrow x^4 + b$, $Z_2 \leftarrow x^2$. {Compute (P,2P)}
- For i from t −2 downto 0 do
  - If $k_i = 1$ then
    T ←$Z_1$, $Z_1 \leftarrow (X_1Z_2 + X_2Z_1)^2$, $X_1 \leftarrow xZ_1 + X_1X_2T\ Z_2$.
    T ← $X_2$, $X_2 \leftarrow X^4{}_2 + bZ^4{}_2$, $Z_2 \leftarrow T^2Z^2{}_2$
  - Else
    T ←$Z_2$, $Z_2 \leftarrow (X_1Z_2 + X_2Z_1)^2$, $X_2 \leftarrow x\ Z_2 + X_1X_2Z_1T$ .
    T ← $X_1$, $X_1 \leftarrow X^4{}_1 + bZ^4{}_1$, $Z_1 \leftarrow T^2Z^2{}_1$.
- $x_3 \leftarrow X_1/Z_1$.
- $y_3 \leftarrow (x + X_1/Z_1)[(X_1 + xZ_1)(X_2 + xZ_2) + (X^2 + y)(Z_1Z_2)](xZ_1Z_2)^{-1} + y$.

Return($x_3$, $y_3$)

2.8 Cout des différentes opérations pour différentes projections

le nombre d'opération élémentaire, addition ,multiplication et la mise au carré dans le corps *K*, nécessaire pour effectuer l'addition de deux points ou dédoublement de points

sur une courbe elliptique *E* selon les différents types de coordonnées**.** Le tableau 2.8 nous dresse le cout de chaque projection.

Tableau 2.12 : Nombre d'opérations pour calculer une addition et un doublement sur :

$$y^2 = x^3 - 3x + b \text{ [60]} .$$

A : Affine, P : Projective standard, J : Jacobienne, I : Inversion modulaire, M : Multiplication, S : Carré

| **Representation** | **Addition de points** | **Doublement de points** |
|---|---|---|
| Affine | 2M+1S+8A+1I | 3M+2S+4A+1I |
| Projective Standard | 13M+1S+7A | 7M+5S+4A |
| Projective Jacobienne | 11M+4S+7A | 5M+5S+4A |
| Montgomery | 6M+1S+2A | 1M+4S+2A |
| Lopez -Dahab | 10M+4S+8A | 5M+5S+4A |

## 2.9 Conclusion

Dans ce chapitre, nous avons présenté les concepts mathématiques et détailler quelques aspects de ces dernières en vue de leur implantation liés aux courbes elliptiques. Etudier les courbes elliptiques définies par une équation de Weierstrass, d'analyser l'arithmétique issue de ces familles pour les corps finis GF(p) et GF($2^{n)}$, ainsi que l'ensemble de techniques qui nous permettent d'accélérer le calcul des multiplications scalaires, qui est considérée comme l'opération la plus importante et coûteuse sur les courbes elliptiques. Il a pu être constaté dans ce chapitre que les courbes elliptiques sont des objets géométriques et algébriques intéressants pour faire de la cryptographie.

Nous avons aussi présenté quelques propriétés importantes des courbes elliptiques, notamment l'équation algébrique à laquelle obéissent ces dernières, leurs formes géométriques, ainsi que le nombre de points qu'elles contiennent. Il est également décrit comment utiliser des courbes elliptiques pour faire de la cryptographie. L'arithmétique des courbes elliptiques est également très riche. Il a été notamment montré que le choix pertinent d'un système de coordonnées permet d'accélérer le calcul sur ces courbes.

# CHAPITRE 3

# ETUDE COMPARATIVE ENTRE UNE COURBE ALEATOIRE DE NIST ET LA COURBE DE KOBLITZ

*« La théorie, c'est quand on sait tout et que rien ne fonctionne. La pratique, c'est quand tout fonctionne et que personne ne sait pourquoi. Ici, nous avons réuni théorie et pratique : Rien ne fonctionne... et personne ne sait pourquoi ! »* ***Albert Einstein***

## 3.1 Introduction

La première façon de fournir des paramètres relatifs à une courbe est de les donner simplement sans explication, difficile alors d'établir une confiance dans ces derniers, à moins justement d'avoir une confiance aveugle dans l'organisme ou la personne qui les a générés. Les paramètres de la courbe de l'ANSSI sont ainsi fournis par le biais de leur site ainsi que du Journal Officiel [3] .

Actuellement deux organismes sont connus pour breveter la plupart des propriétés algorithmiques des courbes elliptiques, à savoir NIST [3] et Certicom [4]. Ils proposent tous deux l'utilisation de courbes basées sur Weierstrass qui utilisent des paramètres a, b et p. Le choix des réglages de ces paramètres rester dans de nombreuses études un secret complet. À savoir que ces deux organismes proposent tous deux d'utiliser des courbes basées sur Weierstrass utilisant les courbes Secp256r1 et secp256k1.

Les clés ECDSA utilisées pour générer des adresses Bitcoin[2] et signer des transactions sont dérivées de certains paramètres spécifiques. En raison de cette caractéristique, plusieurs questions se posent concernant le choix de cette courbe par Satoshi plutôt que celui de la courbe standard NIST secp256r1[3].

Donc dans ce chapitre nous essayons d'enlever l'énigme qui se pose sur le choix des paramètres des équations. Et pour cela nous analyserons la courbe aléatoire secp256r1 et la courbe Koblitz Secp256k1[3] (paramètres, équation, automorphisme…), en donnant les forces et les faiblesses de chacune d'elles, afin de justifier le choix du créateur de Bitcoin.

Les courbes secp256r1 et secp256k1 sont deux exemples de deux courbes elliptiques utilisées dans divers protocoles cryptographies tels que TLS, SSH ECDSA, ECDHE, ECDH et ECDLP.

La sécurité des courbes repose sur plusieurs critères mathématiques qui sont à l'heure actuelle majoritairement partagés par la communauté de la cryptographie. La tension principale autour de la sélection des courbes à normaliser se joue sur l'évaluation des avantages et inconvénients de chaque courbe (l'équation, choix des paramètres de la courbe, performance, résistance aux attaques par canaux auxiliaires, simplicité d'implémentation, efficacité, rigidité, portes dérobée et la sureté).

3.2 Courbes Secp256r1 / NIST P-256 sur les champs principaux

Les courbes elliptiques les plus utilisées sont celles proposées par le NIST sur $(p)$ introduites dans le standard FIPS [61]. Elles utilisent des nombres spéciaux.

Les paramètres de la courbe doivent être choisis avec soin pour éviter d'utiliser une courbe faible, et qui peut résister à toutes les attaques connues. Il peut aussi y avoir d'autres contraintes pour des raisons de sécurité ou de mise en œuvre.

Suivant SEC 2 [62], les paramètres du domaine de la courbe elliptique sur Fp sont un sextuple T = (p, a, b, G, n, h) comme le montre le tableau ci-dessous:

Tableau 3.1 : les paramètres du domaine de la courbe elliptique.

| | |
|---|---|
| p | L'ordre du champ premier Fp |
| Seed | La graine sélectionnée pour générer aléatoirement les coefficients de la courbe elliptique<br>La graine d'entrée SEED de 160 bits SEED à l'algorithme basé sur SHA-1 (c'est-à-dire, le paramètre de domaine de la graine) |
| r | La sortie SHA-1 |
| a,b | Les coefficients de la courbe elliptique $y^2 = x^3+ax +b$ qui satisfait : $r\, b^2 \equiv a^3 \pmod p$.<br>n The (prime) order of the base point P. |
| h | le cofactor |
| x, y | Les coordonnées de x et y du point P. |

3.2.1 Fondements mathématiques:

3.2.1.1 Le nombre premier p

Le $p$ de la courbe P-256 est un nombre premier de Mersenne généralisé, est conseillé pour travailler sur un corps dont la taille est de 256 bits.

Ce nombre premier a la propriété qu'il peut être écrit comme la somme ou la différence d'un petit nombre de puissances de 2 :

$$p256 = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1 \qquad (3.1)$$

Les puissances apparaissant dans cette expression est tous multiples de 32. Ces propriétés donnent des algorithmes de réduction qui sont particulièrement rapide sur les machines avec wordize32 [63]. Cette optimisation est particulièrement efficace sur CPU.

Posons $t= 2^{32}$ alors l'équation (2.1) devient : $P = t^8 - t^7 + t^6 + t^3 - 1$ (3.2)

Nous pouvons ensuite réduire les puissances supérieures de 2 en utilisant les congruences pour l'équation (2.1) donc la relation de congruence est :

$t^4 \equiv t^2 + t \pmod p, 2^{256} \equiv 2^{128} + 2^{64} \pmod p$ (3.3)

P : ce nombre premier a été choisi pour l'efficacité (la multiplication modulaire peut être réalisée plus efficacement qu'en général). L'algoritme 3.1 présente la réduction rapide $P_{256}$.

**Algorithm 3.1** [63] : Fast reduction modulo $p_{256} = 2^{256} - 2^{224} + 2^{192}$

INPUT: An integer $c = (c_{15},..., c_2, c_1, c_0)$ in base $2^{32}$ with $0 \leq c < p^2{}_{256}$.
OUTPUT: c mod $p_{256}$.

1. Define 256-bit integers:
   $s_1 = (c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0)$,
   $s_2 = (c_{15}, c_{14}, c_{13}, c_{12}, c_{11}, 0, 0, 0)$,
   $s3 = (0, c_{15}, c_{14}, c_{13}, c_{12}, 0, 0, 0)$,
   $s4 = (c_{15}, c_{14}, 0, 0, 0, c_{10}, c_9, c_8)$,
   $s5 = (c_8, c_{13}, c_{15}, c_{14}, c_{13}, c_{11}, c_{10}, c_9)$,
   $s6 = (c_{10}, c_8, 0, 0, 0, c_{13}, c_{12}, c_{11})$,
   $s7 = (c_{11}, c_9, 0, 0, c_{15}, c_{14}, c_{13}, c_{12})$,
   $s8 = (c_{12}, 0, c_{10}, c_9, c_8, c_{15}, c_{14}, c_{13})$,

La courbe elliptique est isomorphe à une courbe avec une équation de Weierstrass

3.2.1.2 Le Discriminant et J-invariant

Soit E : $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ une courbe définie sur k. On pose :

$b_2 = a_2{}^1 + 4a_2 \qquad b_4 = a_1a_3 + 2a_4 \qquad b_6 = a_2{}^3 + 4a_6$

$b_8 = a_1{}^2 a6 - a_1a_3a_4 + 4a_2a_6 + a_2a_2{}^3 - a_2{}^4$

$\Delta = -b_2{}^2 b_8 - 8b_4{}^3 - 27b_2{}^6 + 9b_2b_4b^6$

Le nombre $\Delta = \Delta(E)$ est appelé le discriminant de E. La courbe E est une courbe elliptique si et seulement si on a $\Delta \neq 0$.

Dans ce cas, on pose $j(E) = \frac{(b_b^2 - 24b_4\ )^3}{\Delta}$ (3.4) , on dit que j(E) est le j-invariant de E.

Une équation réduite avec a1 = a2 = a3 = 0, c'est-à-dire on peut toujours supposer que la courbe est donnée par : E : $y^2 = x^3 + a_4x + a_6 \mod p \ \ Si \ \ p \neq 2,3$ (3.5)

On dit alors que (3.5) est une équation de Weierstrass courte pour E. Dans ce cas,

on a :

$\Delta = -16(4a_4{}^3 + 27a_6{}^2)$[64] et on pose souvent $a_6 = b$ et $a_4 = a$ et $j(E) = (-48a_4)^3/\Delta$ [65].

- si $\Delta = 0$ alors l'équation (3.5) n'est pas une courbe elliptique : c'est une cubique singulière.

La cubique plane E est singulière (n'est plus une courbe elliptique), son graphe est l'une des formes suivantes:

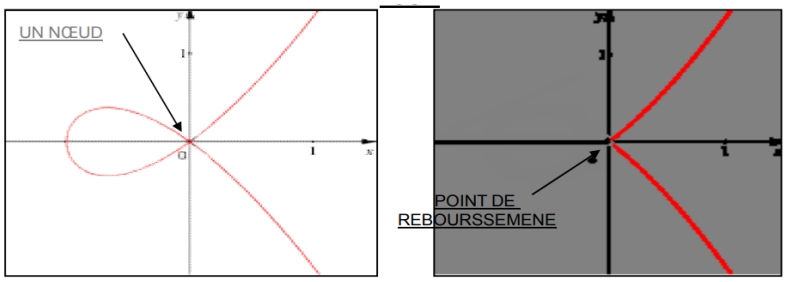
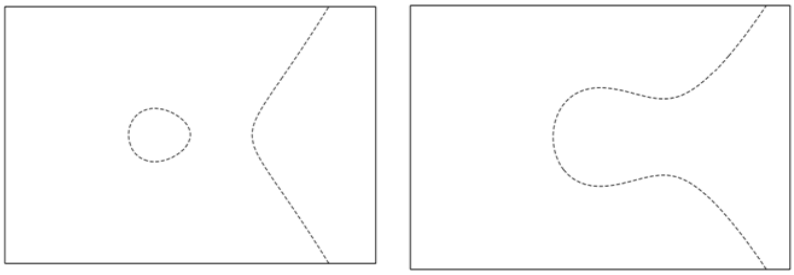



Figure 3.1 : une cubique singulière.

- Si $\Delta < 0$ alors le graphe de la courbe elliptique ne possède qu'une seule composante. Le polynôme cubique $x^3+ax+b$ possède une unique racine qui correspond à l'abscisse du point d'intersection de la courbe avec l'axe des abscisses.
- Si $\Delta > 0$ alors le graphe de la courbe elliptique possède alors deux composantes. Le polynôme cubique $x^3+ax+b$ possède 3 racines, qui correspondent aux abscisses des trois points d'intersection de la courbe avec l'axe des abscisses. J-invariant $\neq 0$ et K est un corps de caractéristique $\neq 2, 3$ alors l'ordre de l'automorphisme est égal à 2.

Si $\Delta(E)\neq 0$: la cubique plane E est non singulière (c'est une courbe elliptique ), son graphe de la forme:



(a) E2 : $y^2 = x^3 - x$ (b) E1 : $y^2 = x^3 - 3/2\ x^2 - 5/4\ x$

Figure 3.2 : Courbe elliptique.

- L'invariant j ne dépend pas de la forme de l'′equation de la courbe elliptique alors que $\Delta$ en dépend.

3.2.1.3 Complexité :

En général, le groupe des points d'une courbe elliptique se comporte comme un «groupe générique », le logarithme discret a une complexité exponentielle [66].Le groupe des points réguliers est alors isomorphe à un groupe additif ou multiplicatif, et le logarithme discret est sous-exponentiel, voire polynomial. Il est impératif que $\Delta \neq 0$ (ce qui arrive avec $P \approx 1$).

Plus précisément, la complexité du logarithme discret est dominée par $\sqrt{q}$ , où q est le plus grand diviseur premier du nombre de points de la courbe donc pour augmenter la complexité il faut avoir un nombre de points (presque) premier.

Il existe des attaques génériques de complexité $O(\sqrt{q})$, où q est le plus grand diviseur premier de N. Une courbe sûre doit donc avoir $q \approx N$ ; idéalement, $q = N$ .

La probabilité qu'une courbe aléatoire ait un ordre premier est approximativement la même que celle qu'un nombre aléatoire de la taille de p soit premier, soit $P \approx 1 / \log p$ [66].

Tableau 3.2: Complexité des attaques génériques.

| méthode | Attaque la plus rapide connue |
|---|---|
| RSA | Number Field Sieve $\exp(1/2(\log N)^{1/3}(\log\log N)^{2/3})$ |
| ECC | Pollard-rho $\sqrt{r} = \exp(1/2 \log r)$ |

3.2.1.4 Sélection du paramètre a = -3

La plupart des normes voir la norme IEEE 1363-2000 [67], choisissent a = -3 car pratiquement toutes les courbes ont des isogénies d'ordre faible et cela pour des raisons d'efficacité donc ce choix n'affecte pas la sécurité. Choisir des petites valeurs pour les paramètres a et b permet d'accélérer l'arithmétique de la courbe. De même, Brainpool [68] utilise cette équation pour ses avantages.

Ce choix permet d'économiser 2 des 10 multiplications requises pour une addition de points.

Une courbe aléatoire sur Fp est isomorphe avec une courbe $a = -3$ avec probabilité : $P = 1/4$ si $p \equiv +1 \pmod 4$ et $P = 1/2$ si $p \equiv -1 \pmod 4$.

Et enfin la sélection a = -3 pour le coefficient dans l'équation de la courbe elliptique a été faite de sorte que les points de la courbe elliptique représentés dans les coordonnées projectives jacobiennes pourrait être ajouté en utilisant une multiplication de champ de moins. L'algorithme 3.2 décrit le doublement de point.

**Algorithm 3.2 : Point doubling ($y^2 = x^3 - 3x + b$, Jacobian**

INPUT: P = (X1 : Y1 : Z1) in Jacobiancoordinates on E/K : $y^2 = x^3 - 3x + b$.
OUTPUT: 2P = (X3 : Y3 : Z3) in Jacobiancoordinates.
1. If $P = \infty$ then return($\infty$).
2. $T_1 \leftarrow Z_1^2$ .{$T_1 \leftarrow Z_1^2$ }
3. $T_2 \leftarrow X_1 - T_1$. {$T_2 \leftarrow X_1 - Z_1^2$ }
4. $T_1 \leftarrow X_1 + T_1$. {$T_1 \leftarrow X_1 + Z_1^2$ }
5. $T_2 \leftarrow T_2 \cdot T_1$. {$T_2 \leftarrow X_1^2 - Z_1^4$}
6. $T_2 \leftarrow 3T_2$. {$T_2 \leftarrow A = 3(X1 - Z_1^2)(X_1 + Z_1^2)$}
7. $Y_3 \leftarrow 2Y_1$. {$Y_3 \leftarrow B = 2Y1$}
8. $Z_3 \leftarrow Y_3 \cdot Z_1$. {$Z_3 \leftarrow B\ Z1$}
9. $Y_3 \leftarrow Y_3^2$ .{$Y_3 \leftarrow C = B^2$}
10. $T_3 \leftarrow Y_3 \cdot X_1$.{$T_3 \leftarrow D = C\ X_1$}
11. $Y_3 \leftarrow Y_3^2$ . {$Y_3 \leftarrow C^2$}
12. $Y_3 \leftarrow Y_3/2$. {$Y_3 \leftarrow C^2/2$}
13. $X_3 \leftarrow T_2^2$. {$X_3 \leftarrow A^2$}
14. $T_1 \leftarrow 2T_3$. {$T_1 \leftarrow 2D$}
15. $X_3 \leftarrow X_3 - T_1$. {$X_3 \leftarrow A^2 - 2D$}
16. $T_1 \leftarrow T_3 - X_3$. {$T_1 \leftarrow D - X_3$}
17. $T_1 \leftarrow T_1 \cdot T_2$. {$T_1 \leftarrow (D - X_3)A$}
18. Y3$\leftarrow$T1 $-$Y3. {Y3$\leftarrow$(D $-$ X3)A$-$C2/2}
19. Return(X3 : Y3 : Z3).

L'ordre des courbes elliptiques utilisées en cryptographie doit respecter certaines contraintes pour éviter des attaques connus. Par exemple, cet ordre doit être un nombre premier de grande taille ou le produit d'un grand nombre premier et d'un petit entier ou cofacteur, qui vaut 1 dans le cas d'une courbe d'ordre premier.

3.2.1.5 Le cofacteur

Le NIST prend le cofacteur aussi petit que possible pour de raisons d'efficacité».

le co-facteur h = l'ordre de la courbe elliptique / n ; avec n ordre du point qui est le plus petit entier tel que n.G =0 ( 0 : élément « identité » du groupe fini) et G doit être choisi de sorte que n est un grand nombre entier.

$$h = \frac{card\ (E(F))}{n} \qquad (3.6)$$

Donc Certains standards cryptographiques, comme FIPS-186-4[61], préconisent l'utilisation de courbes présentant un « petit » cofacteur h. En pratique, les contraintes peuvent différer d'un standard à un autre. Par exemple, la première version de SEC1 (2000) imposait un cofacteur $h \leq 4$ alors que celle de 2009 préconisent plutôt$h \leq 2^{\alpha}$ et α pour un niveau de sécurité plus élevée.

le choix de la valeur du cofacteur donc dépend de sa valeur car :

$$\begin{cases} si \quad h \leq 1 \quad pour \quad des \quad raisons \quad efficacité \\ si \quad h > 1 \quad améliore \quad les \quad performances \end{cases}$$

Citant à titres exemples les courbes de Montgomery utilisées par Apple qui ont un cofacteur $h > 4$ et cela pour améliorer les performances de la courbe.

Le tableau ci-dessous nous résume les Formes de courbes elliptiques sur Fp utilisables en fonction du co-facteur.

Tableau 3.3: Formes de courbes elliptiques sur Fp utilisables en fonction du cofacteur[68] .

| **co-facteur h** | **Forme** |
|---|---|
| 1 | Weierstrass |
| 2 | Quartique de Jacobi étendue |
| 3 | Hessiennegénéralisée |
| 4 | Quartique de Jacobi ou tordue |

3.2.1.6 Paramètre b

Pour le paramètre b de la courbe P-256, la formule suivante est utilisée afin de le générer : $b = \sqrt{\left(-\frac{27}{SHA1(s)}\right)}$ (3.7)

Avec : s = c49d360886e704936a6678e1139d26b7819f7e90 [69]

Cette procédure génère des données aléatoires en alimentant la graine dans SH1[69]. Des paramètres aléatoires vérifiables offrent des caractéristiques conservatrices supplémentaires [2].

Ces paramètres sont choisis parmi une graine utilisant SHA-1comme spécifié dans ANSI X9.62 [70]. Ce processus garantit que les paramètres ne peuvent pas être rédéterminés. Il est donc extrêmement improbable que les paramètres soient susceptibles de subir de futures attaques à des fins spéciales et aucun piège n'a pu être placé dans les paramètres au cours de leur génération.

### 3.2.2 Approche algébrique

**Loi de groupe** : pour E / K: $y^2 = x^3 + ax + b$, char (K)≠ 2,3

1**. Identité :** P + ∞ = ∞ + P = P pour tout P ∈ E (K).

2. **Négatifs :** Si P = (x, y) ∈ E (K), alors (x, y) + (x, -y) = ∞. Le point (x, -y)est noté -P et s'appelle le négatif de P; notez que -P est en effet un point dans E (K). Aussi, -∞ = ∞.

3. **addition :** Soit P = (x1, y1) ∈ E (K) et Q = (x2, y2) ∈ E (K), où P =± Q.

Alors P + Q = (x3, y3), où :

$$x3 = \left(\frac{y2-y1}{x2-x1}\right)^2 - 2x1 - x2 \quad \text{et} \quad y3 = \left(\frac{y2-y1}{x2-x1}\right)^2 - (x1 - x3) - y1$$

4. **Point double**

Soit P = (x1, y1) ∈ E (K), où P = -P. Alors 2P = (x3, y3), où :

$$x3 = \left(\frac{3x^2+a}{2y1}\right)^2 - 2x1 \quad \text{et} \quad y3 = \left(\frac{3x^2+a}{2y1}\right)^2 - (x1 - x2) - y1$$

Le cofacteur est toujours h = 1.

### 3.2.3 sélection des paramètres de la courbe

La sélection de la courbe est conditionnée par les paramètres T = (p, a, b, G, n, h):

Les entiers p et n sont donnés sous forme décimale ; les chaînes de bits et les éléments de champ sont donnés dans hexadécimal.

$y^2=x^3-3x+$

410583637251521421293261297800472684091144410159937255548352563140394674012 91.

Tableau 3.4:Courbes elliptiques aléatoires recommandées par le NIST sur les champs premiers[62].

| Paramètre | Valeur |
|---|---|
| p | 2 ^ 256 -2 ^ 224 + 2 ^ 192 + 2 ^ 96 -1<br>0xffffffff00000001000000000000000000000000ffffffffffffffffffffffff |
| b | 5ac635d8 aa3a93e7 b3ebbd55 769886bc 651d06b0 cc53b0f6 3bce3c3e 27d2604b |
| n | FFFFFFFF 00000000 FFFFFFFF FFFFFFFF BCE6FAAD A7179E84 F3B9CAC2FC632551 |

| | |
|---|---|
| **SEED** | c49d3608 86e70493 6a6678e1 139d26b7 819f7e90 |
| **c** | 7efba166 2985be94 03cb055c 75d4f7e0 ce8d84a9 c5114abc af317768 0104fa0d |
| **G x** | 6b17d1f2 e12c4247 f8bce6e5 63a440f2 77037d81 2deb33a0 f4a13945 d898c296 |
| **G y** | 4fe342e2 fe1a7f9b 8ee7eb4a 7c0f9e16 2bce3357 6b315ece cbb64068 37bf51f5 |

## 3.3 Courbes de Koblitz Secp256k1

Secp256k1 fait référence aux paramètres de la courbe ECDSA utilisée dans Bitcoin et est défini dans Standards for Efficient Cryptography (SEC)[62]. Secp256k1 n'a presque jamais été utilisé avant que Bitcoin ne devienne populaire, mais il gagne en popularité en raison de ses nombreuses propriétés. Celle-ci a été générée par Certicom (une entreprise canadienne) et non pas par le NIST comme la courbe Secp256r1.

### 3.3.1 Fondements mathématiques:

#### 3.3.1.1 Le nombre premier p

Les coefficients de Weierstrass définissant (a, b) de la courbe sont (0,7). Le document SEC2 indique à la section 2.1 que «les paramètres recommandés associés à une courbe de Koblitz ont été choisis en sélectionnant à plusieurs reprises des paramètres admettant un endomorphisme efficacement calculable jusqu'à ce qu'une courbe d'ordre premier ait été trouvée».

La taille du champ définissant p semble être un amorçage 256 bits de la forme spéciale $p = 2 ^ 256 - s$ [1] où s est petit avec la forme $s = 2 ^ 32 + t$, où $t <1024$

« $t = 2^9 + 2^8 + 2^7 + 2^6 + 2^4 + 1$ ».Donc P est le numéro d'amorce utilisé dans secp256k1, Bitcoin l'utilise comme limite haute pour les clés privées valides.

Si une clé privée est générée aléatoirement plus grande que n, elle est rejetée et une nouvelle clé est régénérée. La probabilité d'une telle occurrence est faible car P est "presque" aussi grand que $2^{256-1}$ (256 bits tous mis à 1).

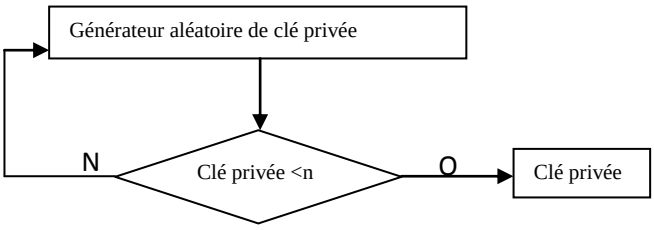



Figure 3.3 : Algorithme du générateur aléatoire de clé privée.

#### 3.3.1.2 Equation

Il a un ordre premier de 256 bits. Fait intéressant, ce choix s'écarte de ceux faits dans FIPS 186-4 en ce que les coefficients de la courbe sont a = 0 et b = 7.

La courbe elliptique est isomorphe à une courbe avec une équation de Weierstrass réduite de la forme ((*p*)): $y^2 = x^3 + b \mod p \quad Sip \neq 2,3$

Comme *une* constante est zéro, la *hache* terme de l'équation de la courbe est toujours nul, d' où l'équation de la courbe devient $y^2 = x^3 + 7$.

3.3.1.3 Le Discriminant et J-invariant

$\Delta = 4a^3 + 27b^2 \neq 0$ et $j(E) = (-48a)^3/\Delta = 0$ car a = 0

Cela signifie que secp256k1 a j-invariant 0 donc cette courbe est dite super-singulière et possède donc une structure très spéciale et des endomorphismes calculables qui peuvent être utilisés pour accélérer les implémentations, par exemple en utilisant la décomposition GLV pour la multiplication scalaire [71]. Cette idée a été introduite par Gallant, Lambert et Vanstone (GLV). Les courbes elliptiques ayant des endomorphismes calculables de manière efficace devraient être considérées comme des courbes elliptiques "spéciales". Utilisation d'instances "spéciales" de schémas cryptographiques est parfois faite pour des raisons d'efficacité [71].

3.3.1.4 Complexité

Cela pourrait conduire à une attaque plus sérieuse sur secp256k1 car un attaquant pourrait obtenir des multiples scalaires avec des scalaires secrets d'un point sur n'importe quelle courbe sur Fp avec coefficient a = 0, c'est-à-dire sur l'un des rebondissements de secp256k1.

3.3.2 Approche algébrique

3.3.2.1 Automorphisme

Les courbes elliptiques ayant des endomorphismes calculables de manière efficace sont considérées comme des courbes elliptiques "spéciales", avec un petit coefficient .Mais les endomorphismes efficaces accélèrent la multiplication scalaire, mais aussi l'algorithme rho de Pollard [83] pour le calcul des logarithmes discrets . Pour cette classe spéciale de courbes, l'accélération peut atteindre jusqu'à 50% par rapport aux meilleures méthodes générales de multiplication ponctuelle [28]. Si J-invariant = 0 et K est un corps de caractéristique ≠ 2, 3 alors l'ordre de l'automorphisme est égal à 6.

3.3.2.2 Multiplication scalaire rapide « La décomposition GLV

Il existe deux méthodes pour accélérer le calcul de la multiplication scalaire Q = kP sur des courbes elliptiques ayant un caractère non trivial endomorphismes calculables de manière efficace qui sont :

- la méthode Solinas [71] : sa méthode ne pouvait être appliquée que pour une courbe elliptique définie sur des champs binaires, l'endomorphisme considéré comme étant le Frobenius.

- La méthode Gallant-Lambert-Vanstone (GLV) **[71]**: sa méthode est appliquée pour les courbes elliptiques définies sur des champs primaires $F_p$, la décomposition est la base de l'accélération de calcul.

Une autre conséquence du groupe d'automorphisme plus large est l'existence de six **rebondissements** (y compris la courbe elle-même et la torsion quadratique standard). Le groupe d'automorphismes de E a l'ordre 6 et est généré par la carte $\psi$ .

La courbe secp256k1: $\equiv 1 \pmod 6$, il existe il existe une 6ème racine primitive de l'unité $\in$Fp , $\zeta \in \mathbf{F}p$, et un automorphisme de courbe correspondant tel que :

$\zeta^6 = 1$ [71]

$\psi: \rightarrow E, (x, y) \rightarrow (\zeta x, -y)$ ( 3.8)

Multiplication scalaire rapide $\psi P = \lambda P$ pour un entier $\lambda^6 \equiv 1 \pmod n$

Le principal avantage de ces courbes est que les algorithmes de multiplication scalaire de points n'utilisent pas de doublages de points.

3.3.2.3 Sélection des paramètres de la courbe de Koblitz spéciale

Les paramètres du domaine de la courbe elliptique sur $F_p$ associés à une courbe de Koblitz secp256k1 sont définis par le sextuple T = ( $p, a, b, G, n, h$ ) où le corps fini $F_p$ .

La courbe du Secp256k1 est sous la forme :

$E : y^2 = x^3 + 7 \bmod p$ sur $F_p$ (3.9)

Tableau 3.5: Paramètre de Secp256k1 [62].

| ***Paramètre*** | Valeur |
|---|---|
| ***p*** | $= 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$<br>FFFFFFFF FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF FFFFFFFE |
| **a** | 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |
| **b** | 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000007 |
| **G** | 04 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798<br>483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8 |
| **n** | = FFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141 |
| **h** | 1 |

Les paramètres a, b et p doivent être choisis tous les trois correctement afin de pouvoir résister aux attaques mathématiques.

3.4 Différents types de générateurs de nombres aléatoires

Il existe deux catégories de générateurs de nombres aléatoires : les générateurs de nombres dits déterministes (également appelés pseudo-aléatoires) et les générateurs de nombres dits réellement aléatoires. On parle dans le premier cas de DRNG et dans le second cas de TRNG . En pratique, les générateurs de nombres aléatoires implantés dans les circuits

intégrés sont des structures combinant un DRNG et un TRNG, on parle alors suivant les cas de DRNG ou de TRNG hybride.

- Un DRNG est un algorithme produisant des nombres aléatoires à partir d'une graine sélectionnée de façon aléatoire et éventuellement d'autres paramètres externes.
- Un TRNG est un composant générant des valeurs issues d'une source imprévisible.

### 3.4.1 Générateurs de nombres pseudo-aléatoires (DRNG)[72]

Les DRNG s'appuient sur des algorithmes déterministes pour générer des nombres qui, en apparence, semblent aléatoires. Leur nature algorithmique fait qu'ils sont facilement implantables dans les circuits numériques et peuvent fonctionner à des débits importants. Cependant, en théorie, les DRNG ne garantissent pas l'imprévisibilité des suites générées.

Les générateurs de nombres pseudo-aléatoires (DRNG) génèrent des séquences binaires qui ont les mêmes propriétés statistiques qu'une suite de séquences aléatoires, mais qui en théorie sont prévisibles. Leur sortie est évaluée à l'aide de tests statistiques sur les suites binaires, et leur niveau de sécurité grâce à la cryptanalyse de l'algorithme utilisé. Cette classe de générateurs est facile à implanter et permet des débits importants tout en produisant des suites qui ont de bonnes propriétés statistiques. Elle est donc très adaptée aux applications ne nécessitant pas l'imprévisibilité des suites (comme la simulation numérique), mais peut également être utilisée dans les applications cryptographiques à condition de respecter certains critères.

Comme illustré sur la figure 3.4, un DRNG se décompose en trois procédures liées à un état interne :

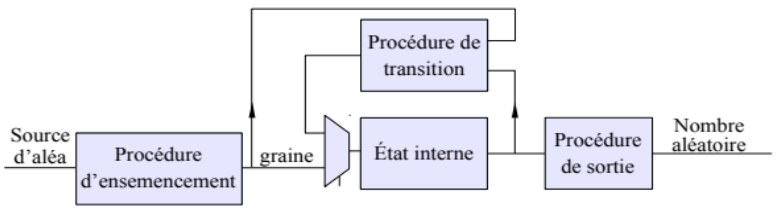



Figure 3.4 : Forme générale d'un générateur de nombres aléatoires déterministe selon **DRNG** dans les applications cryptographiques.

Dans la littérature, un DRNG adapté aux applications cryptographiques est appelé générateur de nombres pseudo-aléatoires cryptographiquement sûr (Cryptographically Secure DRNG).

Le critère principal de ce type de DRNG est le suivant :

Un adversaire qui n'a pas de connaissance de la graine d'initialisation ne doit pas pouvoir distinguer les suites issues du DRNG de suites véritablement aléatoires. Malgré le fait que cette propriété ne peut être rigoureusement prouvée, une forte évidence peut être apportée par la cryptanalyse, en montrant par exemple que prédire la sortie du DRNG revient à résoudre un problème considéré comme difficile ou insolvable, comme la décomposition en produit de facteurs premiers [73].

De plus, ces DRNG doivent être capables de résister à des attaques visant à introduire des imperfections dans l'algorithme (comme l'injection de données). Cette classe de DRNG inclut les algorithmes de chiffrement par flot (stream ciphers), certains algorithmes de chiffrement par bloc (block ciphers with output feedback) ainsi qu'un certains nombre de DRNG basés sur une hypothèse de complexité de calcul (Computational hardness assumption), comme l'algorithme de Micali-Shnorr et le générateur Blum Blum Shub [74]. Parmi ces algorithmes, on peut également citer Yarrow [75] et Fortuna [76].

### 3.4.2 Générateurs de nombres réellement aléatoires

Un TRNG est un composant générant des valeurs issues d'une source imprévisible. Suivant la source utilisée on peut discerner les TRNG dits physiques et les TRNG dits non physiques. Les TRNG non physiques sont les générateurs dont la source d'aléa n'est pas un phénomène physique dédié, mais un composant stockant d'autres informations du système.

## 3.5 Comparaison des courbes secp256r1 et secp256k1

Après avoir étudié les deux courbes, nous mentionnons les principales différences dans le tableau 3.2 . Le site Web SafeCurves**[77]** présente des évaluations de sécurité de diverses courbes spécifiques.

Tableau 3.6: la comparaison entre Secp256r1 et Secp256k1.

| **Courbe** | Secp256r1 | Secp256k1 |
|---|---|---|
| **sécurité [77]** | $= 2\sqrt{\frac{\pi nsec\ 256r1}{4}} = 127.83$ | $2\sqrt{\frac{\pi nsec256r1}{2}} = 127.03$ |
| **Ordre de l'automorphisme** | 2 | 6 |
| **Paramètres « a »** | **-3** sont des revendications d'efficacité, pas des revendications de sécurité. | **a = 0** la *hache* terme de l'équation de la courbe est toujours nulle |
| **Coût pour une attaque combinée[77]** | 2 ^ 120,3 | 2 ^ 109,5 |

Les courbes de Koblitz sont généralement connues pour être quelques bits plus faibles que les courbes de champ de premier ordre, mais quand on parle de Courbes de 256 bits, il a peu d'impact.Bitcoin fonctionne avec une courbe fixe et ne génère que des clés privées et publiques, selon Safecurves [77] la courbe elliptique secp256k1 peut être considérée comme quelque peu «rigide» ce qui signifie que presque tous les paramètres sont transparents pour le public et peuvent donc être supposés ne pas être généré pour être faible.

La méthode rho brise l'ECDLP en utilisant en moyenne des additions d'environ 0,886 $\sqrt{l}$ donc la sécurité est comparable pour les deux courbes.

Le Coût pour une attaque combinée est presque le même pour les deux courbes. Certes la courbe Secp256k1 a une sécurité comparable que celle que la courbe mais elle a des rebondissements supplémentaires [71] , qui mènent à plus de possibilités pour une attaque. En revanche, une courbe elliptique avec j-invariant différent de 0 et 1728 comme le cas de la courbe secp256r1 a seulement un automorphisme groupe d'ordre 2, de sorte que l'accélération de l'algorithme de Pollard rho[78] est un facteur constant jusqu'à √3 sur une telle courbe.

## 3.6 Conclusion

Secp256k1 est souvent plus de 30% plus rapide que les autres courbes si l'implémentation est suffisamment optimisée et le critère de rapidité est un critère très important pour les bicoins car un paiement avec Bitcoin est presque instantané. Cependant, secp256r1 utilise la graine très suspecte « c49d360886e704936a6678e1 139d26b7819f7e90 » qui est étrangement similaire à la porte dérobée dans Dual_EC_DRBG [77]. La courbe elliptique bitcoin a le plus bas | D | de tous les courbes elliptiques standardisés connus, et donc il est potentiellement le moins sécurisé.

De façon générale, il faut être très méfiant sur les courbes elliptiques dès lors que les paramètres ne sont pas justifiés de manière claire et détaillée. Il y a déjà le cas de valeurs parachutées qui se sont avérées plus que douteuses par la suite.

Il ne faut pas non plus sous-estimer les attaques non-publiques ou qui pourraient voir le jour prochainement.

Même si les critères précédents sont respectés, il est toujours possible d'attaquer l'implémentation même avec une possibilité de succès.

Si le concepteur de circuits sécurisés n'est pas dans l'obligation d'utiliser des courbes Standardisées, il peut choisir d'autres courbes à l'arithmétique efficace, comme par exemple celles dites de Montgomery ou d'Edwards.

# CHAPITRE 4
# CONCEPTION ET MISE EN ŒUVRE D'UNE BOITE A OUTILS DE CRYPTOGRAPHIE A COURBE ELLIPTIQUE

*« La science ne renverse pas à mesure ses édifices ; mais elle y ajoute sans cesse de nouveaux étages et, à mesure qu'elle s'élève davantage, elle aperçoit des horizons plus élargis ».*
*Berthelot*

## 4.1 Introduction

Les précédents chapitres ont concernés l'apport mathématique, les operateurs arithmétiques modulaires, le choix d'équation en faisant une étude comparative détaillée sur l'équation de NIST normalisée et équation de Koblitz non normalisée en prenant par exemple les courbes sec256r1 et sec256k1, ainsi que les différents algorithmes de base qui vont nous aidé à la conception et la stratégie de notre travail.

L'implémentation de fonctions cryptographiques est délicate, et la moindre erreur peut empêcher l'atteinte de l'objectif de sécurité initial (intégrité, authentification, confidentialité, etc.). Il y a essentiellement deux types d'erreurs : les erreurs d'algorithmie et les erreurs d'implémentation. Les premières consistent à mal choisir les primitives cryptographiques (algorithmes « maison », algorithmes cassés, les équations dans cas des courbes elliptiques, mauvais choix de taille de clés, etc.) ou à mal les ordonnancer.

Lorsqu'on désire étudier ECC , nous nous trouvons directement référés à CERTICOM [4] et cela pour leur tutoriel d'introduction mais ce dernier explique seulement les opérations de base tels que les additions et les multiplications dans une extension de groupe fini comme le montre la figure 4.1.

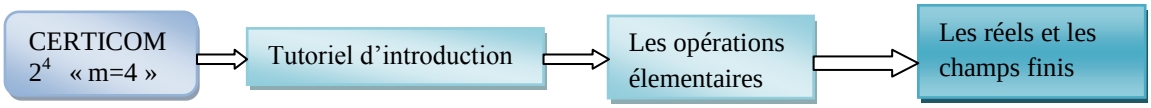



Figure 4.1 : tutoriel d'introduction du Certicom.

Nous présentons dans ce chapitre, une conception d'une boite à outil de la cryptographie à courbe elliptique avec le Protocole « Diffie-Hellman » (ECDH), et cela en développant une nouvelle boîte à outils sur Matlab en lui ajoutant des fonctions inexistantes.

L'état de l'art au niveau des implantations software traite particulièrement de la multiplication modulaire [77,79]. Car c'est l'opération la plus fréquemment utilisée et la plus consommatrice en termes d'espace et de temps.

La bibliothèque ECC-GF a pour but donc de développer un tutoriel sur la cryptographie à courbe elliptique sur GF ($2^m$) pour les différentes courbes citées.

Toutes les fonctions ont été testées en détail sur des nombres réels convertis en un format binaire matriciel comme le montre les figures ci-dessous.

Nous allons au dessous nous intéresser aux algorithmes et à leurs implémentations.

## 4.2 Réduction polynomiale recommandée par NIST

Afin de rendre la résolution du problème du logarithme discret sous-jacent hors de portée, les choix de paramètres sont importants. Afin de normaliser l'utilisation de courbes pratiques et non problématiques, l'Institut national de normalisation et Technology (NIST) a adopté un ensemble de courbes sélectionnées sécurisées, comme indiqué dans le tableau 4.1. Les autres paramètres importants des opérateurs sont la taille de champ m (tailles des opérandes) et le générateur de champ f (x). Les opérateurs arithmétiques conçus devraient servir les applications ECC, le NIST dans [80], nous préconise un polynôme pour chaque degré. Il s'agit de trinômes ou de pentanômes. Les valeurs les plus récentes sont présentées dans le tableau 4.1.

Tableau 4.1 réduction polynomiale recommandée par NIST.

| Taille de m | Reduction polynomiale | Codification de NIST |
|---|---|---|
| 113 | $F(z) = Z^{113}+Z^9+1$ | Sect113r1/sect113r2 |
| 131 | $F(z) = Z^{131}+Z^8+Z^3+Z^2+1$ | Sect131r1/sect131r2 |
| 163 | $F(z) = Z^{163}+Z^7+ Z^6+Z^3+ 1$ | Sect163r1/sect163r2 |
| 193 | $F(z) = Z^{193}+Z^{15}+1$ | Sect193r1/sect193r2 |
| 233 | $F(z) = Z^{233}+Z^{74}+1$ | Sect233k1/sect233k2 |
| 239 | $F(z) = Z^{239}+Z^{158}+1$ | Sect239k1/sect239k2 |
| 283 | $F(z) = Z^{283}+Z^{12}+ Z^7+Z^5 +1$ | Sect283k1/sect283k2 |
| 409 | $F(z) = Z^{409}+Z^{87}+1$ | Sect409k1/sect409k2 |
| 571 | $F(z) = Z^{571}+Z^{10}+ Z^5+Z^2 +1$ | Sect571k1/sect571k2 |

## 4.3 Protocole de l'accord ECC

Dans notre conception nous avons utilisé l'algorithme d'échange de clé du protocole d'accord Diffie-Hellman à courbe elliptique (ECDH), qui est un protocole entre deux utilisateurs finaux utilisant un système commun clé publique cryptographique.

Le protocole ECDH entre Alice et Bob est présenté sur la figure 4.2 , il montre comment les clés privées et publiques sont généré et comment les deux utilisateurs finissent avec la même clé, tout en partant de différents points de la courbe elliptique, l' accord doit porter uniquement sur les paramètres de la courbe et la taille de la clé de chiffrement . L'algorithme ci- dessous nous décrit un exemple du protocole ECDH avec toutes ces étapes.

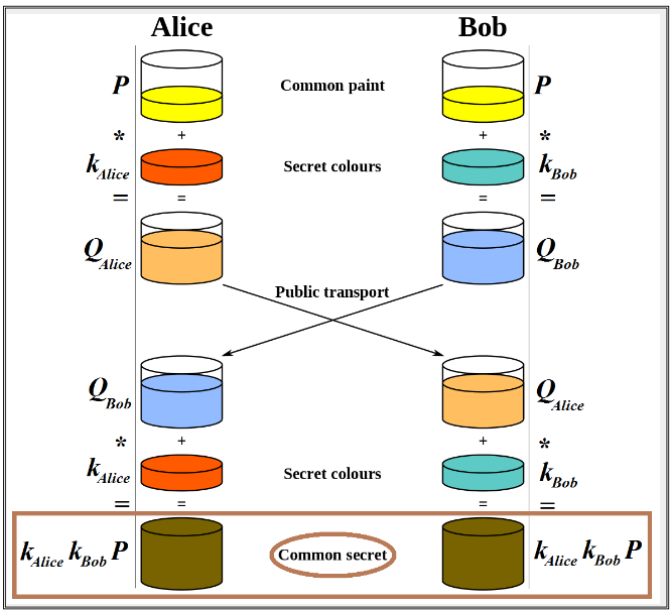



Figure 4.2 : Le protocole ECDH entre Alice et Bob.

## 4.4 Principe de base de l'implémentation des différentes opérations modulaires

L'opération prédominante dans l'implantation de la cryptographie à base de courbes elliptiques est la multiplication scalaire, comme mentionné dans le deuxième chapitre, l'implantation d'algorithmes efficaces, stables, ayant un minimum d'étapes et de variables, serait un réel must.

L'implantation software des algorithmes tendent à manipuler des variables de plusieurs dizaines de bits, voire plusieurs centaines dans notre cas 113 bits. A cela l'implantation d'opérations parfois obsolètes afin de lisser la puissance consommée, en vue de palier à la cryptanalyse [81, 82], cette problématique provient du fait que les avancées en cryptanalyse, forçant ainsi les chemins de données à s'élargir, en incombant une très forte stratégie de management des locations mémoires, des parties de contrôle, de la synchronisation fréquentielle, ainsi que de l'encombrement matériel.

Les opérateurs représentent les briques des ECC, donc l'accélération à ce niveau aura un impact incommensurable sur l'accélération du processus de la multiplication scalaire.

L'état de l'art au niveau des implantations software traite particulièrement de la multiplication modulaire [83,84]. Car c'est l'opération la plus fréquemment utilisée et la plus consommatrice en termes d'espace et de temps.

La multiplication scalaire, dans les courbes elliptiques, est l'opération permettant de générer la clé publique $Q = k \times P$ , $P$ étant un point de la courbe et $k$ la clé privé. Dans ce qui suit, nous allons détailler des différentes étapes de l'implantation matérielle de cette multiplication scalaire, en implantant tous les composants de la figure 4.3.

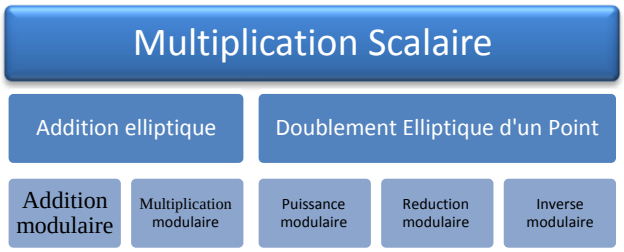



Figure 4.3: Composants de la multiplication scalaire [7].

La courbe elliptique et le champ de Galois GF ($2^m$) forment un anneau, ainsi l'addition et la multiplication de champs conduisent à des éléments au sein de la même courbe (champ).

La multiplication scalaire n'est pas une vraie multiplication, comme on peut l'imaginer pour des nombres décimaux ou même des opérandes binaires, où la relation bit à bit utilise "et" et "ou" opérateurs de base, avec ou sans propagation de retenue.

La multiplication scalaire peut être reprise sous la forme addition incrémentielle de deux points sur la courbe elliptique;

- si les points sont distincts, on l'appelle addition de points ;
- lorsque le point s'ajoute à lui-même, il s'appelle doubler de point.

Comme nous avons vu dans le chapitre 2, il existe plusieurs coordonnées de projections pour calculer un doublet ou une addition , notre choix s'y porté sur l'algorithme de Montgomery [7] car il empêche l'opération d'inversion qui est la plus couteuse et complexe à chaque étape en utilisant des coordonnées projectives, comme présenté dans les équations (2.93,2.94 et 2.95) .

## 4.5 Implantation des différentes opérations modulaires des composants de base de la multiplication scalaire

Une fois définis les différents paramètres et opérations modulaires d'implantation, nous pouvons maintenant détailler l'arithmétique modulaire choisie, et plus précisément les

différents algorithmes que nous implantons pour effectuer l'ensemble des instructions exécutables pour extension de la bibliothèque Matlab .

Les phases d'implantation reposent sur l'implantation des operateurs de base dans les champs de Galois, les différentes opérations définies sur $GF(2^m)$ sont :

- l'addition modulaire,
- la multiplication modulaire et scalaire ;
- le carré/ puissance d'un nombre ;
- la réduction polynomiale/ modulaire ;
- inverse modulaire.

### 4.5.1 L'addition modulaire « L'addition GF XOR »

L'addition dans les champs modulaires $GF(2^m)$ est une opération matérielle très simple à réaliser, car elle est basée sur une complexité linéaire $O(n)$ où $n$ est la taille des operateurs.

Étant donné deux opérandes appartenant à $\mathrm{GF}(2^m)$, l'addition modulaire génère un opérande de même taille $m$, tel que :

$$C(z) = A(z)\ xor\ B(z) \qquad (4.1)$$

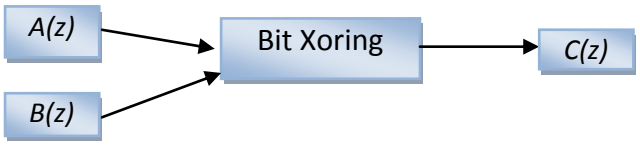



Figure 4.4 : Addition modulaire dans $\mathrm{GF}(2^m)$.

L'estimation matérielle de l'addition modulaire est illustrée dans le tableau 4.2.

Tableau 4.2 : Estimation du nombre de portes XOR de l'addition modulaire.

| Nombre d'entrées | Nombre de sorties | Coût en portes logiques XOR à deux entrées |
|---|---|---|
| 225 | 113 | 113 |

Dans les champs de Galois, tels que GF ($2^m$), addition et soustraction sont similaires, car il n'y a pas de propagation de report, et l'opérateur logique est un "XOR".

Donc addition et la soustraction conduisent à des résultats identiques. En conséquence, l'addition est mise en œuvre, comme indiqué sur la figure 4.5, il utilise la base Fonction "bitxor" de Matlab appliquée à une vectorielle binaire représentation.

**Ajout de la fonction « Cout » dans Matlab**

```
function Cout=G_Addition(Ain,Bin)
```

```
Cout=bitxor(Ain,Bin);
return
```

Figure. 4.5 : la fonction Cout du GF ($2^m$) dans Matlab.

### 4.5.2 La réduction polynomiale / modulaire

Cette opération est nécessaire à l'issue de multiplications et d'élévations au carré, pour ramener le résultat obtenu de degré au plus « 2m – 2 » à un polynôme de degré « m – 1 ».

La réduction modulaire ou polynomiale consiste à annuler donc les bits d'ordre supérieur à "$m-1$", en réalisant un simple « XOR » polynomial, à chaque bit non nul des registres résultats, issus de la multiplication modulaire ou de toute autre opération générant des variables de taille supérieur à celle du champ , comme illustré dans la figure 4.6.

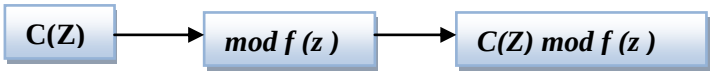



Figure 4.6 : Réduction modulaire dans $GF(2^m)$.

Effectuer la réduction modulaire d'un polynôme a(x) de degré n ≥ m, c'est chercher le reste de la division de a(x) par f(x).

Pour un même groupe $GF(2^m)$, il y'a différents polynômes de réduction polynomiale ou plutôt plusieurs polynômes générateurs [8] comme listé sur le tableau 4.1 .

Dans notre conception et implantation, nous nous sommes alignés à la norme NIST B-113 [29], par l'utilisation du polynôme générateur suivant :

$$f(z) = z^{113} + z^9 + 1 \qquad (4.2)$$

Différents puissances polynomiales de $z$ sont déduites de l'équation (4.2), comme illustré dans la figure 4.7 dans le cas où m =113.

| | |
|---|---|
| $Z^{113}=Z^9+1$ | $Z^{114}=Z^{10}+Z$ |
| $Z^{115}=Z^{11}+Z^2$ | $Z^{116}=Z^{12}+Z^3$ |
| …….. | $Z^{270}=Z^{10}+Z$ |

Figure 4.7 : Méthodologie de la réduction polynomiale en utilisant le polynôme de réduction$f(z) = z^{113} + z^9 + 1$.

- **Ajout de la fonction « GF Modulo » implémentée sur Matlab**

Éléments du GF ($2^m$) ont le même nombre de bits, chaque fois que le résultat d'une opération dépasse « m » bits, une réduction polynomiale est nécessaire, elle est effectuée par additions successives avec le polynôme générateur de degré « m + 1 », l'opération se termine lorsque le résultat final contient « m » bits.

Notre fonction « modulo » utilise une matrice générée à partir de « XOR » répétée des positions de bits, conduisant à une matrice de « m » lignes par « M » colonnes contenant ceux à des positions nécessitant une « XOR », comme présenté dans la fonction de la Figure 4.8.

```
GenerateModulo (113) %% m=113
        function GenerateModulo1(m)
    1.  P= bringP(m); P=P(end:-1:1);
    2.  A=eye(m);     A(m+1,1:m+1)=P(1,1:m+1);
    3.  for i=m+2:(m-1)*2+1;
    4.  A(i,2:m+1)=A(i-1,1:m);
    5.     if A(i,m+1)==1;
    6.  E=xor(A(i,1:m+1),P);
    7.  A(i,1:m)=E(1,1:m);
    8.       end;
    9.  end;
    10. Modulo_Matrix=A(:,1:m);
    11. filename=['Galois_Modulo',num2str(m),'.m']
    12. ConvertToModulo(Modulo_Matrix,m,filename);
    13. return
```

Figure 4.8 : Générer une fonction modulo dans GF ($2^m$).

A cette étape, une matrice «Modulo_Matrix» de taille « 2.m-1 » par « m» est générée, puis envoyée à la fonction « ConvertToModulo.m » afin de générer une fonction, comme illustré à la Figure. 4.9.

```
function ConvertToModulo(Modulo_Matrix,m,filename)
 for i=1:m
 Col1=A(1: size(Modulo_Matrix,1),i);
indices = find(Col1); sizeXors=size(indices,1);
 if sizeXors>=1;
text1=['A(',num2str(indices(end)-MyShift),')'] ;
  for j=sizeXors-1:-1:1;
text1=['bitxor(A(',num2str(indices(j)-MyShift),'),',
text1,')'];
   end;
      T(i)={['B(',num2str(i-MyShift),')=',text1,';']};
text1='';
  end;
 end;
```

Figure 4.9 : Conversion d'une matrice en une fonction.

La fonction générée, dans notre cas où m = 113, est représenté sur la figure 4.10. notons que la boite à outils génère aussi le code VHDL ECC , pour l'essai du ECDH sur une plate forme FPGA.

```
function B =Galois_Modulo(A,113)
  B(1)=bitxor(A(1),bitxor(A(114),A(218)));
  B(2)=bitxor(A(2),bitxor(A(115),A(219)));
   ……...
  B(6)=bitxor(A(6),bitxor(A(119),A(223)));
  B(7)=bitxor(A(7),bitxor(A(120),A(224)));
  B(8)=bitxor(A(8),bitxor(A(121),A(225)));
  B(9)=bitxor(A(9),A(122));

B(10)=bitxor(A(10),bitxor(A(114),bitxor(A(123),A(218))));
 ……
B(17)=bitxor(A(17),bitxor(A(121),bitxor(A(130),A(225))));
   ……...
  B(111)=bitxor(A(111),bitxor(A(215),A(224)));
  B(112)=bitxor(A(112),bitxor(A(216),A(225)));
  B(113)=bitxor(A(113),A(217));
```

Figure 4.10 : Script modulo généré automatiquement pour GF ($2^{113}$).

4.5.3 Le carré / puissance modulaire

Étant donné un opérande A(z) appartenant à $GF(2^m)$, le carré de A(z) est défini comme suit : $C(z) = A(z)^2 \bmod f(z)$ (4.3)

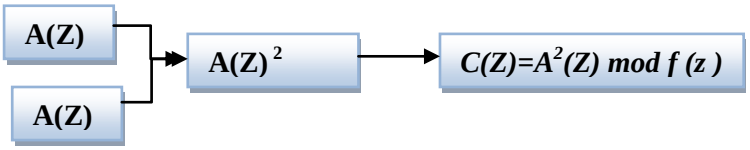



Figure 4.11 : Le carré / puissance modulaire dans $GF(2^m)$.

La quadrature d'un $GF(2^m)$, élément peut être fait par une multiplication d'un opérande par lui-même, ou en insérant des zéros à les positions paires de l'opérande initial, puis en exécutant une Réduction « modulo Galois »; l'opération ultérieure nécessite moins Ressources.

Nous avons mis en œuvre deux méthodes pour l'élévation du carré dans un champ Galois ( $A^{2^n}$), avec un élément de $GF(2^m)$ et « n » un entier inférieur à "m".

- **La première méthode :**

En utilisant le carré successif, qui utilise "n" fois l'opérante, cette fonction fonctionne bien pour la partie logicielle, mais le temps peut être un problème cruciale si «n» augmente. La fonction utilisant la quadrature imbriquée successive est montrée à la figure. 4.12.

```
function
B=Galois_2PowerX(A1,Xtimes,m)
 a0=A1;
for i=1:Xtimes
b0=Galois_2Power1(a0,m);
```

```
a0=b0;
end;
B=a0;
```

Figure 4.12 : Fonction de quadrature en GF($2^m$).

- **Deuxième méthode :**

Par insertion de zéro rangée, aux positions paires d'opérande, formant une matrice globale, utilise fondamentalement la même procédure que le script modulo, mais le choix des lignes restantes sélectionnées dépend des multiples du pouvoir carré.

La contrainte en informatique $\boldsymbol{A^{2^n}}$ , est relativement grand de zéros insérés nombre de zéros à GF($2^m$), donc on aura «$\boldsymbol{A^{2^{n-1}}}$ » Lignes de zéro qui ont les mêmes positions de l'opérande au carré, et cela augmenterait la matrice polynomiale à: (m-1) *$2^n$ lignes, par exemple pour n = 30 et m = 113, une matrice de : 117 440 513 lignes par 114 colonnes) seraient nécessaires.

**Remarque :**

Aussi que cette méthode a un deuxième inconvénient lié à la taille de n, dans le sens qu'il augmente la relation entre les différentes lignes de la matrice (rangées non clairsemées), cela aura tendance à imbriquer plus Fonctions «bitand» et «bitxor», bien que Matlab ne permette pas plus de trente niveaux de d'imbrication.

Pour notre échange de protocole, nous avons surtout utilisé le premier approche d'imbrication, en prenant "n" petit par rapport à m (n $\leq$ 10), comme il est illustré sur la Figure 4.13.

L'implantation matérielle d'un circuit permettant cette opération peut se réaliser via une multiplication modulaire et une réduction polynomiale, ou par la méthode suivante :

L'équation 4.3, en forme développée s'écrit de la manière suivante :

$$A(z)^2 = (a_{m-1}z^{m-1} + \cdots + a_2z^2 + a_1z^1 + ca_0).(a_{m-1}z^{m-1} + \cdots + a_2z^2 + a_1z^1 + a_0) \quad (4.4)$$

De l'équation (4.4), on obtient deux types de termes :

- $\big((a_{i-1}).(a_{j-1})\big)\ xor\ \big((a_{j-1}).(a_{i-1})\big) = 0$ (4.5)
- Ainsi que $((a_{i-1}).(a_{i-1})) = (a_{i-1})^2$ (4.6)

Après simplification binaire des puissances restantes, on obtient :

$$A(z)^2 = \sum_{i=0}^{2.m-2}(a_{i-1})^{2i}\,.\,z^{2i} = \sum_{i=0}^{2.m-2}(a_{i-1})^{i}\,.\,z^{2i} \quad (4.7)$$

Cet opérande contient « $2.m-1$ » éléments, avec « $m-1$ » coefficients nuls, donc la réduction polynomiale contient moins de portes XOR lors d'une puissance modulaire, que la réduction polynomiale après une multiplication modulaire

```
function GenerateSquareFiles(113)
GenerateSquareFiles(113)
ToThePower=10; % A^(2^(ToThepower))
Exponent=2;
Squaring=Exponent^(ToThePower);
P= bringP(m);
P=P(end:-1:1);
A=eye(m); A(m+1,1:m+1)=P(1,1:m+1);
for i=m+2:(m-1)*Squaring+1;
A(i,2:m+1)=A(i-1,1:m);
if A(i,m+1)==1; E=xor(A(i,1:m+1),P);
 A(i,1:m)=E(1,1:m); end; end;
SquaringMatrix=A(1:Squaring:end,1:m);
ConvertToModulo(SquaringMatrix,m,filename)
```

Figure 4.13 : Générateur de fonctions « GF Squaring » dans le toolbox .

La fonction « Galois_2PowerX.m » de la Figure 4.13 utilise la fonction de script générée automatiquement de la figure 4.14, pour écrire un nouveau script comme le montre la Figure 4.14.

```
function B=Galois_2Power1(A,m)

B(1)=A(1);
      B(2)=bitxor(A(58),A(110)); B(3)=A(2); B(4)=bitxor(A(59),A(111));
B(5)=A(3);       B(6)=bitxor(A(60),A(112)); B(7)=A(4);
B(8)=bitxor(A(61),A(113)); B(9)=A(5); B(10)=A(62);
B(11)=bitxor(A(6),bitxor(A(58),A(110))); B(12)=A(63);
……..
B(107)=bitxor(A(54),A(106));                    B(108)=A(111);
B(109)=bitxor(A(55),A(107)); B(110)=A(112);
B(111)=bitxor(A(56),A(108));                    B(112)=A(113);
B(113)=bitxor(A(57),A(109));
```

Figure 4.14 : Vue partielle des fonctions de mise au carré GF générée automatiquement.

4.5.4 L'inverse modulaire par la méthode d'Itoh-Tsjuii

Étant donne un élément A, son inverse est représenté par :

$$A(z)^{-1} = A(z)^{2.m-2}\ mod\ f(z) \qquad (4.8)$$

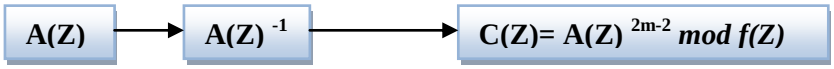



Figure 4.15 : l'opération d'inversion modulaire.

L'inverse modulaire [7], reste parmi les opérations arithmétiques les plus complexes, en termes de temps et de ressources, en effet, pour réaliser une seule opération d'inverse, une série de carrés modulaires ainsi que de multiplications incluant des réductions polynomiales sont nécessaires.

Pour cette raison la conversion des coordonnées affines en coordonnées projectives, reste une solution permettant la temporisation de l'inverse à l'étape finale.

L'opération inverse du GF est limitée à m = 16 dans le Matlab, notre boîte à outils étend le GF inverse à toutes les courbes préférées du NIST, et à d'autres nombres premiers raisonnables (limités par la possibilité de calcul et taille de la machine à mots).

L'inverse en GF est défini comme :

$$\frac{1}{A} = A^{2^m-1} \bmod P \qquad (4.9)$$

où A est un élément de GF ($2^m$), P le polynôme de réduction, m le champ.

Pour traiter un inverse, il est nécessaire de diviser le $A^{2^m-1}$ à un ensemble de puissance imbriqués; comme mentionné par la méthode de l'algorithme Itoh-Tsujiii [85], où l'exposant « $2^m$ -1 » 'Est scindée en une chaîne d'addition [86]; cette chaîne peut être modifié dans notre boîte à outils, en sélectionnant un sous-index différent étape d'addition, qui consiste à l'origine en une scission par deux des exposant précédent.

Afin d'implanter cet operateur, nous avons utilisé l'algorithme d'Itoh-Tsjuii [85], qui consiste à réécrire $A(z)^{2.m-2}$ sous formes de produits et carrés imbriqués, par la règle appelée «Multiplier et élever au carré», selon les indices de multiplication et d'addition générés, comme illustré dans le tableau 4. 3.

Tableau 4.3: Méthodologie de génération des indices de la chaine d'addition pour m= 113.

| Ordre | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Puissance | 1 | 2 | 3 | 6 | 7 | 14 | 28 | 56 | 112 | 113 |

Pour notre implémentation nous avons utilisé la fonction « Main_Itoh.m » tel que décrite dans la figure 4.16 , est divisé en deux parties , une génération de la chaine d'addition puis un inverse en utilisant de la chaine générée précédente.

```
function Main_Itoh
  m=113
TypeOfSequence='Modified'; % 'Original'
 Field=m-1; % In order to have a nonprime number
 SubIndex=2^4 % 16 in decimal
[Index_Left, Powers_Right]=
```

```
GenerateItohSequence(TypeOfSequence,Field,SubIndex);
MakeMatlabInverseOnSequence(m,Index_Left,Powers_Right)
```

Figure 4.16 : Fonction de génération de séquence inverse pour le GF($2^m$) bibliothèque.

La chaîne d'addition de la Figure 4.16 est ensuite utilisée comme entrée dans une Fonction "Make Matlab Inverse On Sequence.m" de Matlab, afin de générer l'inverse comme une fonction distincte comme indiqué dans Figure 4.17, avec un opérande d'entrée de test comme échantillon pour la validité de la règle de la chaîne.

Ce qui doit être notifié est que, dans la mesure où une réduction polynômiale existe sur un GF($2^m$), son inverse peut être calculé dans un délai raisonnable, avec une grande précision, par le méthode proposée.

```
m= 113
      B1=B0;
IndexLeft = [1  2  4  5 10 20  30  40  50  60  70   80  90  100  110  111  112  113];
IndexRight  = [ 1  2 1  5 10  10   10   10   10   10   10   10   10   10   1  1   1   0];
      B_1=Galois_2PowerX(B1,1,m);          B2=Galois_Mult(B1,B_1,m);
      B_2=Galois_2PowerX(B2,1,m);          B3=Galois_Mult(B1,B_2,m);
      B_3=Galois_2PowerX(B3,3,m);          B6=Galois_Mult(B3,B_3,m);
      B_6=Galois_2PowerX(B6,1,m);          B7=Galois_Mult(B1,B_6,m);
      B_7=Galois_2PowerX(B7,7,m);          B14=Galois_Mult(B7,B_7,m);
      B_14=Galois_2PowerX(B14,14,m);       B28=Galois_Mult(B14,B_14,m);
      B_28=Galois_2PowerX(B28,28,m);       B56=Galois_Mult(B28,B_28,m);
      B_56=Galois_2PowerX(B56,56,m);   B112=Galois_Mult(B56,B_56,m);
      B_112=Galois_2PowerX(B112,1,m);
      Apowerminus1=B_112;
Final1=Galois_Mult(B1,Apowerminus1,m);
```

Figure 4.17: calcul d'Inverse dans GF($2^{113}$).

### 4.5.5 Multiplication scalaire par la méthode de Montgomery

L'algorithme de Montgomery concernant la multiplication scalaire s'avère parmi la méthode la plus efficace [6,7,30,39,53,87], donc nous allons développés la stratégie d'implantation autour de cet algorithme. Ce dernier est basé sur l'implantation de deux routines élémentaires, l'addition scalaire et le doublement elliptique toutes deux sont effectuées sur la courbe elliptique $E(GF(2^m))$. Notons que tous les opérandes utilisés dans cette section ont la particularité d'appartenir à la même courbe elliptique E, satisfaisant l'équation élémentaire (2.59).

Une vue globale de l'implantation de la multiplication scalaire est présentée dans la figure 4.18.

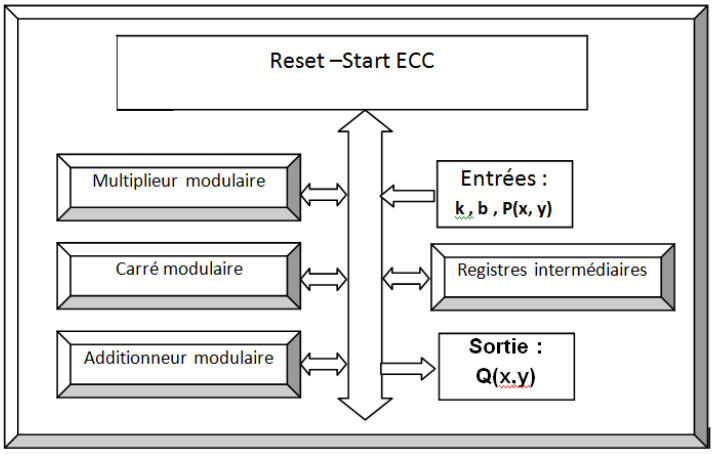



Figure 4.18 : Schéma de l'implantation de la Multiplication Scalaire.

Étant donné un point $P$ de la courbe, dont les coordonnées sont $\{x, y\} \in GF(2^m)$ , une clé $k$de taille $m$, et l'équation (2.59) de la courbe $E: y^2 + x.y = x^3 + a.x^2 + b$

Il y'a lieu de calculer : $Q = k \times P$ (4.10)

Comme décrit dans la section 2.6.1., la première étape dans la multiplication scalaire, selon l'approche de Montgomery, consiste à convertir les cordonnées affines en coordonnées projectives [2] évitant ainsi la division modulaire, à chaque cycle, en posant :

$$x = \frac{X}{Z} \quad (4.11)$$

$$y = \frac{Y}{Z} \quad (4.12)$$

Ce qui transforme l'équation (2.59), $E: y^2 + xy = x^3 + ax^2 + b$ en :

$$Y^2Z + XYZ = X^3 + aX^2Z + bX^3 \quad (4.13)$$

### 4.5.6 Algorithme de Montgomery

L'algorithme de Montgomery concernant la multiplication scalaire se divise alors en 4 étapes principales :

**Étape 1** : Conversion des coordonnées affines en coordonnées projectives

$$X_1 = x\ ;\ Z_1 = 1\ ;\ X_2 = x^4\ ;\ Z_2 = x^2$$

**Étape 2** : Addition et doublement scalaire

Boucle de i = m-2 à 0 faire

Si $k_i$ =1 faire

($X_1$, $Z_1$)= Montgomery_Add($X_1$, $Z_1$, $X_2$, $Z_2$);

($X_2$, $Z_2$)= Montgomery_Double($X_2$, $Z_2$)

Sinon faire

($X_2$, $Z_2$)= Montgomery_Add($X_2$, $Z_2$, $X_1$, $Z_1$);

($X_1$, $Z_1$)= Montgomery_Double ($X_1$, $Z_1$)

Fin si

Fin boucle

**Étape 3** : Conversion des coordonnées projectives en coordonnées affines

$$x_3 = \frac{X_1}{Z_1}$$

$$y_3 = (x + x_3)[(X_1 + x.Z_1)(X_2 + x.Z_2) + (x^2 + y)(Z_1.Z_2)](x.Z_1.Z_2)^{-1} + y$$

**Étape 4** : Génération du résultat $Q(x_3, y_3)$ [3]

Les blocs/ fonctions « Montgomery_Add et Montgomery_Double » représentent le cœur de notre implantation, nous allons présenter, dans les sections suivantes, leur fonctionnement ainsi qu'une estimation des ressources nécessaires à leur implantation.

4.5.6.1 <u>La conversion des coordonnées affines en coordonnées projectives</u>

Depuis le début de l'étude, nous considérons un point de courbe elliptique comme le couple $[(x; y) \in K.K]$. On dit que le point est représenté en coordonnées affines. Une alternative est le système de coordonnées projectives. Grâce au système projectif, on évite le calcul d'inverses dans les opérations d'additions et doublements de points. Cela est très intéressant lorsqu'on désire implémenter ces opérations car sur de nombreuses plateformes le coût d'une inversion dans un corps fini est prohibitif.

La conversion des coordonnées affines en coordonnées projectives s'effectue au début de la procédure multiplication scalaire, par de très simples équations matérielles, nécessitant deux carrés modulaires, en 3 cycles, consécutifs.

$$X_1 = x\ ;$$

$$Z_1 = 1\ ;\ \text{(Sur 113 bits)}$$

$$Z_2 = x^2$$

$$X_2 = x^4\ ;$$

$x$ : étant la coordonnée du point $P$. Remarquons que l'ordonnée $y$ n'intervient dans les calculs que lors de l'étape 3 de l'algorithme de la section 4.3.1.5.1, ce point fut la prouesse de l'algorithme de Montgomery dans son approche de calcul de la multiplication scalaire dans les courbes elliptiques.

4.5.6.2 L'addition scalaire (Montgomery_Add)

L'addition scalaire se réalise sur la courbe elliptique $E$, donc le point résultant appartient, forcément, soit à la courbe en question, soit c'est un point à l'infini, qui suggère que l'on a additionné un point avec son symétrie. Ce qui ne doit jamais être fait en cryptographie, pour cette raison l'algorithme de la section 2.17.5, détecte les courbes ou points malicieux, que le NIST [29] décommande très fortement.

L'algorithme d'addition scalaire défini par "Montgomery_Add" est représenté par l'équation (4.14).

$$\begin{cases} X_2 = X_1^4 + b \times Z_1^4 \\ Z_2 = X_1^2 \times Z_1^2 \end{cases} \quad (4.14)$$

La complexité de l'équation (4.14) est définie dans le tableau 4.4.

Tableau 4.4 : Complexité de l'addition scalaire de Montgomery dans$GF(2^m)$.

| Operations de base | Nombre |
|---|---|
| Multiplication modulaire | 2 |
| Carré modulaire | 4 |
| Addition modulaire | 1 |

4.5.6.3 Le doublement elliptique (Montgomery_Double)

Le doublement d'un point sur la courbe elliptique défini comme "Montgomery_Double" est représenté par l'équation (4.15).

$$\begin{cases} Z_3 = (X_1 \times Z_2 + X_2 \times Z_1)^2 \\ X_3 = x \times Z_3 + (X_1 \times Z_2)(X_2 \times Z_1) \end{cases} \quad (4.15)$$

La complexité de l'équation (4.15) est illustrée dans le tableau 4.16.

Tableau 4.5 : Complexité du doublement elliptique de Montgomery dans $GF(2^m)$.

| Operations de base | Nombre |
|---|---|
| Multiplication modulaire | 3 |
| Carré modulaire | 1 |
| Addition modulaire | 2 |

Les différentes opérations d'addition scalaire et de doublement elliptique sont répétées 112 fois dans $GF(2^{113})$, celles-ci sont conditionnées alternativement par la valeur des bits de la clé $k$.

4.5.6.4 La conversion des coordonnées projectives en coordonnées affines

Les opérations d'addition et de doublement elliptique de Montgomery s'arrêtent lorsque tous les bits de k sont testées, la conversion vers les coordonnes d'origine est alors entamée, selon les équations (4.16) et (4.17).

$$\begin{cases} x_3 = \frac{X_1}{Z_1} & (4.16) \\ y_3 = (x + x_3)[(X_1 + x.Z_1)(X_2 + x.Z_2) + (x^2 + y)(Z_1.Z_2)](x.Z_1.Z_2)^{-1} + y & (4.17) \end{cases}$$

L'utilisation des coordonnées projectives permet d'éviter de calculer des inversions Modulaires, en considérant qu'un point *P* de la courbe représentée par un triplet (*X* : *Y* : *Z*) en coordonnées projectives correspond au point représentée en coordonnées affines (*X/Zc, Y/Zd*), où les paramètres *c* et *d* dépendent du système de coordonnées projectives choisi.

Les dénominateurs sont ensuite éliminés dans l'équation de la courbe de Weierstrass ainsi que dans les formules d'addition et de doublement (: il n'y a donc plus d'inversions modulaires à calculer pour effectuer l'addition et le doublement de point.

4.5.7 La multiplication modulaire

La multiplication modulaire est l'une des opérations les plus complexes, après l'inverse modulaire, vue que la taille des opérandes, en cryptographie, ne correspond pas toujours à des multiples de la taille machine standard (8, 16, 32, 64 bits). Il est alors impératif de réaliser des multiplieurs à base d'opérandes de très grande taille (128, 137, 163, 413, 512, 1024). La complexité de l'operateur multiplicatif est basée sur des additions modulaires décalées, sans retenue, ainsi que des produits binaires (porte ET) [84], en plus d'une réduction modulaire.

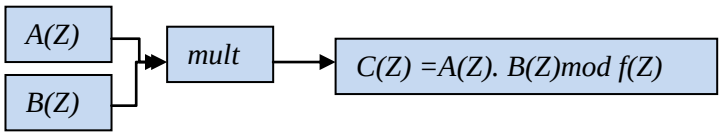



Figure 4.19 : opération de la multiplication modulaire.

Étant donné un polynôme résultant de la multiplication modulaire sous la forme suivante :

$$C(z) = A(z).B(z)\ mod\ f(z) \quad (4.18)$$

Où $A(z)$ $B(z)$ sont des éléments de $GF(2^m)$,

Le résultat $C(z)$ avant réduction polynômiale contient « $2.m - 1$ » éléments :

$$C(z) = c_{2m-2}z^{m-1} + \cdots + c_{m-1}z^{m-1} + \cdots + c_2z^2 + c_1z^1 + c_0 \quad (4.19)$$

Qui peut être écrit sous une forme vectorielle plus simple :

$$C = (c_{2m-2}, \ldots, c_2, c_1, c_0) \quad (4.20)$$

Tels que $c_i$ sont des éléments binaires.

Dans notre implantation, nous nous sommes basés sur l'algorithme de Karatsuba-Ofman, qui est basé sur une segmentation optimale des opérandes, utilisant des multiplieurs d'ordre réduit.

Si l'on réécrit les opérandes A et B sous une forme polynomiale :

$$A(z) = (L_A \, xor \, M_A.z^{\delta} xor \, H_A.z^{2\delta}) \quad (4.21)$$

Respectant : $taille(L_A) + 2.\delta = m$ (4.22)

Avec ($L_A$ , $M_A$ , $H_A$ ) les sous opérandes de A, et ($L_B$ , $M_B$ , $H_B$ ) les sous opérandes de B. Lorsque deux éléments du même GF ($2^m$), sont tous multipliés ensemble, le résultat initial contient des éléments «2.m - 1». Par «m -1» Xoring successif avec le polynôme générateur de degré « m + 1 » ou via un remplacement symbolique du XOR répété, par un ensemble d'équations « m » calculées dans Matlab séquentiellement, le résultat est remis à m bits.

La multiplication GF est développée en deux étapes :

- la première étape utilise l'algorithme Karatsuba-Offman [88] pour la scission et conquérir,
- la deuxième étape utilise une multiplication de base générateur pour l'état final des petits multiplicateurs, en utilisant la base Fonctions "bitand" et "bitxor" de Matlab, comme illustré à la Fig. 4.20.

```
function C=Mult13bits_113(A,B)
 %% inputs A and B are binary inputs of size m
%%Output C=A*B is of size 2*m-1 elements
C(1)=bitand(A(1),B(1));
e(1)=bitand(A(1),B(2));
C(2)=bitxor(bitand(A(2),B(1)),e(1)); ....
clear e
 e(1)=bitand(A(17),B(18));
C(34)=bitxor(bitand(A(18),B(17)),e(1));
 C(35)=bitand(A(18),B(18));
return
```

Figure 4.20 : Vue partielle du multiplicateur de base en GF ($2^{113}$), sans Réduction du Polynôme.

### 4.5.8 Concept de Karatsuba-Ofman

Le but des algorithmes de division et de conquête est de réduire les gros problèmes (difficiles) en un ensemble de problèmes plus petits (plus faciles). Dans notre cas, nous partitionnons les polynômes d'entrée (multiplicandes) donc à la place d'effectuer une multiplication de très grands polynômes, nous effectuons de nombreuses multiplications de polynômes beaucoup plus petits et enfin de combiner les résultats partiels. Il est même possible de partitionner nos entrées en coefficients uniques, mais cela peut ne pas être très efficace.

Une des plus populaires les améliorations de la méthode de multiplication classique sont l'astuce de Karatsuba-Ofman [7]. L'astuce améliore la version « diviser pour mieux régner » de l'algorithme de multiplication.

Théoriquement, cela diminue le nombre des opérations les plus complexes que nous devons effectuer pour obtenir le produit de a et b.

Différents choix de segmentations minimales ont été testées, voir tableau 4.5 et 4.6 . La plus optimale a concerné la segmentation pyramidale (113, 45, 15, 5). Notons que, le nombre 113 est un nombre premier, donc non divisible par 45, donc il y'a lieu de compléter le registre de 113 bits au premier nombre multiple de 45, la même procédure est réalisée au multiplieur de 15 bits, ceci est illustré dans le tableau 4.6.

Tableau 4.6 : Segmentation de l'opérande d'entrée en trois Sous-opérandes symétriques.

| Multiplieur brique de base Niveau 1 | Taille multiplieur Niveau 2 | Taille multiplieur Niveau 3 | Taille du résultat optimal | Taille maximale |
|---|---|---|---|---|
| 5 | 15 | 45 | 135 | 180 |
| 7 | 21 | 63 | 189 | |
| **7** | **19 à 21 bits** | **57** | **171** | |
| 9 | 27 | 81 | 162 ou 243 | |
| 11 | 33 | 99 | 297 | |

Tableau 4.7 : Segmentation de l'opérande d'entrée en deux Sous-opérandes symétriques.

| Multiplieur brique de base Niveau 1 | Taille multiplieur Niveau 2 | Taille multiplieur Niveau 3 | Taille multiplieur Niveau 3 | Taille multiplieur Niveau 4 | Taille du résultat non-optimal |
|---|---|---|---|---|---|
| 5 | 10 | 20 | 40 | 80 | 160 |
| 7 | 14 | 28 | 56 | 112 | 224 |
| 9 | 18 | 36 | 72 | 114 | 228 |

| | | | | | |
|---|---|---|---|---|---|
| **11** | **22** | **44** | **88** | **176** | |

Tableau 4.8 : Segmentation des opérandes du multiplieur de 113 bits.

| Taille de l'opérande | Taille identique des sous-opérandes |
|---|---|
| 113 | 45 x3 |
| 45 | 15 x3 |
| 15 | 5 x 3 |

La construction de la pyramide multiplicative à base d'opérandes de 113 bits étendus à 135 bits est illustrée dans le tableau 4.9.

Tableau 4.9 : Pyramide des composants de la multiplication modulaire pour des opérandes de taille 135 bits maximum.

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 135 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 45 | | | | | | | | | 45 | | | | | | | | | 45 | | | | | | | | | | | | |
| 15 | | | 15 | | | 15 | | | 15 | | | 15 | | | 15 | | | 15 | | | 15 | | | 15 | | | | | | |
| 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |

Le composant 5- du bas de la pyramide pour 113bits et 6-bits du bas de la pyramide est un composant issu d'une multiplication binaire avec addition à base de portes Et et XOR.

L'addition pyramidale décalée des multiplieurs (5, 15 et 45-bits) est illustrée dans la figure 4.21.

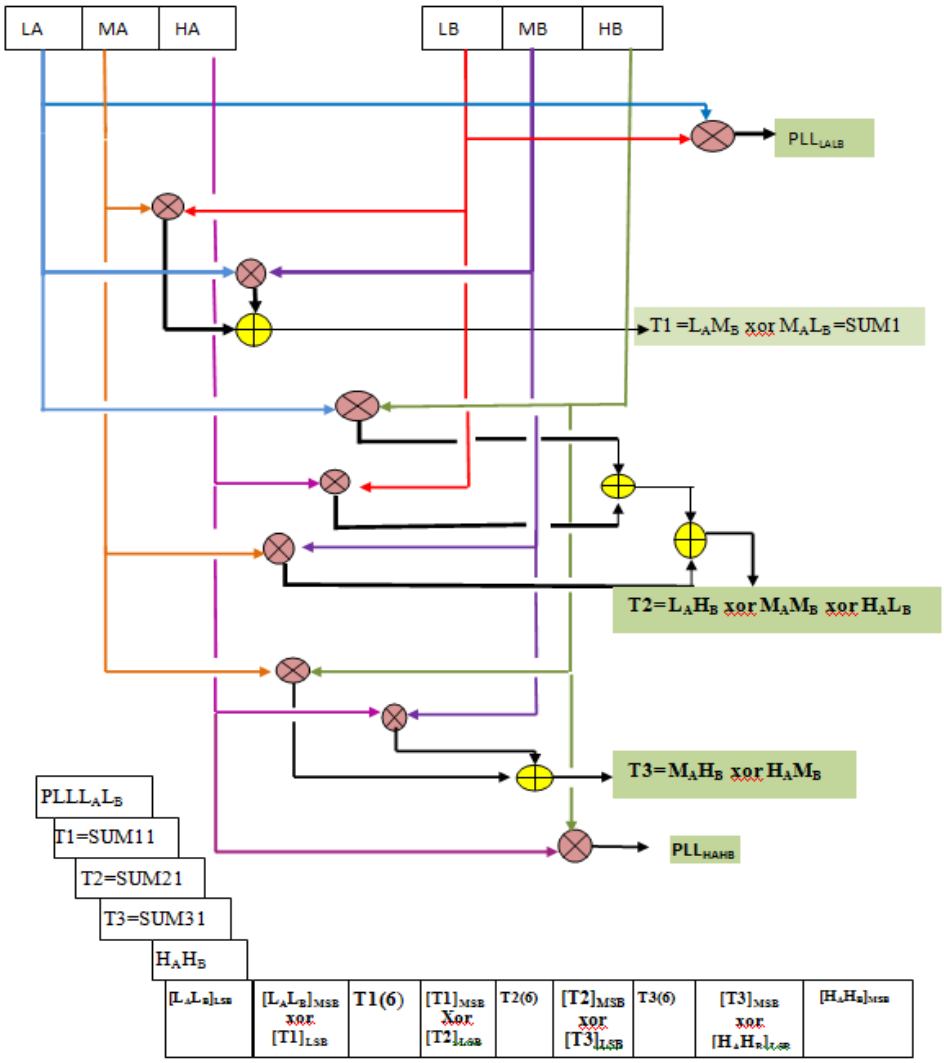



Figure 4.21 : Méthode de l'addition décalée du multiplieur de l'algorithme de Karatsuba-Ofman.

La notation [$L_AL_B$] représente une partie du registre $L_AL_B$ issu du sous produit $L_A$ par $L_B$ et T1(6) représente le bit de position 6, du registre T1.

La segmentation d'opérandes en trois sous-opérandes, permet une plus grande profondeur pyramidale verticale contre un développement horizontal [37].

L'impact des coefficients nuls adjoint est plus au moins considérable, toutefois il résout le compromis des multiplieurs asymétriques, comme illustré dans le tableau 4.10.

Tableau 4.10 : Pyramide des composants de la multiplication modulaire pour d'opérande de taille 113 bits maximum, cas asymétrique.

| 113 | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 47 | | | | | | | | | 33 | | | | | | | | | 33 | | | | | | | | |
| 17 | | | 17 | | | 13 | | | 11 | | | 11 | | | 11 | | | 11 | | | 11 | | | 11 | | |
| 5 | 5 | 7 | 5 | 5 | 7 | 5 | 5 | 3 | 1 | 5 | 5 | 1 | 5 | 5 | 1 | 5 | 5 | 1 | 5 | 5 | 1 | 5 | 5 | 1 | 5 | 5 |

On remarque que la segmentation asymétrique implique l'utilisation d'une panoplie de multiplieurs de base dans le cas :

- l'opérante 113 = (1,3,5,7,11,13,17,33 et 47 bits ) ;

Ce qui complique considérablement la procédure des additions décalées.

Le résultat issu de la multiplication modulaire est un opérande de taille "$2 \times m - 1$" bit :

- opérante de taille « 2x113 -1 » soit 225 bits

Ce qui requiert une réduction polynomiale. Celle-ci est décrite dans la section 4.3.1.2.

## 4.6 Conception de la boite à outils

Avant de mettre en œuvre notre boite à outil de courbe elliptique, plusieurs procédures doivent être effectuées concernant le champ fini, la courbe elliptique et le protocole cryptographique ainsi que les différentes opérations tels que les multiplications scalaires, multiplications modulaires ainsi que les différentes coordonnées.

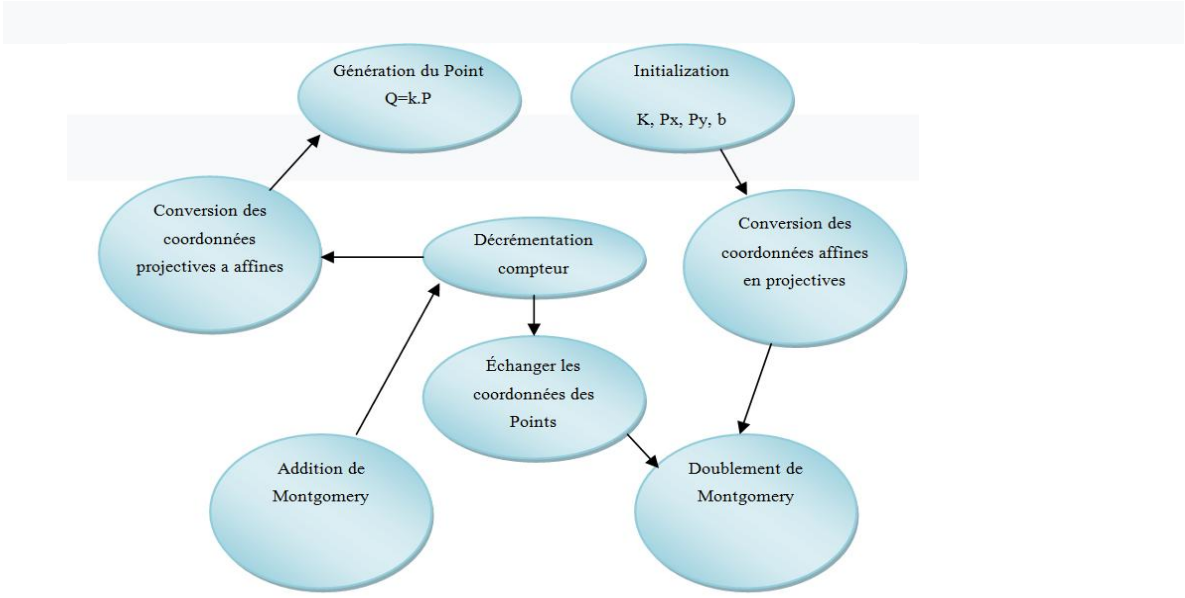



Figure 4.22 : implémentation des différentes opérations modulaires optimisées.

La figure 4.22, nous décrit l'implémentation des différentes opérations modulaires optimisées.

La conception du protocole d'échange à base de courbe elliptique se déroule sur plusieurs étapes.

1- La première chose à faire c'est qu'Alice et Bob doivent se mettre d'accord et ensemble et publiquement sur :

a. Alice et Bob s'accordent sur un groupe (G, +, O), générateur du groupe, sur la sélection de l'équation de la courbe de NIST et de la taille de l'opérande, E(a,b,k) c'est-à-dire choisissent un corps fini K.
b. Choisissent aussi ensemble et toujours publiquement un point P sur la courbe.

2- Alice et Bob choisissent leurs clés secrètes $K_A$ et $K_B$.
3- Vérification si point initial P( Px et Py) se trouve sur la courbe ECC. Pour cela nous devrions vérifier la partie droite et égale à la partie gauche.
4- Démarrez l'échange de protocole en calculant $K_A$ (Px,Py) et $K_B$ (Px,Py)
5- Vérification que le résultat de cette multiplication se trouve sur la courbe ECC. Pour cela nous devrions vérifier la partie droite et égale à la partie gauche. « Coté Alice »
6- Démarrez l'échange de protocole en calculant $K_B$ (Px,Py)
7- Vérification que le résultat de cette multiplication se trouve sur la courbe ECC. Pour cela nous devrions vérifier la partie droite et égale à la partie gauche « coté Bob ».
8- Echange de point $Q_A$ et $Q_B$ , c'est la clé publique
9- Alice reçoit $Q_B$ et utilise sa clé privée $K_A$ pour calculer $K_A\ Q_B = K_A(K_BP)$
10- Trouvez la clé de chiffrement secrète en calculant $K_B\ Q_A$ coté Alice
11- Vérification que le résultat de cette multiplication se trouve sur la courbe ECC.
12- Bob reçoit $Q_A$ et utilise sa clé privée $K_B$ pour calculer $K_B\ Q_A = K_B(K_AP)$
13- Trouvez la clé de chiffrement secrète en calculant $K_A\ Q_B$ coté Alice
14- Vérification que le résultat de cette multiplication se trouve sur la courbe ECC.
15- Tester que $K_A\ Q_B = K_B\ Q_A$
16- Vérification que le résultat de cette multiplication se trouve sur la courbe ECC.

## 4.7 Les procédures de l'Implémentation du protocole ECDH sur Matlab et étape intermédiaire de la simulation.

L'implémentation passe par plusieurs procédures et donc plusieurs opérations telles que :

- Multiplication scalaire par la méthode de Montgomery
- L'addition (XOR)
- La multiplication modulaire
- Elévation au carré
- Inverse d'un nombre par la methode Itoh-Tsujiii.[7]

La conception du protocole d'échange à base de courbe elliptique se déroule sur plusieurs étapes comme le montre la figure ci-dessus.

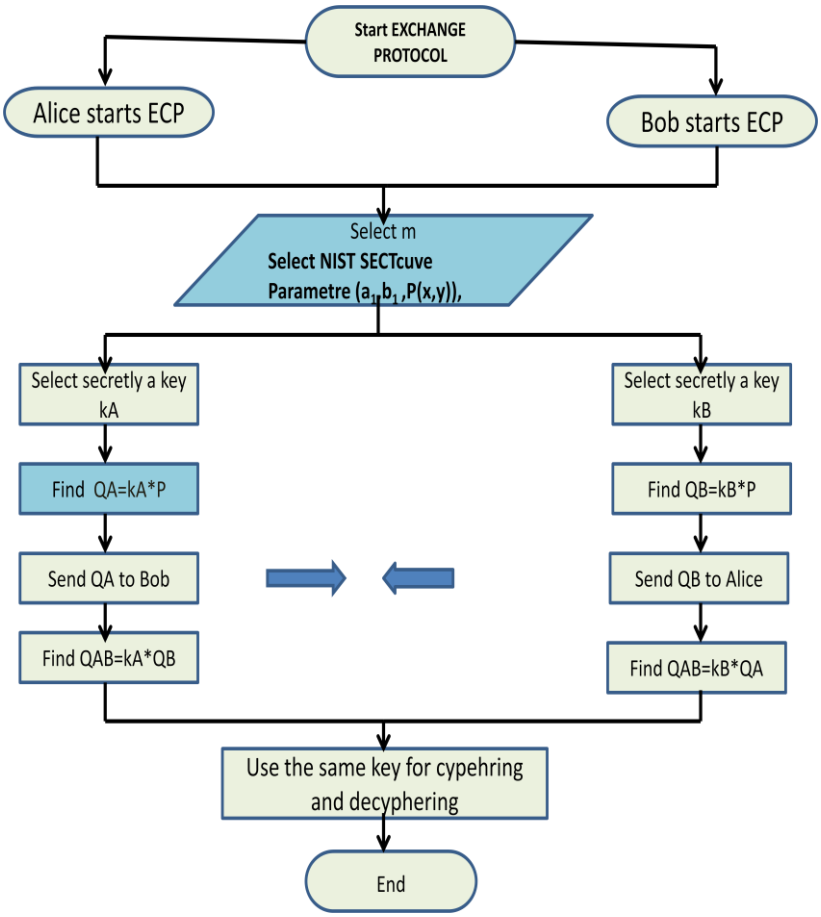



Figure 4.23 : La conception du protocole d'échange à base de courbe elliptique.

4.7.1 Procédure de génération de l'équation de la courbe elliptique

Alice et Bob se mettent d'accord ensemble et publiquement sur une courbe elliptique E(a,b,k), les paramètres « a et b » , ainsi que le point P ( Px ,Py) qui appartiennent à cette courbe choisi .

- C'est-à-dire choisissent un corps fini K ;
- L'équation de cette courbe ;
- Et le point P qui le point générateur.

```
NISTCurve=1
[a,b,x,y]=select_nist_curve(m,NISTCurve);
```

Figure 4.24 : sélection des paramètres de la courbe de NIST.

4.7.2 Sélectionnons la courbe NIST recommandées souhaitée type 1 commune à Alice et Bob.

En vue de valider l'implantation software, nous avons suivi les recommandations de la norme NIST, B-113 [29], spécifiant les valeurs de $a, b, P$, et $k$, comme suit, en notation hexadécimale :

```
function [a,b,x,y]=select_nist_curve(m,NIST_Points)
% Gives the parameters of the selected Curve

switch m
   case 113
      %%       3.2.1 Recommended Parameters sect113r1
      % % f (x) = x^113+x^9+1
      switch NIST_Points
         case 1
          a1 ='003088250CA6E7C7FE649CE85820F7';
            b1 ='00E8BEE4D3E2260744188BE0E9C723';
            %     S = '10E723AB14D696E6768756151756FEBF8FCB49A9';
            %                 G=
'04009D73616F35F4AB1407D73562C10F00A52830277958EE84D1315ED31886';
            x1 ='03009D73616F35F4AB1407D73562C10F';
            y1 ='00A52830277958EE84D1315ED31886';
```

Figure 4.25 : choix des paramètres de la courbe de Nist pour m=113.

### 4.7.2 Conversions des valeurs hexadécimales vers les valeurs binaires

En exécutant cette procédure de la figure 4.8 les valeurs en hexadécimales des pointes « a , b, Px et Py » vont être convertis en binaires comme le montre la figure ci-dessous.

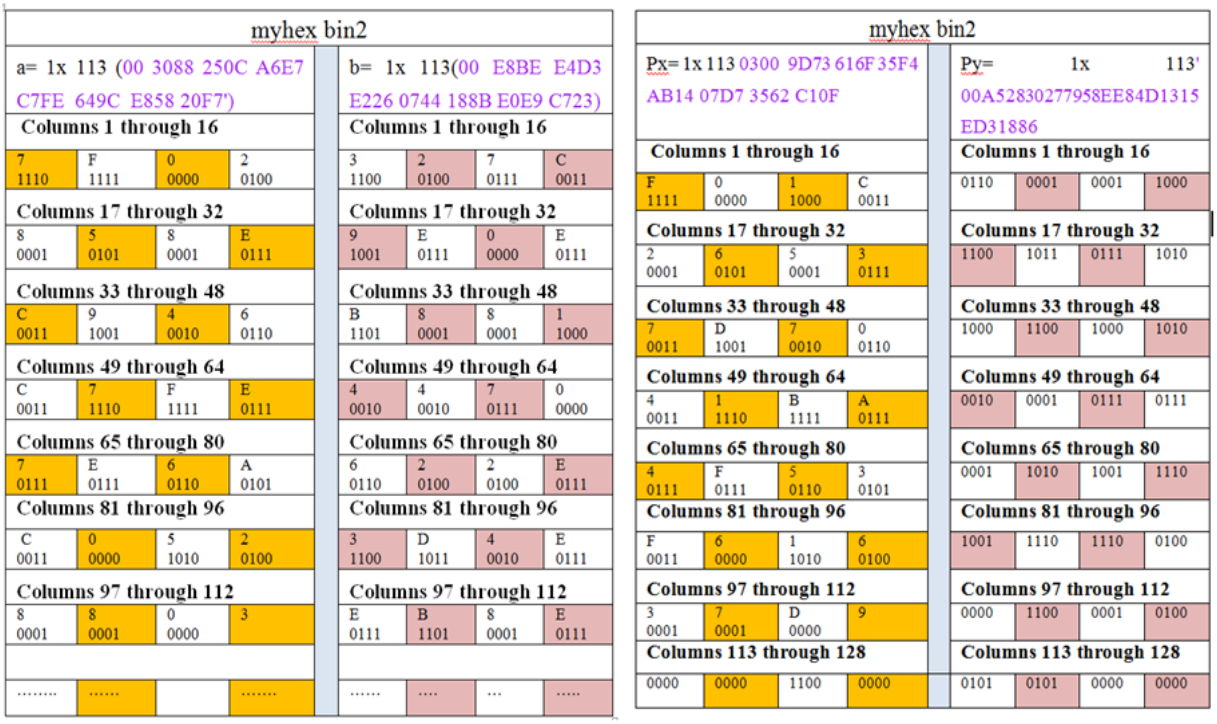



Figure 4.26 : Étape de la conversion des paramètres a,b et p
de la courbe de l'hexadécimal en binaire.

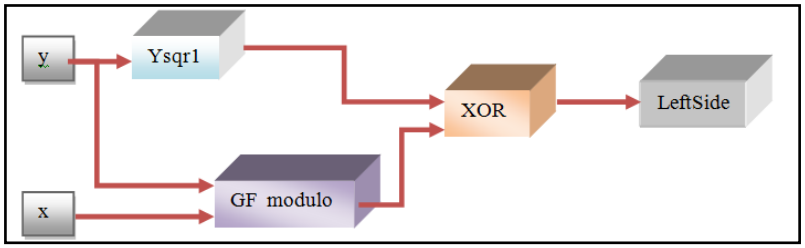
### 4.7.3 Procédure de vérification des deux points sur la courbe

L'étape de vérification nécessité une brique d'opération comme illustré sur la figure 4.28 (a) et (b), nous constatons que pour réaliser cette étapes il faudrait plusieurs opérations tels que :

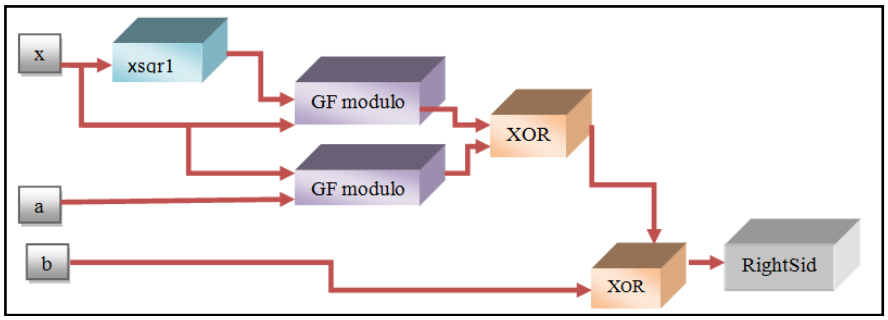
- Deux opérations d'élévations au carré (ysqr1 et xsqr1), **« GF Squaring «**
- Trois opérations de multiplication modulaire **« GF modulo »**
- Trois opérations d'addition « **GF XOR »**

y^2 +x*y = x^3 +a*x^2 +b.

Appelons « y^2 +x*y » = **LeftSide** et « x^3 +a*x^2 +b » = **RightSide**




(a) : opération modulaire « LeftSide ».




**(**b)**:** opération modulaire « RightSide ».

Figure 4.27: Procédure de vérification des deux points sur la courbe

Les figures ci-dessous illustrent les programmes qui vont réaliser cette procédure.

```
function
[LeftSide,RightSide,P_belongsToTheCurve]
=VerifyPointsOnCurve(x,y,a,b,m)
ysqr1=Galois_2PowerX(y,1,m);
xy=Galois_Mult(x,y,m);
LeftSide=G_Addition(ysqr1,xy);
xsqr1=Galois_2PowerX(x,1,m);
x3=Galois_Mult(xsqr1,x,m);
a_xsqr1=Galois_Mult(xsqr1,a,m);
add1=G_Addition(x3,a_xsqr1);
RightSide=G_Addition(add1,b);
```

(a)

```
function
[LeftSide,RightSide,P_belongsToTheCurve]
=VerifyPointsOnCurve(x,y,a,b,m)
ysqr1=Galois_2PowerX(y,1,m);
xy=Galois_Mult(x,y,m);
LeftSide=G_Addition(ysqr1,xy);
xsqr1=Galois_2PowerX(x,1,m);
x3=Galois_Mult(xsqr1,x,m);
a_xsqr1=Galois_Mult(xsqr1,a,m);
add1=G_Addition(x3,a_xsqr1);
RightSide=G_Addition(add1,b);
```

(b)

Figure 4.28: implémentation de l'étape de vérification de la partie (a) « LeftSide » (b) « LeftSide » de l'équation**.**

La figure 4.29 vérifie l'égalité de l'équation de la courbe choisie ; et cette vérification nous permettra de déduire que le point p appartient à la courbe.

```
LeftSideHex1_x_y= mybin2hex(LeftSide,m)
RightSideHex1_x_y= mybin2hex(RightSide,m)
if RightSide==LeftSide
   P_belongsToTheCurve=1
else
   P_belongsToTheCurve=0;
end
return
```

Figure 4. 29 : implémentation du programme de vérification du point p sur la courbe.

- Les polynômes binaires se représentent sous la forme de tableaux hexadécimaux. La figure 4.30 montre le calcul manuel de la partie gauche « LeftSide » : y^2 +x*y en polynômes binaires.

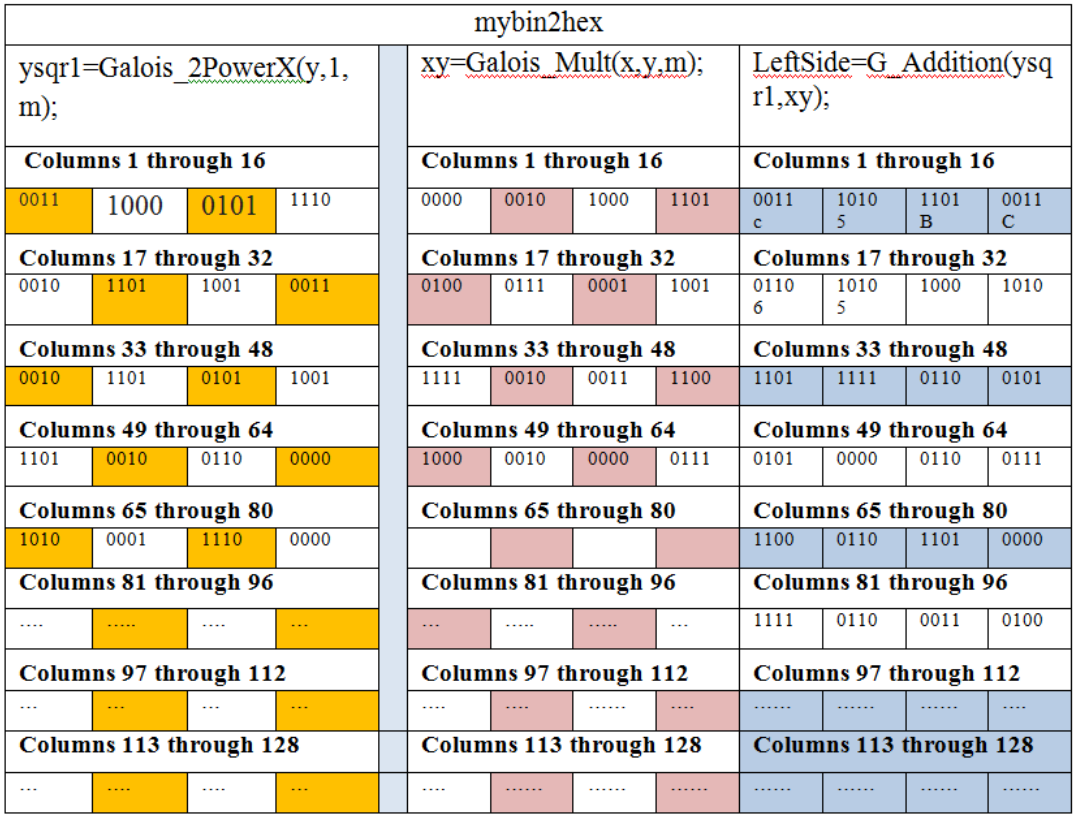



Figure 4.30 : Résultat de l'opération de la partie LeftSide : y^2 +x*y .

Le résultat de l'opération est :

| | |
|---|---|
| LeftSideHex1_x_y = | 1 5276 2C6F 0B63 E60A A6FB 5156 CB5C |

- La figure 4.31 montre calcul manuel de la partie droite RightSide: x^3 +a*x^2 +b

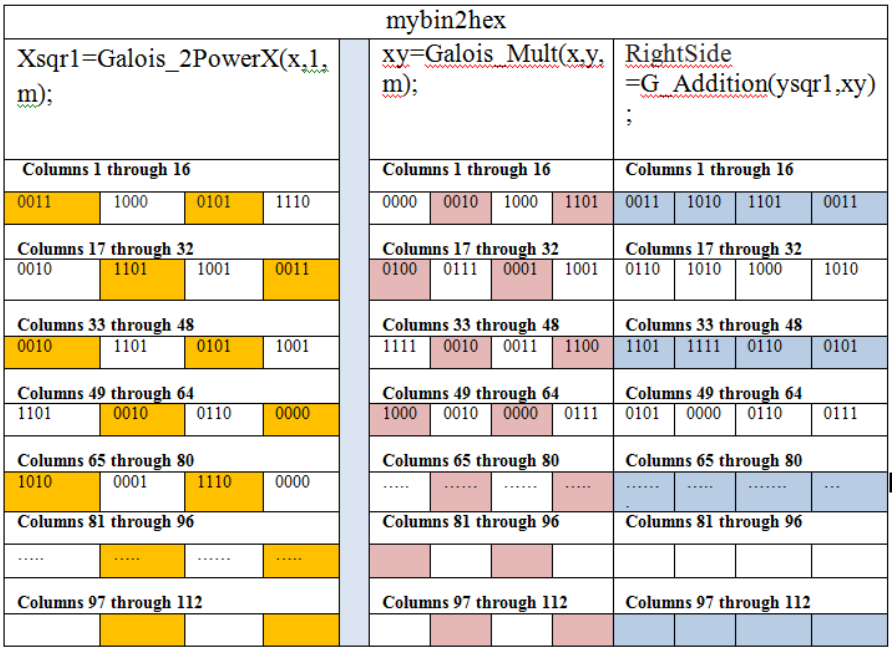



Figure 4.31 : Résultat de l'opération de la partie RightSide.

- Résultat de l'opération : x^3 +a*x^2 +b

| RightSideHex1_x_y = | 152762C6F0B63E60AA6FB5156CB5C |
|---|---|

- **Interprétation** : on constate que la partie droite de l'équation est égale à la partie gauche donc les deux points appartiennent à la courbe choisie.

| P_belongsToTheCurve = 1 | |
|---|---|
| LeftSide : y^2 +x*y | RightSide: x^3 +a*x^2 +b |
| 152762C6F0B63E60AA6FB5156CB5C | 152762C6F0B63E60AA6FB5156CB5C |

Figure 4.32 : résultat de la simulation de la vérification.

Ceci est implémenté dans la fonction VerifyPointsonCurve.m

### 4.7.4 Choix des entiers $K_A$ et $K_B$

Après avoir choisi la courbe commune, Alice et Bob choisissent séparément et au hasard des clés $k_A$ et $k_B$ chacun de son coté et secrètement, comme indiqué sur la Figure 4.33 et 4.34.

```
kbitsA= GenerateRandom_k(m);    % by Alice
kbitsB= GenerateRandom_k(m);    % by Bob
```

Figure 4.33 : génération du kA et kB par Alice et Bob.

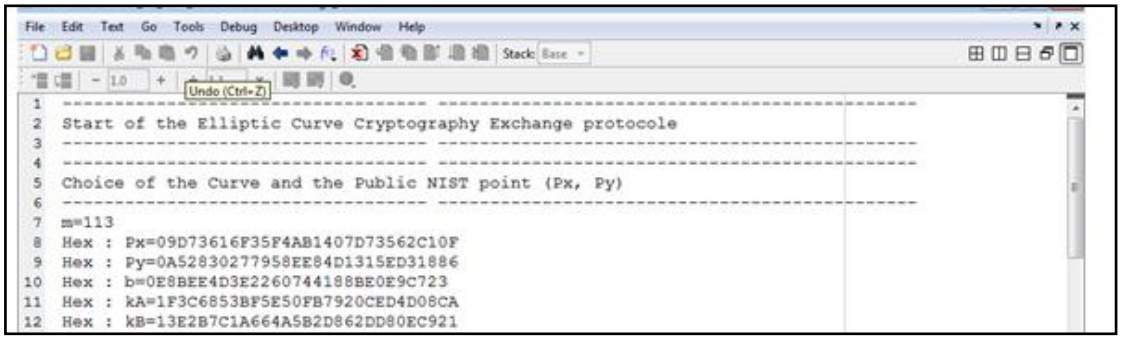



Figure 4.34 : fenêtre de la simulation.

### 4.7.5 Calcul de la clé secrète « $Q_A$ et $Q_B$ »

Alice et Bob calculent séparément $Q_A$ et $Q_B$ comme illustré sur la figure 4.35 , avec $Q_A = K_A (Px ,Py)$ et $Q_B = K_B (Px ,Py)$ pour cela on utilise la conversion des cordonnées Affine to Projective avec « m -1 » itérations. La multiplication scalaire est la principale opération effectuée.

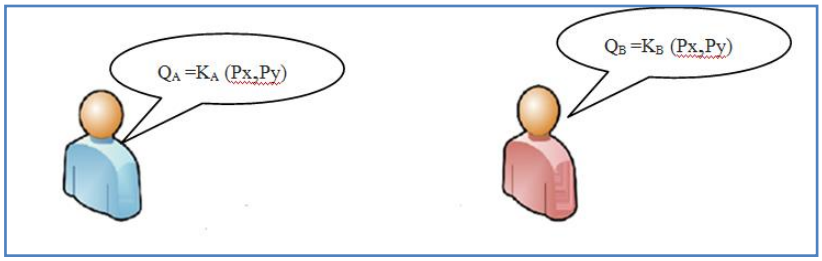



Figure 4.35 : multiplication scalaire du $Q_A = K_A (Px ,Py)$ et $Q_B = K_B (Px ,Py)$ par Alice et Bob.

```
if Do_ECC_QA
   %% Compute the QxA and QyA secretely
   %% A uses its curve to find the Point QA from P such that QA=kA*P
   [QxA QyA]=ECC_Scalar_mult(kbitsA,x,y,b,m)
   ECCQxA= myhex2ECCbits(QxA,m);
   ECCQyA= myhex2ECCbits(QyA,m);
……..
if Do_ECC_QB
   %% Compute the QxA and QyA secretely
   %% A uses its curve to find the Point QB from P such that QB=kB*P
   [QxB QyB]=ECC_Scalar_mult(kbitsB,x,y,b,m)
   ECCQxB= myhex2ECCbits(QxB,m);
   ECCQyB= myhex2ECCbits(QyB,m);
```

Figure 4.36 : implémentation de la multiplication scalaire.

Après 112 itérations pour m= 113 le résultat final sera reconverti des coordonnées Projectives vers coordonnées Affines.

**Résultat de la multiplication et verification du Qi sur la courbe**

- Calcul de Alice « $Q_A = K_A (Px ,Py)$ ».

**Conversion From Affine to Projective**
iteration : 111 …….iteration : 0
Nearly finished
**Conversion From Projective to Affine**
QxA = 1268AB7391CE10EC0F9D0D90528E1
QyA = 0589E64507F977298FFC9A4DF2441
**Verification de calcul des operations la partie RightSide et LeftSide**
LeftSideHex1_x_y = 0F5369987BFA607D5EE508429D2BA
RightSideHex1_x_y = 0F5369987BFA607D5EE508429D2BA
**P_belongsToTheCurve = 1**
LeftSideHex1 = 0F5369987BFA607D5EE508429D2BA
RightSideHex1 = 0F5369987BFA607D5EE508429D2BA
**the point belongs to the curve.**

Figure 4.37 : résultat de la multiplication scalaire $Q_A = K_A$ (Px ,Py) et vérification «y^2 +x*y = x^3 +a*x^2 +b » .

- Calcul de Bob « $Q_B = K_B$ (Px ,Py) ».

**Conversion From Affine to Projective**
iteration : 111 ……iteration : 0
Nearly finished
**Conversion From Projective to Affine**
QxB = 1188AA0732D5E188242960B96BFE7
QyB = 00647F891D2C18D97FE2DB15C00AD
**Verification de calcul des operations la partie RightSide et LeftSide**
LeftSideHex1_x_y = 1095628D52B8BA88FD6477315E0E0
RightSideHex1_x_y = 1095628D52B8BA88FD6477315E0E0
**P_belongsToTheCurve = 1**
LeftSideHex2 = 1095628D52B8BA88FD6477315E0E0
RightSideHex2 = 1095628D52B8BA88FD6477315E0E0
**the point belongs to the curve.**

Figure 4.38 : résultat de la multiplication scalaire $Q_B = K_B$ (Px ,Py) et vérification «y^2 +x*y = x^3 +a*x^2 +b ».

### 4.7.6 Protocole d'échange entre Alice et Bob

- Alice envoi à Bob $Q_A$ et Bob envoi à son tour $Q_B$.

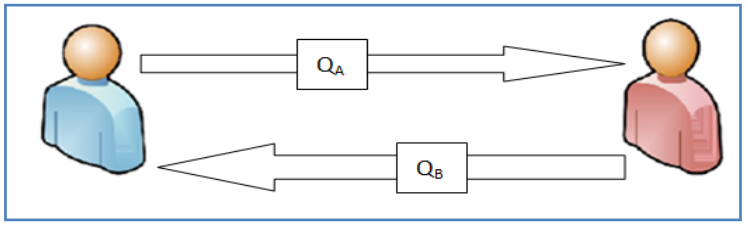



Figure 4.39: échange de clés.

- Ils calculent indépendamment leurs clés publiques.
  - Alice reçoit $Q_B$ et calcul : $Q_A = Q_B . K_A = (K_B$ (Px ,Py).$K_A$
  - Bob reçoit et calcul : $Q_B = Q_A . K_B = (K_A$ (Px ,Py).$K_B$

Une fois que Alice et Bob échangent leurs clés publiques, ils commencent à calculer leur clé commune secrète indépendamment tel que présenté à la Figure 4.39.

- Le résultat de la simulation et vérification que le nouveau point $Q_{ANEW}$ $k_A.Q_B$ appartient à la courbe est représenté sur la figure 4.41.

Conversion From Affine to Projective

iteration : 111 iteration : 2 iteration : 1 iteration : 0

Nearly finished

**Conversion From Projective to Affine**

QxNew_A =03674F3E75D8435BF02B9171B5675

QyNew_A =1C0B24773E34E3BF4166499A070A4

**Verification de calcul des operations la partie RightSide et LeftSide**

LeftSideHex1_x_y =1A2A1006E6D070D4505A6C130F376

RightSideHex1_x_y =1A2A1006E6D070D4505A6C130F376

**P_belongsToTheCurve = 1**

LeftSideHex3 =1A2A1006E6D070D4505A6C130F376

RightSideHex3 =1A2A1006E6D070D4505A6C130F376

Figure 4.40 : Résultat de la simulation et vérification du nouveau point $Q_{ANew} = Q_B. K_A$ .

- Résultat de la simulation et vérification que le nouveau point $Q_{BNEW} = k_B x Q_A$ appartient à la courbe est représenté sur la figure 4.41.

**Conversion From Affine to Projective**

iteration : 111 iteration : 2 iteration : 1 iteration : 0

Nearly finished

Conversion From Projective to Affine

QxNew_B = 03674F3E75D8435BF02B9171B5675

QyNew_B =1C0B24773E34E3BF4166499A070A4

**Verification de calcul des operations la partie RightSide et LeftSide**

LeftSideHex1_x_y =1A2A1006E6D070D4505A6C130F376

RightSideHex1_x_y =1A2A1006E6D070D4505A6C130F376

P_belongsToTheCurve = 1

LeftSideHex4 =1A2A1006E6D070D4505A6C130F376

RightSideHex4 =1A2A1006E6D070D4505A6C130F376

Figure 4.41 : Résultat de la simulation et Verification du nouveau

point $Q_{BNew} = Q_A . K_B$ .

Si quelqu'un a espionné leurs échanges, il connait E(a,b,K) , P , $k_A . P$ et $k_B . P$ . Pour pouvoir retrouver la clé $k_A . k_B . P$ , il faut pouvoir calculer $k_A . P$ connaissant P et $k_B . P$ .

4.8 Résultat de l'implémentation avec m=113

L'ECDH étant complet, un fichier journal automatique est généré afin de tester les résultats intermédiaires, la dernière clés de chiffrement-déchiffrement, ainsi que l'appartenance des résultats à la courbe, comme indiqué sur la figure 4.42.

**Start of the Elliptic Curve Cryptography Exchange protocole**

**Choice of the Curve and the Public NIST point (Px, Py) /** m=113

**Hex : Px**=09D73616F35F4AB1407D73562C10F /

**Hex : Py**=0A52830277958EE84D1315ED31886

**Hex : b** =0E8BEE4D3E2260744188BE0E9C723

**Hex : Ka**=16319AF2C3193B7B944141938F107 /

**Hex :Kb** =1F055F5A60D252BF9C9891EAF1BF8

**Check if following values are equal => Initial Points are on the ECC curve**

**LeftSide** =152762C6F0B63E60AA6FB5156CB5C

**RightSide**= 152762C6F0B63E60AA6FB5156CB5C

**Start Exchange protocole by computing kA.(Px,Py) :side 1 :Alice**

**QxA**=1268AB7391CE10EC0F9D0D90528E1/
**QyA**=0589E64507F977298FFC9A4DF2441

**Check if following values are equal => kA.(Px,Py) is on the ECC curve**

**LeftSide** =0F5369987BFA607D5EE508429D2BA

**RightSide**=0F5369987BFA607D5EE508429D2BA

**Exchange of the Points QA and QB : New computed Public keys**

**Find Secret Cryptographic key by computing QA.kB :side 1 (Alice)**

**QxNewA**=03674F3E75D8435BF02B9171B5675

**QyNewA**=1C0B24773E34E3BF4166499A070A4

**Check if following values are equal => QAB=QA.kB is on the ECC curve**

**LeftSide**=1A2A1006E6D070D4505A6C130F376

**RightSide**=1A2A1006E6D070D4505A6C130F376

**Aside Test : is QAB==QBA**

**Alice receives QB and uses its private key kA to compute kA.QB=kA.(kB.P)**

**Find Secret Cryptographic key by computing QA.kB :side 1 (Alice)**

**QxNewA**=03674F3E75D8435BF02B9171B5675
QyNewA=1C0B24773E34E3BF4166499A070A4
**Check if following values are equal => QAB=QA.kB is on the ECC curve**
**LeftSide** =1A2A1006E6D070D4505A6C130F376
**RightSide**=1A2A1006E6D070D4505A6C130F376
**x coordinate of the Points (within A and B)**
QABx=03674F3E75D8435BF02B9171B5675
QBAx=03674F3E75D8435BF02B9171B5675
**y coordinate of the Points (within A and B)**
**QABy=1C0B24773E34E3BF4166499A070A4**
**QBAy=1C0B24773E34E3BF4166499A070A4**
**End of the Elliptic Curve Cryptography Exchange protocole**

Figure. 4.42 : protocole ECDH d'accord Alice et Bob.

La figure 4.42 , montre l'étape finale de la simulation matérielle, les résultats $Q_x$ et $Q_y$ sur 113 bits, représentent les coordonnées du résultat. (Clé publique).

## 4.9 Conclusion

Dans ce chapitre, nous avons présenté les différentes étapes de la conception de notre boite à outils. Nous avons détaillé l'opération de la multiplication scalaire, l'inverse modulaire avec la méthode de d'Itoh-Tsjuii et Karatsuba-Ofman et bien sure la multiplication de Montgomery qui a pour objectif l'optimisation des opérations de calculs.

Nous nous sommes aligné aux normes internationales représentées par l'organisation Américaine NIST, en implantant les courbes recommandées par le NIST pour la valeur de m = [113].

Il y a deux opérations principales définies dans GF ($2^m$): l'addition et la multiplication. Toutes les autres opérations (inversion, division, quadrature, etc.) peuvent être effectuées par multiplication et addition.

La complexité de certaines opérations dépend de la représentation des éléments du champ.

Chacune de ces étapes fait appel à un outil bien spécifique. Nous avons présenté également, les résultats partiels de la simulation ainsi que les résultats d'implémentation obtenus.

A travers ces résultats, nous avons constaté que notre toolbox fonctionne très bien et le temps d'exécution est vraiment faible malgré que m =113 bits. Cette implémentation de l'ECDH a été testée avec succès avec le nombre premier m=113 bits.

# CONCLUSION GENERALE ET PERSPECTIVES

Pour conclure ce document de thèse, nous allons rappeler les différentes contributions proposées puis présenter certaines perspectives pour des travaux futurs sur l'extension de la boite à outils.

La cryptographie à clé publique joue un rôle fondamental pour la sécurité car elle offre des services que n'offre pas la cryptographie à clé secrète telle que la non répudiation entre autre. L'échange de clé peut être assuré par des protocoles qui utilisent l'idée du protocole Diffie-Hellman et cette dernière repose sur le logarithme discret, beaucoup de schémas utilisent le problème du logarithme discret dont le meilleur cadre d'utilisation actuel est le domaine des courbes elliptiques issues de la théorie des nombres. C'est un sujet très en vogue depuis une quinzaine d'années en cryptographie asymétrique.

Visant à contribuer au domaine de cryptographie à courbe elliptique, nous avons développé une nouvelle boîte à outils Matlab, qui peut gérer de champs Galois du type $GF(2^m)$ sur les courbes elliptiques, afin de réaliser le protocole d'échange ECDH en utilisant l' optimisation de l'opération de la multiplication scalaire. Pour cela un état de l'art a été élaboré survolant les différentes approches de la cryptographie classique et moderne ainsi que la cryptanalyse.

Puis nous avons détaillé les opérations modulaires ainsi que l'outils mathématiques dans le chapitre 2, qui est considéré comme la colonne vertébrale de notre conception car au niveau le plus haut, on trouve la multiplication scalaire notée [k]**P** avec k un grand entier appelé scalaire et **P** un point de la courbe. Un algorithme de [k]**P** est essentiellement une boucle qui parcourt tous les chiffres de k et effectue des ADD et/ou DBL en fonction de la valeur de chaque chiffre $k_i$. Plusieurs techniques de recodage de clés ont été proposées pour réduire le nombre d'opérations au niveau courbe en littérature. Certaines de ces techniques de recodage permettent d'accélérer encore plus la multiplication scalaire au prix de quelques pré-calculs.

Le chapitre 3, a fait l'objet d'un article scientifique parût dans le journal : « Indonesian Journal of Electrical Engineering and Computer Science Vol. 13, No. 3, March 2019, pp. 910~918 ». Nous avons constaté que les courbes de NIST sont munies d'une porte dérobée dans un générateur de nombres aléatoires basée sur les courbes elliptiques. Un seul point d'entrée dans le mécanisme de génération des nombres aléatoires et c'est l'ensemble de la sécurité du système qui est fortement compromis par une attaque.

Les constantes, relatives au générateur basé sur les courbes elliptiques ( Dual_EC_DRBG), voient ici leur valeur fixée à des nombres définis par défaut dans le standard ; lors de l'analyse de ce générateur, on a constaté que l'ensemble des séquences de nombres générées par Dual_EC_DRBG présente une relation étroite avec un deuxième ensemble de nombres secrets pouvant faire office de Skeleton key. Cette comparaison nous a permis de relever le mystère qui se pose sur le paramètres prédéfinis de NIST et surtout répondre à la question qui est « peut-on faire confiance à des courbes standards » et justifier le choix de Satoshi pour l'utilisation de la courbe Sec256k1 dans son protocole de Bitcoin qui suscita l'intérêt de beaucoup de spécialistes.

Donc le but ultime de ce projet est la mise en œuvre d'un Toolbox Matlab pouvant gérer de champs de Galois du type GF ($2^m$) et optimiser l'opération de multiplication. L'originalité des travaux présentés ici est qu'ils ne traitent pas chaque aspect séparément. En effet nous avons toujours cherché à développer l'arithmétique à un niveau donné en gardant à l'esprit son lien avec les niveaux inférieurs ou supérieurs. La difficulté principale est d'effectuer des calculs sur des opérantes de très grande taille sachant que Matlab manipule des nombre à 16 bits seulement or la cryptographie basée sur les courbes elliptiques manipule des opérantes de taille minimum 113 bits dans notre cas pour plus de sécurité. Comme mentionné précédemment, MATLAB est incapable de traiter avec précision les grands nombres entiers que nous rencontrons en cryptographie sur courbe elliptique. Les problèmes que cela crée peuvent parfois être surmontés, souvent grâce à l'utilisation des algorithmes comme celui utilisé pour l'exponentiation modulaire. Cependant, ce n'est pas toujours possible; au-delà d'un certain point, même des opérations de base comme la quadrature créent des nombres trop grands pour MATLAB. La limitation fondamentale résulte de la façon dont MATLAB stocke les nombres. L'IEEE double précision la forme à virgule flottante que MATLAB utilise normalement peut stocker avec précision des nombres entiers jusqu'à 4 503 599 627 370 495 (52 bits). Pour notre boite, nous avons ajouté des fonctions à la bibliothèque Matlab déjà existante. Cette boite est appelée ECC_NIST, utilisant la courbe de Weierstrass standardisée de NIST dans un champ de Galois de 113 bits, donc les paramètres de l'équation sont prédéfinies par ce standard. Comme mentionné précédemment, la multiplication scalaire est une boucle qui parcourt tous les chiffres de k et effectue des additions et des doublements de points sur la courbe donc le choix des algorithmes et méthodes de projections sont importants. Avec notre approche, une multiplication dans $GF(2^m)$ nécessite moins de multiplications comparativement aux algorithmes classiques .Nous avons utilisé les

coordonnées projectives modifiées des équations de Montgomery, L'algorithme de Montgomery concernant la multiplication scalaire s'avère parmi la méthode la plus efficace. Ce dernier est basé sur l'implantation de deux routines élémentaires, l'addition scalaire et le doublement elliptique toutes deux sont effectuées sur la courbe elliptique $E(GF(2^m))$. La première étape dans la multiplication scalaire, selon l'approche de Montgomery, consiste à convertir les cordonnées affines en coordonnées projectives évitant ainsi la division modulaire, à chaque cycle, donc nous avons une seule division au cycle final d'où l'optimisation de notre multiplication scalaire.

La deuxième étape consiste à faire de l'addition et doublement scalaire. La deuxième étape consiste à faire de l'addition et doublement scalaire. L'opération de l'inverse modulaire a été réalisé par la méthode de d'Itoh-Tsjuii , l'inverse en $GF(2^m)$ est défini comme $\frac{1}{A} = A^{2^m-1}$mod P . Pour traiter un inverse, il est nécessaire de diviser le $A^{2^m-1}$à un ensemble de puissance imbriqués, qui consiste à réécrire $A(z)^{2.m-2}$ sous formes de produits et carrés imbriqués, par la règle appelée «Multiplier et élever au carré», selon les indices de multiplication et d'addition générés.

La multiplication modulaire est l'une des opérations les plus complexes, après l'inverse modulaire, vue que la taille des opérandes, en cryptographie, ne correspond pas toujours à des multiples de la taille machine standard (8, 16, 32, 64 bits) et le Matlab est incapable de traiter cela. La complexité de l'operateur multiplicatif est basée sur des additions modulaires décalées, sans retenue, ainsi que des produits binaires en plus d'une réduction modulaire. Dans notre implantation, nous nous sommes basés sur l'algorithme de Karatsuba-Ofman, qui est basé sur une segmentation optimale des opérandes, utilisant des multiplieurs d'ordre réduit. La troisième étape consiste à faire une conversion des coordonnées projectives en coordonnées affines. Et enfin la dernière étapes Génération du résultat $Q(x_3, y_3)$.

Nous avons atteint l'état de l'art en matière de performance sur les produits scalaires. Et nos travaux ont été concluants, très encourageants et ouvrent plusieurs pistes de recherche et les perspectives de nos travaux sont nombreuses tels que :

- La boîte à outils sera également complétée par l'ajout des opérations de Galois sur GF (p), pour p premier comprenant également la génération de signature et de vérification de la courbe elliptique Digital Signature Algorithm (ECDSA).

- Implantation des différents algorithmes de chiffrement et déchiffrement, des standards NIST (131, 163,193, 233, 283, 409 et 507 bits), prenant en compte l'utilisation d'un ou plusieurs multiplieurs.
- Implémentation de la courbe Sect113r1 sur Tensorflow.
- Implémentation sur FPGA de la courbe Sect113r1 et l'avantage des FPGA est que nous pouvons optimiser notre division exacte et construire une architecture adaptée.
- Une implantation parallèle, l'utilisation des mémoires peut améliorer la conception, avec un coût supplémentaire des accès répétés.

Nous incitons fortement, les chercheurs de différentes universités Algériennes à s'orienter vers le domaine de la cryptographie, en vue de trois aspects :

- ✓ Comprendre comment les organismes étrangers chiffrent et déchiffrent nos communications téléphoniques, messageries, comptes bancaires, dossiers médicaux, etc.…
- ✓ Participer dans les normes de chiffrement / déchiffrement en vue de les adapter à nos stratégies de sécurité, soit nationale ou au niveau des fichiers personnels enregistrés dans nos "micro-ordinateurs", que l'on croit bien cachés, alors que ce sont de vraies portes ouvertes, surtout actuellement, avec les failles des systèmes d'exploitation et les accès illimités à l'Internet.
- ✓ Développer nos propres mathématiques de chiffrement / déchiffrement, en vue de modifier et/ ou adapter les algorithmes actuels de cryptographie à nos besoins.
- ✓ Ces opérateurs représentent les briques des ECC, donc l’accélération à ce niveau aura un impact incommensurable sur l’accélération du processus de la multiplication scalaire.

**REFERENCES**

.


1. Cybersecurity market report. Website. URL http://cybersecurityventures. com/cybersecurity-market-report/.
2. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf, 2009 .
3. National Institute of Standards and Technology. FIPS 186–2. Digital Signature Standard. Technical report, NIST, 2000. http://csrc.nist.gov/publications/fips/ archive/fips186-2/fips186-2.pdf.
4. www.certicom.com/index.php/ecc-tutorial.
5. S. Singh and C. Coqueret. Histoire des codes secrets : de l'Egypte des Pharaons à l'ordinateur quantique. Le Livre de poche. Librairie générale française, 2001. ISBN 9782253150978. URL. https://books.google.fr/books?id=awSxHAAACAAJ.
6. R. Rivest, A. Shamir, and L. Adleman. Method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21 :120–126, 1978.
7. F. Rodriguez-Henriquez, N.A. Saqib, A. Diaz-Perez and C. Kaya Koc, "Cryptographic Algorithms on Reconfigurable Hardware", Springer Editions, ISBN: 0387338837, 2006.
8. B. Schneier, "Applied Cryptography, Protocols, Algorithms, and Source code in C", John Wiley & Sons, Inc., ISBN: 0471128457,2nd Edition, 1996.
9. David Martins and Hervé Guyennet. Steganography in mac layers of 802.15. 4 protocol for securing wireless sensor networks. In Multimedia Information Networking and Security (MINES), 2010 International Conference on, pages 824–828. IEEE, 2010.
10. H.C.A. van Tilborg, "Fundamentals of Cryptology", KLUWER Academic Publishers, 2002.
11. S. Burnett, S. Paine, "RSA Security Official Guide to Cryptography", Mc Graw Hill Press, DOI: 10.1036/0072192259, 2001.
12. National Institute of Standards and Technology, FIPS PUB 186-2: Digital Signature Standard (DSS), Jan. 2000.
13. P-F. Bonnefoi Version du 19 novembre 2016 « Cours Sécurité et Cryptographie » ecole universitaire de Management Limoges : http://p-fb.net/
14. M. Wembo, "Modern Cryptography: theory and Practice", Prentice hall PTR, ISBN: 0-13-066943-1, 2003.

15. M. Mogollon, "Cryptography and security services: mechanisms and applications", Cybertech Publishing, ISBN 978-1-59904-837-6, 2007.
16. A. Young, M. Yung, "Malicious Cryptography: Exposing cryptovirology", Wiley publishing, 2004.
17. T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory 31, pages 469–472, 1985. doi : 10.1109/TIT.1985.1057074.
18. A. Menezes, P. van Oorschot, Vanstone, "Handbook of Applied Cryptography", CRC Press, 1996.
19. N. Koblitz, "Elliptic Curve Cryptosystems", Mathematics of Computation, American Mathematical Society, Vol 48, No. 177, pp: 203-209, 1987.
20. A. Young, M. Yung, "Malicious Cryptography: Exposing cryptovirology", Wiley publishing, 2004.
21. ICCES ICCES, vol.5, no.4, pp.255-261 Faster RSA Algorithm for Decryption Using Chinese Remainder Theorem G.N. Shinde1 and H.S. Fadewar 2.
22. M. Bellare and P. Rogaway. Optimal Asymmetric Encryption – How to Encrypt with RSA. In Eurocrypt ’94, LNCS 950, pages 92–111. Springer-Verlag, Berlin, 1995.
23. Second NESSIE Workshop (september 12 – 13, 2001, Egham, UK). RSA–REACT: An Alternative to RSA–OAEP Tatsuaki Okamoto1 and David Pointcheval2 1 NTT Labs, 1-1 Hikarino-oka, Yokosuka-shi, 239-0847 Japan.
24. Ian Blake, Gerald Seroussi, Gadiel Seroussi, and N Smart. Elliptic curves in cryptography, volume 265. Cambridge university press, 1999.
25. I. F. Blake, G. Seroussi, N. P. Smart, "Advances in elliptic curve cryptography", Cambridge University Press, ISBN-13 978-0-521-60415-4, ISBN: 0-7645-4975-8, 2005.
26. BARKER, E. B. SP 800-57. Recommendation for Key Management, Part 1 rev4. Rapp. tech. Gaithersburg, MD, United States, 2016.
27. A. Henkerson, A. Menezes, S. Vanstone, 2004. Guide to Elliptic Curve Cryptography. Springer-Verlag, ISBN: 038795273X, pp: 102,264.
28. J. M. Pollard “Monte Carlo methods for index computation (mod p)” Mathematics of Computation, volume 32, number 143 july 1978, pages 918-924.
29. NIST, "Standards For Efficient Cryptography: Recommended Elliptic Curves For Federal Government Use", 1999.

30. A. Henkerson, A. Menezes, S. Vanstone, 2004. Guide to Elliptic Curve Cryptography. Springer-Verlag, ISBN: 038795273X, pp: 102,264.
31. Barker, E., W. Barker, W. Burr, W. Polk and M. Smid, 2007. NIST SP 800-57: Recommendation for key management-part 1: General.http://www.citeulike.org/ user /dhein1030/ article/3730728.
32. D. Stinson, "Cryptography: theory and Practice", CRC Press, ISBN: 0849385210, pp.:40, 1995.
33. DOTTAX (E.) et PROUFF (E.), « La carte à puce, cœur de sécurité des systèmes mobiles », MISC n°41, 2009.
34. J.H. Silverman, "The Arithmetic of Elliptic Curves", Springer-Verlag Editions, ISBN 0-387-96203-4, 1986.
35. D. G. Mesquita, "Architectures reconfigurables et cryptographie : une analyse de robustesse et contremesures face aux attaques par canaux caches", These de Doctorat, Universite de MontPellier II, France, 2006.
36. N.Anane,M.Anane,H.Bessalah,M.Issad ;K.Messaoudi ‘Hardwired Algorithm for variable and long precision multiplication on FPGA » the Mediterranean journal of computers and networks, vol.5 no.4.pp.132-137,2009.
37. N. Nedjah, "A review of Modular multiplication and respective hardware implementations", Informatica, Vol 30, pp :111-129, 2006.
38. E.-H.Y. Wajih, M. Mohsen, Z. Medien, B. Belgacem,"Efficient hardware architecture of recursive Karatsuba-Ofman multiplier", nternational Conference on Design and Technology of Integrated Systems in Nanoscale Era, 2008. DTIS 2008.
39. T.J.Shokrollahi, "Efficient Implementation of Elliptic Curve Cryptography on FPGAs", Doctorate thesis, Bohn University,2006.
40. CERTICOM Research, "SEC 2: Recommended Elliptic Curve Domain Parameters", 2000.
41. D.E.KNUTH ; The Art of Computer Programming : seminumerical algorithms volume 2 .Reading. MA: Addison- Wesley second edition, mars 2005
42. J. Guajardo, C. Paar, "Itoh-Tsujii inversion in standard basis and its application in cryptography and codes", 2002.
43. T. Itoh, S. Tsujii, “A fast algorithm for computing multiplicative inverses in GF(2 ) using normal bases,” Inf. Comput., vol. 78, no. 3,pp. 171–177, 1998.

44. G. M. de Dormale, P. Bulens, J-J. Quisquater “An Improved Montgomery Modular Inversion Targeted for Efficient Implementation on FPGA”, IEEE Proceedings of the International Conference on Field-Programmable Technology, pp: 441–444, 2004.
45. F. Crowe, A. Daly, W. Marnane, "Optimized Montgomery domain inversion on FPGA", Proceedings of the European Conference on Circuit Theory and Design, Aug. 28-Sept. 2, IEEE Xplore Press, USA. , pp: I/277-I/280, 2005.
46. S. Okada, N. Torii, K. Itoh, M. Takenaka, “Implementation of elliptic cruve cryptographic coprocessor over $GF(2^m)$ on an FPGA,” in Proc. Workshop on Cryptographic Hardware and Embedded Systems, LNCS, pp. 25–40, 2000.
47. D. Husemöller, "Elliptic Curves", Second edition, ISBN 0-387 95490-2, Springer-Verlag, New York, Inc., 2002.
48. C.J.McIvor, M. McLoone, J.V. McCanny, "Hardware Elliptic Curve Cryptographic Processor Over GF(p)", IEEE. Trans. Circ. Syst., 53: 1946-1957, 2006.
49. A. Daly, W. Marnane, T. Kerins, E. Popovici, “An FPGA implementation of a GF(p) ALU for encryption processors”. Appli. Des. Vol. 28: 253-260, 2006.
50. Ian Blake, Gerald Seroussi, Gadiel Seroussi, and N Smart. Elliptic curves in cryptography, volume 265. Cambridge university press, 1999.
51. R.C.C. Cheung, N.J. Telle, W. Luk, P.Y.K. Cheung, “Customizable elliptic curve cryptosystems”. IEEE. Trans. Very Large Scal. Integrat. Syst., Vol.13, pp: 1048-1059, 2005.
52. K. Sakiyama, L. Batina, B. Preneel, I. Verbauwhede, “High-performance public-key cryptoprocessor for wireless mobile applications”, Mobile Networks Appli., 12: 245-258, 2007.
53. C. Grabbe, M. Bednara, J. Tech, G.J. Von Zur, J. Shokrollahi,, "FPGA designs of parallel high performance $GF(2^{233})$ multipliers: cryptographic applications" Proceedings of the International Symposium on Circuits and Systems, May 25-28, IEEE Xplore Press, USA., pp: 268-271, ISCAS 2003.
54. Y.K. Lee, L. Batina, K. Sakiyama, I. Verbauwhede, "Elliptic curve based security processor for RFID". IEEE. Trans. Comput., 57: 1514-1527, 2008.
55. Neal Koblitz. A course in number theory and cryptography, volume 114. Springer Science & Business Media, 1994.
56. Lawrence C Washington. Elliptic curves : number theory and cryptography. Chapman and Hall/CRC, 2003.
57. D. J. Bernstein and T. Lange. Explicit-formulas database. Website. URL

https: //www.hyperelliptic.org/EFD/.

58. M. Joye. Fast point multiplication on elliptic curves without precomputation. 2nd International Workshop, WAIFI, pages 33–46, 2008. doi : 10.1007/ 978-3-540-69499-1 4.
59. Shokrollahi, J., 2006. Efficient Implementation of Elliptic Curve Cryptography on FPGAs. Doctorate thesis, Bohn University. http://deposit.ddb.de/cgibin/ dokserv?idn=983089183&dok_var=d1&dok_ext=pdf&filename=983089183.pdf
60. U.S. Department of Commerce/National Institute of Standards and Technology. Digital Signature Standard (DSS). FIPS-186-4, 2013. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.
61. SEC2"Standards for Efficient Cryptography Group": Recommended Elliptic curve Domain Parameters. Version 1.0, 2000.
62. Marie-Angela Cornelie « Implantations et protections de mécanismes cryptographiques logiciels et Matériels ». Autre [cs.OH]. Université Grenoble Alpes, 2016. https://tel.archives-ouvertes.fr/tel-01377372v2.
63. Men12] Alfred J Menezes. Elliptic curve public key cryptosystems, volume 234. Springer Science & Business Media, 2012.
64. Younsung Choi "Cryptanalysis on Privacy-aware Two-factor Authentication Protocol for Wireless Sensor Networks "TELKOMNIKA" Vol. 8, No. 1, February 2018, pp. 605-610.
65. Jean-Pierre Flori, Jérôme Plût, Jean-René Reinhard, Martin Ekerå « Diversité et transparence : choix des courbes elliptiques » NSSI/SDE/ST/LCR, 2015. https://perso.telecom-paristech.fr/flori/gtbac/slides/150528floriplut.pdf.
66. IEEE 1363-2000 "IEEE Standard Specifications for Public-Key Cryptography" Sponsor Microprocessor and Microcomputer Standards Committee of the IEEE Computer Society Approved 30 January 2000 IEEE-SA Standards Board.
67. ECC Brainpool. ECC Brainpool Standard Curves and Curve Generation, 2005. http://www.ecc-brainpool.org/download/ Domain-parameters.pdf.
68. Rogel L.Quilala, Ariel M. Sison, Ruji P. Medina "Modified SHA-1 Algorithm" TELKOMNIKA (Indonesian Journal of Electrical Engineering and Computer Science) Vol. 11, No., pp. 1027-1034, 2018.
69. ANSI X9.62. Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA). American National Standards Institute, X9-Financial Services 2005.

70. R. P. Gallant, R. J. Lambert, and S. A. Vanstone.” Faster point multiplication on elliptic curves with efficient endomorphisms” In J. Kilian, editor, CRYPTO, volume 2139 of LNCS, pages 190-200. Springer, 2001.
71. W. Killmann and W. Schindler. A proposal for : Functionality classes for random number generators. Online. Available at : https ://www.bsi.bund.de, 2011.
72. Andrey Sidorenko and Berry Schoenmakers. Concrete security of the blum-blum-shub pseudorandom generator. In NigelP. Smart, editor, Cryptography and Coding, volume 3796 of Lecture Notes in Computer Science, pages 355–375. Springer Berlin Heidelberg, 2005.
73. Lenore Blum, Manuel Blum, and Mike Shub. A simple unpredictable pseudo-random number generator. SIAM J. Comput., 15(2) :364–383, 1986.
74. John Kelsey, Bruce Schneier, and Niels Ferguson. Yarrow-160 : Notes on the design and analysis of the yarrow cryptographic pseudorandom number generator. In In Sixth Annual Workshop on Selected Areas in Cryptography, pages 13–33. Springer, 1999.
75. Niels Ferguson and Bruce Schneier. Practical Cryptography. John Wiley & Sons, Inc., New York, NY, USA, 1 edition, 2003.
76. Daniel J. Bernstein and Tanja Lange. SafeCurves: choosing safe curves for elliptic-curve cryptography. https://safecurves.cr.yp.to, accessed 1 December 2014.
77. Bruce Schneier. "How to remain secure against the NSA." September 15, 2013. https://www.schneier.com/blog/archives/2013/09/how_to_remain_s.htm.
78. Quan, G., J.P. Davis, S. Devarkal and D.A. Buell, "High-level synthesis for large bit-width multipliers on FPGAs: A case study".
79. Hinkelmann, H., P. Zipf, J. Li, G. Liu and M. Glesner, "On the design of reconfigurable multipliers for integer and Galois field multiplication". Microproc. Microsyst., Vol. 33, pp 2-12. 2008.
80. P. Gallagher. FIPS PUB 186-3 Federal Information Processing Standards Publication Digital Signature Standard (DSS), 2009.
81. T. Wollinger, J. Guajardo, C. Paar, "Security on FPGAs: State of the art implementations and attacks", ACM. Trans. Embedd. Comput. Syst., 3: 534-574, 2004.
82. E.De Mulder, et al. "Electromagnetic Analysis Attack on an FPGA Implementation of an Elliptic Curve Cryptosystem", EUROCON 2005, 1-4244-0049/05, IEEE, 2005.
**83.** Sutikno, S. and A. Surya, "An architecture of F2^2n multiplier for elliptic"**.** Proceedings of the 2000 IEEE International Symposium on Circuits and Systems, 2000.

84. Hinkelmann, H., P. Zipf, J. Li, G. Liu and M. Glesner, "On the design of reconfigurable multipliers for integer and Galois field multiplication". Microproc. Microsyst., Vol. 33, pp 2-12. 2008.
85. Chester Rebeiro, Sujoy Sinha Roy, D. Sankara Reddy, and Debdeep Mukhopadhyay "RevisIting the ItohTsujiii Inversion Algorithm for FPGA platforms", IEEE, transactions on very large scale integration (vlsi) systems, 2010.
86. http://wwwhomes.unibielefeld.de/achim/addition_chain.html.
87. M. Benaissa, W.M. Lim, "Design of flexible GF ($2^m$) elliptic curve cryptography processors", IEEE. Trans. Very Large Scale Integrat. Syst., Vol. 14, pp: 659-662. , 2006.
88. Ashraf B. El-sisi, Sameh M. Shohdy and Nabil Ismail, "Reconfigurable Implementation of Karatsuba Multiplier for Galois Field in Elliptic Curves", Novel Algorithms and Techniques in Telecommunications and Networking, , 87-92, DOI: 10.1007/978-90-481- 3662-9_14, 2010.

**APPENDICE A**

LISTE DES SYMBOLES ET ABREVIATIONS

K : Un corps fini.
Fp : Corps fini d'ordre p
Z/NZ : Pour N ∈ Z, anneau quotient des résidus modulo N.
E/K : Une courbe elliptique E définie sur un corps fini K
ΔE/K : Discriminant d'une courbe elliptique E définie sur un corps fini K.
jE/K : j-invariant d'une courbe elliptique E définie sur un corps fini K
ℤ : L'ensemble des entiers relatifs
ℝ : L'ensemble des réels
ℚ : L'ensemble des rationnels
C : L'ensemble des nombres complexes
ℕ : L'ensemble des entiers naturels
E/K : Une courbe elliptique E définie sur un corps fini K
ℤ : L'ensemble des entiers relatifs
ℝ : L'ensemble des réels
ℚ : L'ensemble des rationnels
ℕ : L'ensemble des entiers naturels
C : L'ensemble des nombres complexes
E(Fp) : Courbe elliptique E définie sur Fp
#E(Fp) Nombres de points de la courbe E(Fp)
AES : Standard avance de chiffrement
ANSI : Standard Américain de normes et information
CCA : chosen-cipher textattack
COA : Attaques à l'aide de textes chiffrés seulement
CPA : Attaques à base de textes clairs choisis
DES : Data Encryption Standard
DL : Logarithme discret
DRNG : Générateurs de *nombres* pseudo-aléatoires
ECDH : Elliptic Curve Diffie-Hellman
ECDLP : Problème du logarithme discret dans les courbes elliptiques
ECC : Cryptographie a base de courbes elliptiques
ECDSA : Algorithme de signature numérique à base courbe elliptique
FIPS PUB : Procédure standard publique de l'information fédérale
FPGA : Circuit à logique programmable
IDEA : International Data Encryption Algorithm
IPSEC : Internet Protocol Security
KPA : Attaques à base de textes clairs-connus
LSB : Least signifiant bit
MAC : code d'authentification des messages
MITM : Meet-In-The-Middle
MLB : Most signifiant bit
NAF : forme non adjacent
NIST : Institut national des standards et technologies
NSA : Agence de sécurité nationale
PLD : Problème du logarithme discret
PLDCE : Problème du logarithme discret sur les courbes elliptiques
*PRNG : génération de nombre pseudo-aléatoires*

| | |
|---|---|
| RSA : | Rivest shamir adleman |
| SEAL: | Software Encryption *Algorithm* |
| SEC: | Standards for Efficient Cryptography |
| SECG: | Standards for Efficient Cryptography Group |
| SHA : | Secure Hash Algorithm |
| SSL: | Secure sockets layer |
| TLS: | Transport Layer Security |
| TRNG : | True Random Number Generator |
| WTLS: | *Wireless Transport Layer Security* |
| VHDL : | very high-level design language |